



**Mixed-Language Sentiment Analysis: A Deep Learning
Approach**

Shravan Chandrashekharaiiah

18043038

Supervised By:

Dr. Nikola Nikolov

**Submitted to the University of Limerick, August 2020 in
partial fulfilment of the requirements for the Degree of
Master of Science in Software Engineering**

(Data Analytics)

Table of Contents

Table of Contents	2
List of Figures	4
Declaration	6
Acknowledgement	7
Abstract	8
1. Introduction:	9
2. Literature Review:	11
2.1 Deep Learning:	11
2.2 Natural Language Processing:	12
2.3 Language Modeling.....	13
2.3.1 Various Usage of Language Modeling:	14
2.4 Neural Networks:	15
2.4.1 Recurrent Neural Networks:	15
2.4.2 Long Short-Term Memory (LSTM)	20
2.4.3 Bidirectional RNN (BRNN):	22
2.5 Encoder-Decoder	22
2.5.1 Drawbacks:	23
2.6 Attention Mechanism:	24
2.6.1 Drawbacks:	25
2.7 Transformers:	25
2.7.1 Encoders and Decoders:	27
2.7.2 Working of the Encoder:	27
2.7.3 Working of Decoder:	34
2.8 BERT (Bidirectional Encoder Representations from Transformers):	36
2.8.1 Types of BERT	36
2.8.2 Model Inputs:	37
2.8.3 Model Outputs:	37
2.9 Multi-Lingual BERT (M-BERT):	38
3. Experiment and Analysis:	39
3.1 Data:	39
3.1.1 Data Pre-Processing and Extraction:	39
3.1.2 Data Exploration:	41
3.2 Google Colab:	43

3.3 TensorFlow:	43
3.4 Training Model:	44
3.4.1 Load Dataset:.....	44
3.4.2 Tokenization:.....	45
3.4.3 Building a Model:	46
3.4.4 Model Compilations:	49
3.4.5 Model Training:	50
3.4.6 Saving:.....	51
3.4.7 Plotting:	51
3.5 TensorFlow Serving:	52
3.5.1 Load the Model:.....	52
3.5.2 TensorFlow Serving:	52
4. Web Application:	60
4.1 Flask:.....	60
4.1.1 HTML:	61
4.1.2 API Request:	62
4.1.3 TensorFlow Serving API Server:	63
4.1.4 TensorFlow Serving API:.....	63
5. Limitations and Future Work:	64
5.1 Limitations:	64
5.1.1 Polarity distribution:	64
5.1.2 Mixed Language:	64
5.2 Future Work:	65
5.2.1 Pre-Processing:	65
5.2.2 Hyper-Parameter Tuning:.....	65
5.2.3 Hardware Infrastructure:	66
Conclusion	67
References.....	68

List of Figures

Figure 2.1: Recurrent Neural Network (Mittal 2019)	15
Figure 2.2: Activation function (Stanford 2019)	16
Figure 2.3: Recurrent Neural Network Internals (Stanford 2019)	16
Figure 2.4: Types of RNN (Stanford 2019)	18
Figure 2.5: Limitation of RNN with lengthy sequence (Pai 2020)	19
Figure 2.6: Gradient Clipping (Stanford 2019)	20
Figure 2.7: RNN and LSTM (Hoffman 2018)	20
Figure 2.8: LSTM Gates (West 2020)	21
Figure 2.9: Bidirectional RNN (Lee 2018)	22
Figure 2.10: Encoder-Decoder (Kostadinov 2019)	23
Figure 2.11: Attention Mechanism (Olah and Carter 2016)	24
Figure 2.12: Visualization of Attention for text sentence (Olah and Carter 2016)	25
Figure 2.13: Transformer (Alammar 2018a)	25
Figure 2.14: Internals of Transformer (Alammar 2018a)	26
Figure 2.15: Encoder-Decoder Stack inside Transformer (Alammar 2018a)	26
Figure 2.16: Individual Encoder and Decoder Unit (Alammar 2018a)	27
Figure 2.17: Working of Encoder (Alammar 2018a)	27
Figure 2.18: Details of Each encoder Unit (Alammar 2018a)	28
Figure 2.19: Self-Attention – Score calculation (Alammar 2018a)	29
Figure 2.20: Self-Attention – Softmax (Alammar 2018a)	30
Figure 2.21: Self-Attention – Final Step (Alammar 2018a)	31
Figure 2.22: Visualization of multi-head self-attention (Alammar 2018a)	32
Figure 2.23: Order of Sequence (Alammar 2018a)	32
Figure 2.24: Layer Normalization (Alammar 2018a)	33
Figure 2.25: Layer Normalization at Decoder Stack (Alammar 2018a)	34
Figure 2.26: Working of Decoder (Alammar 2018a)	34
Figure 2.27: Softmax Layer in Decoder Stack (Alammar 2018a)	35
Figure 2.28: Types of BERT (Alammar 2018b)	36
Figure 2.29: Model Inputs (Alammar 2018b)	37
Figure 2.30: Model Output (Alammar 2018b)	37
Figure 3.1: Pre-Processing Steps	40
Figure 3.2: Polarity Distribution of Training samples	42
Figure 3.3: Polarity Distribution of Validation samples	42
Figure 3.4: Loading dataset helper function	45
Figure 3.5: Tokenization Example	46
Figure 3.6: Dropout working (Maklin 2019)	47
Figure 3.7: Summary of the Model	48
Figure 3.8: Helper function to create BERT model	48
Figure 3.9: Model compilation	50
Figure 3.10: Code Snippet for Training the mode	50
Figure 3.11: Loss vs Validation Loss	51

Figure 3.12: Accuracy vs Validation Accuracy	51
Figure 3.13: Loading the Model	52
Figure 3.14: TensorFlow Serving	53
Figure 3.15: TensorFlow Serving Architecture (“Architecture TFX” 2020)	56

Figure 4.1: Page 1: with input text	61
Figure 4.2: Page 2 with Prediction	61
Figure 4.3: Page 3 list of tweets and its Sentiment	62
Figure 4.4: API request lifecycle	62
Figure 4.5: TensorFlow Serving Start Command	63
Figure 4.6: API request for TensorFlow Serving	63

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Shravan Chandrashekharaiiah August 26th, 2020

Acknowledgement

First, I would like to sincerely thank my supervisor Dr. Nikola Nikolov who guided me and gave me advice on my thesis.

I would like to thank my family who encouraged me and supported me during my studies for the whole year.

Finally, I would like to thank Dr. Michael English and all the lecturers in the CSIS department who helped and supported me sharpen my knowledge and skills throughout this Masters course.

Abstract

At many times it is often seen that a non-English speaker tends to speak or write in mother tongue and then mix it with English explanation. It is also seen at some times that while speaking in mother tongue to stress on something or to say that is known to many people around, English words are used. This can also be seen in the social media platforms like Twitter, Facebook, etc. Published content of the user in any of the social media platform may contain more than one language.

There are many applications on text data that is available and still in research phase. Most of the NLP related application contains a single language as the input to the application. Such example application could be sentiment analysis on a certain domain. But with the same approach dealing with the input data that contains one or more language can we get the same result?

In this paper we are analysing the performance of the sentiment analysis problem dealt on a mixed language dataset. To do the analysis we are going to use the deep learning approach. Also, will be addressing if the Multi-Lingual BERT be able to perform well on mixed-Language dataset.

Chapter 1

1. Introduction:

It is nowadays very common that all use social media in one or the other form. Social media does for the sure impact the people's opinion on certain topics of discussion. It is a platform for the people, by the people, and to the people. Thus, things that are discussed in such platforms do impact the people's insights towards the topic of discussion. In such a case, certain people or the organization would like to know their status in social media by looking at or analysing the opinions about them. Same as in the case of the current Covid-19 situation. Government or any organization which is in the interest of knowing the opinions that are spreading in social media looks to get an insight. This is possible with the sentiment analysis technique.

Sentiment analysis is a technique in the Natural Language Processing field where, given a text sentence, its sentiment behind it will be identified. Sentiment could be positive, negative, neutral, et. It is a supervised machine learning process, wherein there, the machine learning/deep learning model is trained on a dataset that is marked according to their classification. In another way, the dataset will be labelled positive, negative, neutral, etc, for each sentence that is involved in the training process. The model then learns the context of the words that are used for the labelled and starts to map each word to the label that it is given.

In sentiment analysis often dataset collected will be of only one language alone. There will be no mixture of the language. This is a very common task in the field of natural language processing. English dataset is very easy to obtain and can train a model easily with the source and model that are available. BERT (Devlin *et al.* 2019) provides a start of the art method to solve many NLP tasks.

Social media platform like twitter is a very popular way of getting text data to perform NLP related tasks. Twitter allows us to tweet in multiple languages. At certain case, it also often to find the tweet in cross languages. The native language will be used to convey the message, in between in order to use the reference to some other context, other languages might also get used.

In this paper, we will analyse the performance of the Multi-Lingual BERT (Pires *et al.* 2019) on such cross-language text that we encounter on social media. The question that this paper tries to address is “**Multi-lingual BERT performance on a mixed-language text data using Sentiment Analysis**”. In this paper, we will be analysing the data which contains English text inside a Kannada¹ data. We will be analysing this for 71,000 texts articles which are labelled to be positive or negative.

Multi-lingual BERT is apparently the best choice in approaching this problem, as it supports 104 languages. It is a state-of-the-art method for NLP tasks. It uses the deep learning approach, which many neural networks connected to each other and attention mechanism to solve the contextual problem that is encountered on a very long text sentence.

¹ <https://en.wikipedia.org/wiki/Kannada>

2. Literature Review:

2.1 Deep Learning:

Deep learning is a branch of Machine Learning, where neural networks are built to connect between the input and output, and the parameter which is initialized with some random number is tuned to actually find a relation between the input and output values. It basically consists of three layers in it, input layer, hidden layers, and output layers. The number of input layers depends on the features that a particular problem consists of. Similarly, output layers depend on the type of output problem desires. One thing that can vary and also which is tricky is the hidden layers. Hidden layers are where the networks learn the relation between the input and the output. Hidden layers are composed of many nodes in each layer. Hidden layers (Karsoliya 2012) can be composed of just a single layer to many layers. Each layer consists of nodes connecting to each other. Each node is composed of two components, a linear component, and a nonlinear component. Both components are very important to find a good model for a given problem. If the node is composed of only a linear model, then for every other problem model would predict a very optimized straight line to fit the problem. Which is very bad when the problems are scattered across the plots. There has to be a nonlinearity to actually learn and adjust to the given problem tracing the output optimistically.

There are two steps involved in the training of deep neural networks (Liu *et al.* 2017). That is forward propagation and backward propagation. Forward propagation involves learning the parameters from the input to the output. Backward propagation involves updating the parameters according to the previously learned parameters in the forward propagation steps. This can be imagined as, one person performing the work based on the input and output that he has presented with, and another person giving the feedback the person who is doing the work so he could improve the work he is doing it.

Backward propagation optimizes the network by updating the learned parameters for each node in the hidden network. There are many optimization techniques that are available like Relu (Agarap 2019) (Gupta 2020), LeakyRelu (Dubey and Jain 2019) (Gupta 2020), Tanh (Sharma

et al. 2020) (Gupta 2020), Sigmoid (Dubey and Jain 2019) (Gupta 2020), Softmax (Dubey and Jain 2019) (Gupta 2020), Elu (Gupta 2020), etc.

Problems that are often seen in the deep learning approach are overfitting and under-fitting. Overfitting is the most common problem in the field of machine learning, with these networks almost tries to byheart the input and output values and become over-optimistic in the prediction. In deep learning techniques, careful thought has to be given in terms of the number of layers that are chosen and the number of nodes per each layer. Often this could lead to overfitting if many layers are used. Whereas when there are very few layers and nodes are used, underfitting might happen, because the network won't learn much from input and output values. Parameters that relate to input and output will not be sufficient to trace it accurately. There has to be an optimal number of layers and nodes when choosing a deep neural network. This has to be done with many trials and errors.

Hyper-parameters tuning (Maclaurin *et al.* 2015) will help overcome overfitting or underfitting problems as well. Hyper-parameter involves many things like learning rate (Maclaurin *et al.* 2015) (Smith *et al.* 2018), batch size (Smith *et al.* 2018), number of epochs (Smith *et al.* 2018), etc. Also, there are other methods like regularization. This will help the model to learn better and avoid overfitting problems. There are many regularization techniques

2.2 Natural Language Processing:

It is a branch in artificial intelligence, wherein deep learning model is built to work around the task involving text. Natural Language Processing involves tasks such as language translation, sentiment analysis, question answering, getting summary, fill in the blanks, etc. It is a field where constant research is going and improving a lot.

NLP has improved very immensely over a short period, from mapping words to a sequence to word embeddings to now unsupervised learning. Word embeddings (Rezaeinia *et al.* 2019) approach was a very popular approach, wherein there is certain vocabulary size is chosen and out of vocabulary is given a certain token and these embeddings will be trained over huge datasets. It is basically a vector representation of a word in multiple dimensions (Rekabsaz *et al.* 2017).

2.3 Language Modeling

Language Modeling is a technique where it involves the prediction of the word that comes after a sequence of words. techniques understand the context by probabilistic approach and as well statistical approach in order to determine the best possible word that can occur after a sequence of words. The model that is developed will get trained to learn the features of the language and the context inside the languages and then predict new sentences or the task it needs to be done.

There are many approaches to this, those are n-gram (Cavnar and Trenkle 2020) (Rouse 2020), unigram (Tillmann and Xia 2003) (Rouse 2020), bidirectional (Rouse 2020), exponential (Rouse 2020), etc. All the types depend on the types of tasks they are meant to perform. Also, it is technically different from each other from the amount of data that the model process to make predictions, although all of the models employ a probabilistic approach.

Dealing with language problems is a very difficult problem as the language itself is very complex in its structure and the sequence is unpredictable as it can take many different forms for the same meaningful sentence. Good language models are the one that handles all these cases well. It should understand long and short sentences, it should counter the punctuations to get the literal meaning, it should be able to get the nouns and pronouns, it should be able to refer to other words easily, it should be able to deal with sentences of different length, etc. It is a very complex task to deal with when it comes to NLP and hence the model would be complex enough to deal with all the above situations described.

Language Modeling is very important in the recent application which involves NLP. It is because of this we can see many different applications like search, voice assistance, autosuggestion works the way it does. It is used is a diverse group of industries like medical, finance, etc to name a few.

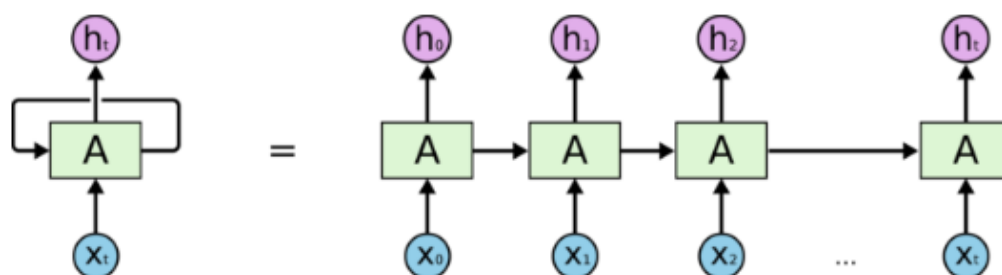
2.3.1 Various Usage of Language Modeling:

- **Speech Recognition:** This involves converting an audio input into a text output. This model will be trained to recognize the words of a specific language and identify each of them.
- **Machine Translation:** This involves, translation of one language text to another one. This is a very difficult task as data to train requires both sets of sentences, one is source language and another in target language. Also, there can be many different variations as well to provide. Data collection will be the biggest task.
- **Parts-of-speech tagging:** This involves marking various parts of a sentence into its categories based on the grammatical characteristics.
- **Parsing:** This involves scanning through the sentence and identifying a certain pattern in the sentences, like dates, address, entity recognition, etc. Also, this can be found in applications like spell-checking.
- **Sentiment Analysis:** This is the most common application that can be seen in the field of NLP. Here a sentiment of a sentence will be learned by the model based on the words that are used to represent the meaning. This can be tricky too, the model trained on a certain domain like entertainment can give bad results if it is used in the field of politics or other. This may be because there will be more sarcastic comments in the entertainment domain. This is mainly used to find people's opinions on a certain product of the campaign.
- **Optical Character Recognition:** This involves, recognizing the characters of language from an image. This is also a type of speech recognition where input is in different forms and output is in different forms. This also has a lot of applications like invoice recognition, certificates details, finance documents, etc.
- **Information Retrieval:** This involves training a specific model to identify patterns over a document. For example, a model trained to identify all the fields that are available on a certain CPA certificate, to fetch all the details of the student, grade, date of expiry, etc. And hence this model when fed with any CPA certificates will provide the text data that is matching the fields and its values from the certificate.

2.4 Neural Networks:

2.4.1 Recurrent Neural Networks:

As mentioned before in the deep learning section, neural networks are a process in which each node connected to each other in a predefined structure learns certain parameters that best represent the relationship between the input and the output. Recurrent neural networks (Stanford 2019) (Kombrink *et al.* 2011), on the other hand, are a special type of neural network, wherein it has a memory component built inside it. In this type of network, not just the input is fed to the network but also parameters that were learned in the previous step are fed. There are in total three interfaces to this network, one each for input and output and the third is the node parameters. Node parameters are passed from one step to another step. In this way, neural networks have the context of the previous steps that the network has seen. This accounts for the memory component that is mentioned above.



An unrolled recurrent neural network.

Figure 2.1: Recurrent Neural Network (Mittal 2019)

A - Activation function

h - output

X - Input

The activation function basically sets the tone/path of learning the relation between input and output. There are various types of activation functions that can be used in a sequence-related deep learning problem.

2.4.1.1 Activation Function Most Often Used in RNN:

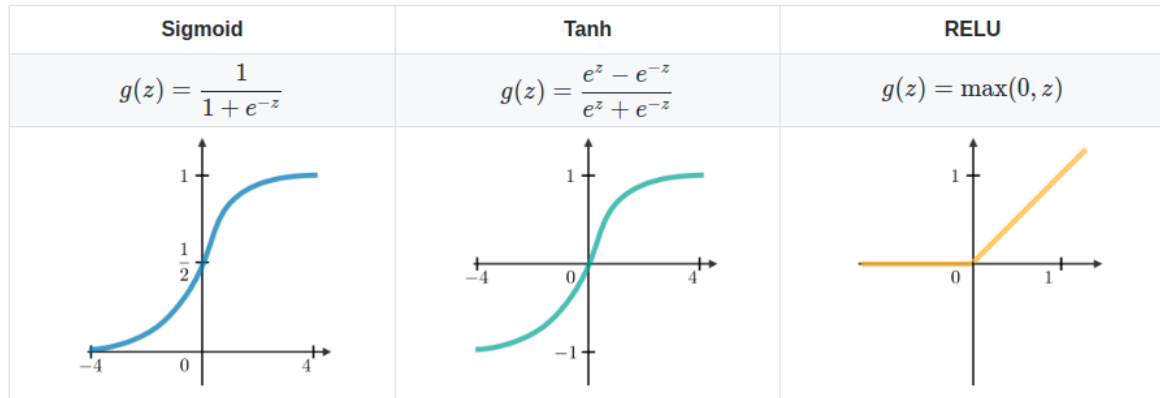


Figure 2.2: Activation function (Stanford 2019)

Sigmoid activation allows the neural network to keep the values between +0 to +1.

Tanh activation allows the neural networks to keep the values between -1 to +1.

Relu activation allows neural networks to keep only positive values, ignoring negative values.

Based on the task that is dealt with appropriate activation functions are to be used.

2.4.1.2 Details of the RNN Network:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \quad \text{and} \quad y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

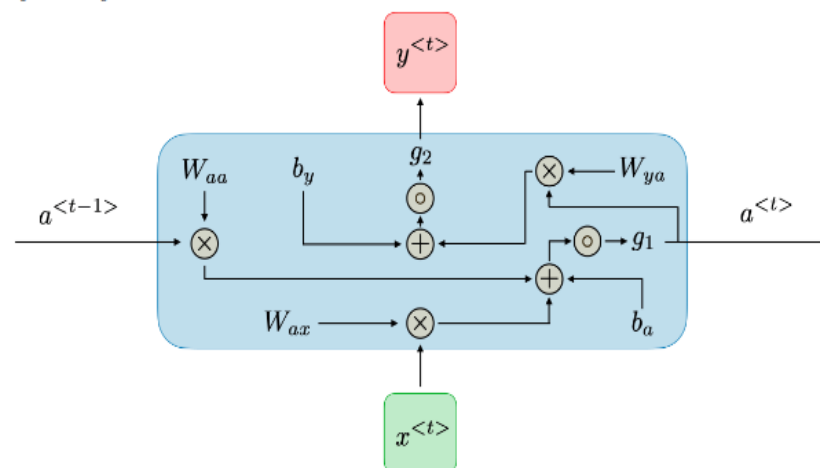


Figure 2.3: Recurrent Neural Network Internals (Stanford 2019)

2.4.1.3 Advantages and Disadvantages of RNN are as below:

Advantages:

- The network will be able to deal with the sentences/input of different lengths.
- As this is a sequence problem there is no increase in the size of the model and the length of the input increase. The same network is used in sequence.
- With the recurrence factor in the network, it provides the memory for the network to get the context of previous steps.
- As the data from previous steps are passed to the next steps, history is maintained. Parameters learned in each step are shared for the upcoming steps.

Disadvantages:

- As this is a sequence problem, the main disadvantage for the network is parallel computation is ruled out and hence delay in the learning process.
- Though the parameters learned in each step are passed to the next step, if the input length is very big, it will fail to counter the early set of words to make an impact at the end of the sentence.
- Often times, while analysing text in a sequence might be misleading in the prediction, this is because the context for might is at the end. The unidirectional network might fail very badly in such a case.

Types of RNN:

1. One-to-one
2. One-to-many
3. Many-to-One
4. Many-to-Many

2.4.1.4 Types of RNN:

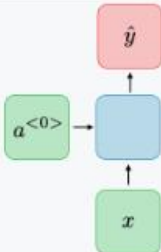
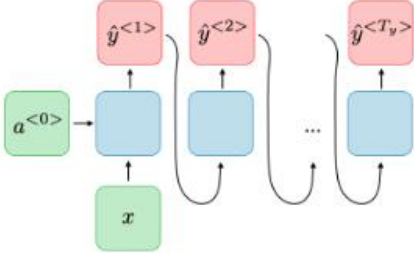
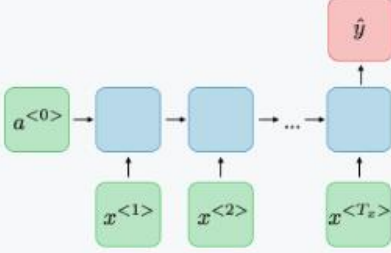
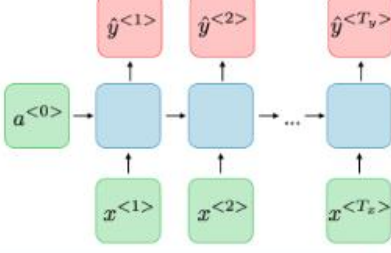
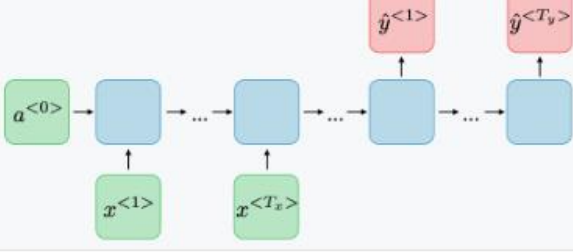
Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

Figure 2.4: Types of RNN (Stanford 2019)

2.4.1.5 Problems Encountered in RNN:

The most common problem that is seen in the sequential analysis is its vanishing/exploding gradients. This is seen mainly because of the fact that RNN may have a memory unit built inside it, but it fails to address this if the sequence that the network is dealing with is very long. This is because gradients might increase/decrease exponentially.

Any neural networks undergo two processes in order to learn the relation between the input and output data, those are forward propagation and backward propagation.

Forward propagation (Hirasawa *et al.* 1996) (Li *et al.* 2014) is a process where the input to each node is processed and activation function is applied to the calculated weights and bias and then it is passed on to finally get the output. Here in this process, the weights that were learned previously or randomly initialized weights in case of the first time are used to calculate the output. This step is nothing but a feed-forward neural network.

Backward propagation (Li *et al.* 2014) on the other hand often referred to as a gradient step, is a process where loss is calculated from the output generated and the weights and bias are adjusted to best reduce the loss and updated to each node of the network in each layer from output to the input layers. The vanishing gradient (Hochreiter 1998) (Wang 2019) is seen where the network is having a greater number of layers. In this case, the network fails to update the weights and bias to each node. It keeps reducing per each layer it reaches the input layers. This will cause the network to not learn anything.

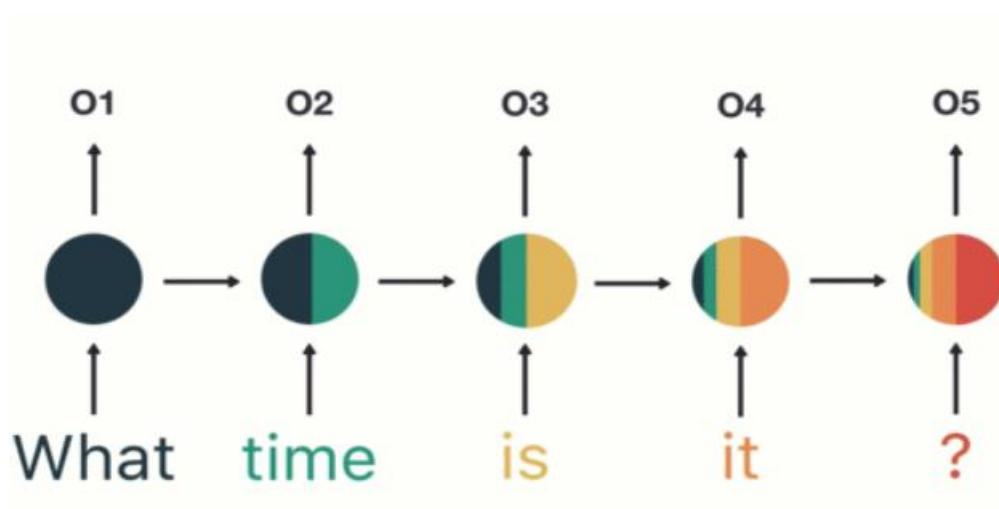


Figure 2.5: Limitation of RNN with lengthy sequence (Pai 2020)

From the above image, it is clearly visible that at time step 05 the context of the word at time step 01 is very less.

In terms of the exploding gradient, this can be avoided by a technique called gradient clipping (Xiangyi *et al.* 2020). It is a technique where threshold value will be kept and when the weights or bias exceeds this limit, it will be clipped to a certain value.

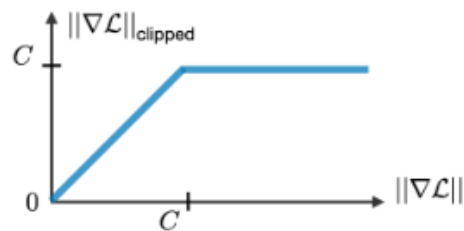


Figure 2.6: Gradient Clipping (Stanford 2019)

Vanishing gradient (Xiangyi *et al.* 2020) is challenging to deal with. Whereas there are modified versions of RNN that can deal with this problem of vanishing gradient.

2.4.2 Long Short-Term Memory (LSTM)

As the name itself suggests that, this network is able to hold the memory of previous steps for a longer sequence length. This network also deals better with vanishing gradients. In order to encounter the vanishing gradients, it employs three gates.

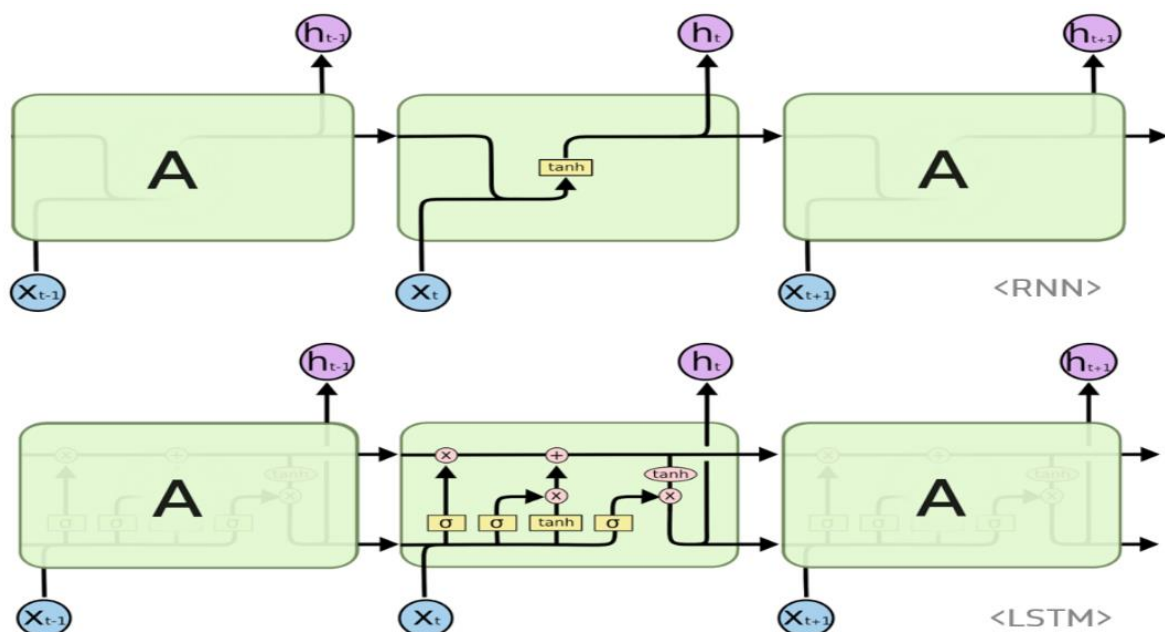


Figure 2.7: RNN and LSTM (Hoffman 2018)

The above image shows the internals of both RNN and LSTM (Sundermeyer *et al.* 2015). Visually we can notice that LSTM is more complex when compared to a simple RNN network. Another image representing the gates can be shown as below:

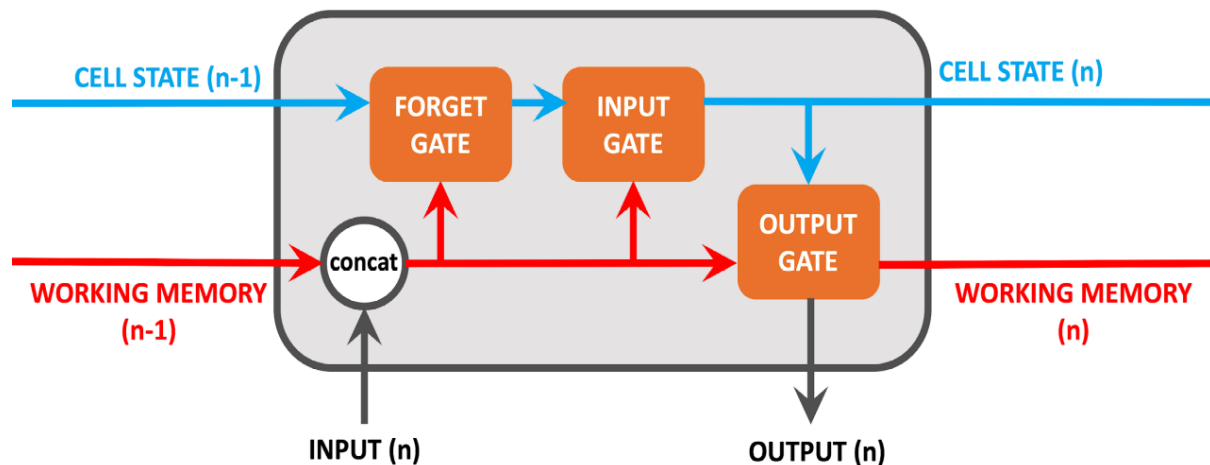


Figure 2.8: LSTM Gates (West 2020)

2.4.2.1 Forget Gate: This gate basically decides how much of the past that it should remember. Decides which information to be omitted before it can pass on to the next time step. It is decided by a sigmoid function. It looks into the previous state and current input and computes the function.

2.4.2.2 Input Gate: In this gate, there are two layers, one is the Sigmoid function and another is the Tanh. In Sigmoid function, it decides which to let out and in Tanh function, it decides which parameters take high importance or low.

2.4.2.3 Output Gate: These gates decide which part of the current state makes it to the output. The sigmoid function gets to decide this operation.

The only drawback of LSTM is that it is computationally expensive and it is sequential, ruling out the parallel computation.

2.4.3 Bidirectional RNN (BRNN):

This network is a variant of RNN. This can address the drawback that is mentioned before which is to get the information that is there in the latter part of the sequence. With this network dependency of data in the latter part of the sequence is eliminated.

BRNN (Schuster and Paliwal 1997) consists of two separate feedforward networks. One network process the sequence from the start and another network process the sequence from the end. Once the output from both the network is obtained, it is then combined and applied certain activation functions and the final output is obtained.

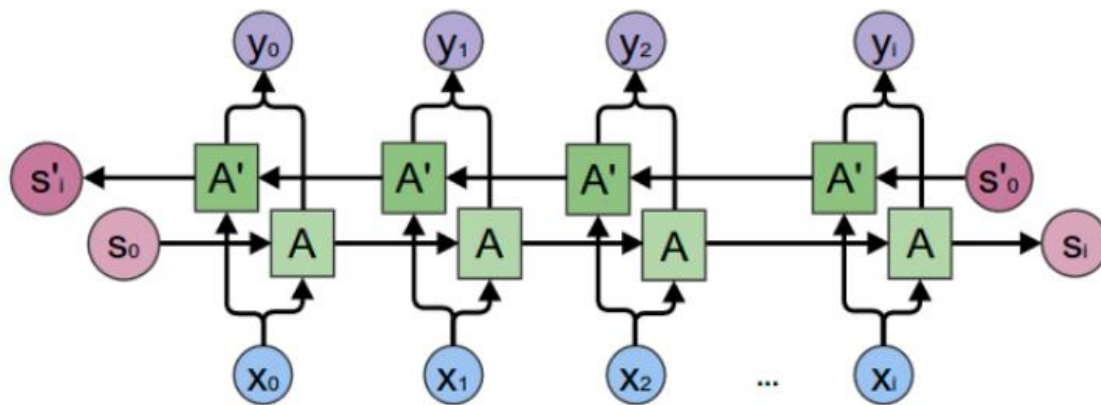


Figure 2.9: Bidirectional RNN (Lee 2018)

From the visual diagram above it is evident that the output from two separate feed-forward network one processing from start and another from the end will be concatenated and output is obtained. This network is rich with information from both the end and prediction will improve with information from the near future or near past of the sequence that is fed.

2.5 Encoder-Decoder

Encoder-Decoder is a network of type many-to-many RNN network (Sak *et al.* 2017). This network consists of two networks, one for the encoder and another for the decoder. Either of the networks is often composed of LSTM. A popular application of the encoder-decoder network is a machine translation. The encoder is often fed with the source language and the decoder is used to get the target language.

Encoder: This network consists of a fully connected RNN/LSTM network where the input for this network is the sequence of the source language. Encoder learns the weights and biases for the input language sequences.

Decoder: This is also a network that is similar to an encoder with a fully connected RNN/LSTM network. Only changes with that of the encoder are, weights and bias that is learned in the encoder step is then fed to the decoder as initialized weights and bias. This way there is a connection between input and output languages.

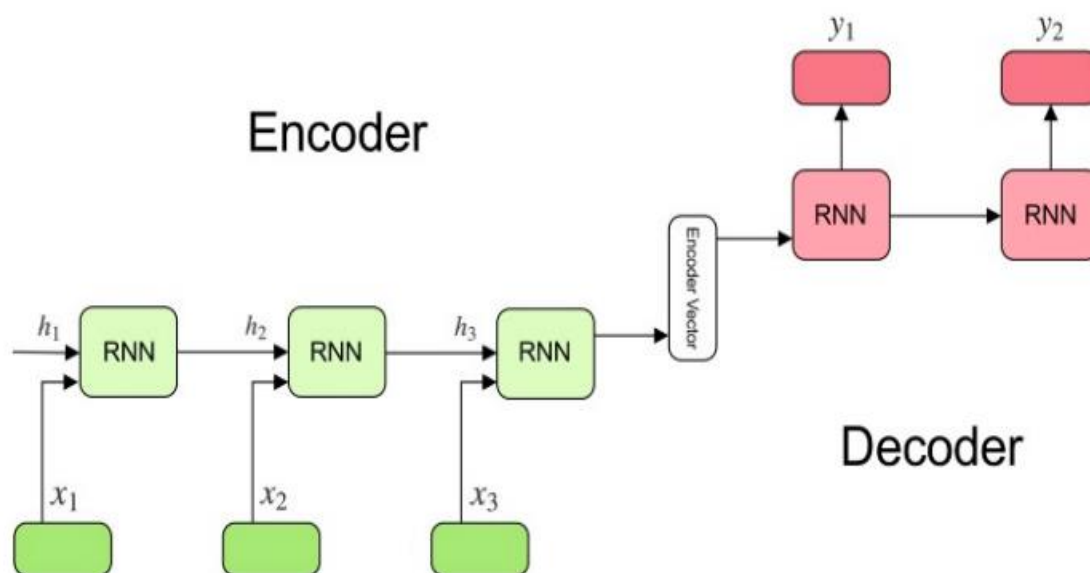


Figure 2.10: Encoder-Decoder (Kostadinov 2019)

From the diagram above it is evident that weights and bias that are learned from the encoder are fed as custom initialized weights and parameters to the decoder network.

2.5.1 Drawbacks:

The drawback of the network is that, if the length of the input sequence is long, the entire sequence will be represented with the same set of weights and bias. This will become hard to represent a complete sequence for the decoder network to decode. The performance of the network will get low as the length of the sequence keep increasing.

2.6 Attention Mechanism:

As mentioned in the above section, encoder-decoder alone fails to address the sequence which is very long. To address this problem, the Attention mechanism (Vaswani *et al.* 2017) (Bahdanau *et al.* 2015) is used. In this mechanism, the network pays attention to a specific part of the sequence to obtain the optimized result. This network has an encoder and decoder network, whereas decoder decodes once the encoder is done for each sequence component of the input. Whereas in the above section, once encoder is completely done with the input sequence then and only then the decoder will start to decode the output sequence. With attention, the mechanism starts decoding along with the encoder networks.

In this network, the encoder and decoder network are built using the Bidirectional neural network BRNN or BLSTM network.

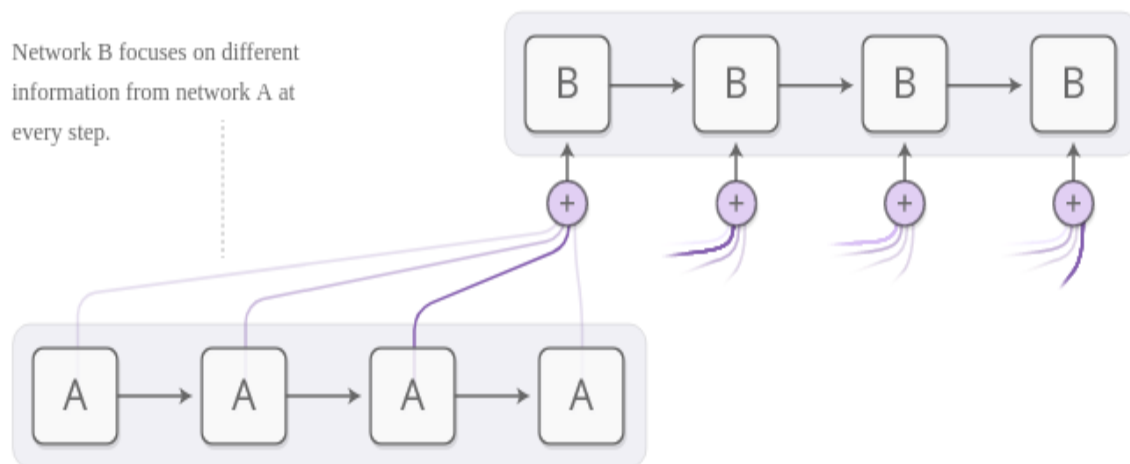


Figure 2.11: Attention Mechanism (Olah and Carter 2016)

From the figure, it is evident that not just one vector is passed from encoder to decoder network. Instead, it calculates a context vector to decode each word at a time.

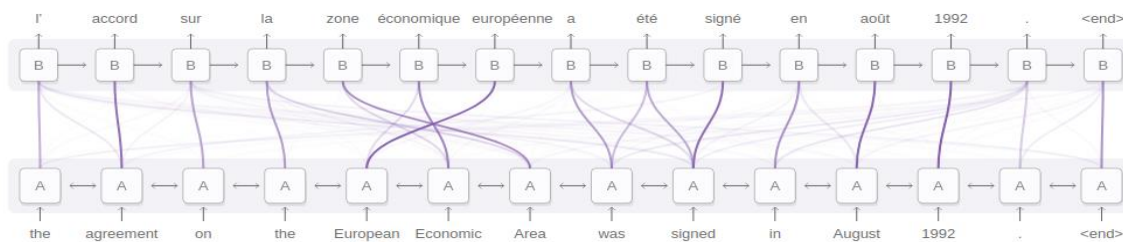


Figure 2.12: Visualization of Attention for text sentence (Olah and Carter 2016)

An example of the attention mechanism that the network implies in translating from source language to target language. This gives a clear picture of how the attention is given while decoding each target sequence.

2.6.1 Drawbacks:

This model works closely like human translation, part by part. Whereas encoder-decoder does encode for the entire input sequence and then decodes back to the target language. Attention mechanism despite being very good in sequence to sequence problems suffers in parallel computation because of the architecture that is very sequential in nature.

2.7 Transformers:

In order to encounter the parallelization, transformer (Alammar 2018a) (Joshi 2019) (Thiruvengadam 2019) (Vaswani *et al.* 2017) architecture was introduced. This model architecture is purely based on the attention mechanisms. With the help of Scaled Dot-Product Attention (Singh 2020) and Multi-Head Attention (Singh 2020) implementation, the model is able to process the input sequence parallelly and with attention mechanism intact.

If a transformer model is considered to be just black blocks it will appear as below:



Figure 2.13: Transformer (Alammar 2018a)

Now on opening the transformer model, it basically contains an encoder and decoder component construction. Will be going in-depth in further discussion.

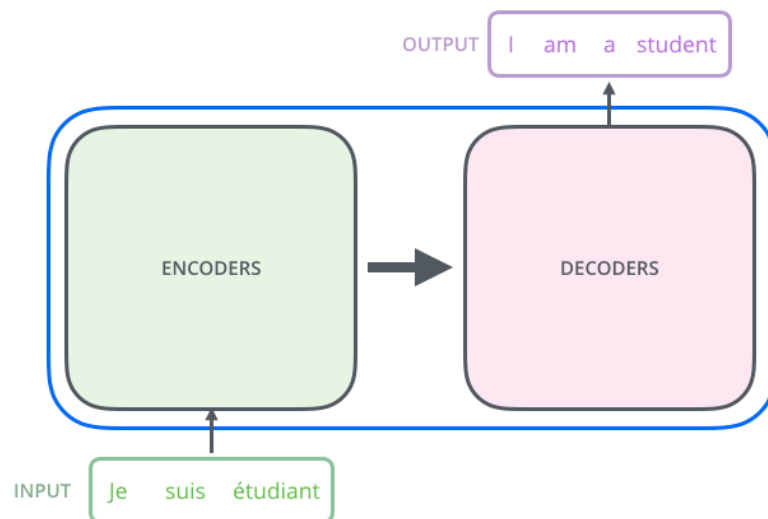


Figure 2.14: Internals of Transformer (Alammar 2018a)

From the paper which introduced the transformer say it encoder and decoder component above shown is a stack of many encoders and decoders, it can be represented as below:

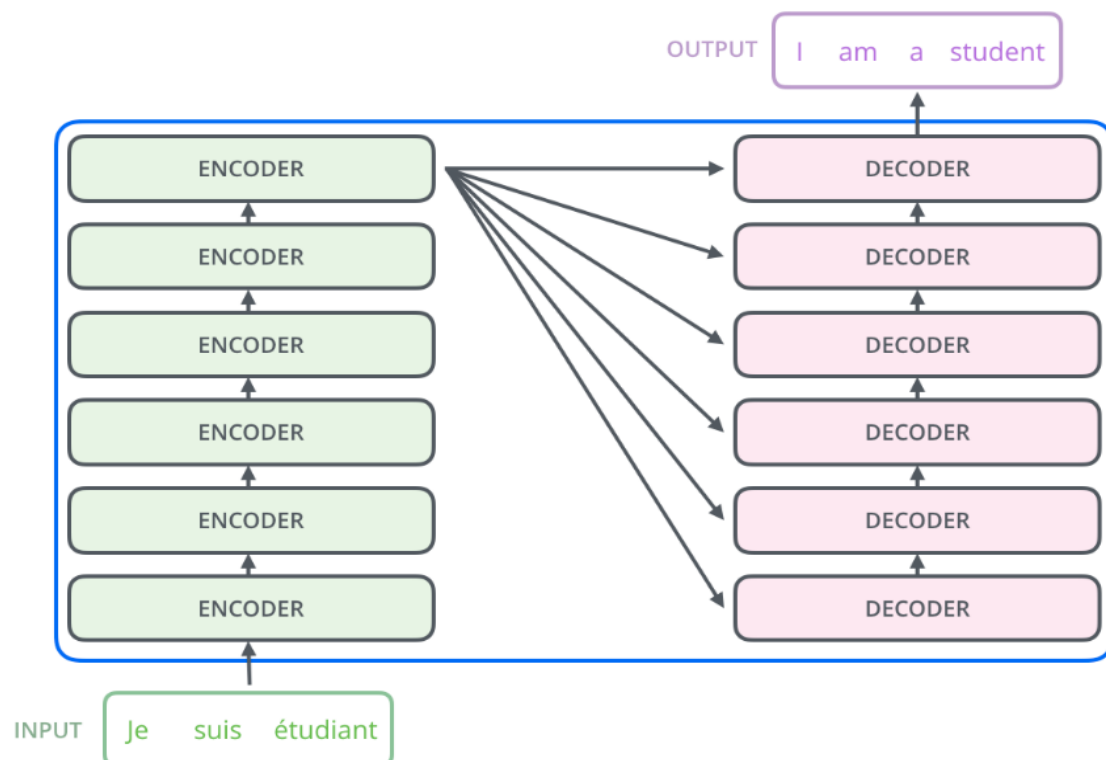


Figure 2.15: Encoder-Decoder Stack inside Transformer (Alammar 2018a)

2.7.1 Encoders and Decoders:

Both the component consists of identical layer of encoders and decoders as shown in the above figure. In this paper, the self-attention concept is used. This layer is present in both the encoder and decoder layers. Self-Attention will be covered in detail in the further section. Encoder and decoder representation with self-attention layer and feed-forward neural network can be represented as below:

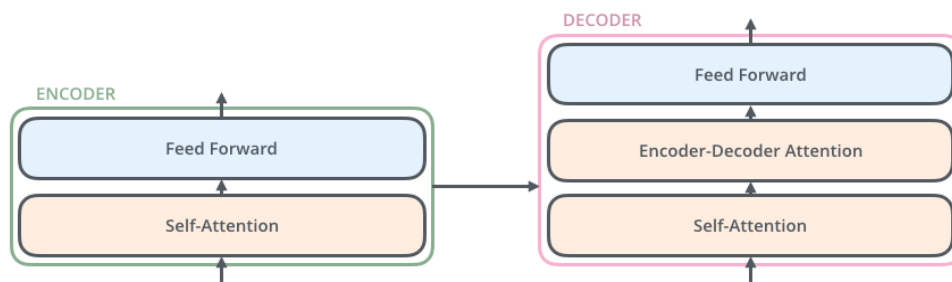


Figure 2.16: Individual Encoder and Decoder Unit (Alammar 2018a)

2.7.2 Working of the Encoder:

At the beginning of the process, the encoder is fed with the vectors obtained from the word embedding layer converting the text into a vector representation.

Conversion of text to vector happens only once at the beginning of the translation process. After that output from one layer is fed as an input to the other and it continues till all the encoder are served with the input from the preceding layers. The length of the input vector is a hyperparameter and can be varied according to the needs.

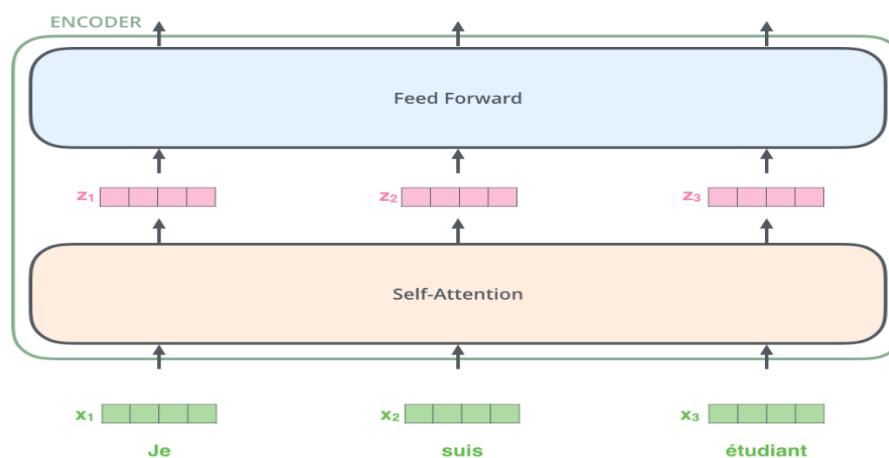


Figure 2.17: Working of Encoder (Alammar 2018a)

As we can see here from the figure, it is clear that each word is independent and goes on its own path. It has a little dependency on the self-attention layer but when it comes to the feed-forward layer, it has no dependencies and can process independently and in parallel making it fast.

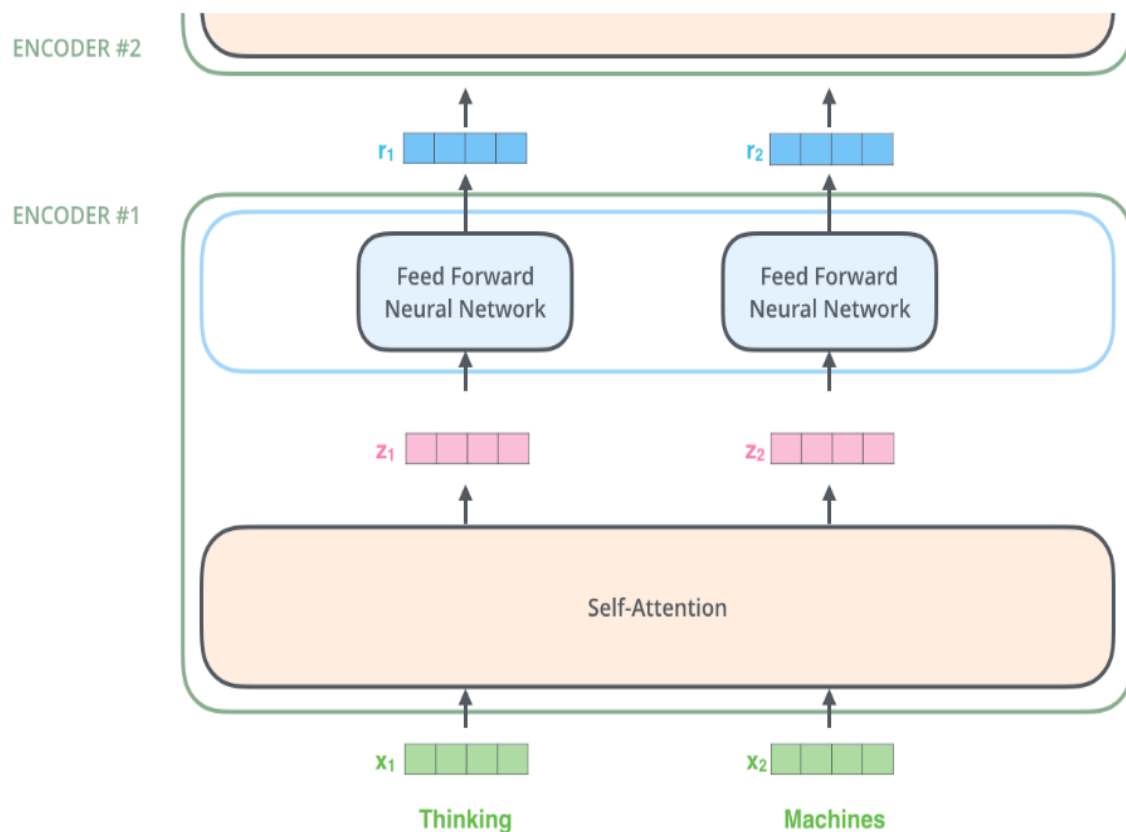


Figure 2.18: Details of Each encoder Unit (Alammar 2018a)

It is evident from the figure the working of the encoder layer, output vector coming from the self-attention layer is executed independently, and out of one encoder unit is then passed onto the next encoder unit.

2.7.2.1 Self-Attention:

This layer helps find the relevance between the input word sequence. It analyses each word and its correct relevant word for it and helps better encode each word. This involves four steps:

First step:

This step involves calculating three other vectors from the embedding vector of the input word sequence. The three vectors are “Query”, “Keys” and “Values”. These three vectors are obtained by multiplying three vectors that are obtained during the training phase. These three vectors help in calculating the attentions among the input sequence.

Second step:

This step involves in calculating the score. This means that each word in the sequence will get a scorecard has to where the dependency or the relevance lies more among the other words in the sequence. This is basically obtained by taking a dot product between the query and the key.

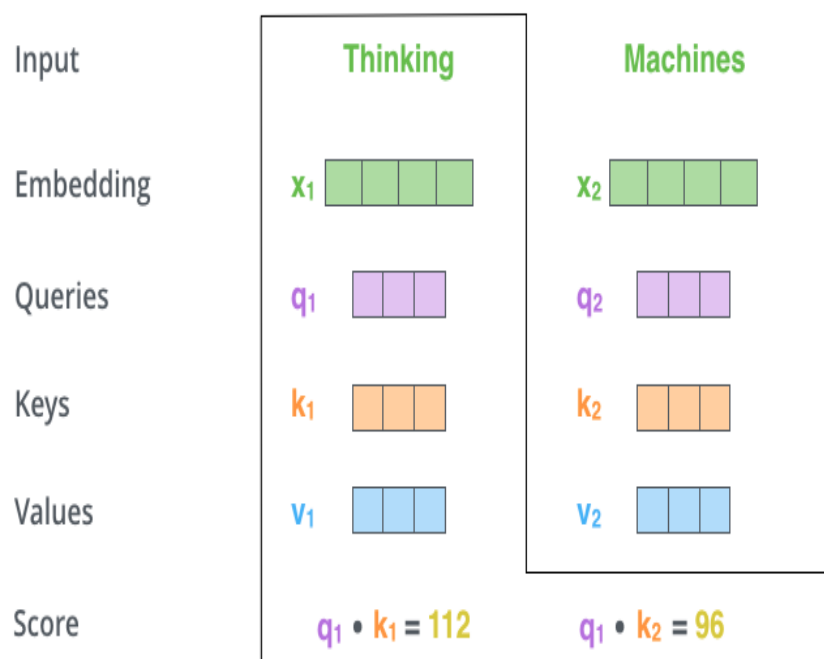


Figure 2.19: Self-Attention – Score calculation (Alammar 2018a)

From this image, we can see the score for the first word to the second is 96. This way each word gets a scorecard for other words in the sequence.

The Third and Fourth step:

This step involves dividing the score by 8 and applying Softmax to it. Softmax will modify the score in a way that all the values get added to 1. In this way, all the negative values are ruled out.

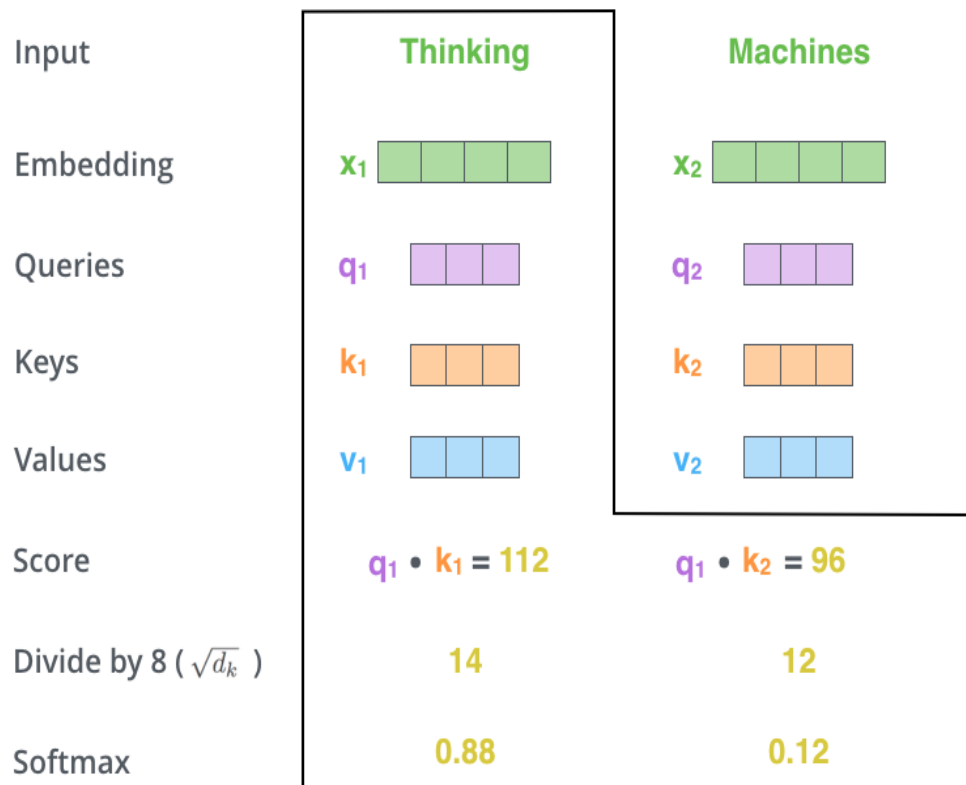


Figure 2.20: Self-Attention – Softmax (Alammar 2018a)

The number 64 is the dimension used in the paper for dimension of the “Key” vector. This could vary. Softmax score also gives an idea about the word relevance among all the words in the sequence with respect to the current word.

The Fifth and Sixth step:

This step involves multiplying the value vectors with the Softmax score. The reason behind this is to eliminate the word that is irrelevant to the current word. And sixth steps involved in adding up all the value vectors. This finally produces the self-attention layer’s output.

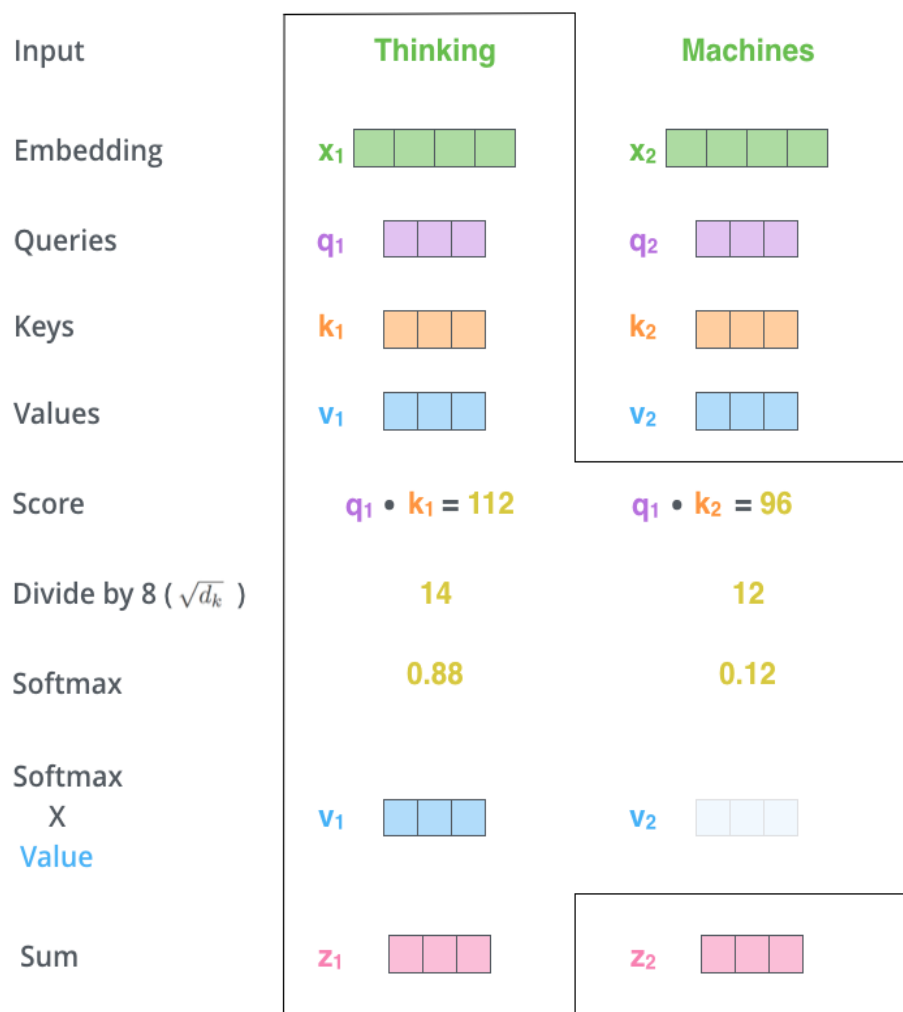


Figure 2.21: Self-Attention – Final Step (Alammar 2018a)

The paper further refines the self-attention layer to a multi-headed self-attention layer.

There are two advantages of this multi-headed implementation.

1. It allows the model to get an idea about the word sequence from many different angles. It allows us to pay attention in many different ways.
2. It will provide multiple representations of vectors to look into in order to come to a better attention model. Multiple sets of Key, Query, and Value vectors are initialized randomly and after training much different attention representation is obtained.

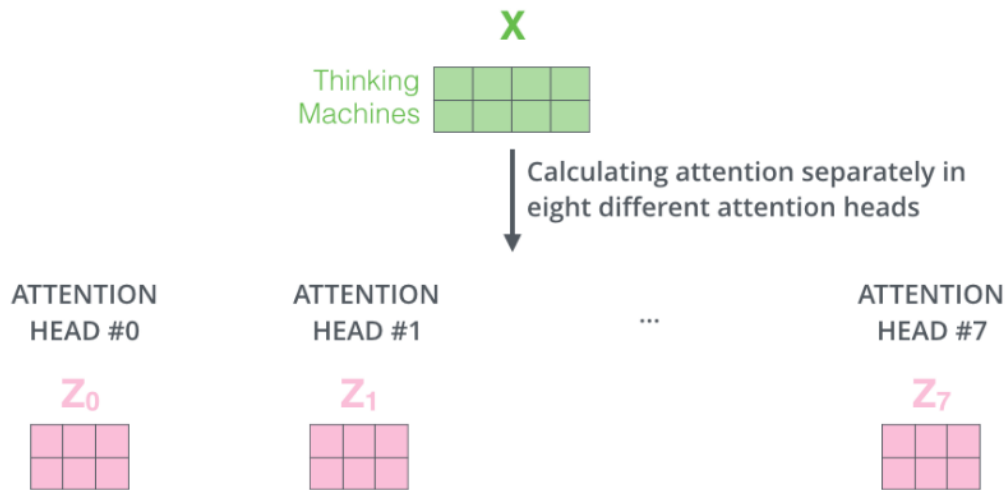


Figure 2.22: Visualization of multi-head self-attention (Alammar 2018a)

Order of the Sequence:

To address/influence the sequence of the word to account for the prediction of the model, transformer model uses positional encoding. Like said before, input to the transformer is fed in the form of an embedding vector. Apart from just passing the embedding vector, the transformer model also adds the positional encoding vector to it.

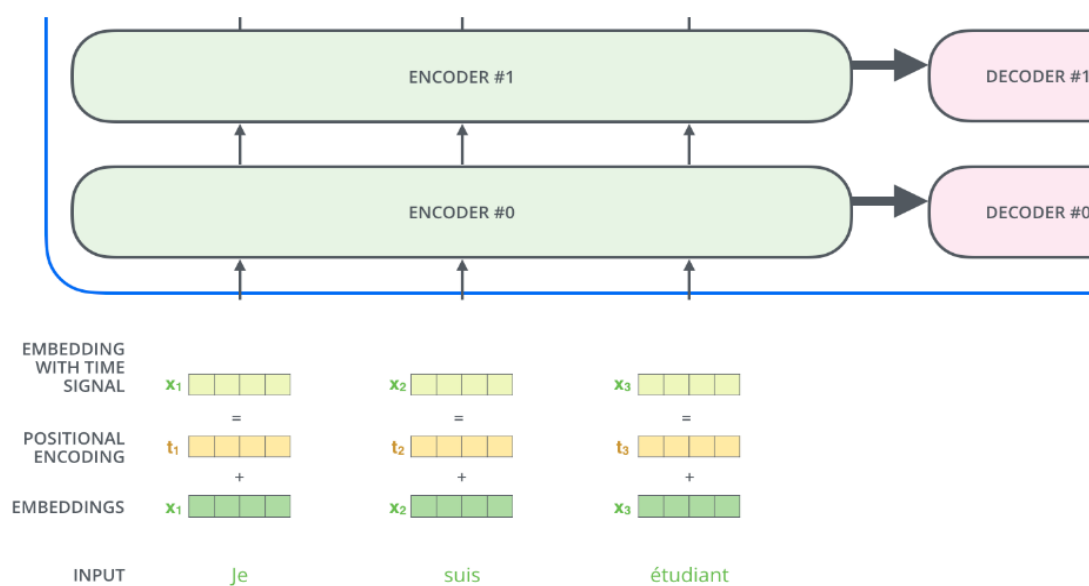


Figure 2.23: Order of Sequence (Alammar 2018a)

Adding the positional vector to the embedding vector further helps to locate the words and make the network well aware of its positions and the distance between each other word.

Layer Normalization:

Each layer inside the encoder is fed to other i.e. self-attention layer is fed to the feed-forward neural network through a normalization layer in between. And again, output from the feed-forward layer is fed to layer normalization.

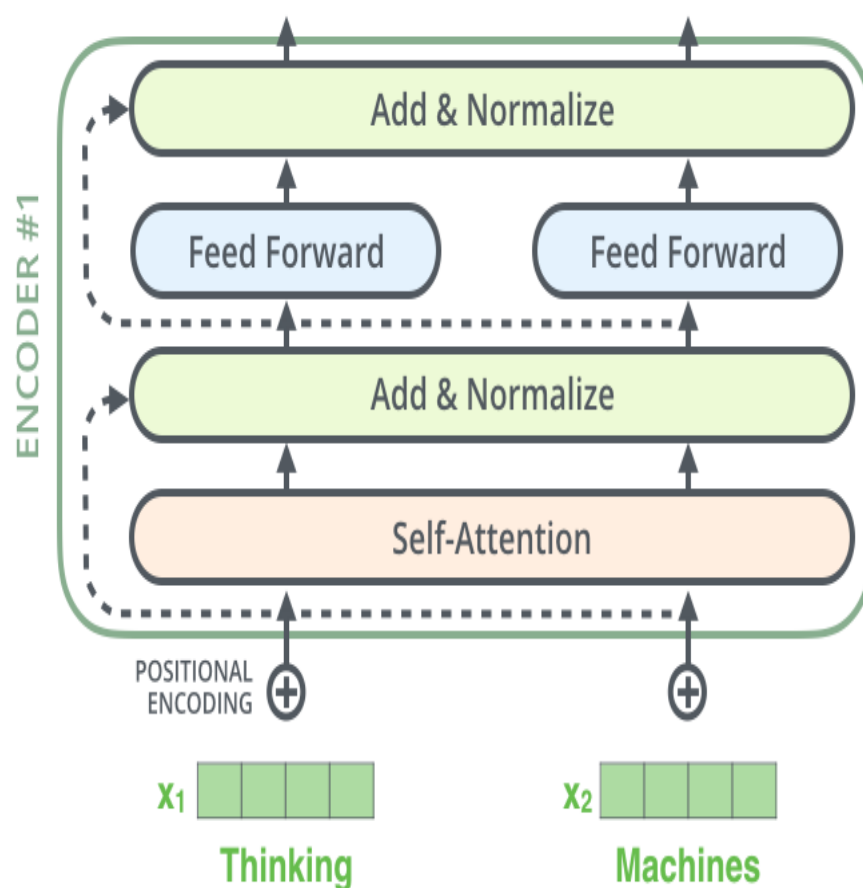


Figure 2.24: Layer Normalization (Alammar 2018a)

The same applies to the decoder stack as well. If the transformer is assumed to contain only two stacks of encoder instead of 6 as proposed in the paper, it will appear as below:

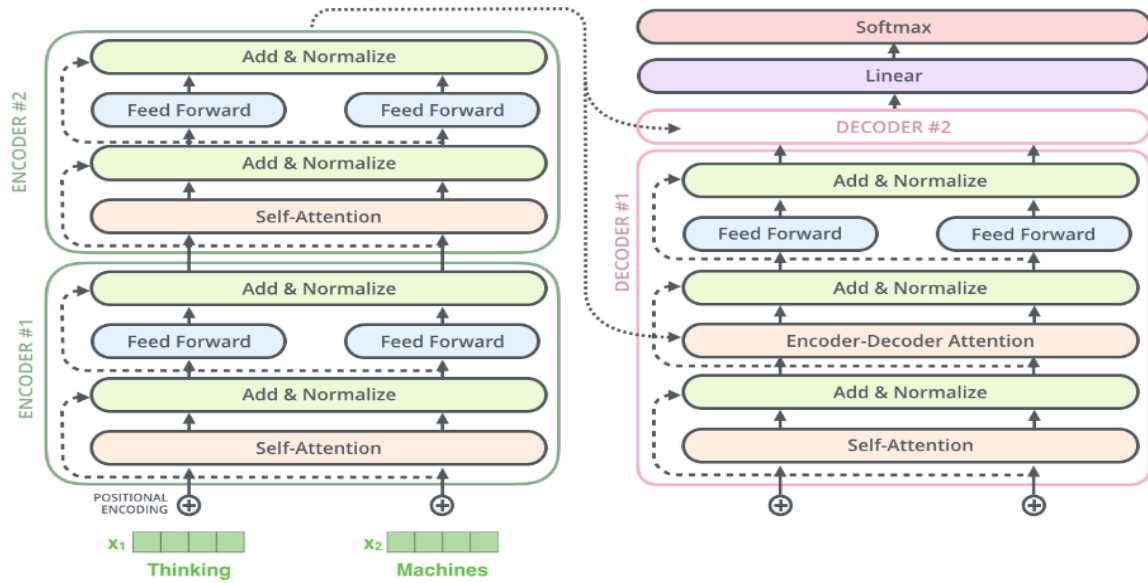


Figure 2.25: Layer Normalization at Decoder Stack (Alammar 2018a)

2.7.3 Working of Decoder:

From the above, it is familiar that how each component works together in order to obtain attention and importance of input sequence and its positions. Working with the decoder is also similar in many ways.

The output from the top of the encoder layer is now transformed into two vectors Query and Values. These are then fed into the decoder to predict the output. This process will look as below:

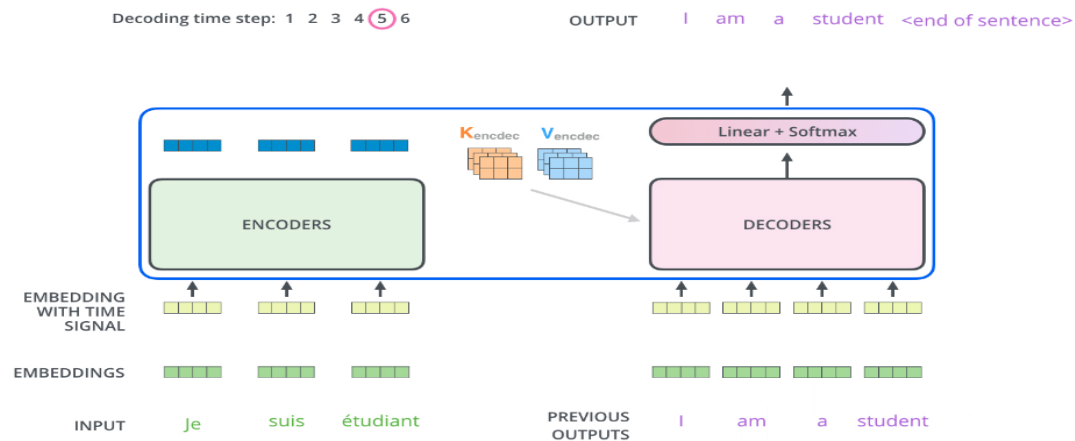


Figure 2.26: Working of Decoder (Alammar 2018a)

The process of prediction of the output sequence continues until the end of the sentence symbol is reached. Self-attention layer operation differs from the encoder self-attention operation in a way that, in the decoder, will block the future word sequence. It will only process the previous word sequences. It will be masked before it is passed into the Softmax layer. Another important note is that the decoder gets the query and value vectors from the encoder rather than the initialized one.

2.7.3.1 Linear and Softmax:

In order to obtain the word sequence back from the output vectors, linear and Softmax (Radečić 2020) layer is used. Here linear layer is just a fully connected neural layer that will just produce a very large output vector called logits vectors. The length of the vector can be a hyperparameter for the model architecture.

The Softmax layer will then turn these vectors back into probabilities of the number of words selected as a vocabulary list. If the vocabulary is consisting of 10,000 words then the Softmax layer gives the probabilities of each word that can come after a particular word.

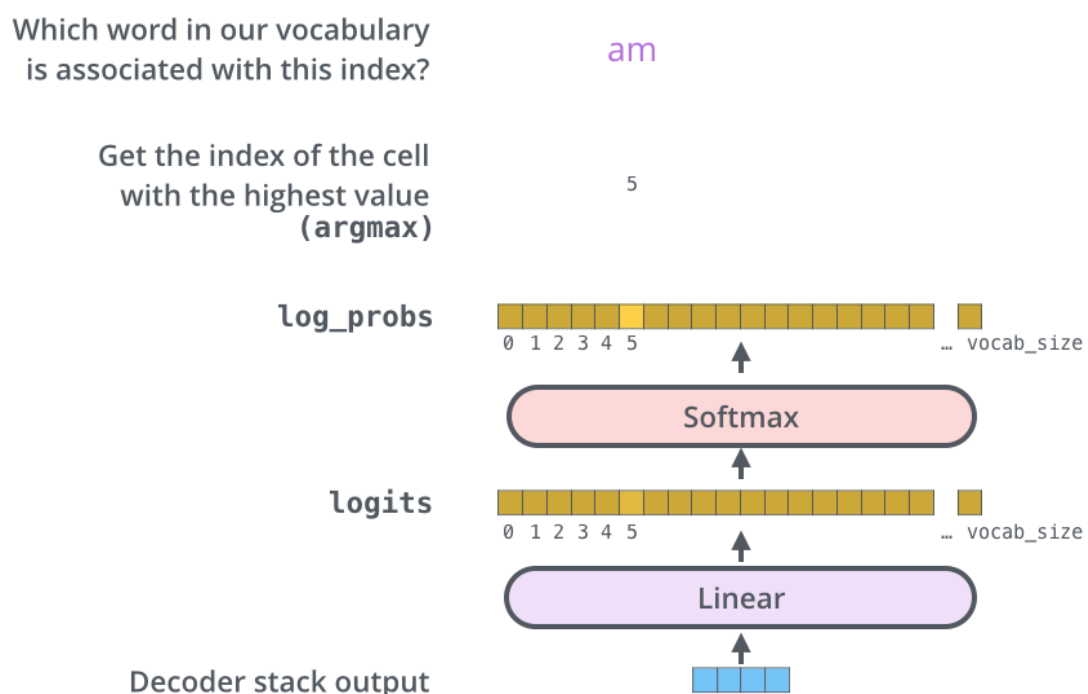


Figure 2.27: Softmax Layer in Decoder Stack (Alammar 2018a)

Argmax is nothing but to get the index of the maximum probability found in the Softmax output layer. Once the index is obtained the word corresponding to that index is the final output.

2.8 BERT (Bidirectional Encoder Representations from Transformers):

BERT (Devlin *et al.* 2019) (Horev 2018) is a model that will make use of transformer model architecture to understand the context of the words and their relationship with each other in a text. As the name itself suggests, it's bidirectional in processing its input, and it processes the entire input at once hence making the processing parallel. Since its bidirectional model will have context better with the knowledge of the entire words in a text.

BERT model is pre-trained on a large amount of text data, almost entire Wikipedia data and books. This training process makes the model understand the language better. This is usually trained by masking out 15% of the data in the corpus and the model will be trained to predict the missing data and parameters are updated as the training goes on for huge data. This model gets better when it is trained for particular tasks like sentiment analysis, text generation, entity recognition, etc. BERT uses its own tokenization technique called word piece vocabulary which is data driven approach.

2.8.1 Types of BERT

- BERT BASE
- BERT LARGE

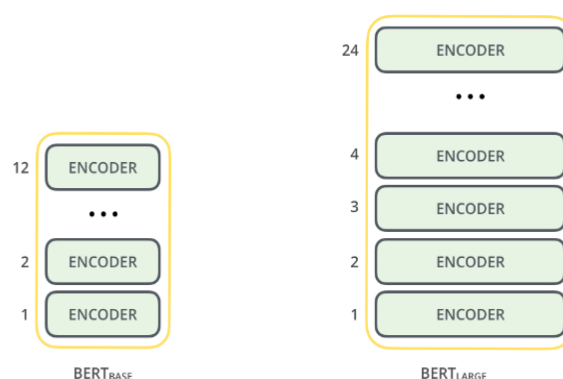


Figure 2.28: Types of BERT (Alammar 2018b)

2.8.2 Model Inputs:

The input word sequence that is passed to the BERT model is converted into tokens. The first token that is passed is represented as [CLS]. CLS here means the classification. The rest of each layers works as illustrated above in the transformer section.

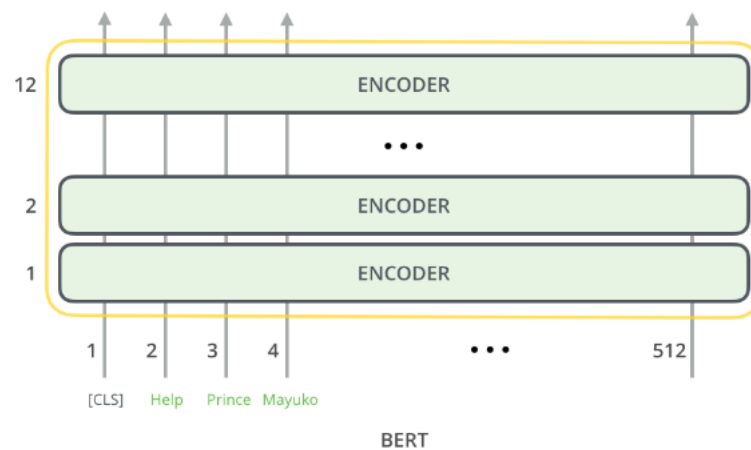


Figure 2.29: Model Inputs (Alammar 2018b)

2.8.3 Model Outputs:

For Sentiment analysis or text classification of two classifications, the first output sequence is considered to predict the classification of the text.

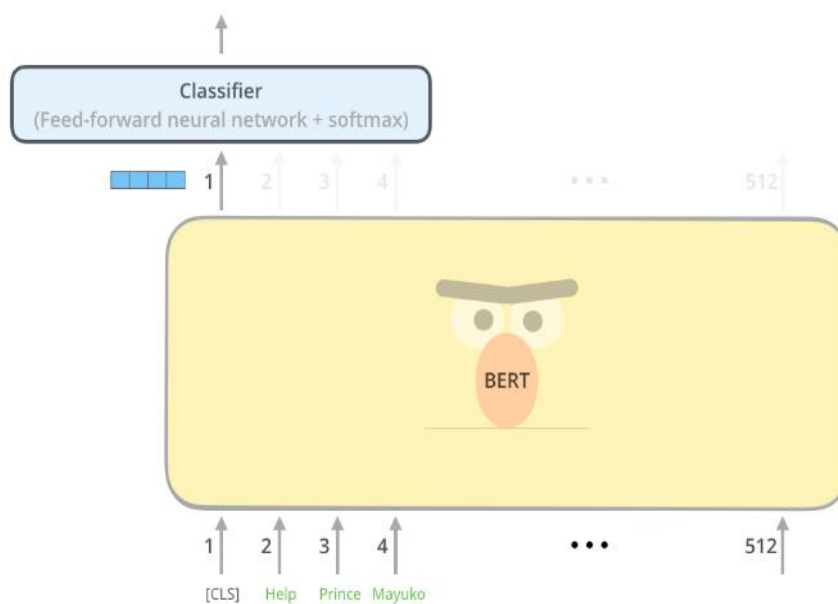


Figure 2.30: Model Output (Alammar 2018b)

2.9 Multi-Lingual BERT (M-BERT):

Multi-lingual BERT (Pires *et al.* 2019) is also trained in the same way as it is trained for monolingual but this uses texts from 104 languages in the Wikipedia. In order to match with the set of data collected for each language in Wikipedia, sampling was done on the languages where less data was obtained. Also, during the training there was no such task as cross lingual training. In M-BERT the Modeling technique for WordPiece makes the application share linguistic embedding.

3. Experiment and Analysis:

3.1 Data:

Data that is used in the project is about Covid-19². Dataset consists of 1,673,353 news articles which are of size 7.6GB. Dataset is a collection of news articles from around 440 global news sources. Each article is a collection of information regarding the new article. Each article is represented in a JSON format. Each JSON object contains fields like author, body, categories, charecters_count, entities, hashtags, id, keywords, language, media, paragraphs_count, published_at, sentence_count, sentiment, social_share_count, source, summary, title, and word_count. This many overall information we get from this data. It is also labelled data. The dataset contains a sentiment dictionary that gives the sentiment behind the news articles for the entire body of the article and the title of the article.

Each article contains positive or negative sentiment for body and news titles. In this project, we will be using the body of the news articles and the sentiment for it. The rest of the other fields are discarded. Also, this data is obtained from November 2019 to July 2020.

3.1.1 Data Pre-Processing and Extraction:

Since the dataset, much different information that is not required for the project has to be removed. To extract the body of the article “summary” field is used from each JSON object. A summary is a collection of sentences. To extract the polarity/sentiment of the news articles, the “sentiment” field is used from each JSON object. Inside the “sentiment” field it provides sentiments for both body and the title. Sentiment for the only body is obtained.

Since the data contains much unnecessary information like hashtags, @ mentions, Html tags, emojis, etc. Dataset is cleaned before it is processed. In this project it is done in the below steps:

² http://info.aylien.com/coronavirus-dataset?utm_campaign=Coronavirus%20Dataset%20Requests&utm_source=Blog%20post&utm_medium=Dataset%20blog

3.1.1.1 Pre-processing

Denoise:

This step involves cleaning up all Html tags present in the text.

Clean tweets:

This step involves cleaning up all the hashtags and @ mentions.

Remove Extra Spaces:

This step involves removing up all the extra spaces at the end of each line or in between the line. Also, it removes the newline characters present in the text.

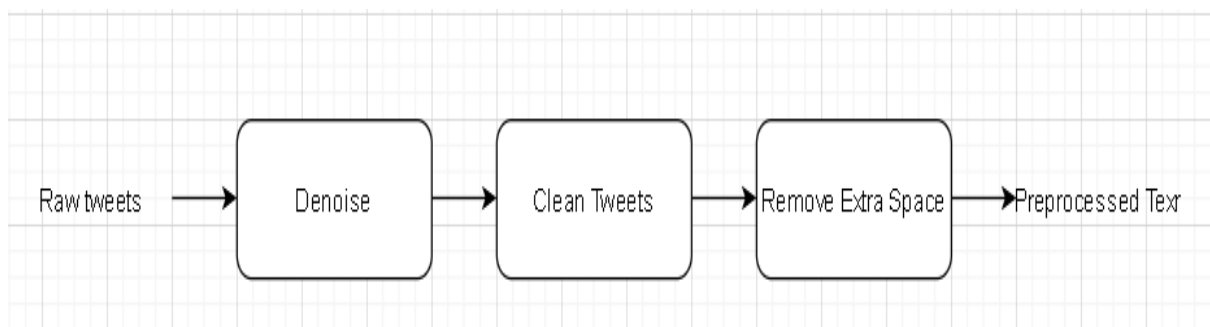


Figure 3.1: Pre-Processing Steps

And also, sentiment/polarity is extracted from the JSON for each news article. While extracting both the data i.e. body and the sentiment of the news article, CSV files are created, with heading with the first column for the text and second column for polarity. All the 1,673,353 news articles are pre-processed in the same way to create a CSV file of around 400 news articles. At the end of the pre-processing step, there will be many CSV created containing around 400 articles inside each CSV file. Also, in this project, we have marked the positive articles to have 1 and negative articles to have 0 as its polarity.

3.1.1.2 Translation:

Since the dataset is in the English language, we have to translate the data into to the Kannada language. In this project, we have used the “Textblob”³ open source library for the translation. The open-source library says it uses google translate to do the translation. The library has a limitation of 400 translations per day. Because of this reason, CSV files of 400 entries are

³ <https://pypi.org/project/textblob/0.9.0/>

created. So, this can be run on Google Colab⁴ on many different Colab notebooks to do the translation. Likewise, translation is done close to 80,000 news articles. There is no reason for choosing 80,000 news articles. Google Colab notebook contains a program, that takes a CSV as input containing text in English and then translates it to the Kannada language and creates a new CSV. Once the translation is done for all the news articles newly create CSV file is downloaded. This step was done for one month to get the data we need to do this project.

Since Google translation is not translating all the words from English to Kannada, it is ideal for the project. It does leave some of the words in English without any translation. This leaves with a mixed language labelled dataset. Also translated is tested manually to check its translation accuracy.

3.1.2 Data Exploration:

To understand the data better before going to run the training process, here are the few analytics.

Training Samples: 68,176

Validation Samples: 2,824

Totals samples: 71,000

To understand more on the distribution of the polarity in the dataset, two plots are plotted.

1. Distribution of Polarity in Training Samples
2. Distribution of Polarity in Validation Samples

⁴ <https://colab.research.google.com/>

3.1.2.1 Distribution of Polarity in Training Samples:

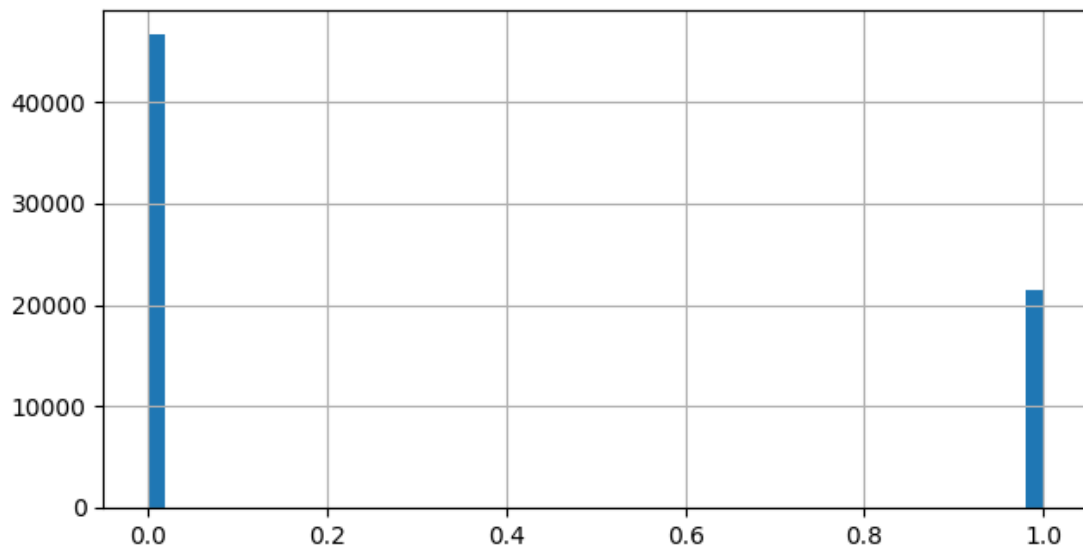


Figure 3.2: Polarity Distribution of Training samples

3.1.2.2 Distribution of Polarity in Validation Samples:

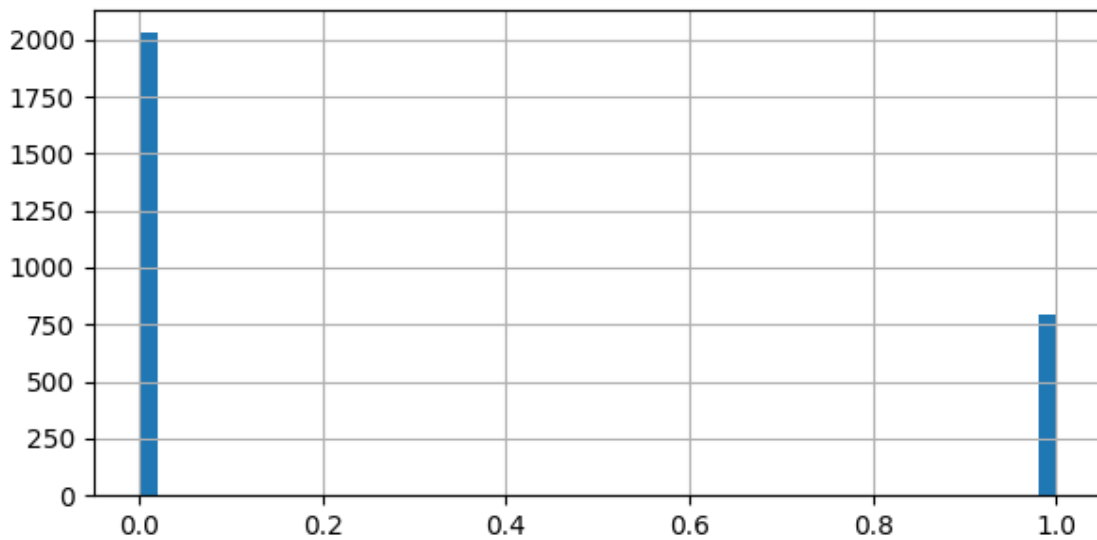


Figure 3.3: Polarity Distribution of Validation samples

Data distribution is reasonable, in both the case, positive sentiment samples are more than half of the negative sentiment samples.

3.2 Google Colab:

Google Colab is a Jupyter notebook setup that will not need any setup that is required to perform a simple operation, it comes with prebuilt installation for basic operation. Whenever a notebook is opened a machine is launched and can perform various operations on the cloud. We are using Google Colab because of the limitation we have in the Textblob open source library. With the help of Google Colab, we were able to translate many news articles to Kannada. Otherwise from local machines per day, only 400 records were the limit. With Google Colab⁵ we were able to translate close to 8000 per day. Only Textblob open-source library installation was necessary for the Google Colab, rest everything required for the program to translate and the download comes prebuilt in the Google Colab.

Google Colab was used in the initial stages to test the BERT model. It has GPU and TPU support. It was tried to run on the GPU. The disadvantage of running the training process in Google Colab was that any notebook opened has a fixed session timeout. Once the timeout is reached, it has to be refreshed and a completely new session will span. All the data that was imported to the notebook will be lost. Because of this limitation, we were not able to run the training process on Google Colab. Also uploading a training model and then downloading back to local storage is a time-consuming process.

3.3 TensorFlow:

TensorFlow is an open-source platform for Machine Learning. It provides excellent tools and materials to explore and research various machine learning tasks. It also provides various methods and tools to scale the ML (Machine Learning) application. It has great support from development to deployment of any machine learning applications. Its main motto is to abstract the machine learning computation so that we focus mainly on the application. It has excellent Deep Learning API support from Keras integrated into TensorFlow backend. It helps us visualize and understand the machine learning / deep learning task effectively easily. It provides complete support from data pre-processing, data ingestion model evaluation, visualization, and serving. It can scale from single CPU, GPU to clusters of GPUs, and also TPU. It also built in a mindset that it should run all the platforms.

⁵ https://colab.research.google.com/drive/1UWG_g1jQOjJC6WU8ZM_EtsHnZrG2pqEi?usp=sharing

In our project, TensorFlow is used to pre-process, train, and deploy the model. With the data we have, it took one day to complete one iteration of training. All the training process is done on a local machine. It was run on a machine with 24GB Ram, 4 GB NVIDIA GTC 1080 graphic card. Also, the model is also deployed and served via TensorFlow serving. Will be discussed in detail further. TensorFlow also supports transfer learning. Transfer learning is a process of loading an already trained model and perform the prediction on the new dataset. It is a great tool to experiment with any machine learning task.

Other similar frameworks for machine learning tasks are Torch, Caffe, Apache Spark, Scikit-Learn, etc. The reason to choose TensorFlow is that it has an extensive library and support that is available from the building model to deploying are excellent. And also, it supports GPU operation, making the training faster than usual. TensorFlow Serving is another excellent tool to deploy the model. We will explore its use in the later section.

3.4 Training Model:

Dataset is divided into training and validation sets. Validation sets are mainly used to check if the model is overfitting. Plotting both the training curve and validation curve together will give us a good idea about the model if it is overfitting or not. The validation curve should sync with that of the training curve to indicate it is not overfitting. Training is done in steps as below:

3.4.1 Load Dataset:

Since the dataset is spread across many CSV of around 400 news articles each, need to iterate over the entire set of CSVs to aggregate all the news articles. This is done by iterating over files and loading it in ram. This is done for all the files iteratively, meanwhile, all the files loaded are concatenated with the previous one to form one big table. This is done using the Pandas python library. The same process is repeated for the validation dataset spread across the file's invalidation data path. DataFrame of each train and valid contains sentences and polarity as the columns.

```
def load_datasets():
    train_df = pd.DataFrame()
    for name in os.listdir(train_base_dir):
        file_path = os.path.join(train_base_dir, name)
        train_df = pd.concat([train_df,
                               pd.read_csv(file_path, sep=',', names=["sentences", "polarity"]),
                               ignore_index=True
                              ])

    valid_df = pd.DataFrame()
    for name in os.listdir(valid_base_dir):
        file_path = os.path.join(valid_base_dir, name)
        valid_df = pd.concat([valid_df,
                               pd.read_csv(file_path, sep=',', names=["sentences", "polarity"]),
                               ignore_index=True
                              ])

    return train_df, valid_df
```

Figure 3.4: Loading dataset helper function

3.4.2 Tokenization:

Tokenization is a process where the text is converted to numerical values. In this process, each word is converted into its base form, and then it has given a specific number to it in the vocabulary size. BERT has a method called WordPiece tokenization (Wu and Dredze 2019) for performing this tokenization task.

Tokenization involves below steps:

1. Lower casing
2. Accent removal
3. Punctuation splitting
4. Whitespace tokenization

Even though accent matters in the same language, it is removed to improve effective vocabulary. A strong BERT model will be able to counter this removal of the ambiguity caused by the removal of accents. In total 110k shared WordPiece vocabulary is used. The Chinese language does not have whitespace to it, before sending it to the tokenization white spacing is added after each character. It will have character-wise tokenization.

3.4.2.1 List of Languages Supported:

BERT tokenization supports 104 languages⁶. An example of tokenization of individual languages and as well when the two languages like English and Kannada are mixed. Following figure shows how the tokenization looks using BERT tokenization.

⁶ <https://github.com/google-research/bert/blob/master/multilingual.md#list-of-languages>

```
===Only Kannada===
Input Text: ಶುಭ ದಿನ
Tokenized Form: ['ಶ', '##ು', '##ಭ', 'ದ', '##ನ']
===Only English===
Input Text: Good Day
Tokenized Form: ['good', 'day']
===Both language
Input Text: Good ದಿನ
Tokenized Form: ['good', 'ದ', '##ನ']
```

Figure 3.5: Tokenization Example

Above it is shown that tokenization works on many languages. At first, only Kannada is used, then only English language input is used and at last, both the language in the same sentence is used to do the tokenization.

3.4.2.2 Working:

In this process, we first append each news article with special tokens [CLS] at the beginning of the sentence and [SEP] at the end of the sentence. This is the requirement for a BERT model to work. This process is repeated for all the sentences in the dataset. Once this is done, padding is applied. Since in machine learning any computation is done by matrix multiplication, we need to have a fixed number of tokens for each sentence to represent a sentence. In order to do this, we will pad each sentence to fixed number of lengths. This maximum sequence length can be a hyperparameter to try out different combinations.

3.4.3 Building a Model:

In this process, a pre-trained BERT model is downloaded from the source⁷. It comes with three files, BERT config file in JSON format, which gives us the details of the BERT model with a number of attention heads and other information. Another is the vocab file and the checkpoint directory, which is actually the model parameters stored after the initial training.

Since it is a pre-trained BERT model, while in the training process, input sequence and its length could be different from our requirement for the project. For this reason, we will create our own custom input layer with the required input sequence to it and then add this as the input

⁷ <https://github.com/google-research/bert/blob/master/multilingual.md>

layer for the pre-trained BERT model. This way we are adjusting the trained model to suit our needs. This is done using Keras.

Likewise, we are using the BERT model to do the classification task. As the BERT model can be used for many different applications or the use cases, the output layer could vary from one application to another. Here in our application to address the classification problem of only two classes we are using the “Softmax” activation function at the last layer with two units. Two units will represent or say if the given sentence is a positive or negative sentiment. Softmax gives two values which is probability values which on adding gives 1. This way it is easy to predict which class the given input belongs to.

Before adding a Softmax layer, we do add one “Dense” layer with 768 units with the “Tanh” activation function to learn the classification problem we are training the model for. With the addition of this layer, the model will get fine-tuned to our particular problem. We can add more such dense layer, in our project we stick with only one layer of Dense connected network layer is added. In order to address the overfitting and also the regularization issue that is commonly seen in the deep learning models, we have added the “Dropout” layer.

3.4.3.1 Dropout:

Dropout is a regularization technique used in deep learning techniques. On adding this layer, this is will turn off the percentage (input for the dropout layer) of the nodes in particular layers. This will allow only few nodes to react to the input that is received for that layer. This will allow learning of the input parameters more compare to without the dropout layer.

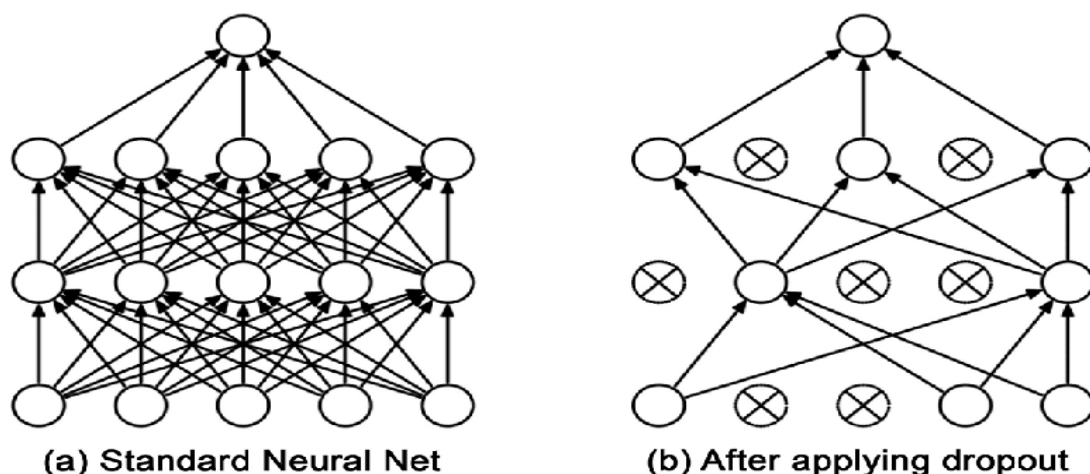


Figure 3.6: Dropout working (Maklin 2019)

This way by adding a dropout layer we get the regularization and also edge over the overfitting problem. Input to the dropout layer could be hyperparameter to adjust the learning curve.

3.4.3.2 Model Structure:

Model: "model"

Layer (type)	Output Shape	Param #
input_ids (InputLayer)	[(None, 128)]	0
bert (BertModelLayer)	(None, 128, 768)	177261312
lambda (Lambda)	(None, 768)	0
dropout (Dropout)	(None, 768)	0
dense (Dense)	(None, 768)	590592
dropout_1 (Dropout)	(None, 768)	0
dense_1 (Dense)	(None, 2)	1538

=====
Total params: 177,853,442
Trainable params: 177,853,442
Non-trainable params: 0
=====

Figure 3.7: Summary of the Model

3.4.3.3 Model Constructions looks like below:

```
def create_model(max_seq_len, bert_ckpt_file):
    with tf.io.gfile.GFile(bert_config_file, 'r') as reader:
        bc = StockBertConfig.from_json_string(reader.read())
        bert_params = map_stock_config_to_params(bc)
        bert_params.adapter_size = None
        bert = BertModelLayer.from_params(bert_params, name='bert')

    input_ids = keras.layers.Input(shape=(max_seq_len,), dtype='int32', name='input_ids')
    bert_output = bert(input_ids)

    cls_out = keras.layers.Lambda(lambda seq: seq[:, 0, :])(bert_output)
    cls_out = keras.layers.Dropout(0.5)(cls_out)
    logits = keras.layers.Dense(units=768, activation='tanh')(cls_out)
    logits = keras.layers.Dropout(0.5)(logits)
    logits = keras.layers.Dense(units=len(classes), activation='softmax')(logits)

    model = keras.Model(inputs=input_ids, outputs=logits)
    model.build(input_shape=(None, max_seq_len))

    load_stock_weights(bert, bert_ckpt_file)

    return model
```

Figure 3.8: Helper function to create BERT model

3.4.4 Model Compilations:

From the above, model architecture is constructed to handle input and output according to our application. Deciding and constructing the input and output layer is the challenging part of any deep learning model. Once this is done, we are left with compiling the model with suitable optimization function and the loss function to compliment the model architecture.

3.4.4.1 Optimization Function:

In this model compilations, we are using Adam optimization (Khan *et al.* 2019). **Adam** Optimization is abbreviated as **Ad**aptive **M**oment estimation. This method contains both momentum (Wilson *et al.* 2018) and RMSProp (Hinton *et al.* 2012). Combining both the method gives Adam optimization four hyperparameters to tune.

1. Learning Rate Alpha.
2. Beta for momentum (default 0.9)
3. Beta2 for RMSprop
4. Epsilon

In our project, we are only using the learning rate hyperparameter.

3.4.4.2 Loss Function:

Since we are having only two classes in our project. We could have used only one output layer to denote the output classes to be either 1 or 0. In order to make this project adaptive to many different classes and to avoid the bias in using 1 for positive and 0 for negative, we used two layers. Two layers are one-hot encoded. In order to handle this one-hot encoding, we used sparse categorical entropy as our loss function.

3.4.4.3 Metrics:

There are many different metrics that TensorFlow framework provides⁸, in our project we are using accuracy metrics to analyse the model performance.

⁸ https://www.tensorflow.org/api_docs/python/tf/keras/metrics

In total model compilation looks like below:

```
model.compile(  
    optimizer=keras.optimizers.Adam(1e-5),  
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
    metrics=[keras.metrics.SparseCategoricalAccuracy(name='acc')]  
)
```

Figure 3.9: Model compilation

3.4.5 Model Training:

Once the model is compiled successfully the last part left is to train the model with training dataset and the validation dataset. Since combined training and validation datasets we have 71,000 news articles, it does take a lot of time to train. Because of this reason, we are running the training process one iteration at a time. Once one iteration is completed, we will save it to the local directory. And to run it again, we will load the save model and then train it for one more iteration and then save it back again to the same path. It is done in this way to avoid any interruption that may happen in the local setup. If on some interruption it will be of the loss of only one iteration.

TensorFlow allows us to pass both training and validation datasets at once. In this way TensorFlow does both training and validation for each iteration and it will log it. This helps us understand and compare the training and validation curves for each iteration.

Also, TensorFlow allows us to train the datasets in batches. TensorFlow is optimized to train in parallel and batch size will help a greater in reducing the time taken for each iteration. Batch size is also hyperparameter for the model. It can be studied with various different values. And shuffle is also used along with this to shuffle the dataset that is passed to the training while passing the data to the training loop.

```
history = model.fit(  
    x=data.train_x,  
    y=data.train_y,  
    validation_data=(data.test_x, data.test_y),  
    batch_size=16,  
    shuffle=True,  
    epochs=1,  
)
```

Figure 3.10: Code Snippet for Training the mode

3.4.6 Saving:

Saving the model after training is a very important process in the project. TensorFlow provides a various different method to save a model. Here in this project, we are saving it away it is useful when we are serving the model not just loading the model and serving. The advantage of doing is discussed further in the serving section. This also helps to version our model.

3.4.7 Plotting:

As discussed in the training section, we are using the only accuracy as metrics. And since we are also passing the validation dataset for training, we also get validation accuracy for each iteration. Using Matplotlib library we will plot the accuracy versus validation accuracy to better understand if the model is overfitting.

Loss vs Validation Loss:

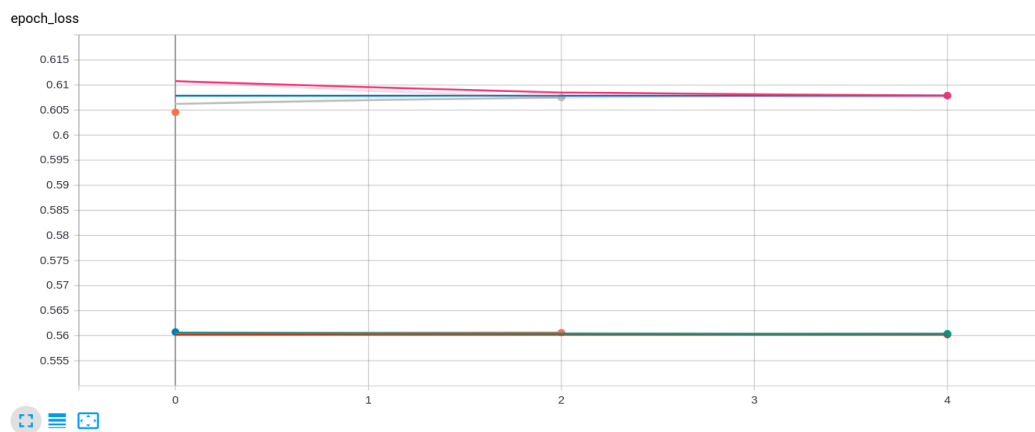


Figure 3.11: Loss vs Validation Loss

Accuracy vs Validation Accuracy:

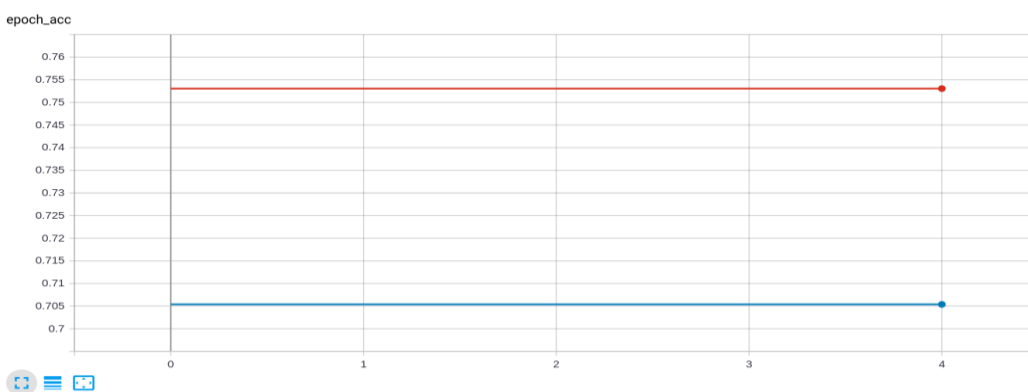


Figure 3.12: Accuracy vs Validation Accuracy

3.5 TensorFlow Serving:

If training is big task, serving the trained model is another big task. It is considered to be a big task because of the number of trained parameters. BERT is a very big model that consists of a very huge number of parameters to load while loading. There are two in which we can deploy the trained model.

3.5.1 Load the Model:

In this method, the model that is trained is loaded using TensorFlow methods, and then it is asked to make the prediction. Before passing the inputs to the model for predictions it has to be pre-processed to fits its input layer. Once the tokenization and padding are applied to the input text, it is then given to the model to predict.

This method has a disadvantage, that is every time to load a model, it takes a few minutes. If a request is made to make the prediction, then the model has to be loaded, input has to be pre-processed, and then it has to be passed to the model. This will easily take 3-5 mins to respond to new input.

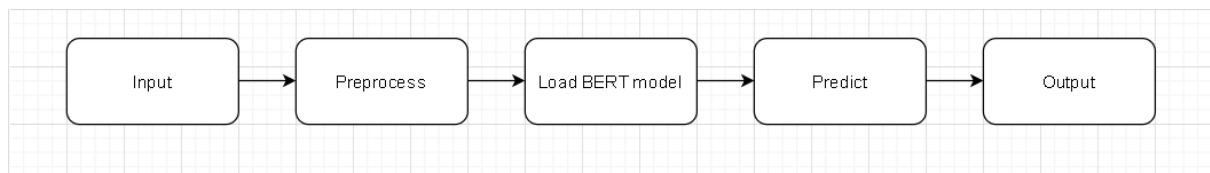


Figure 3.13: Loading the Model

This is a very time-consuming method to serve the model. Loading the model consumes a lot of time and it will also send the other requests to hold.

3.5.2 TensorFlow Serving:

In this method, the TensorFlow model that is saved in a particular format and directory structure will be served in a separate process. It is a very flexible and high-performance serving system for machine learning models. It is a production-ready solution provided by the TensorFlow. It is a very convenient and handy method to follow as you don't have to worry about loading and

then serving every time a request is received from the web applications. This also provides an easy way to experiment with different models or versions of the model. Just save the different models or different versions of the model in a version directory and provide the path while starting the service. Since this gives such flexibility this is the ideal way of working around serving a model. Also, as the API interphase remains the same though the serving from any version of the model, experiments of the models becomes very easy. Here as we can see the loading of the model is eliminated and becomes simpler compared to earlier method.

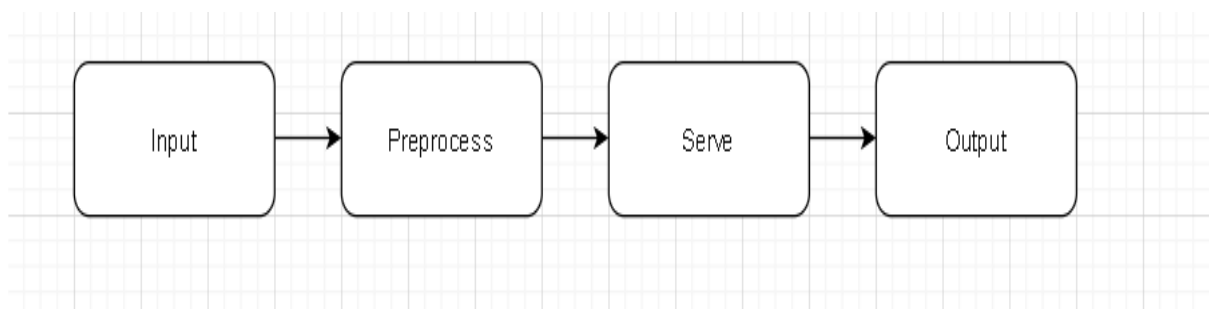


Figure 3.14: TensorFlow Serving

The architecture of the serving:

It is an out of the box integration with the TensorFlow models which allows easy integrations with many different models.

Key Concepts:

Key concepts to understand the architecture of TensorFlow serving is as below:

3.5.2.1 Servable:

It is an abstraction of TensorFlow serving. These are the object on which the clients perform the action. The Servable objects are very flexible in terms of size and its granularity. Single Servable objects might contain a collection of lookup tables, a single model, or a tuple of models. This will allow flexibility and future improvements such as:

1. Streaming results

2. Experimental APIs
3. Asynchronous modes of operation.

3.5.2.2 Servable Versions:

One main advantage of TensorFlow Serving is it can handle one or more versions of servable over a lifetime of single server instance. This will allow new algorithm configurations, weights, and other data to be loaded over time. Versions enable serving more than one version of models at a time, allowing smooth rollout and experimentation of the new version. At the time of serving, clients will get to make calls to either of the versions. If in case the new version is not up to the desired mark, it may not be used and continue to serve the old version till the new version gets the fix that is causing the issue in the new version. This feature gives the developer a very healthy way to test the new version of the model in the production environment. Once the testing is done and satisfied, clients may use the new version of the API just by changing the version number.

3.5.2.3 Servable Streams:

A Servable stream is nothing but the list of versions that are saved to be served. This is mainly sorted in ascending order.

3.5.2.4 Models:

TensorFlow Serving addresses to each Model through Servable. It can be one servable or more than one servable. Machine Learning models may include one or more algorithms and lookup or embedding tables.

The composite model can be seen below:

1. Multiple independent Servable.
2. Single Composite Servable.

Servable may also correspond to a fraction of a model. For example, a large lookup table could be sharded across many TensorFlow Serving instances.

3.5.2.5 Loaders:

The lifecycle of any Servable is managed by Loaders. Loaders enables loading and unloading the Servables with standardized API and common interface is given to learn algorithms.

3.5.2.6 Sources:

Sources acts as a plugin module that helps find and deliver servables. Each Source deliver zero or more servables streams. Source makes sure that each servable stream is getting one Loader instance and for it also takes care of all the version that are available to be loaded.

The Sources module of TensorFlow Serving is able to discover the servables from arbitrary storage structures. This also contains common implementations of Source. For example, RPC mechanism and polling the file system can be accessed by the Sources.

States is maintained by the Sources which are shared across many servables or versions. This is an added advantage for servables that make use of delta (diff) updates between versions.

3.5.2.7 Aspired Versions

This represents the number of servable versions that is available to be loaded and ready to serve. Sources makes sure that each servable version is available for single servable stream at any point of time. Whenever there is an update for aspired with new list to the manager, the new list takes the preference over the old one. The manager abandons all the previous version list servables and loads new servable.

3.5.2.8 Managers

Managers handle the full lifecycle of Servables, including:

1. Loading Servables
2. Serving Servables
3. Unloading Servables

Managers responds to the Sources and keep track of all the versions it handles. Manager job is to deliver all the requests that it demands and at times it may also refuse to load an aspired version if the resources that are required are not available to be served. Managers may also delay the process of unloading. For example, managers monitor if the newer version of the servables are loaded completely before it actually unloads the previous version. This depends on the configuration while starting the server.

3.5.2.9 Core:

Using the standard TensorFlow Serving APIs, TensorFlow Serving Core manages the following aspects of servables:

1. Lifecycle
2. Metrics

TensorFlow Serving Core treats servables and loaders as opaque objects.

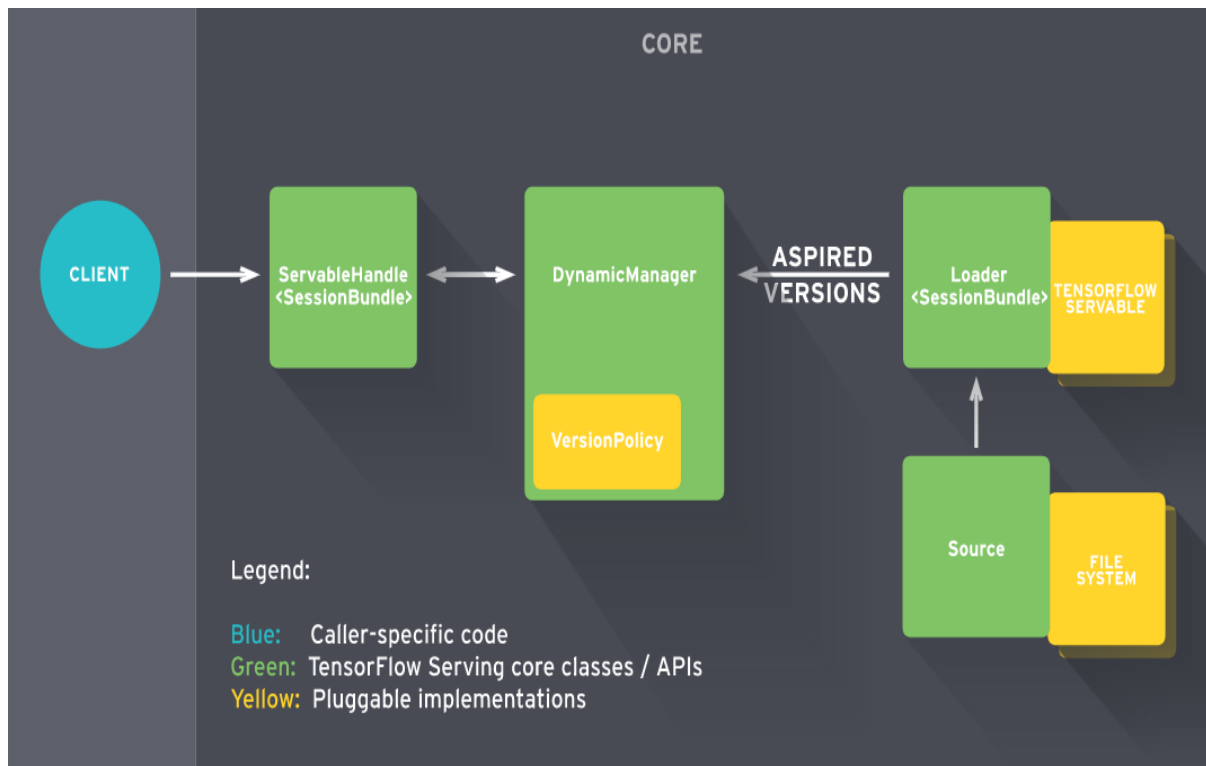


Figure 3.15: TensorFlow Serving Architecture (TensorFlow 2020)

3.5.3 Overall Working Involves Two Steps:

1. Sources create Loaders for Servable Versions.
2. Manager receives Loaders as Aspired Versions, which then decides to load and make it available to the API requests.

In detail:

1. Specific version of loader is created by source plugin. The Loader contains whatever metadata it needs to load the Servable.
2. Manager will get notified using a call-back about the Aspired Version.
3. Manager decides the next action based on the configuration of the Version Policy. This action could be to unload or load the new version in place of old version.
4. If the Manager determines that it's safe, it gives the Loader the required resources and tells the Loader to load the new version.
5. Clients are served with the API request either specifying the version or by default the latest version available. Manager is responsible in providing the clients with the respective Servable based on the version that is sent via API request.

For example, say a Source represents a TensorFlow graph with frequently updated model weights. The weights are stored in a file on disk.

1. Whenever a new model is made available in the saved directory, The Source identifies the changes in the directory and it automatically creates a Loader. The newly created Loader then will point to the model weights that were created newly.
2. The Manager gets notified with the same of the Aspired Version.
3. After Manager is notified about the new Aspired Version it will then decides whether to roll in the new version of model based on Version Policy configuration.
4. The Dynamic Manager now informs the Loader with memory information. After this TensorFlow graph is initiated with the new weights.
5. Last step, where the client requests for the newer version of the model to respond to the request. The Dynamic Manager identifying the version from the client's request will then handle the Servable which is point to new version.

3.5.3.1 Extensibility

TensorFlow Serving provides several extension points where you can add new functionality.

3.5.3.2 Version Policy

This is a policy in which it states the sequence of version that is loading and unloading inside a single servable stream.

TensorFlow Serving has two main tasks to handle that is to accommodate two main use-cases.

1. **Availability Preserving Policy:** This policy deals with the version handling and making the latest version available by terminating the older one. This is responsible of smooth transition from one version to another.
2. **Resource Preserving Policy:** This policy deals with resource handling. It will make sure that only one version of servable is available at a time. This leads to saving the resources required to multiple servable lifecycles.

The two types of usage of TensorFlow Serving:

1. **Simple Usage:** Here resource costs has to be low and the availability of the model is very important. In this case Availability Preserving Policy will deal with loading the newer version of the model before the old version is brought down. This makes sure that new version is always available and at a time only one servable will be serving to avoid the limitation of the resource
2. **Sophisticated Usage:** Here in this use case where multiple instances of server is important. In this the resource Preserving Policy has no extra burden of making sure the newer version is able and then bring down the old version. In this memory is saved to load newer version.

3.5.3.3 Source

The support provided by the Source might extend to handle different file system, new algorithm backends and cloud solutions. TensorFlow Serving has made it very easy to create new sources and it also consumes less time. For Example, TensorFlow Serving has a method to bind the

behaviour of polling around the source. Sources are tightly coupled with Loaders for data hosting servables and particular algorithms.

3.5.3.4 Loaders

Loaders are the place where adding algorithms and backends for data can be extended. One of the algorithm backend is TensorFlow. For example, It is possible to implement a custom Loader in which it will load, access and unroll an instance of a new type of Servable machine learning model. It is designed to be very customisable by having a lookup mapping and additional support for algorithms.

3.5.3.5 Batchers

Batching is an important component in TensorFlow when it comes to the matter of scaling. This component helps in grouping or collection of requests into multiple batches and then make a single access to the model. And in the presence of GPU accelerator this place very important role. Upon this facility TensorFlow also provides a way to client to specify batches to the similar kind of model access. This will enhance the performance and in total it becomes a very efficient and optimal solution.

Entire code for the project is available in the GitHub repository⁹.

⁹ https://github.com/shravanc/msc_project

Chapter 4

4. Web Application:

In the above section, the training of the model is discussed. In this section, we will be using the trained model to make predictions of sentiments on Kannada tweets. It is a simple application that pulls in tweets of certain hashtags or accounts of any choice which is related to Covid-19 is then processed and analysed to predict the sentiment. This application also provides another page to enter text and get the sentiment.

The web application is built using the python programming language with the Flask microframework.

4.1 Flask:

Flask is said to be a microframework because of its simple nature. It contains minimal components to start with and its main principle is to be simple yet extensible. Like other frameworks such as Ruby on Rails, it will not force us to use certain structures and conventions strictly, but this gives flexibility for the developers using the framework. It supports almost all the popular SQL and NoSQL databases like MySQL. In this project, we are not using any database as there is no use of storing anything in the application. Though it is said as a microframework it is a production-ready framework. It provides various extensions for database integration, form validation, authentication mechanism.

This application is designed in a way that, after every refresh for the page, will make an attempt to fetch the latest tweets at that moment and once all the tweets are gathered, the prediction will happen on each of the tweets. Once all the prediction is done, HTML table is generated with text sentence and its sentiment.

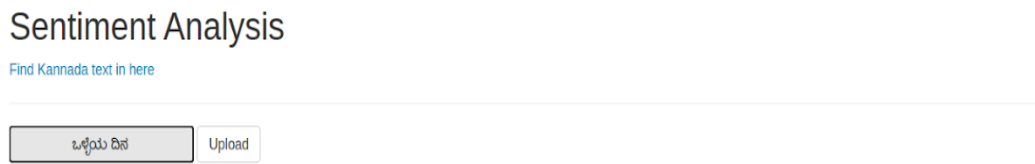
4.1.1 HTML:

In this project, there are three pages and to different flows involved.

4.1.1.1 Flow 1:

In this flow, the user is asked to enter the text in an input text box and button to submit the text. Once the page is submitted it will redirect to another page where it displays the given text and its sentiment. For this flow, the input text given should be in Kannada. There is no validation as to which the input language is given.

Page 1:




Sentiment Analysis

[Find Kannada text in here](#)

Figure 4.1: Page 1: with input text

Page 2:



Sentiment: Positive

Text: ಒಳ್ಳೆಯ ದಿನ

Figure 4.2: Page 2 with Prediction

4.1.1.2 Flow 2:

In this flow, when the page loads, it will make an API request to the backend server and it displays the table of tweets. In this HTML page, the table is displayed with each tweet and its corresponding sentiment.

Sentiment Analysis

Analysing the tweets for hashtag: #ಕೋವಿಡ್_19

Tweets	Sentiment
ನಮಸ್ಕಾರ... 🙏🙏🙏 ಅನ್ನಾಕ್ ಪ್ರೇಮ ಈಗ ಪೂರ್ತಿ ದೇಶದಲ್ಲಿ ಅರಂಭವಾಗಿದ್ದು... ಕೊತ್ತಂಬರಿ ಸೊಪ್ಪು ತರುವುದಿದ್ದರೆ ಮಾತ್...	Positive
ಭಾರತದಲ್ಲಿ ದಾವಲೆಯು 8,99,864 ಪರೀಕ್ಷೆಗಳನ್ನು ಆಗಸ್ಟ್ 17ರಂದು ಒಂದೇ ದಿನ ಮಾಡಲಾಗಿದೆ.	Negative

Figure 4.3: Page 3 list of tweets and its Sentiment

4.1.2 API Request:

API (Application programming interface) it is a very common mode of communication between the client and the server. There are various different methods like POST, GET, PUT, PATCH, UPDATE, DELETE, etc. GET is usually used to retrieve the resource objects from the application, POST is to create the resource objects into the application likewise each method is used to do various operations on the application.

In this project, we have defined one API, where the clients when the page is loaded it will perform business logic defined to do. In this API, it will fetch all the tweets for the given or selected hashtag, it will pre-process all the tweets to remove unnecessary characters, and then it will be passed to sentiment analyser. In this, it will iterate over each tweet and then get the prediction from the saved model. This prediction is obtained by making another API request to the saved model. The saved model running as TensorFlow serving will support API requests to make the prediction from the saved model.

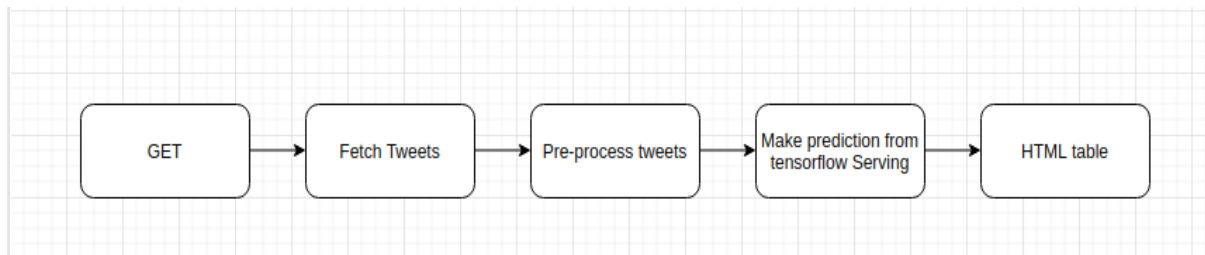


Figure 4.4: API request lifecycle

4.1.3 TensorFlow Serving API Server:

TensorFlow Serving API plays a main role in responding to any requests to make prediction from the webpage. Without the help of TensorFlow serving each request would take more than 5 mins to make prediction for each text input. With TensorFlow serving API it is able to serve within one minute.

Starting TensorFlow Serving:

```
tensorflow_model_server \
--rest_api_port="8501" \
--model_name="saved_model" \
--model_base_path="/home/shravan/dissertation/bert_model/test_model/"
```

Figure 4.5: TensorFlow Serving Start Command

This command will start the TensorFlow serving on 8501 port number, reference name to the serving model is “saved_model” and path to the saved model is specified.

4.1.4 TensorFlow Serving API:

Using Postman client to test the GET request for TensorFlow Serving. As we can see in the figure, need to pass in the sequence of maximum length to the model. Sequence is obtained from the word piece tokenization provided by the BERT. In return, the response contains two values, which says whether the text input sequence that is provided to the model for prediction is either positive or negative.

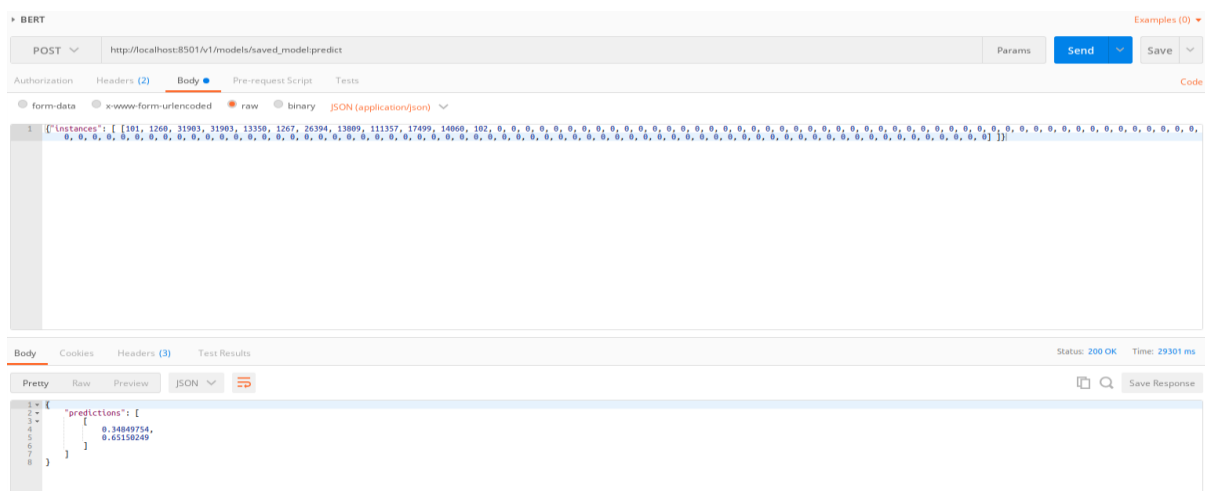


Figure 4.6: API request for TensorFlow Serving

5. Limitations and Future Work:

5.1 Limitations:

The main intention of this project is to build a deep learning model that understands the context of the mixed language. Projects aim to train the model with a large amount of data to better understand the meaning or the mapping of the word to its polarity. This way model does better prediction. The two main limitations of these projects are as below:

5.1.1 Polarity distribution:

Sentiment Analysis is classification problem. Here that data needs to labelled in order to train the model. The limitation that a Sentiment Analysis application faces are the distribution of polarity/sentiment of the labelled data. Most of the time, labelled data might not be equally distributed. One or more of all the labels might be a majority given the model a false statement as to except text of that class.

The model trained with unequal distribution might tend to bias over to the label that is majorly available. This might affect the accuracy of the model on a newly seen data. This is the main limitation that the Sentiment Analysis problem has to attend to.

5.1.2 Mixed Language:

One another limitation that we have in this project is all the data that is used for training is obtained from the translation from English to Kannada. In this process of translation, some of the words might get left out in translation and by giving mixed language datasets. Since the translation is also another process which is under research and developing, it cannot be trusted completely.

Another part is to collect the actual data and then label the data manually for a very large amount of data. There is no freely available data to train the model for Kannada and English language. This has to be obtained either by translation or by manual labelling of the data. Both the solutions are time consuming and the translation service available out there is also premium service.

5.2 Future Work:

This project tries to answer that Sentiment Analysis can be done with good accuracy even if there are mixed language text input. Future work can be of below:

5.2.1 Pre-Processing:

In this project, the main pre-processing is done is the removal of all the unnecessary data like hashtags, URLs, HTML tags, whitespaces, etc. Then directly the input is given for tokenization and then it is converted to a number sequence of maximum length and fed to the model for training. Here stop words are not removed. This is done because the BERT is rich in tokenizing the input text data in many languages. Should train the model by removing the stop words and check the accuracy.

5.2.2 Hyper-Parameter Tuning:

There are many hyper-parameters that can be tuned in this model like, batch size, maximum sequence length, learning rate, etc. Also, there are many other optimization techniques that can be used. Many other loss functions that can be used. Many metrics to better understand the model performance. TensorFlow has better tools to also scale the training process by using estimators. This also helps in providing the hyper-parameter range wherein the model will train in those ranges and provide a better parameter for a good model.

5.2.3 Hardware Infrastructure:

As seen above in the experiment and analysis section the BERT model has a very huge number of hidden parameters. With the 71,000 news articles to train for one iteration, it almost took 1 complete day. All this project is done on the local machine on the CPU. Because of this limitation model was trained only for one iteration and saved. This process for repeated for several days. This has to be upgraded to a cloud-based solution wherein a huge dataset more than 71,000 news articles should be trained and then understand the model performs better.

Conclusion

Deep learning becoming more popular day by day in the field of Artificial intelligence and Machine Learning, its applications are growing very rapidly. Deep learning is gaining its usage in almost all industry. As the data availability is more in the modern days in almost all the area deep learning becomes the first choice to deal with the automation or any application that can be think of to solve using machine learning approach. BERT especially have made very impact in the field of Natural Language Processing. Google is using the BERT for its search engine (Nayak 2019) to improve better contextualize the query the user enters.

Since the BERT is performing very good in the field of Natural Language Processing, can this be applied to other language, as most of the research happen on English, Spanish, French, German and Chinese language. The curiosity concern of solving Natural language processing problem for other language is solved satisfactorily with minimum hyperparameter tuning. Similar work on other language are done (Conneau *et al.* 2018) (Wu and Dredze 2019).

From this project we able to get an accuracy of above 70%. This was done on a good amount i.e. 71,000 of news articles related COVID-19. With the BERT layer and a Dense layer to learn the new parameter and to map it to the polarity and with just the max sequence length of 125 we were able to achieve above 70% accuracy. We are sure to increase the accuracy to above 80 percent with more on pre-processing techniques. This project is done without any pre-processing steps like removing stop words. We believe model can be improved further to perform even better with removal of stop words for Kannada language and then by increasing the max sequence length the model will improve its prediction ability.

References

Agarap, A.F. (2019) Deep Learning Using Rectified Linear Units (ReLU), available: <https://arxiv.org/pdf/1803.08375.pdf> [accessed 18 Aug 2020].

Alammar, J. (2018) The Illustrated Transformer [online], Github.io, available: <http://jalammar.github.io/illustrated-transformer/> [accessed 18 Aug 2020].

Alammar, J. (2018) The Illustrated BERT, ELMo, and Co. (How NLP Cracked Transfer Learning) [online], Github.io, available: <http://jalammar.github.io/illustrated-bert/> [accessed 25 Aug 2020].

Bahdanau, D., Cho, K., Bengio, Y. (2015) Neural Machine Translation by Jointly Learning to Align And Translate, available: <https://arxiv.org/pdf/1409.0473.pdf>.

Cavnar, W.B., Trenkle, J.M. (2020) Download Limit Exceeded [online], citeseerx.ist.psu.edu, available: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.3248&rep=rep1&type=pdf> [accessed 24 Aug 2020]

Conneau, A., Rinott, R., Lample, G., Schwenk, H., Stoyanov, V., Williams, A., Bowman, S. (2018) XNLI: Evaluating Cross-Lingual Sentence Representations, Association for Computational Linguistics, available: <https://www.aclweb.org/anthology/D18-1269.pdf> [accessed 24 Aug 2020].

Devlin, J., Chang, M.-W., Lee, K., Google, K., Language, A. (2019) BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding, available: <https://arxiv.org/pdf/1810.04805.pdf>

Dubey, A.K., Jain, V. (2019) ‘Comparative Study of Convolution Neural Network’s Relu and Leaky-Relu Activation Functions’, Lecture Notes in Electrical Engineering, 873–880,

available: https://link.springer.com/content/pdf/10.1007%2F978-981-13-6772-4_76.pdf
[accessed 31 Oct 2019].

Gupta, D. (2020) Activation Functions | Fundamentals Of Deep Learning [online], Analytics Vidhya, available: <https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/>.

Hinton, G., Srivastava, N., and Swersky, K. (2012) Lecture 6d - a separate, adaptive learning rate for each connection. Slides of Lecture Neural Networks for Machine Learning

Hirasawa, K., Ohbayashi, M., Koga, M., Harada, M. (1996) Forward Propagation Universal Learning Network [online], IEEE Xplore, available: <https://ieeexplore.ieee.org/abstract/document/548917> [accessed 24 Aug 2020].

Hochreiter, S. (1998) 'The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions,' International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 06(02), 107–116

Hoffman, G. (2018) Introduction to LSTMs with TensorFlow [online], O'Reilly Media, available: <https://www.oreilly.com/content/introduction-to-lstms-with-tensorflow/> [accessed 18 Aug 2020].

Horev, R. (2018) BERT Explained: State of the Art Language Model for NLP [online], Medium, available: <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>.

Joshi, P. (2019) Transformers In NLP | State-Of-The-Art-Models [online], Analytics Vidhya, available: <https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/> [accessed 24 Aug 2020].

Karsoliya, S. (2012) Approximating Number of Hidden Layer Neurons in Multiple Hidden Layer BPNN Architecture. International Journal of Engineering Trends and Technology, 3, 714-717.

Khan, I., Jais, M., Ismail, A., Qamrun, S. (2019) ‘Adam Optimization Algorithm for Wide and Deep Neural Network,’ 2(1), 41–46, available:
<https://core.ac.uk/download/pdf/287322851.pdf> [accessed 24 Aug 2020].

Kombrink, S., Mikolov, T., Karafiát, M., Burget, L. (2011) Recurrent Neural Network Based Language Modeling in Meeting Recognition, available: https://www.isca-speech.org/archive/archive_papers/interspeech_2011/i11_2877.pdf [accessed 24 Aug 2020]

Kostadinov, S. (2019) Understanding Encoder-Decoder Sequence to Sequence Model [online], Medium, available: <https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346>.

Lee, C. (2018) Understanding Bidirectional RNN in PyTorch [online], Medium, available: <https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66>.

Li, H., Zhao, R., Wang, X. (2014) Highly Efficient Forward and Backward Propagation of Convolutional Neural Networks for Pixelwise Classification, available:
<https://arxiv.org/pdf/1412.4526.pdf> [accessed 24 Aug 2020]

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., Alsaadi, F.E. (2017) ‘A survey of deep neural network architectures and their applications,’ Neurocomputing, 234, 11–26, available:
<https://www.sciencedirect.com/science/article/pii/S0925231216315533> [accessed 23 Dec 2019]

Maclaurin, D., Duvenaud, D., Adams, R. (2015) Gradient-Based Hyperparameter Optimization through Reversible Learning, available:
<http://proceedings.mlr.press/v37/maclaurin15.pdf> [accessed 24 Aug 2020]

Maklin, C. (2019) Dropout Neural Network Layer In Keras Explained [online], Medium, available: <https://towardsdatascience.com/machine-learning-part-20-dropout-keras-layers-explained-8c9f6dc4c9ab> [accessed 22 Nov 2019].

Mittal, A. (2019) Understanding RNN and LSTM [online], Medium, available: <https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e>.

Nayak, P. (2019) Understanding Searches Better than Ever Before [online], Google, available: <https://blog.google/products/search/search-language-understanding-bert/> [accessed 24 Aug 2020].

Olah, C., Carter, S. (2016) ‘Attention and Augmented Recurrent Neural Networks,’ Distill, 1(9), available: <https://distill.pub/2016/augmented-rnns/> [accessed 18 Aug 2020].

Pai, A. (2020) CNN vs. RNN vs. ANN - Analyzing 3 Types of Neural Networks [online], Analytics Vidhya, available: <https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/> [accessed 25 Aug 2020]

Pires, T., Schlinger, E., Garrette, D. (2019) How Multilingual Is Multilingual BERT?, available: <https://arxiv.org/pdf/1906.01502.pdf>

Radečić, D. (2020) Softmax Activation Function Explained [online], Medium, available: <https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60> [accessed 24 Aug 2020]

Rekabsaz, N., Lupu, M., Hanbury, A. (2017) ‘Exploration of a Threshold for Similarity Based on Uncertainty in Word Embedding,’ Lecture Notes in Computer Science, 396–409.

Rezaeinia, S.M., Rahmani, R., Ghodsi, A., Veisi, H. (2019) ‘Sentiment analysis based on improved pre-trained word embeddings,’ Expert Systems with Applications, 117, 139–147.

Rouse, M. (2020) What Is Language Modeling? [online], SearchEnterpriseAI, available: <https://searchenterpriseai.techtarget.com/definition/language-modeling> [accessed 24 Aug 2020]

Sak, H., Shannon, M., Rao, K., Beaufays, F. (2017) ‘Recurrent Neural Aligner: An Encoder-Decoder Neural Network Model for Sequence to Sequence Mapping,’ Interspeech 2017, available: <https://pdfs.semanticscholar.org/7703/a2c5468ecbee5b62c048339a03358ed5fe19.pdf> [accessed 24 Aug 2020].

Schuster, M., Paliwal, K.K. (1997) ‘Bidirectional recurrent neural networks,’ IEEE Transactions on Signal Processing, 45(11), 2673–2681

Sharma, S., Sharma, S., Athaiya, A. (2020) ACTIVATION FUNCTIONS IN NEURAL NETWORKS [online], available: <https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf> [accessed 18 Aug 2020].

Singh, P. (2020) Multi-Head Self-Attention in NLP [online], Oracle.com, available: <https://blogs.oracle.com/datascience/multi-head-self-attention-in-nlp> [accessed 24 Aug 2020].

Smith, S., Kindermans, P.-J., Ying, C., Quoc, V., Le Google Brain (2018) Published as a Conference Paper at ICLR 2018 DON’T DECAY THE LEARNING RATE, INCREASE THE BATCH SIZE, available: <https://arxiv.org/pdf/1711.00489.pdf> [accessed 24 Aug 2020]

Stanford (2019) CS 230 - Recurrent Neural Networks Cheatsheet [online], Stanford.edu, available: <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>.

Sundermeyer, M., Ney, H., Schluter, R. (2015) ‘From Feedforward to Recurrent LSTM Neural Networks for Language Modeling,’ IEEE/ACM Transactions on Audio, Speech, and Language Processing, 23(3), 517–529, available: <https://pdfs.semanticscholar.org/f9a1/b3850dfd837793743565a8af95973d395a4e.pdf>.

TensorFlow (2020) Architecture | TFX [online], TensorFlow, available: <https://www.tensorflow.org/tfx/serving/architecture> [accessed 18 Aug 2020].

Thiruvengadam, A. (2019) Transformer Architecture: Attention Is All You Need [online], Medium, available: <https://medium.com/@adityathiruvengadam/transformer-architecture-attention-is-all-you-need-aeccd9f50d09> [accessed 24 Aug 2020].

Tillmann, C., Xia, F. (2003) A Phrase-Based Unigram Model for Statistical Machine Translation, available: <https://www.aclweb.org/anthology/N03-2036.pdf> [accessed 24 Aug 2020].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Brain, G., Research, G., Jones, L., Gomez, A., Kaiser, Ł., Polosukhin, I. (2017) Attention Is All You Need, available: <https://arxiv.org/pdf/1706.03762.pdf>.

Wang, C.-F. (Ed.) (2019) The Vanishing Gradient Problem [online], Medium, available: <https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>.

West, M. (2020) Explaining Recurrent Neural Networks [online], Bouvet Norge, available: <https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks> [accessed 18 Aug 2020].

Wilson, A., Recht, B., Jordan, M. (2018) A Lyapunov Analysis of Momentum Methods in Optimization, available: <https://arxiv.org/pdf/1611.02635.pdf> [accessed 24 Aug 2020].

Wu, S., Dredze, M. (2019) Beto, Bentz, Becas: The Surprising Cross-Lingual Effectiveness of BERT, available: <https://arxiv.org/pdf/1904.09077.pdf> [accessed 24 Aug 2020].

Xiangyi, C., Zhiwei, S., Hong, M. (2020) Understanding Gradient Clipping in Private SGD: A Geometric Perspective, available: <https://arxiv.org/pdf/2006.15429.pdf> [accessed 24 Aug 2020].