

Abstract

Language Translation is a pivotal application in the field of Natural Language Processing, which helps in cross-linguistic communication between people for better understanding. Our project focuses on training and building a Machine Learning Model that can translate one language to another. We make use of two types of technologies to achieve said goals, which are the Encoder-Decoder architecture and Transformers. Both models leverages deep learning techniques for accuracy.

The Encoder-Decoder model is the simpler of the two models that uses the much simpler Encoder-Decoder Architecture. It uses a Seq2Seq framework to process the input text and convert it to a Context Vector in the Encoder side and then converts the Context Vector to a corresponding output text in a different language on the Decoder Side. We also make use of an Attention Layer to minimize the loss of long-term memory of the LSTMs when the input text is long. However despite the simplicity of the architecture and effectiveness it faces challenges with Vanishing Gradients due to limitations of Recurrent Neural Networks(RNNs).

To overcome these limitations, we implemented a Transformer model. Due to the significant complexity of a Transformer we used transformers from Hugging Face. The transformer architecture consists of stacker encoder-decoder layers, and each layer employs multi-head attention and feed forward neural networks to better handle long range dependencies and parallel processing.

Both models were trained with the same dataset after pre-processing and cleaning and their differences in performance and accuracy(in the form of BLEU scores) were calculated which shows that Transformers provided significantly better results in terms of accuracy of the translation, making Transformers the obvious better choice in context of language translation.

Acknowledgment

We would like to express our deepest gratitude to all individuals and organizations who have contributed to the successful completion of this project regarding Language translation using deep learning.

First we would like to extend our sincere appreciation to Dr. Sumit Srivatsava, without whose guidance and expertise this project would not have existed. His expertise, support and feedback were invaluable in the completion of this project.

We sincerely thank him for his mentorship.

We would like to thank all our colleagues and fellow researchers without whose collaboration, feedback and discussions, this project would not have seen the light of day. Their perspectives and efforts in this matter facilitated fruitful sharing of ideas.

We would like to thank the creator and maintainers of Kaggle Datasets that played an important role in the accuracy and reliability of our model. And we would also like to thank BIT Mesra for facilitating this project.

We sincerely appreciate the contributions of all the people who were directly or indirectly involved in this project. Your support and efforts were pivotal in the making of this project, without which this project would never have been possible.

Date: 16/07/24

Soumyaditya Saha

Btech/10099/21

Chapter 1:

Language Translation using Deep Learning

1.1 Introduction

Language Translation is an important area of study in the field of Natural Language Processing. It helps us to communicate with people regardless of language barriers and facilitates the exchange of information. It also helps us to use various softwares and services without knowing the language they support.

The ability to translate languages accurately and efficiently helps us bridge cultural and linguistic gaps, making technology and services accessible to a broader audience. As deep learning models continue to evolve, they hold the promise of achieving near-human translation quality, thereby transforming how we communicate and interact in a multilingual world.

Deep Learning has revolutionized the field of Language Translation by significantly improving accuracy and efficiency. Traditional methods, such as rule-based and statistical approaches, have been replaced by much more accurate and efficient neural machine translation (NMT) models that utilize recurrent neural networks (RNNs) and attention mechanisms for better accuracy of translation. These models have demonstrated remarkable success.

The objective of this project is to use such Deep Learning techniques to develop an accurate and efficient language translation system. The goal is to create a Seq2Seq model that can translate the input sequence in a particular language and convert it into an output sequence in a different language while still preserving the meaning and the context of the sequence. The models were created using both Encoder-Decoder Architecture and using the more modern technology of Transformers, and to find out which model functions better in this use-case their differences were laid out and compared.

A dataset containing English Phrases and their corresponding French translations were fed to both the models for training them. The models were then compared to check their efficiency and accuracy. To test their accuracy, per-sentence BLEU(bilingual evaluation understudy) scores were calculated and compared. The models were also differentiated based on the time taken for each of them to be trained using the same dataset using the same hardware.

A Local environment was set up using CUDA and CuDNN to utilize a GPU to drastically lower the training times of both the models. A python environment was created in a WSL environment and Jupyter Notebook was used to write the code for the model. The dataset was pre-processed and cleaned and then fed to the model during the training phase. The model was then evaluated using standard metrics like accuracy, precision, BLEU scores etc.

Successful outcomes from this project has the potential to advance the field of language translation, enabling more accurate and efficient systems. Real-time recognition on user input will also be incorporated, providing practical utility and feedback.

In conclusion, this project aims to leverage the power of deep learning, especially RNNs and Transformers to build a robust and accurate language translation system suitable to be used for various applications. The proposed approach of using transformers, ensures the most accurate and efficient system that can train accurate models on relatively small datasets, compared to traditional methods like Encoder-Decoder architectures.

Chapter 2

2.1 Literature review

Language Translation has been a subject of research for a long time. Various language translation methods have been extensively studied and researched through the years for better accuracy and robustness. In this Literature review, we aim to provide an overview of advancements in this field, and highlight the strengths and limitations of these approaches.

1. RULE BASED MACHINE TRANSLATION(RBMT):

In 1954 a collaboration between IBM and Georgetown University marked the inception of RBMT. This was used to translate Russian sentences into English using a machine with a vocabulary and grammatical rules. By the 1970s RBMT became much more sophisticated and was used by USAF and Atomic Energy Commission. However this method was very resource intensive and needed a lot of effort to develop and maintain, but yet struggled with fluency and needed post-editing.

2. STATISTICAL MACHINE TRANSLATION (SMT):

In the 1990s IBM was involved in developing SMT models that used statistical methods to predict translations based on bilingual corpora. It analyzed similar texts in two languages and tried to understand the patterns. The method was much more efficient and accurate than RBMT and no linguists were needed. The more texts used, the better the translations got. Google Translate used SMT during the majority of its existence.

3. NEURAL MACHINE TRANSLATION (NMT):

In 2014, a research paper on using neural networks for translation was published, which mostly went unnoticed, except by Google. In 2016, Google made an announcement that Translate would be using Neural Networks in place of SMT. Since, the meaning of every word is also dependent on the words surrounding it, CNNs were not ideal for the job. Instead Recurrent Neural Networks(RNNs) were used due to their capability of

remembering the previous result, or in this case the previous word. However, Neural translation might fail to translate short phrases due to lack of context. Yandex Translate uses a hybrid of NMT and SMT and outputs the more appropriate answer.

There are different types of NMT models that are listed below:

4. **SEQ2SEQ (sequence to sequence):**

Seq2Seq models are the foundation of NMT. They consist of two major components: an Encoder and a Decoder. The Encoder converts an input sequence into a fixed-length Context Vector. On the decoder side these Context Vectors are converted back into a sequence in the target language. Seq2Seq models are implemented using RNNs, in the forms of LSTMs(Long Short Term Memory)

Attention Mechanisms are used to address the limitations of Seq2Seq models since they can't handle long sequences without forgetting older words. Attention mechanism provides a path for each LSTM directly to the Decoder.

5. **Transformers:**

Originally introduced in 2017, Transformers represent a significant step-up in the field of NMT. They completely replace RNNs for self-attention mechanisms which help them perform much better for longer input sequences. The Self-Attention mechanism enables the model to determine the importance of words in a sentence relative to each other, which allows for better context understanding. They also use Encoder-Decoder architecture like a Seq2Seq model but with multiple layers of self attention and feed-forward neural networks. This makes sure that long term and short term memories have separate paths. The efficiency and the superior performance of Transformers has made them the industry standard in the field of NMT.

6. Data Pre-Processing:

The literature review highlights the importance of data preprocessing techniques in improving translation accuracy. Techniques such as normalization, resizing, removing empty strings and duplicate strings and mismatched number of sentences were used to enhance the accuracy and consistency of the translation. <start> and <end> tokens were added to mark EOS or SOS for the encoder and decoder.

7. Evaluation metrics:

In this project, we have created a Seq2Seq model using Encoder-Decoder architecture with an added Attention Layer to enable the model to handle longer input sequences. We have also created a Transformer based model that uses MarianMT models which support a wide range of language pairs, while being efficient and accurate. The models were compared based on the time needed to train and their respective per-sentence BLEU (bilingual evaluation understudy) scores were calculated to check which produced more accurate results

8. Future Trends and Innovation:

While Transformers are very accurate and robust when it comes to Language Translation, Large Language Models (LLMs) are already more modern technology which are more capable than transformers. This literature review suggests the use of LLMs for similar projects in the future, which are much more accurate for both longer sentences and document translation.

9. Conclusion:

This Literature Review aims to encapsulate the advancements in the field of Language Translation in brief. From RBMTs to SMTs and to the use of Deep Learning in the form of Seq2Seq models and then to the modern day Transformers, Language Translation has come a long way from all the way back in 1954. While each method has its strength and limitations, one can't help but wonder how far we have come.

This review sets the foundation for the project, and lists out the appropriate methodologies and techniques needed to develop an effective NTM.

Chapter 3

Methodology

3.1. Basic steps to construct a machine learning model

Here we list the basic steps one must follow to successfully replicate this project.

1. Data Collection:

The accuracy of a Machine Learning model is very closely dependent on the quantity and quality of data used to train the model. Accurate and large datasets are essential to ensure the model created is usable. Services like Kaggle and UCI are good sources of datasets with quality data and in quite large quantities.

2. Data Preparation:

Data preparation is essential for ensuring high data quality. Data preparation includes data cleaning where inconsistencies and duplicate data is removed. Errors are corrected or removed as well. Data Transformation involves normalizing the data, scaling it and encoding it to fit the requirements of the algorithm. Data integration involves stitching data from different sources together for more data. And data reduction involves dimensionality reduction and so on.

3. Choosing/creating a model:

Every model has its merits and flaws. Hence, it is important to wisely choose the right model based on our needs.

4. Splitting the Data:

The pre-processed data is now split into two parts, most commonly in the 80:20 ratio.

This means, usually 80% of the data is used for training the model and 20% of the data is used for testing the model once it is trained.

5. Train the model:

The training data is then used to train the model so that it can make predictions based on the data fed to it. Linear regression is a simple example of Training. This happens over multiple training steps after each of which the models parameters are updated. This might happen over multiple Epochs, which are one complete iteration through the dataset, depending on the complexity of the model.

6. Evaluate the model:

Once training is complete the model is evaluated based on its functioning. The testing data is now used to check if the model can make accurate predictions.

This testing data is a representation of the models performance in the real world with real world data, and is very helpful in tuning the model.

7. Parameter Tuning:

Once we have a working model it is often a good idea to tune the various parameters to squeeze out more performance and accuracy from the model. These tunable hyperparameters often include number of training steps, learning rate, number of Epochs etc.

8. Making Predictions:

Once we have a working model that we are satisfied with, the model is used to make predictions based on data the model has not been fed before. This is a much better approximation of how well the model will perform in the real world.

3.2 Methodologies for Neural Language Translation.

3.2.1 Choosing a dataset

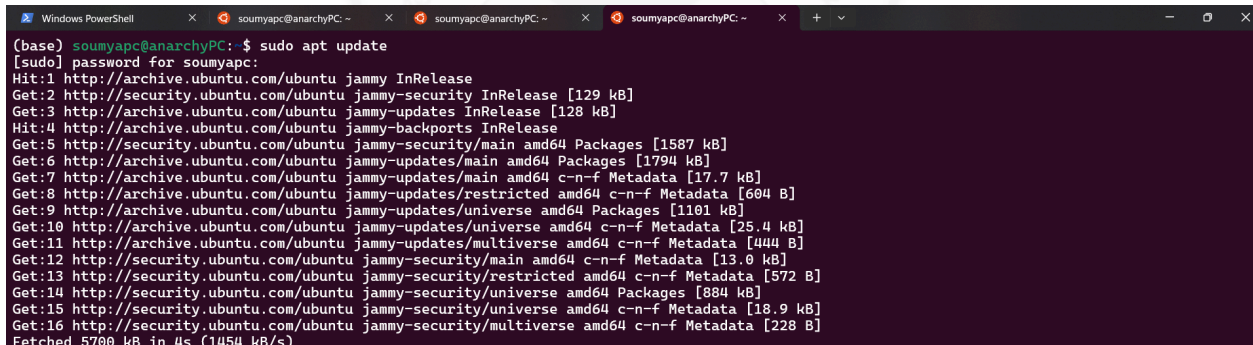
The dataset used for this project is a collection of common English phrases and their French translations. This dataset was downloaded from Kaggle and it consists of over 12000 phrases in English and French.

3.2.2 Setting up the Environment

In this project the model was originally planned to be trained in Google Colab using the T4 GPU. But this idea was later scrapped as the three hour access time limit on the free plan was not enough to train the transformer model. So we ended up running the model locally using a discrete GPU to accelerate the training.

Choice of Platform

Windows is not the ideal choice of platform since it is impossible to run the latest version of Tensorflow which is not compatible with the latest version of CUDA or CuDNN. The ideal platform for this project was Ubuntu or any other Ubuntu based Linux Distro. An alternative to that is using the Windows Subsystem for Linux.



```
(base) soumyapc@anarchyPC: ~$ sudo apt update
[sudo] password for soumyapc:
Hit:1 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:4 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:5 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [1587 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1794 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [17.7 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 c-n-f Metadata [604 B]
Get:9 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1101 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 c-n-f Metadata [25.4 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 c-n-f Metadata [444 B]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/main amd64 c-n-f Metadata [13.0 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64 c-n-f Metadata [572 B]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages [884 kB]
Get:15 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Metadata [18.9 kB]
Get:16 http://security.ubuntu.com/ubuntu jammy-security/multiverse amd64 c-n-f Metadata [228 B]
Fetched 5700 kB in 4s (1454 kB/s)
```

Installing CUDA and CuDNN

CUDA is a parallel computing platform that can significantly reduce waiting time in Training and evaluating neural networks by enabling the use of parallel processing. CuDNN is a GPU-accelerated library for deep neural networks, providing highly optimized implementations for standard routines that are essential for training and inference in deep learning models. This is important to run Tensorflow-gpu.

Creating a Deep Learning Environment

A deep learning environment needs to be created locally. This environment must run Python 3.9 because as of 2024 Tensorflow supports only upto Python 3.9 even though the latest python version is python 3.12. For achieving this a Miniconda installation is preferred over python because we can change python versions quite easily through the terminal.

3.2.3 Installing Deep Learning Libraries

Libraries required for Deep Learning are Tensorflow, Kera with a Tensorflow back-end, NumPy, Scikit-Learn.

3.2.3.1 Tensorflow

TensorFlow is an open-source machine learning framework developed by Google. It helps us create and deploy machine learning models. TensorFlow supports a wide range of machine learning tasks, from simple linear regression to complex deep learning models. It can use GPUs to accelerate machine learning and computational tasks. It serves as the base for Deep Learning models using user-friendly wrapper libraries. Since it uses GPUs by default for computational tasks it can significantly reduce wait times while training models especially while using large complicated datasets, as GPUs can handle parallel processing more efficiently than CPUs. This makes TensorFlow the preferred choice for researchers and production in the field of machine learning.

3.2.3.2 Keras

Keras is a high-level deep learning API. It was developed with python and it runs on top of TensorFlow or PyTorch. It comes pre-packaged with TensorFlow so we can simply install Keras when we run this command in the terminals.

```
/home/soumyapc/.hushlogin file.  
(base) soumyapc@anarchyPC:~$ pip install tensorflow  
Collecting tensorflow
```

Its simplicity and user-friendliness allows for rapid prototyping and experimentation, which makes it particularly useful in the context of Machine Learning. Keras support for various flexible models like the Sequential model and the Functional API model, enables the creation of complex models like the Encoder-Decoder and the Transformers. Its tight integration with

TensorFlow, enables it to use GPU for parallel computing, which significantly improves speeds while handling and training large datasets and complex computations. Keras built-in support for Attention Mechanisms makes Encoder-Decoder models much more accurate for longer sequences.

3.2.3.3 NumPy

In 2005, Travis Oliphant, created NumPy which was a free open-source package for Python which was very useful for scientific computing. NumPy's biggest feature is the support for large and multi-dimensional arrays, which was a massive upgrade from Lists, and matrices and a vast collection of mathematical functions that can be used on these arrays or anywhere else in python for that matter. This makes it an invaluable tool for mathematical operations, making it an indispensable tool for machine learning and deep learning. It can handle extremely large datasets with ease, making it ideal for a translation model as it has to iterate through large data sets. It also introduces the concept of Vectorization which helps us calculate the dot product of two vectors using parallel processing using the **np.dot()** function. Additionally, NumPy integrates seamlessly with other scientific libraries, such as SciPy and Matplotlib. Its robustness and efficiency makes it a must for a Machine Learning project.

3.2.4 Dataset

The dataset used in this project comes in the form of two .txt files called small_vocab_en.txt and small_vocab_fr.txt.

```
# Define file paths
english_file_path = 'small_vocab_en.txt'
french_file_path = 'small_vocab_fr.txt'
def read_lines(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()
    return lines
english_sentences = read_lines(english_file_path)
french_sentences = read_lines(french_file_path)

num_lines_to_display = 10
for i in range(num_lines_to_display):
    english_sentence = english_sentences[i].strip()
    french_sentence = french_sentences[i].strip()
    print(f"EN: {english_sentence} | FR: {french_sentence}")

EN: new jersey is sometimes quiet during autumn , and it is snowy in april . | FR: new jersey est parfois calme pendant l' automne , et il est neigeux en a
vril .
EN: the united states is usually chilly during july , and it is usually freezing in november . | FR: les états-unis est généralement froid en juillet , et
il gèle habituellement en novembre .
EN: california is usually quiet during march , and it is usually hot in june . | FR: california est généralement calme en mars , et il est généralement cha
ud en juin .
EN: the united states is sometimes mild during june , and it is cold in september . | FR: les états-unis est parfois légère en juin , et il fait froid en s
eptembre .
EN: your least liked fruit is the grape , but my least liked is the apple . | FR: votre moins aimé fruit est le raisin , mais mon moins aimé est la pomme .
EN: his favorite fruit is the orange , but my favorite is the grape . | FR: son fruit préféré est l'orange , mais mon préféré est le raisin .
EN: paris is relaxing during december , but it is usually chilly in july . | FR: paris est relaxant en décembre , mais il est généralement froid en juillet
.
EN: new jersey is busy during spring , and it is never hot in march . | FR: new jersey est occupé au printemps , et il est jamais chaude en mars .
EN: our least liked fruit is the lemon , but my least liked is the grape . | FR: notre fruit est moins aimé le citron , mais mon moins aimé est le raisin .
EN: the united states is sometimes busy during january , and it is sometimes warm in november . | FR: les états-unis est parfois occupé en janvier , et il
est parfois chaud en novembre .
```

Small_vocab_en.txt consists of English sentences of varying lengths each separated by a line break, meaning the file contains lines of English Sentences. The small_vocab_fr.txt contains the same sentences but in French. Each of these txt files have over 137860 lines of text.

Heres the first few sentences from each dataset next to their french translations.

And following is the sample of the datasets separately:

```
small_vocab_en1.txt
File Edit View

new jersey is sometimes quiet during autumn , and it is snowy in april .
the united states is usually chilly during july , and it is usually freezing
california is usually quiet during march , and it is usually hot in june .
the united states is sometimes mild during june , and it is cold in september
your least liked fruit is the grape , but my least liked is the apple .
his favorite fruit is the orange , but my favorite is the grape .
paris is relaxing during december , but it is usually chilly in july .
new jersey is busy during spring , and it is never hot in march .
our least liked fruit is the lemon , but my least liked is the grape .
the united states is sometimes busy during january , and it is sometimes warm
the lime is her least liked fruit , but the banana is my least liked .
he saw a old yellow truck .
india is rainy during june , and it is sometimes warm in november .
that cat was my most loved animal .
he dislikes grapefruit , limes , and lemons .
her least liked fruit is the lemon , but his least liked is the grapefruit .
california is never cold during february , but it is sometimes freezing in ju
china is usually pleasant during autumn , and it is usually quiet in october
paris is never freezing during november , but it is wonderful in october .
the united states is never rainy during january , but it is sometimes mild in
china is usually pleasant during november , and it is never quiet in october
the united states is never nice during february , but it is sometimes pleasant
india is never busy during autumn , and it is mild in spring .
paris is mild during summer , but it is usually busy in april .
france is never cold during september , and it is snowy in october .
california is never cold during may , and it is sometimes chilly in march .
he dislikes lemons , grapes , and mangoes .
their favorite fruit is the mango , but our favorite is the pear .
france is sometimes quiet during may , and it is never chilly in august .
paris is never pleasant during september , and it is beautiful in autumn .
he dislikes apples , peaches , and grapes .
california is usually freezing during december , and it is busy in april .

Ln 1, Col 1 9,085,267 characters
```

```
small_vocab_fr1.txt
File Edit View

new jersey est parfois calme pendant l'automne , et il est neigeux en avril .
les états-unis est généralement froid en juillet , et il gèle habituellement en
california est généralement calme en mars , et il est généralement chaud en juin
les états-unis est parfois légère en juin , et il fait froid en septembre .
votre moins aimé fruit est le raisin , mais mon moins aimé est la pomme .
son fruit préféré est l'orange , mais mon préféré est le raisin .
paris est relaxant en décembre , mais il est généralement froid en juillet .
new jersey est occupé au printemps , et il est jamais chaude en mars .
notre fruit est moins aimé le citron , mais mon moins aimé est le raisin .
les états-unis est parfois occupé en janvier , et il est parfois chaud en novemb
la chaux est son moins aimé des fruits , mais la banane est mon moins aimé .
il a vu un vieux camion jaune .
inde est pluvieux en juin , et il est parfois chaud en novembre .
ce chat était mon animal le plus aimé .
il n'aime pamplemousse , citrons verts et les citrons .
son fruit est moins aimé le citron , mais son moins aimé est le pamplemousse .
californie ne fait jamais froid en février , mais il est parfois le gel en juin
chine est généralement agréable en automne , et il est généralement calme en oct
paris est jamais le gel en novembre , mais il est merveilleux en octobre .
les états-unis est jamais pluvieux en janvier , mais il est parfois doux en octo
chine est généralement agréable en novembre , et il est jamais tranquille en oct
les états-unis est jamais agréable en février , mais il est parfois agréable en
l'inde est jamais occupé à l'automne , et il est doux au printemps .
paris est doux pendant l'été , mais il est généralement occupé en avril .
france ne fait jamais froid en septembre , et il est neigeux en octobre .
californie ne fait jamais froid au mois de mai , et il est parfois frisquet en m
il déteste les citrons , les raisins et les mangues .
leur fruit préféré est la mangue , mais notre préféré est la poire .
la france est parfois calme au mois de mai , et il est jamais froid en août .
paris est jamais agréable en septembre , et il est beau à l'automne .
il déteste les pommes , les pêches et les raisins .
la californie est le gel habituellement en décembre , et il est occupé en avril

Ln 1, Col 1 9,847,064 characters
```

3.2.4.1 Data Pre-Processing

Data preprocessing is a crucial step in the machine learning workflow. It improves Model accuracy by cleaning and formatting the data. Preprocessing makes sure that the machine learning model is trained only on relevant and accurate information, which helps in improving the model's accuracy and performance by eliminating noise from the data.

It also replaces missing values with actual data and removes outliers which can negatively impact the models accuracy.

Steps like normalization ensures standardisation of the data so as to not confuse the model with different scales of data.

Removing redundancy from the data also prevents over-fitting and reduces the training time.

There are a lot of steps involved in data pre-processing and they will be listed and explained below:

Data Quality Assessment:

Data Quality Assessment is a very important step in data management. It involves evaluating the data based on its qualities such as accuracy, completeness, consistency, reliability, and validity.

The primary goal of DQA is to ensure that data is fit for its intended purpose and won't negatively impact the predictions of the model. Data quality assessment ensures the accuracy of the data. Accurate data means the reliable predictions can be made using the trained the model. It also assesses that the data is complete and any gaps in the data must be filled or removed. DQA is also important to assess the consistency of the data with no conflicting value.

Data quality assessment is done at many steps of the process. It is doen after the initial data collection phase to check if the data is of good quality or not. It is also done while merging data from different sources to resolve inconsistencies and errors. Before training begins DQA is performed to ensure the data is fit to train the model with.

Data Cleaning:

In data Cleaning or data scrubbing, errors, inconsistencies, and inaccuracies in datasets are identified and corrected (or removed) . This step is crucial in ensuring that the data used for analysis is of high quality and reliability. Data cleaning enhances the quality of data. It removes duplicate records, corrects misspelled information, and standardizes data formats. Clean data reduces the time and effort required to process and analyze data, and also reduces the training time of the model. This also reduces chances of false predictions from the model. Also clean data is much easier to integrate with other sources of data. Data Cleaning involves the following steps:

1. Removing irrelevant and duplicate data, which reduces the training times.
2. Handle missing values by replacing appropriate values or removing them completely.
3. Standardizing the data so that the data is in a consistent format.

Data Transformation:

Yet another crucial step in data pre-processing. It involves formatting the raw data into a format that is suitable for analysis and modeling. This is crucial to ensure compatibility of the data with different models and enhance data quality. This also ensures possibilities of feature engineering such that new features can be created that can help in improving the model. The process involves converting raw, isolated, and normalized data into a cohesive, dimensionally modeled, and de-normalized state, ready for analysis. Data transformation helps in creating organized and pure data which is the bedrock of accurate predictions using ML models. Transformed Data is very valuable.

Data reduction:

Data Reduction is the process of creating a more compact representation of the original data without losing any of the essential information contained in it. It is achieved by employing various sophisticated techniques. Data reduction minimizes the volume of data while preserving its meaning. However, one must ensure that the time spent on data reduction does not exceed the time saved by training the reduced data. We must strike the right balance which can make data reduction an invaluable tool for extracting the meaning of the data without being overwhelmed by its sheer magnitude.

Reducing the size of the dataset is essential to enhance the efficiency of machine learning algorithms, making them faster and more practical to use. Smaller datasets require less storage space, leading to significant cost savings in data storage.

Techniques of data reduction:

1. Dimensionality reduction:
 - a. Wavelet transforms is a mathematical process to transform data into a different domain to make it easier to analyze and compress
 - b. Principal Component Analysis(PCA): This is a statistical model that reduces the number of features by transforming them into a set of linearly uncorrelated variables called principal components.
 - c. Attribute Subset Selection
2. Numerosity reduction:
 - a. Parametric Methods: Models like regression summarizes the data within a set of parameters.
 - b. Non-Parametric Methods: Clustering and sampling reduces the data size without assuming any specific data distribution.
3. Data compression:
 - a. Lossless compression: this involves using advanced algorithms to compress the data without suffering any loss of data
 - b. Lossy compression: removes some information to decrease the size of the dataset. This may cause some loss in data quality.

In case of Transformers and Encoder-Decoder architectures, data might also go through:

1. Conversion from a Pandas DataFrame to Hugging Face Datasets
2. Tokenization : data is often tokenized to have minimum and maximum lengths before its fed to a pretrained model.
3. Data Splitting: data is split into training and testing data for the training and testing phase of the model.

3.2.5.1 Use of RNNs in Natural Language Translation

Introduction

Language barriers have always hindered with global communication and even local communication among people of different ethnicities and culture. However, advancements in neural machine translation(NMT) make use of Recurrent Neural Networks (RNNs) to emerge as one of the most promising solution. This report discusses the use of RNNs in NMT by analyzing their impact in Translation technology.

Advantages of RNNs

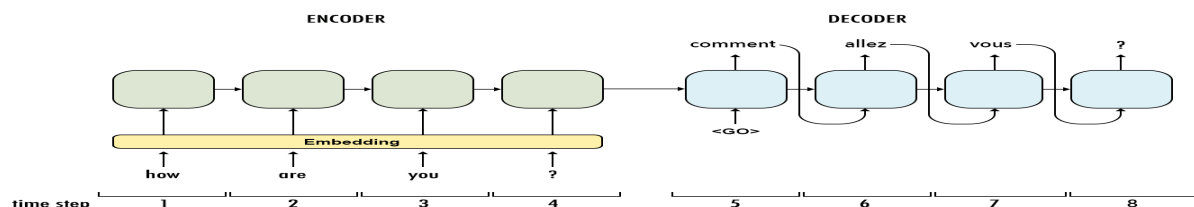
- Unlike traditional methods, RNNs can handle long sequential sentences. Use of Context Vectors means they understand the meaning of each word wrt the words surrounding it.
- RNNs possess retention memory which allows them to retain information from previous words in a sentence. Context awareness makes sure translations remain accurate and efficient.
- Bidirectional RNN architectures analyze sentences in both directions, resulting in more comprehensive understanding and more natural-sounding translations.
- Use of attention mechanisms allows RNNs to focus on earlier parts of the input sentence during translation, improving accuracy for long and complex sentences.

The adoption of RNNs in NMT has seen incredible results. Google Translate's impressive real-time translation capabilities are powered by RNNs, which can help translating sentences in real time.

RNNs have revolutionized NMT, but yet, they face significant challenges in the future. Running RNNs is computationally very expensive. This makes then not scalable for massive projects.

Even though RNNs can handle long sequences quite well, they can struggle with extreme long sequences due to the problem of vanishing gradient.

While RNNs have been a significant advancement in propelling NMT forward, compared to more traditional methods, their limitations have triggered further research in this field. Their ability to handle long sentence, understand context and using attention mechanisms has ensured accurate translations. However, Transformer Models have already seen a significant rise in this field and will most likely overtake RNNs as the most used technology in this field.



3.2.5.2 Use of Transformers in Neural Machine Translation

Introduction

Arrival of Transformers has marked a significant step-up in the field of Neural Machine Translation (NMT). Unlike traditional methods, Transformers use the innovative technology of "self-attention" to understand the meaning of the entire sentence. Unlike RNNs, which processes the sequences step-by-step, Transformers do it in one single step. This Report discusses the use of Transformers in NMT and how they are one of the best technologies in this field.

Advantages of using Transformers

- The biggest advantage of Transformers is their self-attention mechanism. This approach allows the model to understand the significance of various words within a sentence relative to other words. RNNs often struggle with understanding the meaning of lengthy sequences, but transformers work much better under similar scenarios.
- RNNs require word-by-word processing while Transformers can analyze entire sentences simultaneously. This parallel processing capability reduces training and inference times, making them better for large datasets.
- By stacking multiple layers of self-attention and feed-forward networks, Transformers can handle large translation tasks with great accuracy. This makes transformers very scalable and hence are preferred over the use of RNNs
- RNNs, including variants like Long Short-Term Memory (LSTM) networks, suffer from the vanishing gradient problem that makes them useless for long sequences or documents. Transformers are much more capable at handling large sequences and even documents and the use of Parallel computing make them much more efficient.
- Transformers exhibit much better performance than RNNs while training on languages for which low amount of resources are available like Swahili etc which is why Google uses Transformers in their Google's Neural Machine Translation (GNMT) systems

The introduction of Transformer has marked another significant step-up in the field of Neural Machine Translation. Their ability to handle long sequences, their efficiency in parallel processing, and superior performance make them the preferred choice over RNNs. As research and development in this domain continue, we can anticipate further advancements in translation quality and further growth in the field of NMT.

3.2.5.3 LLMs as the future of NMT

In recent years, Large Language Models (LLMs) have become very popular in the field of Neural Machine Translation (NMT). They use vast amounts of data and sophisticated architectures and offer significant improvements over traditional NMT models like Recurrent Neural Networks and even Transformers.

Advantages of LLMs

- LLMs are trained on extensive multilingual datasets containing multiple billions of parameters, enabling it to capture intricate language patterns and nuances better than smaller models
- LLMs utilize self-attention mechanisms which allow them to process entire sentences simultaneously and effectively, resulting in more accurate and contextual translations.
- LLMs are highly scalable, meaning their performance improves significantly as they are trained on larger datasets.

Large Language Models are the future of Neural Machine Translation. They offer unparalleled flexibility, scalability, and performance unlike RNNs or Transformers. They can handle complex language tasks, adapt to contexts, and provide high-quality translations. This makes them a far superior choice over traditional NMT models. As research continues, LLMs are expected to further revolutionize machine translation, breaking down language barriers, thus facilitating global communication.

3.2.6.1 Working of RNNs (Encoder-Decoder Model)

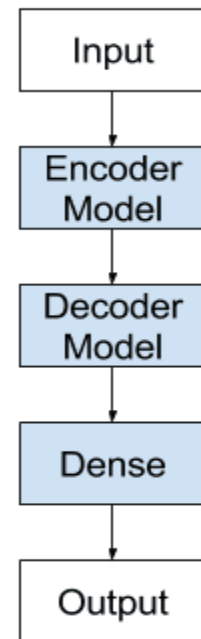
An encoder-decoder model is a type of Recurrent Neural Networks(RNNs) designed for sequence-to-sequence (seq2seq) tasks, where the input is a sequence and so is the output but of different lengths. This is a widely used Architecture and can be used in a lot of Machine Learning models, including Neural Machine Translation (NMT). This report aims to provide a detailed explanation of how these models work.

Encoder-Decoder Architecture

The encoder-decoder model consists of two main components: the encoder and the decoder.

The job of the encoder is to convert a sequence into a context vector. And the job of the decoder is to convert that context vector back to a sequence but in a different language in this case. More details about them:

- **Encoder:** The encoding layer processes the input sequence and compresses the information into a vector and feeds it to a layer of Long Short-Term Memory (LSTM). The Last unrolled LSTM outputs this input in the form of a fixed length Context Vector . This vector captures the meaning and context of the input sequence. This Context Vector is fed to the Decoder for decoding. The encoder uses multiple layers of Long Short-Term Memory (LSTM) units or Gated Recurrent Units (GRUs) to handle variable-length input sequences and maintain long-range dependencies. Layers of LSTMs are stacked/rolled and each unit is supposed to store one token of the sequence, that is one word. The process is listed sequentially here:
 1. The input sequence is tokenized and the embedding layer converts these to numerical representations. These are also known as Vectors.
 2. The vectorised input is fed into LSTM layers, where each LSTM cell processes one token at a time, while also being able to capture information from previous tokens.
 3. The output from the last layer of LSTM cells is in the form of a context vector, which is capable of encapsulating the meaning and essence of the input sequence.



- **Decoder:** The context vectors produced by the encoder are fed to the decoder. It is the job of the Decoder to convert the Context vectors back to an output sequence using a different vocabulary while retaining the essence of the sentence. The decoder generates the output sequence one token at a time by making use of the context vector provided by the encoder. It also employs LSTM units and an embedding layer to handle the output sequence and has a fairly similar structure to an Encoder. Here, the functioning of the Decoder is listed sequentially:

1. The decoder's LSTM cells are fed with the context vector from the encoder. The output from the last layer of LSTM in the Encoder acts as the Input for the First Layer of LSTMs for the Decoder.
2. The decoder starts with finding a special token (e.g., <SOS> or <EOS>) and generates the next token based on the present hidden state and the last token.
3. The Decoder keeps generation tokens until it reaches another special token which can either be the <SOS> token or the <EOS> token.

Training the model:

This is achieved by making use of supervised learning and the model is trained on a set of input and output sequences (x and y respectively) after which it is evaluated on the basis of the predicted value of y (y') based on the input x.

Known words are often plugged into the decoder and the decoder is often force stopped at known phrases depending on the context. This method is termed as **Teacher Forcing**.

The proper Training of the model depends on appropriate values of weights and biases. These appropriate values are calculated using Back-Propagation, by minimizing the error between the actual and predicted output.

Advantages

1. Encoder-Decoder models can handle input and output sequences of different lengths thus making them ideal for machine translation.
2. Encoder-Decoder models are capable of understanding the context of the sequence and hence understand the meaning and essence of each word with respect to the other words.

However, when Long sequences are fed to this model the Long Term and Short term memory share the same path way making the model forget the earlier parts of long sequences . This is called Vanishing Gradient. This is solved by making use of Attention Mechanisms.

Attention Mechanisms

The attention mechanism resolves one of the most significant encoder-decoder model. Fixed Length Vectors can struggle to capture the context of long input sequences. This leads to poor performance on longer sentences. Attention allows the model to focus on different parts of the input sequence as needed by providing a path from each LSTM layer directly to the Decoder. This mechanisms works in the following basis:

1. **Similarity Scores:** For each and every output token, the decoder calculates similarity scores between its present hidden state and all the hidden states of the encoder. The similarity determines which one gets more attention from decoder.
2. **Attention Weights:** Similarity scores that are normalized using a softmax function produce attention weights. If these weights add up to 1, it indicates the importance of each input token for curretn output token.
3. **Token Generation:** The decoder uses context vector and its current hidden state to generate the next token in the output sequence.

Similarity Scores are calculated using Cosine Similarity. The following is the formula of Cosine Similarity:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Advantages of Attention Mechanisms:

1. Improved accuracy
2. Ability to handle longer sequences
3. Interpretable due to attention weights

By allowing an Encoder-Decoder model to automatically and dynamically focus on only the relevant parts of the input sequence, attention mechanisms increase translation accuracy and enable the model to handle longer and more complex sentences.

3.2.6.2 Working of Transformers

Transformers have been a major revolution in the field of Natural Language Processing. In Neural Translation Models they provide a more efficient and much more accurate method for translating text from one language to another. They were originally developed to address the limitations of Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). Transformers utilize very innovative self-attention mechanisms which lets them process entire sequences and sentences simultaneously while understanding the context and meaning of each word. This allows Transformers to capture the meaning and essence of long-range sequences more effectively. This report studies the workings of Transformer models, their pre-trained versions, and their application in NMT.

Architecture of Transformers

In the following part of the report we will delve into the actual architecture and working of Transformer models.

Transformers consist of what essentially is an encoder-decoder architecture, but unlike traditional RNN models, they employ various self-attention mechanisms which allow them to make use of parallel processing to process the input sequences.

1. Encoder:

The encoder processes the input sequence. It generates a set of continuous representations of this sequence. It consists of multiple layers and each layer contains two main components:

- a. Self-Attention Mechanism:** Self-Attention mechanisms allow the encoder to understand the context and importance of different words in the input sequence compared to the words surrounding it. It calculates attention scores for each word pair in the sequence which enables the model to focus on relevant parts of the input.
- b. Feed-Forward Neural Network:** after passing through the self-attention layer the output is passed through a feed-forward neural network.

2. Decoder:

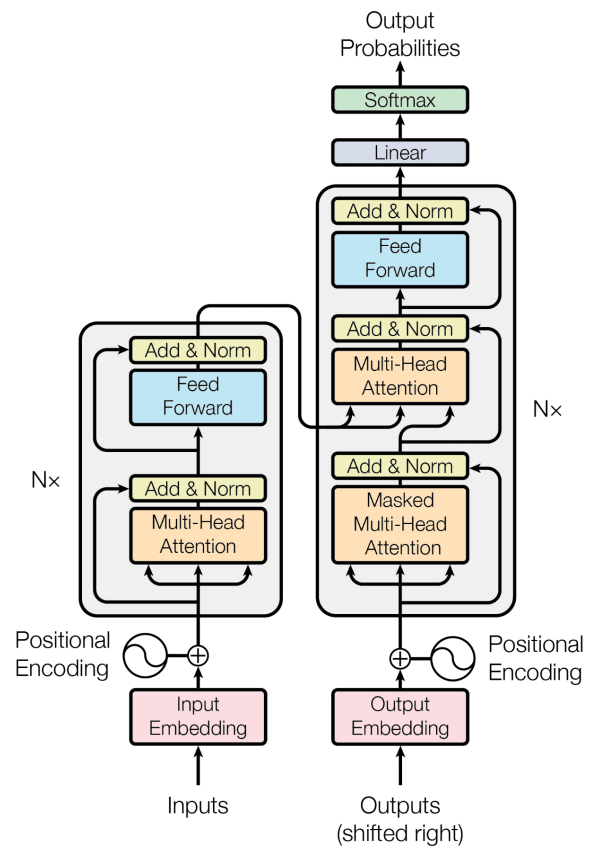
The output from the encoder is fed to the decoder, where the decoder generates an output sequence after it attends to the encoder's output while keeping in context the previously generated tokens. It also consists of multiple layers where each layer contains three main components:

a. Self-Attention Mechanism: much like the Encoder, the Decoder also contains a self-attention mechanism. This mechanism lets the Decoder focus on relevant parts of the output sequence generated this far.

b. Encoder-Decoder Attention mechanism: This layer helps the decoder focus on relevant parts of the input sequence while decoding the output from the Encoder. This is achieved by attending to the encoder's output. This helps the decoder understand the context and meaning of specific parts of the output thus judging its relevance.

c. Feed-Forward Neural Networks:

much like the encoders, decoders have Feed-Forward Neural Networks of their own. This is the final layer of decoder, which possesses all the information of the previous layers.



Self-Attention Mechanism

The technology that makes transformers as powerful as they are is the Self-Attention Mechanism. This helps the model to process the entire input sequence simultaneously, rather than processing it token after token. This mechanism works on three major steps:

- a. **Calculating Attention Scores:** For every word in the input sequence, the Transformer calculates attention scores with respect to the other words. These scores help determining the relevance of every single word with respect to the current word.
- b. **Applying SoftMax tech:** The softmax function normalizes the attention scores to produce attention weights. These weights are used to indicate the importance of each word in the input sequence.
- c. **Weighted Sum technology:** Attention scores are used to weigh the Input Words and added to produce a new representation for each word. This helps the model to capture the context of the entire input sequence.

Using Pre-trained Transformers:

In this project we have made use of the MarianMT transformer from the Machine Learning and Artificial Intelligence Repository called Hugging Face. This is a pre-trained model that is specialised in the field of NLP. This model is extremely competent in understanding sequences and translating them to different language after being trained on a particular dataset.

Other pre-trained Transformer models, such as BERT, GPT, and T5, have been trained on massive datasets using highly powerful hardware like TPUs. These can be fine-tuned for specific tasks as well as that includes NMT. Once a pre-trained Transformer model is loaded, which is already been trained, it is fine-tuned on a specific dataset for the specific task. In our case it was trained using an english to french dataset. The weights may be adjusted to improve the performance of the model.

Advantages of using Transformers for NMT

RNNs process sequences sequentially. But, Transformers process entire sequences simultaneously which lead to much faster training times. They make use of self-attention mechanisms which allow Transformers to capture the context of long-range sequences more effectively, improving translation accuracy. And last but not least, Transformers can be scaled up very easily by increasing the number of layers and the number of attention heads. This leads to better performance on complex tasks.

Chapter 4

EXPERIMENTAL ANALYSIS AND RESULTS

4.1 System Configuration:

Platform :

1. Ubuntu 22.04 LTS or,
2. Windows Subsystem for Linux with Ubuntu.

Windows is not the ideal choice of platform since it is impossible to run the latest version of Tensorflow which is not compatible with the latest version of CUDA or CuDNN. The ideal platform for this project was Ubuntu or any other Ubuntu based Linux Distro. An alternative to that is using the Windows Subsystem for Linux.

Drivers and other Softwares: if you are planning to use a discrete GPU for the training purposes then installing the following is must:

1. Latest Nvidia Studio drivers for said GPU.
2. CUDA 12.5 or previous
3. Compatible CuDNN

Hardware requirements: These are the Hardware interfaces used Processor: Intel Pentium 4 or equivalent.

1. VRAM(GPU): atleast 4GB
2. RAM: Minimum of 4GB or higher HDD: 50 GB or higher
3. CPU: i3 or greater
4. Mouse: 1x mice
5. Keyboard: yes

Python: recommended version 3.9.x. Tensorflow does not support versions of python more than 3.10.x. From my personal experience, I believe python 3.9.13 is the best version for the project. Created in the late 1980s by Guido van Rossum, Python was initially a hobby project. It was released in 1991. Python's design had readability and simplicity as its main priority. Python aimed to provide an elegant and efficient language for developers. Its clear and simple syntax and structure encourage clean and maintainable code, thus making it useful for both beginners and professionals.

Deep Learning Environment: To create this project a deep learning environment was locally created. In this deep learning environment Tensorflow, Keras, Scikit-learn, NumPy, PyTorch and Matplotlib were installed. Keras was installed with the Tensorflow back-end.

Jupyter Notebook: in this Deep Learning Environment Jupyter Notebook was used as an IDE. Jupyter Notebook is an interactive computing environment. It allows its users to create and share documents containing live codes, python notebooks etc. It supports various programming languages, including Python, R, and Julia. Users can execute code real-time, view results, and so on. Jupyter notebook is one of the most popular IDE in the field of Data Science and Machine Learning due to its simplicity and ease of use, and also how fast it performs. Jupyter Notebook is the recommended IDE for this project.

It can be easily installed in your local environment using the command:

pip install jupyter

And can be easily launched using the command:

Jupyter notebook

```
4 packages can be upgraded. Run 'apt list --upgradable' to s
(base) soumyapc@anarchyPC:~$ pip install jupyter
```

```
4 packages can be upgraded. Run 'apt list --upgradable' to s
(base) soumyapc@anarchyPC:~$ jupyter notebook
```

4.2 Translation Results:

4.2.1 Using the Encoder-Decoder Model (RNN)

At first the encoder and a decoder was created in Jupyter Notebook in a local environment using the following code.

```
# Encoder
encoder_inputs = Input(shape=(None,))
encoder_embedding = Embedding(eng_vocab_size, 256)(encoder_inputs)
encoder_lstm = LSTM(256, return_sequences=True, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(encoder_embedding)
encoder_states = [state_h, state_c]

# Decoder
decoder_inputs = Input(shape=(None,))
decoder_embedding = Embedding(fr_vocab_size, 256)(decoder_inputs)
decoder_lstm = LSTM(256, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(decoder_embedding, initial_state=encoder_states)
```

This model was trained using the English to French Dataset and then evaluated. But the lack of an attention layer meant that the accuracy of the translation was not very high. So the attention mechanism was then coded:

```
# Attention layer
attention = Attention()([decoder_outputs, encoder_outputs])
decoder_concat_input = Concatenate(axis=-1)([decoder_outputs, attention])

decoder_dense = Dense(fr_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_concat_input)
```

This definitely improved the accuracy of the translation. The model was then defined:

defining the model

```
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Training the model gave this output:

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, None)	0	-
input_layer_3 (InputLayer)	(None, None)	0	-
embedding_2 (Embedding)	(None, None, 256)	58,880	input_layer_2[0]...
embedding_3 (Embedding)	(None, None, 256)	91,648	input_layer_3[0]...
lstm_2 (LSTM)	[(None, None, 256), (None, 256), (None, 256)]	525,312	embedding_2[0][0]...
lstm_3 (LSTM)	[(None, None, 256), (None, 256), (None, 256)]	525,312	embedding_3[0][0]... lstm_2[0][1], lstm_2[0][2]
attention_1 (Attention)	(None, None, 256)	0	lstm_3[0][0], lstm_2[0][0]
concatenate_1 (Concatenate)	(None, None, 512)	0	lstm_3[0][0], attention_1[0][0]
dense_1 (Dense)	(None, None, 358)	183,654	concatenate_1[0]...

Total params: 1,384,806 (5.28 MB)

Trainable params: 1,384,806 (5.28 MB)

Non-trainable params: 0 (0.00 B)

The model was then trained for 20 epochs (20 iteration of the entire data set) for a batch size of 128 (the model was updated after going through 128 lines of data). Time taken to train the model:

On evaluating the model the test loss was found to be 0.81% and the test accuracy was found to be 99.7%.

862/862 ————— 8s 9ms/step - accuracy: 0.9976 - loss: 0.0082
Test Loss: 0.008161053992807865, Test Accuracy: 0.9976314306259155

With CPU it took a total of 27 minutes and 34 seconds to complete the training. However, with the use of GPU and parallel computing the total training time was reduced to just 5 minutes and 24 seconds.

The translations from this model was relatively accurate, for shorter sentences but inaccurate for longer sentences

```

sample_sentence = "i like oranges"
sample_sequence = eng_tokenizer.texts_to_sequences(['<start>' + sample_sentence + '<end>'])
sample_padded = pad_sequences(sample_sequence, maxlen=eng_padded.shape[1], padding='post')
translated_sentence = translate_sentence(sample_padded)
print(f'English: {sample_sentence}')
print(f'French: {translated_sentence}')

1/1 ————— 0s 81ms/step
1/1 ————— 0s 85ms/step
Iteration: 0, Sampled Char: <start>
1/1 ————— 0s 17ms/step
Iteration: 1, Sampled Char: j'aime
1/1 ————— 0s 17ms/step
Iteration: 2, Sampled Char: les
1/1 ————— 0s 17ms/step
Iteration: 3, Sampled Char: oranges
1/1 ————— 0s 17ms/step
Iteration: 4, Sampled Char: .

1/1 ————— 0s 17ms/step
Iteration: 5, Sampled Char: <end>
English: i like oranges
French: <start> j'aime les oranges .
<end>

```

English: she likes apples
French: <start> elle aime les pommes .
<end>

English: eat an apple
French: <start> at n'aimez pas les citrons
<end>

This model managed to translate small sentences but was extremely inaccurate in translating large sentences. No wonder it only managed to get a BLEU score of

Sentence BLEU score: 0.4164756475

Corpus BLEU score: 0.3293743843

```
[24]: sample_sentence = "Photography is my favourite hobby"
sample_sequence = eng_tokenizer.texts_to_sequences(['<start> ' + sample_sentence + ' <end>'])
sample_padded = pad_sequences(sample_sequence, maxlen=eng_padded.shape[1], padding='post')
translated_sentence = translate_sentence(sample_padded)
print(f'English: {sample_sentence}')
print(f'French: {translated_sentence}')
```

1/1 ————— 0s 45ms/step
1/1 ————— 0s 20ms/step
Iteration: 0, Sampled Char: <start>
1/1 ————— 0s 19ms/step
Iteration: 1, Sampled Char: t
1/1 ————— 0s 20ms/step
Iteration: 2, Sampled Char: mon
1/1 ————— 0s 21ms/step
Iteration: 3, Sampled Char: sont
1/1 ————— 0s 21ms/step
Iteration: 4, Sampled Char: mon
1/1 ————— 0s 19ms/step
Iteration: 5, Sampled Char: fruit
1/1 ————— 0s 20ms/step
Iteration: 6, Sampled Char: préféré.

1/1 ————— 0s 18ms/step
Iteration: 7, Sampled Char: <end>
English: Photography is my favourite hobby
French: <start> t mon sont mon fruit préféré.
<end>

The model produced terrible results for longer sentences.

Since encoder-decoder model failed to produce accurate results even after longer training sessions we shifted to the use of transformers.

4.2.2 Using the MarianMT transformer from Hugging Face

At first a new notebook was created in Jupyter Notebook in a local environment using the following code.

```
!pip install transformers datasets
!pip install transformers[torch]
!pip install accelerate -U
!pip install sentencepiece
!pip install tf-keras
!pip install nltk
```

At first some necessary libraries were installed

Following this the open source Opus-MT model was imported

```
from transformers import MarianMTModel, MarianTokenizer
import sentencepiece as spm

# Load the tokenizer and model
model_name = 'Helsinki-NLP/opus-mt-en-fr'
tokenizer = MarianTokenizer.from_pretrained(model_name)
model = MarianMTModel.from_pretrained(model_name)
```

After preprocessing the data the model training was started

```
from transformers import Trainer, TrainingArguments

training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy='steps',
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    save_total_limit=2,
    load_best_model_at_end=True,
    save_steps=500, # Save every 500 steps
    eval_steps=500 # Evaluate every 500 steps
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train_dataset,
    eval_dataset=tokenized_test_dataset,
)

trainer.train()
```

Step	Training Loss	Validation Loss
500	0.037900	0.009741
1000	0.010000	0.007866
1500	0.008400	0.007066
2000	0.007300	0.006662
2500	0.006900	0.005839
3000	0.005800	0.004991
3500	0.005400	0.004472
4000	0.004900	0.004113
4500	0.004500	0.003850
5000	0.004300	0.003558
5500	0.004000	0.003374

Three epochs were used to train the model. And the model was saved and evaluated at every 500 steps. When GPU accelerated it took a total of 2 hours and 38 minutes to train the model.

The translations from this model is extremely accurate for both short and long sentences. Even complex sentences are quite accurately translated. Here are some examples:

```
english_text = "my name is Soumyaditya|"
french_translation = translate(english_text)
print(french_translation)
```

```
['Mon nom est Soumyaditya']
```

```
english_text = "this is my summer project"
french_translation = translate(english_text)
print(french_translation)
```

```
["C'est mon projet d'été."]
```

```
english_text = "i have been doing photography since I was 5"
french_translation = translate(english_text)
print(french_translation)
```

```
["Je fais de la photographie depuis que j'ai 5 ans."]
```

```
english_text = "photography is my hobby and i love doing it"
french_translation = translate(english_text)
print(french_translation)
```

```
["La photographie est mon passe-temps et j'adore le faire"]
```

The model produced excellent per-sentence BLEU score ranging from 0.7469 to 1.0 which makes this model highly accurate in its translation

Sentence BLEU score: 0.7469344843203212
Corpus BLEU score: 0.7469344843203212

Sentence BLEU score: 1.0
Corpus BLEU score: 1.0

This model showcases the high accuracy of transformer based models in the field of NMT thus making Transformers the superior technology over RNNs.



Chapter 5

Conclusion

5.1 Recommendation to all

For anyone who is planning to perform this project themselves, we highly recommend the use of Transformers over RNNs for more accurate translations. Not only are they more accurate but can also handle longer sentences unlike Encoder-Decoder Models.

5.2 Future work

While transformers are quite good at their job, LLMs are already overtaking transformers in terms of capacity, capability and accuracy. So, this Translation system would be much more effective if LLMs are used in the near future, with more computing resources.

5.3 Conclusion

We set out to on this project aiming to create a Language Translation model which uses modern era technology like RNNs and transformers. By the end of this report we are happy to announce that the project has been a success as we have successfully made a Machine Learning Model that has the capability to translate languages with a very high accuracy.