MSc Final Project Declaration

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in 7COM1075-0509-2020 – Data Science and Analytics Masters Project at the University of Hertfordshire (UH).

It is my own work except where indicated in the report. I did not use human participants in my MSc Project. I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

Signed: Harsha Vardhan Bashetty
Date: 27/08/2021

Abstract

The Encoder-Decoder model is used to perform neural machine translation. Instead of a simple recurrent neural network, Long Short-term Memory (LSTM) and Gated Recurrent Unit (GRU) are used to build four different models. The performance of the four models on three sizes of the dataset is evaluated using BLEU (BiLingual Evaluation Understudy) score. The model with GRU as both encoder and decoder is found to be more appropriate for smaller dataset sizes with fewer translation pairs. The encoder-decoder model with LSTM as an encoder and GRU as a decoder had performed consistent best for larger dataset sizes with more translation pairs.

Acknowledgement

Firstly, I would like to acknowledge my warmest thanks to my supervisor Dr Thiago Matos Pinto who made this work possible. He has provided constant support, valuable guidance, and insightful feedback at every stage of the project.

Next, I want to thank the module team, especially module leader Dr Helen Xiang for providing all the materials and information in a timely manner.

Finally, I want to thank my parents for being the source of unwavering support my whole life and made me realise the ambition of pursuing masters at the University of Hertfordshire a reality.

UNIVERSITY OF HERTFORDSHIRE
School of Computer Science

MSc Data Science and Analytics with Placement

7COM1075: Data Science and Analytics Masters Project

Date: 27/08/2021

LSTM vs. GRU for Encoder-Decoder Neural Machine Translation

Name: Harsha Vardhan Bashetty
Student ID: 18055897
Supervised by: Thiago Matos Pinto

TABLE OF CONTENTS

1. Introduction

1.1. Motivation

There are around 7000 unique languages that exist in the world. It is crucial to have a good translation system to be able to communicate one's expression in the form of text or speech from one language to another in the best way possible. The aim of any system that generates translations or a human translator, is to restore the full meaning of the text or voice from the original language to the target language. It has to be quick and inexpensive so many people and businesses could benefit from such a system. The task of translation is not that straight forward, replacing the text in the original or source language with the target or desired language. This approach of word-to-word substitution will be ineffective given the complex nature of grammar in most languages. The translation system must be able to analyse all the elements of the text and how each word is connected. The knowledge of underlying grammar, syntax, semantics and most importantly context has to be understood to be able to produce impressive translations.

The human translators traditionally used are replaced by machine translation systems. The field of machine translation is an active area of research over the years. Machine translation investigates the use of computer software to translate text or speech from one language to another. It involves analysing the structure of each phrase or term present in text or speech to be translated. This structure is used to break the elements that can be used to translate input language and recomposes the terms of the same structure in the output/target language. Machine translation systems can generate translations quickly even for tremendous amounts of data. Human translators tend to take more amount of time for translating a lengthy article. This is because humans tend to forget the figures of speech used for describing certain words. They would require to constantly look up related references. While machine translation can automatically translate and can memorize key terms and phrases specific to any industry. This characteristic of machine translation leads to the very consistent translations generated across the complete document, which is highly arduous for multiple machine translators. Machine translation is inexpensive and can be used in a wide range of applications like websites, portals, emails, customer reviews, medical claims, research instructions, online comments and social media posts.

1.2. Aim of the project

Comparing the four possible encoder-decoder models for neural machine translation built using two versions of RNN architectures LSTM and GRU with varying sizes of the dataset.

1.3. Objectives of the project

- Comparing the performance of encoder-decoder model with GRU cell as both encoder and decoder for three different dataset sizes
- Comparing the performance of encoder-decoder model with LSTM cell as both encoder and decoder for three different dataset sizes
- Comparing the performance of encoder-decoder model with GRU cell as encoder and LSTM cell decoder for three different dataset sizes
- Comparing the performance of encoder-decoder model LSTM cell as encoder and GRU cell as decoder for three different dataset sizes

1.4. Report Organisation

2. Literature Review:

2.1.Background

2.1.1. Rule-based Machine Translation

Research in the field of machine translation had evolved over the years from traditional rules-based methods to statistical methods to recent encoder-decoder models, improving the quality and speed of the translations between the languages.

In the rule-based machine translation, for every language pair, the emphasis is on a large number of built-in linguistic pairs and thousands of bilingual dictionaries. The software creates an internal representation for each input text by parsing them, using which the text in the target language will be derived. The rule-based machine translation process requires large sets of rules and extensive lexicons with morphological, syntactic, and semantic information. Using the complex set of rules, the software transfers the grammatical structure of the source language into the target language.

2.1.2. Statistical Machine Translation

Statistical machine translation is based on the translation models which analyse the monolingual and bilingual corpora to stem parameters to be used in the model. This performance of the model is heavily dependent on the large availability of multilingual corpora with a minimum of few million domain-specific words and even more amount of data is required for a general language. This poses a huge challenge even when building the statistical translation model is relatively quick. The lack of large amounts of data with the organizations is a bottleneck for reaching optimal threshold levels of performance. The statistical machine translation is a resource-intensive task, requiring higher hardware configurations of CPUs and GPUs. Even availability of high computational resources, there is a thin chance of getting higher performance levels.

2.1.3. Neural Machine Translation

The neural machine translation approach is purely based on neural networks. This alternative to statistical machine translation models was proposed by (Kalchbrenner and Blunsom, 2013) and (Sutskever, Vinyals and Le, 2014).

This approach has proven to be having a high significance not only theoretically but practically as well. In contrast to statistical machine translation models requiring a large amount of memory, the Neural machine translation model just requires a fraction of that. The neural machine translation models trained needed only 500MB of memory while the statistical machine translation models expecting tens of gigabytes of memory. In contrast to the rules-based and phrase-based statistical translations, the neural networks based approach the model is trained jointly improving the performance of translation significantly (Cho, Merrienboer, Bahdanau and Bengio, 2014).

2.1.4. Encoder-Decoder model

The neural machine translation models often consist of an encoder and a decoder. The encoder-decoder model is initially developed for machine translation problems. It uses neural networks for sequence-to-sequence predictions. There are two main components in this architecture: encoder and decoder. The encoder converts the sentence from a given source language to an internal representation of a fixed-size vector called an encoder vector or context vector. The decoder takes the context vector as input and predicts output in the target language. This approach can solve the problems that are not possible with recurrent neural networks like input and output sequences with different lengths and the text having complicated non-monotonic relationships.

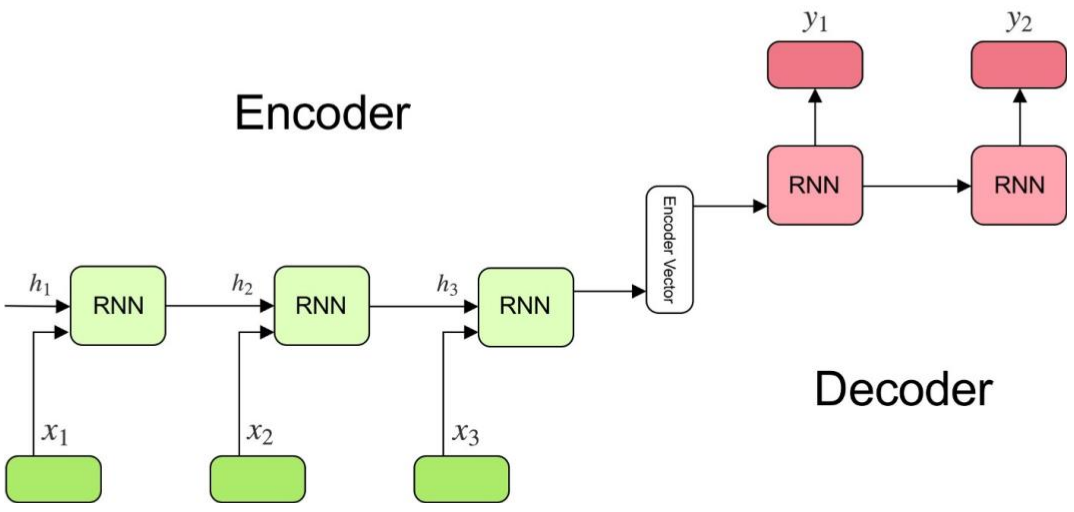


Figure (1). Encoder-Decoder model (Kostadinov, 2019)

The aim of the encoder-decoder architecture is to estimate the conditional probability of the output sequence given an input sequence. These input and output sequences may vary in their length. The recurrent neural network cell in the decoder computes conditional probability from the encoder vector given by the last recurrent neural network cell for a given input sequence.

Generally in this model, the encoder will map sequences of the input vector, x = (x1, x2, ., xn) to the hidden vector sequence h = (h1, h2, .., hn). The generated vector is crucial in predicting the target output sequence. Specifically, each time the model generates a word of the output sentence, the model focuses on the words in the source sentence where the most relevant information is concentrated. Then, the model predicts the next word of the output sentence based on the context as well as the previously hidden vector generated respectively by the decoder. Finally, this process is repeated until the end of the source sentence (Bensalah, Habib and Abdellah, 2021).

2.1.5. Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network which uses temporal or sequential data. This popular neural network was introduced by Elman (Elman, 1990). They are similar to convolutional neural networks, in the sense that both make use of the training data for learning. They are different from CNN's having a memory unit. At each time step the RNN's take information from the inputs from previous timestamps and also the input of the current timestamp. In the traditional CNN's the previously trained data does not play any role in predicting the output but in RNN's the inputs and outputs are not independent of each other.

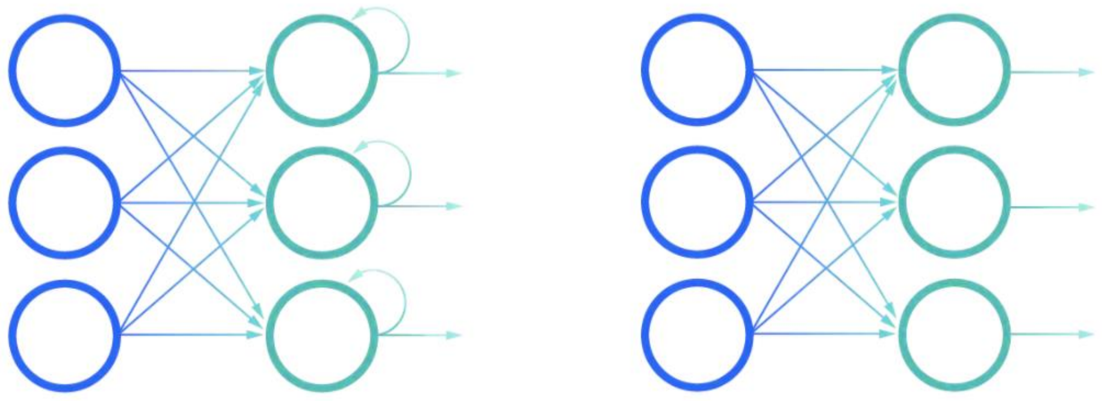


Figure (2): Recurrent Neural network vs Feedforward neural network (IBM Education, 2019)

The same weight parameter is shared within each layer of the RNN's, unlike the feedforward neural networks that have different weight parameters across each of the nodes. But, the feedforward networks and RNN's have their weights updated using the same backpropagation and gradient descent algorithms.

The recurrent neural network updates its internal state ht at each time step, based on a new input xt and the previous state ht−1 and output yt is generated. Usually, they are computed at each time step, recursively by applying the following two operations:

$$ht = f(Wixt + Whht-1 + bh)$$

$$yt = f(Woht + bo)$$

Where f is an element-wise non-linearity activation function like sigmoid, tanh or rectified linear unit (ReLU) and bh is the bias term.

But backpropagation used during training to update the weights of the recurrent network is slightly different because of the sequential nature of the data, it is called backpropagation through time (BPTT). Here the gradient-based technique unfolds the network through time so as to compute the actual gradients of parameters in each time step. In the RNN model errors are calculated during training from output to input similar to traditional backpropagation but the biggest distinguisher is the BPTT sums error at every time step, but the feedforward networks do not share weight parameters and the errors are not summed.

The backpropagation through time will result in two potential problems for RNNs. They are vanishing and exploding gradients problems. The vanishing gradient problem occurs when the gradient is too small and in the process of BPTT, they become even smaller making the updating of weights insignificant freezing the algorithm learning process. The exploding gradient problem is the opposite of the vanishing problem. In this case, the resulting larger gradients are used in backpropagation will make the model unstable making the weights grow too quick eventually leading to the halting of the program.

Two popular alternatives to the RNN are used to fix the problem of vanishing and exploding gradients. They are LSTM and GRU architectures that help to fix these issues and perform well in capturing long-term dependencies.

2.1.6. Long Short-term Memory

Long short-term memory units are capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber (1997). LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn.

LSTM incorporate some gates to control the information flow into and out of the cell. In addition to the hidden units in RNN, memory cells are used to store long-term information, which is updated linearly. Empirically, LSTM can preserve information for arbitrarily long periods of time. Figure 1 shows LSTM's overall architecture.

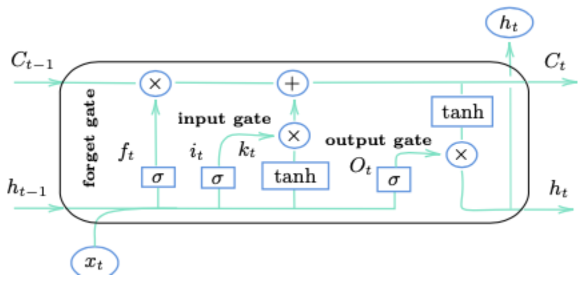


Figure (3). LSTM architecture (Bensalah, Habib and Abdellah, 2021)

The forget gate is used to remember the significant information from the past steps. Then, the input gate decodes the data to be added from the present step. Finally, the output gate determines the next hidden state. To compute the states of the LSTM, we act as the following: First, we calculate at the forget gate and then evaluate the input gate:

$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$

$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$

$k_t = \tanh(W_k h_{t-1} + U_k x_t + b_k)$

We calculate C_t using the above three vectors, which represents the LSTM's memory state and at the output gate O_t is computed:

$C_t = C_{t-1} (*) f_t + i_t (*) k_t$

$O_t = \sigma(W_O h_{t-1} + U_O x_t + b_O)$

Where (*) is the element-wise Hadamard product, x_t is the word representation at time t, W_O, U_O, W_k, U_k, W_i, U_i, W_f , U_f are weight matrices, b_O, b_k, b_i, b_f are the bias, σ and tanh represent respectively the element-wise sigmoid activation and the tangent hyperbolic functions (Bensalah, Habib and Abdellah, 2021).

2.1.7. Gated Recurrent Unit

Gated recurrent unit (GRU) is originally proposed in (Cho, Merrienboer, et al., 2014). Similarly, to LSTM unit, GRU also has gating units to control the information flow. While LSTM unit has a separate memory cell, GRU unit only maintains one kind of internal states, thus reduces computational complexity. It utilizes an update and a reset gates to avoid the issue of the vanishing gradient that penalizes the standard recurrent neural network. These are basically two vectors that agree which data will be transferred to the output.

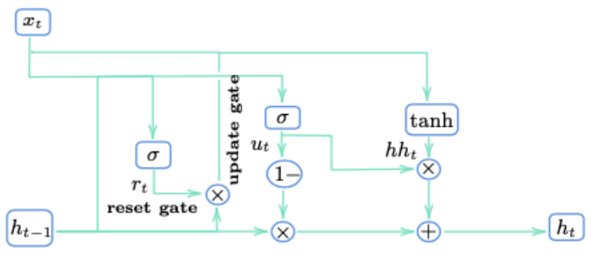


Figure (4): GRU architecture (Bensalah, Habib and Abdellah, 2021)

As shown in Figure 2, the update gate enables the system to determine the amount of the past data that needs to be passed, and which is computed as follows:

$u_t = \sigma(U_u h_{t-1} + W_u x_t)$

Essentially, the reset gate is used to determine the amount of data that needs to be forgotten, and it is evaluated hence:

Finally, the hidden state h_t at step t is computed using hh_t:

$hh_t = \tanh(U_{hh}(r_t (*) h_{t-1}) + W_{hh} x_t)$

$h_t = (1-u_t) (*) \tanh(h_{t-1}) + u_t (*) hh_t$ if $t > 0$ and $h_t = 0$ if $t = 0$

where: W_r, U_r, W_u, U_u, W_{hh}, U_{hh} are weight matrices.

2.1.8. BLEU score

BLEU (BiLingual Evaluation Understudy) is an evaluation metric for automatic machine-translated text. It has values between 0 and 1. The value of 0 means the machine-translated text is of low quality having no overlap with the reference or source translations. The value of 1 means the machine-translated text is of high quality with highly overlapping with the source or reference translations. There are certain situations where the BLEU metric shows bad performance. For an instance, the two sentences can possibly have had a low BLEU score even be able to capture whole or most of the meaning. This phenomenon happens because this BLEU score metric is based on corpus and for the individual sentences the metric cannot be factorized. But its statistics are based on the entire corpus for generating a score. (Google Cloud, n.d.)

$$\text{BLEU} = \underbrace{\min\Big(1, \exp\big(1 - \frac{\text{reference-length}}{\text{output-length}}\big)\Big)}_{\text{brevity penalty}} \underbrace{\Big(\prod_{i=1}^{4} precision_i\Big)^{1/4}}_{\text{n-gram overlap}}$$

$$precision_i = \frac{\sum_{\text{snt}\in\text{Cand-Corpus}} \sum_{i\in\text{snt}} \min(m^i_{cand}, m^i_{ref})}{w^i_t = \sum_{\text{snt'}\in\text{Cand-Corpus}} \sum_{i'\in\text{snt'}} m^{i'}_{cand}}$$

where

- m^i_{cand} is the count of i-gram in candidate matching the reference translation
- m^i_{ref} is the count of i-gram in the reference translation
- w^i_t is the total number of i-grams in candidate translation

The mathematical definition of BLEU score (Google Cloud, n.d.)

The formula has two parts:

1. **Brevity Penalty:** This term will penalise the translations that generated of short length when compared with the closest reference term length having exponential decay. This term can be understood as a compensation parameter for the metric having no recall term. So, it makes the evaluation metric more versatile.
2. **N-Gram Overlap:** This term in the formula looks for the generated translation's 1gram, 2gram, 3gram and 4gram matches for the n-grams in the translations used for reference. This term makes the BLEU metric powerful by acting like a precision term. The better the longer n-grams count matching the reference translation will mean

good fluency of the model. And the smaller n-grams like 1gram or uni-gram will be accounting for adequacy. There is a problem with n-grams being over count and resulting poor performance of the evaluation metric, to prevent this situation from happening the count of n-grams is clipped to be equal to maximal n-gram count occurred in the reference text sentence. (Google Cloud, n.d.)

2.2. Related studies

Sutskever, Vinyals and Le, (2014), They have developed an encoder-decoder model with LSTM used as both encoder and decoder. This was the first model that had outperformed the well-established statistical machine translation system on large scale problems. In the encoder-decoder architecture, the idea was to encode a large fixed-length internal vector representation from the input sequence given at one timestep at a time to LSTM in the encoder. Then another LSTM in the decoder is used on the internal vector to extract the output sequence. Their model was applied to the task of English to French translation. In the output sequences, the end-of-sentence token is added at the end signifies the end of the translated sequence allowing the proposed model to have the capability to predict output sentences with varying lengths. The model was trained on a subset of 12 million sentences in the dataset for the WMT 2014 translation task. This subset was already tokenized and contain 348 million and 304 million French and English words respectively. In the pre-processing step, they had reduced the source and target vocabularies to 160000 most frequent English words and most frequent French words respectively. The UNK token was used to replace all the out-of-vocabulary words. In the model configuration, the input sequences are reversed with input words represented by a 1000-dimensional word embedding layer. In the output layer SoftMax activation function was used. There were 4 layers with 1000 units per layer in the input and output models. The step of learning rate decay is performed and to mitigate the chances of gradient explosion, gradient clipping is used. During training batch sizes of 128 sequences was used. To increase the computation speed, roughly the same length of sentences was used in the batches. The model outperformed the statistical machine system with a BLEU score of 34.81, compared to the statistical systems baseline score of 33.30.

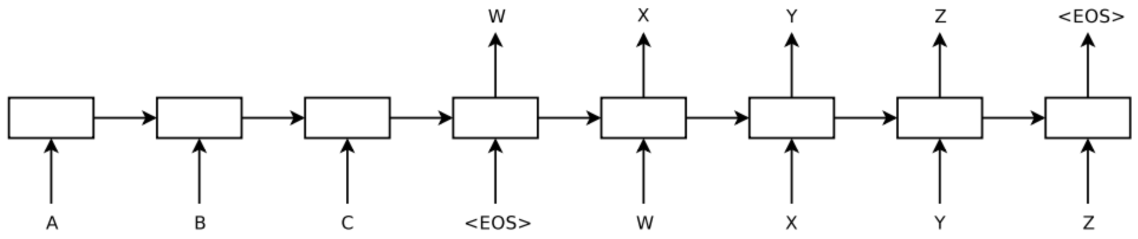


Figure (5): Encoder-Decoder model propose by (Sutskever, Vinyals and Le, 2014)

Cho et al., (2014) in their paper presented an implementation based on a simpler recurrent neural network, gated recurrent unit (GRU) instead of the LSTM as the previously mentioned study. They have used a batch size of 64 sentences during training. Their model has input words represented by 100-dimensional word embeddings with 1-layer of 1000 GRUs used to configure both encoder and decoder. After the decoder, 500 Maxout units pooling 2 inputs were used.

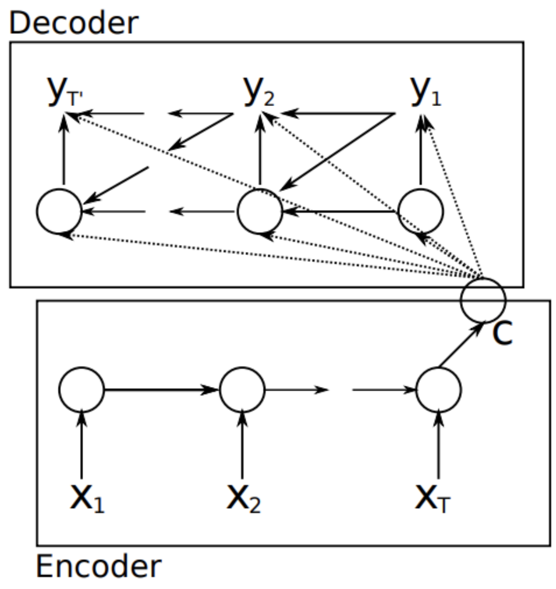


Figure (6): Encoder-Decoder model proposed by (Cho et al., 2014)

Cho, Merrienboer, Bahdanau and Bengio (2014), had discovered two contributing factors for degrading performance for their previously proposed model. They are the length of input sentences and the presence of the number of words outside the corpus vocabulary. An increase in both factors had resulted in a quick decrease in the model performance.

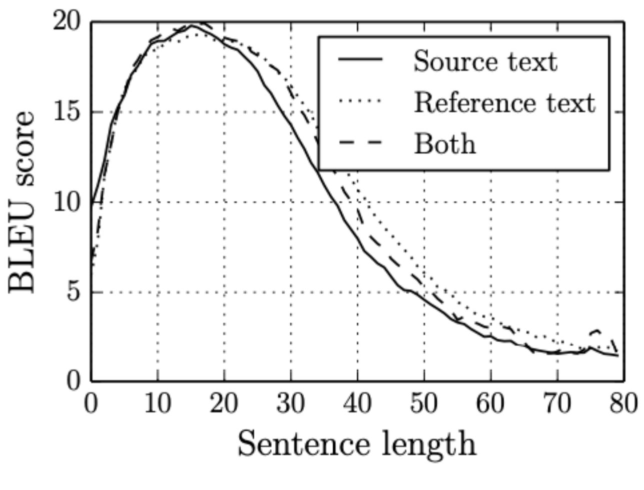


Figure (7): Impact of sentence length on the performance of encoder-decoder model (Cho, Merrienboer, Bahdanau and Bengio, 2014)

In the next paper, Bahdanau, Cho and Bengio (2015), have proposed the attention mechanism to solve both the above-mentioned issues. In this newly proposed method, they have used a fuller representation of encoded input rather than the previously used fixed vector representation. The idea is to allow the model to pay attention to the different parts of input for each output to be generated by the decoder.

Bidirectional layers are used instead in the previously proposed model and the vocabulary is generated on the text corpus to contain 30000 most common words. They have trained the model with the sentences up to maximum lengths of 20 and the extend it to 50. The model is fit for only 4 to 6 epochs. Beam search is used along with a batch of 80 sentences for the model.

By using the attention mechanism, the new proposed architecture was able to derive a translation quality similar to the phrase-based statistical machine translation models at that time.

2.3.Summary of literature review

This series of papers had a great impact on the field of machine translation paving the path for a better understanding of natural languages. The use of LSTM cell for both the encoder and decoder of the encoder-decoder model had resulted in a better BLEU score than the statistical machine translation models. The newly proposed simple RNN cell, gated recursive unit (GRU) is computationally less expensive in comparison to the LSTM. The use of attention mechanism had resulted solved the problem of degrading performance with an increase of the length of input sentences and the presence of larger proportions of unknown words in the vocabulary.

3. Methodology

In this project, the French sentences of varying lengths are translated to English using encode-decoder model. The individual translation pairs may of different length in the source and target languages. The raw data is cleaned first, then split into training, validation, and test datasets. Four encoder-decoder models are built. These models are trained on training data, optimized using validation data and the derived model's performance is evaluated on the testing data. This method is carried out for three datasets of varying sizes generated randomly on the original raw text data. The evaluation of these models is carried out using BLEU score.

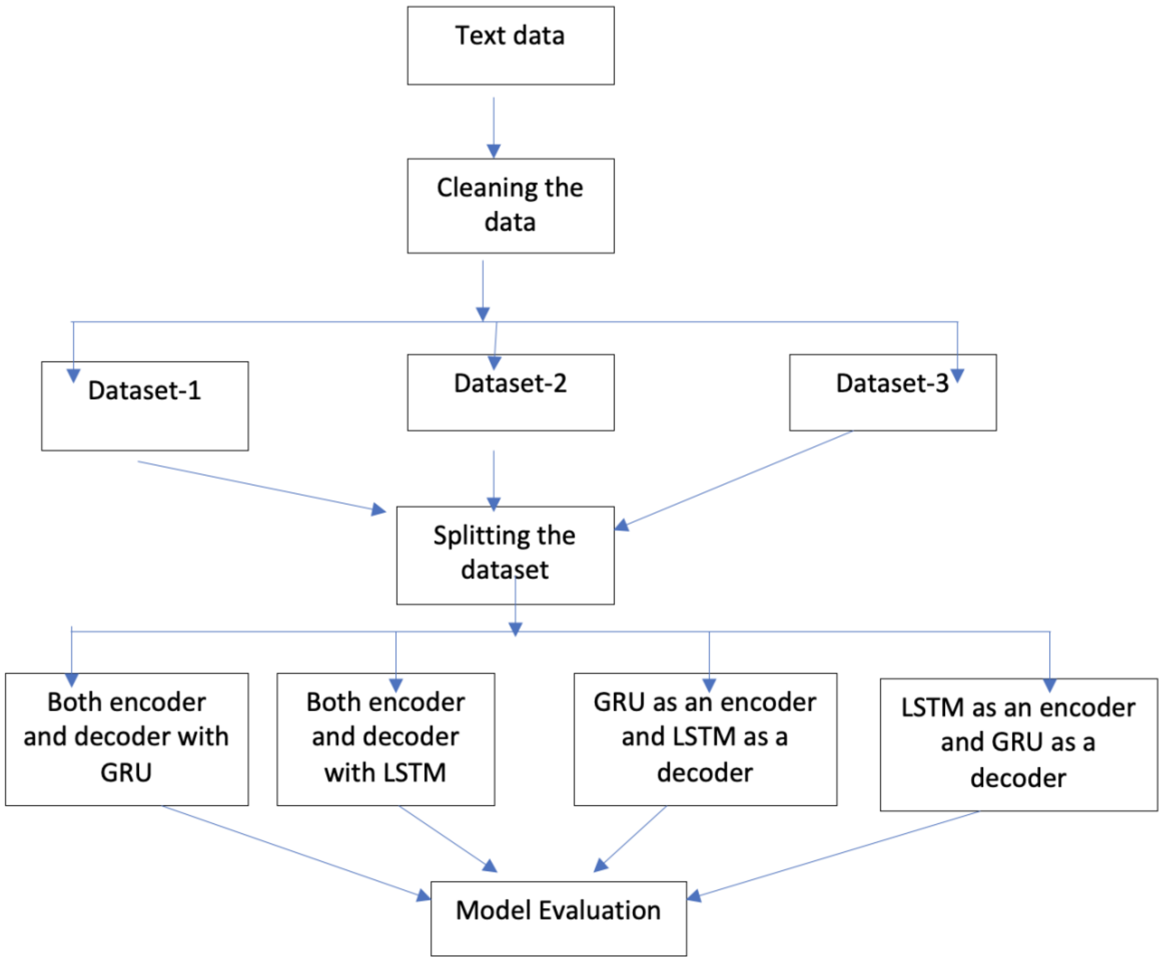
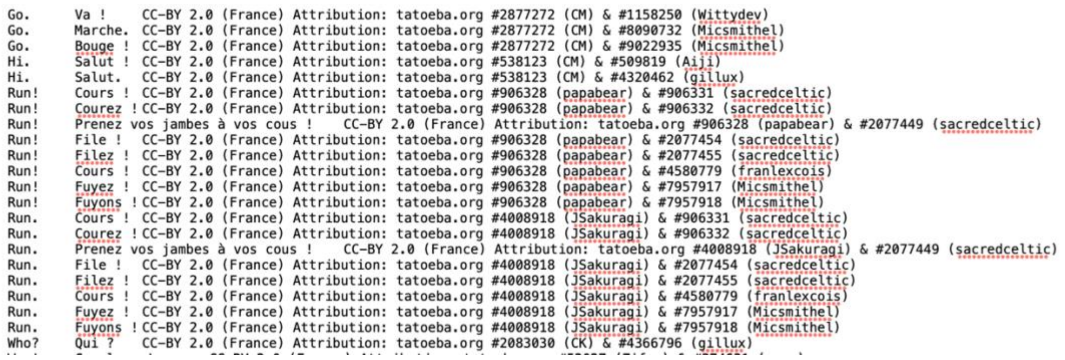

Text data
Cleaning the data
Dataset-1
Dataset-2
Dataset-3
Splitting the dataset
Both encoder and decoder with GRU
Both encoder and decoder with LSTM
GRU as an encoder and LSTM as a decoder
LSTM as an encoder and GRU as a decoder
Model Evaluation

Figure (8): Workflow of the project methodology

3.1.Building neural translation model

3.1.1. Dataset

The dataset for this project derived from ManyThings.org website (English-French Sentences from the Tatoeba Project, ud). It has examples taken from Tatoeba project. As seen in Figure (9), this dataset contains English phrases and their appropriate French translations. This is intended to be used with Anki flashcard software. The data is requested using python's curl tool by providing the dataset URL and the obtained zip file is decompressed. The text file contains 190206 pairs of English to French phrases.



```
Go.     Va !    CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #1158250 (Wittydev)
Go.     Marche. CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #8090732 (Micsmithel)
Go.     Bouge ! CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #9022935 (Micsmithel)
Hi.     Salut ! CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #509819 (Aiji)
Hi.     Salut.  CC-BY 2.0 (France) Attribution: tatoeba.org #538123 (CM) & #4320462 (gillux)
Run!    Cours !  CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #906331 (sacredceltic)
Run!    Courez ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #906332 (sacredceltic)
Run!    Prenez vos jambes à vos cous !    CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #2077449 (sacredceltic)
Run!    File !  CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #2077454 (sacredceltic)
Run!    Filez ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #2077455 (sacredceltic)
Run!    Cours ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #4580779 (franlexcois)
Run!    Fuyez ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #7957917 (Micsmithel)
Run!    Fuyons ! CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #7957918 (Micsmithel)
Run.    Cours ! CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #906331 (sacredceltic)
Run.    Courez ! CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #906332 (sacredceltic)
Run.    Prenez vos jambes à vos cous !    CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #2077449 (sacredceltic)
Run.    File !  CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #2077454 (sacredceltic)
Run.    Filez ! CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #2077455 (sacredceltic)
Run.    Cours ! CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #4580779 (franlexcois)
Run.    Fuyez ! CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #7957917 (Micsmithel)
Run.    Fuyons ! CC-BY 2.0 (France) Attribution: tatoeba.org #4008918 (JSakuragi) & #7957918 (Micsmithel)
Who?    Qui ?   CC-BY 2.0 (France) Attribution: tatoeba.org #2083030 (CK) & #4366796 (gillux)
```

Figure (9): Sample English to French translation pairs

3.1.2. Data cleaning

Firstly, data must be loaded without changing or tampering the Unicode French characters. The loaded text file is split by each line and then by phrase as each line of the text file contain English phrase and its French translation.

Next, non-printable characters like NULL, BACKSPACE, HORIZONTAL TAB and VERTICAL TAB are removed. Punctuation characters are removed as they are not important for the task. Using Unicode normalisation, the sequence of Unicode characters is converted to their equivalent ASCII format. To remove any situations of case sensitivity, all the characters are converted to lowercase. The non-alphabetic tokens which have less importance for translation task are removed. The code used for cleaning is present in Appendix 10.1.

3.1.3. Data splitting

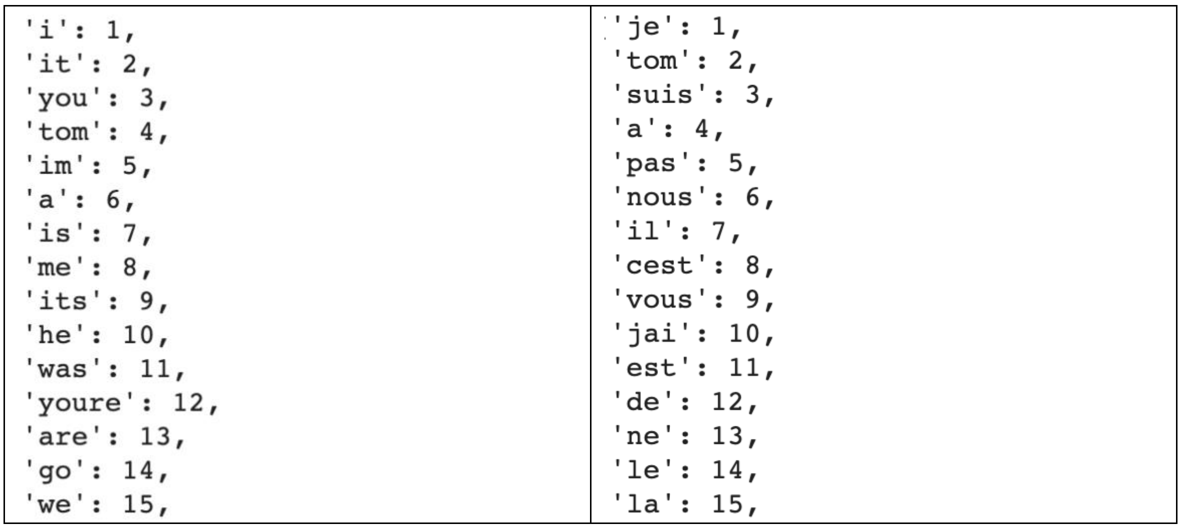
There are a total of 190206 phrase pairs. To study impact of the size of data, three separate datasets are randomly sampled from complete data. The sizes of three datasets are 15000, 25000 and 50000. These three datasets are split in to training, validation and test datasets in 70%, 10% and 20% proportions respectively. The table-1 shows the number of phrase pairs in each dataset available for training, validation and testing.

	Total number of pairs	Number of pairs used for training	Number of pairs used for validation	Number of pairs used for testing
Dataset-1	15000	10500	1500	3000
Dataset-2	25000	17500	2500	5000
Dataset-3	50000	35000	5000	10000

Table-1: Total number of pairs available for training, validation and testing for each dataset

3.1.4. Developing translation model

Firstly, tokenization is carried out to convert the text in both English and French phrases into integers used for modelling. Based on the given list of texts, a word to index dictionary is generated mapping each word with unique integer value. Integer 0 is a reserved index and won’t be assigned to any word. The words associated with lower integer value are more frequent ones, mostly being stop words. Fig-2 shows a sample of words in English and French along with their associated integer from tokenization. Table-2 shows the size of vocabulary, determining the number of unique words in text corpus of each language and the maximum length of sentence in both the input and target languages for each dataset.



```
'i': 1,
'it': 2,
'you': 3,
'tom': 4,
'im': 5,
'a': 6,
'is': 7,
'me': 8,
'its': 9,
'he': 10,
'was': 11,
'youre': 12,
'are': 13,
'go': 14,
'we': 15,
```

```
'je': 1,
'tom': 2,
'suis': 3,
'a': 4,
'pas': 5,
'nous': 6,
'il': 7,
'cest': 8,
'vous': 9,
'jai': 10,
'est': 11,
'de': 12,
'ne': 13,
'le': 14,
'la': 15,
```

Figure (10): English and French words along with their associated integers

	French vocabulary size	French maximum sentence length	English vocabulary size	English maximum sentence length
Dataset-1	5753	10	2776	5
Dataset-2	8021	12	3949	5
Dataset-3	12184	14	6010	7

Table-2: Total vocabulary size and maximum sentence length French and English phrases in each dataset.

Word embedding is a learned representation for text where words having similar meaning are similarly represented in the form of real-valued vectors. It allows dense distributed representation of words than sparsely distributed one hot encoding using significantly large number of dimensions in contrast to much smaller dimensions of word embedding. The dense representation allows better generalization by providing similar representation to the features providing similar clues.

As per mentioned in the previous section, the encoder-decoder model architecture is considered. The input sequence is given to the encoder, and it will encode the given input sentences/ text sequences at the front-end of the sequential model that’s being built. The input from the encoder is converted to a hidden state. The hidden state is only given to the decoder for generating output or predicting the target sequences, only after all the input is finished. The number of time steps the model must wait before moving to decoder is determined by the length of input sentence. It is the French sentence maximum length that’s being used to pad the sentences with shorter number of words. The decoder receives this hidden state and bases on the RNN architecture use, i.e., GRU or LSTM, the output or target sequence is generated. One of the important factors at the decoder is number of words it has to map from the input to the output. As we are aware from the dataset there are varying lengths of sentence for each of the three dataset sizes that are randomly generated from the whole available text dataset.

3.2.Model-1: GRU as both encoder and decoder

In this experiment, the encoder and decoders of the encoder-decoder model are GRU. The model is built with word embedding with arguments, 'input_dim' describing the size of the vocabulary is set to be equal to french vocabulary size. which is 5753, 8021 and 12184 for dataset-1, dataset-2 and dataset-3 respectively.

The argument of the embedding layer, 'output_dim' which describes the dimension of dense embedding is set as 256, 'input_lenght' describing the length of input sequences is set to be equal to the maximum length of French sentences. Which is 10, 12 and 14 for dataset-1, dataset-2 and dataset-3 respectively. The argument 'mask_zero' which specifies whether or not the value 0 in the input is a special value that should be masked out is set as True. Then the GRU is added as an encoder taking input from the embedding layer. The arguments of Keras GRU layer are, 'units' that describes the dimensionality of output space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as False as the output from the encoder is not used by the decoder and all other arguments taken with their default values. Next, the RepeatVector layer of Keras is used to repeat the input a set number of times. It has one argument ‘n’ which specifies the repetition factor is set to the maximum sentence length of English phrases. Which is 5, 5 and 7 for dataset-1, dataset-2 and dataset-3 respectively. The output from the encoder vector from the previous layer is given as input to the decoder GRU layer. The arguments of the GRU decoder, 'units' specifying the dimensionality of input space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as True and all other arguments are taken with their default values.

The Keras TimeDistributed layer, which allows the application of a layer to every temporal slice of input is taken with a fully connected using Dense layer having softmax activation function mapping the prediction to the target English vocabulary.

Sample code used to build this mode is present in Appendix 10.2.

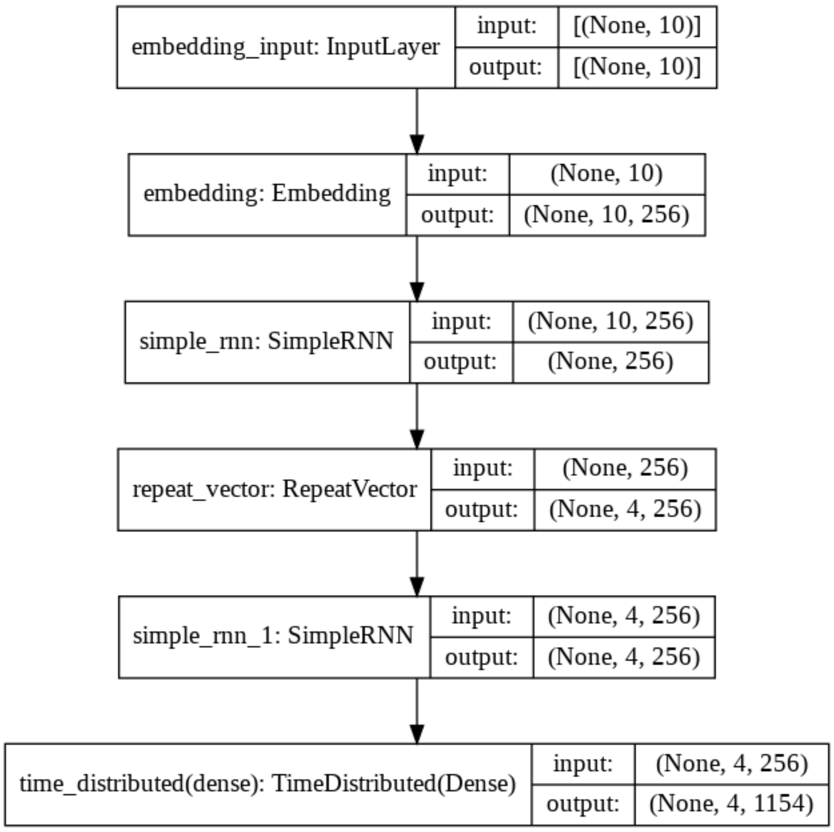


Figure (11): Summary of the model with GRU as both encoder and decoder generated on dataset-1

3.3.Model-2: LSTM as both encoder and decoder

In this experiment, the encoder and decoders of the encoder-decoder model are LSTM. The model is built with word embedding with arguments, 'input_dim' describing the size of the vocabulary is set to be equal to french vocabulary size. which is 5753, 8021 and 12184 for dataset-1, dataset-2 and dataset-3 respectively. The argument of the embedding layer, 'output_dim' which describes the dimension of dense embedding is set as 256, 'input_lenght' describing the length of input sequences is set to be equal to the maximum length of French sentences. Which is 10, 12 and 14 for dataset-1, dataset-2 and dataset-3 respectively. The argument 'mask_zero' which specifies whether or not the value 0 in the input is a special value that should be masked out is set as True.

Then the LSTM is added as an encoder taking input from the embedding layer. The arguments of Keras LSTM layer are, 'units' that describes the dimensionality of output space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as False as the output from the encoder is not used by the decoder and all other arguments taken with their default values. Next, the RepeatVector layer of Keras is used to repeat the input a set number of times. It has one argument ‘n’ which specifies the repetition factor is set to the maximum sentence length of English phrases. Which is 5, 5 and 7 for dataset-1, dataset-2 and dataset-3 respectively. The output from the encoder vector from the previous layer is given as input to the decoder GRU layer.

The arguments of the LSTM decoder, 'units' specifying the dimensionality of input space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as True and all other arguments are taken with their default values. The Keras TimeDistributed layer, which allows the application of a layer to every temporal slice of input is taken with a fully connected using Dense layer having softmax activation mapping the prediction to the target English vocabulary.

Sample code used to build this mode is present in Appendix 10.3.

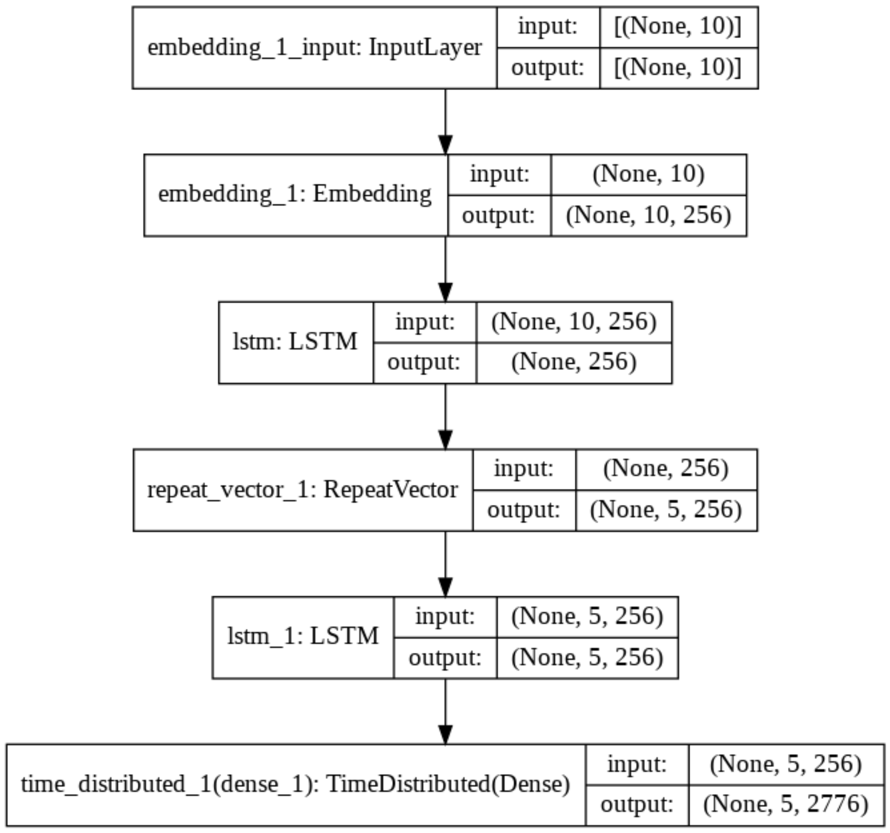


Figure (12): Summary of the model with LSTM as both encoder and decoder generated on dataset-1

3.4. Model-3: GRU as an encoder and LSTM as a decoder

In this experiment, the encoder is GRU and the decoder is LSTM for the encoder-decoder. The model is built with word embedding with arguments, 'input_dim' describing the size of the vocabulary is set to be equal to french vocabulary size. which is 5753, 8021 and 12184 for dataset-1, dataset-2 and dataset-3 respectively. The argument of the embedding layer, 'output_dim' which describes the dimension of dense embedding is set as 256, 'input_lenght' describing the length of input sequences is set to be equal to the maximum length of French sentences. Which is 10, 12 and 14 for dataset-1, dataset-2 and dataset-3 respectively. The argument 'mask_zero' which specifies whether or not the value 0 in the input is a special value that should be masked out is set as True.

Then the GRU is added as an encoder taking input from the embedding layer. The arguments of Keras GRU layer are, 'units' that describes the dimensionality of output space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as False as the output from the encoder is not used by the decoder and all other arguments taken with their default values. Next, the RepeatVector layer of Keras is used to repeat the input a set number of times. It has one argument ‘n’ which specifies the repetition factor is set to the maximum sentence length of English phrases. Which is 5, 5 and 7 for dataset-1, dataset-2 and dataset-3 respectively. The output from the encoder vector from the previous layer is given as input to the decoder LSTM layer.

The arguments of the LSTM decoder, 'units' specifying the dimensionality of input space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as True and all other arguments are taken with their default values. The Keras TimeDistributed layer, which allows the application of a layer to every temporal slice of input is taken with a fully connected using Dense layer having softmax activation mapping the prediction to the target English vocabulary.

Sample code used to build this mode is present in Appendix 10.3.

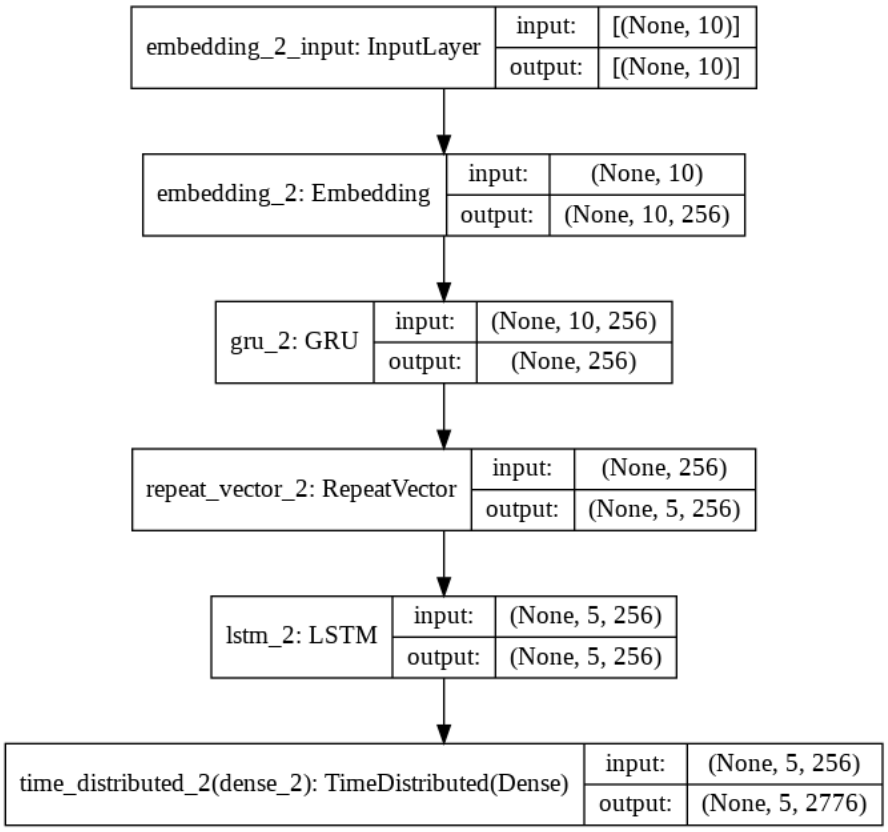


Figure (13): Summary of the model with GRU as encoder and LSTM as decoder generated on dataset-1

3.5. Model-4: LSTM as an encoder and GRU as a docoder

In this experiment, the encoder is LSTM and the decoder is GRU for the encoder-decoder. The model is built with word embedding with arguments, 'input_dim' describing the size of the vocabulary is set to be equal to french vocabulary size. which is 5753, 8021 and 12184 for dataset-1, dataset-2 and dataset-3 respectively. The argument of the embedding layer, 'output_dim' which describes the dimension of dense embedding is set as 256, 'input_lenght' describing the length of input sequences is set to be equal to the maximum length of French sentences. Which is 10, 12 and 14 for dataset-1, dataset-2 and dataset-3 respectively. The argument 'mask_zero' which specifies whether or not the value 0 in the input is a special value that should be masked out is set as True.

Then the LSTM is added as an encoder taking input from the embedding layer. The arguments of Keras LSTM layer are, 'units' that describes the dimensionality of output space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as False as the output from the encoder is not used by the decoder and all other arguments taken with their default values. Next, the RepeatVector layer of Keras is used to repeat the input a set number of times. It has one argument ‘n’ which specifies the repetition factor is set to the maximum sentence length of English phrases. Which is 5, 5 and 7 for dataset-1, dataset-2 and dataset-3 respectively. The output from the encoder vector from the previous layer is given as input to the decoder GRU layer.

The arguments of the GRU decoder, 'units' specifying the dimensionality of input space is set as 256, return_sequences which specify whether to return the last output in the output sequence or full sequence is set as True and all other arguments are taken with their default values. The Keras TimeDistributed layer, which allows the application of a layer to every temporal slice of input is taken with a fully connected using Dense layer having softmax activation mapping the prediction to the target English vocabulary.

Sample code used to build this mode is present in Appendix 10.4.

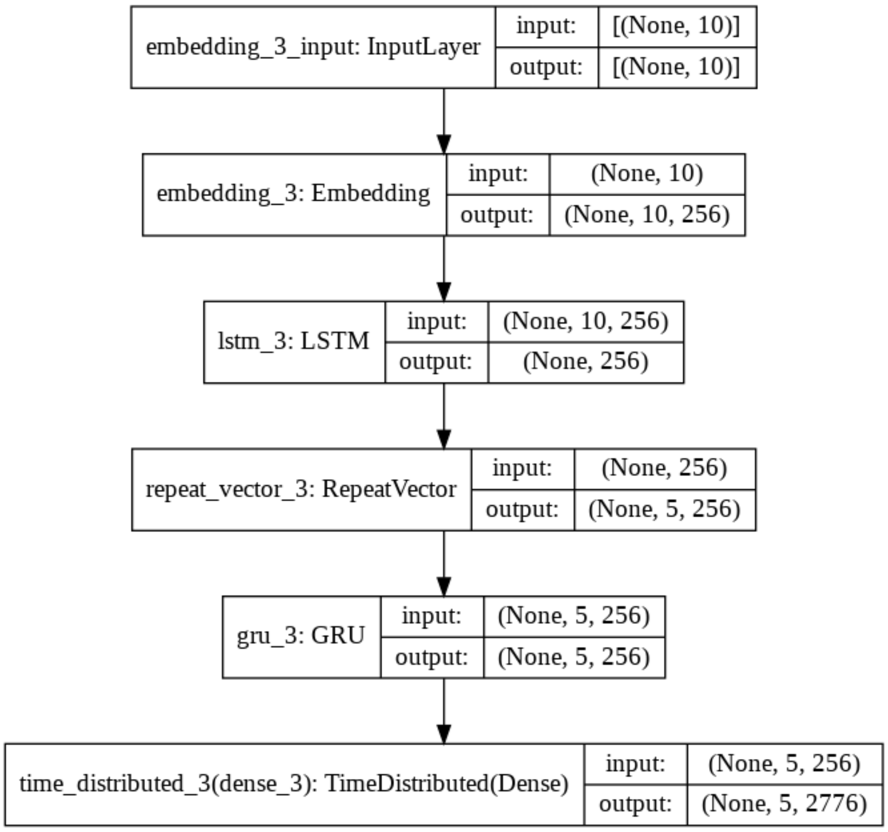


Figure (14): Summary of the model with LSTM as encoder and GRU as decoder generated on dataset-1

4. Results and Discussion

4.1. Model-1: GRU as both encoder and decoder

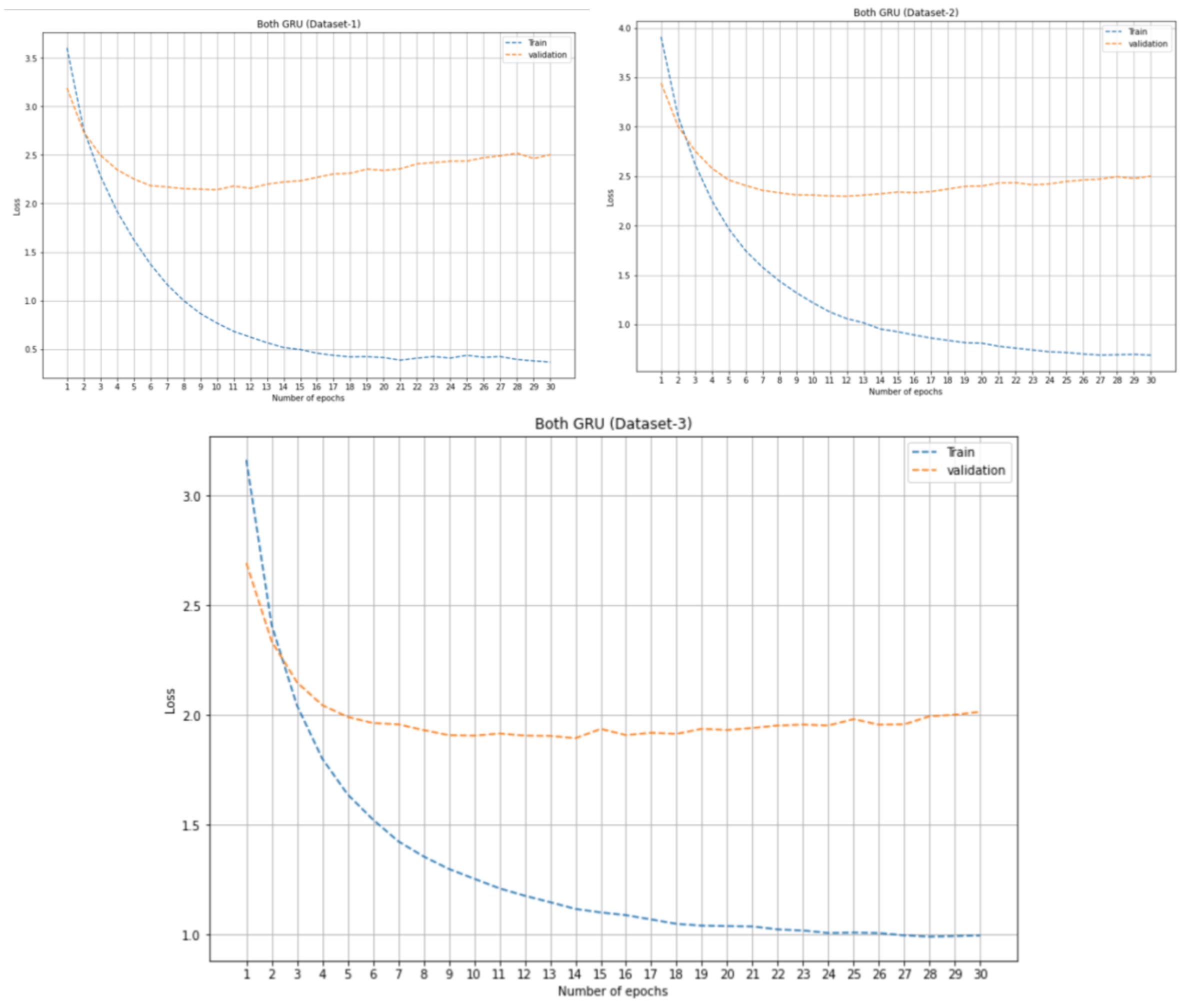


Figure (15): Training and Validation loss values for the model with GRU as both encoder and decoder with 30 epochs. The three subplots are for the three datasets, dataset-1, dataset-2 and dataset-3

The model architecture is discussed in the section 3.2, are compiled with Adam optimizer for learning rate 1e-2. Categorical cross-entropy is considered as a loss function to be optimised with categorical accuracy as the metric. Each model is fitted on the training dataset. The batch size is taken as 64. The maximum number of epochs is kept as 30. The model trained is evaluated on the validation data.

The average time taken by the model per epoch is 12 seconds on dataset-1, which has increased drastically for dataset-3 to 83 seconds. The number of epochs required to reach the optimal state, neither underfitting nor overfitting, is increased from 10 to 14 over the datasets. The value of the BLEU score had improved slightly from 9.9814 to 12.9878 over the three dataset sizes.

	Both GRU		
	Dataset-1	Dataset-2	Dataset-3
BLEU score	9.9814	10.9242	12.9878
Average time take per epoch (s)	12	22	83
Number of epochs	10	12	14

Table-3: The metrics, BLEU score, Average time take per epoch (s) and Number of epochs on the encoder-decoder model with GRU as both encoder and decoder generated for three datasets: Dataset-1, Dataset-2 and Dataset-3

4.2.Model-2: LSTM as both encoder and decoder

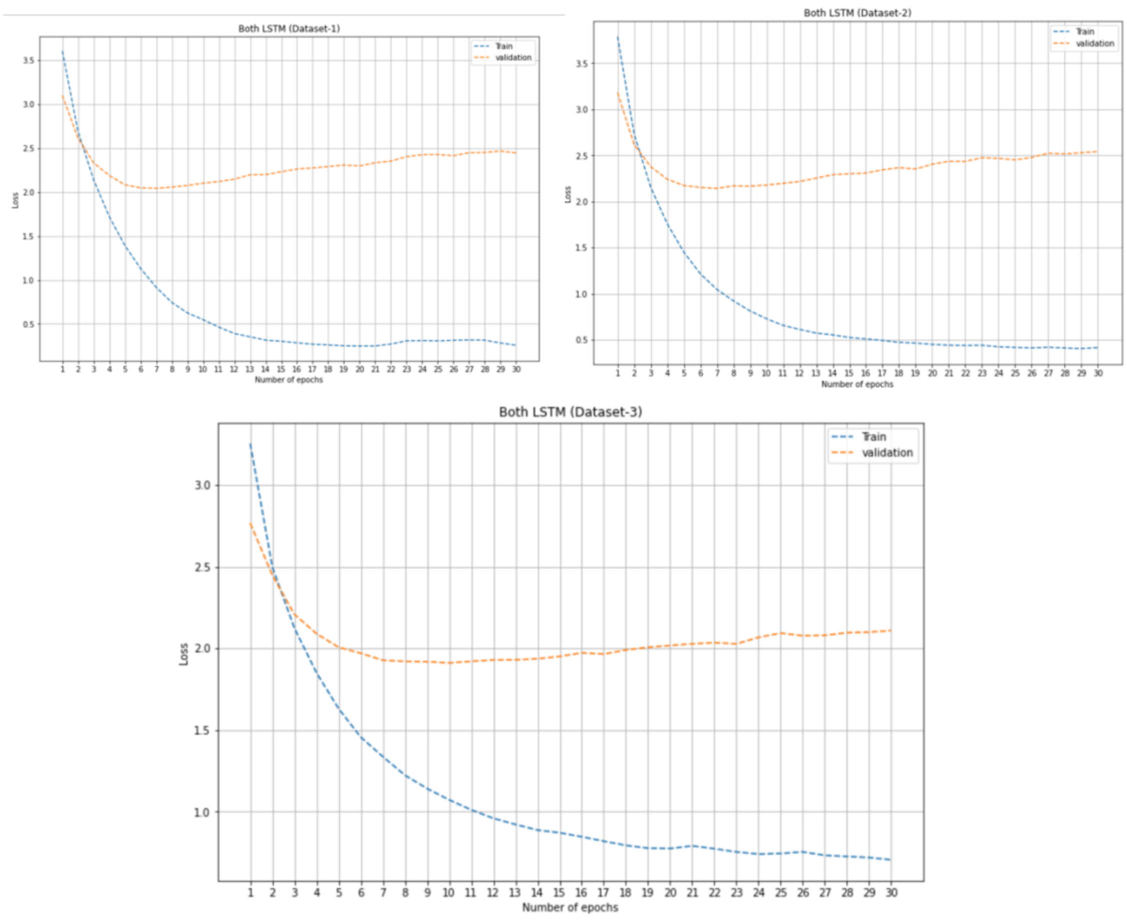


Figure (16): Training and Validation loss values for the model with LSTM as both encoder and decoder with 30 epochs. The three subplots are for the three datasets, dataset-1, dataset-2 and dataset-3

The model architecture is discussed in the section 3.3, are compiled with Adam optimizer for learning rate 1e-2. Categorical cross-entropy is considered as a loss function to be optimised with categorical accuracy as the metric. Each model is fitted on the training dataset. The batch size is taken as 64. The maximum number of epochs is kept as 30. The model trained is evaluated on the validation data.

The average time taken by the model per epoch is 13 seconds on dataset-1, which has increased drastically for dataset-3 to 90 seconds. The number of epochs required to reach the optimal state, neither underfitting nor overfitting, is increased from 7 to 10 over the datasets.

The value of the BLEU score had improved slightly from 10.7468 for dataset-1 to 13.0864 for dataset-2 but it is worth noting that the model has shown worse performance the dataset-3 having the higher number of translation pairs of data.

	Both LSTM		
	Dataset-1	Dataset-2	Dataset-3
BLEU score	10.7648	13.0864	12.382
Average time take per epoch (s)	13	23	90
Number of epochs	7	7	10

Table-4: The metrics, BLEU score, Average time take per epoch (s) and Number of epochs on the encoder-decoder model with LSTM as both encoder and decoder generated for three datasets: Dataset-1, Dataset-2 and Dataset-3

4.3.Model-3: GRU as an encoder and LSTM as a decoder

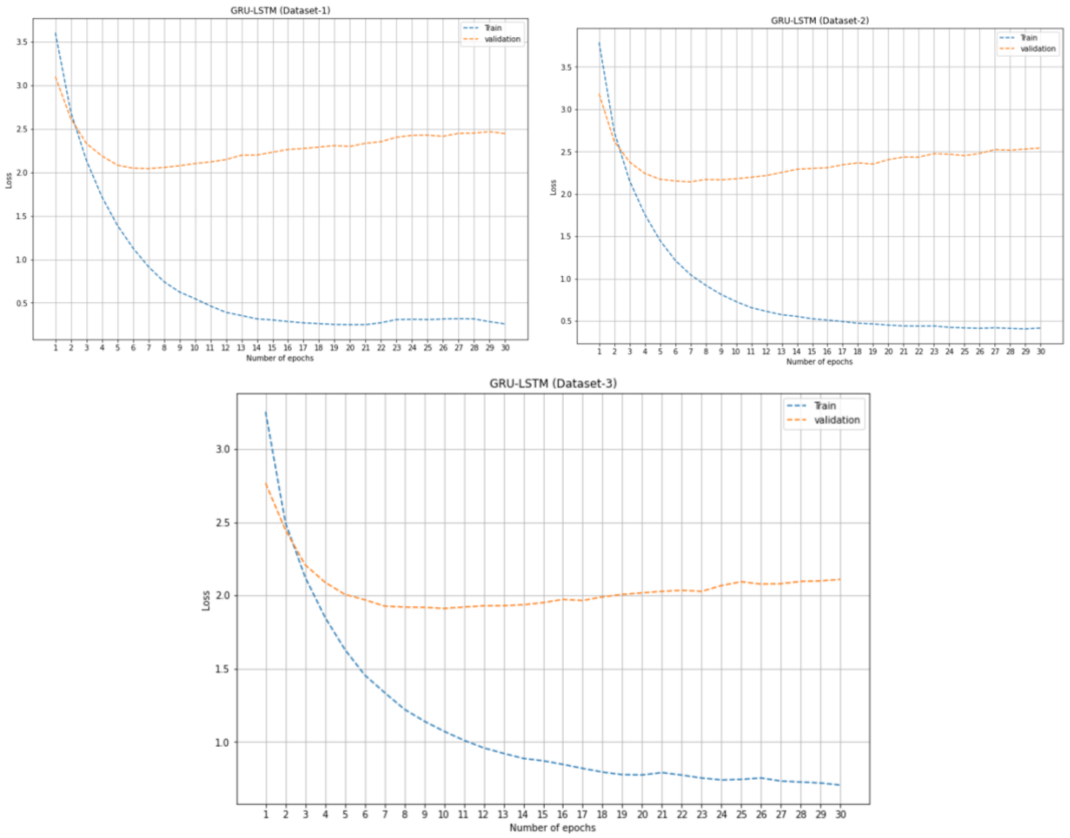


Figure (17): Training and Validation loss values for the model with GRU as an encoder and LSTM as an decoder with 30 epochs. The three subplots are for the three datasets, dataset-1, dataset-2 and dataset-3

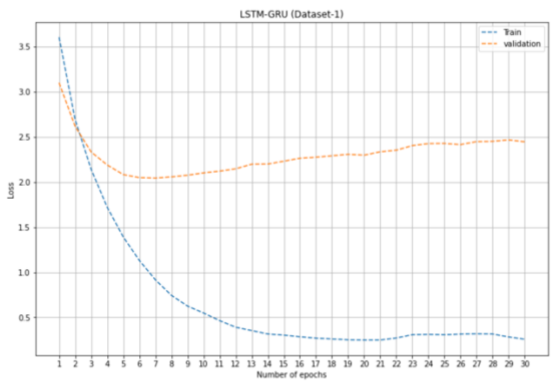
The model architecture is discussed in the section 3.4, are compiled with Adam optimizer for learning rate 1e-2. Categorical cross-entropy is considered as a loss function to be optimised with categorical accuracy as the metric. Each model is fitted on the training dataset. The batch size is taken as 64. The maximum number of epochs is kept as 30. The model trained is evaluated on the validation data.

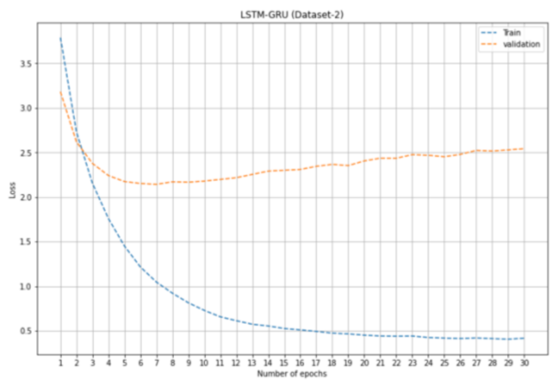
The average time taken by the model per epoch is 11 seconds on dataset-1, which has increased drastically for dataset-3 to 88 seconds. The number of epochs required to reach the optimal state, neither underfitting nor overfitting, is increased from 8 for dataset-1 to 10 for dataset-2. But is worth noting that the number of epochs had decreased to 7 for dataset-3 having the higher number of translation pairs of data. The value of the BLEU score had improved slightly from 10.5928 to 14.3882 over the three datasets.

	GRU-LSTM		
	Dataset-1	Dataset-2	Dataset-3
BLEU score	10.5928	12.7554	14.3882
Average time take per epoch (s)	11	22	88
Number of epochs	8	11	10

Table-5: The metrics, BLEU score, Average time take per epoch (s) and Number of epochs on the encoder-decoder model with GRU as an encoder and LSTM as a decoder generated for three datasets: Dataset-1, Dataset-2 and Dataset-3

4.4.Model-4: LSTM as an encoder and GRU as a decoder





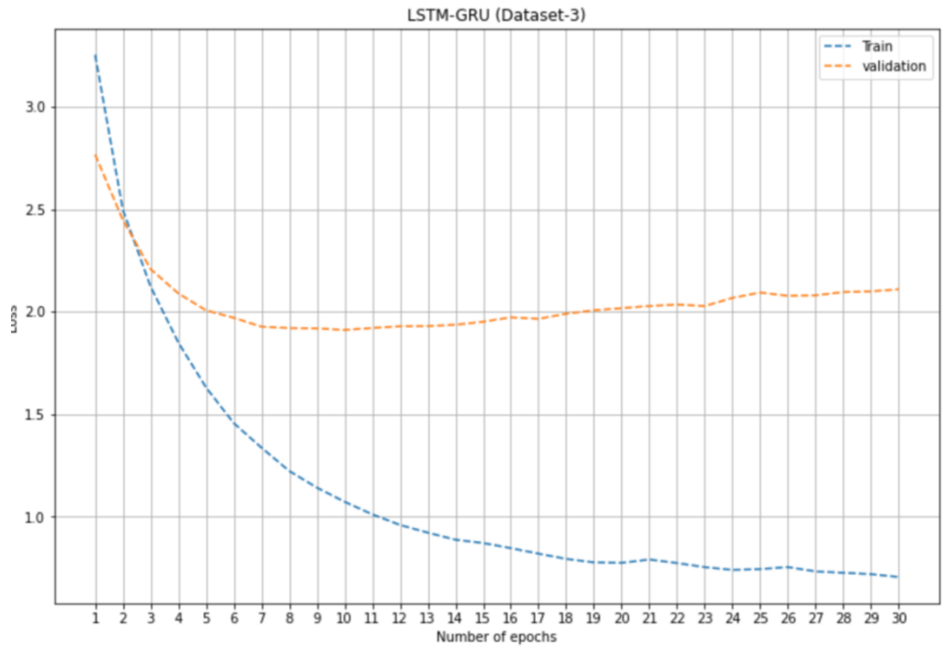


Figure (18): Training and Validation loss values for the model with LSTM as an encoder and GRU as an decoder with 30 epochs. The three subplots are for the three datasets, dataset-1, dataset-2 and dataset-3

The model architecture is discussed in the section 3.5, are compiled with Adam optimizer for learning rate 1e-2. Categorical cross-entropy is considered as a loss function to be optimised with categorical accuracy as the metric. Each model is fitted on the training dataset. The batch size is taken as 64. The maximum number of epochs is kept as 30. The model trained is evaluated on the validation data.

The average time taken by the model per epoch is 12 seconds on dataset-1, which has increased drastically for dataset-3 to 90 seconds. The number of epochs required to reach the optimal state, neither underfitting nor overfitting, is increased from 6 for dataset-1 to 9 for dataset-2. But is worth noting that the number of epochs had decreased to 8 for dataset-3 having the higher number of translation pairs of data. The value of the BLEU score had improved from 9.2681 to 14.459 over the three datasets.

	LSTM-GRU		
	Dataset-1	Dataset-2	Dataset-3
BLEU score	9.2681	12.5938	14.459
Average time take per epoch (s)	12	23	90
Number of epochs	6	9	8

Table-6: The metrics, BLEU score, Average time take per epoch (s) and Number of epochs on the encoder-decoder model with LSTM as an encoder and GRU as a decoder generated for three datasets: Dataset-1, Dataset-2 and Dataset-3

4.5.Comparison of four encoder-decoder models:

Comparing the four encoder-decoder models together over the three datasets from Figure (19) , the value of the BLEU score for dataset-1 is slightly better for Both-LSTM and GRU-LSTM models in comparison to the other two. The values of BLEU score for dataset-2 is slightly better for Both-LSTM model, followed closely by GRU-LSTM and LSTM-GRU models. For dataset-3, the values of BLEU score for LSTM-GRU and GRU-LSTM are close and better than Both-GRU and Both-LSTM models.

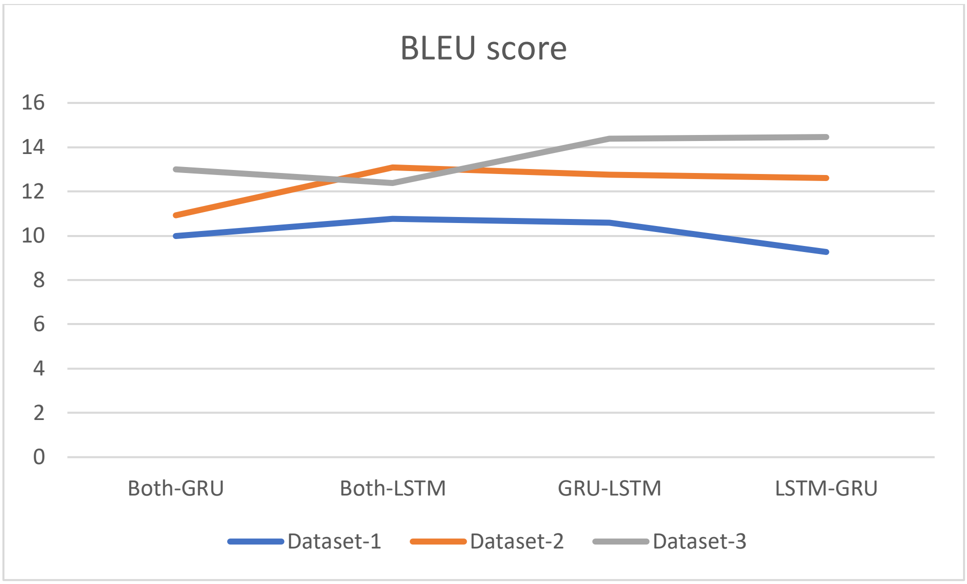


Figure (19): BLEU score for the four models with three datasets

Coming to the approximate average time taken per epoch over the three datasets, they are very close to each other for all the models. The Both-GRU model having a slightly better time, in comparison to others.

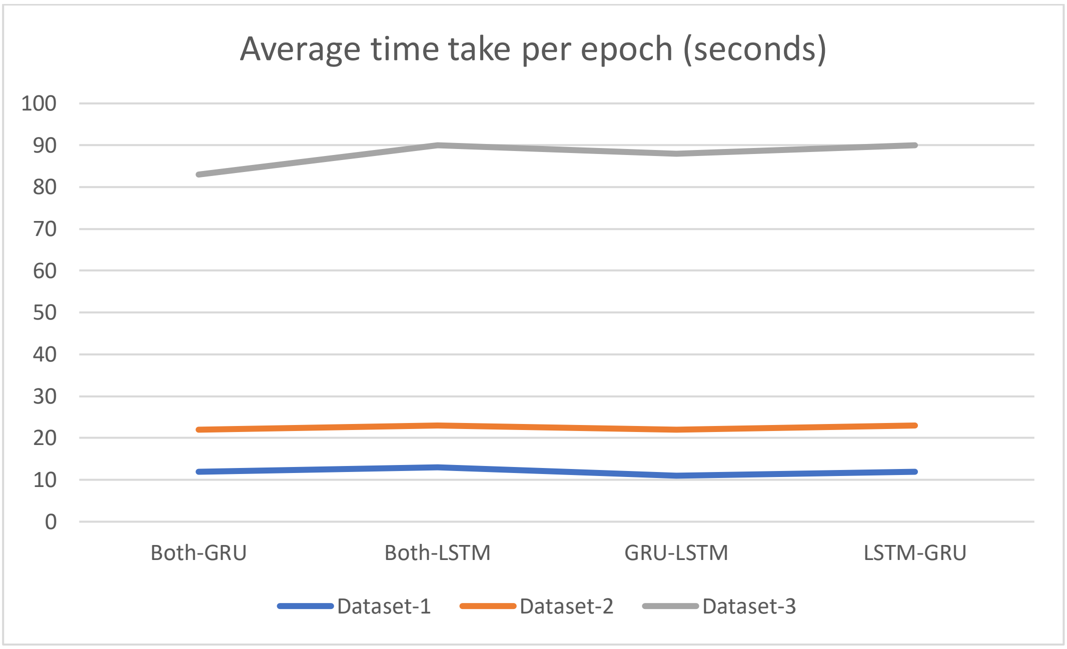


Figure (20): Average time taken per epoch for the four models with three datasets

Finally, for the number of epochs taken by models to reach optimal state, neither overfitting nor underfitting is compared over the three datasets. The LSTM-GRU model is converging to the optimal state in a fewer number of epochs in comparison to the other three models and the model is taking lesser epochs for dataset-3 in comparison to dataset-2. The Both-LSTM is converging relatively quicker than the GRU-LSTM model. With both outperforming the Both-GRU model.

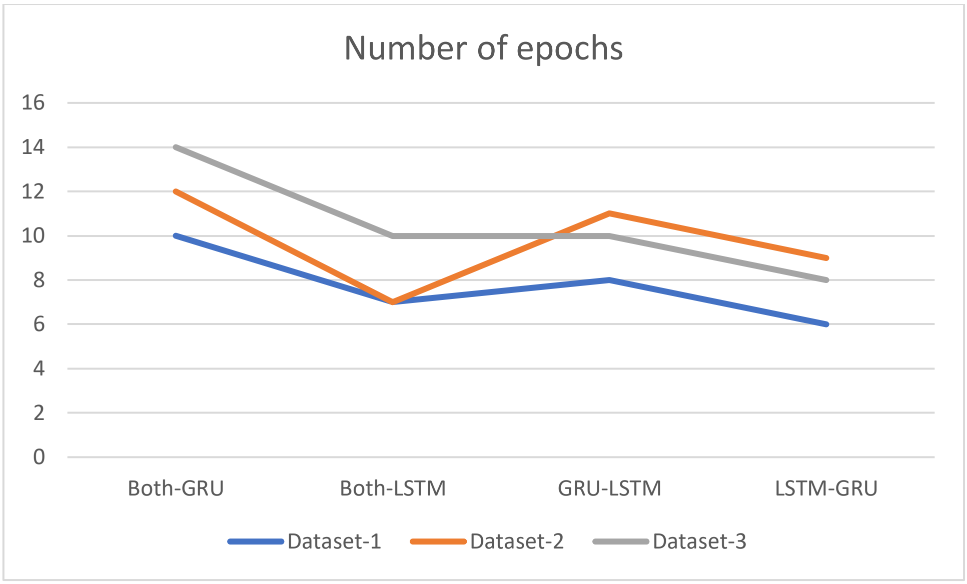


Figure (21): Number of epochs required by each model to reach optimal value while training

5. Conclusion

The main aim of the project is to compare the four possible encoder-decoder models built using two versions of RNN architectures LSTM and GRU with varying dataset sizes. From the analysis carried out over the four models using the BLEU score metric, for smaller dataset sizes, the encoder-decoder model with both encoder and decoder as GRU, will be quicker due to its computationally inexpensive nature than LSTM and will result in similar levels of translation quality. But for larger dataset set sizes, using the encoder-decoder model with LSTM as an encoder and GRU or model with GRU as an encoder and LSTM as a decoder will result in similar translation quality, but the prior model will converge to optimal state in slightly fewer epochs in comparison to the later one. Hence, it is preferred.

6. Future Scope

In this project, only three different sizes of the original dataset having over 194000 translation pairs. Considering the original dataset or large proportion of it will definitely improve the quality of translation using the mentioned model.

In this project, the baseline models are built with default values for arguments and there is scope for massive improvements by hyperparameter tuning each model for optimal performance.

As investigated by Cho. the use of an attention mechanism had proven to improve the performance of the encoder-decoder model by considering the fuller representation model allowing the model to pay attention to the different parts of input for each output to be generated by the decoder.

The developed encoder-decoder models can be used on applications involving temporal data, exploring the possibility beyond the neural machine translation.

7. Ethical, Legal, Professional and Social issues

7.1. Ethical and Legal issues:

- At any stage of the project, human participants were not involved
- The dataset used in the project is open-sourced and have permission to use for educational purposes.
- No personal information of the users is present in the dataset.

7.2. Professional issues:

- I have tried to conduct the research at every stage of the project to best of my abilities following the guidelines set by the University.
- I have used proper citations acknowledging the original author while using his/her work

7.3. Social issues:

- The project at its core compares the performance of encoder-decoder model using different RNN cells. Any part of the project does not a chance for human participation in any shape or form.

8. References

- Alharbi, A., Smith, P. and Lee, M., 2021. Enhancing Contextualised Language Models with Static Character and Word Embeddings for Emotional Intensity and Sentiment Strength Detection in Arabic Tweets. *Procedia Computer Science*, 189, pp.258-265. [Accessed 14 June 2021].
- Bahdanau, D., Cho, K. and Bengio, Y., 2015. NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE. [online] Available at: < https://arxiv.org/pdf/1409.0473.pdf >[Accessed 28 July 2021].
- Bensalah, N., Habib, A. and Abdellah, A., 2021. LSTM vs. GRU for Arabic Machine Translation. [online] Available at: <https://www.researchgate.net/publication/350904826_LSTM_vs_GRU_for_Arabic_Machine_Translation> [Accessed 28 July 2021].
- Cho, K., Merrienboer, B., Bahdanau, D. and Bengio, Y., 2014. On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. [online] Available at: <https://arxiv.org/pdf/1409.1259.pdf> [Accessed 14 June 2021].
- Cho, K., Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. [online] Available at: <https://arxiv.org/pdf/1406.1078.pdf> [Accessed 20 June 2021].
- Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R. and Makhoul, J., 2014. Fast and robust neural network joint models for statistical machine translation. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, [online] Available at: <https://aclanthology.org/P14-1129.pdf> [Accessed 25 August 2021].
- Elman, J., 1990. Finding Structure in Time. *Cognitive Science*, 14(2), pp.179-211 [Accessed 14 June 2021].
- Google Cloud. n.d. *Google Cloud*. [online] Available at: <https://cloud.google.com/translate/automl/docs/evaluate#automl_translate_get_model_evaluation-python> [Accessed 13 August 2021].
- Hanin, B., 2018. Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients?. [online] Available at: <https://arxiv.org/pdf/1801.03744.pdf> [Accessed 25 August 2021].
- Hochreiter, S. and Schmidhuber, J., 1997. Long Short-Term Memory. [online] Available at: <https://watermark.silverchair.com/neco.1997.9.8.1735.pdf > [Accessed 25 August 2021].
- Ibm.com. 2019. *What are Recurrent Neural Networks?*. [online] Available at: <https://www.ibm.com/cloud/learn/recurrent-neural-networks> [Accessed 28 July 2021].
- Kalchbrenner, N. and Blunsom, P., 2013. Recurrent Continuous Translation Models. [online] Available at: <https://aclanthology.org/D13-1176.pdf> [Accessed 25 August 2021].
- Kane, V., 2021. Interpretation and machine translation towards google translate as a part of machine translation and teaching translation. *Applied Translation*,
- Kostadinov, S., 2019. *Understanding Encoder-Decoder Sequence to Sequence Model*. [online] Medium. Available at: <https://towardsdatascience.com/understanding-

encoder-decoder-sequence-to-sequence-model-679e04af4346?gi=66a48b516c87> [Accessed 28 July 2021].

- Luong, M., Pham, H. and Manning, C., 2015. Effective Approaches to Attention-based Neural Machine Translation. [online] Available at: <https://arxiv.org/pdf/1508.04025.pdf> [Accessed 25 August 2021].
- Ma, C., 2021. Encoder-Decoder Attention ≠ Word Alignment: Axiomatic Method of Learning Word Alignments for Neural Machine Translation. *Journal of Natural Language Processing*, 28(2), pp.694-699.
- Manythings.org. 2021. *English-French Sentences from the Tatoeba Project*. [online] Available at: <http://www.manythings.org/bilingual/fra/> [Accessed 27 August 2021].
- Nagao, M., 1984. A framework of a mechanical translation between Japanese and English by analogy principle. [online] Available at: <https://dl.acm.org/doi/10.5555/2927.2938> [Accessed 17 August 2021].
- Sekkate, S., Khalil, M., Adib, A. and Ben Jebara, S., 2019. An Investigation of a Feature-Level Fusion for Noisy Speech Emotion Recognition. *Computers*, 8(4), p.91.
- Sutskever, I., Vinyals, O. and Le, Q., 2014. Sequence to Sequence Learning with Neural Networks. [online] Available at: <https://papers.nips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf> [Accessed 28 June 2021].
- Xu, X. and Liu, H., 2020. ECG Heartbeat Classification Using Convolutional Neural Networks. *IEEE Access*, 8, pp.8614-8619.

10. Appendices:

10.1. Cleaning the dataset

```
# Cleaning the lines by removing all the non-printable characters,
punctuation characters
# Given a list of lines cleaning them

import re
def pairs_clean(lines_from_text):

        new_cleaned = list()

        # using regular expression for removing non-printable characters
        regular_non_print = re.compile('[^%s]' %
re.escape(string.printable))

        # using regular expression for removing punctuation characters and
obtaining translation table
        table = str.maketrans('', '', string.punctuation)

        for pair in lines_from_text:

                clean_pair = list()
                for new_line in pair:

                        # normalization to remove canonical and
compatibility related issues
                        new_line = normalize('NFD',
new_line).encode('ascii', 'ignore')
                        new_line = line.decode('UTF-8')

                        # tokenizing the white space
                        new_line = new_line.split()

                        # normalizing the text to lowercase
                        new_line = [word.lower() for word in new_line]

                        # removing punctuation from each token using regular
expression table
                        new_line = [word.translate(table) for word in
new_line]

                        # removing non-printable characters using the above
regular expression
                        new_line = [re_print.sub('', w) for w in new_line]

                        # removing the non-alphabetic tokens such as numbers
                        new_line = [word for word in new_line if
word.isalpha()]

                        # store as string
                        clean_pair.append(' '.join(new_line))
                new_cleaned.append(clean_pair)
        return array(new_cleaned)
```

10.2. Both GRU

```
# Model 1: Both Encoder GRU and Decoder GRU cell
def both_gru(french_vocab, english_vocab, french_timesteps,
english_timesteps, num_units):
        learning_rate = 1e-2
        model = Sequential()
        model.add(Embedding(french_vocab, num_units, input_length=
french_timesteps, mask_zero=True))
        model.add(GRU(num_units))
        model.add(RepeatVector(english_timesteps))
        model.add(GRU(num_units,return_sequences=True))
        model.add(TimeDistributed(Dense(english_vocab,
activation='softmax')))
        return model
```

10.3. Both LSTM

```
# Model 2: Both Encoder LSTM and Decoder LSTM cell
def both_lstm(french_vocab, english_vocab, french_timesteps,
english_timesteps, num_units):
        learning_rate = 1e-2
        model = Sequential()
        model.add(Embedding(french_vocab, num_units, input_length=
french_timesteps, mask_zero=True))
        model.add(LSTM(num_units))
        model.add(RepeatVector(english_timesteps))
        model.add(LSTM(num_units,return_sequences=True))
        model.add(TimeDistributed(Dense(english_vocab,
activation='softmax')))
        return model
```

10.4. GRU as an encoder and LSTM as a decoder

```
# Model 3:  Encoder GRU and Decoder LSTM cell

def gru_lstm(french_vocab, english_vocab, french_timesteps,
english_timesteps, num_units):
        learning_rate = 1e-2
        model = Sequential()
        model.add(Embedding(french_vocab, num_units, input_length=
french_timesteps, mask_zero=True))
        model.add(GRU(num_units))
        model.add(RepeatVector(english_timesteps))
        model.add(LSTM(num_units,return_sequences=True))
        model.add(TimeDistributed(Dense(english_vocab,
activation='softmax')))
        return model
```

10.5. LSTM as an encoder and GRU as a decoder

```
# Model 4:  Encoder LSTM and Decoder GRU cell

def lstm_gru(french_vocab, english_vocab, french_timesteps,
english_timesteps, num_units):
        learning_rate = 1e-2
        model = Sequential()
        model.add(Embedding(french_vocab, num_units, input_length=
french_timesteps, mask_zero=True))
        model.add(LSTM(num_units))
        model.add(RepeatVector(english_timesteps))
        model.add(GRU(num_units,return_sequences=True))
        model.add(TimeDistributed(Dense(english_vocab,
activation='softmax')))
        return model
```

10.6. Model evaluation using BLEU score

```
# Evaluating the performance of each model using BLEU score by comparing
predicted result to original/expected sequences


def model_evaluation(model_name, tokenizer_used, sources_text,
raw_text_dataset):
        actual_value, predicted_value = list(), list()
        bleu_scores = []
        for i, j in enumerate(sources_text):
                # translating the encoded input text sequence
                j = j.reshape((1, j.shape[0]))
                translation = predict_sequence(model_name, eng_tokenizer,
j)
                raw_target, raw_src = raw_text_dataset[i][0],
raw_text_dataset[i][1]

        # Printing 50 French to English translations by the model

                if i < 50:
                        print('French(Source) : %s\nTarget : %s\nPredicted :
%s \n' % (raw_src, raw_target, translation))


                actual.append([raw_target.split()])
                predicted.append(translation.split())

    # Calculating BLEU score
        bleu_score = corpus_bleu(actual, predicted, weights=(0.25, 0.25,
0.25, 0.25))



        # Print BLEU score
        print('BLEU score: %f' % bleu_score)

        return bleu_score
```