DEEP NATURAL LANGUAGE GENERATION USING BERT FOR SUMMARIZATION

by

Sourav Roy Choudhury

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2020

Approved by:

Dr. Samira Shaikh

Dr. Sara Levens

Dr. Tiffany Gallicano

ABSTRACT

SOURAV ROY CHOUDHURY. Deep Natural Language Generation using BERT for summarization. (Under the direction of DR. SAMIRA SHAIKH)

Summarization is one of the core facets of Natural Language Processing. Text summarization is the task of producing a concise and fluent summary while holding the most essential or salient part of the content and preserving the original meaning. The main aim of abstractive summarization is to generate concise version of original text while keeping intact the meaning. Since manual text summarization is a time expensive and generally a laborious task, the automatization of the task has gained immense popularity and therefore constitutes a strong motivation for academic research.

There has been some prime application of text summarization in current day such as news summarization, opinion summarization and headline generation to name a few. We will be looking into two main summarization techniques used in current day NLP tasks; extractive and abstractive. Extractive summarization generates summary by selecting salient sentences or phrases from the source text, while abstractive methods paraphrase and restructure sentences to compose the summary.Our main focus here would be on abstractive summarization as it is more flexible and can generate more diverse summaries. BERT (Bidirectional Encoder Representation from Transformers) is primarily a transformer based architecture which has been able to overcome the limitations of Recurrent Neural Networks (RNN) as long term dependencies. In this work we use a unique document level encoder-decoder based on BERT which works on a two stage process. For the first stage we use encoders to encode the input sequence into context-rich representations, for the decoder we use a transformer based decoder to generate a output sequence. must have an abstract.

DEDICATION

This work is dedicated to my father Chandi Roy Choudhury, my mother Gita Roy Choudhury for the constant strength and motivation they have provided throughout my masters.

ACKNOWLEDGEMENTS

I would like to gratefully acknowledge various people who have been journeyed with me in recent years as I have worked in this thesis.Throughout the struggles of my thesis they have been a constant source of joy. Thank you.

I would like to thank my advisor, Dr. Samira Shaikh, for giving me an opportunity to work with her. My work would not have been possible without her constant guidance, valuable input and feedback at each and every step. I would also like to take this opportunity to thank Sumanta Bhattacharyya, PhD student at UNC Charlotte, for his excellent suggestions and valuable feedback throughout my thesis and Rahul Patel, Masters student at UNC Charlotte who has been supportive through out my thesis journey.

I would also like to express my gratitude to my committee members, Dr. Sara Levens and Dr. Tiffany Gallicano for showing interest in my research work and accepting to join my thesis committee as panel members. They played a vital role instructing my thesis by providing ideas and feedback that guided me towards accomplishing my goal. My sincere appreciation to the University of North Carolina at Charlotte for accepting me as a master student and for providing necessary infrastructure and support to successfully complete my masterâs degree with thesis. I am very glad to work on one of the most interesting research topics and complete my masterâs degree with a wonderful research experience.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

In this present era of big data, retrieval of relevant information from humongous amount of text documents has posed out to be a challenging task.The unprecedented growth and rise of blogs, news articles and other documents has had a major impact to it's explosive growth. Automatic text summarization provides an effective solution to this problem using varied approaches.

The sole aim of summarization is to produce shortened and condensed summaries of large text documents. Short summaries enable the text to be retrieved, processed and digested effectively and efficiently.

1.1 Motivation

Text Summarization is considered to be a complex task in modern day Natural Language Processing. It has to produce a concise version of text while preserving the meaning and key ideas of the original source. Summarization involves several aspects of semantic and cognitive processing. The motive behind extractive summarization to construct summaries is by extracting the most vital or crucial points from original text. If one has to summarize legal documents it is advised to use extractive approach and not abstractive to avoid any interpretation. Abstractive summarization on the other hand is used for multi-document summarization of news articles as extraction might give summaries which are overly verbose or biased for certain sources. Traditional NLP techniques for summarization were heavily dependent on TF (term frequency), IDF (inverse document frequency) or cosine similarity which are frequency based approaches which try measure the importance of each sentence and their relationships with each other rather than utilising the context.

Abstractive summarization on the other hand is understanding the context of the text and generating summaries on the basis of the context. There has been much progress in this since the arrival of modern NLP methods such as neural word embedding, word2vec, glove and with the rise of Deep Learning these modern approaches have been the state of the art. There has been much success using Deep Learning approaches such as RNN(Recurrent Neural Networks) and LSTM(Long short term memory) and then with the use of Transformer based architectures [3] [4] [5].

1.2 Objective

There has been work on neural sequence to sequence framework [6] for generating abstractive summaries. Neutral networks based on sequence to sequence encoder-decoder models used attention mechanism to generate robust summaries that had high ROUGE scores but these models have been most successful in summarizing short sentences to generate even shorter summaries. Apart from this most of the approaches use a left context only decoder thus they do not have the entire context when predicting words. There has been prior work on applying abstractive summarization model on the CNN/daily mail dataset but most of the results generate unnatural summaries consisting of repetitive phrases. We found a two stage decoding process along with the use of BERT (Pretrained language models) to generate significantly better results.

1.3 Contribution

To address the problem we use a computationally efficient Pre-trained Natural Language Generation model. Using a Supervised approach for Abstractive Text Summarization using a neural attentive sequence to sequence framework. There are two parts of this framework, a neural network for the encoder and another neural network for the decoder. The aim of the model being maximizing the probability of generating correct target sequences. Precisely the main contributions of the thesis are:

1. We generate a natural language generation model based on BERT. BERT [7] has not been an immediate choice for most language generation models since it is primarily successful in text classification, question answering and Named Entity Recognition. We make use of the BERT model in the encoder and decoder process and the model can be trained end to end without any custom feature extraction.

2. We use a two stage decoder process where the model uses both left-right context for the decoder and thus has complete context while predicting the word.

3. We conduct experiments on CNN/Daily Mail dataset, where the model is evaluated using ROGUE-1, ROGUE-2 and ROGUE-L metrics.

1.4 Real World use of Automatic Text Summarization

In present day I believe automatic summarization finds place in most enterprise domains. There has been an ever going problem of information overload and automatic summarization can help by condensing large informative texts to smaller pieces of information. Another major impact sector can be using it for search queries for search engine optimization, multi document summarization can be a great way of understanding and analyzing dozens of search results and shared themes to evaluate the most important points.

Another major domain can financial research, investment banking companies where large chunks of informative texts are analysed on daily basis to make efficient decision making. It is always more beneficial to condense the financial reports to just the most salient points to save up on time.

With the rise of more and more online content like blogs, white-papers, e-books and etc summarization can lead to efficiently reuse the old content as reference in newer contents. Tele-health supply chains across the world have already started using summarization as a means of managing medical cases in a better way sine most

medical documents have turned digital in recent past.

Academic papers have always used a human constructed summary as the introduction and abstract. It is however overwhelming to read through all the papers during researching and literature survey, models that are able to compress the academic papers will be of major help. The same issue persists in technology where skimming through large chunks of documentation leads to major time kill to debug or understand the larger picture, summarised text can help in this regards and give users a well rounded understanding of the context.

CHAPTER 2: BACKGROUND

2.1 Deep Learning

Deep learning is a method of machine learning, which teaches algorithms how to do what naturally comes to humans. They can be very roughly and broadly be segregated into supervised,semi supervised and unsupervised learning based on annotated data.In the process of a fundamental education, a computer model discovers how to recognize pictures, text or sound explicitly. Deep learning models may attain cutting-edge precision and sometimes surpass human performance. Models are equipped by a large array of marked and multi-layered information and neural network architectures. The word "Deep" in Deep Learning comes from the use of multiple neural network layers in the network architecture. Deep Learning neural networks are interpreted in terms of universal approximation theorem or probabilistic inference.

2.2 Neural Network

Neural artificial networks can better be seen as weighted maps. In layman's terms a neural network is composed of layers of node and each node is designed to behave similarly to the neurons in our brain. The first layer of a neural net is called the input layer, which is followed by the hidden layers and the end layer of the network is called the output layer. Each node or neurons in the network perform some calculation which is passed onto the next or previous nodes in the network.

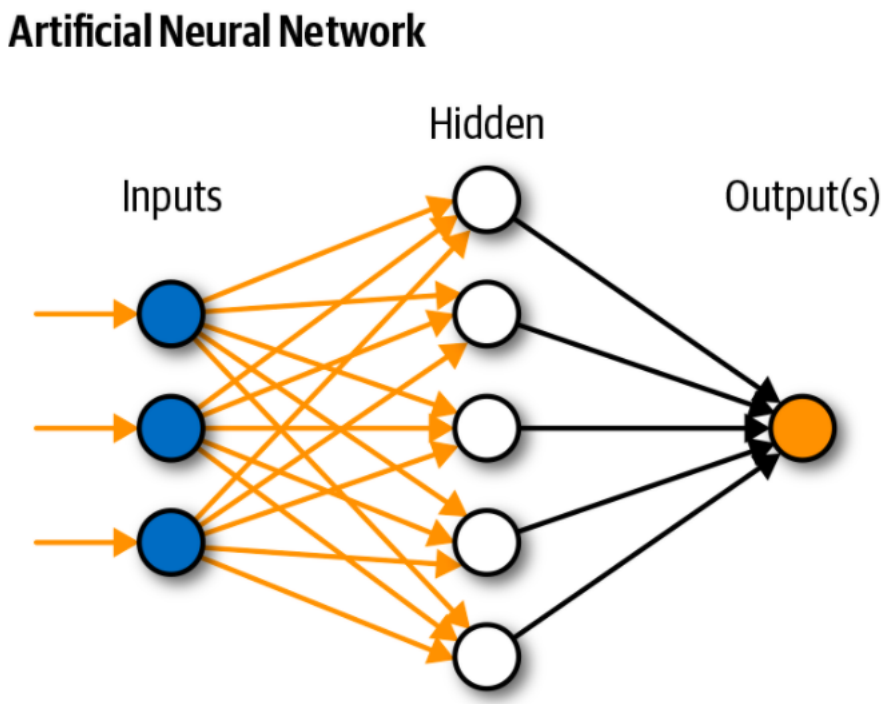

Figure 2.1: Basic Neural Network Architecture

As seen in the above figure the neurons in the network are capable of having connections to multiple preceding neurons, the weight of the synapse is the multiplying factor which imparts importance of that specific neuron in the model. Adjusting weights is one of primary ways in which neural networks are trained. On receiving the input to a neuron, it adds up each signal multiplied by it's corresponding weight to pass them through a mathematical function called activation function. It is through the activation function that the output value of the neuron is determined, which is then carried forward to the next layer and so on. It is through the activation functions that the neurons in the network talk or communicate to one another.

The process through which neurons determine which input values to be taken forward is called training. All neural nets are trained using something called as the cost function, which calculates the error in the network prediction as compared to it's true value.

- Input Layer: The Initial layers include the synthetic neurons that transmit

feedback from the outside environment (called units). This is where the real network training takes place, for otherwise it will be recalled.

- Output Layer: The output layers include units that react to the data fed into the process and also to know whether or not a function has been completed.
- Hidden Layer: The hidden layers between input layers and output layers are described. The only task of a secret layer is to transform the data into something useful that can be used by the output layer / unit.

CHAPTER 3: LANGUAGE MODELS

3.1 Why Language Models ?

Language is the most powerful medium of communication. In an archaic sense of the model language models learn to predict the probability of a sequence of words. In a machine translation example a bunch of words are taken from a source language and then these words are converted into a target language. There can be multiple potential translations that a system can give and one would like to compute the probability of each one of these translations to understand which one is the most accurate.

Word ordering:
p(the cat is small) > p(small the is cat)

In the above example we know that the probability of the first sentence being a correct translation is much higher than that of the second one and it is this ability to model the laws and directives of a language as a measure of probability allows NLP to automate a series of tasks such as text summarization, part-of speech tagging, information retrieval, machine translation etc.

3.2 Statistical Language Models

Statistical Language Models is the development of probabilistic models that are able to predict the next word in the sequence given the words that precede it. It is simple a probability distribution P(s) over all possible sentences. N-gram models are the most widely used Statistical Language models today. The goal is to compute the

probability of an upcoming word i.e

$$P(W_n|W_1, W_2, W_3...W_{n-1})$$

The chain rule is applied to compute the joint probability of a sentence but since there are too many possible outcomes so we will never see enough data to estimate these. Thus the hypothesis is simplified using Markov Assumption so now we can approximate each component in the product as

$$P(W_n|W_1, W_2, W_3...W_{n-1}) = P(W_n|W_{n-i}....W_{n-1})$$

In an n-gram model the probability of $P(w_1, ..., w_n)$ of observing the sentence $w_1, ...w_n$ is approximated as

$$P(w_1, ..., w_n) = \prod_{i-1}^{n} P(w_i|w_1, ..., w_{i-1}) \approx \prod_{i-1}^{n} P(w_i|w_{i-(n-1)}, ..., w_{i-1}) \tag{3.1}$$

It is assumed that the probability of observing the i^{th} word w_i in the context history of preceding i-1 words and it can be approximated by the probability of observing it in context history of preceding n-1 words.

$$\prod_{i-1}^{n} P(w_i|w_{i-(n-1)}, ..., w_{i-1}) = \frac{count(w_{i-(n-1)}, ..., w_{i-1}, w_i)}{count(w_{i-(n-1)}, ..., w_{i-1})} \tag{3.2}$$

The terms unigram, bigram and trigram language models are used to denote n-gram modesl of n=1 or n=2 or n=3 respectively.

3.3 Neural Language Models

In recent times the use of Neural Networks in developing language models has been the most preferred approach. In the context of learning algorithms the curse of dimensionality is the need of huge training data to map or learn highly complex

functions.In language models the problem arises from the huge number of possible sequences of words, for example a sequence of 5 words taken from a vocabulary of 10000 words gives rise to 5^{40} possible outcome of sequences. As the number of input variables increases the number of required examples grows exponentially. Neural Networks mitigate the effect of curse of dimensionality by thir ability to learn distributed representations. The core idea is to learn to link each word in the dictionary with a continuous valued vector representation. Words are now learned feature vector representations.

It can be visualized as each word pointing to a position in feature space and each dimension of the space representing a semantic or grammatical characteristic of the given word. The aim is to cluster functionally similar meaning words closer at least along some dimension or more. The task of the network is to map the sequence of feature vectors to a prediction of interest. The advantage of this distributed representation is that it helps the language model to generalize well to sequences that are not in the training set but are similar in terms of their feature vectors. As many different combinations of feature values are possible so a huge number of possible meanings can be represented compactly. If is infact quite similar to how a human chooses features of word i.e he might pick grammatical features like number(singular/plural), person(1st, 2nd, 3rd) and semantic features such as visible or invisible or animate or inanimate or features like shape, size and material. One of the task of the learning algorithm in neural network is to locate these features in feature space correctly and accurately.

Back in early days of Deep Learning Recurrent Neural Networks (RNN) were successful as powerful sequence models. Later come on other variations such as LSTM usually called Long Short Term Memory networks which were nothing but a special class of RNN capable of learning long term dependencies. LSTM's were followed by GRU commonly called Gated Recurrent Units which had to gates , a reset gate and

a update gate. In present day Language models the most successful architecture is however Transformers which is something we will delve in detail.

CHAPTER 4: RECURRENT NEURAL NETWORK

RNNs [8] are a popular model that have shown much promise in many NLP tasks. The main idea behind RNNs is use sequential information for prediction. The assumption in traditional neural networks are that the inputs and the outputs are independent of each other. It might be beneficial in some cases but in most others it is not the best idea to go forward it. Let's say you are predicting the next work in a sentence it is essential to know the preceding words. RNNs recurrently perform the same task for every element of a sequence, where the current output is dependent on previous input and previous computations. One intuition is to think about RNNs as having a memory which captures information about the previous calculations.

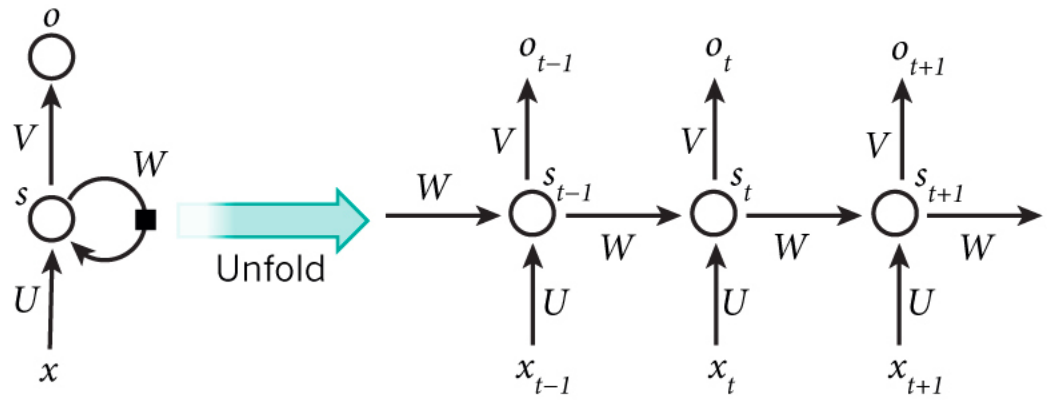

Figure 4.1: A recurrent neural network and the unfolding in time of the computation involved in its forward computation.

The above diagram is of a RNN which has been unfolded into a full network. By unfolding we write out the network for a complete sequence. To simplify this, let us assume we want to predict a sentence with sequence of 7 words, the network then would be unfolded into a 7 layer neural networks where each layer outputs one word.

To explain the diagram:

1. x_t is the input at time step t i.e. the first input is the first word. (x_0) is the one hot vector representation corresponding to the first word of the sentence and so on.

2. s_t is the hidden state at time step t.

3. s_t is the memory of the network which is calculated using previous hidden state and input of the current step.

$$s_t = f(Ux_t + Ws_{t-1}) \tag{4.1}$$

   The function f here is non-linearity such as $tanh$ or ReLU.

4. o_t is the output at step t. Let's say we are predicting the next word in a sentence, so it will be a vector of probabilities across the vocabulary $o_t = softmax(Vs_t)$.

5. The output at step o_t is calculated solely based on the memory at time t.

6. RNNs share the same set of parameters at each time step (U, W, V) as diagram. So as mentioned RNNs perform the same specific task at every time step with just change in the inputs resulting in reduced number of learn able parameters.

7. RNNs might not have output at every time step, i.e incase of classification or sentiment analysis we need the final sentiment and not the sentiment of each input word.

8. Training a Recurrent Neural Network or RNN is similar to other Neural Networks.

4.1 Training RNN - Forwardpropagation

The input x to the model will be a sequnce of words as shown in Fig 4.1. Each input we feed into the RNN is a single word. A network does not identify words as

words, but as numbers so we need to put a word at index 20 as a one word vector which is of the size of vocabulary i.e the word at index 20 would be a vector which is filled with 0's at every position and 1 at just position 20. Thus x is represented as a matrix which is a combination of all the vectors ranging from x_0 to x_{t+1} (each row of the matrix is a word represented as vector). The output of the neural network is of a similar format where o_t is a vector of the vocabulary size elements and each element represents the probability measure of that word being the next word in sentence.

To work with examples, let's consider the vocabulary size to be $C = 5000$ and the size of the hidden layer to be $H = 200$. We can go forward with the intuition that the hidden layer is the memory of the network, a bigger memory is more efficient in learning complex patterns but it results in higher computational cost.

$$s_t \in R^{200} \tag{4.2}$$

$$x_t \in R^{5000} \tag{4.3}$$

$$O_t \in R^{5000} \tag{4.4}$$

$$U \in R^{200\times5000} \tag{4.5}$$

$$V \in R^{5000\times200} \tag{4.6}$$

$$W \in R^{200\times200} \tag{4.7}$$

We want to learn the parameters U,V and W from the data. The total number of parameters we meed to learn are $2HC + H^2$. Here since $C = 5000$ and $H = 200$ so we need to learn a total of 20,40,000 parameters. The first task is to initialize the parameters, in short weight initialization comprises setting up the weights vector for all the neurons for the first time just before the neural network training process starts. Initializing the parameters U,W,V is tricky; initializing every weight vector to zero is

one approach but is not preferred because that results in symmetric calculations in all layers and the neurons start off being dead. So when the weight vector becomes 0 the output becomes 0 as well. The input vector x no longer plays a role in computing the output of the neuron. Another popular approach is initializing the weights randomly where one can use either standard or normal distribution. As the numbers are more than zero this time so the neurons would not be dead, the performance for the initial epochs might be low as the random values do not correspond to the actual distribution of the underlying data. It has been seen that initialization depends on the activation function that is being used in the network. Let's say *tanh* is the activation used so one recommended approach is to initialize the weights randomly in the interval of $[\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$ where n is the number of connecting from the previous layer.

The forward propagation here is just predicting word probabilities defined by the equations stated above. We return both the calculated output and the hidden states. Each o_t is a vector of probabilities representing the words in our vocabulary. So for each word in sentence, the model makes 5000 predictions representing the probabilities of the next word. The goal of a neural network is to minimize the loss function L, here our goal is to find parameters U,V and W that minimize the loss function of our training data. A very common choice is cross-entropy loss while a few others being Mean Squared Error Loss, Huber Loss, KL Divergence Loss to name a few. For crosss entrophy if we have N training examples or words in our data set and C classes i. the size of the vocabulary the loss wrt to o is given by:

$$L(y, o) = -\frac{1}{N} \sum_{n \in N} y_n \log o_n \tag{4.8}$$

It is simply doing a sum over the training examples and is adding to the loss based on how off the predictions are from the ground truth. The further away y and o are from each other the greater is the loss. We know we have C words in our vocabulary

so each word should be predicted with a probability of $1/C$ which makes the loss as $L(y, o) = -\frac{1}{N} N \log \frac{1}{C} = \log C$. Minimizing the loss by optimizing U,V and W is done through Stochastic Gradient Descent or SGD, the idea behind SGD is however quite simple and intuitive. During each epoch we iterate over all the training examples and during iteration it pushes the parameters to a direction that reduces the error. These directions are given by gradients on the loss: $\frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial W}$. Gradient descent uses something called as learning rate which can be defined as how big of a step we want to make in each iteration towards the correct parameter. Now the questions arises as to how one can calculate the gradients are calculated, in traditional neural networks it is done through back propagation. But for RNNs Back propagation through time is used.

4.2 Backpropagation Through Time (BPTT)

The goal of the network is to calculate the gradient of the error with respect to U,V,W and learn efficient parameters using SGD. So just as we add the errors we also add the gradients at each step for all training examples ($\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$). This is calculated using the chain rule of calculas (We have used E_1 as an example to show the calculations instead of E):

$$\frac{\partial E_1}{\partial V} = \frac{\partial E_1}{\partial \hat{y_1}} \frac{\partial \hat{y_1}}{\partial V} = \frac{\partial E_1}{\partial \hat{y_1}} \frac{\partial \hat{y_1}}{\partial z_1} \frac{\partial z_1}{\partial V} = (\hat{y_1} - y_1) \otimes s_1 \tag{4.9}$$

So $\frac{\partial E_1}{\partial V}$ depends on the values of the current time step $\hat{y_1}, y, s_3$. For $\frac{\partial E_1}{\partial W}$ the equation stands as:

$$\frac{\partial E_1}{\partial W} = \frac{\partial E_1}{\partial \hat{y_1}} \frac{\partial \hat{y_1}}{\partial s_1} \frac{\partial s_1}{\partial W} \tag{4.10}$$

$S_1 = \tanh(Ux_t + Ws_0)$ so we can see that s_1 depends on s_0 and all the previous steps. We need to to take the derivative of W but we cannot treat s_0 as a constant, so the

chain rules is applied again and the equation stands as:

$$\frac{\partial E_1}{\partial W} = \sum_{k=0}^{1} \frac{\partial E_1}{\partial \hat{y_1}} \frac{\partial \hat{y_1}}{\partial s_1} \frac{\partial s_1}{\partial s_k} \frac{\partial s_k}{\partial W} \tag{4.11}$$

We sum up the contributions of each step to the gradient so we need to backpropagate [9] from $t = n$ to $t = 0$

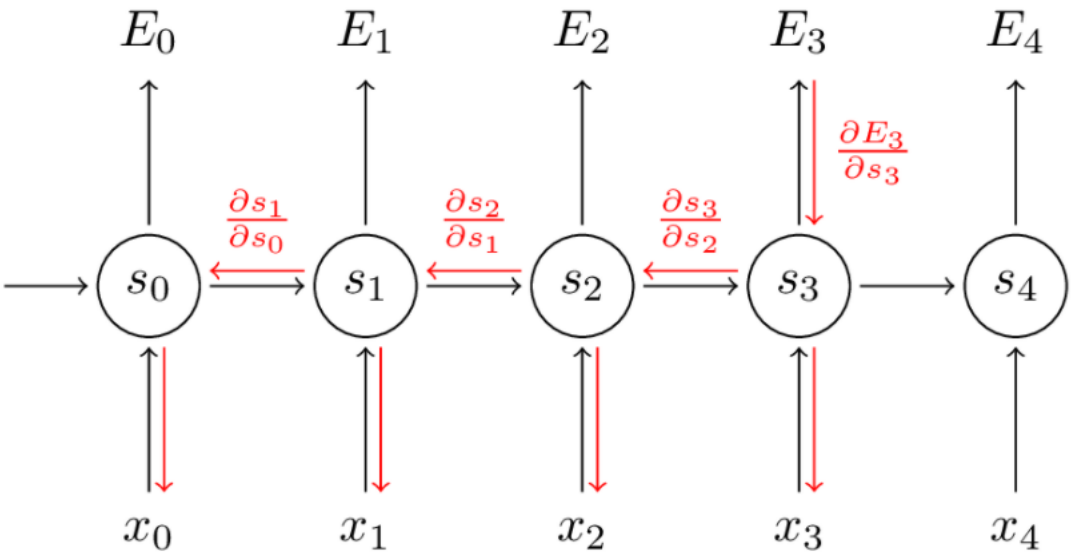

Figure 4.2: BPTT by summing up the gradients at each time step

Te key difference is that we sum up the gradients for W at each time step. This is one major reason why RNNs are hard to train.

CHAPTER 5: LSTM

Recurrent neural networks suffer from short term memory, so if a sequence is long they suffer carrying information from earlier steps to later ones. LSTM's and GRU's solved the short term memory problem. The core of a LSTM unit is the cell which is a bot of memory so that it can remember the past. Lets take an example:
The dog which already ate..... was full.
The dogs which already ate..... were full.
Here the dots represent the presence of another sentence in between. In the first sentence the cat is signular so LSTM uses 'was' while in the second sentence as dogs were used so it signified a plural quantity and 'were' was used. This shows the use of memory in language models.

LSTM units are made up of three gates; Input Gate, Forget gate and the Output Gate.The gates decide which information is relevant to and has to passed further or can be forgotten during training. The gates use sigmoid activation functions which are quite similar to tanh hyperbolic activation. They squish the values between 0 and 1. 0 blocks everything while 1 makes everything pass forward. These gates also help in tackling the problem of vanishing or exploding gradients through a gating mechanism. In a forget gate the gate decides what information is to be thrown away or kept, so a sigmoid activation is used where an output closer to 0 means to forget and an output closer to 1 means it has to be kept.

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{5.1}$$

Here f_t represents the forget gate, σ is the sigmoid function, w_f is the weight of forget

gate, h_{t-1} is the output of previous lstm block, x_t is the input at current step and b_f is the bias for the forget gate.

The cell state is updated with the help of input gate. The previous hidden state and the current input is passed into a *sigmoid* function resulting the output values between 0 and 1. The same operation is passed through a *tanh* function to regulate the network. The two outputs are then multiplied.

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{5.2}$$

Here i_t represents the input gate, σ is the sigmoid function, w_i is the weight of input gate, h_{t-1} is the output of previous lstm block, x_t is the input at current step and b_i is the bias for the input gate.

The cell state gets point-wise multiplied by the forget vector, if the values are multiplied by 0 then they are dropped. The the output of input gate is added point-wise with the result. This gives the new cell state.

$$\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c) \tag{5.3}$$

Here $\tilde{c}_t$ represents candidate for cell state at timestamp (t)

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{5.4}$$

Where c_t is the cell state memory at time stamp t

The last gate is output gate which decides the next hidden state. The new cell state and the new hidden state are carried over.

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{5.5}$$

Here o_t represents the output gate, σ is the sigmoid function, w_o is the weight of output gate, h_{t-1} is the output of previous lstm block, x_t is the input at current step and b_o is the bias for the output gate.

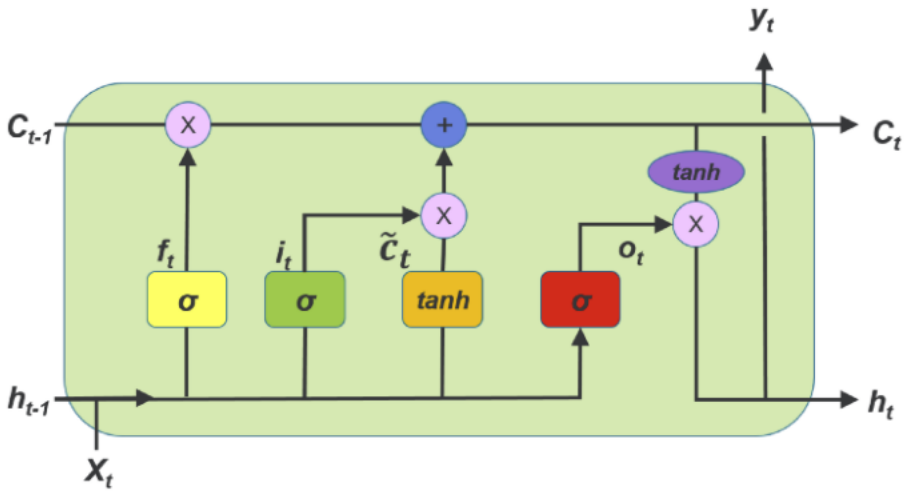

Figure 5.1: LSTM Block

CHAPTER 6: GATED RECURRENT UNIT (GRU)

The idea of GRU originated from LSTMs primarily and thus they are also a newer variation in the family of recurrent neural networks. GRUs got rid of three gates and used two gates; reset gate r and an update gate z. GRUs are abit faster than LSTMs as they have lesser tensor operations as compared to LSTMs. Also GRUs donot have an internal memory c_t and unlike LSTMs a second nonlinerity is not applied in GRUs. The update gate is quite similar to the forget gate and input gate of LSTM, it primarily decides which information to keep or which to be dropped.

$$z = \sigma(x_t U^z + s_{t-1} W^z) \tag{6.1}$$

$$r = \sigma(x_t U^r + s_{t-1} W^r) \tag{6.2}$$

$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h) \tag{6.3}$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1} \tag{6.4}$$

Here x_t is the input vector, h_t is the output vector, W is the weight of the respective gates and b are the biases for the respective gates.

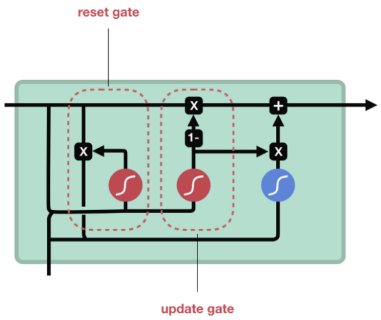

Figure 6.1: Gated Recurrent Units with Reset and Update Gate

CHAPTER 7: SEQUENCE TO SEQUENCE MODELS

RNNs can be used to predict the future elements of a sequence given prior elements. However in the context of translation or generation models (summarization) two sequences are required, one as a input sequence and one as an output sequence. Seq2seq models [10] are built on top of language models in a two step process. The encoder which converts an input sequence to a fixed representation. The decoder is trained both on the output sequence and the fixed representation generated by the encoder network. The ability of the decoder to learn from the output sequence and the fixed representation from the encoder helps in making word predictions. Let us consider an example where our language model sees the word "Jordan", it is not exactly sure whether the next word should be about the city or the athlete but since we also pass an encoder context the decoder can utilize the context to understand whether it is speaking about the city or the person.

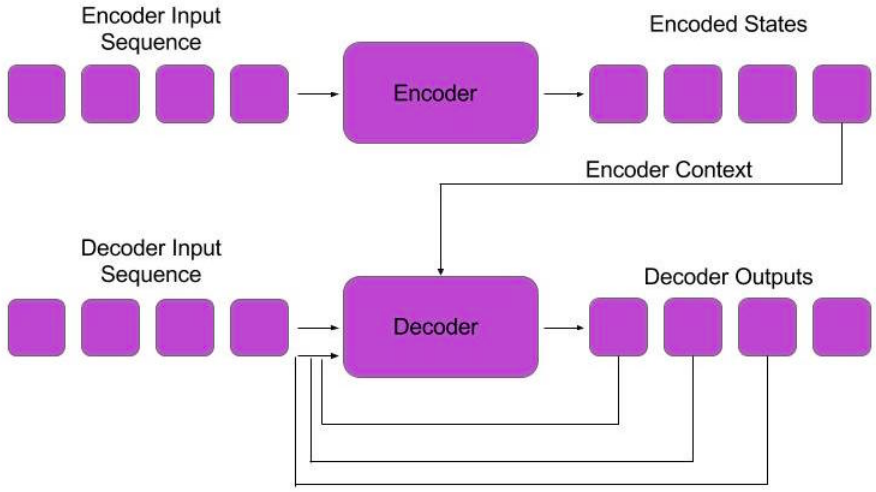

Figure 7.1: Sequence to Sequence Model - the encoder outputs a sequence of states. The decoder is a language model with an additional parameter for the last state of the encoder. [1]

CHAPTER 8: TRANSFORMER

Sequence to Sequence models have been successful and with the use of attention the results have been even better but the issue with RNNs lies in their inability of being able to be parallelized. Transformer has been able to handle the problem of long term dependencies while parallelizing the task [11]. A very high level overview of the model can be seen in the below figure: The encoding component shown here is a

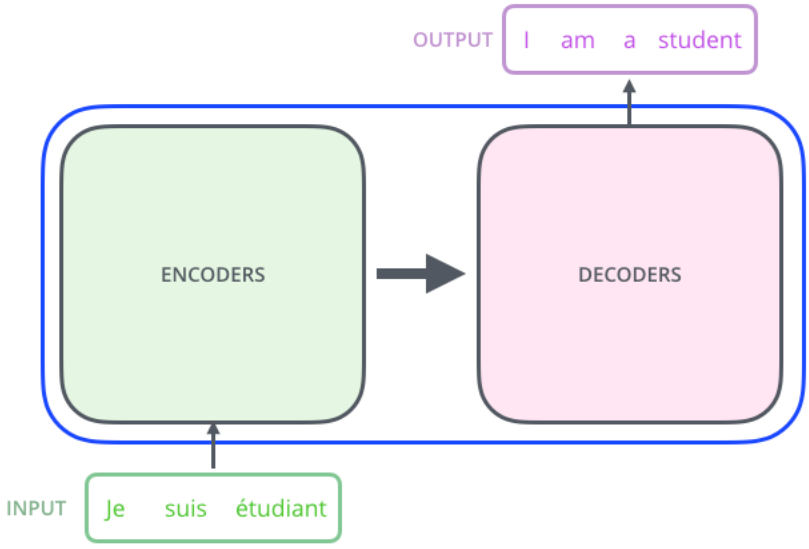

Figure 8.1: Basic Transformer Architecture used for Machine Translation [2]

stack of encoders and the decoding component is another stack with the same number of decoding components. The way the attention mechanism is applied or customized is what makes makes the Transformer architecture so successful and novel.

8.1 Structure of Transformer?

All the encoders are identical in structure and each one of them can be primarily divided into two parts a self attention layer and a feed forward neural network. The

encoders input first flows through the self attention layer where the encoder looks at other words in the input sequence as it encodes a specific word. The outputs of the self attention layer are fed into a feed forward neural network. The decoder has three layers; the first one being self attention, the second layer is the encoder-decoder attention layer and the final layer is a feed forward neural network.

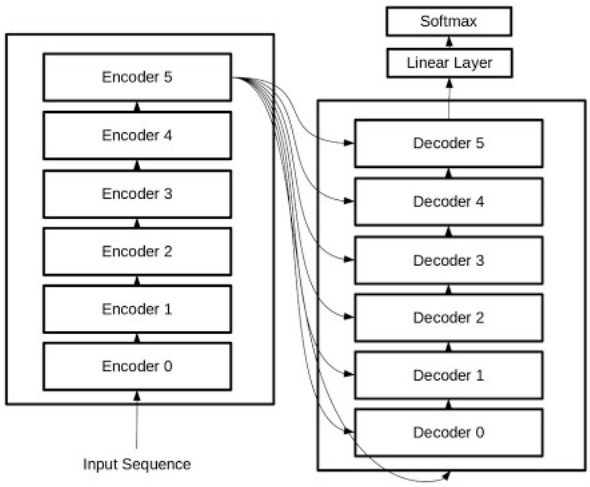

Figure 8.2: Overview of complete Transformer

Just like any other language processing task the first step is converting the input words as vectors using word embedding. The bottom most encoder handles the word embeddings, the vectors are of size 512. In all the encoders above the bottom most one get their input which is basically the output of the encoder just below them. The general norm is to set the size of the list as the length of the longest sentence in training data set. Words in each position runs through it's own path in encoder i.e that means the position of each word is fixed.

8.2 Self Attention

Self attention [12] is the mechanism that transformer uses to refine the understanding of other relevant words in respect to the one that is currently taken into account. It's a way the decoder can focus on certain specific parts of the encoded represen-

tations. The first step is constructing the three vectors (Query, Key, Value) from input vectors from the encoders or word embeddings. These vectors have a size of 64 as compared to the size of encoder's input/output vectors which have a size of 512. Calculating self attention is basically calculating a score. For example we are calculating self attention for a word "Test". We need to score each word of the input sentence w.r.t to this word. The score is a measurement which determines how much focus we can place to other words of a sentence as we encode a word at a certain position. The score is calculated by taking a dot product of the query vector with the key vector of the respective word we are scoring. For example if we are calculating attention for a word in position one, the first score would be dot product of q1 and k1, the second score would be a dot product of q1 and k2 and so on.

The next step is to divide all the scores by the square root of the dimension of key vectors used (here the dimensions are 64 as mentioned above so we need to divide by 8).

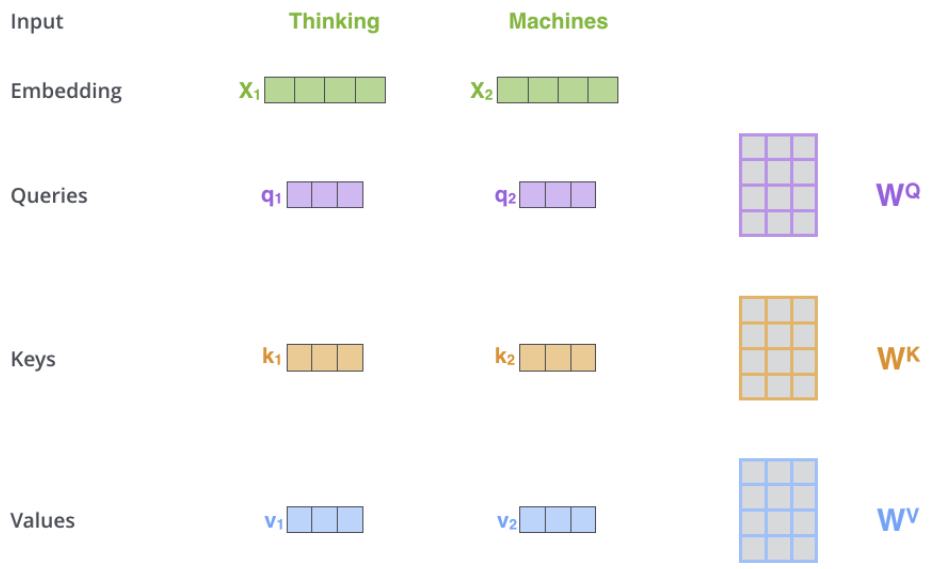

Figure 8.3: Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

The scores are divided by the square root of dimensions to give more stable gradients. The next step is doing a softmax operation to normalize the scores so that they all add up to one. The softmax decides how much each word is expressed at this position. So the first word at position one will have the highest softmax score but there might be other high scores if the first word is referring to something that comes later in the sentence.

The next step is multiplying the softmax score and the value vector, this is done to drop out irrelevant words and keep intact important words. This can be alternatively thought as a weighted value vector. The final step is taking a sum over all weighted value vectors which gives us the output of the self attention layer for that specific position (first word here). The results of this calculation are then feed to the feed forward neural network .

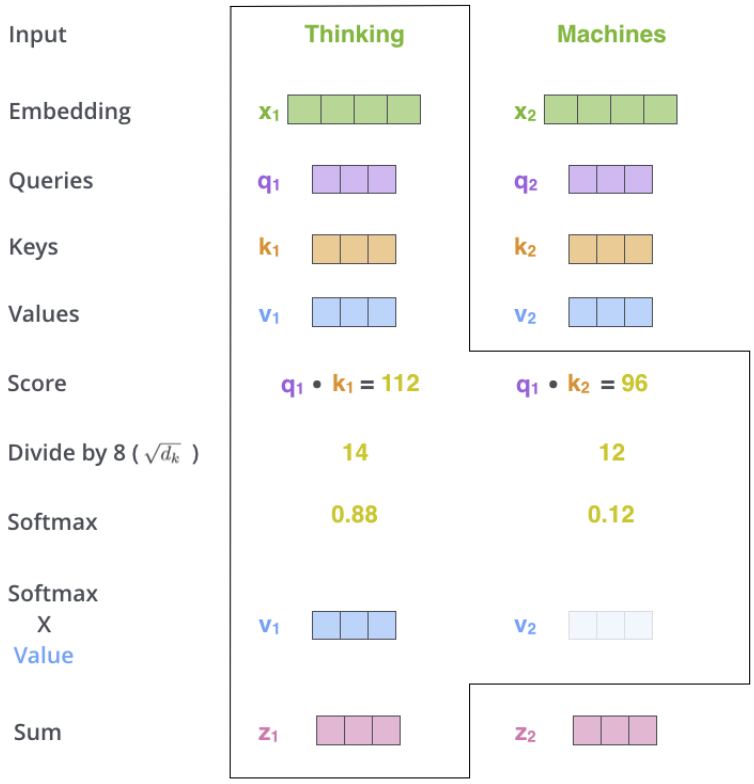

Figure 8.4: Entire Self Attention calculation in Visual Format [2]

However this entire calculation inside a neural network is done in a matrix multiplication operation for faster computation. We calculate the word embeddings and then fit them into a single matrix X and then we multiply it with the trained weight matrices (WQ,WK,WV).

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Figure 8.5: Attention score formula

8.3 Multi-head Attention

The approach of self attention or dot product attention was succeeded by Multi-head attention mechanism. In multi-head attention we have multiple sets of Query/Key/Value weight matrices. Each of these sets are randomly initialized but after training each set represents a different representation subspace. On all of these versions of queries, keys and values the attention function in performed in parallel which leads to n dimensions of output values [11].

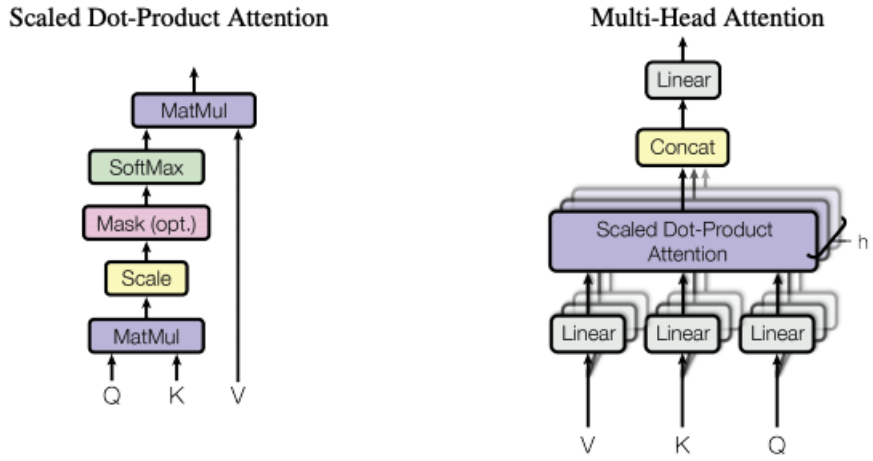

Figure 8.6: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

This gives us n different Z matrices which needs to be condensed into one matrix. Multi-head attention allows the model to jointly attend to information from different representation subspaces.

$$\begin{aligned}\text{MultiHead}(Q,K,V) &= \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O \\ \textbf{where } \text{head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

Figure 8.7: Multihead Attention Score formula

Transformer copies the classical attention mechanism (Bahadanau et al) where the encoder decoder attention layer queries are from previous decoder layer while the keys and values are from encoder output.

8.4 Point-wise Feed Forward Neural Network

Each layer in the encoder and decoder is processed by a feed forward neural network. The point-wise feed-forward neural network is a two layer linear transformation with ReLU activation which is used identically throughout the model after the attention blocks. The dimensions of input and output are $d_{model} = 512$ and the inner layers has dimensions of 2048.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{8.1}$$

8.5 Positional Encoding

As the model is not using any recurrence or convolutions so in order for the model to account for the order or sequence of words in the input positional encoding is used. It is the method of assigning a vector to each of the input embeddings, these vectors follow a specific pattern and helps in identifying the position of each word and the distance between words in the sequence. The goal of adding these vectors is that the information of the sequence and the information of the distance between words are

meaningfully captured. The encoding allows the model a sense of which portion of input it is currently dealing with . The positional embedding can be learned or fixed parameters with comparable results. For the transformer architecture the positional encodings are represented through the following functions:

$$PE(_{pos,2i}) = sin(pos/10000^{2i/d_{model}}) \tag{8.2}$$

$$PE(_{pos,2i+1}) = cos(pos/10000^{2i/d_{model}}) \tag{8.3}$$

8.6 Layer Normalization

Layer Normalization directly calculates the normalization statistics from the summed inputs to neurons within a layer. It has been observed that for RNNs it is a more preferred normalization technique instead of batch normalization.

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{8.4}$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{8.5}$$

Here H stands for the number of hidden units in the layer.

CHAPTER 9: BERT

9.1 Why was BERT required ?

One of the main challenges while working in NLP as compared to Computer Vision or Image Processing is the dearth of enough trainable data. One might think that with rise of big data there is enormous amount of text data available over the internet but when we want to split the data into task specific fields we are only left with corpus which is merely of few thousand or few hundred examples. Unfortunately to perform well in Deep learning based NLP models one requires millions or billions of annotated training examples. Modern research has been majorly in the direction where general purpose language models are trained on unannotated text data over the internet which is known as pre-training and then these pre-trained models are fine tuned on task specific datasets which are comparatively smaller. This has resulted in great accuracy improvements in almost all major Language processing tasks. BERT has been a brainchild of this very idea and it can easily be fine-tuned on tasks like sentiment analysis, question-answering and etc.

9.2 Transfer Learning

Transfer learning is a technique where a deep learning model is trained on a large dataset is used to perform similar tasks on smaller datasets. The model is called a pre-trained model. What ImageNet did for computer vision was followed much later by Transformer for language processing tasks. Briefly stating the advantages of using transformer based architectures are that these models do not process serially one by one rather the entire sequence as one, so the models can be parallelized unlike RNNs. Henceforth huge amount of unlabelled data was no longer required for to train models

from scratch thus this led to use of transfer learning in NLP.

9.3 Fine Tuning

BERT[7] leverages a huge architecture and over 100 million parameters so training it from scratch would result in overfitting. The general approach in using BERT is thus using an already pretrained model on a larger dataset and then further training on a smaller task specific dataset. There a few fine-tuning approaches which are discussed as below:

- Training entire architecture - It means training the complete pre-trained model on a smaller dataset and feeding the output to a softmax function.
- Training some layer and freezing the other layers - It means training the model partially. It is the most used method where the weights of the initial layers are frozen and the higher layers or the layers towards the end of the network are retrained.
- Freezing the entire architecture - It is quite similar to the last approach but we freeze the entire architecture and few new layers at the end are added as per the task on which the model needs to be trained.

9.4 BERT Input

A pre-trained BERT model uses a BERT Tokenizer which takes raw text strings and converts them into tokens that the model can utilise. One of the major reasons of using an inbuilt tokenizer is that BERT has it's own vocabulary of tokens. One minor drawback in using pretrained embeddings is that one cannot use custom created embeddings since all of the knowledge is based on the embeddings it is trained on. On a different vocabulary the model is not sure of what to do.

Now one might assume that what happens if an out of vocabulary word is present in your dataset. It is not a problem since BERT has an efficient way of handling such

words on which it has not been trained before. BERT uses a Wordpiece model which is a word segmentation algorithm to handle OOV words. It forms a new subword on the basis of maximum likelihood. The original BERT has a vocabulary of the size of 30000 tokens. Approximately one fourth of the tokens are subwords and not complete words. Breaking into subwords is shown in the below figure:

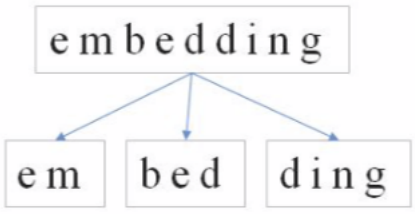

Figure 9.1: Sub-word selection

In case one of the subwords are missing then it further breaks them down to individual characters. BERT also has tokens for the punctuation's used further helping in understanding the text.

9.5 Padding

BERT process each of the words independently so it allows parallelizing the process and helps in running all the input words through BERT at once.

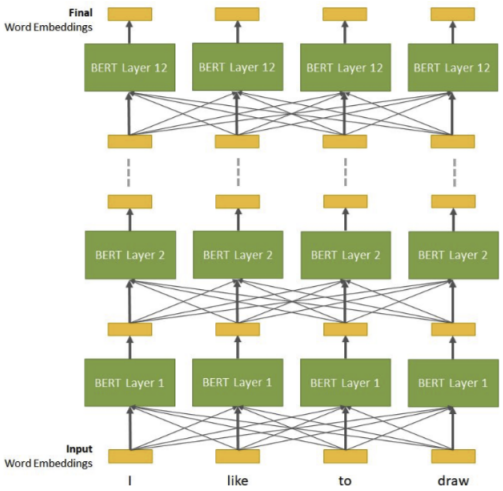

Figure 9.2: BERT encoder layers

Since the model is a stack of 12 encoders so each one of the 12 layers takes the enhanced embeddings from the previous one and further enriches the embeddings. BERT can be applied to sentences of length up to 512 tokens but in order to aid parallel processing of multiple texts at once, a single fixed length input is required to be fed to the model. All the sentences are thus either padded or truncated to a single fixed input length of 512.

9.6 Embeddings

BERT just like other language processing models requires inputs in the form of numerical vectors, so this means converting the features such as vocabulary and parts of speech into numerical representations. While there are multiple approaches to do so like one hot encoding, neural word embeddings, word2vec etc the procedure that BERT takes into account is more explicit than any other embedding algorithms. It takes into account the context and produces representations that are dynamically informed by the surrounding words. Let's take an example:
"The man had a fractured arm"
"It is important to arm yourself with solid education"
Here unlike other embedding algorithms like word2vec BERT assigns different numerical representations to the same word arm as it has been used differently in both the sentences.

BERT uses a sum of three types of embeddings [13] in it's model. Each one of the embedding types serve a specific purpose and help in better learn able features.

- Token Embeddings - The role of token embeddings is to transform words into vector representation of fixed dimensions. BERT assigns a 768 dimensional vector for each word. The first thing that is done is adding a start [CLS] and end token [SEP] to the sentences. The [CLS] token is added at the beginning of the sentence and [SEP] (separator) is added at the end of the sentence. The tokenization is done with Wordpiece tokenization, which enables the model to

store 30,522 words and the token embeddings layer converts the word into a 768 dimensional vector. To visualize this with an example will be a much better idea:

"I went fishing today" - 4 WORDS

"[CLS]", "I", "went", "fish", "##ing", "to", "##day", "[SEP]" - 8 Tokens

So this results in 8 input tokens which is transformed to a matrix of (8,768).

- Segment Embeddings - Apart from text classification there are couple of two sentence tasks that the BERT model is a benchmark. For example: Natural language inferencing where two sentences are fed into the model and the task into determine how they are logically connected to each other (entail, contradict or neutral). In such tasks sentence pairs are concatenated and fed into the model and with the help of segment embeddings the inputs are differentiated. The segment embedding layer has just two vector representations, a vector of index 0 is assigned to all the tokens that belong to input 1 and a vector of index 1 is assigned to all the tokens of input 2. Incase of single sentences the vector is a o indexed vector. The example below explains it further:

  "I went fishing. It was great" - 6 WORDS

  "[CLS]","I","went","fish","##ing","[SEP]","It","was","great", "[SEP]"

  "0","0","0","0","0","0","1","1","1","1" - SEGMENT EMBEDDINGS

- Position Embedding - As BERT is built on the transformer architecture such just as transformers it doe not encode the sequential nature of the inputs so positional embeddings are used to distinguish between same words but in different positions i.e a word [A] in position 3 will have a different encoding that [A] in position 10, this leads to the model learning a vector representation based on it's position. The model is allowed to process an input length of maximum 512, so the positional embedding layer is a lookup table of dimensions (512,768).

The 3 embedding representations are summed element wise to produce a single tensor of shape (1,n,768) which is further processed to the encoder layer.

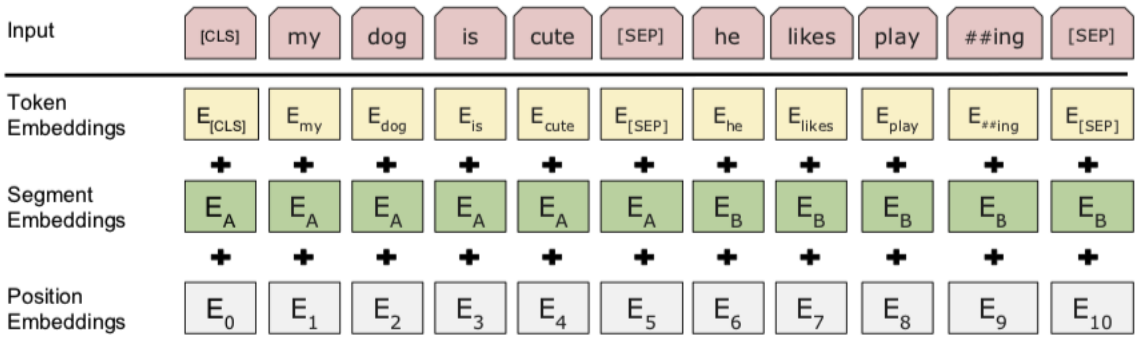

Figure 9.3: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

9.7 Pretraining BERT

Unlike other architectures BERT is not trained on traditional left to right or right to left models, on the other hand it is trained on two unsupervised tasks as described below:

- Masked Language Model (MLM): Before feeding word sequences in BERT 15% of the words in sentences are replaced with a mask[MASK] token. It forces the model to predict original values of the masked words on the basis of the context provided by other words. The entire process is carried out by adding a classification layer on top of encoder and multiplying the output vectors by the embedding matrix so that the vectors resemble the vocabulary. Finally a softmax operation is done which assigns a probability measure to the vectors and the vector of the word with highest probability is chosen.

  In training 15% of the masked tokens are selected randomly. All of these tokens are not masked, 80% of these tokens are masked with [MASK] token, 10% are replaced with a random word and in the rest the original word token is

kept. Masking all the time does not always produce good meaningful token representations.

- Next sentence Prediction (NSP) - Another pretraining task on which BERT has been trained is NSP, where the model receives pairs of sentences as input and learns to predict whether the second sentence is logically subsequent between the two sentences. During training 50% of the sentences are taken as paired sentences while 50% of the remaining sentences are randomly chosen where the second sentence does not logically follow or relate to the first. In training both MLM and NSP are used and trained together with the goal of minimizing the combined loss of both the tasks.

9.8 Architecture

Post tokenization of the words, the BERT model generates a set of enriched embeddings for every token. The core idea behind this is Attention which we have skimmed through in the transformers model previously. Let's take an example sentence for clarity:

"The tiger is an endangered species, their scientific name is Panthera Tigris"

Here the pronoun "their" refers to a word in the first half of the sentence but how does a language model understand that ? This is done using self attention. When producing enhanced embeddings for the word "their", the self attention mechanism takes a weighted average of embeddings of other surrounding words. This weight is the number which quantifies how much of attention should be given to the context words.

Figure 9.4: Attention with respect to "their" word

Here a darker shade means more weight. The weights are assigned on the word

"their". All the words are assigned some weight (I have kept the rest of the words white even though they have weight here). The weights are calculated with a softmax function.

9.8.1 Calculating Attention Weights

In order to build the intuition of self attention let's look at how Self-Attention mechanism assigns weights to words in a sentence. The input word is "their" and the weight needs to be calculated for the context word "name". The first step is calculating the dot product between the embeddings of "their" and "name". The result is then passed to a softmax function to get a distribution and these are the weights.

9.8.2 Projecting the embeddings into query, key and value vector spaces

The calculation of attention weights is done only after projecting the embeddings into 3 different vector spaces. Projection is technique where we take the vector representation of anything and multiply it with a projection matrix to place the object to a different vector space.

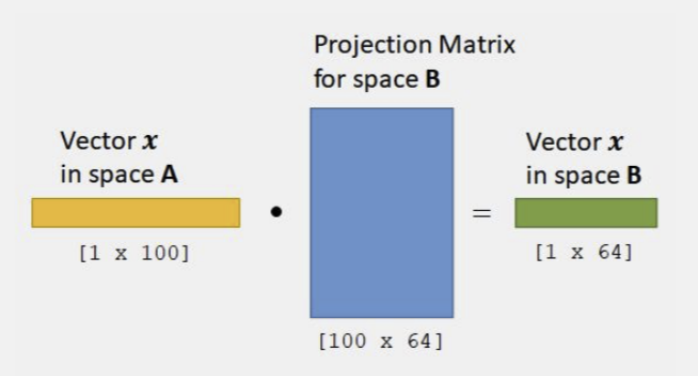

Figure 9.5: Both the yellow and the green vectors are representations of âx', but the two spaces (A and B) expose different aspects of âx'

The 3 vector spaces are key,query and value. The input word "their" is first projected into query space, next every context word is projected into key space, then a softmax operation is done on the basis of dot product of query and key. Finally the

product of the softmax scores and the value vectors done and then the weighted value vectors are summed up.This gives the self attention score with respect to the word "their".

9.8.3 Feed Forward Neural Network

After self attention has been applied to the word embeddings, the output is passed to a 3 layered neural network.

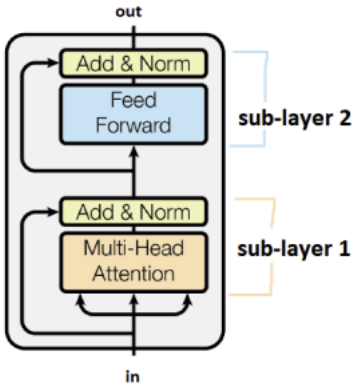

Figure 9.6: Transformer Encoder showing multi-head attention layer and the Feed Forward Neural Network

The feed-forward neural network is sub-layer two as shown in figure. This layer is a normal fully connected neural network with weights $W1$ and biases $b1$, the non-linearity used is $Gelu$ and then a second fully connected layer is applied with weights $W2$ and biases $b2$. The Gelu activation used here stands for Gaussian error linear unit. Activation functions in general allow faster and better convergence of neural networks. Dropout are used to regularize where some of the activations are multiplied by 0. Another regularizer called zoneout is used to multiply the input by 1. All of these are combined by stochastically multiplying the input by 0 or 1 and getting the output value deterministically.

The neural network is designed in a way so that you feed a $1 * 768$ input (word embedding after attention) and get a $1 * 768$ output. The neurons in hidden layer are

3072, which is set as per convention i.e. 4 times the embedding size.

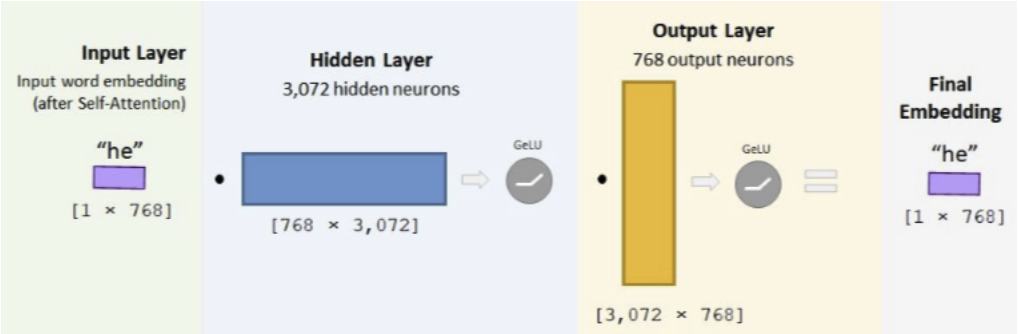

Figure 9.7: Feed Forward Network with Gelu activation

9.8.4 Multi-headed Attention

Multi-head attention is a mechanism which runs through an attention mechanism multiple times in parallel. The independent attention calculations are concatenated and linearly transformed into a required dimensional output. They are multiple instances of the self attention mechanism , where each one is trained to perform different functions. The output matrix is as follows:

$$Multihead(Q, K, V) = concat(head_1, ..., head_h)W^O \tag{9.1}$$

where each of the heads mean:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{9.2}$$

Each instance of self attention is referred to as attention heads and BERT is designed with 12 heads. Each one of the heads have their own unique instance of Q, K, V projection matrices.

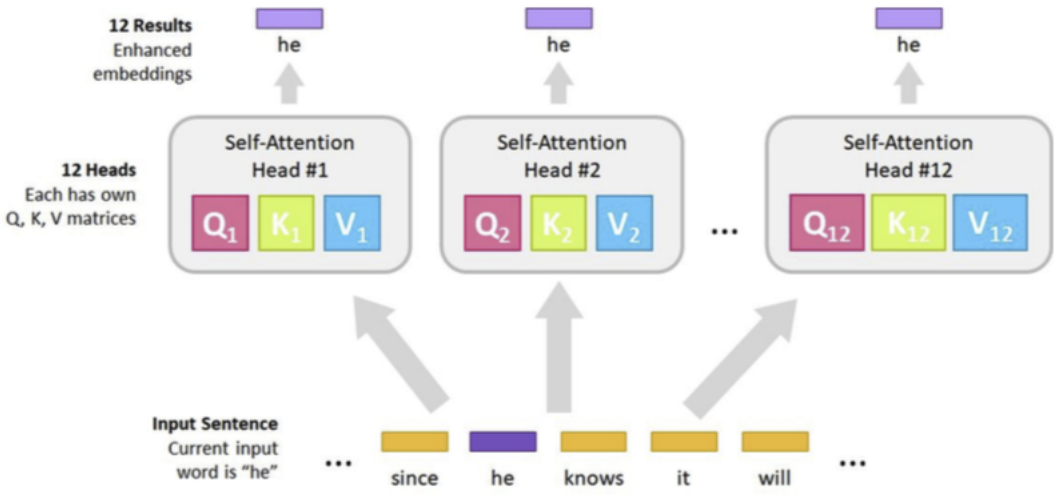

Figure 9.8: Visual Illustration of multi head Attention

As shown in the above figure for a single input word it is run through 12 unique instances of self attention, so this outputs 12 different enriched embeddings. While each BERT layer only produces one embedding per input word so the 12 outputs are needed to be combined into one.

9.8.5 Residual Connections

We know each of the encoders can be broken down into two sub-layers. A residual connection is used around each sub-layer which is followed by layer normalization. The figure below will help us in visualizing the residual connections further:

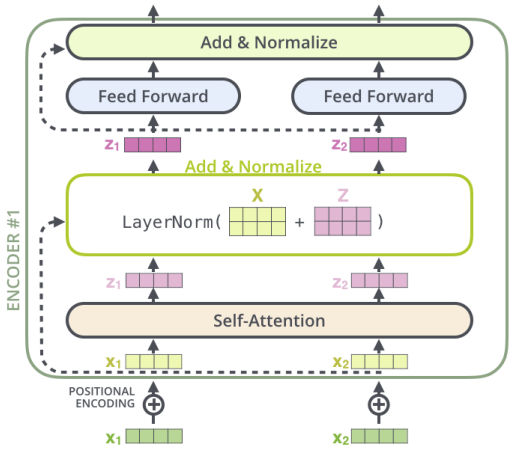

Figure 9.9: Visual Illustration of Residual connections

The output of the sub-layers can be thought as LayerNorm(x +Sublayer(x)) where sublayer(x) is the function implemented by the sublayer. This is followed in the decoder side as well.

CHAPTER 10: METHOD

10.1 Dataset

The experiments are carried out on CNN/Daily Mail dataset which is a collection of news articles used specifically for summarization tasks. A reason for using this corpus is because it consists of multi-sentence summaries. The dataset [14] is comprised of an average of 3.75 sentences per summary or approximately 56 tokens on average. The pre-processed version of the dataset has 287,226 training set samples, 13,368 validation set samples and 11,490 test set samples. We operated on the non-anonymized version of the dataset as it is preprocessed. We tokenize the dataset using WordPiece tokenizer, the tokenization decreases the average size of text to something around 650-690 and the average summary length to 50 tokens.

10.2 Model

The main architecture is built on Sequence-to-sequence framework built on top of BERT, we incorporate a decoder network to the BERT architecture. The input document is denoted as $X = \{x_1, x_2, ..., x_m\}$ and the output summary is denoted as $Y = \{y_1, y_2, y_3, ..., y_L\}$. Given an input document X a rough version of the summary is predicted by a transformer decoder. The output of the decoder can be considered as summary in stage 1 which is then processed through another BERT model and the summaries are refined in this process.

10.2.1 Encoder

In the encoder size of the network, BERT has been used as an encoder which helps in converting input document texts to encoded vector representations. Firstly the sequence which has been feed as input is transformed to embeddings and then

the encoders [15] construct sentence embeddings using attention to compute context aware representations of words leveraging both the ordering of words and identity of other words with respect to the current word.

$$H = BERT(x_1, x_2, ...x_i) \tag{10.1}$$

10.2.2 Decoder

The model has a two stage decoding process [16] where in the first stage a N layered Transformer decoder is used. The decoder learns the conditional probability $P(A|H)$, as we train the decoder attention mechanism of the model helps the decoder learn soft alignments within summary and source documents. At this stage the decoder's predictions are based on previous outputs and the hidden encoded representations. The learning objective of the decoder is to minimize the negative likelihood of the conditional probability.

$$L_{dec1} = \sum_{t=1}^{|a|} -logP(a_t = y_t^*|a_{<t}, H) \tag{10.2}$$

Here y_t^* denotes the ground truth word of summary at the t-th position. The stage 2 of the decoding process incorporates a refine process, which helps the decoder to leverage BERT's contextualized and enriched representations. The rough summary output from the stage 1 decoding is passed to BERT model to generate better context rich vectors. The output is then passed to the final transformer decoder which predicts the final summary. The output of BERT which goes inside the final decoder is masked, at t-th time step the t-th word of summary is masked which is then predicted by decoder on the basis of other words present in the summary.

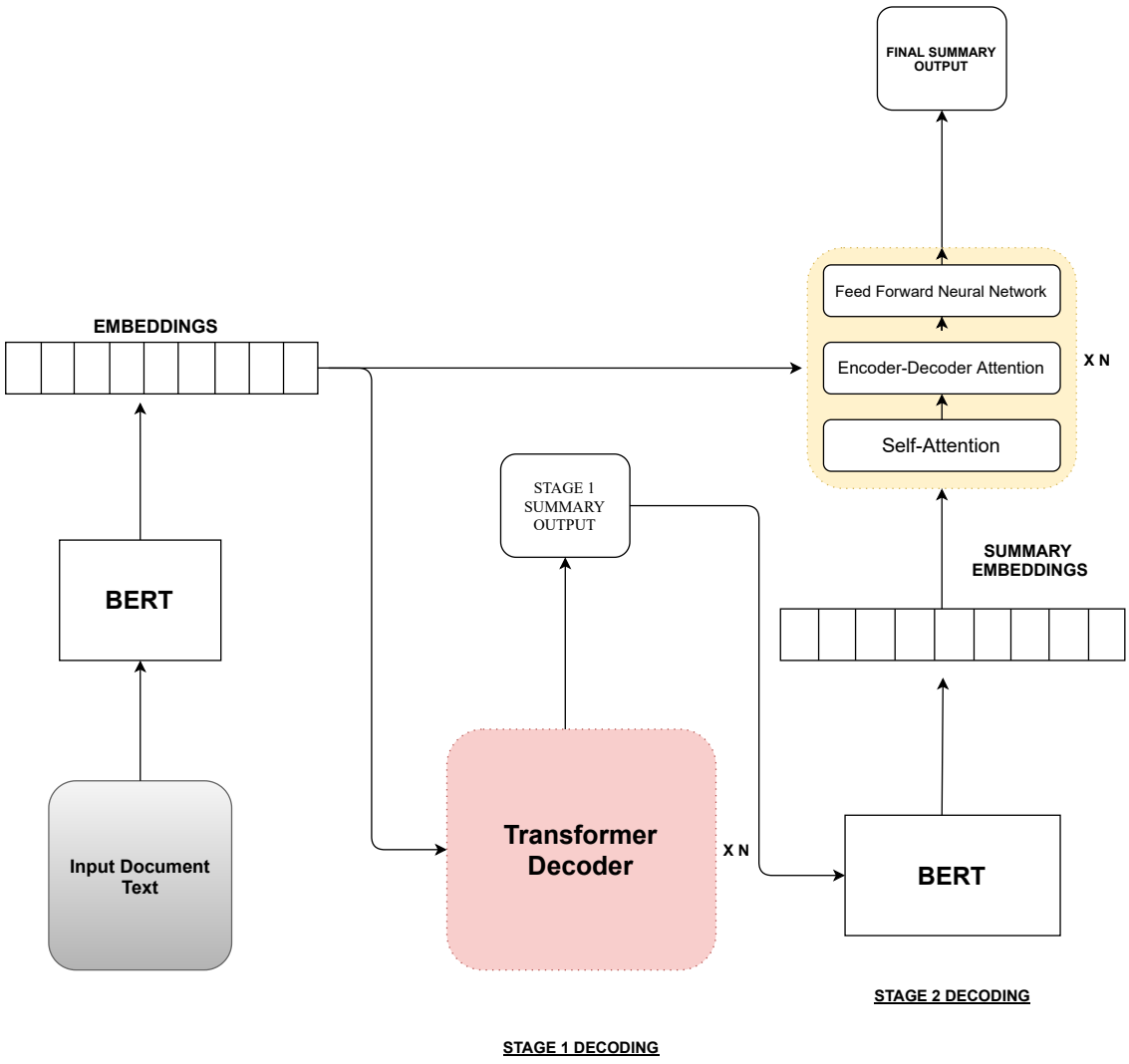

Figure 10.1: The novel BERT architecture with multi stage decoding

The objective function required for learning is shown in below figure where y_t is the t-th summary word and y_t^* is for the ground truth summary word.

$$L_{dec2} = \sum_{t=1}^{|y|} -logP(y_t = y_t^* | a_{\neq t}, H) \tag{10.3}$$

From the perspective of a language model, the stage two of the decoding process provides a more context rich input sequence. One way to think of it is that the stage one of the decoding process generates an initial summary while the stage two decoder fine-tunes the generated summary. The reason why a portion of the sentences are

removed as they are feed to the final decoder is solely because BERT is pre-trained in an almost similar task where a certain percentage of the inputs are masked i.e MLM or masked language modelling. We can think of it as a cloze task, wherein we mask out a certain percentage of the words and feed the rest of the sequences to the model. The model then predicts the masked words based on the sequence of context words, that's exactly similar to BERT's pre-training objective. It thus succeeds in generating more fluid and natural sequences since it has already been trained on that task. Both the decoder transformers are trained using Teacher Forcing, where the ground truth is used to help the training process, here the target word is passed to the decoder to help make better predictions for words at future time steps. Teacher forcing helps in converging the network faster, during early stages of training the predictions of the decoder are bad so if teacher forcing is not used then the error-ed model is carried forward and the errors accumulate over time making the model learn very less.

Another scenario occurs where a certain section of the tokens in input document are out of vocabulary words, an approach called copy mechanism [17] is followed as done in the original transformer architecture. In this certain segments in the input sequence are selectively replicated in the output sequence. This is done using a model called $COPYNET$, it can integrate both the normal word generation process in the decoder along with the copy mechanism which chooses sub sequences in the input sequence and place them at proper positions in the output sequence.

Let's say $V = \{v_1, v_2, ...v\}$ denotes the target vocab and X denotes the unique set of words in source sentence $x = \{x_1, x_2, ..., x\}$. The unknown words are denoted by $< UNK >$ token which is not a part of Vocab V. The extended voab thus is an union of $V \cup X \cup < UNK >$.

$$p(y_t|\cdot) = \underbrace{p_g(y_t|\cdot)}_{\text{generation prob.}} + \overbrace{p_c(y_t|\cdot)}^{\text{copy prob.}}$$

Figure 10.2: Caption

The above figure shows the final probability of a given token y from the extended voacb at the t-th time step. We will now look at it from the perspective of our model. At time step t attention probability distribution is calculated over source X using a dot product of last layer of decoder o_t and encoder output h_j.

$$u_t^j = o_t W_c h_j \tag{10.4}$$

$$a_t^j = \frac{exp u_t^j}{\sum_{k=1}^{N} exp u_t^k} \tag{10.5}$$

The next step involves calculating g_t which makes a soft choice between selecting from source or generating. g_t gives the weighted sum of copy [17] and generation probability put together to get the final probability.

$$P_t(w) = (1 - g_t) P_t^{vocab}(w) + g_t \sum_{i=w} a_t^i \tag{10.6}$$

As discussed earlier during training process the two processes are trained using teacher forcing and the objective functions is:

$$L_{model} = L_{dec1} + L_{dec2} \tag{10.7}$$

The ground truth is feed to each decoder and the objective function is minimized at training however at test we do not have the ground truth summary. We choose the argmax of the probability $P(y|x)$. This leads to a situation called Exposure Bias as due to the discrepancy between objectives in learning/training and inference.

As we are dealing with a language model for a language generation task here, so it is the task of the decoding algorithm(decoder) to sample the probability distributions to generate the most likely sequence of words. Decoding the most likely output sequence requires searching through all possible output sequences based on likelihood. In order to resolve this problem in inference beam search, topK and nucleus sampling has been used. All of these are techniques to sample from language models as explained below.

1. Greedy Search simply selects the word with the highest probability as the next word. $w_t = argmax_w P(w|w_{1:t-1})$ is the equation for greedy search at each time step t. One disadvantage of greedy approach is the it often misses high probability words hiding behind low probability words. The below diagram shows such a scenario:

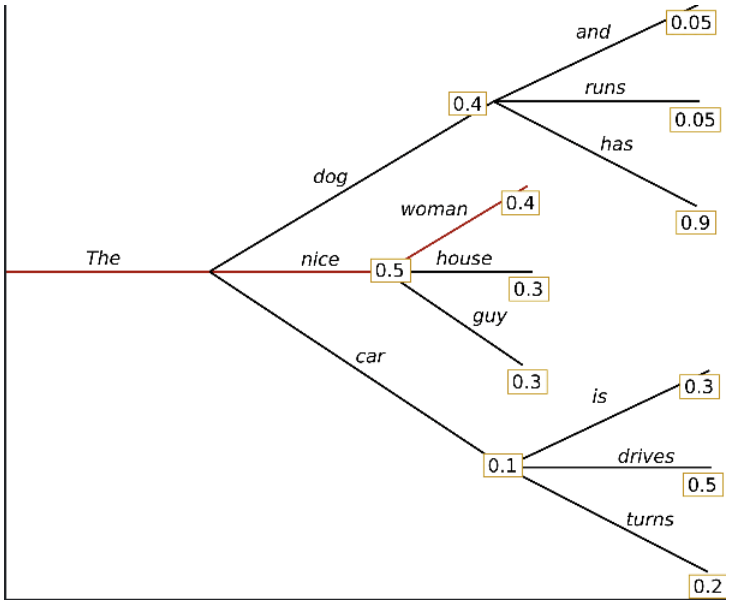

Figure 10.3: Visual Illustration of Greedy Search

As seen the word "has" has a higher conditional probability of 0.9 here which is behind the word "dog" of 0.4. Thus the sequence of "the dog has" is missed by greedy search.

2. Beam search thus came into the picture due to the inability of greedy decoding

to provide best suited samples. Instead of greedily choosing the most likely step, beam search increases all the possible next steps and keeps the "k" most likely where k is super specified parameter. It control the number of beams throughout the sequence of probabilities. One way of thinking about it can be that greedy search is beam search with k=1. Increased beam width results in better performance as there are multiple candidates which increase the likelihood of better matching at target sequence but it also slows down the decoding speed. One downside of beam decoding is that it suffers from repetitive generation. Also high quality human language does not follow the distribution of high probability next words which also means humans do not always use the same word even if the context is same.

3. Top k sampling primarily means sorting by probabilities and zero-ing out the probabilities below a threshold. It improves the quality by omitting a part of the distribution below threshold which makes it less likely to go off topic. This provides better accuracy but in some cases there are words which are good candidates and can be sampled from a broad distribution while in other situations a narrow distribution is preferred to sample next words as shown in below figure.

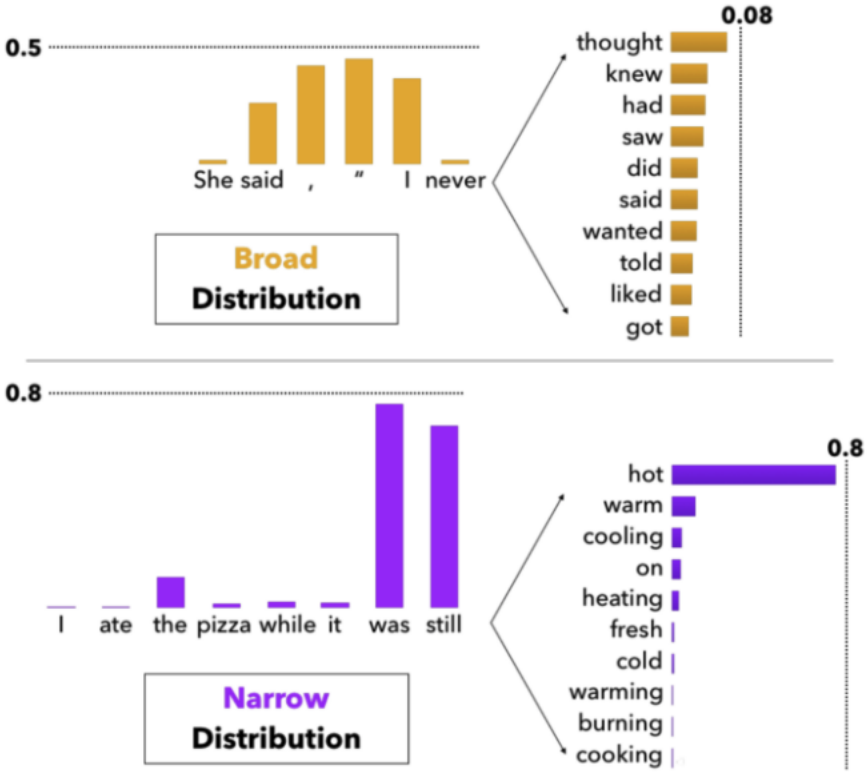

Figure 10.4: Narrow and broad distribution

4. Top p sampling also known as nucleus sampling. Instead of sampling only from the most likely k words, Top p chooses from the smallest possible set of words whose cumulative probability exceeds the probability p. The probability mass is then redistributed among the next set of words. In this way the size of set of next words either dynamically increases or decreases on the basis of next word's probability distribution. In a broad distribution it might take more samples to exceed the probability mass value while in a narrow distribution it generally takes less likely words or candidates.

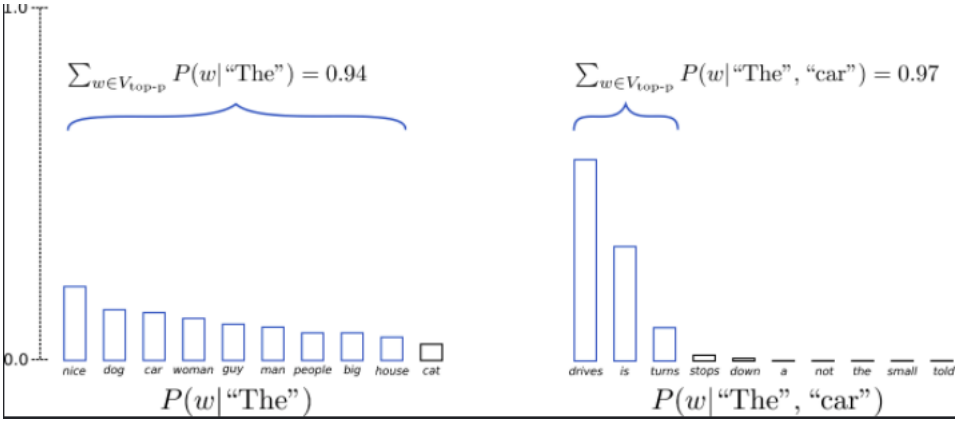

Figure 10.5: Top p sampling

In the above example p is set to 0.92 or 92% of the probability mass, so Top p sampling picks the minimum number of candidates that are required to exceed the p value. In case of the first example 9 words are needed while in the second example which is probably froma narrow distribution only 3 words are needed.In our approach we have used Top-k and Top-p in combination.

10.3 Settings

The entire architecture has been built on $BERT_{BASE}$, which is the smaller of the two predominantly used BERT models. It has 12 encoder stacks also to maintain uniformity we have set the decoder stacks to 12 as well. The attention heads are fixed at 12. The sub-word selection algorithm used is Word piece embeddings like BERT. The vocabulary size is 30000. Accumulation steps are kept at 36 for gradient accumulation. The batch size is set to 2 and Adam optimizer has been used for training. The model has been trained for 4 epochs on 1 Tesla V100 GPU for around 190 hours or 8 days. The beam sizes used are 2,3,4 and the length penalty has been set to 1. For regularization we have used dropout [18] of 0.1 and label smoothing=0.1 [19].

10.4 Evaluation and Results

We have used a very common evaluation metric ROUGE [20] for analysing the results. ROUGE stands for Recall oriented understudy for gisting evaluation, it is essentially used for evaluating automatic summarization of texts as well as machine translation. It compares automatically produced summaries against a set of human summaries. Recall in context of rouge means how much of the reference summary the generated summary has been able to overlap which is nothing but

$$\frac{number - of - overlapping - words}{total - words - in - reference - summary} \tag{10.8}$$

But in terms of text summarization a machine generated summary can be extremely long capturing all words in reference summary but a majority of the words in the generated summary can be useless. This leads to the use of precision in evaluation which is determining how much of the system summary is relevant i.e.

$$\frac{number - of - overlapping - words}{total - words - in - system - summary} \tag{10.9}$$

ROUGE-N measures the unigram, bigram and trigram or other n-gram lexical overlaps.

ROUGE-L measures the longest matching sequence of words using LCS.

The table below gives the scores on our model tested on around 1650 test samples along with the current 3 SOTA models.

	ROUGE -1	ROUGE -2	ROUGE -L	ROUGE-AVG
ProphetNet	.442	.21	.41	-
PEGASUS	.441	.21	.41	-
BART	.440	.20	.40	-
Our model	.245	.05	.18	.15

Figure 10.6: Comparison of ROUGE scores between 3 current SOTA models and our model

The below table shows the summarized versions as generated by our model. We can see the model learns from the source and is summarizing the text, to a certain extent it has been successful in generating meaningful sequences but as the sequence length increases it loses the fluidity in it's generated sequences. Also the model can some problem in linking two sentences or jumping from one context to another and is quite unnatural in selecting the next sentence. However it effectively captures the most salient parts of the text.

Since ROUGE solely relies on the lexical overlaps of the terms and phrases between source and summary therefore in certain cases of abstractive summarization where the model paraphrases ROUGE scores might not be effective or the best indicator of evaluation. Though ROUGE is not the best metric we report these scores as they are the standard metric for evaluation for all summarization tasks. In all the results displayed below we have a comparatively high Rouge-Avg score but if we evaluate and analyse the results it's clear that certain summaries are more natural and coherent than the others.

Let's state for example the third summary in Figure 10.7 we see it has a comparatively lesser Rouge score than the second example. However if one analyses the third example we can see firstly it captures the meaning of the reference summary somewhat entirely. It paraphrases the first sentence into two sentences. One can notice that the generated summary jumps abruptly from one context to another, the sentence is syntactically correct in parts but is not true or factually correct. There has been some work in fact checking the generated summaries against reference using Reinforcement learning which is a scope for future work.

Upon looking at the first example in Figure 10.7 we see a summary which in my opinion is most natural and semantically coherent even though it does not have the highest Rouge measure. It is much shorter than the reference output and is quite good in condensing the source text. The transitions from one sentence to another are more

natural than the rest of the examples, also one major point that can be noticed in all the generated summaries are that they generated words which are legitimate words and no non-words are generated i.e logical generation of words. Another take away from the experiment is that we do not come across repetitive words which has been an age old issue with generative language models. Another example of a fairly good generated summary would be the last example where the first half of the summary is coherent and natural but the text becomes illogical and unstructured as the sequence gets longer. It converts the word "selfie" with "self ied" as selfie is not a part of the BERT. vocabulary and it generates the nearest natural word by breaking into word piece embedding.

I believe there is scope for improvement in the model as as the sequences increase in length we can notice the summaries tend to become less coherent and less natural. Also another drawback is the abruption and disambiguation in text as the model jumps from one sentence to another or from one context to the next one.

REFERENCE SUMMARY	GENERATED SUMMARY	ROUGE-AVG
a wiki is run using wiki software, ot herwise known as a wiki engine. A wiki engine is a type of content ma nagement system, but it differs fro m most other such systems, includi ng blog software, in that the content is created without any defined owne r or leader, and wikis have little inh erent structure, allowing structure t o emerge according to the needs of the users. There are dozens of differ ent wiki engines in use, both standa lone and part of other software, suc h as bug tracking systems. Some wi ki engines are open source, whereas others are proprietary. Some permit control over different functions (lev els of access); for example, editing rights may permit changing, adding, or removing material others may pe rmit access without enforcing acces s control other rules may be impose d to organize content	wiki software allows state of use le vel control rules some rules include allowing access to content others ha ve restrictions comes from software computers however some may be a nonymous software computers som e have power some have click grap hic systems others allow other soft ware software files like specific soft ware or others have limited leaders hip	0.62
kevin la died in the angeles national forest on monday	kevin la had been hiking in the los angeles national forest in barker national forest happened monday during california park friends of friends started diving into pool friends had died after jumping into pool teenager had been killed monday california in california in california in california in california in area of lake angeles california	0.78
new brittany jones and her father have been located at a wisconsin hospital police say new an abduction alert was issued for jones after she was taken from a st louis hospital police say jones who underwent a kidney transplant in was in the custody of illinois child protective services	brittany jones was in a hospital at st louis hospital says police in wisconsin she underwent kidney transplant incident suspect was taken by father dana from hospitals was being treated police say missing is not receiving care police a hospital in chicago police say she was not getting care at home	0.40
scott sinclair and former coronation street actress helen flanagan shared photos of bangkok the pair are enjoying a daybreak now the football season has ended	scott sinclair and helen flanagan have been sharing holiday photos a number of the pair have been sharing a selection of self ied since holiday in bangkok has a in trade target behaviour as i moved from swansea a number of snaps sharing a number of snaps with helen then posted a loan from sunderland from swansea and posted	0.35

Figure 10.7: Generated Summaries with Rouge scores

CHAPTER 11: CONCLUSION

The novelty of the architecture lies in the implementation of using BERT in a sequence to sequence framework [6]. We have used Transformer decoder as the decoder along with BERT as encoder. We implement a two stage decoding process using a learning objective which is a summation of the objective of decoder phase 1 and decoder phase 2. The model has been trained end to end without another feature engineering. Also the use of search and sampling in inference to refine the results has been a new approach in our model. ROGUE scores show that the model learns from the summaries and ROUGE-1 score on our test dataset is around 0.25 which is around 25, the current state of the art for summarization is around 44. I believe we can further much better if we trained the model for longer (the current model is trained on just 4 epochs).

I believe even though we used the architecture on a text summarization task but it can be implemented for other language generation tasks such as neural machine translation, question-answering to name a few.

I believe we could further improve the model by using a Stochastic beam search [21], which is a stochastic variant of beam search which draws samples without replacement from a sequence model. There has been success using Stochastic beams in sequence models like translation and image captioning. Another approach could be using adversarial learning along with transformers, where in problem where we jointly train two systems, a generative model to produce response sequences,and a discriminator analogous to the human evaluator in the Turing test to distinguish between the human-generated dialogues and the machine-generated ones. The idea is similar to GAN's but is not exactly the same [22] [23].

REFERENCES

[1] N. Lintz, "Sequence modeling with neural networks (part 2): Attention models."

[2] J. Alammar, "The illustrated transformer."

[3] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," *CoRR*, vol. abs/1704.04368, 2017.

[4] R. Nallapati, F. Zhai, and B. Zhou, "Summarunner: A recurrent neural network based sequence model for extractive summarization of documents," *CoRR*, vol. abs/1611.04230, 2016.

[5] W. Kryscinski, R. Paulus, C. Xiong, and R. Socher, "Improving abstraction in text summarization," *CoRR*, vol. abs/1808.07913, 2018.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018.

[8] D. Britz, "Recurrent neural networks tutorial, part 1 - introduction to rnns."

[9] D. Britz, "Recurrent neural networks tutorial, part 3 - backpropagation through time and vanishing gradients."

[10] J. Alammar, "Visualizing a neural machine translation model (mechanics of seq2seq models with attention)."

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[12] R. Draelos, "The transformer: Attention is all you need."

[13] C. McCormick, "Bert word embeddings tutorial."

[14] D. Chen, J. Bolton, and C. D. Manning, "A thorough examination of the cnn/daily mail reading comprehension task," *CoRR*, vol. abs/1606.02858, 2016.

[15] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser, "Universal transformers," *CoRR*, vol. abs/1807.03819, 2018.

[16] S. Su, K. Lo, Y. T. Yeh, and Y. Chen, "Natural language generation by hierarchical decoding with linguistic patterns," *CoRR*, vol. abs/1808.02747, 2018.

[17] J. Gu, Z. Lu, H. Li, and V. O. K. Li, "Incorporating copying mechanism in sequence-to-sequence learning," *CoRR*, vol. abs/1603.06393, 2016.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[19] C. Szegedy, S. Ioffe, and V. Vanhoucke, "Inception-v4, inception-resnet and the impact of residual connections on learning," *CoRR*, vol. abs/1602.07261, 2016.

[20] K. Ganesan, "ROUGE 2.0: Updated and improved measures for evaluation of summarization tasks," *CoRR*, vol. abs/1803.01937, 2018.

[21] W. Kool, H. van Hoof, and M. Welling, "Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement," *CoRR*, vol. abs/1903.06059, 2019.

[22] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *CoRR*, vol. abs/1705.04304, 2017.

[23] J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky, "Adversarial learning for neural dialogue generation," *CoRR*, vol. abs/1701.06547, 2017.