A Robust Deep Bidirectional Model With Lower Parameters Size And Better Sentence Prediction Using Deep Learning And Modeling Techniques

Author

Muhammad Shah Jahan

Reg. Number

00000275028

Supervisor

Dr. Muhammad Usman Akram

DEPARTMENT OF SOFTWARE ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

DECEMBER, 2020

A Robust Deep Bidirectional Model With Lower Parameters Size And Better Sentence Prediction Using Deep Learning And Modeling Techniques

Author
Muhammad Shah Jahan
Reg. Number
00000275028

A thesis submitted in partial fulfillment of the requirements for the degree of
MS Software Engineering

Thesis Supervisor
Dr. Muhammad Usman Akram

Thesis Supervisor's Signature: ___________________________________

DEPARTMENT OF SOFTWARE ENGINEERING
COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY,
ISLAMABAD
DECEMBER, 2020

DECLARATION

I certify that this research work titled "*A Robust Deep Bidirectional Model With Lower Parameters Size And Better Sentence Prediction Using Deep Learning And Modeling Techniques* " is my own work. The work has not been presented elsewhere for assessment. The material that has been used from other sources it has been properly acknowledged / referred.

Signature of Student

Muhammad Shah Jahan

Reg. Number

00000275028

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Thesis is also according to the format given by the university.

Signature of Student
Muhammad Shah Jahan
Reg. Number
00000275028

Signature of Supervisor
Dr.Muhammad Usman Akram

Copyright Statement

Acknowledgements

All praise and glory to Almighty Allah (the most glorified, the most high) who gave me the courage, patience, knowledge and ability to carry out this work and to persevere and complete it satisfactorily. Undoubtedly, HE eased my way and without HIS blessings I can achieve nothing.

I would like to express my sincere gratitude to my advisor Dr. Muhammad Usman Akram for boosting my morale and for his continual assistance, motivation, dedication and invaluable guidance in my quest for knowledge. I am blessed to have such a co-operative advisor and kind mentor for my research.

Along with my advisor, I would like to acknowledge my entire thesis committee: Dr. Sajid Gul Khawaja and Dr. Arslan Shaukat for their cooperation and prudent suggestions.

My acknowledgement would be incomplete without thanking the biggest source of my strength, my family. I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of my life and my loving brothers and sisters who were with me through my thick and thin.

Finally, I would like to express my gratitude to all my friends especially Muhammad Talha Riaz and the individuals who have encouraged and supported me through this entire period.

Dedicated to my Late parents, supportive brother and sisters whose tremendous support and cooperation led me to this accomplishment

Abstract

In transfer learning, a model is pre-trained on a large unlabeled dataset and then fine-tuned on downstream tasks. These pretraining and fine-tuning models are powerful and produced the best results on Natural Language Processing (NLP) tasks. These models are unidirectional but BERT introduced the first full deep bidirectional model which can read input from both sides of the input. BERT was pre-trained on Wikipedia and Book corpus dataset and fine-tuned with an extra layer. We present a replication study of BERT and provide a detailed analysis of the effect of hyperparameters during pre-training on downstream tasks. Due to the public unavailability of the Books Corpus dataset, we pre-trained the BERT from scratch on Wikipedia (2100M) and compares it with our model which trained on Wikipedia (531M). Our model Modified BERT "MBERT" achieves better results on GLUE (74.94) which consists of 8 tasks excepts STS-B, SQuADv1.1(57.40/69.50) and SQuADv2.0(56.19/59.38) dataset while saving pretraining from 53 hours to only 17 hours, six times less computational power and was also trained on four times smaller dataset. We also present a detailed study of why MBERT achieves these results on the SQuAD dataset.

Key Words: *MBERT, BERT, bidirectional language modeling, language modeling, modified BERT, transformer.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

CHAPTER 1: INTRODUCTION

Language models are used in Natural Language Processing tasks. These models were trained on supervised data and then transfer knowledge to downstream tasks. Due to the limitation of human-labeled supervised datasets these models were trained on limited data and produce limited results on NLP tasks such as Question Answering, Classification, etc. Data is doubling which produced very big unsupervised datasets so these models started to train on these big unsupervised datasets like WIKIPEDIA, BOOKSCORPUS, etc, and then fine-tuned on a supervised data set. These new pretraining techniques revolutionize transfer learning and produced SOTA results. These pre-trained models were built upon RNN architecture stores limited memory so when a sentence goes big the memory required to store the relationship also goes big which makes the model very limited and slow. Transformer architecture revolutionizes the Pretraining of Language models because it removes the RNN and trained the model only on the Attention mechanism. All of these models were uni-directional which means these models can read from left-to-right or from right-to-left but BERT pre-trained the model bidirectionally by using Mask Language Model(MLM)[1] and outperform the existed unidirectional language models and produced SOTA results. This research presents a modified BERT language model.

1.1 Background&Motivation

The data volume is doubling every year and 79% of data is text data. Google is the most used Search Engine which performs the 92.6% of search and 10% of these searches gives out of context results because these searches cannot cover the context of a search. Previous techniques like word2vec and GloVe used very shallow models means very limited memory for information due to which Models did not take the context of a word into account. For example:" I am standing on the bank of the river" now the shallow model will confuse with a bank as the shore of a river or a financial institute. The result of queries changes when the context is taken into account. The bidirectional model increases the speed of search and gains a deeper context because it read the sentence from both ends which increases the speed and memory. The context matter in all the tasks like in Question Answering, text Classification, Semantic Analysis, and Machine translation much better than an average human. Shortage of training data because of a few thousand or a few hundred thousand human-labeled training data. We have a lot of unsupervised data like Wikipedia. Train on many unsupervised data and fine-tune on specific data. The bidirectional model impacts 1 in 10 searches and works on complex queries.

Transfer learning is a training of a model on a large text-corpus and transfers that knowledge to a downstream task[2]. Pre-training was done via supervised learning [3]but now it is done via a large corpus of unsupervised learning. Pre-training of language models has proved to be very effective such as Natural Language inference (NLI) [4, 5]and paraphrasing [6]at sentence-level and Question Answering and entity recognition at token level [7]. Unsupervised pre-training is very effective due to the abundance of data [8]like Wikipedia, Books corpus, News and some domain-specific dataset like PUBMED and PMC[9], Scientific[10] which, make models very effective for downstream tasks. Unsupervised learning with span [11], semantic[12-14], lexical [15], syntactic [16] information make models more effective on downstream tasks. A large pre-training objectives [17], unlabeled Datasets[18, 19], benchmarks [20, 21]and _netuning ,methods [22, 23]are available. Pre-trained models built upon transformer architecture have produced state of the art(SOTA)results due to better use of parallel computing. These models are also used in other domains specific [24]like the business [25], medical [26, 27], and science [28]. The performance of downstream tasks directly depends upon the re-training of the model. The pre-training of models contains the size of a dataset, batch size, step size, sequence size, parameters, layers, hidden layers, attention heads, cross-layer sharing, and diversity of dataset.

Training of language models on large unsupervised datasets like Wikipedia, Books corpus and then use this gained knowledge on downstream tasks become an effective technique of Natural Language Processing. Almost all of these models were unidirectional and consist of RNN but with the introduction of the transformer, [29]developed a deep bidirectional model BERT.BERT reads the input from both left-to-right and right-to-left with a Masking Language Model (MLM). BERT outperforms the unidirectional models like ELMO[30].

Training of the BERT[29] model on a large unsupervised dataset is computationally very expensive and requires a lot of pretraining time. Training of BERT for 1M steps with smaller batch size and learning rate takes required more than 50 hours of training. BERT pre-trained with longer sentences make the pretraining very computationally expensive.BERT was deep but not enough as many other existed models showed that a deeper model generates better results. BERT pre-trained with bigger input layers that limit the number of hidden layers. Smaller attention heads reduce the performance because BERT works on transformer architecture which solely depends upon the attention mechanism.

Training of Language models on large datasets is computationally expensive. Trained models BERT, XLNET, XLM, ALBERT, ROBERTa,[18], etc. all trained on different datasets which means it is complicated to measure the performance of these models like BERT was trained on Wikipedia and Books corpus but ROBERTa which is a replication of BERT is trained on 160+ GB of different datasets which is 13 times more with no Next sentence prediction(NSP). Pretraining of BERT with different parameters could save a lot of time and computational cost and could produce the same or better results even on a smaller dataset.

ROBERTa increased the batch size and decrease the step size and used a deeper model but ROBERTa was pre-trained on 13 times more and diverse datasets (160GB) which overshadow the effect of these hyperparameters. XLNet built upon transformer-xl architecture which used permutation of factorization, segment recurrence mechanism, relative encoding scheme, and trained on 126GB of the dataset.FreeLB used adversarial training to minimize the maximal risk for label-preserving input perturbation It created a virtual adversarial example embedding space and perform parameter updating. AlBERT introduced parameter reduction by factorized embedding parameterization and cross-layer parameter sharing and use inter sentence coherence loss. AlBERT used sentence order prediction (SOP) instead of NSP. Many papers used BERT model and pre-trained it on different data like Medical[26], scientific text [10], Chinese [31]. [11] used MLM at span level and some used distillation methods.

Many models that used BERT were either trained on a bigger and diverse dataset and remove NSP. Some used BERT to pre-trained on domain-specific datasets. Paper used bigger batch and small step size produced better results but the problem is none of any paper which built upon transformer architecture and used BERT model analysis the effect of different parameters on BERT because these paper either used BERT as it is or trained it on bigger datasets and changed Settings like XLNet, ROBERTa and AlBERT.

We present a new model which is a replication of BERT with different parameters. Unlike ROBERTa we trained our model on a smaller dataset Wikipedia (531M) while BERT on 2100M from scratch. As the Books corpus dataset went private and is not publicly available so we trained both models on Wikipedia MBERT (531M) and BERT(2100MM).our modification are as follows:(1) trained with longer batch size for smaller steps (2) more hidden layers (3) small input layers (IV) bigger attention heads (V) higher learning rate (VI)training on smaller sequences (VIII) trained with bigger vocabulary size (iX) task-specific fine-tuning.

MBERT reduces the training size from 53 hours for BERT to just 17 hours on google collab tpu. Fine-tuning took the same time for both. MBERT produces a better result on GLUE dataset where MBERT produces 75% against BERT (73%). MBERT faces difficulty on SQuAD (-9.4) and SQuadv2(-3.4) which we assumed due to limitations of memory MBERT faced during pre-training as MBERT used feedforward 2H instead of 4H and max_positional embeddings 512 from 1024.

1.2 Problem Statement

To Produce a robust deep bidirectional model for better sentence prediction with small training time and size using deep learning and modeling techniques. The purpose of this research is to explore the different settings for BERT and the Effects of these settings during pre-training on Downstream tasks.

1.3 Aims and Objectives

The major objectives of the research are as follow:

- Pre-training of BERT model from scratch with different settings and compare them
- Training the model on the smaller size but producing the same results
- Reducing the computational power required for BERT
- Reducing the Pre-Training Time for BERT

1.4 Structure of the Thesis

This work is structured as follows:

Chapter 2 What is the Language model and how it works
Chapter 3 Presents the Detailed Literature review and work of researcher on Language models
Chapter 4 Consists upon Methodology
Chapter 5 Consists upon the Experimental Setup and Results
Chapter 6 Concludes the thesis and reveals the future scope of this research.

Chapter 2

Language and Model Transfer

CHAPTER 2: LANGUAGE MODEL AND TRANSFORMER

Language modeling is not a whole new field but it got attention in the previous decade because of the improvement in the computational power of computers. Language models first pre-trained and then fine-tuned on downstream tasks with few parameters adding instead of training the whole model from scratch. Language models are pre-trained on a supervised, unsupervised, and semi-supervised dataset. Language models are of two types unidirectional and bi-directional. Language models built upon RNN, CNN, LSTM, and Attention architecture.

2.1 Language Model

Language models are one of the most crucial components of Natural Language Processing (NLP). These language models are the backbone of Google Assistant, Amazon's Alexa, Apples' Siri. The language model learns to predict the probability of a sequence of words in a sentence using different statistical and probabilistic techniques. These models analyze the data to predict the words. These models are vital for machine translation, speech recognition, and spell checking. For example, we have a sentence in English and we want to predict it in Urdu language model will analyze the whole sentence and then translate every word in the sentence to Urdu respectively. It is a machine translation component of NLP and with language models, we are unable to use it.

English: " I like Swimming"

Urdu: "مجھے تیرنا پسند ہے"

2.2 How Language Models work

The language model predicts the words by analyzing the text data and then interpret this data by feeding to the algorithm which then makes the rules for the context in natural language. Language then applies these rules to natural language task to predict or produce sentences. These models learn the basic characteristics and features of basic language which helps these models to understand new phrases.

Many probabilistic algorithms depend upon the requirement and purpose of the model. The models also depend upon the size and type of data and math these models use in it. For example, a model built to find the likelihood of the search queries and results of these queries has different requirements than a model used to generate automatic tweets for a Twitter boot.

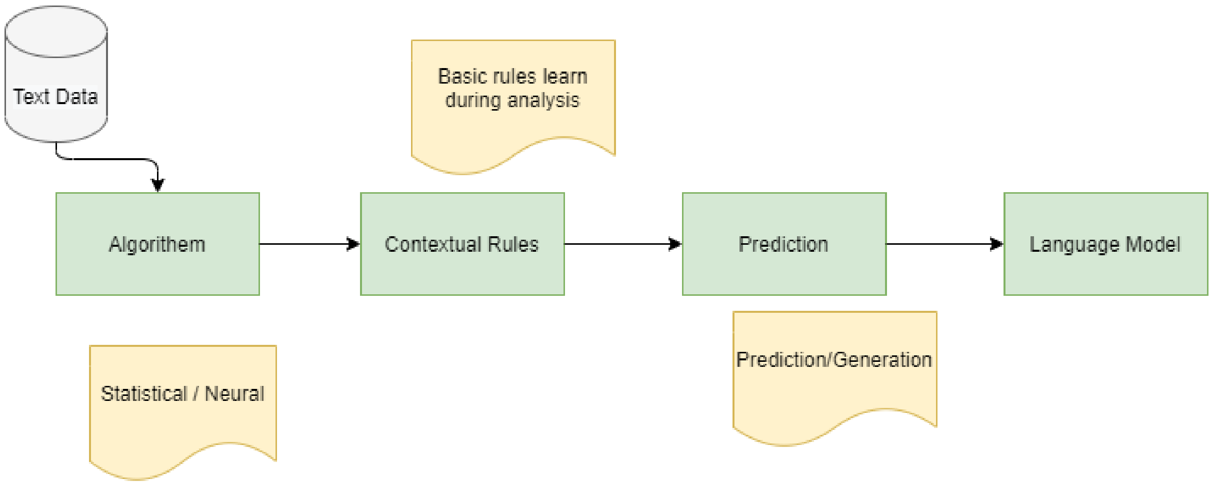

Figure 2.1: Language Model

2.3 Types of language models

2.3.1 N-gram

In the n-gram approach simple create a probability distribution for a sequence of ‘n’ where ‘n’ can be any number and can be letters, symbols, and words. It defines the size of the sequence of words by assigning the probability. If a gram is n=3 then it could look like “ I love swimming” and then the model assign probability using the sequence of n-size. Here the ‘n’ is the amount of text data that the model needs to consider like if n=2 it could be two words of sequence ‘please turn”. The n-gram models read the input in only one direction either from left-to-0right or from right-to-left. This is why these models are expensive to train because when a sentence size grows the computational power and memory requirements also grow that is why these models have limitations. It is very hard for these models to make a difference in a sentence where a word has the same context-free representation as a bank has the same context-free representation for “bank of river” and “bank account”. For example, we have a sentence “ I am standing at the bank of river” the n-gram models will start to read the input form ‘I’ then “am” but when it reaches the “bank” these models would represent it by “I am standing on” but not from “of river” and will have to wait till the model read the whole sentence. The sentence in the example is short and if we had an example of a very long sentence then n-gram models will lose their ability as the sentence grows then the memory decreases and computational power required also increases.

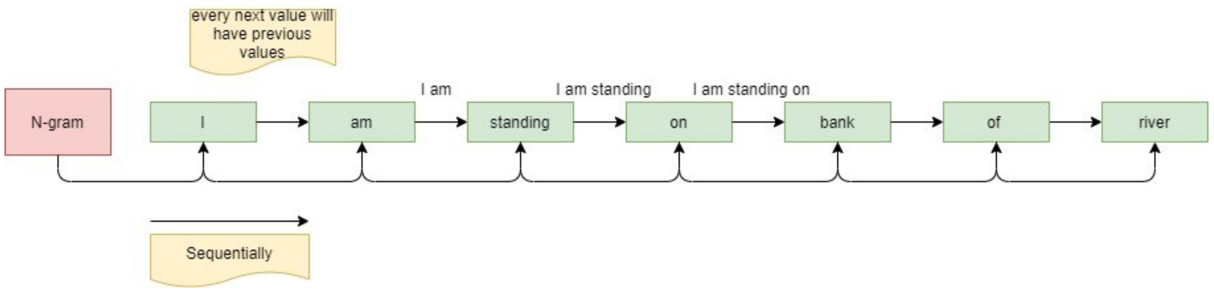

Figure 2.2: N-gram model

2.3.2 Unigram

Unigram is commonly used for information retrieval due to the simplicity of the model. It ignores conditioning context in its calculation because it evaluates each word independently in context. Unigram is used for the query likelihood model which finds the likelihood of query results on a document. The unigram takes one word in a sequence because here the value of n=1. For example, we have a sentence "I love swimming" then the unigram will evaluate each word separately.

I love Swimming			
Unigram	I	Love	Swimming

2.3.3 Bidirectional

Bidirectional models read the input from both directions which means these models can read from left-to-right and from right-to-left at the same time and unlike N-gram models these models can predict every word in the sequence which can bee seen in Figure 2.3. These models evaluate each word with every other word in the sentence. Due to the bidirectionality of these models more accurate than the N-gram models. For example, we have a sentence "I am standing on bank of river" the bidirectional model will start to read the input both forward and backward size means from "I" and from "river" and will predict that the "bank" is actually a shore of the river, not an account. These bidirectional models do not grow with an increase in sentence size and their required memory remains low when compared to n-gram models. Due to the dependent evaluation of sequences these models can predict a word in sequence from each other word in a sentence. These type of models mainly use in machine learning as google search engine use these bidirectional models to predict the search results while user typing and also suggest search queries.

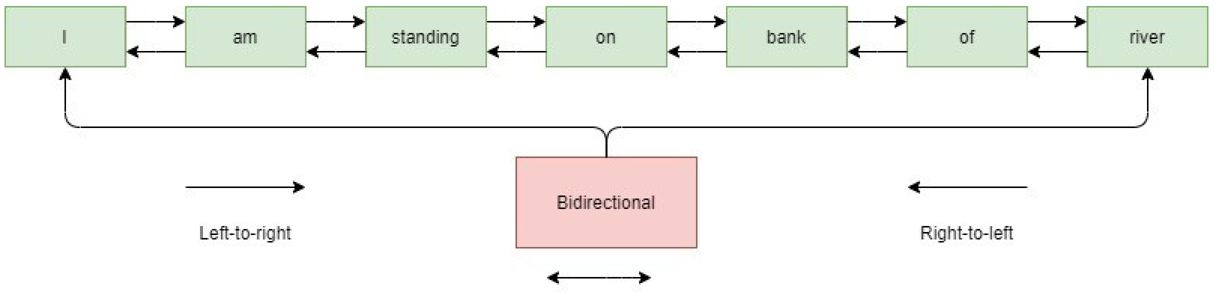

Figure 2.3: Bidirectional Model

2.3.4 Exponential

Exponential models are also called maximum entropy models because they rely heavily on entropy. These models are built upon the entropy principle which means the probability distribution with most options will be most accurate and will be selected. A model will be more suitable if it is built upon chaos and with zero assumptions. This type of model is complex than n-gram where we only put input to some algorithms and then the model learns basic presentations but here in these models simply put the input sentence to an equation which then evaluates the text by combining n-grams and feature functions. This type of model leaves the parameters in an ambiguous form instead of sizing individual grams and specifies features and parameters for specific results. These models are used to design the maximum entropy which in results minimize the amount of statical choices need to be made and increases the trust level of the user by giving only related results.

2.3.5 Continuous Space

The large datasets contain many unique and rarely used words and the presence of these words can create problems in linear models like n-grams. This type of model is best suited to these types of problems because these models represent words in a sentence as a non-linear combination of neural network weights. These weights are assigning to every word and this process is called word embedding. In linear models' presence of these unique words increase the size of the word sequence and weaken the pattern that informs the results. This problem is solved by weighting words in a non-linear distributed way which allows these models to learn more approximate words and can handle unknown words. One of the main advantages of using these models that understanding these models does not rag by immediate words.

2.4 Pre-training

Pre-training of a model is like human training where a human first learn something from experience and then apply that experience to solve another task without learning from scratch. In pre-training, a model is trained on a dataset where it forms parameters which then use to solve other tasks. If we have a task of classification then train the model on some dataset where the model can form parameters and initialize the weights of the neural network randomly. These weights are random so that when the model optimizes and starts to make fewer mistakes than you can save it. For example, we have a classification task and we trained our model on some image dataset and when the model start to make fewer mistakes means low error rate then save it.

Now apply this same model to a new and different task of image classification. Now we do not need to train this model from scratch, we just need to set some of its parameters and this model will be ready to perform the classification on this new task. We do not need to randomly initialize weights and can use that saved weight for this task and this will save a lot of training time and effort. The pre-training can be supervised, unsupervised, and semi-supervised.

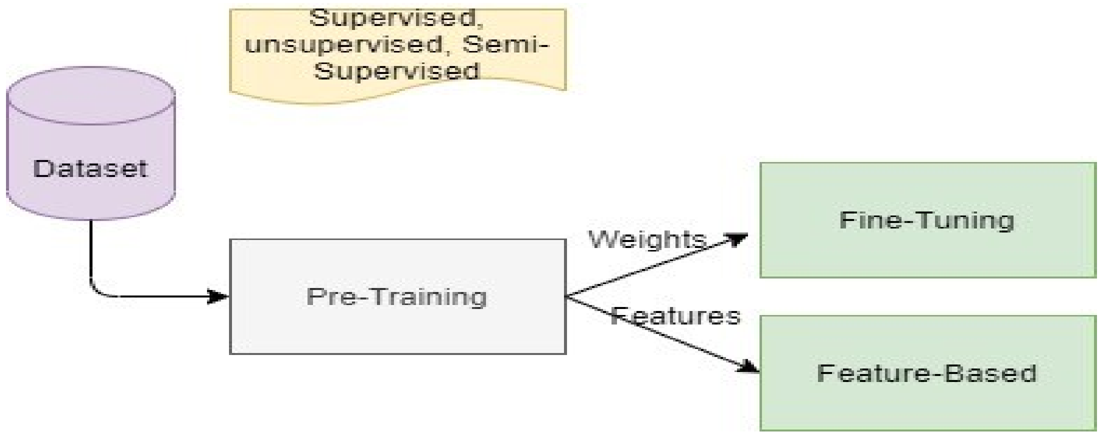

Figure 2.4: Pretraining

2.4.1 Supervised pre-training

Supervised learning is a type of learning where a model is trained on a supervised dataset. This labeled dataset has a 'y' value against every 'x' value which means every value has a target value. Models pre-trained on labeled data have the best accuracy as models learn during pretraining about the data and target values it is also the simplest type of pretraining and at the start of pretraining most of the models were pre-trained on labeled data. Models trained on supervised data produces best results but these models have limitation because of limited supervised datasets available and size of these datasets as language models needs to train on very large dataset sets so that these models can perform on other subtasks but here we few thousand or few hundred thousand of labeled data. Shortage of supervised data is the main cause that these types of models can only use for small tasks and where training data is not an issue. Due to this reason few models use supervised pre-training.

2.4.2 Unsupervised

Due to limited labeled dataset available for pre-training of language models started to pre-train on unsupervised datasets. We have an abundance of unsupervised datasets like Wikipedia, Books corpus, news, and many more. The size of these datasets is so big that they easily cover the supervised datasets. In the unsupervised dataset we have inputs of values but we do mot have target outputs which means we have "x" but we do not have "y". the models pre-trained on a large unsupervised dataset are able to perform better on another subtask because in this type of pretraining the models so much new and unique words and form parameters that can predict other unsupervised tasks. The unsupervised pretraining make the model more resilient to the unique inputs.

2.4.3 Semi-supervised

Unsupervised learning is not unsupervised due to the supervision of weights that guide to the weights in neural networks which drive form data itself. Semi-supervised pre-training has both advantages of supervised and unsupervised pretraining because it pre-trained on both labeled and unlabeled datasets. This type of pretraining works in situations where a model needs to learn from both labeled and unlabeled data simultaneously. These types of models when learns both from supervised and unsupervised data then these models will be able to learn more and form parameters that can perform better on other downstream tasks. These models can be pre-trained in situations where we have a lot of unsupervised data and some of the supervised data and we want our model to learn from both. There is very rare situations where we use a semi-supervised datasets and few models pre-trained on semi-supervised datasets.

2.5 Fine-tuning

When a model pre train on some data then it learns some weights and form parameters which we save for later use. When we apply a pre-trained model on other downstream tasks, we use these weights to train our model only for that task instead of pretraining from scratch. It saves a lot of training time and the other advantage of this is that the model uses its previous experience on a new problems.

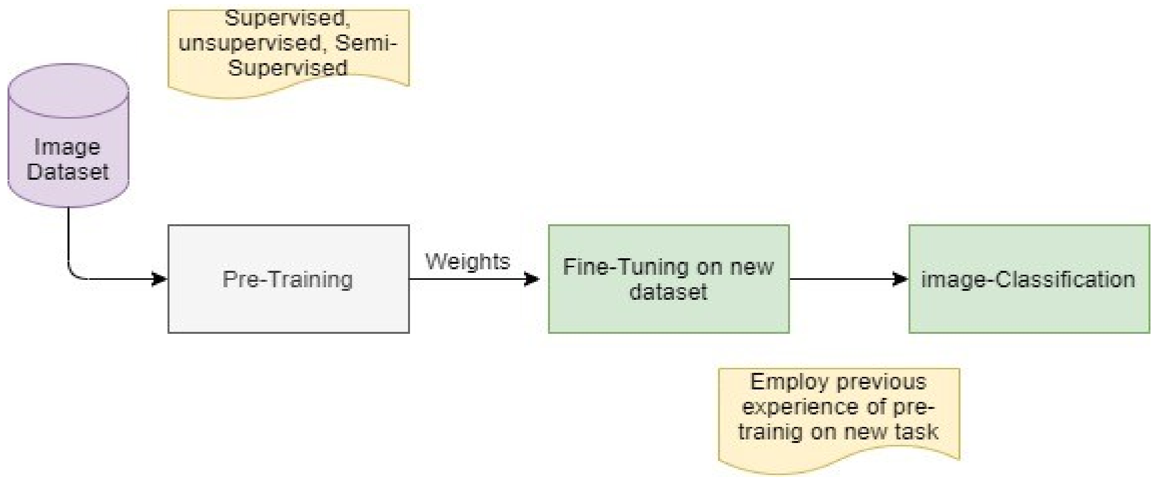
Supervised, unsupervised, Semi-Supervised
Image Dataset
Pre-Training
Weights
Fine-Tuning on new dataset
image-Classification
Employ previous experience of pre-trainig on new task

Figure 2.5: Fine-tuning

If we pre-trained a model on image datasets and pre-trained for an image classification task. Now the models initialize some weights and when the error starts to minimize due to self-learn from data we save this model. Now we can use this saved model on another task of image classification where we just need to change some parameters to adjust our model otherwise, we do not need to train the model from scratch and just need to train on a very small dataset for a specific task.

2.6 Feature Base

A feature-based approach is in which a model used task-specific architecture that includes pre-trained representations as an additional feature. This type of approach is used where embeddings are not mapped from word but the context of its surroundings. A feature-based approach is used to generate contextual embeddings for downstream tasks. In fine-tuning a classification layer is added at end of the pre-trained model but in the feature-based approach extract, the specific feature from the pre-trained model makes it more suitable for a specific task. It is slow in the process because adding a classification layer of subtask and extracting very specific features from a very large pr-trained model is very difficult and slow as it required task-specific architecture.

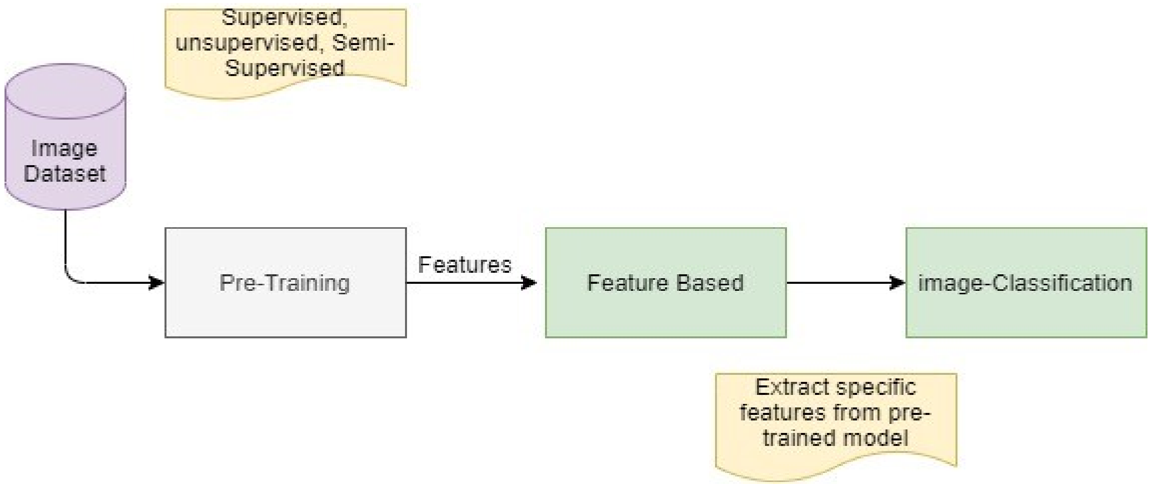
Supervised, unsupervised, Semi-Supervised
Image Dataset
Pre-Training
Features
Feature Based
image-Classification
Extract specific features from pre-trained model

Figure 2.6: Feature Based

2.7 Transformer

A model is built upon an architecture. Language models built upon convolutional neural network (CNN) [32],recurrent neural network (RNN) [33], long-short term memory (LSTM)[34], and attention [35].

2.7.1 RNN architecture

At the start, language models were built upon CNN architectures but with the involvement of RNN and LSTM and advancement of computer hardware, the use of neural networks increase. Most of the best performing language models before the introduction of Transformer architecture were built upon RNN. The RNN architecture reads the input of sentence form one direction either from left-to-right or from right-to-left. For example, we have an example " I love you" and want to translate it in Portuguese then RNN will start reading from 'I' than'love' and then 'swimming' and store it into Context vector which then gives it to the decoder to translate.

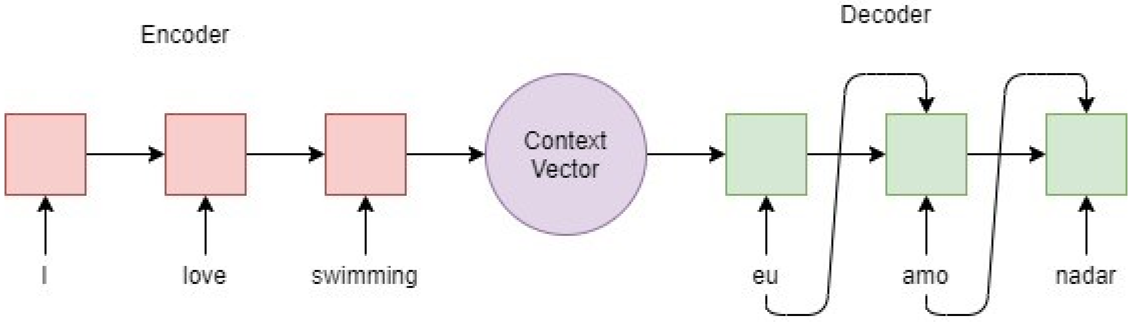

Figure 2.7: RNN based encoder decoder

A context vector is a fixed size vector that is used to compress all the information store in ti which could result in information lost. context vector takes all cells' outputs as input to compute the probability distribution of source language words for each single word decoder wants to generate. By utilizing this mechanism, the decoder can capture somewhat global information rather than solely to infer based on one hidden state. RNN uses encoder decodes with an attention mechanism. Figure 2.8 to figure 2.11 shows how RNN works and how it remembers its previous values using the attention mechanism.

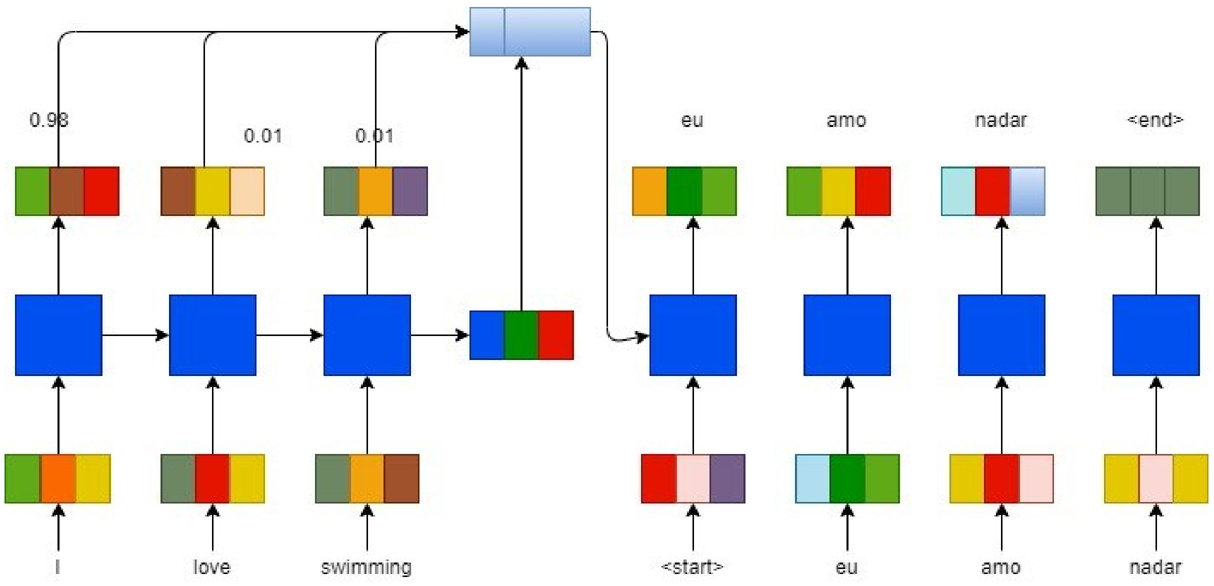

Figure 2.8: RNN encoder-decoder with attention

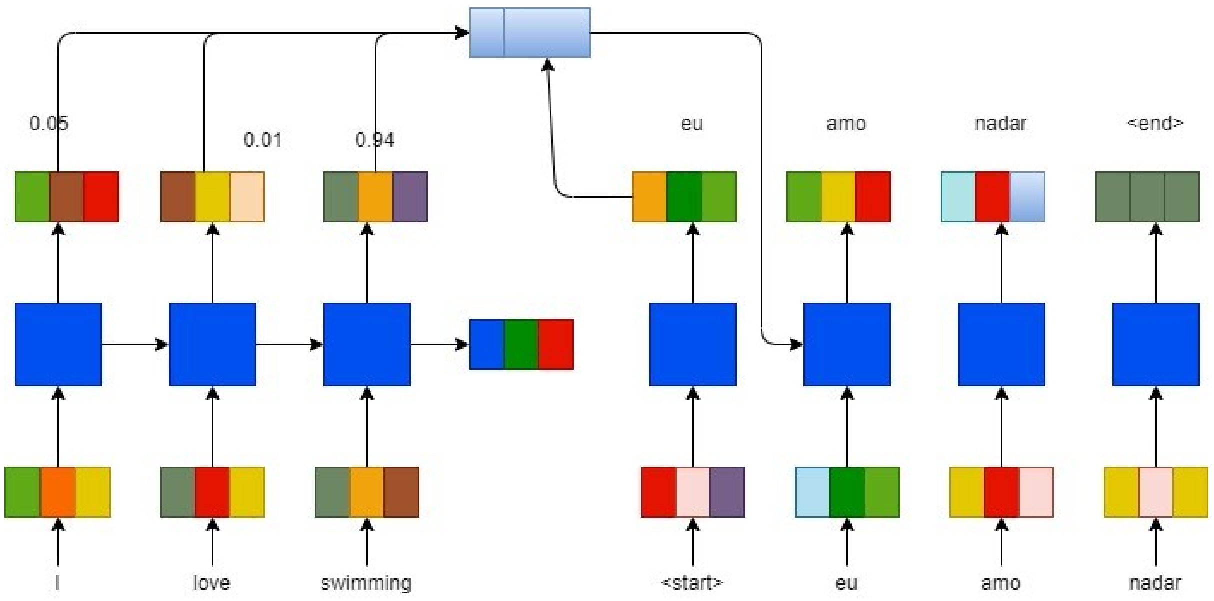

Figure 2.9: RNN encoder-decoder with attention(con)

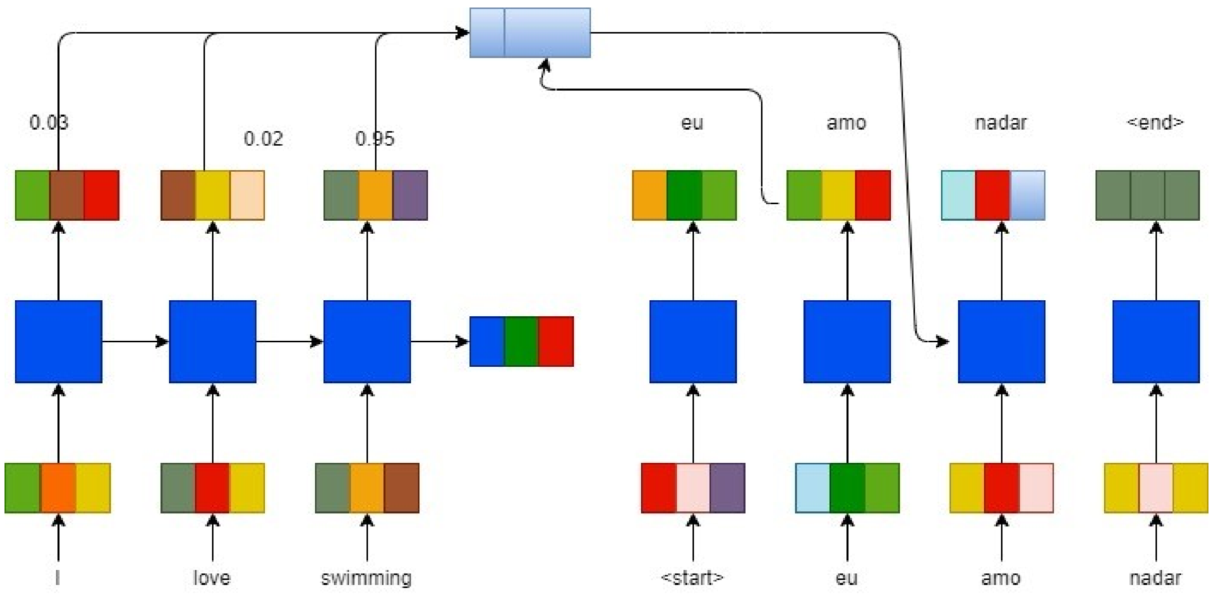

Figure 2.10: RNN encoder-decoder with attention (Con)

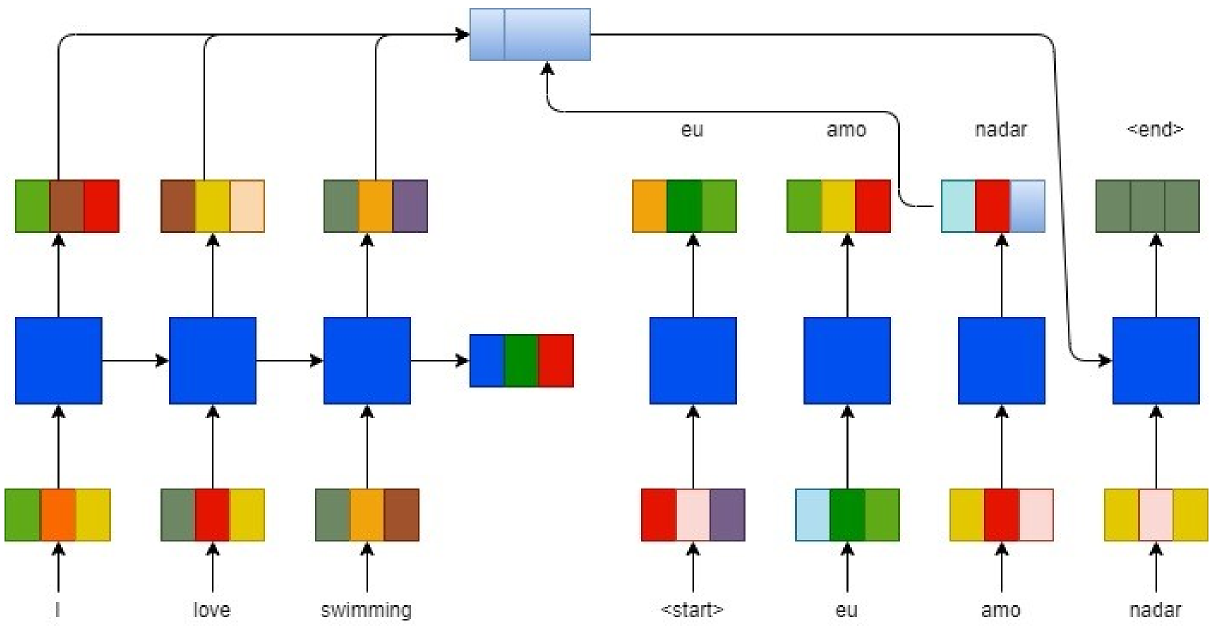

Figure 2.11: RNN encoder-decoder with attention (Con)

2.7.2 Transfomer Architecture

RNN heavily depends upon the size of the context vector which is also limited and when a sentence grows it becomes a problem for the context vector. One disadvantage of RNN in language models is that it is unidirectional which means it can read input from one-directional only which makes it really slow. When the size of a sentence grows the memory required for RNN increase drastically due to which RNN lost the relationship between words in a sequence.

For example, if we have an input of “I was standing on bank of river” RNN can find which bank it belongs to either to the shore of a river or financial institution will start sequentially from left “I” and sequentially compute till the end where it finds the word ‘river'. Then RNN can relate that the bank in this sentence belongs to the shore of the river. With an increase in steps, the computational cost of RNN also increases. To solve this problem a new architecture was introduced which wholly depends upon the concept of attention and remove the RNN shown in Figure 2.14. In the example showed in Figure 2.12 The transformer relates ‘it’ with ‘animal’ and ‘street’ but maps it according to the animal due to the word ‘tired’ But in the 2nd example, it maps it according to ‘street’.

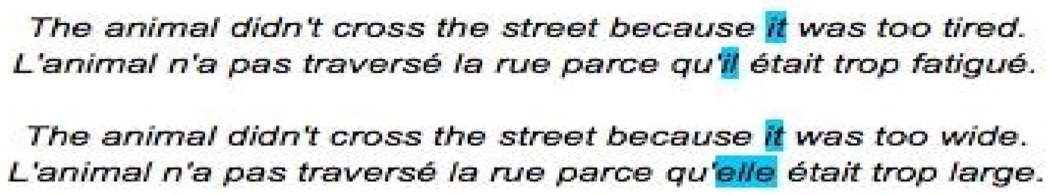
The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.
The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.

Figure 2.12: English to French translation[36]

As we encode the word ‘iI’, one head is focusing on the animal while the other focusing on the tired. The model representation of the word ‘It” bakes in some of the representation of both animal and tired shown in Figure 2.13

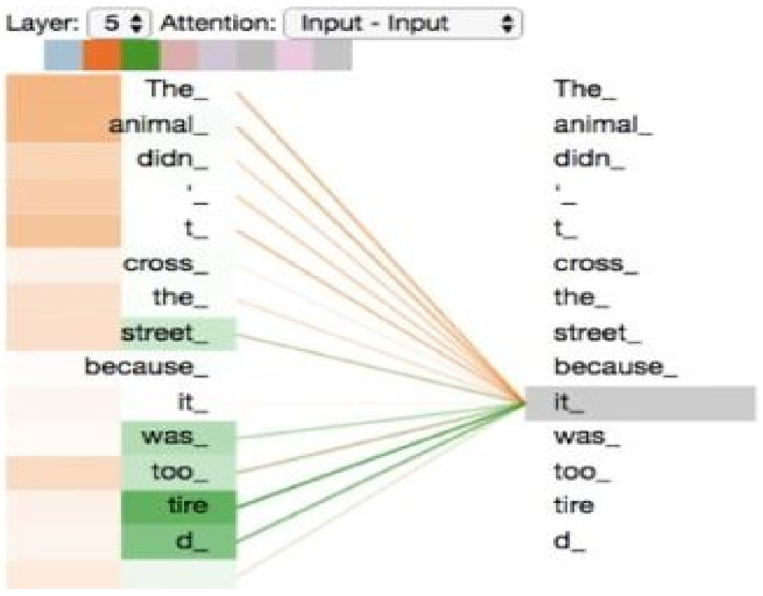

Figure 2.13: English French Example[36]

In a transformer, a self-attention mechanism is being used that directly maps the relationship between all words in a sentence regardless of their respective position which means independence. A transformer can find in one step of the bank. The transformer compares the "bank" with every single word in the sentence and then assigns an attention score. How much a word is contributing to the next presentation of the bank. The attention score is fed to fully connected networks as the weight of the weighted average of all words to generate a new representation for 'bank'. Reflecting that a sentence is referring to a river. Transformer architecture only depends upon the attention mechanism.

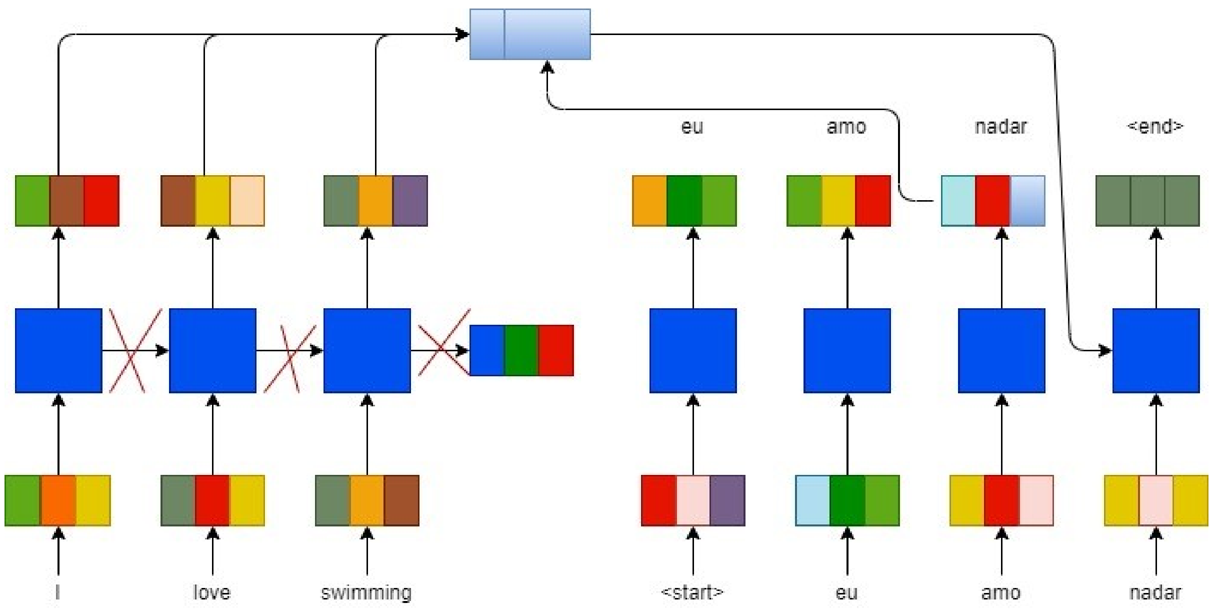

Figure 2.14: Transformer without RNN

Transformer model which eschewing recurrence and rely on attention to draw global dependence between inputs and outputs shown in Figure 2.15 & Figure2 .16. With 12 hours of training on p100 GPUs, we can achieve a better art of the state translation.

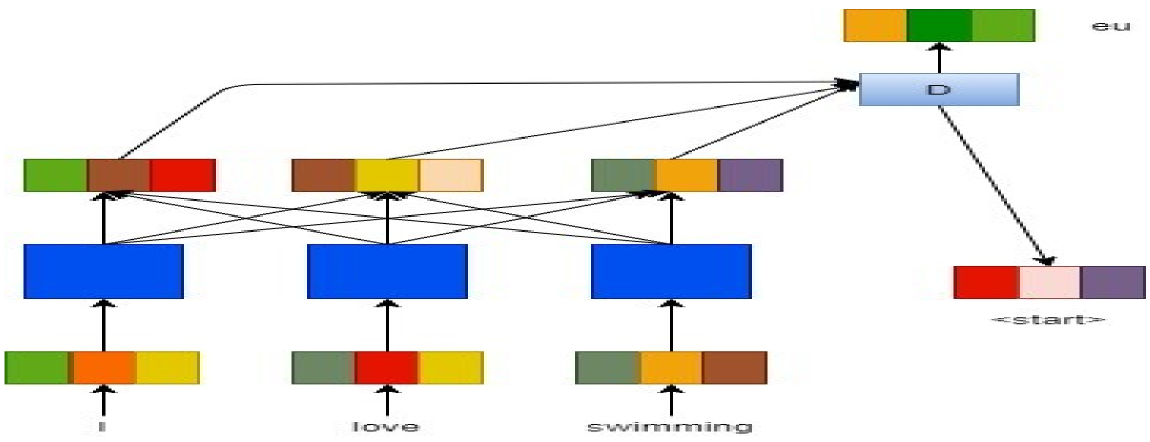

Figure 2.15: One step matrix calculation

Encoder maps the input into a sequence of continuous representation. The decoder then generates an output sequence one element at a time. The model is autoregressive consuming the previously generated symbols as additional input when generating the next shown in Figure 2.16.

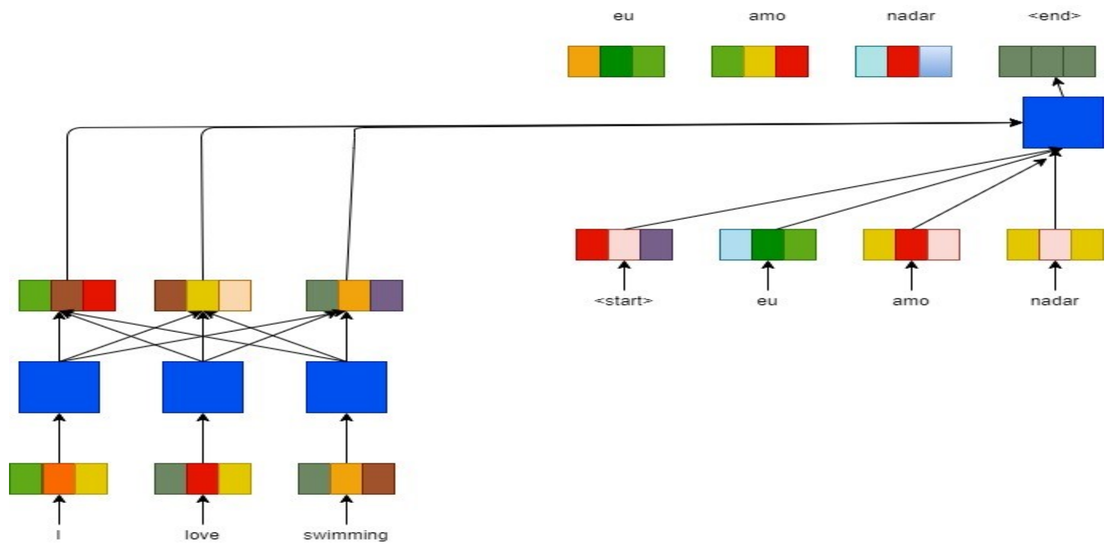

Figure 2.16: One step matrix calculation (con)

Adding relative positional information to every input in both encoder and decoder like 'I'=001,study=0001

$PE_{(pos,2i)=}\sin(pos/10000^{2i/dmodel})$

$PE_{(pos,2+1i)=}\cos(pos/10000^{2i/dmodel})$

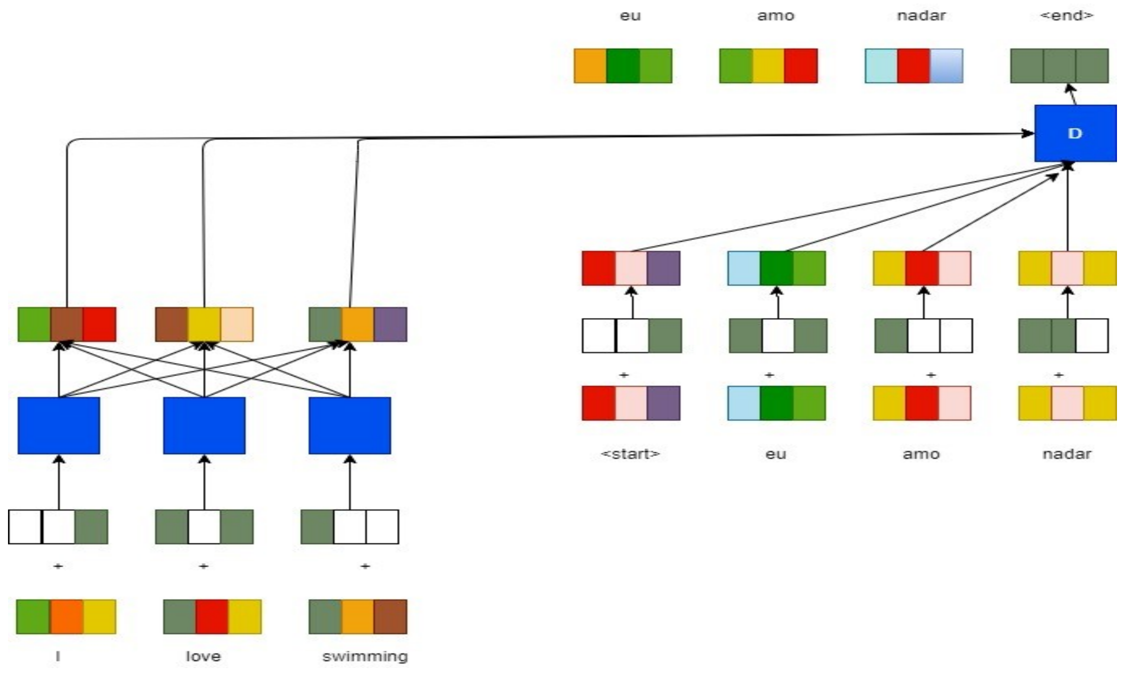

Figure 2.17: Positional encoding

Figure 2.18 and Figure 2.19 presents the transformer model architecture. The encoder has six identical layers and each one of these layers has two sub-layers. A positional encoding is applied in it to remember the global positional of inputs so that encoder can remember the positions. Multi-head attention is used instead of a single function and in multi-head attention queries, keys and values are being linearly applied and an attention function apply which outputs d-dimensional values. These values are then concatenated and projected.

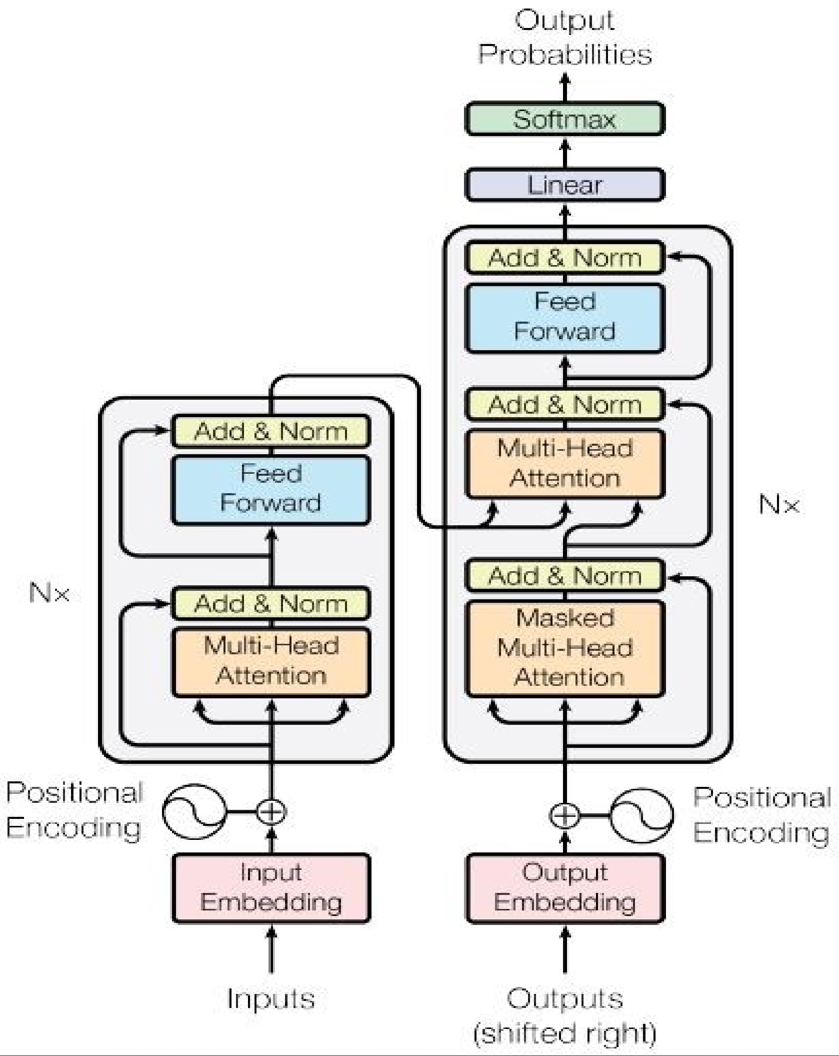

Figure 2.18: Transfomer model architecture[35]

Encoder-decoder attention layers queries come from the previous layer while the keys and values come from the encoder. It allows every position in the decoder at attending overall positions in the input sequence. The encoder contains self-attention layers which consist of the queries, keys, and values from the output of the previous layer. Each position in the encoder can attend all the positions in the previous layer of the encoder. Self-attention layers in the decoder map each input according to the position. Use Multitasked attention to prevent future words to be part of attention.

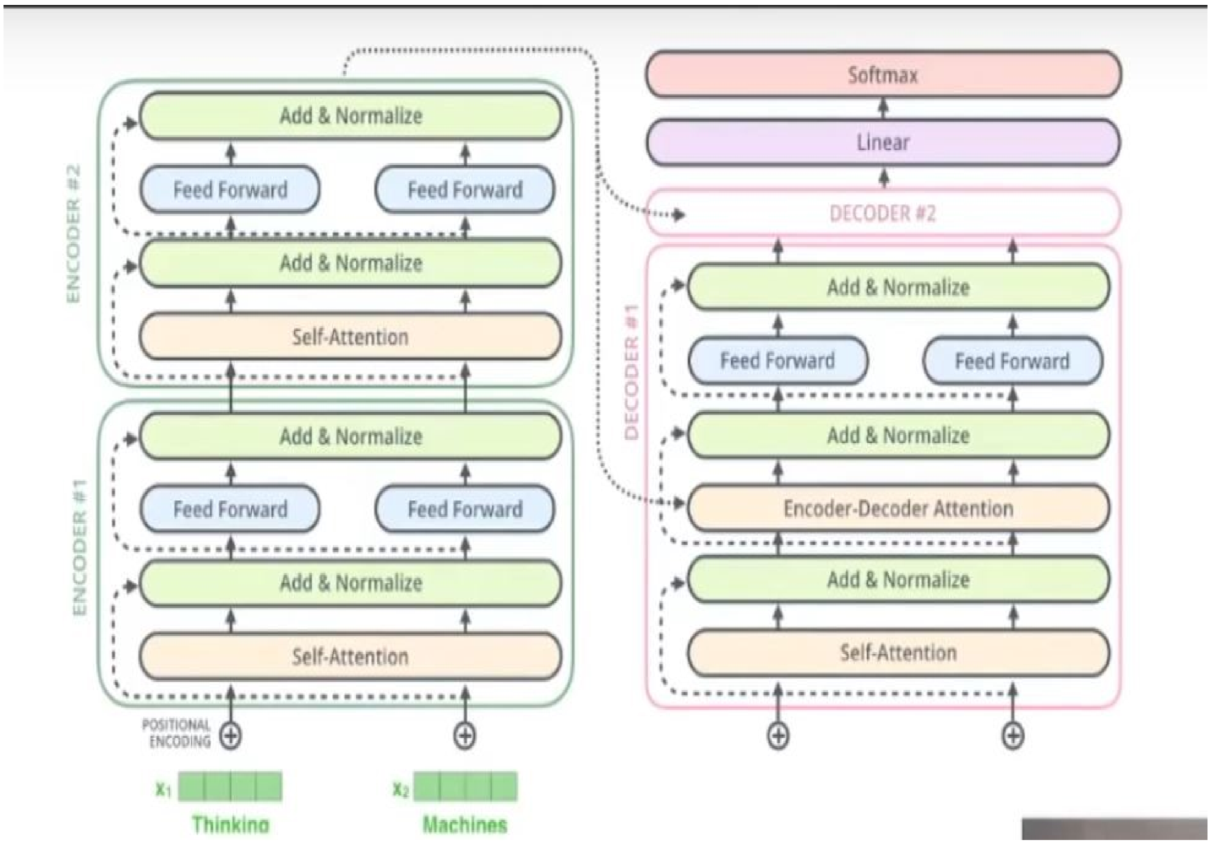

Figure 2.19: Inner working of Transformer[35]

2.8 BERT

BERT is a fir real deep bidirectional pre-Training unsupervised technique which uses the concept of attention (parallel computing) instead of RNN. BERT is naturally bidirectional that the model can be fine-tuned easily for downstream tasks. BERT models beat the state of the art results for many NLP tasks with no need for a special dataset BERT uses Wikipedia and Books Corpus. Pre-trained representations can be both contextual and context-free and contextual representation is unidirectional and bidirectional. For example, we have an example of "I was standing on the bank of river" the unidirectional models will generate a representation for each word, and for the word "bank" it can use words " I was standing on" to generate representation but will fail to understand the context of 'bank' till it read the word 'river' but here the bidirectional BERT will read the input from both sides simultaneously and in a short while will understand the context of the sentence because BERT read 'bank' and 'river' form both sides.

2.8.1 BERT, OpenAI, and ELMO

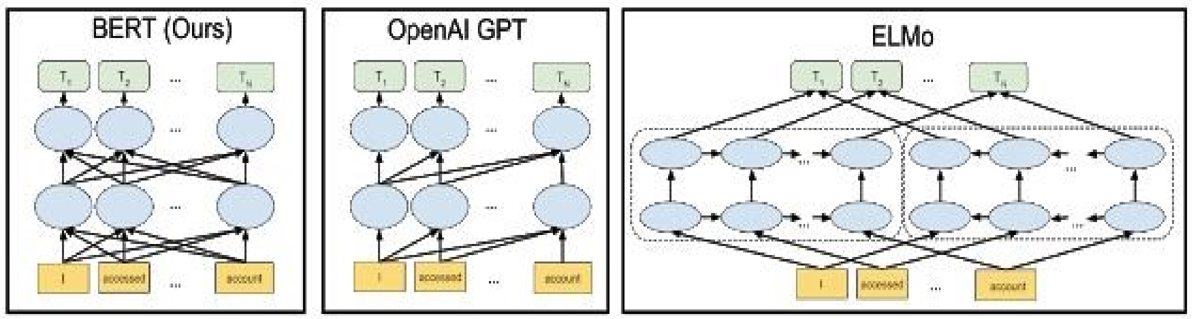

Figure 2.20: BERT, GPT, and ELMO[29]

BERT is bi-directional, OpenAI GPT is unidirectional and ELMo is shallowly concatenated both left-to-right and right-to-left to find out which bank we are referring to either shore of a river or financial institution. In unidirectional we have to read from left to right just to find it which means in this case we have to read the whole sentence but in bi-directional we can easily find it as a model is bidirectional. This is twice as expensive as a single bidirectional model this is non-intuitive for tasks like QA since the RTL model would not be able to condition the answer on the question this it is strictly less powerful than a deep bidirectional model since it can use both left and right context at every layer. BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach. . In bidirectional training each would indirectly see itself.to solve this we masked 15% of the tokens. Cross entropy loss is used to predict the original values. A pre-trained model can tackle a broad task of NLP just by adding a classification layer which we called Fine-Tuning

2.8.2 Pre-training and Fine-tuning BERT

The main advantage of BERT is that we have to pre-train our model just once after pre-training we just need to update some parameters in Fine-Tuning and then we can apply it on task. Pre-Training is a process in which the system is trained on large data from scratch but in fine-tuning which is the successor step of pre-training we just need to specify parameters and not from scratch means just change the values of some parameters. BERT model is first initialized with pre-trained parameters and all of the parameters are fine-tuned using labeled data from downstream tasks. [cls] is added into the input and [SEP] is used to separate the sentences. like question and answer can be seen in Figure 2.21.

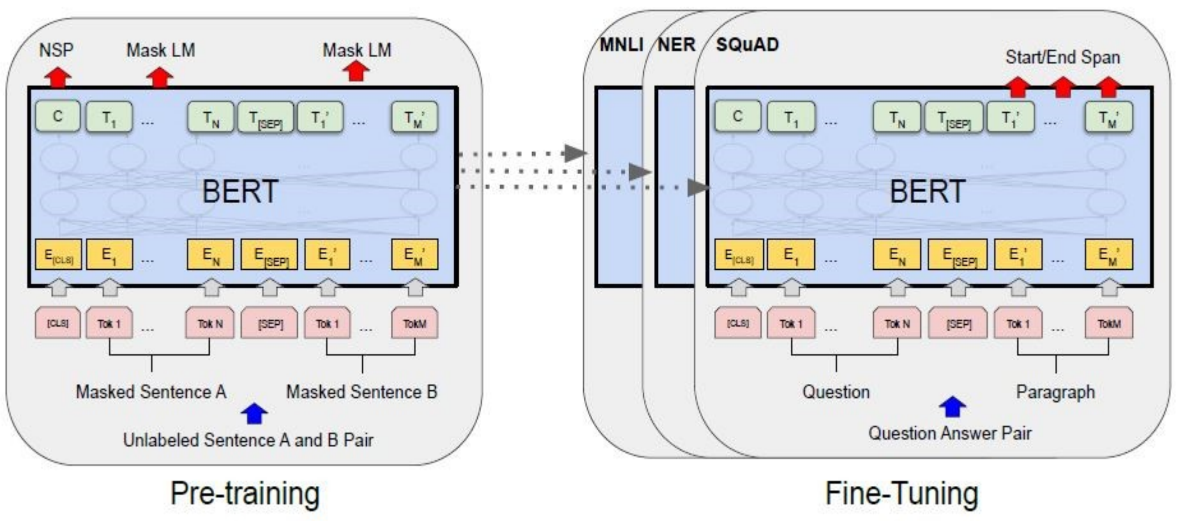

Figure 2.21: BERT Pre-training and Fine-tuning[29]

2.9 Summary

Language modelinig is a field in which a language model first pretrained on a large corpus of data and then it is fine-tuned for other similar task. Language models learns the knowledge during pretraining transfer this knowledge to downstream tasks just like we human do. In pretraining language models analysize the text and then feed it to some algorithem and then make some language vocabulary and then apply vocabulary in understanding other tasks just like human use previous experience in solving new problems which is called fine-tuning. To apply these pre-trained models on downstream tasks we have feature based and fine-tuninig methods but due to complexity and size of models use of feature base is very limited and most of language models use fine-tuning in downstream tasks. We have three types of pre training supervised pretraining, unsupervised, and semi-supervised pretraining. Due to shoragfe of human labled dataset and complexity of semisupervised data almost all big models use unsupervised pretraining but for fine-tuning still supervised datasets are used. We have mahor five types of language models n-gram, uni-gram, bidirectional, exponential and continues space. N-gram and bidirectional models are most used and produces the best resulst among all these types of models. N-grams models read the input in sequence from one direction which could be left-to-right or right-to-left while bidirectional models can read from both sides of the input which make these models to better understand the context and more powerfull then unidirectional models.language models first built upon CNN but then nwith advancement of RNN and LSTM it shifted towards RNN.RNN used an attention mechanism and a contextual vector to read the inputs in sequence and all omdels built upon RNN were unidirectional. The contextual power of the models were limited because these models can not understand context to words as length of sentence increase so the memory required by the models. Transfomer architecture removes the RNN and soly used attention mechanism which enable this architecture to work bidirectionally. BERT was the first deep bidirectional models as ELMO which called itself was not bidirectional as it read the data from both sides but then concatenate both inputs instead BERT read from both sides of input. BERT used MLM which restrain the model to see both inputs and save it to make multilayer context. BERT also used NSP which allow BERT to better ubnderstabd the context and let it to make relationaship between sentences.

Chapter 3

Literature Review

CHAPTER 3: LITERATURE REVIEW

Language modeling has been an interesting field for researchers to do Natural Language Processing tasks. Training of language models on unlabeled(unsupervised text) and then fine-tuned it on labeled dataset revolutionize the language modeling [37] [38].The advantage of using these techniques that we need to change a few parameters to fine-tune the pre-trained model on downstream tasks like [39] GLUE. There are different methods of pre-training and every method has been designed with an objective which includes machine translation[40], language modeling [37] [38], and masked language modeling (MLM)[29] [41]. Some models have used a different fine-tuning for each downstream tasks [38, 42], some used multi tasked[43]. Span prediction[11],entity embeddings[44] and different autoregressive pretraining[45-47]..Almost all of the earlier models were uni-directional which means these models can read the input from left-to-right or right-to-left. BERT[29] is a pioneer in bidirectional language modeling. BERT implemented the concept of MLM and NSP. before BERT ELMo[30] was called bidirectional model but it was not truly bidirectional as it concatenated the output of both right-to-left and left-to-right which doubles the data and also allowed the model to see itself which cause redundancy and also the computational power required for ELMo was much more then BERT. These language models were based upon Recurrent Neural Network (RNN) architecture.RNN has memory constraints which means when a sentence is big RNN loses the dependency due to a limit of context vector. [48] gives the concept of the only use of attention while training of model which removes the RNN which saves a lot of computational power and increase the memory of model which directly effect the better results on longer sentences.This new architecture can remembers sentence for longer length. Most of the language models are built upon tranformer architecture and making changes in BERT and some are making models on only tranformer architecture.

3.1 Models based upon RNN architecture

Jeremy et al. [38] presented a Universal Language Model Fine-tuning a transfer method can be applied to any downstream task and introduced key techniques for fine-tuning. It used techniques of gradual unfreezing, discriminative fine-tuning, and slanted triangle rates to retain previous knowledge. It reduced the loss by 18-24% on six datasets. It achieved a performance of 100 labeled examples where other required 100+.

Matthew et al. [30] presented ELMo which is a deep contextualized word representation that models characteristics (syntax and semantic) of word-use and how these vary across linguistic contexts. It learned functions of hidden layers of deep bidirectional model pre-trained on a large dataset. It shows that it can be added to existing models and improve the overall results. ElMo is first deep bidirectional model. It achieved 71% on GLUE.

Matthew et al. [49] presented a detailed study of how a choice of neural network (CNN, LSTM, and self-attention) of the bidirectional model affects the performance on downstream tasks. All models learn a contextual representation that outperforms word embeddings on four NLP tasks. biLMs is very useful for syntactic tasks due to the use of phrase representation.

Peters et al. [50] introduced a semi-supervised approach that learns from relatively little labeled datasets. A model is pretra9ined on an unsupervised dataset and then at each level compute encoding of context and use it on supervising settings. Due to the use of the neural language model, it encodes the semantic and syntactic role of words in context. It achieved 91.93% FI on CoNL2003 and 96.37% FI on CoNL2000.

Dai et al. [37] presented two approaches to using the unsupervised dataset to improve sequence learning with the recurrent network. First is the language model and the other is the use of sequence encoder which covert the input into vector form and then predict the sequence of input. These approaches are then used for supervised sequence algorithm means parameters from pretraining then use for training other supervised models. After pre-train with these two approaches, LSTM neural networks become more stable to train. The error rate is (7.24%) on IMBD, (16.7%) on Rotten Tomatoes, (15.6%) on20 Newsgroup, and (1.19%) on DBpedia.

Radford et al. [8] introduced the concept of unsupervised pre-training and then supervised fine-tuning due to the abundance of unsupervised data and limited labeled data. It also adds task-specific inputs during the fine-tuning stage to limit the change required in architecture for downstream tasks. It achieved better results on 9 of 12 tasks including 8.9% improvement on Stories Cloze Test, 5.7% on RACE, and 1.5% on MultiNL.

Shaojie et al. [51] presented a novel approach for modeling sequential data which uses root-finding to find the hidden layers that converge to some fixed point in deep sequence models. It is like to run an infinite depth feedforward network which is very expensive in other cases but here it just required constant memory regardless of the depth of the network. It runs on two existing models and it reduces the memory required for these models and improves the performance.

3.2 Models based upon Transformer Architecture

Dai et al[52]increases the memory power of the transformer beyond a fixed length by using context fragmentation, segment level recurrence mechanism, and relative positional encoding.it captures longer dependency and also resolves context fragmentation problems. Transformer-xl leaned 80% and 450% more dependency than RNN and vanilla Transfomer respectively and it was 1800+ times faster than a vanilla transformer. Tranfomer-xl achieved BPC/perplexity of 1.08 on text8,18.3 on WikiText-103,0.99 on wiki8 and 21.8 on Billion word. Yang et al. [46] Yang et al built XLNet on transformer-xl architecture by integrating Segment Recurrence Mechanism and relative encoding scheme. XLNet is both an autoencoding and autoregressive model that used permutation of factorization which covers longer sentences and it also removes the pre-train finetuning discrepancy which occurs in BERT. XLNet outperforms BERT in Race(+7.6), SQuADv1.1(+3.6), SQuADv2.0(+7.0), GLUE(+3.94) on six tasks including MNLI two values.

Lan et al. [53]used factorized embedding parameterization and cross-layer parameter sharing to reduce the model size during pre-training. The first one separates the hidden layers from the size of vocabulary and the second shares the parameters during pre-training. ALBERT also replaces the NSP with sentence order prediction (SOP). Sentence order prediction outperforms the next sentence prediction. Increasing width and depth beyond a specific limit degrade the performance. AlBERT produces much better results for GLEU(+7.3%),Race(+6.3%) and SQuAD(+18.4%).AlBERT has x18 fewer parameters and trained x1.7 faster.

Zhu et al. [54] used adversarial training to minimize the maximum risk for label preserving input permutation. Create a virtual adversarial example embedding space and perform parameter updating. When implemented on fine-tuning of a transformer-based language model it freeLB improves $\text{BERT}_{\text{base}}$(+1.1), ROBERTa (+0.3), on ARC task 85.44% and 67.75% and on Commence sense QA 72.5% to 73.1%.

Raffel et al. [55] it provides a unified language framework that converts language problems into text-to-text formate.it provides a way to convert an input into another output by converting the input into text-to-text formation. It also developed a very large dataset of 29TB which is called Colossal Clean Crawled Corpus.TS5 produced GLUE(89.9%), SQuADv1.1(90.06/95.64), and currently while writing this thesis TS5 is top of GLUE leaderboard.

3.3 BERT trained with different strategy

There are many models built upon BERT architecture. Beltagy et al. [10] pre-trained the BERT model on a scientific dataset instead of a general dataset like Wikipedia. It introduced the concept of in-domain vocabulary. And investigate the fine-tuning on frozen embeddings of subtasks. It achieved (79.27%) on Bio, CS, and Multi fields which is (5%) improvement over $\text{BERT}_{\text{Base.}}$

Joshi et al. [11] used MLM at span level which uses a single contiguous segment and summarizes the intended span. It masks random spans instead of tokens like in BERT and used span boundary representations to predict the masked span without relying on a single token. It used the same training dataset of $\text{BERT}_{\text{large and}}$ gains (94.6% & 88.7%) on SQuADv1.1 and SQuADv2.0 respectively.(79.6%) on Onto Notes and 82.2 on GLUE task.

Zhahg et al. [56] used a shelf-labeler that mapped the semantic labels into parallel embeddings and semantic integration which obtain a joint representation for downstream tasks. SemBERT left the NSP and used $\text{BERT}_{\text{Large}}$ model with a maximum input size of 128 and a learning rate of $2e^2$and achieved 82.9% on GLUE and (82.4/85.2) on SQuADv2.0.

Sun et al. [16] introduced a pre-training framework which first builds the task incrementally and then uses a continual multi-task approach to extract semantic, syntactic, and lexical information from these tasks. It used $\text{BERT}_{\text{Large}}$ model with 400K batch size for 4k steps with a $5e^5$ learning rate.It was pre-trained on English Wikipedia, Book corpus, Reddit, and discovery data and for chinses, it was collect data from an encyclopedia, news, and data from search engine.ERNIE2.0 outperforms the BERT and XLNet on 16 tasks including GLUE and Chines task. It achieved 80.6% on GLUE task.

Liu et al. [18] replicate the BERT model by training it on five large datasets and remove NSP and trained it for a very large batch but small step size.in this paper, we are also replicating the BERT and explaining the effect of every hyperparameter on BERT because these changes in BERT can be applied to every model in a literature review which consists of BERT architecture. It achieved (88.9%) on GLUE, (88.9/94.6) on SQuADv1.1, and (86.5/89.4) on SQuADv2.0.

Houlsby et al. [22] used adapter modules which only used 3% parameters of BERT in the downstream task and add very few additional parameters because it remembers its previous values which makes it a very compact and extensible model. It achieved 80% on GLUE tasks.

Yu et al. [26] presented a novel model that consists of transformer architecture and had the same settings as BERT but was trained on a specific medical domain. It was the first domain-specific BERT model and it was pre-trained on PubMed and PMC. It achieved better results when compared to BERT (0.62% FI) score on biomedical named entity recognition,(2.80% FI) on biomedical relation extraction, and (12.24%MMR) improvement on biomedical question answering.

Wei et al. [31] Proposed a new pre-trained model trained on large Chinese corpus with functional relative positional encoding whole word masking strategy, LAMB optimizer mixed-precision training, and length of the training sequence. It pre-trained on $\text{BERT}_{\text{Large}}$ model with a batch size of 5K for 25K steps with a learning rate of 1.84 and maximum input size of 128. NEZHA used 10.8B tokens in pre-training. It achieved 88.7% on GLUE.

Radford et al. [43] used a specific self-attention mask and shared transformer network to trained on sequential, unidirectional, and multi-directional tasks and then finetuned on language understanding and generating tasks. It pre-trained with $\text{BERT}_{\text{Base}}$ settings with no NSP,7680 batch size for 0.5M steps,6e4 learning rate with 128 maximum input size. It achieved 87.3% on GLUE, (87.1/93.1) on SQuADv1.1, and (83.3/86.1) on SQuADv2.0.

Wang et al. [57] pre-trained the model with a stacking algorithm. It transfers knowledge from shallow to deep model while observing self-attention on different layers and different positions which allow it to find local attention and start of sentence distribution. StructBERT pre-trained $\text{BERT}_{\text{Large}}$ the model with 32 batch size for 1M steps and 512 maximum input size. It achieved 86.7% on GLUE and (87/93) on SQuADv1.1.

Jiao et al. [58] used the transformer distillation method where linguistic knowledge is transferring to student from teacher BERT.it is a two stage learning framework which allow studentBERT to capture general and specific knowledge from teacherBERT by using 28% fewer parameters than teacherBERT. It used only 14.5M parameters with 256 batch size for 1M steps with 128 maximum input size. TinyBERT used 3L,312H and 12A model which achieved (76.5) GLUE,(79.7/87.5) SQuADv1.1 ,and (69.9/73.4) on SQuADv2.0.

Liu et al. [59]used cross-layer sharing and it learns from multiple tasks which allows it to learn general representation due to which adapts to new domains and downstream tasks. It benefited from a large amount of cross-task data and also general representations that adapt new tasks and domains. It used $\text{BERT}_{\text{Large}}$ setting with no NSP and 128 maximum input size. It achieved 86.4% on GLUE task.

Jiang et al. [60] replicate the RoBERTo by implementing Smoothness inducing regularization and Bregman proximal point optimization on downstream tasks which removes the overfitting of the model during pre-training which usually results in memory loss. It was built upon ROBERTo setting with 10^3 learning rate with no NSP and it achieved 88.5% on GLUE. Like

ROBERTo it was also trained on 160GB of data and trained for 125K steps with 2K batch size and 512 maximum input size. It achieved 88.5% on GLUE.

Su et al. [61] extract global learning between layers by squeeze and excitation methods.it captures the neighbor context in fine-0tuning by using Guasing Blurring and used this model on HANS dataset where it adopted shallow heuristic instead of a generalization. It was pre-trained on BERTBase settings with 32 batch size,2e^5 learning rate, and 128 maximum input size. It achieved 81.3% on GLUE task.

Clark et al. [62]masked the input with a plausible alternative during pre-training using a small genitor network which let it predict whether the every masked input is replaced by a generator sample or not which makes it four times faster than BERT.Electra was trained on 126GB of data with no NSP. It used BERT$_{\text{Large}}$ setting with 2K batch size for 1.75M steps with 512 maximum input size and 2e^4 learning rate. It achieved (89.5) on GLUE,(89.7/94.7) on SQuADv1.1, and (88.0/90.7) on SQuADv2.0.

Wang et al. [63] presented a small and compressed model that used deep self-attention knowledge.it followed the distillation method in which students trained on mimicking the self-attention model which is used in the last layers of the larger Teacher model. It used BERT$_{\text{Base}}$ model with no NSP,1K batch size for 400k steps, a learning rate of 5e^4 for 33M parameters. It also allowed a maximum of 512 input size. It achieved an 81.7 FI score on SQuADv2.0.

Xu et al. [64] presented a progressive model that used compress parameters by dividing the BERT and build its compact substitute by increasing the replacing possibility during pre-training. It allows for deeper interaction between original and compact models. It used the same BERT$_{\text{Base}}$ model settings and achieved 81.2% on the development set of GLUE and 78.5% on the test set of GLUE server.

Goyal et al. [65] Presented a novel method to improve inference time of BERT with very little loss in performance. It eliminates the redundant vectors and benefited by redundancy pretraining to word-vectors. It builds a strategy to keep some word-vectors and discard some which are based upon attention mechanism and learn how many vectors are going to eliminate during pre-training. It used BERT$_{\text{Base}}$ model with 108M parameters,128 maximum input size, and no NSP. It achieved 79.1% on GLUE and 66% on the RACE task.

Clark et al. [66] proposed a teacher annealing method that allows the multi-task model to learn from the teacher and surpass it. It used BERT$_{\text{Large}}$ model with no NSP,128 batch size for 1M steps,128 maximum input size, and cross-layer parameter sharing.it achieved 82.3% on GLUE task.

Liu et al. [67] student model trained on an ensemble of teacher models by distilling knowledge.it applied the distill knowledge method on multi-task learning settings. For each task, a different trained a different teacher model outperformed the single model, and hen a student model is trained on this teacher model via multi-task learning to distill knowledge. It used $\text{BERT}_{\text{Large}}$ model with no NSP and achieved 86.4% on GLUE.

Chen et al. [68] used a differentiable neural architecture search to automatically convert BERT into task-specific models.it uses KD loss efficiency loss for search hint and searches constraints respectively.it added more context-to-query and query-to-context to BERT architecture by use of modified transformer encoder units. It used $\text{BERT}_{\text{Larege}}$ model with no NSP, 128 batch size for 50K steps, 512 maximum input size,3e5 learning rate, and only 9.5M parameters. It achieved 80.1% on GLUE.

Bao et al. [69] proposed a pseudo-masked language training procedure. It is autoencoding and partially auto-Regressive. It follows BERT for encoding modeling. AE provides global, asking PAR and PAR to learn inter-relation between masked spans. PMLN learns long-distance context better than the BERT. UNILM pre-trained $\text{BERT}_{\text{Large}}$ with 330 batch size for 777K steps and achieved 40.51 on ROUGE-L CNN/Daily Mail abstract summarization and 35.75 on Gigaword abstract summarization. It achieved 82.4 on CoQA,(80.5/83.4) on SQuADv2.0,22.12 on BLUE-4,2.67 on NIST-4 ,and 82.4 on GLUE.

Lewis et al. [70] presented a denoising autoencoder to pre-train sequence-to-sequence models which trained by corrupting the text with arbitrary noisy function and then reconstruct the original text. It proposes a novel in-filling scheme.it is best to perform for generalization. It differs from BERT as additional cross-attention by decoders' layers perform on the last hidden layer of the encoder. The BART contains 12L, 1024H, 12A,8K batch size for 500K steps. It achieved 88.4% on GLUE, (88.8/94.6) on SQuADv1.1, and (86.1/89.2) on SQuADv2.0.

Chang et al. [71]Proposed X-BERT which finetune on extreme multi-label text classification X-BERT uses semantic label clusters to better model dependencies. It uses both label and input text to build label representation. It consists of semantic label indexing and ensemble ranking component. It achieved SOTA results on XMC benchmark.77.28% on Wiki,68.70% on Parabel(linear),76.95% on AttentionXML(neural), and 10.7% improvement on product2query.

Shoeybi et al. [72] proposed a Megatron-lm pre-trained model which trained on Billions of parameters by using efficient intra0layer model parallelism .attention in the placement of layer normalization in BERT-style model increases the performance. It used a very big model with 48L,2560H,40A,1024 batch size for 1M steps,3.9B parameters,160GB pre-training dataset, and SOP. It achieved (95.5/90.0) on SQuADv1.1,(91.2/88.5) on SQuADv2.0, and 89.5% on Race.

Chadha et al. [73]BERTQA use a set of modified transformer encoder units to add more focused query-to-context (Q2C) and context-to-query (C2Q) attention to BERT architecture. It also adds localized information to self-attention and skips connections in BERT. It pre-trained the model on $\text{BERT}_{\text{Large}}$ settings with 6 batch size for 1M steps,1.5e^{5} learning rate with 512 maximum input size, and no NSP. It achieved (83.42/80.53) on SQuADv2.0.

Kao et al. [74] boost the BERT by duplicating some layer which makes BERT more deep without extra-training which increase the performance of BERT model in downstream tasks. It built upon $\text{BERT}_{\text{Large}}$ model and achieved 91.84% on SNLI,93.62 on SST-2, and (81.20/84.44) on SQuADv2.0.

Minh et al. [75] presented a method for fine-grained information extraction. It exploited the BERT to handle real scenario data and make BERT learn hidden representations for classification using Convolutional Neural Network(CNN). It was built upon $\text{BERT}_{\text{Base}}$ model with no NSP. It achieved 78.4% on GLUE.

Bin et al. [76] presented a model that fuses the information across layers which helps to better sentence prediction. It replaced the BERT based word models with new sentence embedding using geometric analysis of space spanned. It used a model with 6L, 768H, 12A, 32 batch size for 1M steps,2-105 learning rate with 256 maximum input size, only used 66M, and no NSP. It achieved 81.2% on GLUE.

Mehrad et al. [77] presented a model that combines the structural power of BERT and Tensor-Product Representation. It able to learn the shared structure between NLP datasets but BERT failed on it. It used BERTBase settings with a 5-10 learning rate and no NSP. It achieved 88.1% on GLUE.

Table 3.1: Overall Data

Paper	Name	Training Data Size	Tokens	Training Dataset Name	Model Type	Sentence Learning	Cross-layer Parameter Sharing
[18]	ROBERTa (Large)	160GB	- 2.2T	BOOKCORPUS+WIKIPEDIA+CC-NEWS+OPEN	Auto Encoding	None	False

				WEBTEXT+STORIES			
[11]	SpanBERT	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	False
[56]	SemBERT	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	False
[15]	ERNIE,	9GB+	4.5B +140M	English Wikipedia + wikidata	Auto Encoding	NSP	False
[79]	ERNIE2.0	13GB +	8B	Encyclopedia+ Book Corpus+ Dialog+ Discourse Relation Data	Auto Encoding	None	True
[29]	BERT(base)	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	NSP	False
[80]	XLNet	126GB	32.89B	Bookscorpus+ Wikipedia+Giga5+ Clue Web 2012-B+ Common Crawl	Autoencoding+ Auto regressive	None	True
[81]	UNILM	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	NSP	True
[22]		13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	True
[82]	STRUCT BERT	16GB	2.5B +	English Wikipedia + Book Corpus	Auto Encoding	NSP	
[58]	TINYBERT	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	NSP	False
[59]	MT-DNN	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	NSP	True

[53]	AlBERT(XX-large)	16GB		Books corpus + Wikipedia	Auto Encoding	Sop	True
[72]	Megatron-LM	174 GB		Wikipedia +CC-Stories+ Real News+ OpenWebtext	Auto Encoding	SOP	True
[74]	AlBERT(xxlarge-ensemble)	16GB		Books corpus + Wikipedia	Auto Encoding	None	True
[55]	T5	29TB	34B	Colossal Clean Crawled Corpus	Auto Encoding	None	True
[60]	SMART$_{\text{ROBERTa}}$	160GB		BOOKCORPUS+WIKIPEDIA+CC-NEWS+OPENWEBTEXT+STORIES	Auto Encoding	None	False
[54]	FreelbROBERTa	160GB	-2.2T	BOOKCORPUS+WIKIPEDIA+CC-NEWS+OPENWEBTEXT+STORIES	Auto Encoding	None	False
[61]	SESAME BERT	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	False
[83]	Electra1.75M	126GB	33B	Bookscorpus+ Wikipedia+Giga5+ Clue Web 2012-B+ Common Crawl	Auto Encoding	None	True

[63]	MINILM^a	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	False
[64]	SBERT-WK	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	False
[65]	PowerBERT	11B	3.4B	Books corpus + Wikipedia	Auto Encoding	None	False
[66]	Bam	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	True
[78]	StackBERT	11B	3.4B	Books corpus + Wikipedia	Auto Encoding	None	False
[67]	MT-DNN$_{kd}$	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	True
[77]	HUBERT	16GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	True
[68]	AdaBERT	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	True
[73]	BERTQA	13GB	3.8B	Books corpus + Wikipedia	Auto Encoding	None	false
[70]	Bart(large)	160GB	2.2T	160GB+Wikipedia	Auto regressive	None	False
[31]	Nezha		10.5B	Chinese Wikipedia+ Baidu Baike+ Chinese News	Autoencoding+ Auto regressive	NSP	True
[69]	UNILMv2	160GB		BOOKCORPUS+WIKIPEDIA+CC-NEWS+OPENWEBTEXT+STORIES	Autoencoding and Partially Auto-regressive	None	False

Table 3.2: Overall Hyperparameters

Paper	Batch Size	Max Sequence	Learning Rate	Step Size	Parameters	Layers	Hidden	Attention Head
[18]	2K	512	1e^{-6}	125K	360M	24	1024	16
[11]	256	128	1e^{-4}	2.4M	340M	24	1024	16
[56]	32	128	2e^{-5}	1M	340M	24	1024	16
[15]	512	256	5e^{-5}	1M	114M	6	768	12
[79]	400K	256	5e^{-5}	4K	114M	24	1024	16
[29]	256	128	1e^{-4}	1M	110M	12	768	12
[80]	2048	512	1e^{-5}	500K	**340M**	24	1024	16
[81]	330	512	3e^{-5}	777K	340M	24	1024	16
[22]	32	512	1e^{-4},	1M	330M	24	1024	16
[82]	32	512	1e^{-4},	1M	340M	24	1024	16
[58]	256	128	1	1M	14.5M	4	312	12
[59]	256	128	1e^{-4}	1M	340M	24	1024	16
[53]	4096	512	0.00176	125K	233M	12	4096	128
[72]	1024	128	1.0e^{-4}	1M	3.9B	48	2560	40
[74]	4096	512	0.00176	125K	233M	12	4096	64
[55]	2048	128	0.01	2.1M	11B	12	768	12
[60]	2K	512	10$^{□3}$	125 K	356M	24	1024	16
[54]	8K	512	1e^{-6}	500K	360M	24	1024	16
[61]	32	128	2e^{-5}	1M	340M	12	768	12
[83]	2048	512	2e^{-4}	1.75M	335M	24	1024	16
[63]	1024	512	5e^{-4}	400k	33M	12	768	12
[64]	32	256	2 _ 10^{-5}	1M	66M	6	768	12
[65]	256	128	1e^{-4}	1M	108M	12	768	12
[66]	128	128	1e^{-4}	1M	340M	24	1024	16
[78]	256	128	1e^{-4}	1M	110M	12	768	12
[67]	256	128	1e^{-4}	1M	340M	24	1024	16
[77]	256	128	5 _ 10^{-5}	1M	110M	12	768	12

[68]	128	512	$3e^{-4}$	50K	9.5M	24	1024	16
[73]	6	512	$1.5e^{-5}$	1M	340M	24	1024	16
[70]	8000	512	$1e^{-6}$	500K	400M	12	1024	12
[31]	5120	128	$1.8e^{-4}$	25K	340M	24	1024	16
[69]	7680	128	$6e^{-4}$	0.5M	110M	12	768	12

Table 3.3: Results

Paper	Name	GLUE	SQuAD1.1(EM/F1)	SQuAD2.0 (EM/F1)	Race
[18]	ROBERTa(Large)	88.9	88.9/94.6	86.5/89.4	83.2
[11]	SpanBERT	82.8	88.8 /94.6	85.7 /88.7	---
[56]	SemBERT	82.9		82.4 /85.2	
[15]	ERNIE,	79.1			
[79]	ERNIE2.0	89.6			
[29]	BERT(base)	79.6	80.8 /88.5	79.0/81.8	66.9
[80]	XLNet	87.4	89.0/94.5	86.1/88.8	81.7
[81]	UNILM	82.4		80.5/ 83.4	
[22]		80	/90:4	-	
[82]	STRUCTBERT	86.7	87.0 /93.0		
[58]	TINYBERT	76.5	79.7 /87.5	69.9 /73.4	
[59]	MT-DNN	85.4			
[53]	AlBERT(XX-large)	89.4	94.1/88.3	88.1/85.1	82.3
[72]	Megatron-LM		95.5 / 90.0	91.2 / 88.5	89.5
[74]	AlBERT(xxlarge-ensemble)			86.85 /89.99	
[55]	T5	89.9	90.06 /95.64		
[60]	SMART$_{ROBERTa}$	88.5			

[54]	FreelbROBERTa	89.9			85.70
[61]	SESAMEBERT	81.3			
[83]	Electra1.75M	89.5	89.7 /94.9	88.0 /90.6	
[63]	MINILM^a	83.1		81.7	
[64]	SBERT-WK	81.2			
[65]	PowerBERT	79.1			66.0
[66]	Bam	82.3			
[78]	StackBERT	78.4			
[67]	MT-DNN$_{kd}$	86.4			
[77]	HUBERT	88.1(5)			
[68]	AdaBERT	80.1			
[73]	BERTQA			83.42 /80.53	
[70]	Bart(large)	88.4	88.8/94.6	86.1/89.2	-
[31]	Nezha	88.7			
[69]	UNILMv2	87.3	87.1/93.1	83.3/86.1	-

3.4 Literature Summary

BERT[29] is the first deep bidirectional models because ELMO concatenated representation from both sides of the input where BERT used MLM which stop BERT to see values of othersides and save it from producing multilayer contyext. There are many language models which built upon transformer architecture and improve the BERT results. Some of these models were trained on very large datasets likeROBERTa (160GB)[18] ,XLNet(126GB)[46],T5(29TB)[55], and other big model can be seen in Table 3.1 and results in Table 3.3, these models changes in BERT but because these models were trained on big and different datasets that is why the result of these model can not compare directly with BERT which was trained on only 13GB text data. Every model trained smaller dataset than BERT achieved bad results ERNIE [15], PowerBERT [65], StackBERT [78]. ERNIE2[16] which trained on different datasets but of same size produces much better then BERT but as ERNIE2 was pre-trained on other domain specific datasets which adds some advantages to ERNIE2 over BERT.Some models use BERT setting and same dataset but different methods during pretraining and improve results SpanBERT[11],SemBERT[12], and other can be seen in Table 3.1. These models change the style of pretraining but wholly depends upon BERT settings that make these models an extension of BERT and if we have a better settings for BERT then the results of these dependent models will also incease. If a model is trained on smaller dataset with same settings of BERT it will achieve bad results and if a dataset pretrained on large and domain specific dataset it will achieve better result. A model built upon BERT but with different pre-training procedure will improve results but if the BERT improve then all these models will also improve. Many models remove NSP which is very important component of BERT architectureathese models can be seen in Table 3.1. Many models that used BERT were either trained on a bigger and diverse dataset and remove NSP. Some used BERT to pre-trained on domain-specific datasets. Paper used bigger batch and small step size produced better results but the problem is none of any paper which built upon transformer architecture and used BERT model analyze the effect of different parameters on BERT because these paper either used BERT as it is or trained it on bigger datasets and changed one or two paramters like XLNet, ROBERTa and AlBERT.

Chapter 4

Methodology

CHAPTER 4: METHODOLOGY

In this section, we will explain how we built a new model and how we compared it with BERT.

4.1 Dataset

Original BERT pre-trained on Wikipedia and Books corpus dataset. Books corpus dataset is publicly unavailable so we pre-trained the BERT from scratch on only Wikipedia. Due to pre-training on only Wikipedia results of BERT decreases on many tasks and to compare with MBERT we pre-trained MBERT on only Wikipedia.

4.2 BERT

We pre-trained the original BERT from scratch with two changes. First, we pre-trained it on only Wikipedia(2100M) and used a Sentence piece instead of a Word piece tokenizer section 5.1.1 will explain it in detail. Section 5.1 explains all the changes and results of BERT model which is trained from scratch.

4.3 MBERT

MBERT is our purposed model which pre-trained on Wikipedia (529M). Section 5.2 explains all the changes we made in this model and compared it with BERT model we pre-trained from scratch. Figure 4.1 shows the methodology.

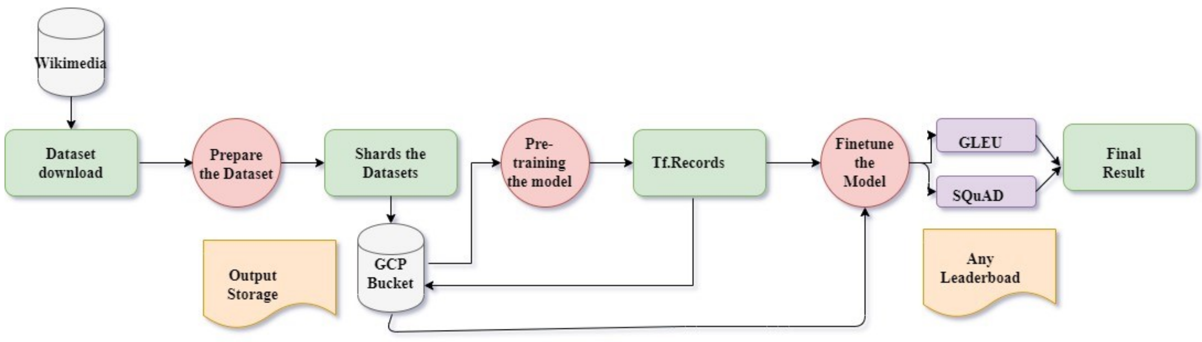

Figure 4.1: MBERT

4.4 BERT

In this section, we will provide a brief overview of BERT pre-training and Fine-tuning and our choices we experiment in BERT

4.4.1 Setup

BERT takes an input of two concatenated segments x_i…. xN and y_i….y_M. These segments are represented as a single input sequence to BERT with special tokens (CLS),x_i…x_N,(SEP) ,y_i….y_M,(EOS) delimiting them. $M+N<T$ where T is maximum sequence length during training. BERT trained on an unsupervised dataset (Wikipedia+Bookscorpus) and Fine-tune on supervised datasets.

4.4.2 Architecture

BERT uses a transformer model that replaces *the recurrent neural network* (*RNN*) and only relies on attention. BERT uses the same transformer model as (VASWAni) which is why we will not discuss it here. We use transformer architecture with L layers, H hidden layers, and A attention Heads.

4.4.3 Training Objectives

BERT uses Masking Language Model (MLM) and Next Sentence Prediction (NSP) objectives during pre-training.

4.4.3.1 Masking Language Model (MLM)

MLM is a masking strategy in which a random token from the input sequence is being masked and replaced with specially taken. The objective of MLM to predict the id of masked tokens based on its context using cross-entropy. MLM fuse the left-right context which allows the model to train bidirectionally without seeing each other otherwise model predicts the word in a multilayer context. Due to this, we do not need to concatenate the right and left data like ELMO.

BERT uses MLM as 15% of all the tokens are being masked and replaced 80% with (MASK) tokens,10% with randomly selected tokens and 10% remain unchanged. BERT uses cross-entropy to predict these masked tokens. BERT uses random masking and replace ones at the beginning of training and Saved it during pre-training. Training data was duplicated 10 times so that over epochs each training sequence can see with the same mask 4 times during the training which avoids masking of the same tokens every time.

4.4.3.2 Next Sentence Prediction

BERT used NSP because many Question Answering (QA) and Natural Language Inference (NLI) required a relationship between a sentence for accurate results. NSP is a binary classification that predict6 that either both sentences are from the same document or not. For a positive example, both sentences are taken from the same document and for negative examples, both sentences were randomly selected. The percentage for both examples is 50%.

4.4.4 Optimization

As Books corpus dataset is publicly unavailable so we pre-trained the BERT from scratch with following settings:B1=0.9,B2=0.999,E=1e-5 ,warmup-steps 10,000 ,weight decay 0.01,attention drop-out 0.1,hidden dropout 0.1,batch size 256 with maximum tokens 128 to step size 1M,evaluation step size 128,Input layers L=12,hidden layers H=768 with feed-forward 4H,Attention head A=12,Maximum position embedding 512,pooler input is 3,pooler size per head 128 and vocabulary of 32000.

4.4.5 Data

BERT (original) was trained on Wikipedia and Books corpus dataset(2500M+800M) 16GB of uncompressed text data but due to unavailability of Books corpus dataset we pre-train the BERT on Wikipedia(2100M) of 13 GB of uncompressed Text data.

4.5 MBERT

We introduced a modified BERT pre-trained model which unlike XLNet and ROBERTa uses a small dataset with comparison to BERT and trained for smaller steps. Both XLNet used 8 times bigger batch size, half step size, big sequence size but the same learning rate. ROBERTa was trained on a 10 times larger dataset with 8 times bigger batch size and same learning rate. MBERT used a model configuration of (L=6, H=1536, A=24), unlike BERT which uses (L=12, H==768, A=12). MBERT is a deeper model (1536) which was trained on 4 times smaller data with smaller sequence size and used 4 times bigger batch size but 20 times smaller step size. MBERT was trained on a larger vocabulary with smaller input and larger attention heads. MBERT also used 2H feedforward instead of 4H used in BERT. Table 2 shows other settings we used to pre-train the MBERT model. We pre-trained the BERT model on T4 TPU of Google Pro with 8 cores.

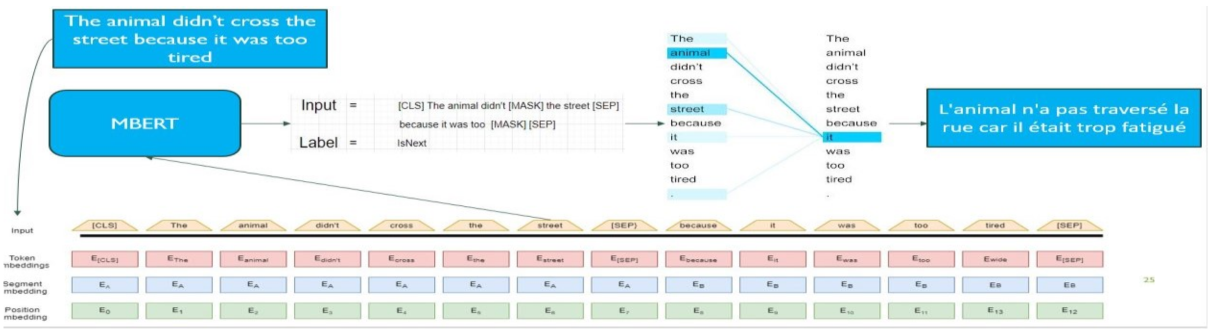
The animal didn't cross the street because it was too tired
MBERT
Input = [CLS] The animal didn't [MASK] the street [SEP] because it was too [MASK] [SEP]
Label = IsNext
L'animal n'a pas traversé la rue car il était trop fatigué
Input
Token Embeddings
Segment Embedding
Position Embedding

Figure 4.2: MBERT Working Example

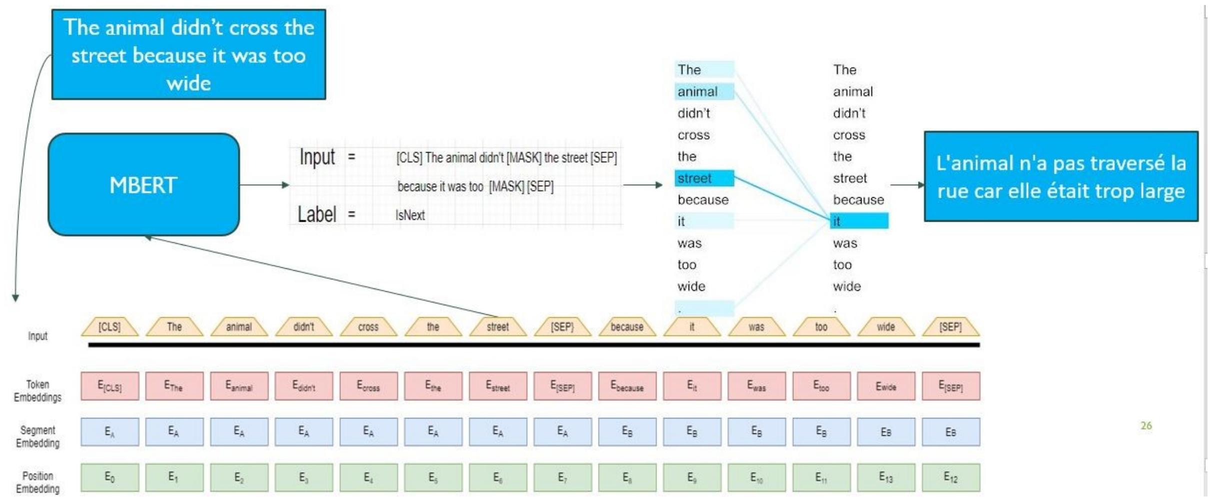
The animal didn't cross the street because it was too wide
MBERT
Input = [CLS] The animal didn't [MASK] the street [SEP] because it was too [MASK] [SEP]
Label = IsNext
L'animal n'a pas traversé la rue car elle était trop large
Input
Token Embeddings
Segment Embedding
Position Embedding

Figure 4.3: MBERT Working Example

4.5.1 Datasets

4.5.1.1 GLUE

GLUE dataset originally consists of nine natural language tasks. Some tasks are single-sentence classification and some are sentence-pair classification. GLUE provides split training and testing data, leaderboard, and server to test the performance of pre-trained models.it allows us to submit our submissions on the GLUE leaderboard and compare our evaluation results on private held-out test data[39].

4.5.1.1.1 QQP

Quora Question Pair is a NLP task which determines that two question on Quora are semantically equivalent are not.it is a binary classification task[84].

4.5.1.1.2 QNLI

Question Natural Language Inference is a binary form of SQuAD[7] task. It contains positive examples and negative examples. Positive examples contained questions and sentence with true answers while negative examples contain questions and sentences which do not exists in paragraph[39].

4.5.1.1.3 MRPC

Microsoft Research Paraphrase Corpus (MRPC)consists of sentence pairs from internet resources with human annotations. The task of MRPC is to check whether pair sentences are semantically equivalent or not[6].

4.5.1.1.4 MNLI

Multi-Genre Language Inference (MNLI) is a crowed source entailment classification task which determines that from two sentences the second is an ailment, contradiction, or neutral to the first one[5].

4.5.1.1.5 SST-2

The Stanford Sentiment Treebank is a binary classification task which consists of sentences extracted from Movie Review with human Annotation[85].

4.5.1.1.6 COLA

Corpus of Linguistic Acceptability (COLA) is a binary classification which checks whether the English sentence is linguistically acceptable or not[86].

4.5.1.1.7 RTE

Recognizing Textual Entailment(RTE) which determines one sentence according to the other just like MNLI[87].

4.5.1.1.8 WNLI

Winograd Natural Language Inference (WINLI) is a small inference dataset that consists upon 2 sentences that differ with only one or two words [88].

4.5.1.2 SQuAD

Stanford Question Answering Dataset (SQuAD) provides a paragraph of content and a question. The task is to answer a question by extracting information from the span. The SQuAD has two versions SQuADv1.1 and SQuADv2.0.

4.5.1.2.1 SQuADv1.1

SQuADv1.1 contains 100k crowdsource Questions/Answers. It contains questions that are answerable from span. The task is to predict the answer span from Wikipedia.

4.5.1.2.2 SQuADv2.0

in SQuADv2.0 the questions are the general type which makes the task difficult because answers to these questions are not provided in paragraphs. It makes the problem more realistic.

Table 4.1: GLUE Sub-task

Dataset	Task Type	Example	Matric	Size		
				Train	Dev	Test
CoLA	Acceptability	They made him angry. 1 = acceptable This building is than that one. 0 =unacceptable	Matthews	10K	1K	1.1K
SST-2	Sentiment	the movie as a whole is cheap junk and an insult to their death-defying efforts = .11111 the movie is funny , smart , visuallyinventive , and most of all , alive . = .93056 has both = .5	Accuracy	67K	872	1.8K
MNLI	NLI	Jon walked back to the town to the smithy.' b'Jon traveled back to his hometown.' = 1 = neutral 'Tourist Information offices can be very helpful.' b'Tourist Information offices are never of any help.' = 2 = contradiction	Accuracy	393K	20K	20K
RTE	NLI	'With its headquarters in Madrid, Spain, WTO is an inter-governmental body entrusted by the United Nations to promote and develop tourism.' b'The WTO headquarters is located in Madrid, Spain.' = 0 = entailed	Accuracy	2.7K	600	3K

MRPC	Paraphrase	"Yesterday , Taiwan reported 35 new infections , bringing the total number of cases to 418 ." "The island reported another 35 probable cases yesterday , taking its total to 418 ."label = 1	Accuracy / F1	4K	1K	1.7K
QQP	Paraphrase	How can I be a good geologist? What should I do to be a great geologist? 1 = similar	Accuracy / F1	400K	6K	391K
QNLI	QA/NLI	"How was the Everton FC's crest redesign received by fans?" b'The redesign was poorly received by supporters, with a poll on an Everton fan site registering a 91% negative response to the crest.' = 0 = Answerable. For not answerable =1	Accuracy	108K	11K	11K
WINLI	Conference / NLI	The trophy didn't fit into the suitcase because it was too [large/small]. Question: What was too [large/small]? Answer: the trophy / the suitcase	Accuracy	706	100	146

Table 4.2: SQuAD Sub-task

Dataset	Example	Matric	Size		
			Train	Dev	Test
SQuADv1.1	Article contains information of super bowl. Where did Super Bowl 50 take place? Answer starts from id=177, text=Denver Broncos	EM/F1	100K	21K	Predictions
SQuADv2.0	Q: What areas did Beyonce compete in when she was growing up? Answer=singing and dancing	EM/F1	150K	15K	Predictions

4.5.2 Performace parameters

Many parameters affect the performance of the pre-trained model. The parameters which we use in MBERT are as follows:

4.5.2.1 Vocabulary Size

Vocabulary size defines the subset of different tokens that can be represented by input. If the input contains words that are out of this vocabulary then the model divides the word into subwords.

4.5.2.2 Sequence Size

If defines the maximum size of an input. If one increases the sequence size then it required a lot of computational power and resources.

4.5.2.3 Batch Size

Batch size is the number of examples that can be utilized in one iteration.

4.5.2.4 Step Size

How many steps a program will run. It takes data points with respect to time.

4.5.2.5 Learning Rate

Learning rate is hyperparameters used in neural networks during the training of the model. The learning rate has weights that update during the training. A large learning rate will take large steps and leans quickly for a large dataset but for small, it is required to be small.

4.5.2.6 Hidden Layers

Number of hidden layers in the transformer. Hidden layers are located in-between between input and out layers on which a function applies weights during execution. The more hidden layers we used the deeper model we get.

4.5.2.7 Input Layers

Number of hidden layers in the transformer. Input layers composed of neuron which brings the data into the system for initial processing and also send it for further processing by hidden layers.

4.5.2.8 Attention Heads

Attention Heads are the number of attention heads on each layer of the transformer model.

4.5.2.9 Intermediate-Size

Intermediate-size also termed as Feed-forward id the dimensionality of transformer encoder. Intermediate-size if of 4H in the standard.

4.5.2.10 Maximum-position-embeddings

It is the size of the maximum sequence which the model can ever have. It should be set large value (e.g. 512,1024)

Chapter 5

Experimental Setup and Results

CHAPTER 5: EXPERIMENTAL SETUP & RESULTS

5.1 EXPERIMENTAL SETUP FOR BERT

In this section, we will present an experimental setup for pre-training of BERT from scratch.

5.1.1 Implementation

While pre-training BERT we follow same procedure as in Delvin BERT. We used Adam 1e-5 learning rate and 128 Maximum sequence length unlike the BERT 90:10 ratio of BERT. A large sequence size consumes so many resources and did not add extra improvement. All other hyperparameters were the same as Delvin and could be seen in Table 5.1.
We use SentencePiece tokens in the unigram model by setting indices to -1 because the Workpiece tokenizer which [29] uses is not open-sourced. The sentence piece adds [BOS] and {EOS} to vocabulary by default. Workpiece prepends sub word occur middle of the word with [##] and if sub word occurs both in beginning and middle then both words are added to the vocabulary. Sentencepiece replaces the Whitespaces "“with "_" and then this is segmented into small pieces. The words after white spaces are prepended with "-“ while others remain changed, due to which words occur in beginning and nowhere else are excluded.to make it compatible with BERT Architecture we just remove "_" from the sub-word and adds "##" to words that do not have "_".We added BERT control symbols which are [PAD],[VNK\,[CLS\,[SEP\.[MASK].
We used a subsample of 25,00,00,000 to generate a vocabulary of 32000 with a num placeholder of 256 because generating vocabulary from the whole dataset would require very high RAM. We pre-trained the BERT model on T4 TPU of Google Pro with 8 cores.

5.1.2 Data

Delvn pre-trained BERT on two datasets Wikipedia and Books corpus. But at the time of writing this paper, the Books corpus dataset is not publicly available. we trained the BERT on only Wikipedia which is of 13GB of uncompressed text data. We download Wikipedia from the official website of Wikimedia and used wiki_extracter to preprocess the dump file.
We did not use the different datasets because we did not want to train BERT on a new dataset which could affect BERT's performance in Both ways, unlike ROBERTa which trained BERT on 5 English datasets of 160GB data we want to remain to original BERT as close as possible.

5.1.3 Evaluation

We used to follow two downstream tasks to evaluate BERT. We used the same parameters as described in BERT paper. Batch size of 32 with a learning rate of 1e-5 for GLUE, batch size of 32 and learning rate of 5e-5 for SQuADv1.1 and 48 batch and learning rate of 5e-5 for SQuADv2.0

5.1.3.1 GLUE

Generalized Language Understanding Evaluation (GLUE) is a collection of 9 tasks. Some of these tasks are single-sentence classification and some are sentence-pair classification. GLUE is used to evaluate Natural Language systems. GLUE provides split training and testing data, leaderboard, and server to test the performance of pre-trained models.it allows us to submit our submissions on the GLUE leaderboard and compare our evaluation results on private held-out test data.

BERT fine-tuned on 8 out of 9 tasks. BERT did not evaluate on STS except this model follows BERT paper fine-tuning settings.

5.1.3.2 SQuAD

Stanford Question Answering Dataset (SQuAD) provides a paragraph of content and a question. The task is to answer a question by extracting information from the span. The SQuAD has two versions SQuADv1.1 and SQuADv2.0. SQuADv1.1 contains 100k crowdsource Questions/Answers. It contains questions which are answerable from span but in SQuADv2.0 the questions are the general type which makes the task difficult because answers to these questions are not provided in paragraph.

Table 5.1: Hyperparameters

Hyperparameter	Values
Vocabulary	32000
Hidden layers	768
Input layers	12
Attention heads	12
Maximum sequence size	128
masking	15%
Batch size	256
Learning rate	1e-5
Intermediate-size	3072(4H)

Pooler size per head	128
Hidden dropout	0.1
Initialize range	0.02
Hidden dropout	0.1
Maximum position encoding	512
Step Size	1000000

5.1.4 Results

Table 5.2: Original vs replicated

Model	CoLA	MRPC	QQP	QNLI	MNLI	WINLI	RTE	SST-2	**Glue**	SQuADv1.1	SQuADv2.0
BERT paper	52.1	88.9	71.2	90.5	84.6	___	66.4	93.5	**78.2**	80.8 /88.5	81.6
BERT	69.1	73.5	88.8	85.2	78.1	43.66	59.9	86.8	**73.1**	66.71/77.10	59.40/63.57

Original BERT did not evaluate on WNLI but we fine-tuned it WNLI. Due to train on only Wikipedia, the results of BERT we pre-trained from scratch has 78.2 average of 7 task while BERT achieved 73.1 results on GLUE dataset. We trained the BERT from scratch so that we can compare our pre-trained model with it. Although this BERT was trained on only Wikipedia it still achieves better results on COLA and QQP. MRPC, SQUADv1.1, and SQuadv2.0 were most affected.

5.2 EXPERIMENTAL SETUP FOR MBERT

In this section, we will present an experimental setup for our pre-trained model MBERT.

5.2.1 Implementation

We reimplement with primarily follow the same procedure as BERT(Delvin). We changed the hyperparameters. We used a subsample of 2500000 with a num placeholder of 256to generate a vocabulary of 52000 as 32000 used in BERT was not enough to produce meaningful tokens.

We also use Sentencepiece to generate tokens. we increase the batch size to 1024 and decrease the step size to 50,000 from 10,00,000(1M).decrease the input layers to L=6 which allows us to use a bigger attention head of A=24.MBERT is the deeper model as we used H=1536 hidden layers and Feedforward 3072.A high learning Rate of 2e-5.128 Maximum sequence size.Table 5.3 shows other hyperparameters. We pre-trained the BERT model on T4 TPU of Google Pro with 8 cores

Table 5.3: MBERT Hyperparameters

Hyperparameter	Values
Vocabulary	52,000
Hidden layers	1536
Input layers	6
Attention heads	24
Maximum sequence size	128
masking	15%
Batch size	1024
Learning rate	2e-5
Intermediate size	3072(2H)
Pooler size per head	128
Hidden dropout	0.1
Initialize range	0.02
Hidden dropout	0.1
Maximum position encoding	512
Step Size	50,000

5.2.2 Data

We trained MBERT on Wikipedia of (530M) of 3GB of uncompressed text data. We used a small dataset because having a large dataset will not improve our model but a domain-specific dataset. But as the book’s corpus dataset is not publicly available and unlike ROBERTa which used a large dataset of 160GB we just trained on Wikipedia with 530M.

5.2.3 Evaluation

We used the same downstream tasks GLUE and SQuAD mentioned in section 3.3.

5.2.4 Results

We evaluate our pre-trained model on GLUE and SQuAD datasets.

5.2.4.1 GLUE

MBERT fine-tuned with task-specific settings that produce better results on COLA (+.12), MRPC (+2), WNLI (+12.67), RTE (+6.53) and on QQP (-1.07), MNLI (-2.76), QNLI (-0.5), SST-2(-2.28). Overall MBERT achieves (+1.84) accuracy on GLUE. Table 5.4 shows the task of specific parameters for MBERT. the task with a larger dataset required a larger batch size compare to small tasks.

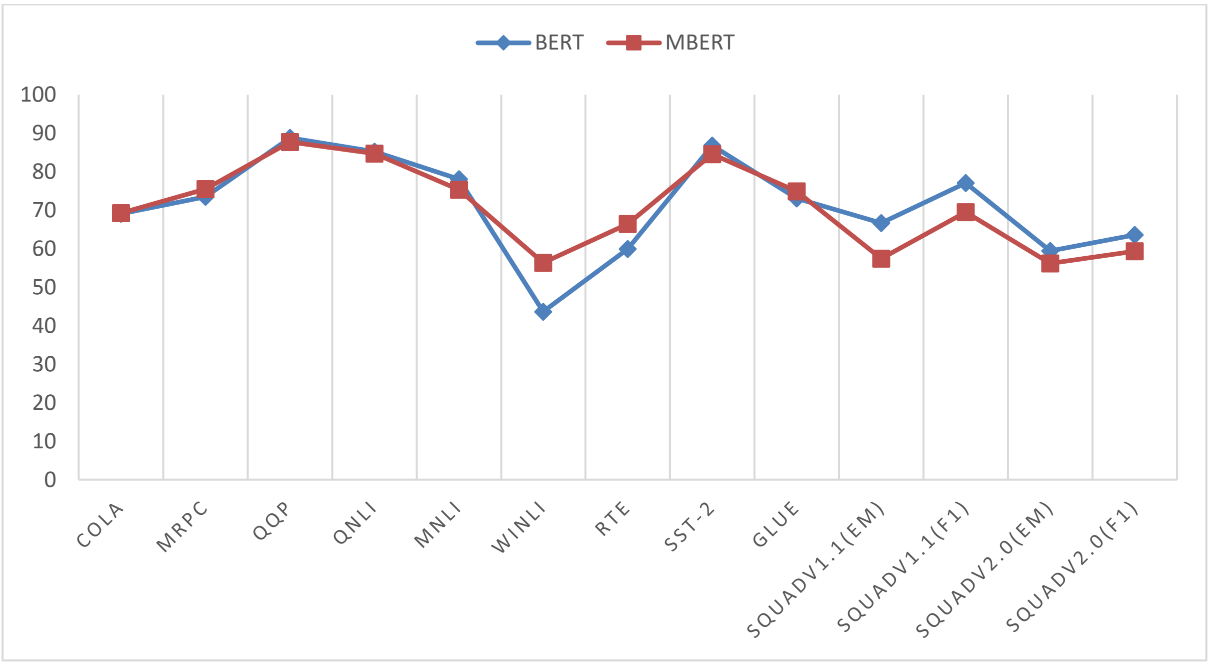

Figure 5.1: Result

Table 5.4: Fine-tuning setting

Task Name	Batch size	Learning Rate
COLA	16	1e-5
MRPC	48	1e-5
QQP	32	2e-5
QNLI	64	2e-5

WNLI	16	2e-5
RTE	24	1e-5
MNLI	64	2e-5
SST-2	32	1e-5
SQuADv1.1	32	5e-5
SQuADv2.o	48	3e-5

5.2.4.2 SQUAD

MBERT has trained on 50k steps which affect its performance on SQuADv1.1 which in section…. we fine-tuned MBERT on SQuAD 32 batch size and 5e-5 learning rate over 3 epochs. Figure 5.1 shows that MBERT produced EM (-9.3) and F1(-7.6) on Squad. BERT fine-tuned on SQuADv2.0 with a batch size of 48 and 3e-5 leaning for 3 epochs produced EM (-3.21) and F1(-4.19).

5.2.5 Result Analysis

In this section, we will analyse the performance of MBERT on GLUE and SQuAD tasks.

5.2.5.1 GLUE

GLUE consists of 9 tasks in which Some of these tasks are single-sentence classification and some are sentence-pair classification. We evaluate MBERT on 8 downstream tasks except for STS-B which behave differently. We even Trained MBERT on the WNLI dataset and produced much better results than BERT.MBERT pre-trained for 50K steps achieves better results on COLA, MRPC, WNLI, RTE, and overall beats BERT.MBERT trained for large step size outperform BERT on COLA, MRPC, WINLI, RTE, and SST with an overall improvement of (+2.49) over BERT.

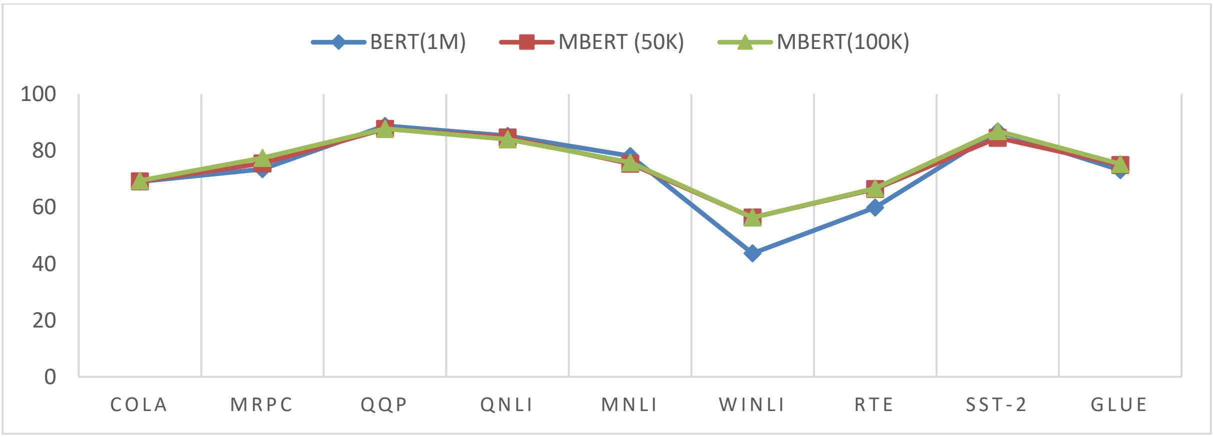

Figure 5.2: GLUE Results

5.2.5.2 SQuAD

On SQuADv1.1 (57.40/69.50) MBERT performs badly but performs relatively better on SQuADv2.0 (56.19/59.38) which is (-9.31/-7.6) and (-3.21/-4.19). the performance of MBERT on SQuADv1.1 is bad but it is relatively better on SQuADv2.0 which can bee seen in Figure 5.3.

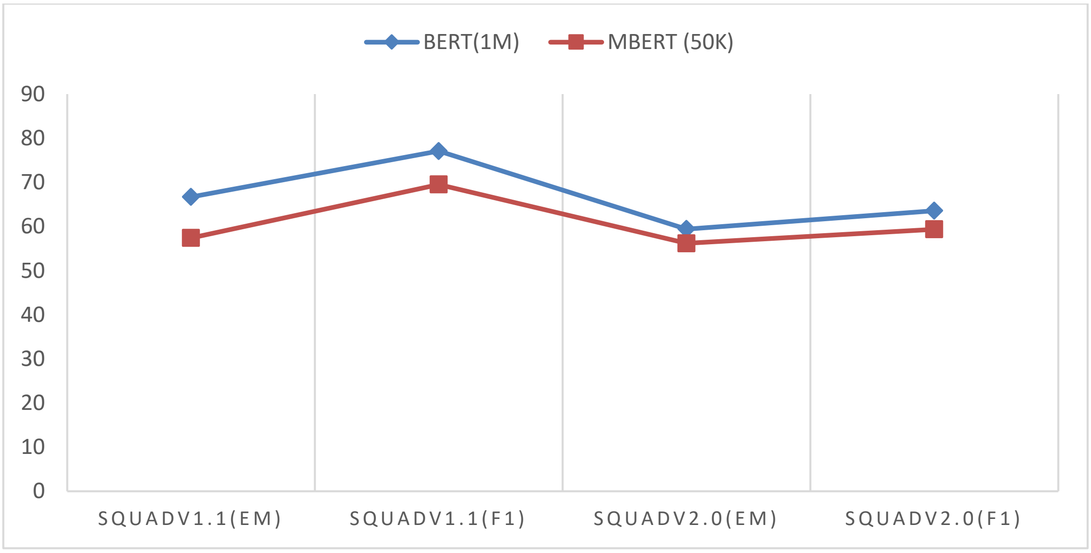

Figure 5.3: SQuAD Results

We proved in section 5.2.5 that SQuAD tasks are very dependent to step size but we also showed that other parameters like feedforward, step size, hidden layers, positional encoding affects the final performance of the pre-trained model on downstream task SQuAD. Due to memory limitations, we can not say the effect of input layers on SQuAD.

5.2.6 Overall Analysis

We pre-trained the BERT from scratch with the same setting as of BERT (paper) with some changes discussed in section #. the training time required for BERT was 53 hours with a computational cost of 256*128*1000000. BERT was trained on a dataset of 13GB of a clean uncompressed dataset which is 2100M. Books corpus dataset is not publicly available that is why we had to train BERT on only Wikipedia. BERT produced 73.1 on GLUE task, EM/F1 66.71/77.10 on SQuADv1.1, and 59.40/63.57 on SQuADv2.0.

We trained a modified BERT model with different parameters shown in table 3 and we called it MBERT.MBERT was trained on a small dataset of 3 GB of uncompressed Wikipedia text data. MBERT trained for a much small step size of 50K with a large batch size of 1024 and hidden layers of 1536 while using feed-forward 3072(2H) due to hbm. MBERT was trained for 17 hours with a computational cost of 1024*128*50000 which is very small then compare to BERT.

MBERT achieved 74.94 on GLUE.MBERT outperforms BERT on 4 tasks while performing very close to the other 4 tasks. On SQuADv1.1 (57.40/69.50) MBERT performs badly but performs relatively better on SQuADv2.0 (56.19/59.38).

In Table 5.5 presented the pre=training setting of diffretn versions of MBERT. In Figure 29 we presented results of Different MBERT models that pre-trained with the same hyperparameters except for one change which can be seen in Figure 5.4. We find out that the GLUE task is very sensitive to learning rate and hidden layers. As we can see in Figure 5.4that 3 tasks perform well below MBERT which are MBERT_7, MBERT_12, and MBERT_13 on GLUE.MBERT_7 was trained with smaller hidden layers while MBERT_12 and MBERT_13 were trained with 1e-5 and 3e-5 learning rates respectively. Other parameters also affect the performance which could be seen in Figure 29.To find out the settings for pre-trained models one can check Table 5.5 in which we presented the settings of parameters.

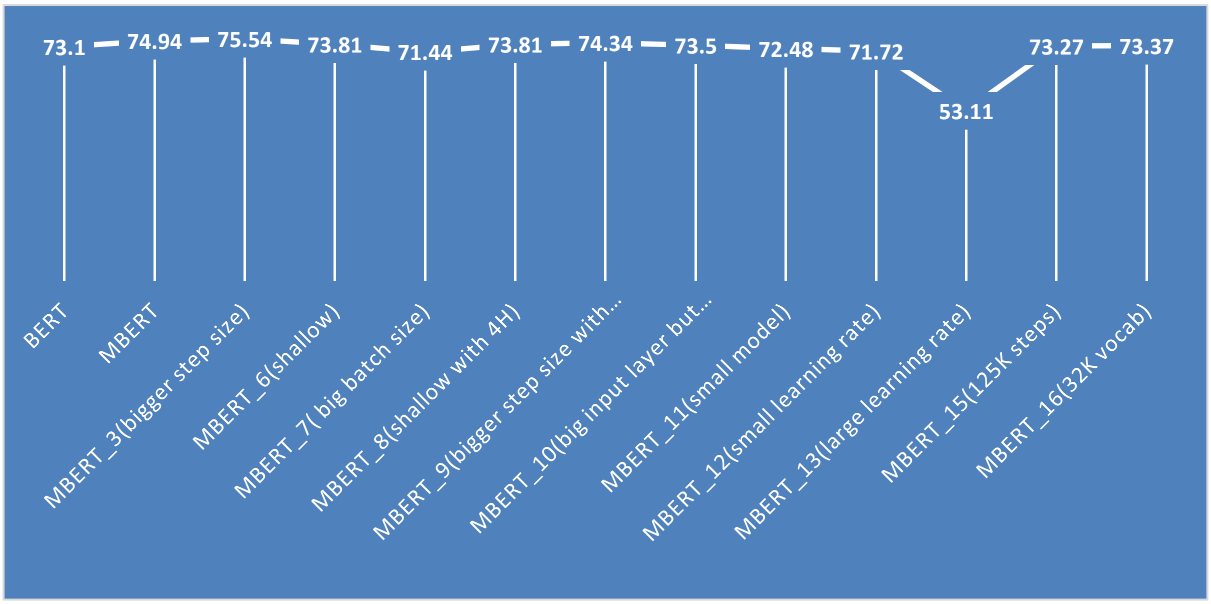

Figure 5.4: GLUE Analysis

Figure 5.5 and Figure 5.6 shows different pre-training settings for MBERT and the effect of these training on SQuAD results. BERT which trained for 1M steps produced the best results.

as we discussed that SQuAD task heavy; it depends upon step size. Other factors affect the MBERT's performance on SQuAD tasks like batch size, input layers, hidden layers, attention heads, maximum position embeddings, and size of feedforward. SQuAD performance is most affected by the Step size. Client and RoBERTo produced better results compare to BERT with smaller steps 500k and 125k respectively but these models were trained on very large and different datasets while MBERT trained on 4 times smaller datasets than BERT.

In Figure 5.5 we can see that MBER_1, MBERT_2, and MBERT_3 which trained with the same setting but for different step sizes produced different results.MBERT_3 which trained for 100K steps perform much better than MBERT_2 and MBERT_1 which trained for 75K and 50K respectively.MBERT_4 which has bigger positional embeddings than MBERT_1 produced better results which are even better for MBERT_2 which trained for 75K. Training with a smaller batch size affects the model as MBERT_5 trained with a smaller batch size and produced bad results than MBERT_1.MBERT_6 used a shallow model means fewer hidden layers with big batch size which affects the results very badly.MBERT_7 trained with large batch size 1344 but smaller positional encoding, feed-forward, and hidden layers and perform worst on SQuAD.MBERT_8 shows that a shallow model even with 4H feed-forward produced bad results.

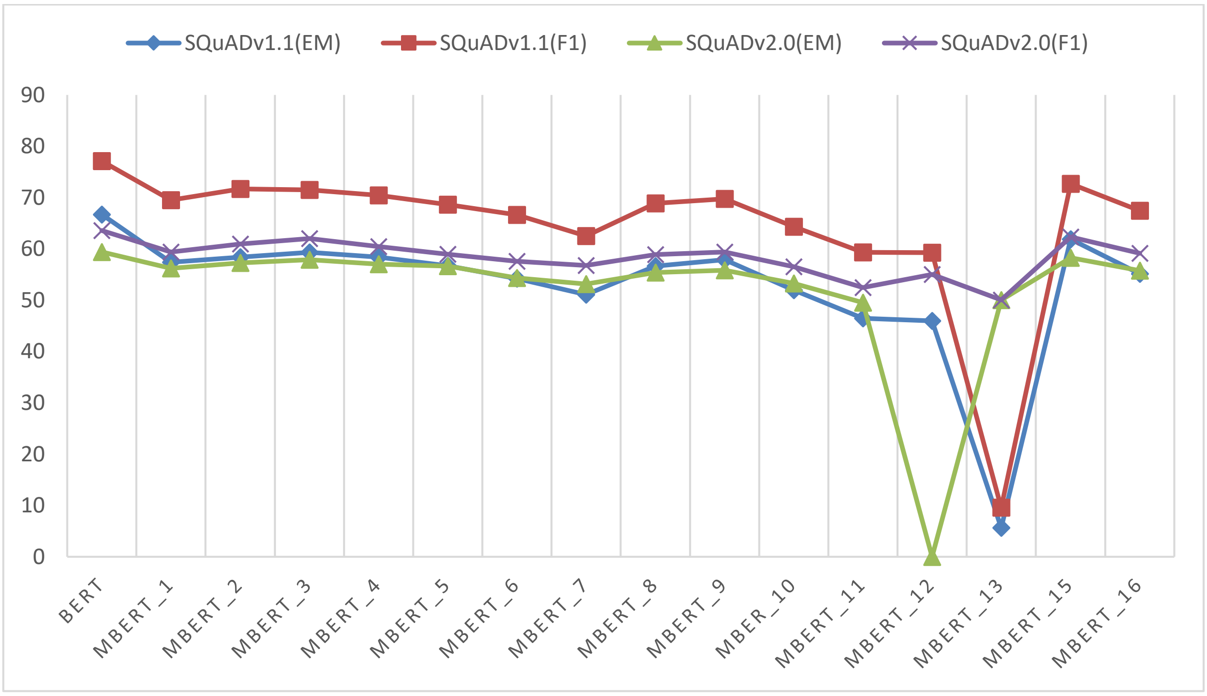

Figure 5.5: Result of different models on SQuAD

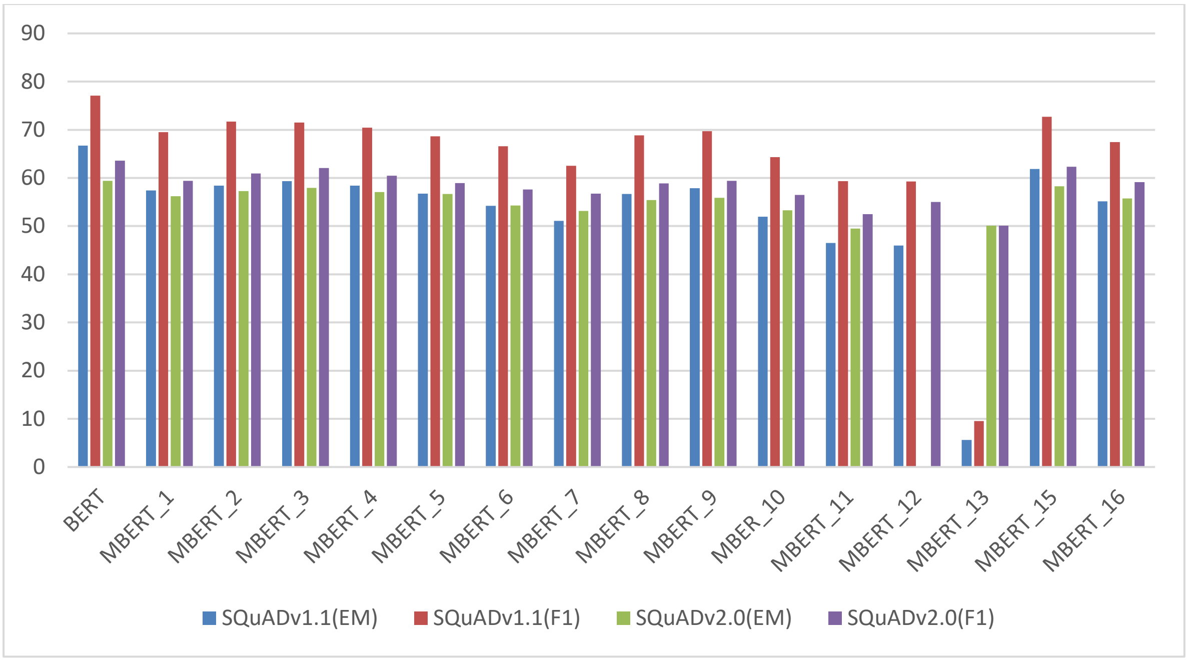

Figure 5.6: Result of different models on SQuAD

MBERT_9 showed that step size important for the SQuAD task as it trained with smaller attention heads but for 75K steps and perform almost identical to MBERT_1 which was trained with 24 attention heads. Figure 5.6 shows that SQuAD depends upon many factors but the step size is most crucial as all models which trained for small step size produced bad results when compared to BERT which was trained for 1M steps although it has a smaller batch size, smaller attention head, smaller positional embedding, and shallow model because it was trained for 1M steps it performs best on SQuAD among all models. Even all models having a large step size than those which has a step size of 50K perform well on SQuAD.

Table 5.5: Different Versions of MBERT

Model	step	Batch size	L	H	A	M.P.E	Feedforward	L.R	Vocabulary
BERT	1M	256	12	768	12	512	3072(4H)	1e-5	32000
MBERT_1	50K	1024	6	1536	24	512	3072(2H)	2e-5	52000
MBERT_2	75K	1024	6	1536	24	512	3072(2H)	2e-5	52000
MBERT_3	100k	1024	6	1536	24	512	3072(2H)	2e-5	52000
MBERT_4	50k	1024	6	1536	24	832	3072(2H)	2e-5	52000
MBERT_5	50k	960	6	1536	24	1024	4608(3H)	2e-5	52000

MBERT_6	50k	1152	6	1032	24	1024	4096(4H)	2e-5	52000
MBERT_7	50k	1344	6	1008	24	512	3072(2H)	2e-5	52000
MBERT_8	50k	1024	6	1320	24	1024	5280(4H)	2e-5	52000
MBERT_9	75k	1024	6	1536	16	1024	3072(2H)	2e-5	52000
MBER_10	50k	1024	12	768	24	512	3072(2H)	2e-5	52000
MBERT_11	50k	1024	6	768	24	512	3072(2H)	2e-5	52000
MBERT_12	50k	1024	6	1536	24	512	3072(2H)	1e-5	52000
MBERT_13	50k	1024	6	1536	24	512	3072(2H)	3e-5	52000
MBERT_15	125k	256	6	1536	24	512	3072(2H)	2e-5	52000
MBERT_16	50k	1024	6	1536	24	512	3072(2H)	2e-5	32000

5.3 TRAINING PROCEDURE ANALYSIS

In this section, we will explore and quantifies which choices are important when pre-training BERT. MBERT has a different configuration of (L=6, H=1536, H12,25M params).

5.3.1 Bigger batch size with a small step size

5.3.1 Batch and Step Size

Figure 5.7 shows the effect of batch size during pretraining and Figure 5.8 shows the effect of step size while keeping all other parameters constant. We can see that when batch size increase the results improve and when step drecrease the results also effects. But when MBERT incrases the batch size but decreases the step size the results improves which shows a relation and effect of batch and step size together.

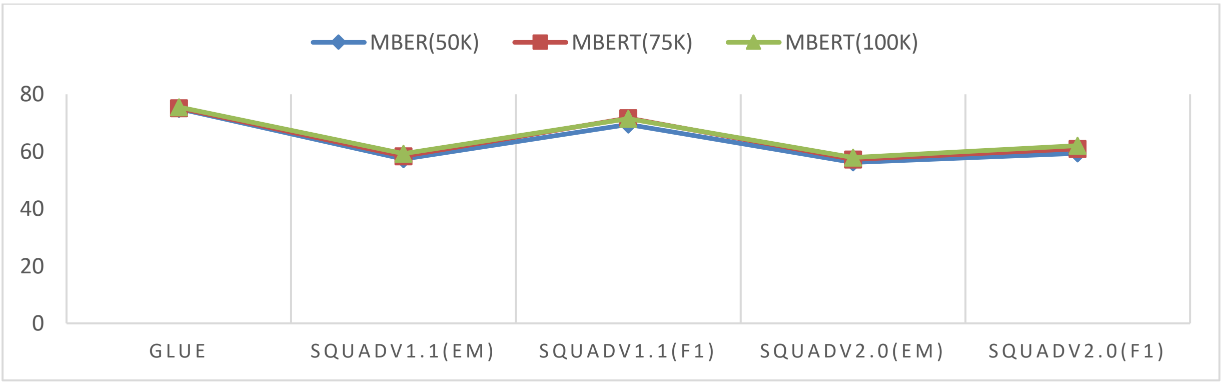

Figure 5.7: Effect of Batch Size

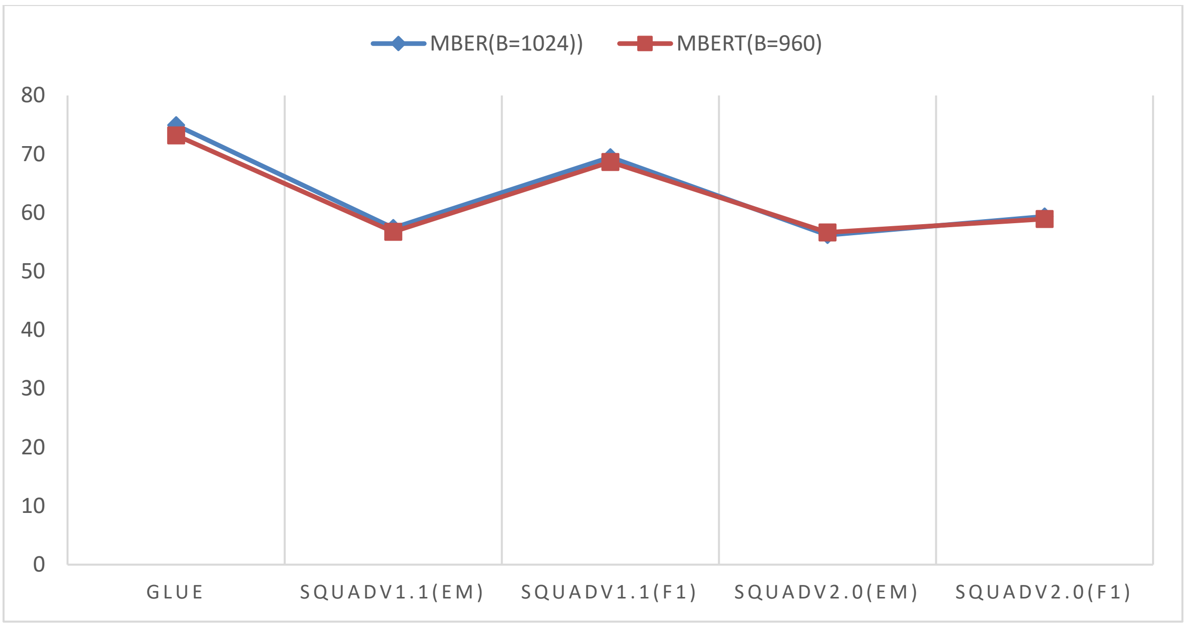

Figure 5.8: Effect of Step Size

5.3.2 Deeper Model:

BERT was trained with 768 hidden layers which are not enough and BERT could be trained with deeper hidden layers.

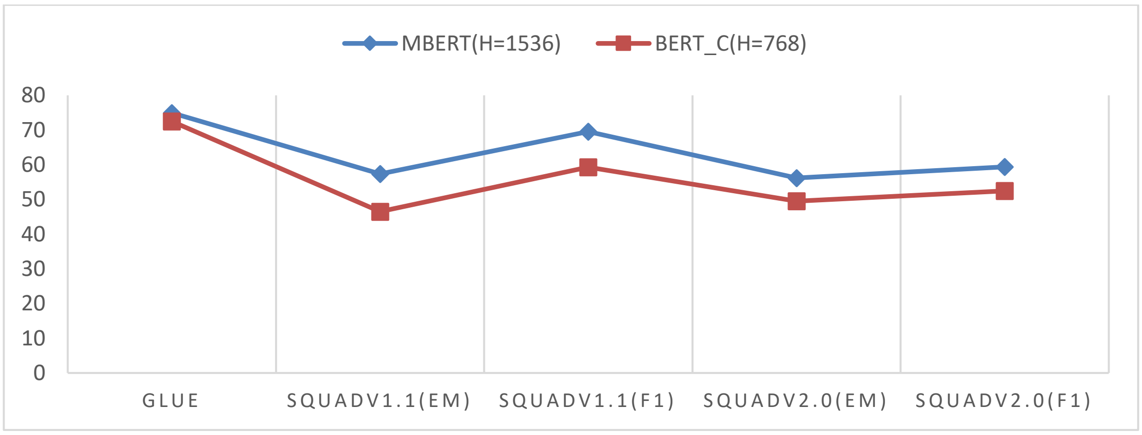

Figure 5.9: Effect of Hidden Layers

Figure 5.9 shows that a deeper model produces better results. MBERT has the same hyperparameters as BERT_C but was trained with deeper hidden layers of 1536 instead of 768 and we can see the improvement of 2.5 in GLUE task and 10.9/10.17 in SQuADv1.1 and 6.67/6.69 in SQuADv2.0 improvement with an increase in hidden layers.

5.3.3 Small Input Layers

Input layers and hidden layers are inverses which means if we increase one then we need to decrease the other due to limited High Bandwidths Memory (hbm). Having both bigger input and deeper hidden layers will defiantly improve the performance.

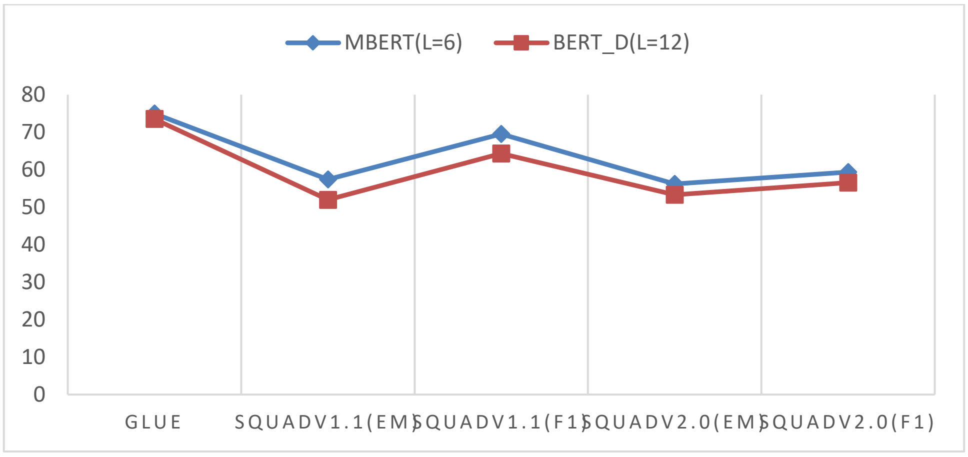

Figure 5.10: Effect of Input layers

Figure 5.10 shows that BERT_D has the same hyperparameters for pretraining as MBERT with one change. BERT_D has L=12 input layers that force it to reduce hidden layers to H=768 and feedforward to 3072(4H) while MBERT has L=6 input layers that allow MBERT to have a deeper model which is H=1536 but small feed-forward 3072(2H).BERT_D which has bigger input layers but shallow hidden layers did not improve any of the tasks of GLUE and SQuAD. Overall, BERT_D decrease the performance to 1.44 on GLUE and 5.53/5.18 on SQuADv1.1, and 2.92/2.88 on SQuADv2.0.

5.3.4 Bigger Attention Head

Change of attention had affected the performance of the pre-trained model and the same happened when the trained model with bigger attention head.

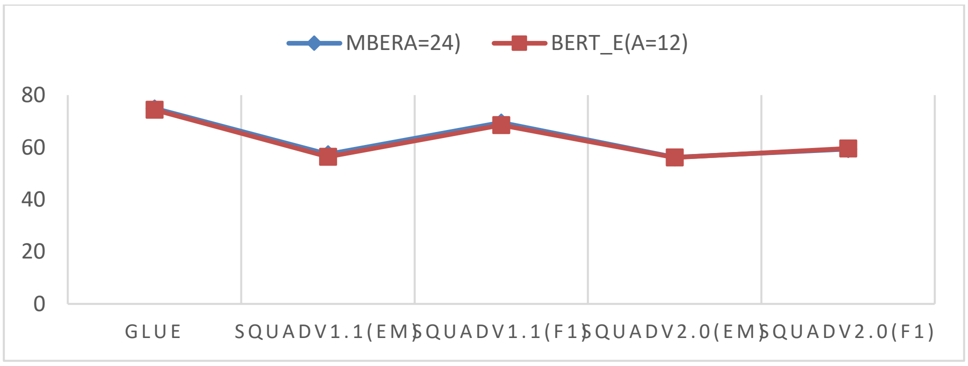

Figure 5.11: Effect of Attention Heads

Figure 5.11 shows that when MBERT trained on bigger attention head then the performance increases as compare to when it trained with 12 attention heads.

5.3.5 Higher Learning Rate

Learning rate has very big effect on the performance of pre-trained model when evaluate on downstream tasks.

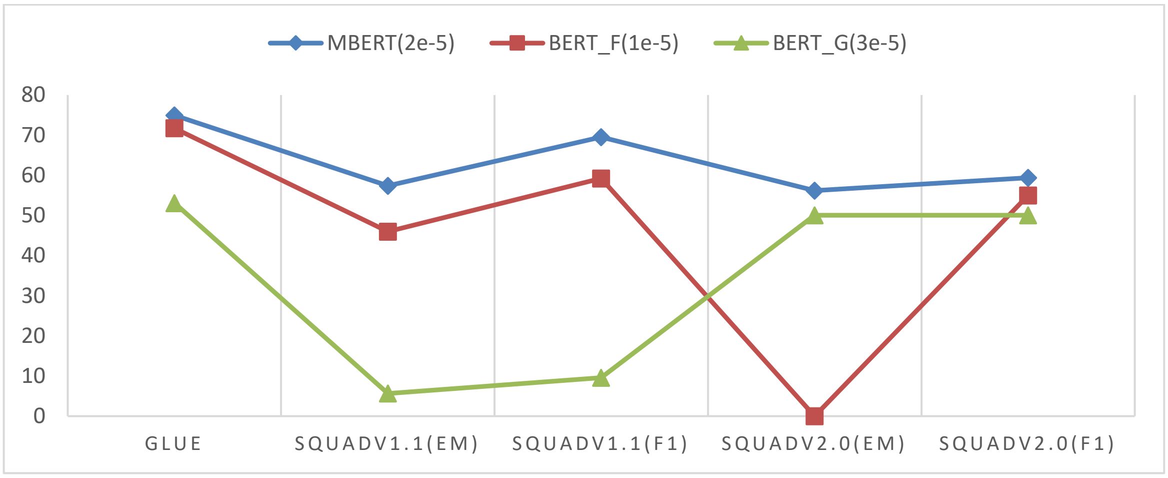

Figure 5.12: Effect of Learning Rate

All models shown in Figure 5.12 has same hyperparameters except one that they are trained with different learning rate. When step size decrease and batch size increases then the learning rate need to be increased.it is very crucial to increase the learning rate with these factors like all of these three models MBERT, BERT_F, and BERT_G has 1024 batch size for 50K steps. BERT_F trained with a smaller learning rate and BERT_G trained with a very high learning rate than MBERT. A small learning rate affects the performance but an extra high learning rate destroys the model's performance. BERT_F which was trained with smaller learning rates 1e-5 then MBERT produces GLUE (-3.22), SQuADv1.1(-11.41/-10.3), and SQuADv2.0(-4.56/4.37) means worst results. BERT_G which is trained with a high learning rate 3e-5 then MBERT produces the worst on GLUE(-21.83), SQuADv1.1(-51.76/-59.93), and SQuADv2.0(-6.13/-9.32).AS we can see that BERT_G which trained with a very high learning rate for 50K step perform well below the average on both GLUE and SQuadv1.1.Setting a learning rate with w.r.t batch and step size is very crucial while pre-training the model as it affects the model on downstream tasks..

5.3.6 Training on Smaller Sequences:

Having a larger sequence size required a lot of resources due to which one need to set other hyperparameters but these large sequences did not improve results much which can be seen Figure 5.13.

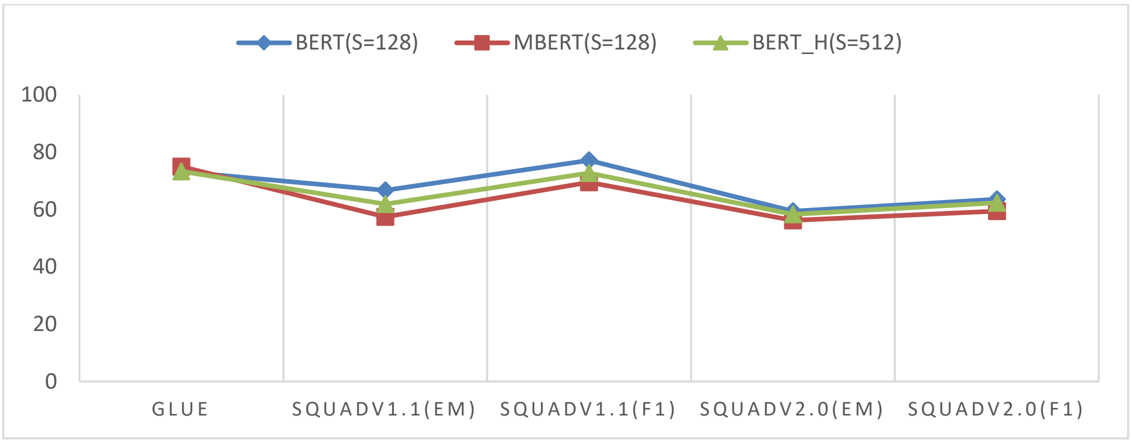

Figure 5.13: Effect of Input Sequence Size

BERT_H was trained on a bigger sequence size of 256 due to which we had to reduce batch size 256 from 1024 but increased the step size to 125K which MBERT trained on 128 sequence with 1024 batch size for 50k steps. BERT_H performs (-1.67) on GLUE, (+4.48/+3.21) on SQuADv1.1 and (+2.09/+2.92) on SQuADv2.0. We chose smaller sequences because we believe that the SQuAD task heavily depends upon steps and here BERT_H was trained for 125K steps. Figure 38 shows that BERT which has 128 sequence size but trained for 1M steps produced the best SQuAD results which show SQuAD depends heavily on step size.

5.3.7 Training with Bigger Vocabulary:

Pre-training of language model with large vocabulary improves the results especially when training data is of English text. Original BERT was trained on 32000 vocabulary which we consider is very small for that large model which trained on Wikipedia and Bookscorpus (2500M+800M) of data.

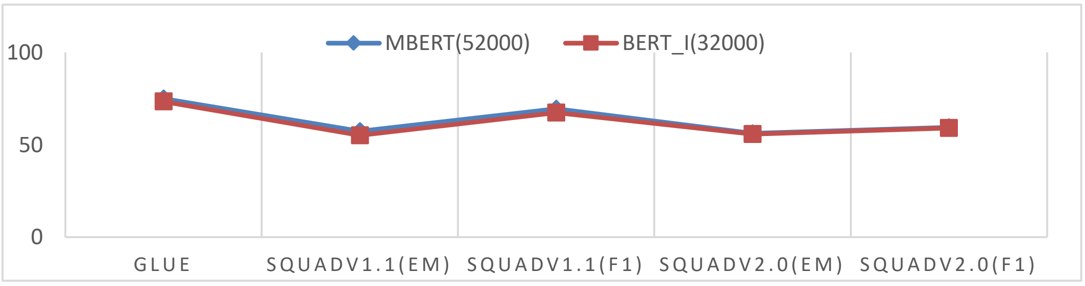

Figure 5.14:Effect of Vocabulary Size

We trained MBERT with 52000 of vocabulary as our dataset is very small compared to both original BERT (Paper) (2500M+800M) and replication of BERT by us (2100M). Figure 5.14 shows that MBERT achieves better results than BERT_I which has the same hyperparameters as MBERT but has a small vocabulary size of 32000.MBERT achieved (+1.57) on GLUE, (+2.28/+2.07) on SQuADv1.1 and (+0.43/+0.25) on SQuADv2.0. MBERT performs well on every single task of GLUE.

5.3.8 Task-Specific Finetuning:

Finetuning of a pre-trained model with the same settings on downstream tasks affects the performance as every task required specific finetuning. The fine-tuning settings of the pre-trained model need to be according to pre-training.

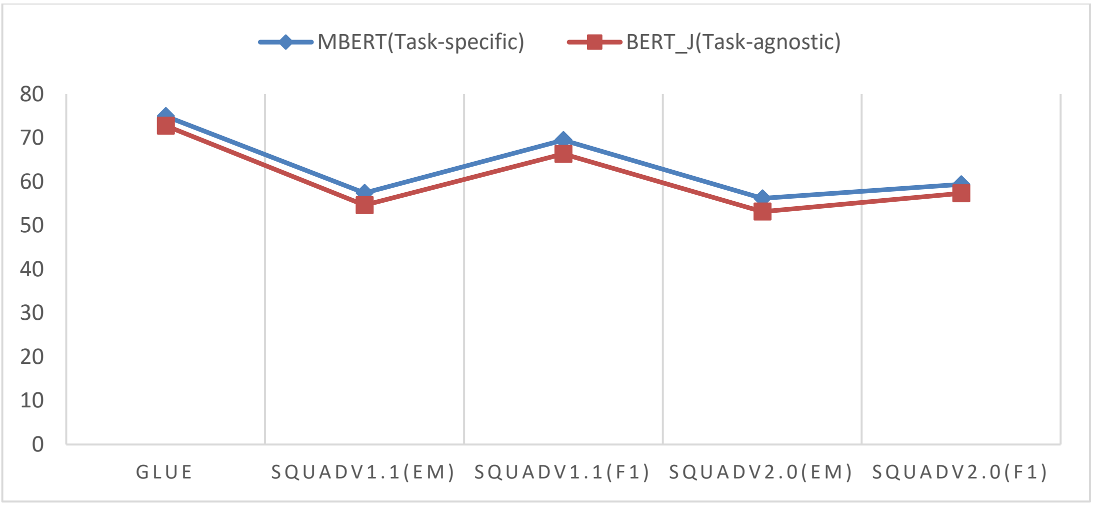

Figure 5.15: Effect of Task-specific Fine-tuning

Figure 5.15 shows that MBERT which is task specific finetuned produced better results than BERT_J which fine-tuned same for all GLUE task.32 batch size with 2e-5 learning rate. 32 batch size with 5e-5 learning rate on SQuADv1.1 and batch size with 5e-5 learning rate on SQuADv2.0. MBERT produces better results on all task. On GLUE MBERT achieves (+2.17), SQuADv1.1 (+3.04/+3.19), SQuADv2.0(+3.04/+2.02).

Chapter 6

Discussion, Conclusion And Future Work

CHAPTER 6: DISCUSSION, CONCLUSION AND FUTURE WORK

Discussion

We pre-trained the BERT(Paper) from scratch because the dataset which BERT used for pretraining is not publicly available. Wikipedia and books corpus were the datasets but books corpus is not publicly available which forces us to pre-trained the BERT from scratch using the same settings as of BERT(Paper) with one basic change and that was the use of sentence piece instead of workpiece we discussed in section#.

We pre-trained the MBERT a modified BERT on a very small dataset of Wikipedia (531M) with compare to (2100M) for BERT in this paper. Training a deeper model with a bigger batch size for smaller steps with a high learning rate produced better results on GLUE tasks while performing badly on SQuADv1.1 but relatively better on SQuADv2.0. Section# shows that the SQuAD task heavily depends upon step size.

High bandwidth memory played an important role while pre-training the MBERT due to which we had to reduce values of some parameters which we believe affects our model very badly. First of all, is the inverse relationship between batch size and size of hidden layers concerning Memory, not performance as when both batch size and hidden size increased then the results improve either parameter effects badly. Training of MBERT 50k steps produced almost identical results when trained for 100K steps which shows that only training for longer steps did not improve the performance of MBERT as much as when a change of other settings. but training for longer steps improve results on the SQuAD task but training for 25K more steps is as same as training for 320 additional positional encodings or feed-forward 3H instead of 2H. In MBERT we limited two parameters (1) positional encoding of 512 instead of 1024 (2) used small feed-forward 2H instead 4H. We showed in section# that if we used the same settings as of 1024 and 4H for MBERT then our results will improve. The learning rate also impacts the performance and need to be set according to the batch and step size.

MBERT performs badly on SQuADv1.1 and except that it performs well for SQuADv2.0 and even beat the BERT on GLUE task. Training time required for BERT for 1M steps was 53 hours of training with the computational cost of 256*128*1000000 while MBERT which trained for only 50K steps 20 times smaller step size and 4 time smaller dataset pre-trained for 17 hours with the computational cost of 1024*128*50000 a much smaller and produced better results on GLUE which consists upon 8 tasks except STS-B which we leave and performed well on SQuADv2.0.

All of these experiments mean training and fine-tuning can be replicated with our setting on Google Collab which is free and provides free TPU. In the whole discussion, we ignore the shard which is used to save the dataset in GCP buckets to pre-train the model. If we consider that factor then BERT which trained on WIKIPEDIA (2100M) took almost 4 times more time than MBERT which rained on WIKIPEDIA (510M).

Conclusion

Due to the unavailability of BOOKS corpus, we were unable to compare our results with BERT (paper) which makes us pre-trained the BERT from scratch on only the Wikipedia dataset. We compare our results with BERT, unlike ROBERTo and XLNET we used a small dataset as BERT. Using the same dataset as for BERT will not improve our model much because with our modified setting use of different and domain-specific data is more important than Wikipedia which is plain text data. We pre-trained BERT on the Whole dataset of WIKIPEDIA because we want to keep BERT as close to BERT (paper) as possible with the same settings except the use of sentence pieces in place of Wordpiece.

In future will train the MBERT deeper with bigger batch size but for the same steps 50K.with limited resource with were unable to trained MBERT with large positional embeddings and bigger feed-forward which we believe need to be 4H instead 2H which we had to use. Training of MBERT on different and especially domain-specific datasets will improve the performance of MBERT. A future direction is to find settings that improve the performance of MBERT on the SQuAD dataset because here MBERT performs badly. Setting choices, we made in MBERT can be used on other models which trained with BERT architecture and we believe their performance will improve like on BioBERT and SCIBERT because they used BERT (paper) architecture but trained on a domain-specific dataset and we discussed that training of MBERT on a domain-specific dataset will improve performance instead of for training for longer steps on the same dataset.

References

1. Taylor, W.L., *"Cloze procedure": A new tool for measuring readability.* Journalism quarterly, 1953. **30**(4): p. 415-433.
2. Mikolov, T., et al. *Distributed representations of words and phrases and their compositionality*. in *Advances in neural information processing systems*. 2013.
3. Mahajan, D., et al. *Exploring the limits of weakly supervised pretraining*. in *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.
4. Bowman, S.R., et al., *A large annotated corpus for learning natural language inference.* arXiv preprint arXiv:1508.05326, 2015.
5. Williams, A., N. Nangia, and S.R. Bowman, *A broad-coverage challenge corpus for sentence understanding through inference.* arXiv preprint arXiv:1704.05426, 2017.
6. Dolan, W.B. and C. Brockett. *Automatically constructing a corpus of sentential paraphrases*. in *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*. 2005.
7. Rajpurkar, P., et al., *Squad: 100,000+ questions for machine comprehension of text.* arXiv preprint arXiv:1606.05250, 2016.
8. Radford, A., et al., *Improving language understanding by generative pre-training*. 2018.
9. Lee, J., et al., *BioBERT: a pre-trained biomedical language representation model for biomedical text mining.* Bioinformatics, 2020. **36**(4): p. 1234-1240.
10. Beltagy, I., K. Lo, and A. Cohan, *SciBERT: A pretrained language model for scientific text.* arXiv preprint arXiv:1903.10676, 2019.
11. Joshi, M., et al., *SpanBERT: Improving pre-training by representing and predicting spans.* Transactions of the Association for Computational Linguistics, 2020. **8**: p. 64-77.
12. Zhang, Z., et al. *Semantics-aware BERT for language understanding*. in *Proceedings of the AAAI Conference on Artificial Intelligence*. 2020.
13. Feng, A., et al., *Modeling Multi-Targets Sentiment Classification via Graph Convolutional Networks and Auxiliary Relation.* Computers, Materials & Continua, 2020.
14. Yin, X., et al., *Deep Entity Linking via Eliminating Semantic Ambiguity With BERT.* IEEE Access, 2019. **7**: p. 169434-169445.
15. Zhang, Z., et al., *ERNIE: Enhanced language representation with informative entities.* arXiv preprint arXiv:1905.07129, 2019.
16. Sun, Y., et al. *ERNIE 2.0: A Continual Pre-Training Framework for Language Understanding*. in *AAAI*. 2020.
17. Logeswaran, L. and H. Lee, *An efficient framework for learning sentence representations.* arXiv preprint arXiv:1803.02893, 2018.
18. Liu, Y., et al., *ROBERTa: A robustly optimized BERT pretraining approach.* arXiv preprint arXiv:1907.11692, 2019.
19. Zellers, R., et al. *Defending against neural fake news*. in *Advances in Neural Information Processing Systems*. 2019.

20. Wang, A., et al. *Superglue: A stickier benchmark for general-purpose language understanding systems*. in *Advances in Neural Information Processing Systems*. 2019.
21. Conneau, A. and D. Kiela, *Senteval: An evaluation toolkit for universal sentence representations.* arXiv preprint arXiv:1803.05449, 2018.
22. Houlsby, N., et al., *Parameter-efficient transfer learning for NLP.* arXiv preprint arXiv:1902.00751, 2019.
23. Peters, M.E., S. Ruder, and N.A. Smith, *To tune or not to tune? adapting pretrained representations to diverse tasks.* arXiv preprint arXiv:1903.05987, 2019.
24. Iwasaki, Y., et al. *Japanese abstractive text summarization using BERT*. in *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. 2019. IEEE.
25. Sousa, M.G., et al. *BERT for Stock Market Sentiment Analysis*. in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. 2019. IEEE.
26. Yu, X., et al. *BioBERT based named entity recognition in electronic medical record*. in *2019 10th International Conference on Information Technology in Medicine and Education (ITME)*. 2019. IEEE.
27. Jiang, S., et al. *A BERT-BiLSTM-CRF Model for Chinese Electronic Medical Records Named Entity Recognition*. in *2019 12th International Conference on Intelligent Computation Technology and Automation (ICICTA)*. 2019. IEEE.
28. Inui, K., et al. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019.
29. Devlin, J., et al., *BERT: Pre-training of deep bidirectional transformers for language understanding.* arXiv preprint arXiv:1810.04805, 2018.
30. Peters, M.E., et al., *Deep contextualized word representations.* arXiv preprint arXiv:1802.05365, 2018.
31. Wei, J., et al., *NEZHA: Neural contextualized representation for chinese language understanding.* arXiv preprint arXiv:1909.00204, 2019.
32. Kalchbrenner, N., E. Grefenstette, and P. Blunsom, *A convolutional neural network for modelling sentences.* arXiv preprint arXiv:1404.2188, 2014.
33. Mikolov, T., et al. *Extensions of recurrent neural network language model*. in *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. 2011. IEEE.
34. Hochreiter, S. and J. Schmidhuber, *Long short-term memory.* Neural computation, 1997. **9**(8): p. 1735-1780.
35. Vaswani, A., et al., *Attention is all you need.* Advances in neural information processing systems, 2017. **30**: p. 5998-6008.
36. Google. *Transformer: A Novel Neural Network Architecture for Language Understanding*. 2017 [cited 2020 3 Dec]; Available from: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html.
37. Dai, A.M. and Q.V. Le. *Semi-supervised sequence learning*. in *Advances in neural information processing systems*. 2015.
38. Howard, J. and S. Ruder, *Universal language model fine-tuning for text classification.* arXiv preprint arXiv:1801.06146, 2018.

39. Wang, A., et al., *Glue: A multi-task benchmark and analysis platform for natural language understanding.* arXiv preprint arXiv:1804.07461, 2018.
40. McCann, B., et al. *Learned in translation: Contextualized word vectors*. in *Advances in Neural Information Processing Systems*. 2017.
41. Lample, G. and A. Conneau, *Cross-lingual language model pretraining.* arXiv preprint arXiv:1901.07291, 2019.
42. Radford, A., et al., *Improving language understanding with unsupervised learning.* Technical report, OpenAI, 2018.
43. Dong, L., et al. *Unified language model pre-training for natural language understanding and generation*. in *Advances in Neural Information Processing Systems*. 2019.
44. Sun, Y., et al., *Ernie: Enhanced representation through knowledge integration.* arXiv preprint arXiv:1904.09223, 2019.
45. Song, K., et al., *Mass: Masked sequence to sequence pre-training for language generation.* arXiv preprint arXiv:1905.02450, 2019.
46. Yang, Z., et al. *XLNet: Generalized autoregressive pretraining for language understanding*. in *Advances in neural information processing systems*. 2019.
47. Chan, W., et al., *KERMIT: Generative insertion-based modeling for sequences.* arXiv preprint arXiv:1906.01604, 2019.
48. Vaswani, A., et al. *Attention is all you need*. in *Advances in neural information processing systems*. 2017.
49. Peters, M.E., et al., *Dissecting contextual word embeddings: Architecture and representation.* arXiv preprint arXiv:1808.08949, 2018.
50. Peters, M.E., et al., *Semi-supervised sequence tagging with bidirectional language models.* arXiv preprint arXiv:1705.00108, 2017.
51. Bai, S., J.Z. Kolter, and V. Koltun. *Deep equilibrium models*. in *Advances in Neural Information Processing Systems*. 2019.
52. Dai, Z., et al., *Transformer-xl: Attentive language models beyond a fixed-length context.* arXiv preprint arXiv:1901.02860, 2019.
53. Lan, Z., et al., *AlBERT: A lite BERT for self-supervised learning of language representations.* arXiv preprint arXiv:1909.11942, 2019.
54. Zhu, C., et al., *Freelb: Enhanced adversarial training for language understanding.* arXiv preprint arXiv:1909.11764, 2019.
55. Raffel, C., et al., *Exploring the limits of transfer learning with a unified text-to-text transformer.* arXiv preprint arXiv:1910.10683, 2019.
56. Zhang, Z., et al., *Semantics-aware BERT for language understanding.* arXiv preprint arXiv:1909.02209, 2019.
57. Wang, W., et al., *StructBERT: Incorporating language structures into pre-training for deep language understanding.* arXiv preprint arXiv:1908.04577, 2019.
58. Jiao, X., et al., *TinyBERT: Distilling BERT for natural language understanding.* arXiv preprint arXiv:1909.10351, 2019.
59. Liu, X., et al., *Multi-task deep neural networks for natural language understanding.* arXiv preprint arXiv:1901.11504, 2019.
60. Jiang, H., et al., *Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization.* arXiv preprint arXiv:1911.03437, 2019.

61. Su, T.-C. and H.-C. Cheng, *SesameBERT: Attention for Anywhere.* arXiv preprint arXiv:1910.03176, 2019.
62. Clark, K., et al., *Electra: Pre-training text encoders as discriminators rather than generators.* arXiv preprint arXiv:2003.10555, 2020.
63. Wang, W., et al., *MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers.* arXiv preprint arXiv:2002.10957, 2020.
64. Xu, C., et al., *BERT-of-theseus: Compressing BERT by progressive module replacing.* arXiv preprint arXiv:2002.02925, 2020.
65. Goyal, S., et al., *PoWER-BERT: Accelerating BERT inference for Classification Tasks.* arXiv preprint arXiv:2001.08950, 2020.
66. Clark, K., et al., *Bam! born-again multi-task networks for natural language understanding.* arXiv preprint arXiv:1907.04829, 2019.
67. Liu, X., et al., *Improving multi-task deep neural networks via knowledge distillation for natural language understanding.* arXiv preprint arXiv:1904.09482, 2019.
68. Chen, D., et al., *AdaBERT: Task-Adaptive BERT Compression with Differentiable Neural Architecture Search.* arXiv preprint arXiv:2001.04246, 2020.
69. Bao, H., et al., *UniLMv2: Pseudo-Masked Language Models for Unified Language Model Pre-Training.* arXiv preprint arXiv:2002.12804, 2020.
70. Lewis, M., et al., *Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.* arXiv preprint arXiv:1910.13461, 2019.
71. Chang, W.-C., et al., *X-BERT: eXtreme Multi-label Text Classification with using Bidirectional Encoder Representations from Transformers.* arXiv preprint arXiv:1905.02331, 2019.
72. Shoeybi, M., et al., *Megatron-lm: Training multi-billion parameter language models using gpu model parallelism.* arXiv preprint arXiv:1909.08053, 2019.
73. Chadha, A. and R. Sood, *BERTQA--Attention on Steroids.* arXiv preprint arXiv:1912.10435, 2019.
74. Kao, W.-T., et al., *Further Boosting BERT-based Models by Duplicating Existing Layers: Some Intriguing Phenomena inside BERT.* arXiv preprint arXiv:2001.09309, 2020.
75. Nguyen, M.-T., et al. *Transfer learning for information extraction with limited data*. in *International Conference of the Pacific Association for Computational Linguistics*. 2019. Springer.
76. Wang, B. and C.-C.J. Kuo, *SBERT-WK: A Sentence Embedding Method by Dissecting BERT-based Word Models.* arXiv preprint arXiv:2002.06652, 2020.
77. Moradshahi, M., et al., *HUBERT Untangles BERT to Improve Transfer across NLP Tasks.* arXiv preprint arXiv:1910.12647, 2019.
78. Gong, L., et al. *Efficient training of BERT by progressively stacking*. in *International Conference on Machine Learning*. 2019.
79. Sun, Y., et al., *Ernie 2.0: A continual pre-training framework for language understanding.* arXiv preprint arXiv:1907.12412, 2019.
80. Yang, Z., et al. *XLNet: Generalized autoregressive pretraining for language understanding*. in *Advances in neural information processing systems*. 2019.
81. Dong, L., et al. *Unified language model pre-training for natural language understanding and generation*. in *Advances in Neural Information Processing Systems*. 2019.

82. Wang, W., et al., *StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding.* arXiv preprint arXiv:1908.04577, 2019.
83. Clark, K., et al. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. in *International Conference on Learning Representations*. 2019.
84. Chen, Z., et al., *Quora question pairs*. 2018, Quora.
85. Socher, R., et al. *Recursive deep models for semantic compositionality over a sentiment treebank*. in *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013.
86. Warstadt, A., A. Singh, and S.R. Bowman, *Neural network acceptability judgments.* Transactions of the Association for Computational Linguistics, 2019. **7**: p. 625-641.
87. Bentivogli, L., et al. *The Fifth PASCAL Recognizing Textual Entailment Challenge*. in *TAC*. 2009.
88. Levesque, H., E. Davis, and L. Morgenstern. *The winograd schema challenge*. in *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*. 2012. Citeseer.