Abstractive Text Summarization

Submitted in partial fulfillment of the requirements
for the degree of

BACHELOR OF TECHNOLOGY
in
Computer Engineering
by

Darshan K. Tank
(18CEUBG040)

Under the supervision of

Dr. Brijesh Bhatt
Prof. Siddharth Shah

Faculty of Technology
Department of Computer Engineering
DHARMSINH DESAI UNIVERSITY

April 2022

CERTIFICATE

This is to certify that the project work titled

Abstractive Text Summarization

is the bonafide work of

Darshan K. Tank
(18CEUBG040)

carried out in the partial fulfillment of the degree of Bachelor of Technology in Computer Engineering at Dharmsinh Desai University in the academic session

December 2021 to April 2022.

Prof. Siddharth P. Shah	Dr. Brijesh S. Bhatt	Dr. C. K. Bhensdadia
Assistant Professor	Professor	Head Of Department
Dept. of Comp. Eng.	Dept. of Comp. Eng.	Dept. of Comp. Eng.

Faculty of Technology
Department of Computer Engineering
DHARMSINH DESAI UNIVERSITY
April 2022

Acknowledgements

"Guide is all you need" to walk in the proper direction, but, to run in the right direction, you need someone experienced who can guide you in your journey to reach the desired destination.

I would cordially like to thank my guide **Professor Dr. Brijesh Bhatt** for giving me this opportunity to work under him. I can not even visualize the beginning of my research journey without him. His perfection, diligence, and tenacity have inspired me much.

This project would not have been possible without one person and that is **Prof. Siddharth Shah**. He is a friendly, very much supportive, and highly motivated person. I learned a lots of things while working with him. Brainstorming sessions and debates with him on some very interesting topics has given me a very deep understanding of Deep learning.

I would heartily like to thank **Clifton Poth**(Master's Student, Computer Science at Technische Universität Darmstadt - adapterHub) for a positive response while working on the adapter and I can not ignore the help from **Anubhav Jangra**(Pre-doctoral Researcher at Google India) for the great Collaboration with IIT Patna and introducing me with **Dr. Sriparna Saha**.

Finally, I would like to thank family for motivating me. I can not forget the support of my roommates during this internship. Sitting with them makes me feel easy and stress-free.

Thanks to all,
Darshan Tank

Abstract

In the age of the internet, where tremendous information is only one click away, it is equally important to get the relevant and necessary information for a better understanding. In the current fastest growing world, where everybody wants to be the cream of the curd, time plays a crucial and major role. Reading plenty of lines for a better understanding of a topic is quite a headache and time-eater for many of us.

Having a short and sweet summary of huge text data works like ice in the summer for humans which not only gives relief but also saves time. It is very difficult for human beings to manually extract the summary of a large document of text. For a human, a summary is not only the link of extracted important words, but it is the process, which requires a high level of understanding, logic, and thinking to come up with 2-3 fluent sentences that represent the entire text document! Even though we, Humans, begin to summarize each and every document, it gonna takes ages to end up nowhere!

It is most important to come up with a robust mechanism to generate the salient information quickly and most efficiently from the text document. Not only this, a mechanism should generate coherent and cohesive text.

As a solution, we can have an automatic text summarization. Text Summarization is a blessing to the readers. It is a process of identifying the most prominent and meaningful information in a document and compressing them into a shorter version preserving its overall meanings.

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
ML	**M**achine **L**earning
DL	**D**eep **L**earning
SOTA	**S**tate **O**f **T**he **A**rt
PTM	**P**re **T**rained **M**odels
RNN	**R**ecurrent **N**eural **N**etwork
GNN	**G**ated **N**eural **N**etwork
LSTM	**L**ong **S**hort **T**erm **M**emory
MLM	**M**ask **L**anguage **M**odeling
NSP	**N**ext **S**entence **P**rediction
GSG	**G**ape **S**entence **G**eneration
C4	**C**olossal and **C**leaned version of **C**ommon **C**rawl

Chapter 1

Introduction

This is a introductory chapter. In this chapter you will see why we selected the Abstractive Text Summarization as a project. Brief introduction about summary. Applications of automatic text summarization and how we can see this problem as a AI's perspective.

1.1 Motivation and Scope

well, traditional methods of summarizing whole documents are quite time-consuming and boring. It not only requires plenty of time but also takes too much manpower. To solve this issue, training the machine to understand the whole document and summarize it for ease and comfort makes more sense and that motivates me to work on this topic.

Documents can be of any type.A whole book can be a document to summarize and one chapter of that book can also be a different document to generate a summary. Not only a book and chapter of a book, a news article can also be one document, a research paper, a Reddit article can also be a document. More specifically, BBC News can also be a different document and CNN/Daily Mail can also be a different document.

While learning, machines also learn about the topic of a document and the writing style of the author. document length also matters. And these things have an impact on our generated summary.

1.2 Types of Summary

A summary is that one or two conclusive sentences by which we can get an idea and understanding of the whole text data. Basically we have two types of summary.

1.2.1 Abstractive

It may generate novel words. That means our summary may have new works which are not there in our text!!!

Example :

Source Text: Ram and Shyam took a taxi to attend the night party in the city. While in the city, Ram collapsed and was rushed to the hospital.

Abstractive Summary: Ram was hospitalized after attending a party with Shyam.

1.2.2 Extractive

It copies informative fragments from the input. That means the summary is totally made up of input texts.

Example :

Source Text: Ram and Shyam took a taxi to attend the night party in the city. While in the city, Ram collapsed and was rushed to the hospital.

Extractive Summary: Ram and Shyam attend party. Ram rushed hospital.

1.3 Benefits and Challenges

Benefits[9] :

- **Saves Time :** By generating automatic summaries, text summarization helps content editors save time and effort, which otherwise is invested in creating summaries of articles manually.

- **Instant Response :** It reduces the user's effort involved in exacting the relevant information. With automatic text summarization, the user can summarise an article in just a few seconds by using the software, thereby decreasing their reading time.
- **Increases Productivity Level :** Test Summarization enables the user to scan through the contents of a text for accurate, brief, and precise information. Therefore, the tool saves the user from the workload by reducing the size of the text and increasing the productivity level as the user can channel their energy to other critical things.
- **Ensures All Important Facts are Covered :** The human eye can miss crucial details; however, automatic software does not. What every reader requires is to be able to pick out what is beneficial to them from any piece of content. The automatic text summarization technique helps the user gather all the essential facts in a document with ease.

Challenges :

- **Efficiency :** We can not blindly trust a machine generated summary. It may have miss out some of the important points which human brain can identify but machine fails to detect.
- **Lengthy Documents :** ML models can not easily digest whole document at a time. It can summarize the small documents with few lines of code. But to summarize whole book or full chapter, ML model are way more behind.
- **Costly :** Huge Machine learning models requires up to the mark GPUs and TPUs, to train efficient model, which are too costly to afford.
- **Less numbers of Labeled Data :** Deep Learning models are data hungry. Fewer the data useless model we get. To make Good model it requires huge data.

1.4 Introduction to Deep Learning for summary

Well, when we heard Automatic text summarization somewhere it automatically gives us a click of Artificial Intelligence and machine learning/deep learning in our mind.

If we look at the problem from the AI's perspective then we are feeding texts and labels to the machine to learn weights and we are making the machine learn from the fed text and label so that in the future when the machine encounters a new text, it can easily predict the label for us and solve our problem.

1.5 Technology/Platform/Tools

Language: Python

Libraries: Pandas, Pytorch, Transformer, adapter-transformer, nltk

Platform: Google Colab, PARAM shavak(super computer)

Chapter 2

Literature Survey

In this chapter, I will give brief overview of some of the old school techniques which were used at the initial stage of research for text summarization and then, from it, we will move to the latest techniques based on neural network which gives us SOTA results for text summarization.

2.1 Overview of Text Summarization

Summarization is the task of converting a piece of text to a shorter version with preserving key informational elements and the meaning of content. Since manual text summarization is a time expensive and generally laborious task, the automatizing of the task is gaining increasing popularity and therefore constitutes a strong motivation for academic research.

At the initial level of research, This problem is treated as classification problem in which machine learning models are trained to classify whether to keep the word from the document in summary or not[10].

Further more, Another approach was extractive summarization, in which score of the words and sentences are calculated and based on that score words and sentences are extracted from the document and stitched together as summary. For Lex-ranking/scoring different algorithms were used[11].

In continuation, Term Frequency-Inverse Document Frequency (TF-IDF) Method the TF/IDF score of sentence was also introduced[12]. In this, the sentence frequency i.e. the number of sentences in the document that contain a particular term is calculated. Then similarity is calculated between query and these sentence vectors and highest scoring sentences are included in summary.

After the immense growth in the deep learning, researchers are diverting towards neural network based methods for text summarization.[13]proposes an extractive text summarization approach for factual reports using a deep learning model, exploring various features to improve the set of sentences selected for the summary.

After the growth in the deep learning techniques and techniques to handle sequential data, RNN architecture for [14], Attention mechanism and latest Transformer architecture[4], has totally changed the picture of the text summarization. All of them are mentioned in upcoming passages.

2.2 Deep Learning based techniques for Abstractive Summarization

By the time, things got updated and the same goes with deep learning approaches for summarization. We start from scratch and end with the current state-of-the-art model.

2.2.1 Seq2Seq

Sequence to Sequence (often abbreviated to seq2seq) models are a special class of Recurrent Neural Network architectures that we typically use (but not restricted) to solve complex Language problems like Machine Translation, Question Answering, creating Chat bots, Text Summarization, etc.

In basic Seq2Seq, we are sending Sequence as input and getting Sequence as output. In our problem, as you can see in fig 2.1 sequences are of sentences. We pass our text document to the encoder of the model as input and the decoder of the model will generate a sequence of text as a summary. In between, at the end of an encoder and before the decoder, it will generate one context vector which keeps the context of the input document and utilizes it while generating a summary.

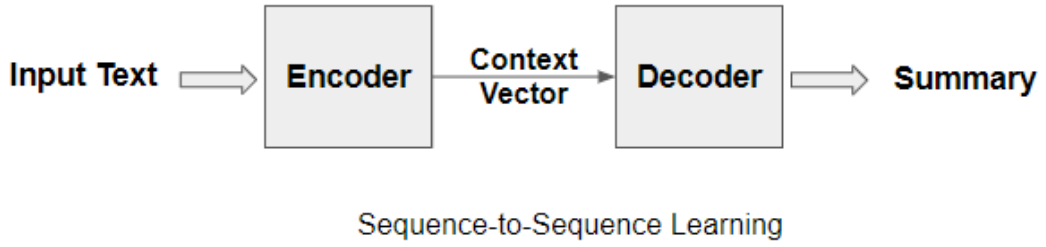

FIGURE 2.1: Overview of Encoder-Decoder for Seq2seq model for text summarization.

Encoder and decoder can be of any type-RNN, LSTM, GRU as shown in fig 2.2

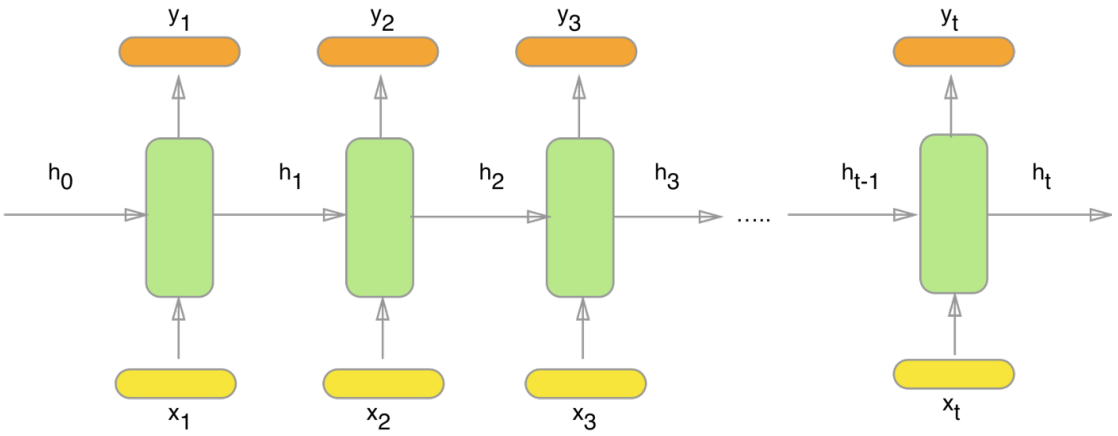

FIGURE 2.2: Sequential model[1]

The only difference between them is, LSTM and GRU overcome the problems of RNN.

A quick overview of the RNN, LSTM, and GRU.

RNN :

[14]A Recurrent Neural Network, as shown in fig 2.3 ,was introduced to handle the data which are in sequence and where order matters, for instance, text data. The order of the words in the text is the main thing it has. Change in order completely changes the meaning of the text.

RNN takes only one token(word) at a time, does processing on that, and tries to remember the information provided by it. For the processing, it uses a simple feed forward neural network. For the next word, it utilizes the hidden state of the previous word to process the new word and go likewise till it covers all the tokens(words).

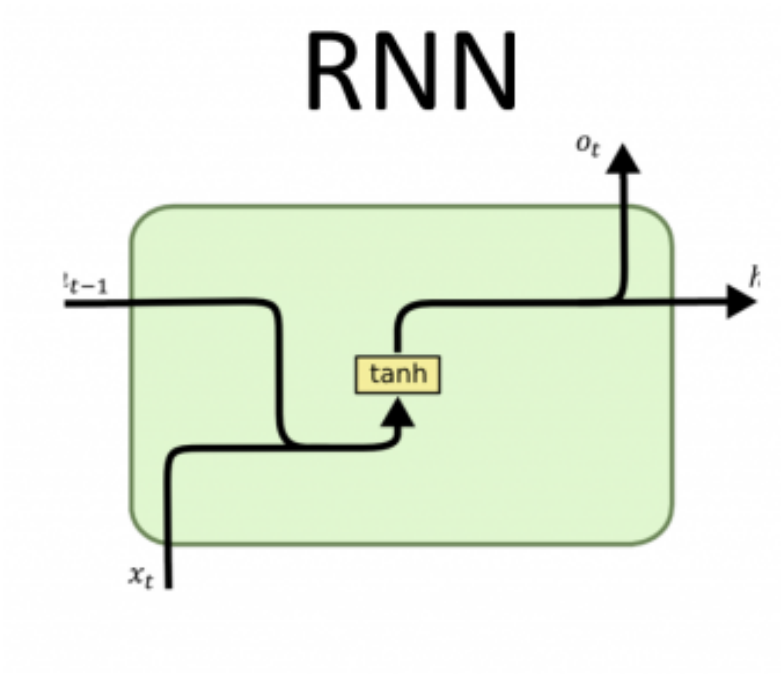

FIGURE 2.3: RNN[2]

The problem with this is, it is storing information on all the words, important as well as unnecessary ones. Not only that, for the longer sequences it becomes difficult for the RNN to keep the information from the first timestamp to the last one. If we pass a larger document to the RNN, it will forget some information from the initial steps. To overcome this issue LSTM and GRU were introduced!

LSTM and GRU :

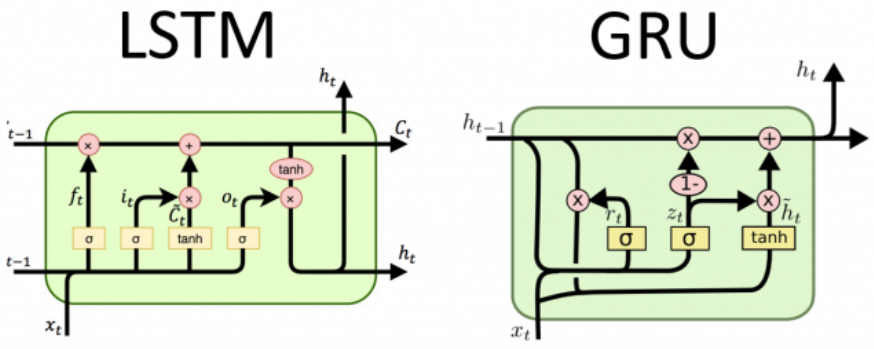

FIGURE 2.4: LSTM and GRU[2]

[15, 16]Both of them, LSTM and GRU as shown in fig 2.4 , work the same as RNN but the only difference is, a way of generating a hidden state for the next token(word). Instead of keeping all the information LSTM and GRU sometimes leave or neglect some

of the information which is not needed or the model finds that unnecessary. Different gates were used to filter out the necessity of the words. Based on the function, tanh, and sigmoid, it will decide whether to keep the information or ignore it.

Limitations with Seq2Seq:

- **Very slow and time-consuming** as we are using RNN / LSTM / GRU which is taking only one token(word) at a time.
- **Vanishing gradient problem** for the larger document size.
  [17]The vanishing gradients problem is one example of the unstable behaviour of a multi layer neural network. Networks are unable to back propagate the gradient information to the input layers of the model.

  In a multi-layer network, gradients for deeper layers are calculated as products of many gradients (of activation functions). When those gradients are small or zero, they will easily vanish. (On the other hand, when they're bigger than 1, it will possibly explode.) So it becomes very hard to calculate and update.
- **loss of information for large documents** as in seq2seq model all the information is stored in the form of context vector and that context vector is created once we iterate through all the text in the encoder side. At decoder, we are passing the context vector and the label while training.

2.2.2 Seq2Seq + Attention

[18]While working with the seq2seq model, we are creating only one vector which is a context vector that has the information of the whole document. In this case, what if while handling the input text we want to focus more on some of the input words which are relevant to the sentence while making predictions. Instead of keeping only one vector to represent the whole document what if we introduce another vector that has the information about which word to focus more!

A new attention mechanism is introduced. Attention is the cognitive process of selectively concentrating on one or a few things while ignoring others. The main idea of Attention is to not be dependent only on one vector, instead, pay attention to the specific input vectors from the input sequences based on the attention weights.

These attention weights are learned during the training. The value of the attention weights are relying on between 0 and 1 and a total sum of all the attention weights is always 1.

This attention is of 2 types[3] :

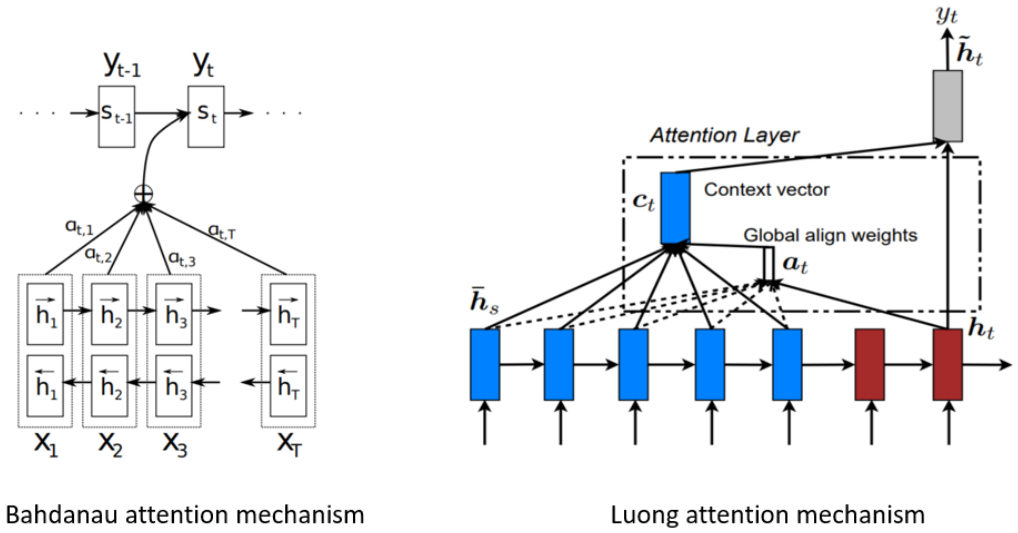

FIGURE 2.5: Type of Attention[3]

1. **Bahdanau attention mechanism**

   Bahdanau[19]proposed an attention mechanism that learns to align and translate jointly, as shown in fig above. It is also known as Additive attention as it performs a linear combination of encoder states and the decoder states.

   All hidden states of the encoder(forward and backward) and the decoder are used to generate the context vector, unlike how just the last encoder hidden state is used in seq2seq without attention.

   The attention mechanism aligns the input and output sequences, with an alignment score parameterized by a feed-forward network. It helps to pay attention to the most relevant information in the source sequence.

   The model predicts a target word based on the context vectors associated with the source position and the previously generated target words.

   Attention layer consist of :

   (a) **Alignment layer**

   The alignment score maps how well the inputs around position "j" and the output at position "i" match. The score is based on the previous decoder's hidden state, just before predicting the target word and the hidden state, h of the input sentence. The decoder decides which part of the source sentence it needs to pay attention to, instead of having encoder encode all the information of the source sentence into a fixed-length vector.

   (b) **Attention weights**

   We apply a softmax activation function to the alignment scores to obtain the attention weights. Softmax activation function will get the probabilities whose sum will be equal to 1, This will help to represent the weight of influence for each of the input sequence. Higher the attention weight of the input sequence, the higher will be its influence on predicting the target word.

   (c) **Context vector**

   The context vector is used to compute the final output of the decoder. The context vector is the weighted sum of attention weights and the encoder hidden states which maps to the input sentence.

2. **Luong attention mechanism**

   Luong's attention [20] is also referred to as Multiplicative attention. It reduces encoder states and decoder state into attention scores by simple matrix multiplications. Simple matrix multiplication makes it is faster and more space-efficient. fig is shown above. Luong suggested two types of attention mechanism based on where the attention is placed in the source sequence

   (a) **Global attention :** where attention is placed on all source positions as shown in fig below .

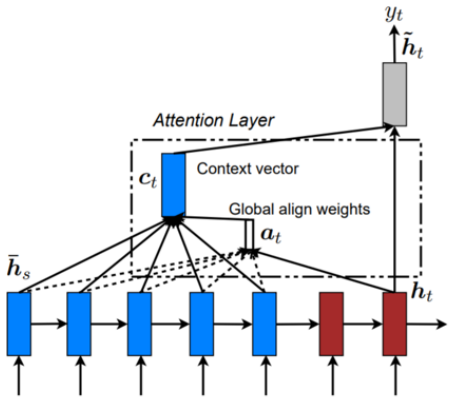

FIGURE 2.6: Global Attention[3]

   (b) **Local attention :** where attention is placed only on a small subset of the source positions per target word. you can see in fig below

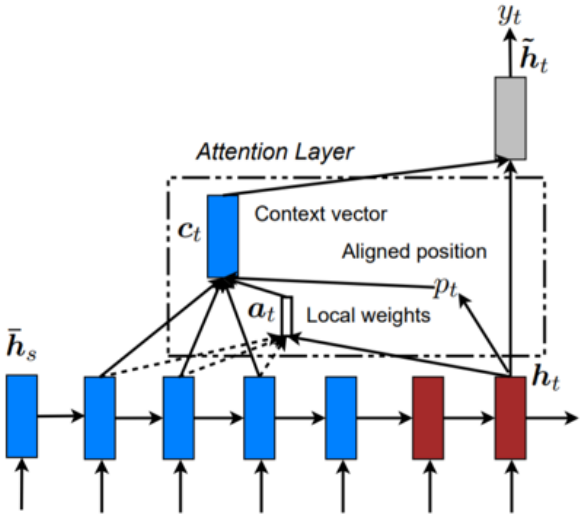

FIGURE 2.7: Local Attention[3]

All in all, these were the two basic types of attention and that's how the Attention mechanism works.

By introducing the attention mechanism we can overcome the problem of vanishing gradient and loss of information. However, Even after utilizing attention, we face the problem with parallel computation as we are dealing with the sequential data. We can only pass one token(word) at a time even after having the best computational power!!!!

2.2.3 Transformer

[4]Well, Transformer is the new architecture like RNN or CNN to overcome the problems of the seq2seq + Attention that is parallel computing. The main idea behind this new architecture is to solve the issue of parallel computing where we can effectively utilize our resources, high-class GPUs and TPUs, for the training of the model.

Architecture :

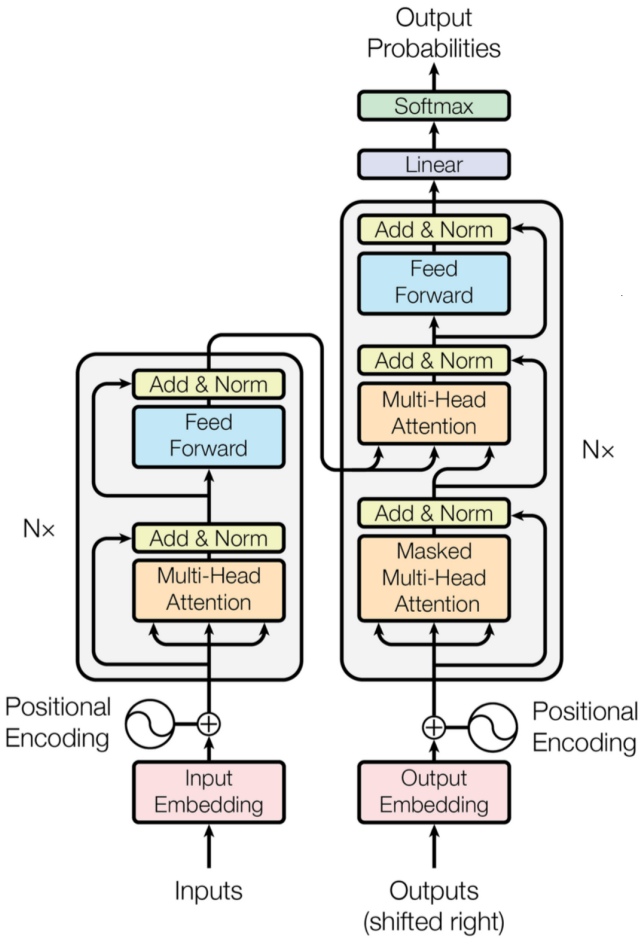

FIGURE 2.8: Architecture of Transformer model[4]

In the fig. 2.5, The Encoder is on the left and the Decoder is on the right. Both Encoder and Decoder are composed of modules that can be stacked on top of each other multiple times, which is described by **Nx** in the figure. We see that the modules consist mainly of Multi-Head Attention and Feed Forward layers. The inputs and outputs (target sentences) are first embedded into an n-dimensional space since we cannot use strings directly.

One slight but important part of the model is the positional encoding of the different words. Since we have no recurrent networks that can remember how sequences are fed into a model, we need to somehow give every word/part in our sequence a relative position since a sequence depends on the order of its elements. These positions are added to the embedded representation (n-dimensional vector) of each word.

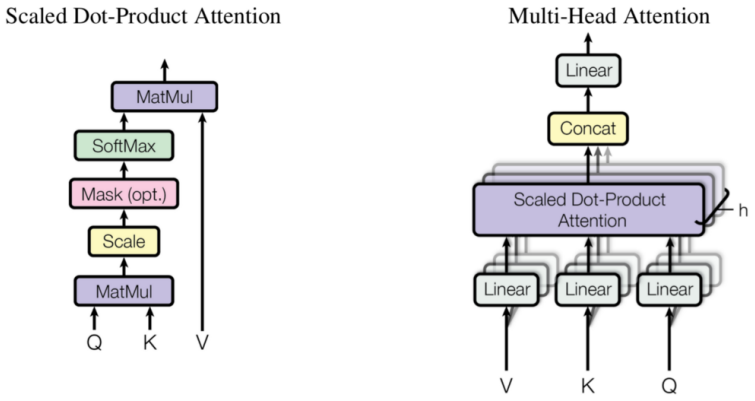

FIGURE 2.9: (left)Scaled Dot-Product Attention. (right)Multi-Head attention consists of several attention layer running parallel.[4]

Q = A matrix that contains the query (vector representation of one word)

K = All keys (vector representations of all the words in the sequence)

V = Values (vector representations of all the words in the sequence)

On the right side, Multi-headed attention is given which consists of multiple attention layers which run parallel and final attention is the average of all the heads. Normally, 8 heads are there in the Multi-head attention. The transformer uses self-attention, fig 2.7

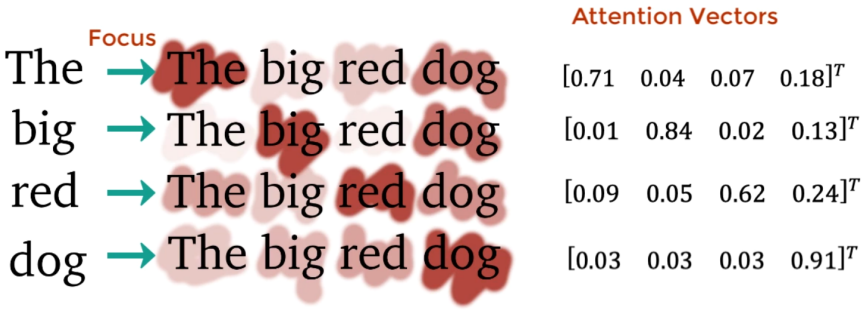

FIGURE 2.10: Example of Self-attention[5]

, where the attention of the sentence is calculated by itself, the same sentence. As we are feeding input in a parallel manner, that sequential knowledge got lost but self-attention

can solve that issue. In self-attention, a word learns how much attention to pay to the remaining words and what word to focus on next within the sequence.

In the encoder side, complete self-attention is used in which all the words are calculating attention on the remaining words in a sentence. wherein the decoder side partial self-Attention is used in which attention is calculated only on the previously generated tokens(words).

Based on this architecture many SOTA models are trained using self-supervised objectives and achieve commendable results.

2.2.4 Pre-trained Models

[21]Transformer models with some self-supervised pre-training have shown to be an impactful framework for producing general language learning, achieving SOTA (state-of-the-art performance) when fine-tuned on a wide variety of language tasks. In additional work, the self-supervised objectives used in pre-training have been somewhat connected with the downstream application in favor of generality. Better performance could be achieved if the self-supervised objective more closely mirrored the final task.

Having sophisticated pre-training objectives and huge model parameters, large-scale PTMs can effectively capture knowledge from massive labeled and unlabeled data. By storing knowledge into huge parameters and fine-tuning on specific tasks, the rich knowledge implicitly encoded in huge parameters can benefit a variety of downstream tasks, which has been extensively demonstrated via experimental verification and empirical analysis.

Transfer learning's effectiveness comes from pre-training a model on abundantly-available unlabeled text data with a self-supervised task, such as language modeling or filling in missing words. After that, the model can be fine-tuned on smaller labeled datasets, often resulting in (far) better performance than training on the labeled data alone.

Many PTMs are created by the different AI research laboratories around the world and that huge-models are used as per our task and requirement by fine-tuning nowadays by training adapters.

Here are some language PTMs: Bart, GPT, Bert, T5, etc.

From all the PTMs I've worked with T5 and PEGASUS as a part of my research work. will see both of them one by one.

2.2.4.1 T5

[6]T5 is a Text to Text Transfer Transformer. T5 model, pre-trained on C4, achieves state-of-the-art results on many NLP benchmarks while being flexible enough to be fine-tuned to a variety of important downstream tasks.

T5, re framing all NLP tasks into a unified text-to-text format where the input and output are always text strings. For a task where the model is fed some text for context or conditioning and is then asked to produce some output text, in contrast to other PTM that can only output either a class label or a span of the input or something else.

Tasks :

Its text-to-text framework allows us to use the same model, loss function, and hyper parameters on any NLP task. It includes machine translation, document summarization, question answering, and classification tasks (e.g., sentiment analysis) as shown in figure. For some specific tasks, all we need to do is to add a prefix before the input text. As an example, to ask the model to translate the sentence "That is good." from English to German, the model would be fed the sequence "translate English to German: That is good." and would be trained to output "Das ist gut."[paper]. Among all of the tasks, I have focused on the Text Summarization task. For it, the prefix "summarize:" is used and all we need to do is add it at the beginning of the text so that model can have an idea of what task to perform.

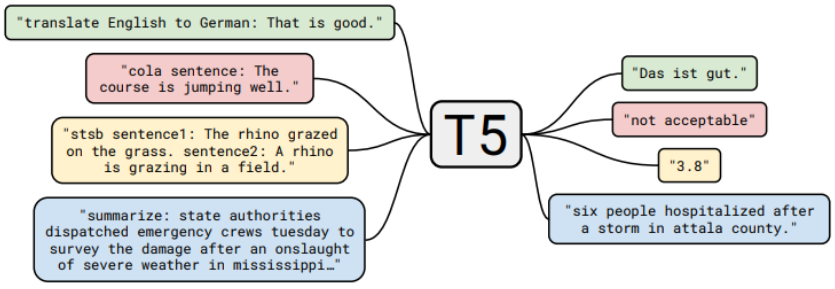

FIGURE 2.11: All the tasks the T5 models can perform. All the tasks are in text-to-text format. It includes-translation, question-answering, summarization, classification. In each task we are feeding text as an input and model will generate text as output[6].

Architecture:

T5 model has used the Transformer base architecture with the self-attention within. Which uses an encoder-decoder-based approach to handle input and output and that is intended for sequence to sequence problems. Model not only has a single transformer in architecture but it has a stack of Transformers. In T5, a different variant of the model has a different stack size. For the small model, T5-small, the stack size is 5 whereas, for the base model, T5-base, the stack size is 11. The performance of the model changes as

the number of stack size increases or decreases because the number of parameters also change.

If we look deeper into the architecture of the model. Before looking at the encoder, first, it takes sequences of tokens as input and maps them with the sequence embedding, and then passes it to the encoder. It uses simple positional embedding to capture the positional information.

The Encoder has the stack of blocks that were mentioned earlier. Block contains two basic modules 1.) Self-attention layer and 2.) Feed-forward network. At the end of each component, layer normalization is applied for re-scaling the activation. Within the feed-forward network, dropout is applied for reducing the over fitting of the artificial neural networks by preventing complex co-adaptations on training data.

Decoder follows the same architecture as encoder but the only difference is in the attention. In the decoder, after every self-attention layer, a standard attention mechanism is there which attends to the output of the encoder. Self-attention of the decoder uses the form of the auto regressive or causal self-attention, which only allows the model to attend to past outputs. The final output of the decoder block is fed to the dense layer with the soft-max output.

Pre-training Objective :

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style Devlin et al. (2018)	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
MASS-style Song et al. (2019)	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

Table 3: Examples of inputs and targets produced by some of the unsupervised objectives we consider applied to the input text "Thank you for inviting me to your party last week ." Note that all of our objectives process *tokenized* text. For this particular sentence, all words were mapped to a single token by our vocabulary. We write *(original text)* as a target to denote that the model is tasked with reconstructing the entire input text. <M> denotes a shared mask token and <X>, <Y>, and <Z> denote sentinel tokens that are assigned unique token IDs. The BERT-style objective (second row) includes a corruption where some tokens are replaced by a random token ID; we show this via the greyed-out word apple.

FIGURE 2.12: Pre-training Objectives[6]

Pre-training objectives decide the behavior of the model and more similar the pre-training objective to the downstream task more effective results model generates. T5

model is pre-trained for the different tasks and with the mixers of the pre-training objectives.

T5 model has the simple denoising objectives for pre-training. some of the objectives are mentioned in the table.

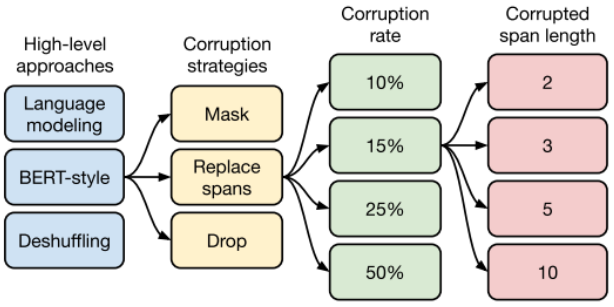

FIGURE 2.13: Pre-training[6]

For the pre-training T5 model used a C4 dataset.

2.2.4.2 PEGASUS

[7]PEGASUS - Pre-training with Extracted Gap-Sentence for Abstractive Summarization. This model was introduced by the research team at google in 2020. PEGASUS is the Current SOTA (State Of The Art) pre-trained model for Abstractive Text Summarization. The uniqueness about this model which makes it stand out and be the SOTA model is its pre-training objective. I will discuss the model in a deep in upcoming passages.

Architecture of the model :

This model is based on transformer architecture which uses an encoder-decoder with a Self-attention mechanism. Positional encoding are also applied. Multi-headed self-attention is used which calculates the self-attention for the given document multiple times and selects the best out of them.

Pre-training Objective :

The main focus behind this model is on its self-supervised pre-training objective. GSG(Gap Sentence Generation) is used as a pre-training objective for the pre-training of the model.

The researcher also tried different pre-training objectives like MLM. But the GSG fits well for the up-to-the-mark results.

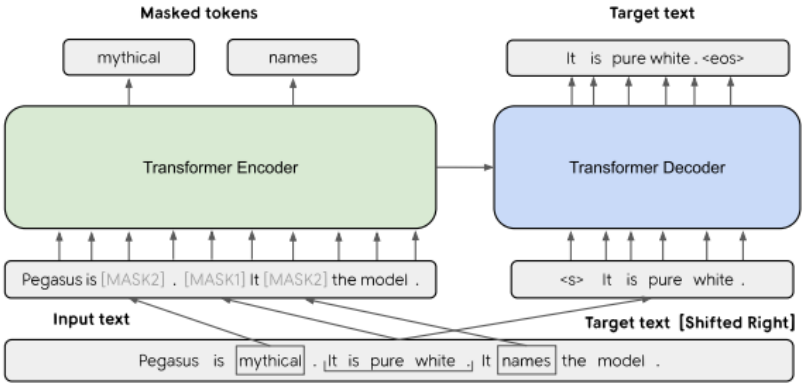

FIGURE 2.14: Pre-training in PEGASUS[7].

To decide what sentences to mask they proposed 3 ideas:

1. Random Sentences:

   In this, the model will randomly mask m sentences from the document and treat them as a label in the self-supervised learning technique whereas the remaining sentences, unmasked, work as a reference to generate that masked sentences.

2. Lead Sentences :

   Sometimes, a Document can have some Leading sentences, one that is at the beginning of the document, as a summary. We can see this kind of summary in news articles often. With having that idea in mind, the Model is also trained by masking m initial sentences from the documents for making the model capable of generating that initial(leading) sentences.

3. Principal Sentences :

   This is quite an interesting idea. As a summary of any document that has a significant sentence from the document, the model is trained by masking the important sentences from the document. In that way, the model learns to generate whole sentences which are important to any given text document. To identify how important the sentence is, a ROUGE score is used. It takes selecting one sentence and finding the ROUGE score with respect to the remaining sentences from the entire document.

Principal sentences were also selected in two manners:

(a) Independently :

In this manner, simply select the top m important sentence from the document and use them for masking.

(b) Sequentially :

In this technique, to identify the next important sentence from the document, keep the previous important sentence in the sequence and in this manner select the sequence with m sentences that have the highest ROUGE score.

From the given 3 ideas to select sentences for masking, Principal Sentences, the last one, did well and finalize for masking.

Now, to decide the value of m, the number of sentences to mask, a researcher did experiments and concluded to select 30

PEGASUS-base model and PEGASUS-large model were proposed. The difference between them is in number of parameters.Base model has fewer parameters than the Large model which have **568M** parameters!!!

Not only this, the Model did well for the zero and low-resource summarization. Starting from zero, it works well for only 10,100 and 1000 data only and produce unbelievable results.

All in all, currently, PEGASUS is truly State Of The Art model for abstractive summarization.

Chapter 3

Training Objectives and Data-sets

Data plays a very crucial role while working with Deep learning, without which nothing is possible. To make a machine learning model efficient we need huge data, the more data you feed to your model more effective it becomes.

DL Models are Data hungry and need a lot of data to train. The training models for all the individual documents don't make any sense. It is like wasting our resources and time. it's difficult to train a model with document size in million and to reach SOTA results.

As a solution, genius people come up with the new idea of training ML/DL models.

3.1 Pre-training

[21]For some specific task, let's say summarization, while training model we are passing whole data lets says 1Million for training. The problem with this is for each different data-set, for instance, Political-news, Sports-news, we need to train a model from the scratch! we can not utilize the previously trained model for a different type of data.

Instead of that, we do train the model on huge data for some common task and then utilize that huge model for any downstream task with less training we can achieve good results. We can also utilize a model created by someone else to solve a similar problem. Instead of building a model from scratch to solve a similar problem. In layman's terms, a model to recognize a car is utilized for truck recognition.

Training of model for some common task is called pre-training. It pursues a self-supervised learning approach where the machine learns from unlabeled data to fill in the blanks for missing pieces. The basic idea is the machine masks some of the sentences or words from the text and tries to learn those sentences or words from the remaining text.

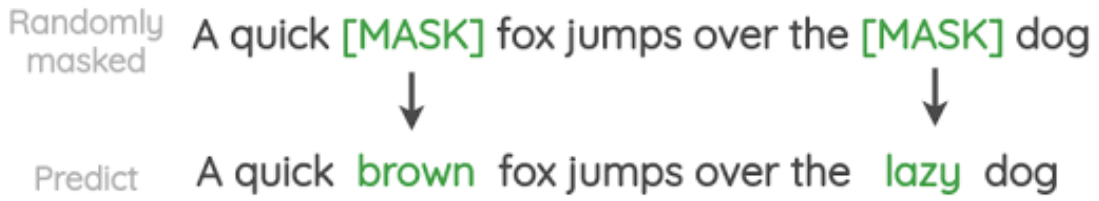

FIGURE 3.1: Masking of the text data for self-supervised training.[7]

Some of the masking techniques

1. MLM (Mask Language Modeling)
2. NSP (Next Sentence Prediction)
3. GSG (Gap Sentence Generation)

3.1.1 Pre-training datasets for Text summarization.

1. C4 (Colossal and Cleaned version of Common Crawl)[22]

   A colossal, cleaned version of Common Crawl's web crawl corpus. Based on Common Crawl dataset: "https://commoncrawl.org". It consists of text from 350M Web-pages (750GB).

2. HugeNews[23]

   A dataset of 1.5B articles (3.8TB) was collected from news and news-like websites from 2013-2019. A whitelist of domains ranging from high-quality news publishers to lower-quality sites such as high-school newspapers and blogs was curated and used to seed a web-crawler. Heuristics were used to identify news-like articles, and only the main article text was extracted as plain text.

3.2 Fine-tuning

It is a process of training an already pre-trained model furthermore according to our downstream task. This is supervised training in which our document is Text and summary is used as a label. In fine-tuning, all the parameters(weights) of the neural network get updated based on the downstream task dataset.

3.2.1 Some of the fine-tuning dataset for text Summarization:

1. XSum[24]

   It consists of 227k BBC articles from 2010 to 2017 covering a wide variety of subjects along with professionally written single-sentence summaries.

2. CNN/Daily-Mail[25]

   A dataset contains 93k articles from the CNN, and 220k articles from the Daily Mail newspapers. Both publishers supplement their articles with bullet point summaries.

3. NEWSROOM[26]

   It is a large dataset containing 1.3M article-summary pairs written by authors and editors in the newsrooms of 38 major publications between 1998 and 2017.

4. Gigaword[27]

   It contains 4M examples extracted from news articles (seven publishers) from the Gigaword corpus.

5. BillSum[28]

   It contains 23k US Congressional bills and human-written reference summaries from the 103rd-115Th (1993-2018) sessions of Congress.

3.3 Adapter training

[29]It is an alternative to Fine-tuning. The main idea behind this approach is to add some additional weights to the pre-trained model and learn those weights while training of adapter. An adapter can be task-specific and easy to utilize.

Training of adapter is the same as fine-tuning, it uses the same datasets as fine-tuning. The only thing required for adapter training is, a pre-trained model should have adapter support. It should be able to adapt adapter within.

Adapters are way more lighter than fine-tuning the model reason behind it is, in fine-tuning we are updating all the millions or billions of parameters whereas in adapter we are just updating the new additional weights. We are not changing the weights of the pre-trained model and that thing makes the adapter superior.

If I talk about results then there is no major difference in the results of pre-training and fine-tuning.

Here are some basic intuition and differences between adapter and fine-tuning All the DL models are just some linear equations that have parameters.

After Pre-training let's say equation is like,

$$F(x) = 3x_1 + 5x_2$$

In fine-tuning we are just updating the weights

$$F(x) = \mathbf{2}x_1 + \mathbf{7}x_2$$

In Adapter we are adding new weights

$$F(x) = 3x_1 + 5x_2 + \mathbf{10}$$

Chapter 4

Experiments

As part from my hunting for a pre-trained Deep learning models for text summarization, we did some experiments which are as follow:

All the experiments below were done on the super computer PARAM SHAVAK available at Dharmsinh Desai University under the observation of Prof. Siddharth P. Shah.

4.1 Fine-tuning

As a part of our first experiment, we took a pre-trained model and tried to fine-tune it, and observed the behavior of that model how it produces the results and how effective it is.

As a pre-trained model we selected T5-small and T5-base model and for a training data we make a choice of CNN-Daily-Mail's news dataset which has around 200k labeled training data and 13k validation data. We kept the batch size of 4 and trained model for 3 epochs.

For small model it took around 4 and half day for fine-tuning and base model took around 21 days!!!

From the numbers we observed that small model took less time, for the same configuration, compared to base model due to less number of parameters.

4.1.1 T5-small model

Model	T5-small
Dataset	CNN Daily-mail
Dataset config	3.0.0
Number of Training Data	287,113
Number of Validation Data	13,368
Training Batch size	4
Validation Batch size	4
Number of epochs	3

Table 4.1: Training

Global step	215,337
Train loss	1.8038359856349437
Train runtime	113.60 hr (4.74 Days)
Train samples per second	2.106
Train steps per second	0.527

Table 4.2: Trainer states

Length penalty	2.0
Max length	200
Min length	30
No repeat ngram size	3
Num beams	4

Table 4.3: Training config

Model	T5-small-fine-tuned
Dataset	CNN Dailymail
Dataset Configuration	3.0.0
Number of Testing data	11,490
Testing Batch size	16
Testing time	6hr 33min

Table 4.4: Testing

4.1.2 T5-base model

Model	T5-base
Dataset	CNN Dailymail
Dataset config	3.0.0
Number of Training Data	287,113
Number of Validation Data	13,368
Training Batch size	2
Validation Batch size	2
Number of epochs	3

Table 4.5: Training

Global step	215,337
Train loss	1.39039504933916
Train runtime	524.88 hr (21.87 Days)
Train samples per second	0.456
Train steps per second	0.228

Table 4.6: Trainer states

Length penalty	2.0
Max length	200
Min length	30
No repeat ngram size	3
Num beams	4

Table 4.7: Training config

Model	T5-base-fine-tuned
Dataset	CNN Dailymail
Dataset Configuration	3.0.0
Number of Testing data	11,490
Testing Batch size	16
Testing time	24hr

Table 4.8: Testing

4.2 Use of already Trained Adapter

In this second experiment, what we did, we took already trained adapter- Bart-large - from the adapter-hub and did experiments and observed the behavior of that model how it produces the results and how effective it is.

while playing with facebook/bart-large adapter for summarization we observed that for testing, as we haven't trained adapter, it took around 1day and 7hr for iterating through 11.5k CNN-daily-mail's test dataset for small model and for base model it surprisingly took only 7.5hr!!!

One thing we observed, which was surprising for us, was the testing time taken by both the models. Both the models were tested with the same configuration but base model took less time compared with the small model which is not an ideal situation.

4.2.1 Use of Bart adapter on T5-small -pre-trained model

Task	Summarization
Pre-trained adapter	@ukp/facebook-bartlarge-sum-cnn-dailymail-pfeiffer
DataSet	CNN Dailymail
Non-linearity	relu
Reduction factor	2
Hidden size	512
Adapter Architecture	pfeiffer

Table 4.9: Adapter configuration

Model	T5-small + adapter(Facebook/Bart-large)
Dataset	CNN Dailymail
Dataset Configuration	3.0.0
Number of Testing data	11,490
Testing Batch size	16
Testing time	1day 7hr 27min

Table 4.10: Testing

4.2.2 Use of Bart adapter on T5-base -pre-trained model

Task	Summarization
Pre-trained adapter	@ukp/facebookbartlarge sum cnn dailymail pfeiffer
DataSet	CNN Dailymail
Non-linearity	relu
Reduction factor	2
Hidden size	712
Adapter Architecture	pfeiffer

Table 4.11: Adapter configuration

Model	T5-base + adapter(Facebook/Bart-large)
Dataset	CNN Dailymail
Dataset Configuration	3.0.0
Number of Testing data	11,490
Testing Batch size	16
Testing time	7hr 29min

Table 4.12: Testing

4.3 Adapter Training

Well, in this experiment, We trained a new adapter and tried to connect it with the T5 model, and observed the behavior of that model how it produces the results and how effective it is.

For training we selected CNN-DailyMails's English dataset with 300k training examples and 13k validation data for the 3 epochs with the batch size of 32 and 16 for T5-small and T5-base model respectively.

Amazing thing we observed was the training times for both the New adapter training. For small model adapter training took only 5 days and 5days were taken by the based model. Not only this, Evaluation timing of both the model were also very less.

4.3.1 T5-small

Model	T5-small
Dataset	CNN Dailymail
Dataset config	3.0.0
Number of Training Data	287,113
Number of Validation Data	13,368
Training Batch size	32
Validation Batch size	32
Number of epochs	3

Table 4.13: Training

Adapter Architecture	pfeiffer
Non-linearity	relu
Reduction factor	16
Hidden size	512

Table 4.14: Adapter configuration

Global step	26,919
Train loss	1.8762163860650238
Train runtime	120.36 hr (5.02 Days)
Train samples per second	1.988
Train steps per second	0.062

Table 4.15: Trainer states

Eval-gen-len	74.2094
Eval-run time	1.47 hr
Eval-samples	13,368
Eval samples per second	2.522
Eval steps per second	0.079

TABLE 4.16: Evaluation states

Length penalty	2.0
Max length	200
Min length	30
No repeat ngram size	3
Num beams	4

TABLE 4.17: Training configuration

Model	T5-small + adapter(CNNDM)
Data-Set	CNN daily-mail
Dataset Configuration	3.0.0
Number of Testing data	11,490
Testing Batch size	32
Testing time -	1 h 17 m

TABLE 4.18: Testing

4.3.2 T5-base

Model	T5-base
Adapter Architecture	pfeiffer
Non-linearity	relu
Reduction factor	16
Hidden size	768
Dataset	CNN Dailymail
Dataset config	3.0.0
Number of Training Data	287,113
Number of Validation Data	13,368
Training Batch size	16
Validation Batch size	16
Number of epochs	3

TABLE 4.19: Training

Global step	53,835
Train loss	1.434505609679025
Train runtime	139.56 hr (5.82 Days)
Train samples per second	1.714
Train steps per second	0.107

TABLE 4.20: Trainer states

Eval-gen-len	72.6234
Eval-run time	4.57 hr
Eval-samples	13,368
Eval samples per second	0.812
Eval steps per second	0.051

TABLE 4.21: Evaluation states

Length penalty	2.0
Max length	200
Min length	30
No repeat ngram size	3
Num beams	4

TABLE 4.22: Training configuration

Model	T5-base + adapter(CNNDM)
Data-Set	CNN daily-mail
Dataset Configuration	3.0.0
Number of Testing data	11,490
Testing Batch size	16
Testing time	3 hr 50 min

TABLE 4.23: Testing:T5-base

4.4 SOTA: PEGASUS

SOTA- State Of The Art - PEGASUS model, While working with this model I did some interesting experiments. I'll discuss what I got from those experiments and the behavior of the model below.

4.4.1 Pre-trained only

While working with PEGASUS, the First experiment I did was try to utilize PEGASUS pre-trained model directly for a text summarization task without any kind of fine-tuning!

As the PEGASUS model is pre-trained with the goal of GSG (Gap Sentence Generation), I got that kind of result in the form of a summary. The model was directly picking up the important sentences from the document and gave us those sentences as a summary. However, that kind of summary is not up to the mark but the behavior was not negligible.

To fulfill my curiosity, I tried the same thing, the same experiment, but with a different language other than English. I used Hindi language and Gujarati language for testing and surprisingly I got the same behavior. It was also picking up the important sentences from the document, no matter what language it is, and giving us it as an output summary of the document.[30]

Language	**tokenizer**	**Vocab size**
English	Default(PEGASUS)	96,103
Gujarati	inltk-Guj	20,103
Hindi	inltk-Hindi	30,103

TABLE 4.24: Model : PEGASUS-large

4.4.2 Fine-tuned

While working with PEGASUS, the Second experiment I did was try to utilize the PEGASUS-XSum fine-tuned model for a text summarization task to see the behavior of the model!

4.4.2.1 On the English language

For English language, I selected the basic tokenizer of the PEGASUS model with 96k vocab size, Xsum dataset, and tested PEGASUS-XSum fine-tuned model on Xsum test data. Due to low resources, I tried only with the 100 data and the surprise was it gave me an amazing result. It did well and gave us the expected summary which was fluent and easy in reading.

4.4.2.2 On the Gujarati language

For Gujarati language, I selected the inltk library's Gujarati tokenizer with a vocab size of 20k. For the Gujarati dataset, I make a choice of the XLSum dataset's Gujarati data[31]. And after changing only the tokenizer of the PEGASUS-XSum model I tested my Gujarati data on the PEGASUS-XSum model.

As the model is pre-trained and fine-tuned on the English data it did well for English language but for Gujarati language, changing the tokenizer only didn't go well. It did not generate any Gujarati summary or any Hindi summary. A model failed to generate any summary as in fine-tuning model has learned only English embedding.

Chapter 5

Testing and Evaluation

For testing the effectiveness of our generated summary, I've used the ROUGE score as a measurement. It measures the accuracy of a language-based sequence when dealing with language. It calculates the score which indicates the similarity between our generated summary and labeled summary.

5.1 Rouge Score

ROUGE(Recall-Oriented Understudy for Gisting Evaluation)[32]

It is the count of numbers of N-grams between our model-generated summary and our reference summary. N-grams are a group of tokens/words.

1. **ROUGE-1** : measure the match rate of uni-grams between our model output and reference.
2. **ROUGE-2** : measure the match rate of bi-grams between our model output and reference.
3. **ROUGE-L** : measures the longest common sub sequence (LCS) between our model output and reference.

We are using the **F1-score** for these N-grams.

- **F1** :

$$\mathbf{2} * \frac{\mathbf{Precision} * \mathbf{Recall}}{\mathbf{Precision} + \mathbf{Recall}}$$

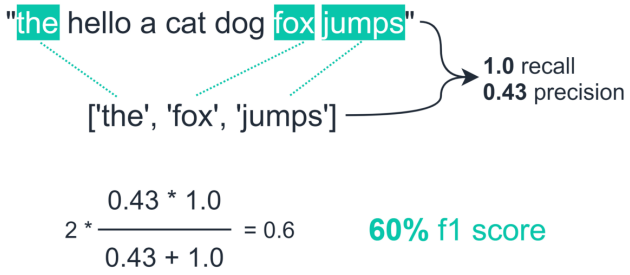

FIGURE 5.1: Example of F1 score[8]

- **Precision** :

$$\frac{\textbf{Number of n grams found in model and reference}}{\textbf{Number of n gram in model}}$$

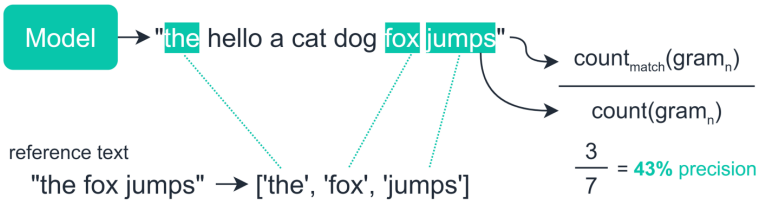

FIGURE 5.2: Example of Precision score[8]

- **Recall :**

$$\frac{\textbf{Number of n grams found in model and reference}}{\textbf{Number of n gram in reference}}$$

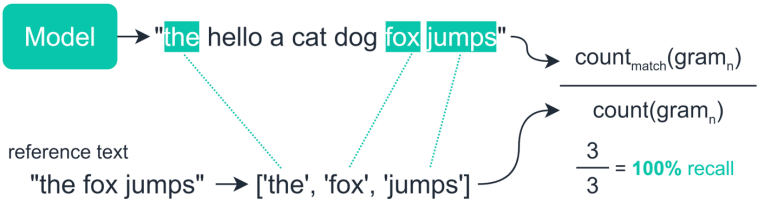

FIGURE 5.3: Example of Recall score[8]

5.2 Results

Well, here results of all the experiments were depicted in the form of table. Each value of the table indicate the Rouge score of that experiments. Rouge score is explained with example previously.

From the values we can observe that, small model has lesser value compared with the base model this is due to the less parameters in the small model. Basically, higher the number of parameters the good results we get.

After fine-tuning of the models we are getting the same results which were proposed by the authors of the model which was the positive sign for us!

Idea of utilizing already trained adapter for summarization didn't go in our favor it gave us the normal results which were way more far from our expectations.

Another observation we can see is between fine-tuned model and New adapter. we can see that both the model gave approximately same results not only this benefit of adapter over fine-tuning is it's lesser training time and light weight. as we are only updating the few new weights in adapter it took less time, while in fine-tuning we are updating all the parameters of the model so it took more time for training.

All in all, Newly trained adapter works well instead of utilizing already trained adapter and use of adapter instead of fine-tuning is also the good idea.

Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-LSum
t5-small	30.63	11.50	21.66	28.41
t5-base	11.50	1.93	8.81	10.54

Table 5.1: Already Trained adapter.

Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-LSum
Validation Results (Ours)				
t5-small	41.62	19.18	29.48	38.82
t5-base	43.81	20.96	31.17	40.97
Test Results (Ours)				
t5-small	41.05	18.73	29.13	38.25
t5-base	43.14	20.35	30.74	40.26
Test Results (From original paper)				
t5-small	41.12	19.56	-	38.35
t5-base	42.05	20.34	-	39.40

Table 5.2: Fine tuning.

Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-LSum
Validation Results				
t5-small	41.20	18.81	29.09	38.42
t5-base	43.43	20.64	30.86	40.59
Test Results				
t5-small	40.56	18.40	28.74	37.80
t5-base	42.86	20.07	30.44	39.97

Table 5.3: New Adapter.

Score	t5-small		t5-base	
	Adapter	Fine-tuned	Adapter	Fine-tuned
ROUGE-1	40.56	41.05	42.86	43.14
ROUGE-2	18.40	18.73	20.07	20.35
ROUGE-L	28.74	29.13	30.40	30.74
ROUGE-Lsum	37.80	38.25	39.97	40.26

Table 5.4: Comparison

Chapter 6

Conclusion and Future direction

6.1 Conclusion

All in all, to put in a nutshell, Transfer learning is at the sky in deep learning research as it is using self-supervised learning which does not require a label data for pre-training and Adapter are the alternative of the fine-tuning. A model, combination of both pre-training with task related objective and fine-tuning on task specific dataset, PEGASUS, can give you the SOTA result for Abstractive text Summarization.

6.2 Future Direction

Pegasus model does not have adapter support right now, We are looking forward to contribute and training of a light-weighted adapter for PEGASUS model.

Bibliography

[1] https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7, 2021.

[2] http://dprogrammer.org/rnn-lstm-gru, 2021.

[3] https://towardsdatascience.com/sequence-2-sequence-model-with-attention-mechanism-9e9ca2a613a, 2020.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[5] https://www.youtube.com/watch?v=TQQlZhbC5ps, 2021.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[7] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.

[8] https://towardsdatascience.com/the-ultimate-performance-metric-in-nlp-111df6c64460, 2021.

[9] https://www.impelsys.com, 2021.

[10] Yogan Jaya Kumar, Fong Kang, Ong Sing Goh, and Atif Khan. *Text Summarization Based on Classification Using ANFIS*, pages 405–417. 03 2017.

[11] Makbule Ozsoy, Ferda Alpaslan, and Ilyas Cicekli. Text summarization using latent semantic analysis. *J. Information Science*, 37:405–417, 08 2011.

[12] Hans Christian, Mikhael Agus, and Derwin Suhartono. Single document automatic text summarization using term frequency-inverse document frequency (tf-idf). *ComTech: Computer, Mathematics and Engineering Applications*, 7:285, 12 2016.

[13] Sukriti Verma and Vagisha Nidhi. Extractive summarization using deep learning. *CoRR*, abs/1708.04439, 2017.

[14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization, 2014. cite arxiv:1409.2329.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[16] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[17] https://analyticsindiamag.com/addressing-the-vanishing-gradient-problem-a-guide-for-beginners/, 2021.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[19] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[20] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[21] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation. *CoRR*, abs/1905.03197, 2019.

[22] Jesse Dodge, Maarten Sap, Ana Marasovic, William Agnew, Gabriel Ilharco, Dirk Groeneveld, and Matt Gardner. Documenting the english colossal clean crawled corpus. *CoRR*, abs/2104.08758, 2021.

[23] Alexander R. Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R. Radev. Multi-news: a large-scale multi-document summarization dataset and abstractive hierarchical model. *CoRR*, abs/1906.01749, 2019.

[24] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *CoRR*, abs/1808.08745, 2018.

[25] Danqi Chen, Jason Bolton, and Christopher D. Manning. A thorough examination of the cnn/daily mail reading comprehension task. *CoRR*, abs/1606.02858, 2016.

[26] Max Grusky, Mor Naaman, and Yoav Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. *CoRR*, abs/1804.11283, 2018.

[27] Tosin P. Adewumi, Foteini Liwicki, and Marcus Liwicki. Corpora compared: The case of the swedish gigaword & wikipedia corpora. *CoRR*, abs/2011.03281, 2020.

[28] Anastassia Kornilova and Vlad Eidelman. Billsum: A corpus for automatic summarization of US legislation. *CoRR*, abs/1910.00523, 2019.

[29] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulic, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers. *CoRR*, abs/2007.07779, 2020.

[30] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *CoRR*, cs.CL/0205028, 2002.

[31] Tahmid Hasan, Abhik Bhattacharjee, Md Saiful Islam, Kazi Samin, Yuan-Fang Li, Yong-Bin Kang, M. Sohel Rahman, and Rifat Shahriyar. Xl-sum: Large-scale multilingual abstractive summarization for 44 languages. *CoRR*, abs/2106.13822, 2021.

[32] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. page 10, 01 2004.