Anish Das

Investigating the effect of Translation Quality on Summarisation Quality

Computer Science Tripos – Part II

Trinity Hall

14th May 2021

Declaration

I, Anish Das of Trinity Hall, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. I am content for my dissertation to be made available to the students and staff of the University.

Signed: Anish Das

Date 14th May 2021

Proforma

Name:	**Anish Das**
College:	**Trinity Hall**
Project Title:	**Investigating the effect of Translation Quality on Summarisation Quality**
Examination:	**Computer Science Tripos – Part II, May 2021**
Word Count:	**11969 words**[1]
Lines of Code:	**1970**[2]
Project Originator:	Dr Andrew Caines & The Author
Supervisor:	Dr Andrew Caines & Dr Zheng Yuan

Original Aims of the Project

This project seeks to investigate the effect of translation quality on summarisation quality in order to estimate the feasibility of a Cross-Lingual Summarisation to be scaled in the absence of good translation models. It involved training Neural Machine Translation (NMT) models using the Transformer architecture for multiple source languages into English. The next was applying a pre-trained state-of-the-art summarisation model to the translations produced by the models. The final step involved testing the correlation between the BLEU scores (translation metric) and the ROUGE scores (summarisation metric).

Work Completed

This project has been successful in achieving the success criteria listed in the project proposal. First, the Transformer architecture was built from scratch and used to create multiple translation models by saving the parameters at regular intervals during the training process. The checkpointed translation models have been used to translate the documents in the evaluation GV-crowd/snippet dataset from[19]. The translated sentences were then summarised. The BLEU and ROUGE metrics were calculated along with a quantitative comparison of the two scores.

[1]Calculated in Overleaf using: `\immediate\write18{texcount -1 -merge -q main.tex > main-words.sum}` and `\input{main-words.sum} words` (which includes captions and footnotes) (`%TC:ignore ... %TC:endigore`) was applied around the content before Introduction and after Conclusion.

[2]Calculated using: `git ls-files | grep '\.py' | xargs wc -l`. This includes the colab notebooks after they were converted to '.py' files.

Special Difficulties

No special difficulties were encountered, apart from having to adjust to the COVID-19 pandemic.

Acknowledgements

A huge thanks to my supervisors, Andrew P. Caines and Zheng Yuan, whose support throughout the project was immense. I would also like to extend my gratitude to my Director of Studies and my friends for their comments to help polish this dissertation.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Within this vast and varied field of NLP lies the task of Cross-Lingual Summarisation (CLS), which aims to produce a summary in one specific target language from a source document in another languages. This is an essential task because it will lift the language barrier, for example enabling easy access to information for all. However, the amount of textual data on the internet is immense and continues to grow day by day. Today it is impossible to keep up with it because a person simply does not have enough time to read it all. By summarising the data, we trim it down to its most salient points, which can be consumed quickly, thus, allowing humans to consume a more significant amount of information and in a shorter amount of time. Suppose a CLS system is deployed on a large scale: it will bring about an exponential increase in the amount of information available to everyone and thus empower humans with the knowledge of the diverse world we live in today. Such a system will have immense and varied applications, for example, providing news from all over the world (very important in countries where the news can be very biased), access to information for academic purposes (there are already annual conferences on medical paper translations which reflects their importance), access to public health information in global health emergencies and many more, all in a condensed format.

1.2 Project Focus

CLS has two main steps: first, **translating** the source document into a target language and second, **summarising** the translated text. Since CLS is a two-step process, any errors in the first step will be carried forward and possibly amplified through the second step and end up in the resulting summary. Further, it is widely known that we require large sentence-aligned datasets to build good quality translation models now that machine translation is largely driven by neural networks. Language pairs that have these large datasets are called 'high-resource' in the NLP community and those without are called 'low-resource' language pairs. The general trend with NMT and corpus size is shown in the Figure 1.1 from paper[13]. Particularly, note that the green-line representing NMT improves as the corpus size is increased. Therefore, translation models trained on '*low-resource*' language pairs do not come across enough sentences and end up having quite a few inaccuracies in their translations. This raises the question of whether

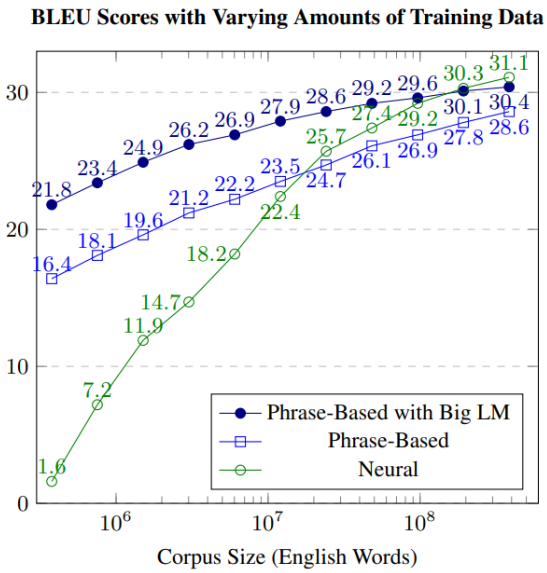

Figure 1.1: The variation of BLEU scores with respect to the corpus sizes.

bad-quality translations will adversely affect the information content and the readability of the resultant summary.

Since there are language pairs which are '*low-resource*', it is necessary to ascertain the feasibility of building a CLS system for such language-pairs. With this as a foundation, through this project, I want to explore the effect of translation quality on summarisation quality. To achieve this I will be creating, training, and comparing translation models of differing quality. Moreover, I will also be comparing the summaries produced against the baselines i.e. the hand-written summaries and web-scraped summaries. Translation models, like most machine learning systems, improve with more training and therefore, a model's features are check-pointed at regular intervals yielding models of varying quality. The earlier models will be used as proxy for worse translation models and the later models for better translation models.

1.3 Related Work

My project is inspired by the paper[19], the authors set out to build a dataset for evaluating cross-lingual summarisation methods. They used the Global Voices[1] to collect news articles in 15 different languages. Moreover, they crowd-sourced human summarisation of the articles collected in English and web-scraped the *'snippet'* summaries available on the website as a gold standard reference. Finally, they trained four translation models, two of which were trained on a large scale dataset and the other two using a much smaller dataset, thus, yielding models with different quality. Finally, they compared the summaries from the resultant translations with "*gold-standard*" using ROUGE scores.

[1] `https://globalvoices.org/`

Example summaries

Example Crowd Summary

```
``There was a conference in Casablanca this past weekend for DABA. DABA is
   an association that aims to mobilize Morocco for the 2007 elections.
   They do this by training people in their citizen journalism skills, so
   that more people can hear about their issues.''
```

Example Snippet Summary

```
``Morocco author Jillian York addressed a youth conference, organised by
   DABA, in conjunction with the National Democratic Institute's Morocco
   branch. The three-day event, which aimed at reevaluating political
   involvement in Morocco in the run-up to the 2007 legislative elections,
   also focused on building the capacity of young opinion leaders in
   citizen journalism skills, with an online emphasis in order to increase
   dialogue about issues of concern to youth.''
```

This project differs from[19] in the following ways. Instead of using multiple language-pairs with different dataset sizes, this project considers multiple models of the same language pair and trained on the same dataset and only varying the time it has been trained. This project extends the investigation carried out by the authors, to compare multiple models of the same language pair as opposed to across different language pairs with differing dataset size. This idea was born from the intuition that some articles may be less compressible to begin with. Further, it is hard to conclusively compare the BLEU score between different languages. Therefore, to truly evaluate the effects of translation on summarisation, I wish to reduce the number of confounding variables, and thus, check-point models during the training process after every epoch were used instead.

The architecture used to implement the translation models is the Transformer architecture[29]. Details of its conception and implementation are detailed in the following chapters. Finally, transformers are a deep architecture which are over 200MB in size, which makes them very unstable in the training process and the investigation carried out by Popel & Bojar in the article[22] helped guide the training process.

Chapter 2

Preparation

2.1 Starting Point

This project uses pre-trained summarisation models from the paper[7]. All code in this project, for the translation model architecture, training algorithm, pre-processing and post-processing and finally evaluation code, was developed from scratch using some well-known libraries which are listed in the Project Management section. The main focus of this project is in building/training/deploying translation models.

2.2 Background Knowledge

This section is meant to elaborate on content that is beyond the Part 1B in the Computer Science Tripos.

2.2.1 Evolution of Translation models

Assuming that the reader knows about the Perceptron and the Neural Network(NN)[17], this subsection aims to explain the steps taken to improve Translation models, which ends with the Transformer.

How do we model sequences?

As NNs have become popular, a need to apply these novel tools to sequences came up such as Machine Translation. However, the sequence modelling problem can not be solved with standard NNs since their input size is fixed whilst sequences can vary. There were attempts made where the input-sequence would be padded or truncated to fit the input-size. Nevertheless, the Neural Nets can't model sequences because it processes the sequence as one item begin whereas we need to processes sequences one word at a time while building up a context. Recurrent-Neural-Networks[24] were suggested to solve this problem, because they can address something vital about the language that is the contextual meaning, relations between words and incremental processing. For every word in the sequence the context (*hidden-vector*) gets updated based on the current context and the word. This structure is key that allows an RNN can accommodate

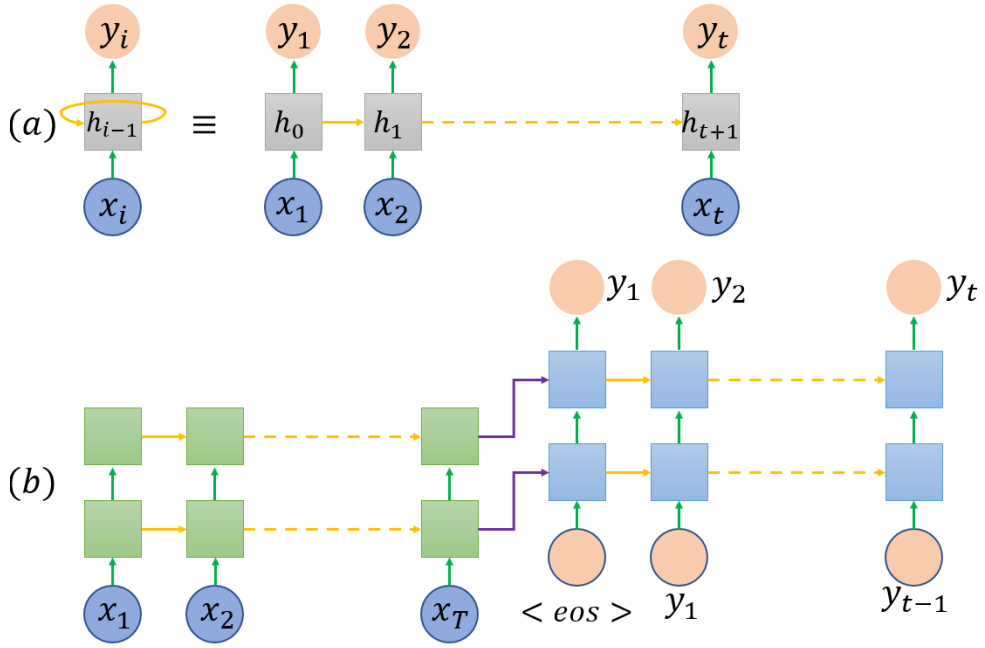

Figure 2.1: (a) sequence model diagram. (b) seq2seq model diagram like NMT model.

input sequences of variable length provided the length isn't too big because of computational restrictions.

Sequence2Sequence models

Sequence models can be used to build NMT models, however, it would be akin to a word by word translation. Because, in the Machine Translation task between Spanish and English the relative positions of the adjectives and the nouns are reversed. Therefore, in order to accurately translate between them the sequence-model would have to know the next word. Therefore, Sequence-to-Sequence (seq2seq) models[27] were born. In NMT, the input sequence is passed through an encoder RNN which produced a context-vector for each word in the sequence. This context-vector is then passed to a decoder RNN which produces the translation of the input.

However, RNNs are unable to identify relations between words over a large distance. Novel recurrent units have been suggested such as the GRU(Gated-Recurrent-Unit) and the LSTM(Long-Short-Term-Memory) which achieved varying levels of success.

Attention

Lately attention mechanisms[4] have been introduced to NMT to improve the Translation quality. The attention mechanism aims to link up parts of sequences separated by a long distance but are still related. For example:

`X is from France and his mother tongue is French`

There are two forms of this Attention mechanism: Bahadanau Attention[4] and Self-Attention[1][29]. Bahadanau Attention is applied in the decoding step and uses a weighted

[1]Note: there are many versions of self-attention but this project implements the self-attention described by Vaswani et al.

sum of all the previous context vectors for each time step when making a prediction. Because of how the weights are calculated, it is very computationally heavy and not parallelizable. On the other hand, self-Attention works on the whole input sequence at once and transforms it into another sequence of the same size but containing the information about how related a word is in the sequence is to others.

Transformers

Finally, in the paper[29], the authors explored which of attention or recurrency was doing the heavy-lifting and therefore, was more important. The architecture they came up with was the Transformer which proved that attention is more important, i.e. which helps improve the translation quality of a models. Since then Transformers have widely become the norm for translation models requiring fewer training steps and resources to surpass state-of-the-art models. This is just a brief introduction to the Transformer. A more in-depth description and the decisions taken in its creation are detailed in section 3.2.

2.2.2 Self-Attention

Basic Self-attention

Self-attention is the most important part of the Transformer architecture. In a nutshell, self-attention is an operation that transforms a sequence of vectors $(x_1, x_2, ..., x_t)$ into another sequence of vectors $(y_1, y_2, ..., y_t)$. And in order to calculate the output-vector y_i the following computation takes place (shown in Figure 2.2):

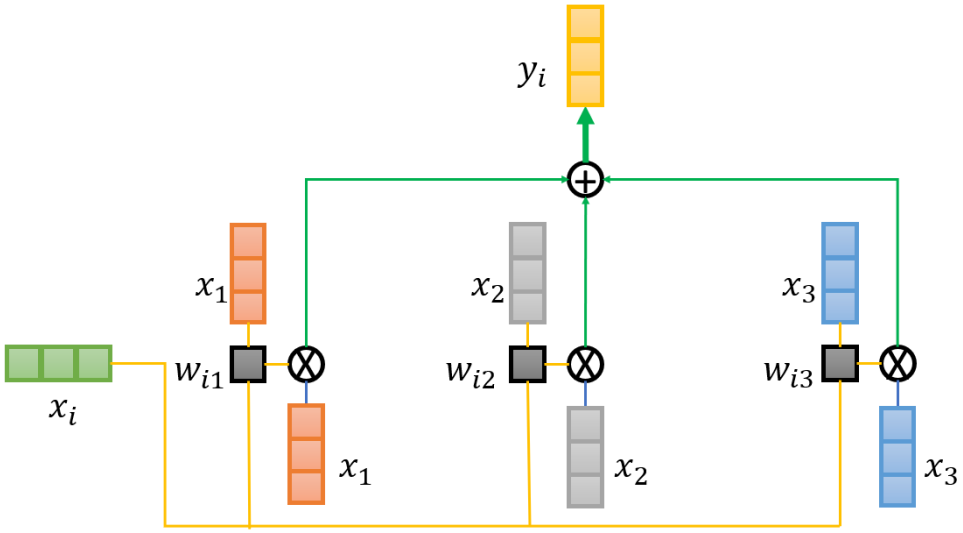

Figure 2.2: Basic self-attention operation illustrating the computation involved in computing y_i

$$w'_{ij} = x_i^\top \cdot x_j \quad \text{then} \quad w_{ij} = \frac{exp(w'_{ij})}{\sum_j exp(w'_{ij})} \quad \text{then} \quad y_i = \sum_j w_{ij} \cdot x_j$$

Here j indexes the whole sequence, and the weights w_{ij} aren't parameters but derived from a function of the input vectors x_i and x_j. The weights w_{ij} are constructed from w'_{ij} by applying

the softmax-operation across each column (or j in this case) in order to prevent the values from becoming too big/small.

Understanding Self-Attention

This is a complicated process and it helps to make sense of why/how it works. Consider the problem of creating a recommendation system for books. One way to go about this is to go through the complicated and costly process of building features or vectors by hand. Let's say each dimension in the vector represents a genre (like drama, romance or action) and for books, it defines how romantic or dramatic or action-heavy it is. For users, it defines how much the user likes drama or romance or action. Then for each pair of a book ($b_1, b_2...$) and user ($u_1, u_2...$), we can take the inner-product of the two vectors to obtain a score:

$$\text{score} = \sum_i b_i \cdot u_i$$

A high score tells us that the user will likely enjoy the book and a low score means they will most likely not enjoy the book. Intuitively, if a book features a lot of romance, it will have a high component in the romance dimension and if a user also likes romance, they will also have a high component in the romance dimension. This means that two large numbers get multiplied together, increasing the score. On the other hand, if both the user and the book have a negative component for a dimension, the product will be positive and augment the score. In all other cases it would be small/negative leading to a low-score.

In a similar way, the translator model learns the embeddings for the words in such a way that related-words should end up having higher scores and therefore, the corresponding embeddings contribute more toward the resultant vector.

Properties of Self-Attention

Self-attention doesn't see its input as a sequence and instead views it as a set. This makes it parallelizable, which is a significant advantage when compared to Bahadanau Attention. Further, if we permute the input sequence, then the output is the same, just permuted with the same operation which means it is not harder to form a link between words that are far apart compared to the ones that are close together.

Finally, the most important property of the Self-attention operation is its ability to decipher a link between words far-apart or close-together with same computation and this improved ability to model relationships is crucial for a translation.

2.2.3 Automatic Text Summarisation

Automatic Text summarisation produces a concise and fluent summary while keeping the critical pieces of information and the overall message intact. This project aims to examine the effect of translation on summarisation and therefore, in this project I chose to concentrate on the Translator model more. However, I still need to build up an understanding of how automatic summarisation is traditionally performed.

Extractive Vs. Abstractive

There are two types of summarisation methods: Extractive and Abstractive as described in the book[10]. As the name suggests, Extractive-Summarisation aims to extract the text's critical information verbatim by employing various methods to rank phrases in the text and pick the ones it deems most important. Hence, it does not lead to very fluent summaries. On the other hand, Abstractive-summarisation methods interpret and understand the text before generating the summary with the most crucial information. Although the summaries produced by abstractive methods output more readable summaries, they often lack essential information because they must deal with the complex problem of semantic correctness.

Bottom-Up Abstractive Summarisation

In the paper[7], the authors came up with a novel method to use extractive methods as a content selector to overproduce phrases and sentences from the text that should be incorporated into the summary. Then they used this content selector as Bottom-Up-attention to restrict the abstractive step to the chosen key phrases, thus boosting the ability of the abstractive summaries to compress text while still retaining the ability to produce fluent summaries.

2.2.4 Byte-Pair Encoding

The necessity of Subwords

All NLP tasks are trained on some fixed vocabulary, even though the problem they are trying to solve almost always should generalise to new texts and therefore an unlimited vocab. This means that we will regularly come across out-of-vocabulary(OOV) words when we are testing and evaluating the system. When it comes to translation models, they are expected to break compound words into their parts and translate them.

Intuitive explanation of why it works

For example

$$\texttt{brushed} \longrightarrow \texttt{brush + ed}$$

As described here, the verb *"brush"* is modified to a past tense form with the addition of the *"ed"* at the end. The model now needs to learn to translate each subword separately which reduces the overall complexity. Now the model does not need to learn the translation of all the conjugations of the verb.

This improved ability to deal with out-of-vocabulary words comes at the cost of computation because the average sequence length is increased.

BPE

One way to accomplish this is to use subwords, proposed by Sennrich et al. in the paper[25]. As described above, subword tokenization performs better than traditional tokenize-by-white-spaces by reducing the complexity of the translation and being able to handle out-of-vocabulary words.

Byte-Pair Encoding is one method to perform subword tokenization suggested in Sennrich et al. It is the method employed in this project. This algorithm has the same name and is based on the Loss-Less data-compression algorithm. Once the entire text is loaded, the algorithm follows the steps:

1. Iteratively all character pairs are counted
2. The most frequently occurring pair is concatenated to form one word

These steps are repeated until the target vocabulary size is reached or the target number of merge operations is exhausted.

2.3 Evaluation Methods

2.3.1 BLEU

BLEU stands for Bilingual Evaluation Understudy. It is a precision-based metric that provides a score between 0-1 calculated for a piece of the translated text by comparing it to one or more reference translations. It was proposed in the paper[20], where the authors suggested a cheaper and faster than manually evaluating machine translation systems. It is language-independent, and most importantly, scores correlate highly with human-evaluation scores.

Modified Precision

A modified version of precision calculates the BLEU scores because MT have been known to predict a larger number of "reasonable words" to raise the score. This is illustrated in an example in the paper[20]:
candidate: `the the the the the the the`
reference 1: `The cat is on the the mat`
reference 2: `There is a cat on the mat`
Since precision is the ratio of the number of words in the candidate sentence that appears in the references to the total number of words in the candidate sentence, we will get 7/7 = 1, even though we see that the candidate sentence is nonsensical.

This brings us to the first modification that was made. We calculate the maximum frequency of all words in any one reference sentence. When we perform the precision calculation, we put a cap on the widespread words, i.e. the words with that most frequent. In the above example, "`the`" had a max frequency of **2**, and when we do the precision calculations, we get 2/7.

Another modification was the addition of *brevity-penalty*. We can trivially see that if we have a shorter candidate sentence, it will get a higher score. In order to prevent shorter sequences from getting a higher score, the brevity-penalty was introduced.

$$\text{BP} = e^{1-r/c}$$

where **c** is the length of the candidate sentence and **r** is the length of reference sentence closest to **c**.

Finally, it is important to remember that we are dealing with natural languages here and therefore, considering only unigrams will not be optimal. Hence, we use **n-grams** instead.

SacreBLEU

BLEU has been widely adopted as the standard for comparing translation models and this is why I will be using it as the metric to measure the quality of the translation models built. One final consideration in applying the BLEU metric comes from the paper[23], where the author discusses the various hidden parameters that significantly affect the final score. To sidestep this issue, the library developed by the authors in[23] called SacreBLEU [2] will be used to calculate the BLEU scores.

In Figure 2.3 an example output is presented. Starting on the left the first number is the BLEU score (*out of 100*) calculated by combining the precision scores. The next four values separated by the '/' are the unigram/bigram/trigram/4-gram precision scores. Finally, within the parenthesis the first item is the 'Brevity-Penalty' followed by the 'ratio' which is the ratio of the 'hyp_len' and 'ref_len'. For this project we are only concerned with the first value.

```
0.73 18.4/3.4/0.2/0.1 (BP=0.618 ratio=0.675 hyp_len=206 ref_len=305)
```

Figure 2.3: Example SacreBLEU output (*pretty printed*)

Interpreting BLEU

During the training process, at regular intervals, the model was evaluated on a test set and during the evaluation, it printed out some example translations. The BLEU scores can be interpreted in the following way:

- < 10: the model learns words and can translate the key information, but there are tons of repetitions.
- $10 - 20$: the translation produced are decent with some grammatical errors.
- $20 - 30$: the model can be classed as a good translator that conserves the meaning.
- > 30: the model is classed as a good fluent translator. Current state-of-the-art translators have achieved such scores.

I got this information from a Lecture[3] on "*Evaluating the Output of Machine Translation Systems*" by Alon Lavie.
Some of the caveats of BLEU is that it is not always interpretable by most professionals. Moreover, one can not compare BLEU scores between different languages and often within the same language with different test sets. Most importantly, Lavie claims that BLEU is not always correlated to the human-evaluation. Therefore, to reduce any variability in the final BLEU scores the SacreBLEU library was used along with the WMT News Translation Task test sets.

[2] https://github.com/mjpost/sacrebleu

[3] https://www.cs.cmu.edu/~alavie/Presentations/MT-Evaluation-MT-Summit-Tutorial-19Sep11.pdf

2.3.2 ROUGE

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. The metric to measure the quality of the summaries produced by the pre-trained summariser from[7] will be ROUGE . It is a set of metrics for evaluating automatic summarisation systems by comparing the candidate summaries produced against the reference summaries provided, which are usually handwritten as follows:

$$RECALL = \frac{\text{No. of words that match in both}}{\text{Total no. of words in the reference summary}}$$

For example:
Candidate summary: `The dog was lying under the bed.`
Reference summary: `The dog was under the bed.`
This would give us a ROUGE score of 1.0 (6 / 6)

However, since recall-oriented, the automatic summarisation system can produce very long sentences covering all the words in the reference, but this adds unnecessary verbosity to the summary produced. for example:
Candidate summary 2: `The big husky dog was under the tiny bed.`

Here we would also get a ROUGE score of 1.0, but it is unnecessarily long. This is where we would employ precision as well. It is defined as:

$$PRECISION = \frac{\text{No. of words that match in both}}{\text{Total no. of words in the candidate summary}}$$

Thus, we combine precision and recall together and report the F_1 score. F_1 is the Harmonic mean of precision and recall:

$$F_1 = \frac{2}{\text{RECALL}^{-1} + \text{PRECISION}^{-1}}$$

Different ROUGE metrics

What we have discussed until now is the ROUGE -1 score which is the unigram score and relates to just words as discussed above. But it is only one metric from one class of ROUGE metrics.

- **ROUGE -N**: extend the unigram model described above to incorporate *bigram/trigram* and further, *n-grams*
- **ROUGE -L**: We consider the candidate and reference summary as a sequence of words and then apply Longest Common Sequence algorithm[4] with the intuition that the length of the LCS is proportional to how similar they are. Further, we can define the F-measure with LCS as follows:

$$R_{lcs} = \frac{\text{LCS}(C, R)}{|C|} \quad \text{and} \quad P_{lcs} = \frac{\text{LCS}(C, R)}{|R|}$$

[4] covered in Algorithms Part 1A

where C is the candidate summary, R is the reference summary and (|.|) returns the total number of words in the summary.

- **ROUGE -W**: for this metric, we follow a very similar procedure to ROUGE -L except we use the weighted-LCS algorithm, which scores the alignment of the sentence with penalties for mismatch and skips.

Therefore, the ROUGE -F_1-score formulae are:

$$\text{ROUGE -N-}F_1 = \frac{2 \cdot |\text{common-n-grams}|}{|C| + |R|} \quad \text{and} \quad \text{ROUGE -L-}F_1 = \frac{2 \cdot \text{LCS}(C, R)}{|C| + |R|} \tag{2.1}$$

```
{'rouge-1': {
    'f': 0.336, 'p': 0.339, 'r': 0.345},
'rouge-2': {
    'f': 0.088, 'p': 0.089, 'r': 0.090},
'rouge-l': {
    'f': 0.280, 'p': 0.283, 'r': 0.285},
'rouge-w': {
    'f': 0.106, 'p': 0.169, 'r': 0.079}}
```

Figure 2.4: Example ROUGE output

py-rouge

The py-rouge library was used to compute the ROUGE sores. Example output presented in Figure 2.4. In the official Perl-script, a re-sampling strategy is used to compute the average confidence interval and thus, the output may differs by $< 1e4$ which is less the level of accuracy considered in this project.

It calculated the scores for all the metrics specified. These scores the averaged scores for all the candidate and corresponding reference summaries. In this project, all the available metrics were chosen and the maximum n-gram was set to 2 and therefore, there are 4 different scores reported. For each metric the values reported correspond to averaged **'f'** for F_1, **'p'** for Precision and **'r'** for Recall scores from which the F_1-scores are picked.

Interpreting ROUGE -(*) F_1scores

The majority of the values reported in the tables are ROUGE scores primarily because there are many versions of the ROUGE scores (1, 2, L & W) and they can't be combined together like BLEU scores.

The values reported in Tables 4.1 & 4.2 are the F_1-scores. The benefit of using $F_1$is that it places equal importance on how similar the candidate summary is to the reference summary & vice versa. From Eq 2.1, it is clear that the score is proportional to the similarity metric that is used, i.e. number of common n-grams or LCS or WLCS. Still, when the scores are averaged, the F_1-scores become somewhat un-interpretable, which is subsequently the consensus online. Therefore, moving forward, the ROUGE -F_1should be thought to be revealing how similar the candidate summaries are to the reference summaries.

2.4 Project Management

2.4.1 Tools Utilised

The rationale behind the language, libraries, corpora and GPU-provider used in the project is discussed in this section.

Language

The Programming Language adopted for the implementation of this project was chosen to be Python. It has become the standard language used for creating, training and deploying machine learning models. On top of this, it has many APIs for processing data, making it a better choice than another language like Java.

Libraries

The main Python Libraries used in the project are:

PyTorch[21] The Machine Learning Library used to implement the Transformer architecture. It also provided implementation for the Embedding, Linear, Layer Normalisation and Dropout used to create the Transformer.

Sentence-Piece[14] In a pre-processing step, it was used for tokenization. It is an unsupervised and language agnostic text tokenizer and detokenizer. It is used in NLP whenever the vocabulary size is already established. Most importantly, they are vital in building end-to-end-systems such as the translation-then-summarise models being tackled in this project.

OpenNMT-py[12] A generic deep learning framework specialised for sequence-to-sequence models. It provided and deployed the pre-trained summarisation model.

SacreBLEU [23] A standard implementation of BLEU [20] which takes the detokenized output and reports shareable and comparable BLEU scores.

py-rouge[3] A Python API that reports the ROUGE metric from the paper[15].

External Software

The external software employed in development and security for the project are listed here:

Jupyter Notebook It is a web application that aids in creating and sharing documents containing code, visuals, and equations. Colab Notebooks, which are used to run the project, are Jupyter Notebooks hosted on Google's server with access to GPUs.

Git A version control software used to backup code to keep it safe along with versioning as well. It aided portability of code by making the changes available on the Notebooks quickly.

Dataset

For training the translation models, the News-Commentary[1] parallel corpora (with about 200K sentence pairs) and a portion of the ParaCrawl[2] parallel corpora (with about 30M sentence pairs) were used.

For testing the model during the training process, a small dataset comprised of the test sets in past WMT[5]. WMT is a series of annual workshops and conferences on Statistical Machine Translation wherein one of the tasks is News-Translation-Task which come with a sentence-aligned test corpora.

For evaluation, the global-voices datasets[6] (from the paper[19]) were used. It has the hand translated articles in markdown files along with two JSON objects (GV-crowd and GV-snippet). Each contains the summary and a list of languages the article is present in along with their filenames as shown in Figure 2.5. The summaries present in each JSON:

- GV-crowd: contains the handwritten summaries obtained by crowd-sourcing the summaries also referred to as the gold standard in NLP.
- GV-snippet: these summaries were web-scraped from the Global Voices website's description of the articles .

The hand-translated articles, which are generally referred to as the *"gold standard"* in NLP, are called PERF-TRANS in this project. PERF-TRANS means Perfect Translation and these translations were used to calculate the baseline scores.

```
{   "title": [title-string],
    "summary": [summary crowd/snippet],
    "summary_tok": [tokenized version of the summary],
    "other_languages": {
        "de": [de-file-name],
        "es": [es-file-name] }}
```

Figure 2.5: Structure of an entry in GV-crowd/snippet JSON

GPU

Translation models using the Transformer architecture are massive, a size larger than 200MB. Therefore, the models were trained on a Cloud Computing Cluster with access to GPUs like the HPC in Cambridge. Nevertheless, the long wait times caused a switch to operate the training procedure on Google-Colab[7], where one can connect the Colab-notebook to Google's Nvidia K80 GPUs for free but only 12 hours at a time.

[5] http://www.statmt.org/wmt20/

[6] Downloaded after filling the form: https://forms.gle/gpkJDT6RJWHM1Ztz9.

[7] http://colab.research.google.com/

2.4.2 Functional Requirements

Corpus Preprocessing The parallel corpora must be constructed. It is then tokenized with longer sentences dropped and divided into bins ready to form batches. The Translator model consumes these batches during training.

Transformer The model implemented, trained and finally, deployed during the evaluation step. In order to achieve this, the architecture along with its sublayers must be understood to a high degree, especially the self-attention layer.

Training The inherently parallel nature of Transformers requires the development of a training algorithm that can exploit this feature, also to validate the model after each epoch.

Evaluation Use the translator models trained with various settings/configurations to translate the articles in the evaluation dataset (GV-snippet and GV-crowd) and then produce summaries. Finally, calculate the BLEU and ROUGE scores on the translation and summaries, respectively, and performing a Kendall's rank correlation test between the them.

2.4.3 Non-Functional Requirements

Storage The datasets used and the translation models have a massive size[8]. Therefore, any programs developed must make effective use of limited memory available on the Colab Notebooks. In a way, being economical with memory is more important than throughput.

Efficiency The second most important consideration to take into account when building machine learning systems is efficiency or throughput. This pertains to developing programs that suit the computing architecture they run on (for example, GPUs) and minimising any unnecessary computation. Thus, making intelligent use of the time available.

2.4.4 Software Engineering Approach

Like any other project, it is paramount that we adhere to an appropriate development methodology. In all machine learning related projects, it is expected that we have to make changes to the requirements constantly. Hence, I decided to use the Agile methodology wherein I could develop the deliverables but also make improvements wherever and whenever necessary.

In this project, I have tried my best to follow the Software Engineering Process, which starts with understanding the core requirements and investigating their feasibility to figure out the restrictions, followed by selecting an approach and designing a solution. Then the solution is implemented before tests and evaluation is carried out.

The code was developed and tested first on a 64 bit Windows 10 (Home) laptop and then on Google Colab's notebooks. Code was periodically backed up to Google Drive and on Github to protect against corruption or failures.

[8]The corpora consisting of 1-million sentence-pairs was 600MB on average and after tokenization into indices the size was 350MB. Further the model was about 200MB and the adam optimizer was 450MB

Chapter 3

Implementation

This chapter contains the crux of the project along with the choices made during the implementation and the reasons behind it. It is split into 4 sections which elaborate on the preprocessing (3.1.1), transformer architecture (3.2), training process (3.3) and the evaluation step (3.4) respectively.

3.1 Data pre-processing

This section describes the steps followed in processing the datasets to mould them into a compatible form with the Transformer implementation. These are very general and can be applied to any large dataset.

3.1.1 Dataset Filtering

The datasets used for this project are ParaCrawl(v5) and News-Commentary(v11)[1]. The data is in the form of two raw text files, one for each source/target language. One sentence per line and the n^{th} sentence in each file are translations of each other. Because of the computational resources and the time available, I decided to put a set a max of 1 million pairs of sentences in the source(fr,es,de)[2] and target(en) language will be used in the training step. However, ParaCrawl and News-Commentary combined have over 30 million sentence pairs and therefore, too large to handle. Thus, a dataset containing sentence pairs from News-Commentary (200K) and ParaCrawl (top 800K) was constructed. Since the final evaluation was meant to be on the GV-crowd/GV-snippet dataset which is mostly news articles and their translation and summaries, the full News-Commentary dataset was used and to construct a dataset with 1 million sentences another 800k sentence-pairs were taken from the ParaCrawl dataset. Finally, because of the constraint on the internal memory, the program written to build the dataset had to be memory-efficient.

[1] Links to the datasets can be found in Section 2.4.1

[2] fr-French, es-Spanish & de-German

3.1.2 Tokenization

After the dataset is constructed, the sentences need to be converted into tokens which can be passed to the model.

Vocabulary Size

Important considerations that need to be taken when tokenizing with BPE is the vocabulary size. During the execution, it came to light that a larger vocabulary size brings the mean sequence length down because you need fewer word pieces from the large vocabulary to encode a sequence. Shorter sequences mean that the computational complexity will be reduced. There is clear shift towards having more short sequences as vocabulary size increases in Figure 3.1.

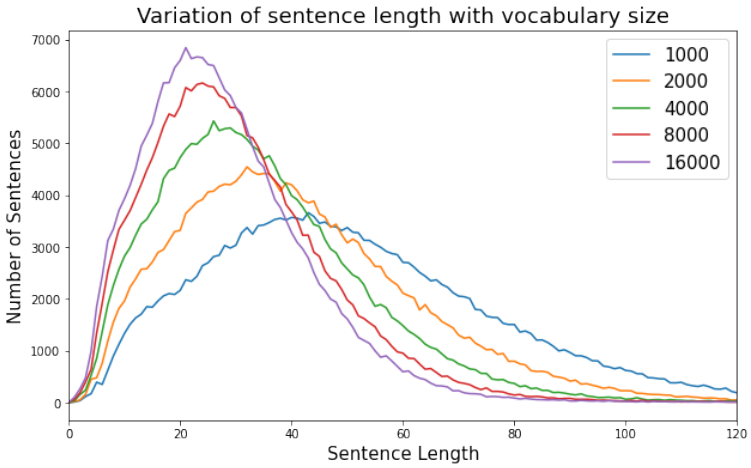

Figure 3.1: The distribution of sequence lengths w.r.t BPE vocabulary sizes(*in the legend*).

Through further investigation, it was concluded that because BPE is a greedy algorithm, when the vocabulary size is large, it will assign one subword to the frequently occurring words. Thus, leading to shorter sequences. Regardless of the computation boost owing to shorter sequences, it undoes most of the benefits described in the subsection 2.2.4. Moreover, with the help of the analysis conducted in the paper[8], the vocabulary size was set to **8000**.

Sentence-Piece

It provides many functions for subword tokenization like the one being used for this project, BPE. A benefit of using Sentence-Piece is that it takes raw files as its input. Consequently, to learn the vocabulary model, the previously generated dataset with a million sentence pairs is used in their raw text file format along with the following parameters:

- `model_type`: to specify 'BPE'
- `model_prefix`: to specify the name of the model. Used later in the pipeline
- `vocab_size`: set to 8000
- `character_coverage`: set to 1.0 to specify complete character coverage.

- `pad_id`: the index reserved for the padding token is set to 3 since the default(-1) causes problems further down the pipeline because the model is expecting a Tensor with indices and -1 is not an index.

The Sentence-Piece model only needs to train once. Whenever it is needed to tokenize or detokenize, it just needs to be loaded into our program by using the `model_prefix`.

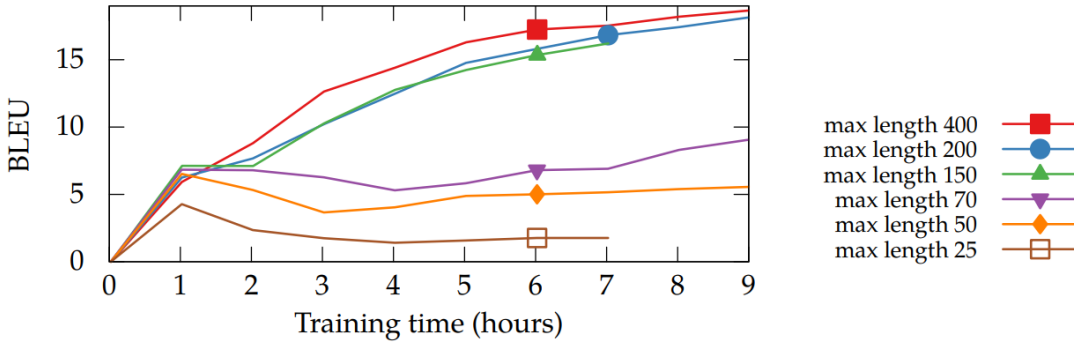

Figure 3.2: The BLEU scores over time for different maximum length thresholds (from the paper[22])

Filtering Long Sentences

Once the vocabulary model has been used to tokenize the sentence into a sequence of IDs (indices in the vocab dictionary), the sequence's size needs to be checked to make sure it is below a threshold. It is impossible to train on very long sentences because the computation graph computed during the backpropagation is so huge(*more than 15GB*) that it leads to an Out-Of-Memory (OOM) Exception.

The threshold for maximum length introduces another hyperparameter that must be set to an appropriate value; else, it can completely wreck the training process. In the article[22], the authors examined and reported the figure 3.2. It depicts that setting max length to a small value like below 70 will adversely affect performance as the BLEU scores stagnate at around 5-10.

We want the maximum-length threshold to be as high as possible to allow for maximum generalisability. That's because training on shorter sequences hampers performance on longer sequences, even if we are using Transformers. This is a result from the *Unit of assessment: Deep Neural Networks*. For this project, the maximum-length was chosen to be 150.

Padding

After tokenization and removing the longer sentences, padding tokens are added to the source and target sequences to ensure they are the same length. Keeping the length of each sequence pair the same helps to calculate the overall tokensize and therefore, aids in building batches. Finally, each pair is added to a Bin based on their length, hence, extra padding tokens may be added to make sure they are the correct length for that bin. Different strategies for constructing bins is covered in the next subsection 3.1.3.

Tokensize	Raw	Effective		
		naïve	big	small
1024	14.4k	1.5k	7.4k	13.4k
2048	14.5k	1.4k	7.1k	13.6k
3072	15k	1.5k	7.4k	14k
4096	**15.5k**	1.4k	7.3k	**14.4k**

Table 3.1: The **Raw** and **Effective** throughput for different tokensizes and batching

3.1.3 Bins

This section details a method adopted to increase efficiency by altering the scheme in which padding tokens are added to the sequences. When we pass a sequence to the Transformer, an equal amount of computation is performed on the actual tokens and the padding tokens.

Consider a naïve approach where all the sequences are padded to be maximum length. It will require a basic batching algorithm that can assign any sequence to a batch. However, considering the above statement, a large portion of the computation will be wasted. Consequently, in a short experiment it was found that in this method only 10% of the overall computation is useful. In other words, to increase the throughput of the Transformer in training, the number of padding tokens in each sequence must be minimised.

Keeping this in mind, the first approach was to use bins of some fixed size like
bins = $[32, 40, 48, 56, 64 \ldots 152]$. This method is referred to as "big". In this method, each pair is assigned to the bin corresponding to b size, provided the source and target sequence lengths are less than or equal to b, where $b \in$ bins. The two sequences then have padding tokens added to them so that their size equals b. This method does better than the naïve approach but only about 47% of the overall computation is used on actual tokens.

Considering how well the first approach faired against the initial naïve approach, the next and final (*"small"*) approach was to use bins of width one. To achieve this, the source/target sequence was padded so that they were the same length and then placed into the bin corresponding to their length. Most importantly, most of the computation involved in this strategy was used on actual tokens (85%). All of the these steps are present to boost efficiency of the translation model.

3.2 Transformer Architecture

After the preprocessing step, the data is ready to be fed into the translator for training. In this section, various parts of the architecture are discussed and how they were implemented in PyTorch. A diagram of the Transformer architecture can be seen in Figure 3.4 towards the end of the section.

For this project, the Transformer(*Base*†) architecture was used. It is smaller and does not perform as well as the Transformer(*Big*†) architecture, but it meets the computational restrictions.

3.2.1 Embedding layers

A batch of sequences is passed to the Transformer in the form of a matrix with the shape [B x S], where B is the number of sequences in the batch and S is the sequence length. The words are one dimensional token IDs and need to be converted to embedded vectors with a size equal to the model dimension (d_{model} = 512 for *Base*). `torch.nn.Embeddings` was used to implement this layer which learns the embeddings for each token. The Embedding Layer is an integral part of the Transformer because the embedded vectors are critical to the attention sublayers, which learn to exploit the embeddings to understand the relationships between words.

3.2.2 Positional Encoding

Each operation in this architecture is either point-wise (like embedding or feed-forward) or treats the sequence as a set (like attention or linear norm). Therefore, the model does not know the relative or absolute position of the token in a sequence. Out of the many ways to overcome this issue, the authors in[29] suggested a point-wise addition of sine and cosine functions with the wavelength in a geometric progression commencing at 2π to $10000 \cdot 2\pi$. These are formulated as follows:

$$PE_{p,2i} = sin(p/10000^{(2i/d_{model})})$$
$$PE_{p,2i+1} = cos(p/10000^{(2i/d_{model})})$$

where p is the position of the token in the sequence and i is the dimension. These Positional Encodings (PE) are added point-wise to the embedding vectors.

During implementation with PyTorch, the matrix with 150 x 512 items was turned into a PyTorch buffer. Doing so allows PyTorch to track and store the PE as parameters. They move between the CPU and GPU like the features, but they are not part of the Stochastic Gradient Descent (SGD).

3.2.3 Multi-Head Attention

The self-attention described in Section 2.2 Background Knowledge is the basic version of Self-Attention. This operation has been enhanced by three keys advances. These are using Query, Key and Value matrices, scaling the dot product and using multiple-heads.

Queries, Keys & Values

Recall, every input-vector x_i is utilised in 3 different ways in the self-attention operation:

Q- Query: it is utilised to calculate the weights for its output vector y_i

†From the paper[29], *Big* has twice the model dimension (d_{model} = 1024), feed-forward dimension (d_{ff} = 4096) and attention heads (h = 16)

K- Key: it is utilised to calculate the weights of all the output vectors y_j

V- Value: it is included in the final weighted sum to get the output vectors after the weights have been ascertained.

In the basic version, we utilise the input vectors for all these three roles of Query, Key and Value. While in this version, learnable parameters are injected in the form of matrices W_q, W_k and W_v to build separate vectors for each role which can help form better output vectors. These matrices are built using `torch.nn.Linear` (which are fully connected neural networks.)

$$q_i = W_q x_i \qquad \text{and} \qquad k_i = W_k x_i \qquad \text{and} \qquad v_i = W_v x_i$$
$$w'_{ij} = q_i^\top k_j \quad \text{then} \quad w_{ij} = \text{softmax}(w'_{ij}) \quad \text{then} \quad y_i = \sum_j w_{ij} v_j$$

Scaled-Dot product

Since there is a softmax operation involved, the large values that appear may completely ruin the results after this operation and cause the learning to come to a complete halt. For an vector in $\mathbb{R}^k$ with all the values equal to a. The Euclidean length is $\sqrt{kc}$, and thus, adding more dimensions increases the length of the average vector. Therefore, the scaling factor is decided to be $1/\sqrt{d_k}$. Where the size of the Query and Key vectors are d_k and the size of the Value vectors is d_v

$$w'_{ij} = \frac{q_i^\top k_j}{\sqrt{d_k}}$$

Multi-head Attention

The current implementation can, over time, learn to pick up on how each word in the sequence is related to another. It is obvious that all of these relationships get summed into one and arguably makes it more complex and, therefore, more troublesome to learn. For example, in the sentence:

`"Ana took chocolates from Bob"`

Here *"took"* is related to “Ana", “Bob" and “chocolates", albeit in entirely different ways: *Ana* is someone who takes; *chocolate* is the something that is taken and *Bob* is the one who gives. This is just one example of the complex relationships that need to be learnt with a single self-attention operation.

A simple solution is to use multiple self-attention operations (instead of just one) and combine the results. These are called **attention heads** and indexed them by $(\cdot)^r$. Consequently, W_q, W_k and W_v turn into W_q^r, W_k^r and W_v^r

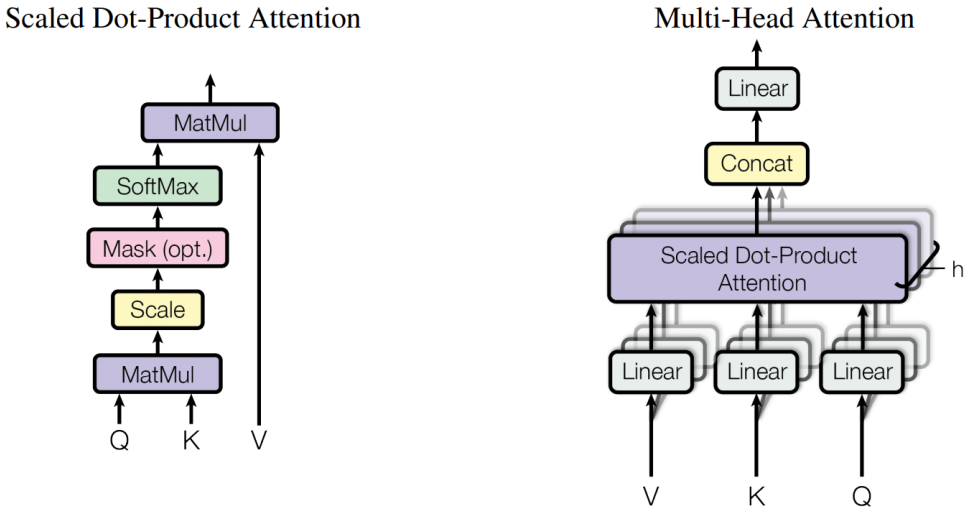

Figure 3.3: Displaying the Scaled Dot Product (left) and Multi-head Attention mechanism (right) with multiple attention layer running in parallel. Taken from the paper[29]

Implementing Multi-Head Attention

There are two ways to implement the Multi-head Attention mechanism. These are the Narrow and Wide approaches. The Narrow Multi-head attention approach was chosen because it is less computationally intensive and moreover, the method that was used in the paper[29] as shown in Figure 3.3. To explain it further, suppose our input vectors have a dimensionality of 512 as before and the number of attention heads we are using is 8. In the Narrow Multi-head attention approach, our matrices W_q^r, W_k^r and W_v^r will have the size 512 x 64. This will transform the input vector into 8 vectors of 64 width and at the end, we will be simply concatenating the output from each head to get back to the size of 512.

Accordingly, the Query, Key and Value matrices were built as `torch.nn.Linear` layers with input and output size set to the same value (d_{model}). After the query, key and value vectors are calculated, they are transformed from a size of [B x S x d_{model}] to [B x S x H x d], where B and S are as defined before; H is the number of heads; and d (=d_{model}/H). Then we follow the self-attention methodology as described before.

3.2.4 Position-Wise Feed Forward Networks

After the attention layers, both the Encoder and decoder contain a two-layer point-wise Feed-Forward network which performs a linear transformation with a ReLU (Rectified Linear[18]) activation in between them and an inner-layer dimensionality of $d_{ff} = 2048$.

A single layer point-wise feed-forward Network is used at the end and applied to the decoder's output. It takes the decoder's output and transforms the vectors from d_{model} to vocabulary size, from 512 to 8000. Lastly, we apply a softmax layer and this gives us the output probabilities for each word at each index.

3.2.5 Regularisation

There are two regularisation methods applied: Residual links and dropout.

Dropout[26] is ignoring some randomly chosen nodes. It is used during training for deep models to ensure that the model does not overfit the training set. It also ensures that the model generalises better to unseen data. The dropout rate was set to 0.1

Residual connections[9] are adding the input of a sublayer to the output from the sub-layer:

$$y = x + \text{SubLayer}(x)$$

First used to build Deep Convolutional Networks for image classification. The reason they work is that unlike traditional models, the data path through residual networks varies in length and allows the model to work as an ensemble of networks. Non-linear activations naturally cause the gradients to explode or vanish for very deep networks. The addition of these 'skip connections' allows the gradient to flow through directly and undercut the non-linear activations' problem.

3.2.6 Layer Normalisation

Between each sublayer (multi-head attention and feed-forward) layer normalisation is carried out. Layer normalisation ensures that there is a smoother gradient, faster training and allows for better generalisation. Layer normalisation is vital; however, when it is carried out is equally relevant, i.e. after (Eq 3.1) or before (Eq 3.2) every sublayer. A significant problem during the training process was that the model developed with the normalise-after methodology would not converge at all. After spending hours experimenting/testing each of the model's components extensively, the solution was found thanks to the explanations in the paper[16]. The authors advised using the normalise-before method because it diminishes the dependence of the output on the residual branches. However, they did criticise the normalise-before method as it did not produce models of the same quality as normalise-after. Considering this trade-off, the normalise-before methodology was selected.

$$y = LN(x + \text{SubLayer}(x)) \tag{3.1}$$

$$y = LN(x) + \text{SubLayer}(LN(x)) \tag{3.2}$$

3.2.7 Encoder and Decoder

After the indices are converted into embeddings and the positional encodings are attached. This input is now handed over to the Encoder & Decoder where the bulk of the computation takes place. Their structure is displayed in the Figure 3.4

Encoder Layer Each Encoder layer comprises two main sublayers, the multi-head attention and the point-wise feed-forward operation with a layer normalisation before each layer and residual connections across each sublayer. The encoder takes as its input the source

sequence/batch and the padding mask[††]. The output is the context for the source sequence that is passed to the Decoder.

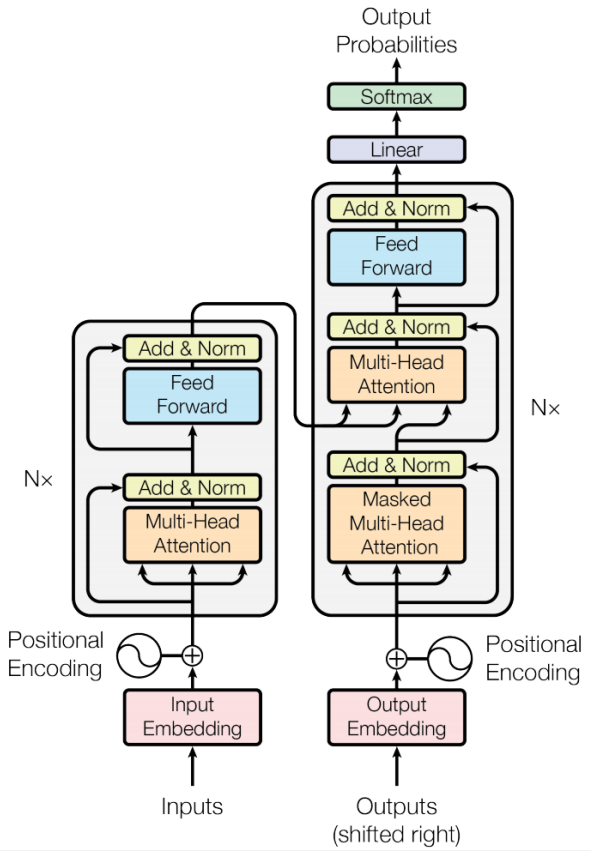

Figure 3.4: The Transformer architecture from the paper[29]

Decoder Layer Each decoder comprises three main sublayers: the multi-head attention layer, point-wise feed-forward layer and an encoder-decoder attention layer. The encoder-decoder attention layer is a multi-head attention layer with the Keys and the Queries set to the Encoder's output. The input to the Decoder is the Encoder output, the target sequence that has been generated (during evaluation) or the target sequence (during training) along with the source padding mask[††] and a combined mask[††] containing the target padding mask[††] and the no-peak-mask[††].

3.3 Training

After covering the architecture of the Translation model, this sections aims to describe the choice of hyperparameters, the training algorithm, the text generation algorithm used and the model validation algorithm carried out periodically.

[††]More information about masks including their purpose, their structure and how they are constructed is covered in 3.3.2

3.3.1 Hyperparameters

Dataset size: A larger dataset size has been shown to improve the translation model's quality. However, as discussed in Section 3.1.1, this was set to one million sentence pairs because of the time and memory restrictions.

As discussed in the subsection 3.1.2:
Vocabulary size was chosen to be 8000.
maximum length was chosen to be 150.

Layer Normalisation was set to before each sub-layer within the encoder/decoder units, as discussed in the subsection 3.2.6.

Batch size for a transformer is often referred to as token size. Token size is defined to be the product of the number of sequences and the length of the sequence. In the first assignment *Deep Neural Networks Unit of Assessment* an experiment on the effect of different batch sizes and learning rates was conducted. It shed light on the fact that having a larger batch size leads to faster convergence, provided that the learning rate is reasonably low. Similar conclusions were reached in the paper[22], which showed that having a larger batch size raises not only the throughput but also improves the BLEU score. Nevertheless, the maximum number of tokens used on Google Colab's GPU was limited because of the memory restrictions and was approx 7500.

After weighing all of these findings, the batch size/token size was set to 4096. A power of two was used, keeping in mind the GPUs' architecture, which tends to have some power of 2 execution units. Hence, when work gets scheduled, we will not have too many execution units sitting idle and thus, improve the throughput.

Optimizer hyperparameters following the implementation from the paper[29], the Adam optimizer[11] maintains a per-parameter learning rate using adaptively estimated moments (1st and 2nd order). It uses $\beta_1 = 0.9$ for the exponential decay rate for the 1st moment and $\beta_2 = 0.98$ for the exponential decay rate for the second moment along with $\epsilon = 10^{-9}$, a small number, to prevent division by zero. The learning rate was first increased linearly for the *warmup_steps* and then decreased proportionally to inverse square-root of the number of steps:

$$lr = d_{model}^{-0.5} \cdot min(\text{step\_num}^{-0.5}, \text{step\_num} \cdot \text{warmup\_steps}^{-1.5})$$

This idea was introduced in[29] and the choice of *warmup_steps* was taken from the paper[22], which concluded that using less than 12k *warmup_steps* can lead to divergent training. Consequently, it was set to **16k**.

Transformer Hyperparameters These hyperparamters were chosen match the *Base*-architecture in the paper[29].
Model Dimensions (d_{model}): **512**
Number of attention heads (h): **8**

Number of encoder-decoder layers (N): **6** each
dropout: **0.1**
Finally, out of the two Narrow and Wide multi-head attention, described in the subsection 3.2.3, the **Narrow** version was implemented. All of these hyperparameters ensure that we do not reach an Out-of-memory exception and that we are maximising the efficiency of the GPUs used.

3.3.2 Training step

Batching

The batching algorithm is implemented as a generator[3] function in Python in order to simplify the code. We begin by shuffling the order of the sentence pairs in each bin. Then chooses a random bin and builds a batch by selecting the required number of sequences from the bin.
Thus, the batch iterator returns two 2d arrays of the shape $[tokensize//b, b]$, one each for the sequence in the source and target languages.
Where the bin size is b and the token size/batch size is $tokensize$, then the required number of sequences is the output of the integer division of the two: $tokensize//b$.

Inputs and Outputs

The 2d-arrays from the batch iterator need to be transformed into a compatible form with the training step. The source array gets converted into a `torch.LongTensor` and is moved onto the GPU.

```
target_tensor_input = target_array[:, :-1]
target_tensor_output = target_array[:, 1:]
```

For the target array, first a beginning-of-sequence character <bos> & a end-of-sequence character <eos> is added at the appropriate position. The above code represents the transformations carried out on the target array. Then for each sequence in the batch, all but the last item is used to build the target tensor. The expected output from the Transformer is set to a `torch.LongTensor` constructed from the target array with every item in each sequence except for the first one.

Consequently, the training algorithm is designed such that, for every position in the input target tensor, we expect to see the following word in the sequence to have the highest probability in the target predictions.
The benefits of the immense parallelizability of Transformers can be seen here. Usually, sequence models recurrent architecture require that the sequence is passed serially for both during the training step. However, with Transformers, the entire sequence moves through the model in parallel.

Masks

Two types of masks need to be employed to ensure that the attention operation works as expected.

[3] Generator Functions allows us to declare a function that behaves like an iterator, i.e. it can be used in a for loop.

Firstly, perhaps the most obvious one, that is the **padding mask**. Pads were added to the source and target sequence to ensure they were the same length. These pads would be considered as part of the sequence if they were removed using masks. To build the padding mask for the sequences, the following code is used.

```
    padding_mask = (sequence != pad)
```

Another mask required is the **no peak mask** that will ensure that the decoder can not peak at the following (future) words when making its predictions. To construct it the following code is used:

Listing 3.1: No Peak Mask generator

```
def nopeak_mask(size):
    np_mask = np.triu(np.ones((1, size, size)), k=1)
    np_mask =  torch.from_numpy(np_mask) == 0
    return np_mask

# output for size=3
# [[[1., 0., 0.],
#  [1., 1., 0.],
#  [1., 1., 1.]]]
```

Here, an upper triangular matrix is constructed and then 1s and 0s get flipped to produce a matrix, as shown above. This nopeak mask is combined with the padding mask discussed before to form the mask for the target tensor.

Mask Application

The mask is applied before the softmax operation. We add a large negative number (-10^9) to the positions we want masked (indices that have a 0 in the mask). Thus, when the exponentiation takes place in the softmax, these values are suppressed to zero.

3.3.3 Text Generation

Once a model has been trained. It will need to generate text. In an ideal world, we would search all possible sequences before returning the sequence with the highest probability. Nevertheless, resource restrictions mean that we have to settle for heuristic-based search algorithms for text generation like Beam Search[28] or Greedy search.

The structure of any text generation algorithm is to generate words in steps, i.e. one 'word' at a time. With Transformers, first, the encoder output is calculated by passing the source sequence through the transformer-encoder as we did in training. The encoder output is used as the context/memory to inform the next generation and is passed to the transformer-decoder. At the start, only the beginning-of-sequence word <bos> is passed to the decoder. As words are generated, all of them are used to inform the next word to be generated. Recognising that the predictions will be a probability distribution on the entire vocabulary at each step, consider two implementations of text generation algorithms attempted in the project.

Greedy Search

To understand how Beam Search works, it helps to look at a Greedy algorithm which is essentially a Beam search with a width of 1. Moreover, it is an easy to execute and quick algorithm that was used initially to evaluate/test the model before moving onto Beam Search. In this algorithm, the most likely word, i.e. the word with the highest value after the final softmax operation, is chosen as the predicted word.

It is trivially straightforward why this algorithm does not do too well. Since the word probabilities might not be accurate initially, the first few words generated may be incorrect. This form of error piles up at each step as we move through the sequence.

Beam Search

Beam Search is a pruned version breadth-first-search. There is some preset search width commonly referred to as k. The most significant difference from the Greedy Search is that we hold the top-k most likely words for each step. Hence, there is only one sequence at the start, and after considering 8000 words (vocabulary size), k words are chosen, which form the k sequences we will consider for the next word. In the next step and all subsequent steps, k sequences are used to generate $8000 \cdot k$ words from which again k words are chosen. The commonly used k is about 5 to 10, and for the evaluation and translation steps, k was set to 10. Finally, for all the finished sentences a length penalty is tacked on to ensure that we select longer sentences.

3.3.4 Model Evaluation

After each epoch the model is tested on a small parallel corpora built with the test sets from WMT. The corpora contain about 2000-3000 sentence pairs. For each sequence, the model produces a translation using the Beam Search/Greedy Search described above in subsection 3.3.3.

Following translation, BLEU scores were calculated (using SacreBLEU) and stored. After the first three epochs, the new BLEU score is compared to the average of the last three, and if there is no substantial improvement in the scores, then the training is stopped. A substantial difference here refers to an increase of > 0.01, i.e. the difference between the new score and the moving average must be > 0.01.

3.4 Pipeline Overview

This section describes all the work involved in the pre-processing, translation, summarisation and post-processing involved in the final evaluation step.

[4] refer to Sub-Section 2.4.2

3.4.1 Pre-Processing

These steps were followed to collect all the relevant data before moving onto translation and summarisation.

1. Load the JSON files (GV-snippet and GV-crowd)[4].
2. Filter both the JSON objects so that they only contain the entries comprising articles in the required source language (de/es/fr). Checking the language also helped to weed out a lot of unnecessary text like links or extra text.
3. Join the JSON objects into one. Here, the auxiliary information in both JSONs was found to be the same and thus, the snippet summary was copied into the current filtered GV-crowd JSON and under a new key.
4. The final step involved reading the articles in both the source and target language, provided as markdown files.

Processing the markdown article

The markdown files had sentences separated by a newline character which reduced some processing required. Moreover, the English articles were quite clean whereas, the articles in French (fr), Spanish (es) and German (de) contained hyperlinks and, most importantly, extra text like
`"Compártelo: Twitter Facebook Reddit google plus"`
Which in Spanish means:
`"Share on Twitter Facebook Reddit google plus"`
These had to be cleaned before being fed the source articles into the translator. French and German had similar problems, but they were not as frequent, and therefore, nothing was done for them.

3.4.2 Translation

The checkpointed models were loaded and then the positional encodings were updated to have a maximum length of 300 to ensure that the model can handle longer sequences during the inference step. Using Beam Search (described above in the subsection 3.3.3), each sentence in the source language is translated into the target language (English). The translated sentences are then collected into one line and written to a file, and ready to be summarised.

Once all the articles were translated, SacreBLEU was used to calculate the BLEU score for the translation and consequently, stored as well.

3.4.3 Summarisation

As described in the 2.2.3, the pre-trained summariser following the Bottom-Up Abstractive architecture is used. The hyperparameters used for the generation are as follows:

minimum length : 35 (*Ensures that the minimum length of the output summary is 35*)

stepwise penalty : A penalty that is applied at every step.

beta (β) : 5 (*Parameter used for the Coverage Penalty*)

coverage penalty : applied (*a penalty used to prevent the repeated attention to the same source word*).

$$cp(X,Y) = \beta \sum_{i=1}^{|X|} log\left(min\left(\sum_{j=1}^{|Y|} p_{i,j}, 1.0\right)\right),$$

where $p_{i,j}$ is the attention probability of the jth target word y_j on the ith source word x_i and $|X|$ is the source length and $|Y|$ is the current target length.

alpha (α) : 0.9 (*Parameter for the Length Penalty*)

length Penalty : "wu"[30] applied (*A penalty added on* `<eos>` *tokens that helps prioritise longer sentences.*)

$$lp(Y) = \frac{(5 + |Y|)^{\alpha}}{(5 + 1)^{\alpha}}$$

Beam size : 10 (*The search width k from subsection 3.3.3*)

3.4.4 Post Processing

The evaluation's final step is to collect the hand-translated sentences and the machine-translated sentences together to calculate the BLEU score. After that the generated summaries are collected together and the mean ROUGE scores are reported in the **Chapter 4 Evaluation** (specifically 4.2 and 4.3) along with inferences we can draw from the results.

3.5 Repository Overview

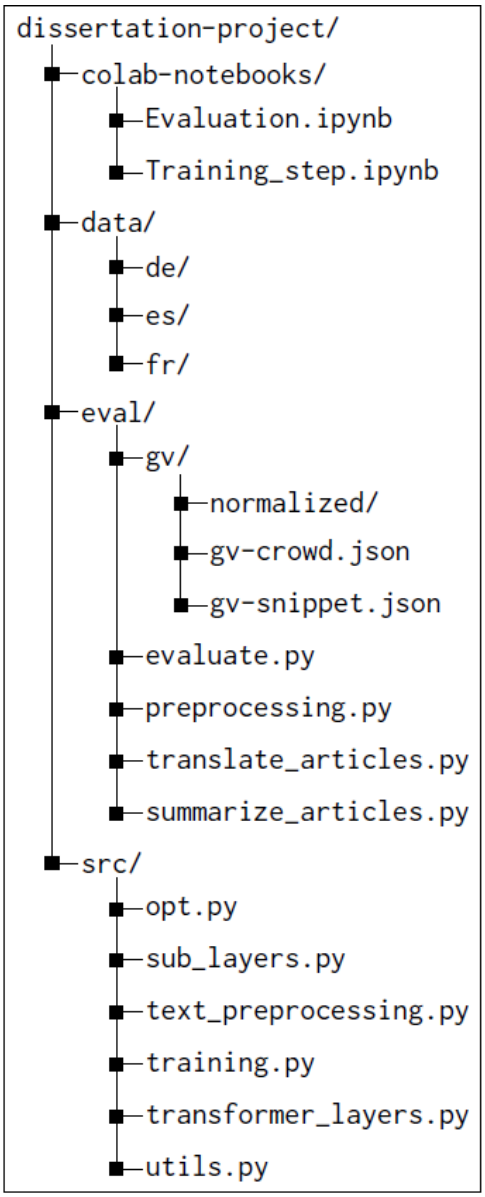

Figure 3.5: Repository Structure

There are four main directories of the repository for this experiment. They are as follows:

colab-notebooks/ This directory contains the two jupyter notebooks[5] that were used on Google-Colab to run the experiments.

data/ This directory contains all the relevant data that is needed to run the programs and all the data generated during the experiments. This directory is further subdivided into directories relevant to each source language (de, es, fr).

eval/ The directory contains the GV-snippet & GV-crowd json files along with the web-scraped articles. The scripts used during the evaluation step are also present in this directory.

src/ This is the most important directory and contains the meat of the project. It contains the preprocessing, training and utility scripts along with the transformer architecture definition. Finally, there is an `opt.py` which contains a singleton class that is used to pass arguments around.

[5]Note: In the final submission these python notebooks were converted to '.py' files in accordance to the guidance on moodle

Chapter 4

Evaluation

4.1 Success Criterion

All of the success criteria set out for this project were met. The criteria for success as listed on the initial project proposal are as follows:

- ✓ Translator has been implemented
  - Transformer architecture implemented (*Covered in Section 3.2*)
  - Preprocessing steps taken before training begins (*Covered in Section 3.1*)
  - Trained models for de, es and fr (*Covered in Section 3.3*)
  - Multiple checkpointed models used to generate translations (*Covered in Section 3.4*)
- ✓ Summariser used to generate summaries (*Covered in Section 3.4*)
- ✓ Comparison between BLEU and ROUGE scores. (*Covered in this Chapter*)

Interpreting BLEU and ROUGE scores

In section 2.3, I have detailed how to interpret BLEU and ROUGE scores. This subsection reiterates the important points to help understand the data in this section.

BLEU is a precision based metric which reports how similar the candidate translation is to the references provided. The main idea is that the closer a candidate translation is to the "*gold-standard*" reference the better the quality, and therefore, the higher the BLEU score. In the following experiments the scores are reported out of 100.

In the case of ROUGE , the F_1-scores are considered meaning that the scores reported express how similar the candidate and reference summaries are to each other. There are different categories of ROUGE namely 1, 2, L & W each of which use a different similarity metric. Like the BLEU scores these are also out of 100 and higher scores imply the summarization quality is better.

4.2 Baseline Results

Table 4.1 reports the Baseline/Upper-bound ROUGE -F_1scores with the candidate summaries in the form of FIRST50 and the summary generated with PERF-TRANS compared against the reference summaries in the form of the '*snippet*' and '*crowd*' summaries from the GV-dataset and for all the languages i.e. de, es and fr.

BASELINE	ROUGE (F_1) GV-snippet				ROUGE (F_1) GV-crowd			
	1	2	L	W	1	2	L	W
				de				
PERF-TRANS	43.4	29.3	42.4	21.1	38.7	16.5	33.3	13.7
FIRST50	73.6	66.9	73.6	44.7	48.3	25.8	41.6	18.8
				es				
PERF-TRANS	44.0	29.6	42.8	21.6	38.1	15.4	32.6	13.2
FIRST50	69.5	62.2	69.6	42.2	47.5	24.1	40.8	18.2
				fr				
PERF-TRANS	42.0	27.2	40.5	20.3	36.8	14.2	31.1	12.4
FIRST50	68.5	61.1	68.8	41.8	47.0	23.4	40.1	17.8

Table 4.1: Baseline Results

The scores achieved by the baseline FIRST50, which is made up of the first fifty words in the article's English version, performs much better when compared to any of the summaries generated. FIRST50 achieves a ROUGE -2 F_1score of 0.6, which is very high and therefore reveals, that the two summaries must be very similar. This trend is observed in all the languages. As explained before, the snippet summary is the description of the article that was web-scraped. Hence, we can claim that snippet summaries are most likely excerpts of text taken from the start of the article. Therefore, I decided to ignore the snippet summaries for the next step.

The ROUGE -F_1-scores for candidate PERF-TRANS summary and reference: GV-crowd can be regarded as the maximum score or upper-bound that can be reached should the translation be perfect. None of the candidate summaries in the Translate-then-summarise category come close. The best scores are at around two-thirds of this value. Considering that the best translation models have a BLEU score of 26(es), 18.8(fr) and 16(de), this is the first evidence towards the hypothesis that the translation quality does affect the summarisation quality.

4.3 Translate-then-Summarise Results

This section aims to discuss the results of the experiment and the trends that appear.
Table 4.2 reports the different ROUGE F_1scores and the BLEU scores for all the languages. These values are indexed by the Model Number which is the number of batches that have been digested by the model during the training process and within the parenthesis is the ratio of the model

number with the epoch size. *Epoch Size*: is the number of training steps involved in one epoch. Figure 4.1 displays the graph with BLEU scores in the x-axis and the different ROUGE -(1, 2, L & W)-scores plotted with blue, orange, green and red respectively.

de, epoch size: 20.6k		
Model	BLEU	*reference* GV-crowd ROUGE -(1|2|L|W)-f1
5,000 (0.24)	3.32	23.7 | 2.9 | 21.4 | 7.4
10,000 (0.48)	12.03	30.5 | 6.9 | 26.1 | 9.5
20,000 (0.97)	14.69	31.0 | 7.2 | 26.2 | 9.7
40,000 (1.94)	15.66	31.8 | 7.6 | 26.3 | 9.8
60,000 (2.91)	15.79	31.8 | 7.8 | 26.2 | 9.7
80,000 (3.87)	16.14	31.4 | 7.7 | 26.5 | 9.9
es, epoch size: 17.7k		
Model	BLEU	*reference* GV-crowd ROUGE -(1|2|L|W)-f1
5,000 (0.28)	4.11	24.1 | 3.5 | 22.4 | 7.8
10,000 (0.56)	11.44	30.1 | 6.2 | 25.8 | 9.4
20,000 (1.13)	20.52	32.4 | 8.0 | 27.7 | 10.3
60,000 (3.39)	24.87	32.8 | 8.2 | 27.5 | 10.3
80,000 (4.52)	25.02	32.9 | 8.2 | 27.7 | 10.4
110,000 (6.21)	26.05	33.6 | 8.8 | 28.0 | 10.6
fr, epoch size: 9.1k		
Model	BLEU	*reference* GV-crowd ROUGE -(1|2|L|W)-f1
6,000 (0.66)	3.09	23.4 | 3.2 | 21.3 | 7.4
12,000 (1.32)	2.41	14.0 | 1.1 | 14.0 | 4.7
18,000 (1.98)	12.29	30.0 | 5.9 | 26.0 | 9.4
33,000 (3.63)	17.36	32.4 | 8.1 | 27.5 | 10.2
48,000 (5.27)	18.85	33.2 | 8.6 | 28.0 | 10.4
72,000 (7.91)	18.19	32.6 | 7.6 | 27.1 | 10.1

Table 4.2: Translate-then-Summarise results

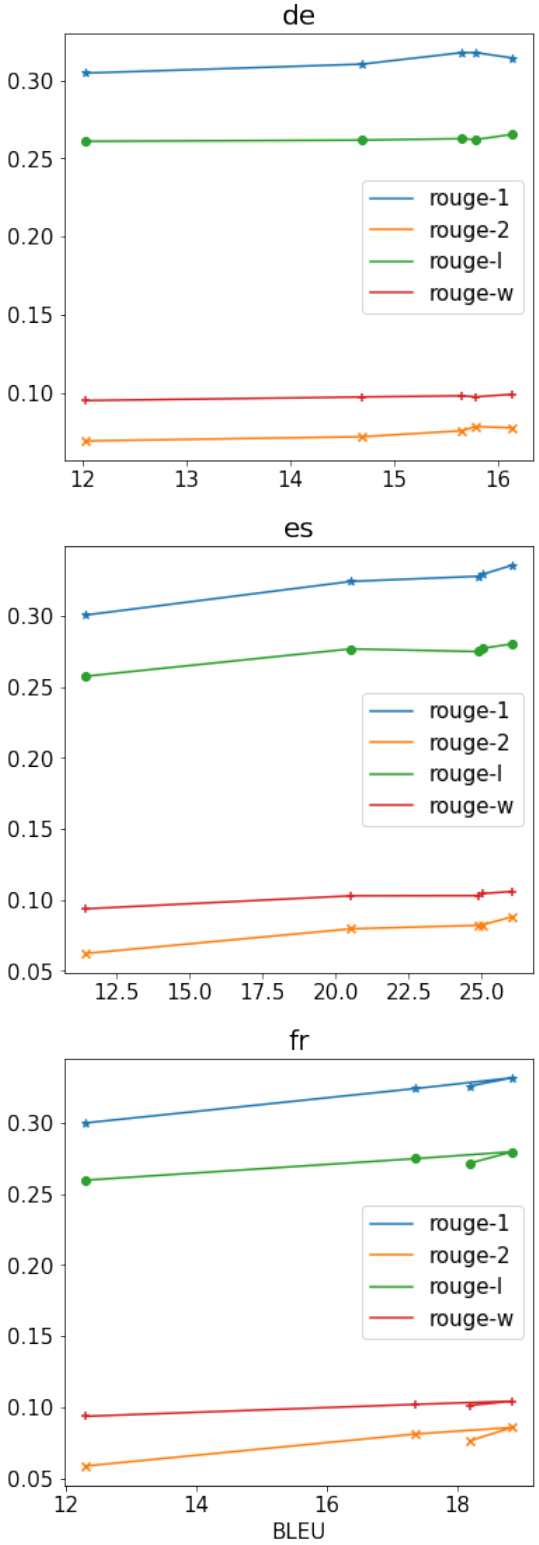

Figure 4.1: Variations of the different ROUGE scores w.r.t. the BLEU scores for all languages

Models chosen

For all three languages, I have tried to choose 3 models from the first/second epoch and they are expected to produce translation of poor quality because they have not been trained for long enough. The rest of the models are chosen from remaining checkpoints and the final model in each case is the last model to be checkpointed before the training was stopped.

4.3.1 BLEU Scores

Out of the models trained, Spanish (es) appears to perform the best, having the maximum BLEU score of 26.05, followed by French (fr) at 18.85 and then German (de) at 16.14. Spanish does the best, however, it took the most time to reach convergence as well. Because it took the longest, it reflects that Spanish is the most different to English out of the three languages chosen. Further, I speculate that the models could have performed better if they were trained longer rather than training till convergence.

French is also very close to English. However, when the parallel corpora were constructed for French, it appears to have ended up with shorter sentences. French was the first languages I trained and I discovered that the French dataset had too many short sentences and an average length of 37 compared to more than 70 for German and Spanish. Thus, as described in subsection 3.1.2, the model needed to see longer sequences to perform well. Sadly, this was discovered when it was too late to make a change. Nevertheless, it took a similar number of training steps to reach convergence.

Finally, German performs worst and that is the general trend seen in most papers where German is lagging behind other languages like Spanish by 10%. This can be attributed to the many differences in the grammar between English and German. Moreover, there are tons of long compound words in German, which would be hard to tokenize in a meaningful way. These long compound words are most likely the root cause of German having very long sentences.

4.3.2 ROUGE scores

This subsection aims to discuss the results in the context of the problem statement. To test for a linear correlation between the different ROUGE and BLEU scores, I chose to use the Kendall's Rank Correlation.

Kendall's τ Rank Test

Kendall's rank test is a distribution free test that measures the strength of dependence between two variables. The two other choices were Pearson's correlation & Spearman's rank correlation and these were not used because Pearson's assumes normal distribution and Spearman's can only be used to test the null hypothesis of independence of two variables. Kendall's Tau (τ) is defined as follows:

$$\tau = \frac{n_c - n_d}{n(n-1)/2}$$

where n_c is the number of concordant pairs and n_d is the number of discordant pairs and n is the number of pairs of points. Any pair (x_1, y_1) and (x_2, y_2) is a Concordant pair when $(x_1 < x_2 \wedge y_1 < y_2) \vee (x_2 < x_1 \wedge y_2 < y_1)$

	ROUGE -1	ROUGE -2	ROUGE -l	ROUGE -w
de	0.733	0.867	0.867	0.867
es	1.000	1.000	0.867	1.000
fr	1.000	0.867	0.867	0.867

Table 4.3: Kendall's τ between the BLEU and ROUGE scores from Table 4.2

ROUGE and BLEU scores are correlated

The first trend to note is that the ROUGE scores increase with increase in BLEU scores. This means that there is a correlation between the translation quality and the summarisation quality. The same can be inferred from the values in Table 4.2, the Kendall's τ in Table 4.3 and the ratios in Table 4.4 . As per the intuitive hypothesis in Section 1.2 Project Focus, in a two-step process, errors in the first step should be carried forward and amplified in the second step. Therefore, as we had anticipated a correlation between summarisation quality the translation quality for all the different languages is present.

		ΔBLEU	ΔROUGE /ΔBLEU
de	80k vs. 10k	4.11	0.22 | 0.19 | 0.10 | 0.10
es	110k vs. 10k	14.61	0.24 | 0.18 | 0.15 | 0.08
fr	72k vs. 18k	5.90	0.44 | 0.29 | 0.19 | 0.12

Table 4.4: ΔROUGE /ΔBLEU for (1 | 2 | L | W)

Readability of summaries

The final item to discuss is the effect of readability on ROUGE scores. Consider the ROUGE scores and the average gradients seen for the reference GV-crowd summaries in Figure 4.1 & Table 4.4. The lines in the graphs appear nearly flat and the ratios in the table are small respectively.

These values are the most important in this experiment because they are meant to convey the similarity between the generated summary and the *"gold standard"*, i.e. hand-written summary. The maximum possible ROUGE score that could be reached in this experiment is obtained from the Perfect-Translation summaries against the GV-crowd (*from Table 4.1*). Because of the inferior Translation quality, which does not even get close to the industry standard, it was expected that best ROUGE scores would be well below the Baseline. However, that is not the case and in fact the best Translate-then-Summarise summaries are within 5% of the scores of PERF-TRANS summaries. Moreover, the low average gradients point to very little correlation between translation quality and summarisation quality.

The ratio of ΔROUGE to the max ROUGE reveals the same, take the 10k & 110k model for Spanish, they have a gap of 14.61 in BLEU but only (3.5 | 2.6 | 2.2 | 1.1) in ROUGE -(1|2|L|W) respectively. Therefore, while the Translation quality has improved greatly, the Summarisation quality has not improved significantly. Further, normalising the ΔROUGE by dividing by the maximum returns (3.5/33.6 | 2.6/8.8 | 2.2/28 | 1.1/10.1) which is equal to

(0.104 | 0.3 | 0.079 | 0.1). Here the BLEU score more than doubled while there was only a 10% increase in ROUGE -(1, L and W) and a 30% increase in ROUGE -2.

Here are two examples of Translate-then-summarise with source language Spanish and the summary corresponding to 110k model appearing first and 10k model appearing second.
2015-07-07-african-startups-win-fintech-for-agriculture-2015-competition

```
``The end of agriculture 2015. The programme supports entrepreneurs making
   financial services more affordable and reachable for small farmers who
   do not own banking account in eastern Africa. It is supported by the
   foundation''
```

```
``The first winners of its invitation program, east to Africa: end of
   agriculture 2015. the program supports the business sector that the
   financial services are more affordable and reach for the small
   agriculture.''
```

2015-10-15-sao-paulo-will-host-the-2016-world-social-forum-on-migrations

```
``Sao Paulo, the largest city of South America, whose history is intimately
    linked to migration, . The event is being planned for the middle of
   2016. The motto of the new edition of the fsmm will be: ``Migrants'':
   constructing alternatives facing disorder .''
```

```
``Sao Paulo, the largest town of the surgery of the south, whose story is
   an intimately attempt to be headquarters of the social forth edition.
   The event was announced by its organizer, the international committee of
    fsmm. This.''
```

We can confidently claim that as the BLEU scores went up, the readability of the translated document also went up and subsequently, the readability of the summary should have risen as well. The same can be seen in the examples above. Nevertheless, the ROUGE -(1, L & W) scores were not affected by much, which means that these metrics are insensitive to the readability of the summary.

Finally, these results corroborate the findings from the paper[19] where the BLEU drops by almost 90% and the ROUGE scores only drop by 30%. They end by stating that summarization model can achieve a high ROUGE score by just copying phrases even if the source document has meaningless and ungrammatical content.

Chapter 5

Conclusion

Achievements

In this project, I have successfully implemented a variation of the experiments carried out in [19]. In order to reach this goal, I developed three Translation models from scratch along with an evaluation pipeline that uses the translation model and a summarisation model to investigate whether the translation quality and the summarisation quality are correlated.

The most influential trends noted in the Evaluation chapter were that the Translation quality and Summarisation quality are correlated. However, the correlation is weak, which reveals that the ROUGE metric is agnostic towards the readability/fluency of the summaries. Moreover, it can be hypothesised that ROUGE will give a high score to ungrammatical and meaningless summaries simply because they copied phrases from the original document.

After finding out that the BLEU scores are not fully comparable across language-pairs, I chose to compare translation models that were checkpointed during the training process. Thus, I went a step further than the authors of [19] and achieved the aim of reducing as many of the confounding variables so that I could be more confident about the claims. To do so, the models were trained and, most importantly, evaluated on the same datasets. Moreover, no comparisons are made outside of a language-pair. However, the same trends were observed in all three language-pairs, strengthening the validity of these trends. Therefore, I can confidently corroborate the findings from the paper [19].

Personal Reflections and Lessons Learnt

In this project, I have amassed a lot of knowledge in the fields of Machine Translation and Automatic Summarisation. Particularly about Transformers and the innermost workings of the attention mechanisms. Furthermore, I have also been exposed to other uses of Transformers/Attention in NLP such as BERT [5] and also in Computer Vision with a paper [6] about image classification using Transformers.

Since this was the biggest project that I have ever undertaken, I was pushed outside of my comfort zone a few times, I met a new set of challenges and I feel empowered by the fact that I carried such a substantial software project to its completion. Further, I have learnt many lessons and skills that will help me excel in the future. One such skill is the ability to quickly read through and understand the contents of an academic paper. Reflecting on my time-management, I can accurately estimate the time it will take to complete a task.

Future Work

The completion of this project has created multiple possible extensions for future work. One of the basic ones is incorporating more languages (especially non-European) languages and testing to see if the correlation between translation and summarisation quality can be reproduced. Such an outcome would mean that building better translation models is essential in order to deploy Cross-Lingual Summarisation (CLS) at scale. Moreover, it would be insightful to try to build and then experiment on a dataset that is not just geared towards English readers but to other languages as well (i.e. the target language should be another language like German with crowd-sourced summaries in German)

Another extension could be using other summarisation systems. New summarisation models are invented all the time and it is possible that a future state-of-the-art summariser would prove to be a better fit for the CLS system.

One of the more exciting extensions would be to build a CLS system that can run on smartphones. Nowadays, most phones have a GPU within them that can be tapped into to run the system. This is better known as '*on-device learning*', thus introducing CLS to the medium millions use to consume information with the power of CLS.

Bibliography

[1] News-commentary parallel corpra: `https://opus.nlpl.eu/News-Commentary-v11.php`.

[2] Paracrawl parallel corpra: `https://opus.nlpl.eu/ParaCrawl.php`.

[3] py-rouge: A full python implementation of the rouge metric, producing same results as in the official perl implementation. (`https://pypi.org/project/py-rouge/`).

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

[7] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[8] Thamme Gowda and Jonathan May. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online, November 2020. Association for Computational Linguistics.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[10] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA, 2009.

[11] Diederik P. Kingma and Jimmy Ba. Adam: 'a' method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, 'ICLR' 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[12] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[13] Philipp Koehn and Rebecca Knowles. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver, August 2017. Association for Computational Linguistics.

[14] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[15] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[16] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5747–5763, Online, November 2020. Association for Computational Linguistics.

[17] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[18] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[19] Khanh Nguyen and Hal Daumé III. Global Voices: Crossing borders in automatic news summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 90–97, Hong Kong, China, November 2019. Association for Computational Linguistics.

[20] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[22] Martin Popel and Ondrej Bojar. Training tips for the transformer model. *CoRR*, abs/1804.00247, 2018.

[23] Matt Post. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Belgium, Brussels, October 2018. Association for Computational Linguistics.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[25] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 15(1):1929–1958, January 2014.

[27] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[28] Christoph Tillmann and Hermann Ney. Word reordering and a dynamic programming beam search algorithm for statistical machine translation. *Computational Linguistics*, 29:97–133, 03 2003.

[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[30] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Appendix A

Examples

2015-07-07-african-startups-win-fintech-for-agriculture-2015-competition

GV-crowd summary

```
African startups have come out with the winners of a contest arranged by
   Village Capital. The East Africa : FinTech for Agriculture 2015 contest
   supports business owners in making their services more affordable, as
   well as serving farmers who are underbanked.
```

GV-snippet summary

```
Two African startups have emerged winners of a regional competition
   organised by Village Capital: Village Capital today announced the first
   winners of its innovative program, East Africa: FinTech for Agriculture
   2015. The program supports entrepreneurs in making financial services
   more affordable and accessible for smallholder farmers and other
   underbanked individuals...
```

PERF-TRANS Summary generated

```
 first winners of its innovative program, East FinTech program supports
    entrepreneurs in making financial services more affordable and
    accessible for smallholder farmers and other underbanked individuals in
     East Africa. It .   the entrepreneur-participants The FinTech program
    is supported by the DOEN Foundation, The MasterCard Foundation, and
    Duncan Goldie-Scot. of Sub-Saharan Africans .
```

model 110k summary

```
 the end of agriculture 2015. The programme supports entrepreneurs making
    financial services more affordable and reachable for small farmers who
    do not own banking account in eastern Africa. It is supported by the
    foundation .
```

model 10k summary

```
the first winners of its invitation program, east to Africa: end of
    agriculture 2015. the program supports the business sector that the
    financial services are more affordable and reach for the small
    agriculture .
```

2015-10-15-sao-paulo-will-host-the-2016-world-social-forum-on-migrations

GV-crowd summary

```
For the second time Brazil will host the World Social Forum on Migrations.
   They hope to discuses how migrant swill be the key to success as they
   take over more social movements and take on more leadership roles. The
   WSFM has held large scale and smaller scale forums worldwide.
```

GV-snippet summary

```
 The   forum represents a recognition of the struggle over many years of
   social movements in the city, above all, that of immigrants, who are
   increasingly taking on more leadership roles"
```

PERF-TRANS Summary generated

```
the largest city in South America, will mark the second time the Forum on
   Migrations makes its home in Brazil. Previously, Brazil .   the event
   is being planned for mid-2016. The slogan for the newest edition of the
    WSFM will be:   Migrants  : building alternatives to disorder .
```

model 110k summary

```
S o Paulo, the largest city of South America, whose history is intimately
    linked to migration, .   the event is being planned for the middle of
   2016. The motto of the new edition of the fsmm will be:   Migrants  :
   constructing alternatives facing disorder .
```

model 10k summary

```
S o Paulo, the largest town of the surgery of the south, whose story is
   an intimately attempt to be headquarters of the social forth edition .
     the event was announced by its organizer, the international committee
    of fsmm. This .
```

Project Proposal

Computer Science Tripos Part II Project Proposal

Investigating the effect of
Translation Quality on Summarisation Quality

2333C

16th October 2020

Introduction

Cross-Lingual Summariation (CLS) aims to produce a summary in a target language from a document in a different source language. This allows people who are fluent in the target language to get the salient details in an article written in foreign languages. In the Cross-Lingual Summarization there are two main steps: 1) translate and then 2) summarize, which introduces the problem of propagation of errors from the translation step to the summarization step.

The motivation behind such a system is that there are many news articles being published around the world which are not accessible to everyone because of the language barrier. Machine Translation helps alleviate this problem, by using statistical methods and neural networks to produce high quality translation without any human intervention. However, the amount of textual material i.e.news articles and blog posts is immense and it continues to grow everyday. With the sheer amount of material to cover people have to resort to searching for keywords and skimming through the results. Automatic Text Summarization allows us to extract the salient details in the text and then join them together to form one cohesive summary.

This project aims to investigate the effect of bad translation on the summary that is produced. Since CLS is a two step process, any error in the first step should be propagated to the next. Moreover, we don't have sufficient resource to train a state-of-the-art Machine Translation system for every language . This means that the translation produced by the low-resource languages will have some inaccuracies in them.

Resource Declaration

I plan to use my own computer (2 GHz-4 Cores; 8 GB RAM; 128 GB SSD & 1TB HDD; Windows10-Home). I plan on maintaining a backup folder on GitHub and External HDD daily

while also using Google Backup and Sync to continuously backup my device to protect against failure. I am not using any paid service and all datasets used are readily available and free to use therefore, in case of failure, I would like to move to an MCS machine where I would be able to commence my work with only a slight delay.
Special Resource required To train the machine learning models access to HPC is requested.
For development: PyTorch, Moses, **TBD**
The datasets used will be:

- Summarization: CNN/Daily Mail dataset (CNN/Daily Mail-link)
- Evaluation: gv-summaries dataset created by the authors. (Gotten after filling the google form available on their paper: Form-link)
- Translations: News-Commentary dataset(NCv11-link) and the ParaCrawl dataset (ParaCrawl-link) for langauges French, Spanish and German.

Starting Point

This project will re-do the experiments done by *Khanh Nguyen and Hal Daumé III in their paper "Global Voices: Crossing Borders in Automatic News Summarization"* with a few alterations and will be starting from scratch for the translators. I have not written any code related to this project.
In terms of relavent course work, I have done the Scientific Computing (from Part 1A) and Artificial Intelligence (from Part 1B) which served as a base for a further MIT Deep Neural Network course I took which had 1 lecture on NLP/Sequence Modelling.
I will be using a pre-trained Summarization tool which need to be specially trained for this task. I have located the dataset to be used but have only downloaded the gv-summaries dataset.

Substance and Structure of the Project

Baselines

- FIRST50: copies the first 50 words of the English version of the Source Article.
- PEFECT-TRANS: using the perfect translations from the gv dataset to produce summaries.
- TRANS-THEN-SUM: Translating using the models trained and then summarizing.

Core

In the paper from which this idea has been borrowed they use languages of different resource availability (i.e. if there are a large volume of parallel aligned sentences then we have a high-resource language pair and because of the large amounts of data our model will perform better and vice versa) to investigate whether the quality of translation affect the summaries produced. I am extending it further by saving the features of the translator model while it is in training so as to have a few worse translation models for the same language that output inferior translations. This differs from what they did in the paper because they used languages of different resource

availability.
Therefore, I have chosen French, German and Spanish as the source languages and the target language is English. All of the translation models will be sentence-level models.
Therefore the Core of the project can be broken down into the following steps which need to follow this order to save time and for smooth functioning:

1. Research about transformers/summarizers
2. Build and test the Transformer
3. Normalize the datasets before training
4. Build/train the Summarization tool.
5. Evaluate the Core part of the project.
6. Evaluate feasibility of extensions and start working on them.
7. Write dissertation.

However, we want the Summarization tool to be good so that it doesn't introduce any errors. Therefore, initially I am going to use a pre-trained Summarization tool which will be further trained for this task. This is also a way to manage risk since the translation models could take a long time to train.

Extensions

Once the core part of the project is complete, in order to build a better understanding of the effect of translation on summarization I think these would be some good extensions:

1. Train translators for more languages and perform qualitative evaluation on them as well
2. Learn about, build, test and train more Summarization tools

For the new language part, they will be chosen from the ParaCrawl/News-Commentary dataset.

Evaluation

I will be using two metrics for this task: BLEU and ROUGE. In general BLEU measures precision i.e. how many of the words in the machine generated text are present in the reference text while ROUGE measures recall i.e. how many of the words in the reference text are present in the generated text. Usually, BLEU is used for assessing translation quality while ROUGE is used for summarization quality.

BLEU

To evaluate the translation produced the BLEU (Bilingual Evaluation Understudy) is used. The better the machine translation's output, the closer it will be to the human translation of the same

piece of text, and this forms the central idea for BLEU. Scores are calculated for segments - usually sentences - by comparing them to a set of good translations. These scores are then averages for the corpus or the document to get the estimate for the quality. Note: readability and correctness are not taken into account. The scores produced are between 0 and 1 (inclusive)

ROUGE

ROUGE or Recall-Oriented Understudy for Gisting Evaluation is a set of metrics for evaluating automatic summarization tools. They compare the summaries produced to a set of reference summaries available.

- **ROUGE-N:** overlap of N-grams in the output and reference summaries.
- **ROUGE-L:** Longest common subsequence(LCS) based statistics.
- **ROUGE-W:** Weighted LCS-based statistic that will favor consecutive LCSes.
- **ROUGE-S:** Skip-bigram based co-occurence statistic.
- **ROUGE-SU:** Skip-bigram and unigram-based statistic.

F Measure

This is the final metric that will used to compare the summaries against each other. Using the BLEU metric we will have a decent idea about the quality of the translations. The reference summaries shall be gv-crowd (crowd sourced summaries) and gv-snippet (a catchy summary of the article meant to draw people in) using the F Measure metric. At the end we will have evaluated the gv summaries against the ones produced by the baselines mentioned above.

Success Criterion

As described above the translation quality will be evaluated using the BLEU metric. The summaries produced will be evaluated on their own using the ROUGE-* metric. Finally, when comparing two summaries i.e. the summaries produced to the reference summary we will use F-Measure where the Precision will be the BLEU metric and Recall will be different ROUGE-* metrics.

Core

- Translators have been implemented
- Summarizer has been implemented
- A qualitative comparison of the performance of the different summaries produced and the baselines.

Extensions

The following criteria define the success in the extension tasks:

- Translator for new language is implemented/trained and quantitative evaluation has been conducted.
- New summarization models have been implemented and further quantitative evaluation has been conducted.

Concepts

There are two main types of models that will be developed - Translation model using Transformer and Summarization model.

Feed-Forward Neural Network

The most basic form of an Artificial Neural Network which is meant to mimic the neural connections in the brain and can be used to compute any function. The most basic part of a neural net is the perceptron. It takes in a number of Real ($\mathbb{R}$) values and performs a weighted sum to an already existing bias value. The sum is then passed through an activation functions usually sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$) or tanh. Activation functions should be continuous and differentiable, and it is chosen to suite the problem. The output from the activation function is the output for the perceptron.

Multiple percetrons with the same input size form a layer and multiple layers from a Neural Network. We generally have weights matrices which store the weights for the weighted sum operation for each perceptron.
Therefore, the trainable features of a Neural Network are the weights matrices and the biases for each perceptron which can be optimized to then simulate any function.

Back-propagation

The widely used algorithm to train Neural Networks is back-propagation algorithm. There are many optimizations that exists for this process but this covers the basic idea behind training a Neural Network.

Using the loss function we will have evaluated our error for the network. We then proceed to compute the partial derivative of this error for each weight and bias in the network. And for each batch of the training we aggregate and scale these values to get the value to be added to the features (weights and biases) to make the network better.
We continue this process until we see there is no improvement in the error value.

Self-Attention

Self attention is a sequence-to-sequence operation i.e. a sequence of vectors of length T go in and we get a sequence of vectors of length T as well. Suppose our input vectors are $I_1, I_2, ...I_T$ and the self-attention outputs are: $O_1, O_2, ...O_T$, then

$$O_i = \sum_j w_{ij} * I_j$$

where $w'_{ij} = I_i^T * I_j$ and we get w_{ij} by using softmax.

Queries, Keys and Values

Each input vector I_i is used in 3 different ways:

1. Query: Compared to every other weight to form the weights for O_i
2. Key: Used to form the weights for every other vector O_j
3. Value: Finally used in the weighted sum for the final output vectors.

Here we add 3 more K x K weight matrices (where K is the input/output vector dimension) W_q, W_k, W_v to compute:

$$q_i = W_q x_i \text{ and } k_i = W_k x_i \text{ and } v_i = W_v x_i$$

and then

$$w'_{ij} = q_i^T * k_i$$
$$w_{ij} = softmax(w'_{ij})$$
$$O_i = \sum_j w_{ij} * I_j$$

Now that we have added a step in between by introducing the attention heads we can add more attention head that can potentially be trained to have greater discrimination power. This is called Multi-head Attention.

Transformer Architecture - Encoder

The first layer in a Transformer Architecture is the self-attention layer however, once we pass through the self-attention layer we loose all spacial information about where each word was located. In order to counter this problem a vector containing multiple sine/cosine waves at different frequencies are used. This allows the transformer to reason about the position of each word.
Following the Self-attention layer there is a Feed-Forward layer that generates the encoded vectors. Once all the tokens have been passed through the transformer we start Decoding. Note: There may be more encoding steps which will take the current output and repeat.

Transformer Architecture - Decoder

The output from the top encoder is used to form the form the encoder-decoder attention layer's W_k and W_v matrix and it takes the Queries matrix from the encoder's self-attention layer.
Input to the Decoder is a special symbol at the start and after that is the previous output, such that it can see what has been predicted before. The first layer is again a self-attention layer. The output from this layer is normalized and then it moves into the Encoder-Decoder Attention layer. Output from there is directed to a Feed-Forward layer. After which all the data is passed to a Linear layer. Finally, a softmax function provides the probability of next word from our corpus.

Automatic Summarization

There are two way to go about summarizing a piece of text: 1) Extraction, or 2) Abstraction. Extractive summaries employ statistical methods to identify important or the salient sections of text and copying them. Abstractive Summaries aim to interpret the material before generating a summary which are shorter and convey significantly more information. Extractive summaries produce better result because they don't need to worry about grammatical correctness since they are afterall copying the text, however, they do fall into the trap of repeating themselves or not picking up on some important points. Abstractive summaries do a better job of picking the important points, however, lack in readability, and most abstractive summaries have to have some form of extractive step in them as well.

Work Plan and Timetable

The preliminary plan is split into 2/3 week components starting after the project proposal deadline. Please note that these are just estimates and the Plan will need to be changed if certain components are failing. However, in the interest of saving time, I am undertaking the implementation of the Transformer for the translator before everything else (like examining datasets) since the training can happen parallel to the dataset examination step. Similarly, the work on the Summarization tool is the third item on the overall plan.

Timetable

1. **24th Oct - 7th Nov** *2 weeks* -
   Research on, and understand Transformers and associated concepts along with Summarization models/methods. Gain familiarity with PyTorch

2. **8th Nov - 28th Nov** *3 weeks* -
   Begin building the Transformer, and then test the implementation.

3. **29th Nov - 19th Dec** *3 weeks* -
   Normalize the dataset if necessary.
   Continue working on the Transformer.
   Begin work on the Summarizer.

4. **20th Dec - 9th Jan** *3 weeks* -
   Aim to finish the programming work related the main Transformer architecture during the

vacation.
Should also re-evaluate the feasibility of the extensions.

5. **10th Jan - 23rd Jan** *2 weeks* -
   Begin training of the Transformer.
   Continue working on the summarization tool.

6. **24th Jan - 6th Feb** *2 weeks* -
   Start building a pipeline using the translator and summarization tool for evaluation and perform some preliminary qualitative assessment.
   If possible start the dissertation.

7. **7th Feb - 20th Feb** *2 weeks* -
   Make a drat/plan for the dissertation.
   Start working on the extensions.
   Extensions will include finding relevant datasets for training the Translators and/or building more Summarization tools.

8. **21st Feb - 6th Mar** *2 weeks* -
   start work on introduction and preparation chapters
   Continue working on the extensions.

9. **7th Mar - 20th Mar** *2 weeks* -
   Start work on Implementation chapter.
   Send first two chapters for review to Supervisor(Andrew Caines) and DoS (Simon Moore)

10. **21st Mar - 3rd April** *2 weeks* -
    Finish working on extensions.
    Start working on evaluation and conclusion chapters. Send in Implementation chapter for review.

11. **4th April - 17th April** *2 weeks* -
    Send in last few chapters for review.

12. **18th April - 14th May** *4 weeks* -
    complete and review the dissertation.