Develop New Transformer Architecture For Question and Answering

Nirbhay P. Tandon
Master's of Science in Data Science
Liverpool John Moore's University

August, 2021

Contents

List of Figures

List of Tables

List of Abbreviations

ALBERT ... A LITE BERT

Ans-Ptr ..... Answer-Pointer layer

BERT ...... Bidirectional Encoder Representations from Transformers

BiLSTM .... Bidirectional LSTM

BLEU ...... Bilingual Evaluation Understudy

BOLT ...... BERT Optimized LSTM Transformer

BPPTs ..... Back-Propagation Through Time

CEC .... Constant Error Carousel

CNNs .... Convolutional Neural Networks

DistilBERT . Distilled Version of BERT

EM .... Exact Match

FFN .... Feed Forward Network(s)

GloVe .... Global Vectors for Word Representation

GPUs .... Graphical Processing Units

LSTMs ..... Long Short-Term Memory Architectures

MCTest ..... Machine Comprehension of Text

MLM .... Masked Language Model

NLP .... Natural Language Processing

NLU .... Natural Language Understanding

QASENT ... A challenge dataset for open-domain question answering

QnA .... Question and Answering

RACE ...... ReAding Comprehension Dataset From Examinations

RNNs .... Recurrent Neural Networks

RoBERTa ... Robustly Optimized BERT

SQuAD ..... Stanford Question Answering Dataset

SuBiLSTM .. Suffix Bidirectional LSTM

VRAM ...... Virtual Random Access Memory

WMT .... Workshop on Statistical Machine Translation

Abstract

Attention-based Transformer architectures have become the norm of current Natural Language Processing applications. Google began this trend back in 2017 with their paper *Attention Is All You Need*, by introducing the Transformer architecture that works solely on attention mechanisms. The purpose of this work will be to introduce a variation of the Transformer architecture, BERT Optimized LSTM Transformer or *BOLT*. BOLT is an attempt at combining LSTM based Pointer networks along with a Bidirectional Transformer variant to produce another lighter model specifically for the use of Question Answering. Compare and contrast its performance against other architectures such as RoBerta, DistilBERT and ALBERT using the SQuAD 2.0 Dataset. Through this research it is shown that BOLT is at par with current state of the art architectures and that by combining an LSTM with a Transformer can actually yield slightly better results when correctly optimized.

Chapter 1

Introduction

One of the most philosophical questions that anyone can encounter is *"Can an answer exist without a question?"*. While that question exists to be solved by philosophers and thinkers, this work shall aim to address the *answer* that can be provided to a question in terms of the context that has been asked. While the scope of exploration in the field Natural Language Processing(NLP) is vast and many unexplored areas exist, in this thesis the focus is entirely on developing a novel architecture that provides *highly contextual and relevant answers* to the questions asked of it.

1.1 Background Of The Study

The earliest examples of Question-Answering(QA) systems can be traced back to 1961, in the form of BASEBALL,(Green Jr et al., 1961), a system developed by Bert F. Green and team to answer basic questions posed using punch cards in ordinary English language about stored data on baseball games. The team implemented a dictionary structure to allow for answers to be looked up and printed. For example, the system could answer very direct questions such as: "Where did each team play on July 7?". A highly impressive feat for the early days of neural networks in the 1960s, the work done by Green paved the way for systems like LUNAR,(Woods and WA, 1977), answered geological questions for the rocks brought back from the Apollo moon missions. LUNAR, which was first presented in 1971, had an astounding 90% accuracy rate for questions that were asked by people who didn't know how the system worked! Question-Answering based systems have gained a lot of popularity, especially in the form of "chatbots". These systems depend highly on contextual understanding of the input, the training corpus and the question asked. They use this knowledge to output an answer that can help the user with whatever their query is. Recurrent neural networks and architectures based on them, have been able to provide great advancements in the field of Question-answering and chatbots in general. However, there is a behaviour of over-fitting and a lack of contextual understanding of the question. This, coupled with long training times and extremely complex mathematical model designs, have often kept the field of Natural Language Processing slightly obscured from the masses.

The task of Question-Answering is a machine comprehension problem. On digging deeper, once can classify it as a sequential problem too. One might wonder how it is a sequential problem? The answer to that is simple, traditionally QA systems work by "sequentially reading" the input and producing an output based on what they

understand semantically or contextually. In a more mathematical context, one can say that the problem is of *answer selection* and not *answer provision*. Through numerous studies that have been done, some of which shall be analyzed in detail in chapter 2, it becomes clear that the problem is not how to provide an answer to a question, it is if the answer provided is the correct one or not in the *context* in which the question has been posed.

The research presented forth here is aims to propose a new variation to an already successful architecture in the field of NLP, namely *Transformers*, created by the team at Google (Vaswani et al., 2017). A critically assessed attempt is made at combining state of the art approaches that have been treated as separate elements so far and to show that the proposed improvements are at par, if not better than the current approach.

A Transformer is a form of transduction model that relies solely on self-attention to figure out how to represent its inputs and outputs. It does so without the use of any sequence aligned recurrent neural networks(RNNs) or convolutions.

The area of Natural language Processing has taken significant leaps in the last two decades. Work done towards improving the ability of machine learning models to first recognize words, then sentences, followed by contextual understanding has led to several interesting and novel approaches in the field. From early on neural networks to creating Long Short-Term Memory architectures (Schmidhuber and Hochreiter, 1997) by Sepp Hochreiter and Jurgen Schmidhuber in the mid-'90s that resolved the vanishing gradient problem of classical neural networks, NLP architectures have come a long way.

The latest advancements in this field come from Google's research lab in the form of *Transformers*. This is studied in a bit more detail later in 2. However, no model can be successful without a good dataset to train on. This is where the SQuAD 2.0 dataset (Rajpurkar et al., 2018) comes in. This dataset is what forces the machine learning models to do contextual understanding. One might even say that it forces the models to "think" for themselves before answering a task.

1.2 Aims And Objectives

The main aim of this research is to propose an improvement to the transformer architecture that can perform better at Q&A using the SQuAD 2.0 dataset. This shall require understanding the nuances of various Transformer-type architectures like AL-BERT (Lan et al., 2019), RoBerta (Liu et al., 2019) and DistilBERT (Sanh et al., 2019) which are variations of the BERT (Devlin et al., 2018) architecture.

To achieve the main aim of this research it is important to first outline a few steps that will be taken:

1. Obtain SQuAD2.0 dataset and prepare it for training

2. Implement the existing models such as ALBERT, RoBerta and DistilBERT that are available via libraries such as HuggingFace (HuggingFace, 2021) and PyTorch on the dataset

3. Obtain F1, Exact Match(EM), Training Loss and Validation Loss scores for existing models to treat them as benchmark values

4. Identify drawbacks of the current architectures

5. Design BOLT architecture and evaluate its performance against the provided benchmark scores

6. Fine-tune the architecture, re-evaluate and report improvements

7. Compare the results of BOLT model with the benchmark scores.

1.3 Scope Of The Study

This research work is entirely focused on the area of Question-Answering based networks. This focus helps in significantly narrowing down the scope and critically assessing the problem that needs to be solved. The practical applications of the field of QA are endless. From chatbots to QA systems on domain-specific knowledge or conversational applications for voice assistants, a light, robust, highly accurate Transformer based architecture can help ease the daily lives of millions of users.

1.4 Significance Of The Study

The current state of the art systems, some of which have been highlighted in section 1.1, suffer from various drawbacks and even the best models have only been able to achieve an EM score of 90.871 with an F1 score of 93.183 on the SQuAD 2.0 leaderboard (Rajpurkar, 2021). Current models such as ALBERT and DistilBERT are highly useful, however, they aren't scalable entirely for QA applications. A narrowed focus on the specific field of Question-Answering can help eliminate certain issues with the current models such as contextual uderstanding and being able to get correct responses to questions.

1.5 Structure Of The Study

This report is divided into 6 main parts, each chapter is further broken down into sections and subsections to localize the reference and use it further in the global context of this research.

In chapter 1, an introduction to the concept of Question-Answering, a brief look at the problem, motivation, scope of this research and its significance have been provided. Chapter 1 lays the basis of the problem, takes the reader briefly back in time to where QA systems first started and how they have advanced today.

Chapter 2, looks in detail at the background studies that have been briefly mentioned in section 1.1. In this chapter critical analysis is conducted how some of the work that has been selected is relevant to what this research aims to highlight, it helps in understanding the merits and shortcomings of current state of the art approaches. Detailed analysis of Long Short-Term Memory(LSTM) and Transformer architectures is written, with special attention to applications in the field of Question Answering. There is particular focus on the advancements and applications of existing architectures as well as acknowledgement of why LSTMs reigned supreme for almost 2 decades. Transformers provide a unique opportunity to perform highly accurate

predictive tasks in NLP. The most key advances to both LSTMs and Transformers are their bi-directional applications which are also discussed. Finally, this chapter ends with a summary of the approaches and highlights certain drawbacks. These drawbacks are discussed in more detail in 4.

Next, chapter 3 proposes the research methods that have been implemented. It explains the various metrics that have been used and why they were chosen, specifically highlighting why Accuracy isn't a good measure for creating a successful model and highlighting the use of Precision, Recall and F1 as useful metrics for the experiments conducted in this particular thesis, Exact Match scores are also introduced and discussed in this chapter. The chapter ends with a research flow diagram describing the processes followed for the experiments and application stages of various methods in a visual manner.

In chapter 4 the proposed architecture is discussed along with drawbacks of current architectures. The chapter also describes ways in which the proposed architecture can be refined.

Finally, in chapters 5 and 6 the results of the models are discussed while comparing them to current architectures and their benchmarks. The outcomes and various recommendations are outlined that can help improve the work presented in this thesis further.

Appendix A is where the reader can refer to the Gantt chart for the thesis work done.

Chapter 2

Literature Review

To be able to effectively deliver on the aims and objectives defined in section 1.2, it is important to first critically analyse the work that has been done in the field of NLP and, in particular, the field of Question-Answering based networks. This chapter explores a few key model architectures and how they led to reached Transformers, which form the base of the proposed architecture.

2.1 Long Short-Term Memory

In 1997 (Schmidhuber and Hochreiter, 1997) introduced to the world a gradient descent based model in the form of Long Short-Term Memory. They set out to solve the problem of the vanishing gradient which was often seen in "Back-Propagation Through Time"(BPPTs) based neural networks. Extensive studies done on this, some by Hochreiter himself, showed that the problem of vanishing gradients is a real one. It was also seen that in case of BPTTs, the error back-flow mechanisms would either blow up or also suffer from vanishing gradients leading to either oscillating weights or learning to bridge long time gaps would not work. To remediate this, (Schmidhuber and Hochreiter, 1997) introduced the concept of a *constant* error flow through the internal states of special units.

Long Short-Term Memory by (Schmidhuber and Hochreiter, 1997), highlights that the work previously done using various gradient-descent variants, time-delays, time constants, use of higher order units to influence connections when the network receives conflicting error signals, Kalman filters, Second order nets etc. still suffer from some of the same problems that have always existed. In the case of Ring's approach (Ring, 1993), the network can be really fast but also suffers from memory constraints and lack of generalization. This happens because to bridge n time lags, the model adds n additional units.

It is essential for to understand how LSTMs function as it forms the basis of Transformers to exist. Hochreiter and Scmidhuber designed a *memory cell* that allows for a constant error flow. This is achieved by introducing *constant error carousel*(CEC), a multiplicative input gate unit and a multiplicative output gate unit. These *multiplicative* units help shield the contents of the memory cell from irrelevant inputs. The CEC, input and output gates together form the *memory cell*, as seen in Fig. 2.1. The central feature for LSTM's memory cell is the CEC. The self-recurrent connection, where weight is 1.0, is what helps create a time-delayed feedback loop by 1.

The CEC uses gate units to circumvent the issue with conflicting weights in the
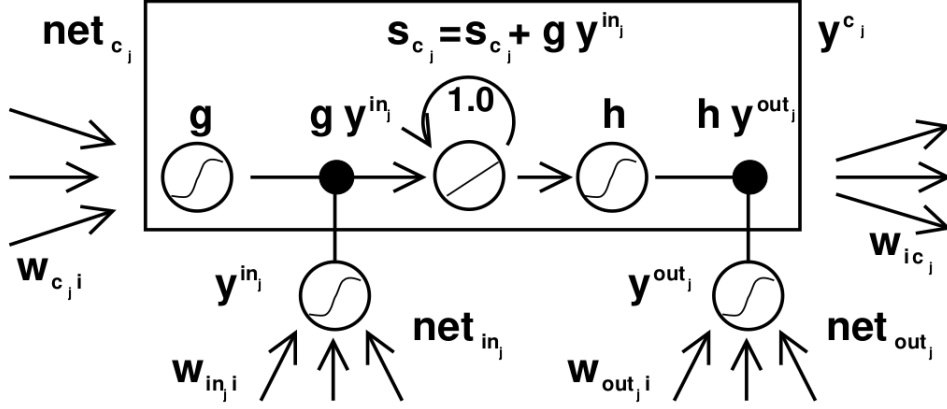
Figure 2.1: LSTM Memory cell with a Constant Error Carousel having fixed weight 1.0, (Schmidhuber and Hochreiter, 1997)

input and output layers. The input layer has control over when to use and when to override the information in the CEC but has no control over the error signals that are in the memory cell. The output gate, however, uses a feedback mechanism to assess when to superimpose different error signals. It also falls on the output gate to *learn* which errors to trap in the CEC by appropriately scaling them. The gates have to learn when to release and trap errors therefore controlling the access to the CEC, helping maintain constant error flow. To mathematically understand the memory cell from fig. 2.1, Schmidhuber and Hochreiter proposed the following set of equations 2.1 - 2.3. More details on this can be found by referring to the appendix section A.1 in (Schmidhuber and Hochreiter, 1997).

$$y^{out_j}(t) = f_{out_j}\left(net_{out_j}(t)\right); y^{in_j}(t) = f_{in_j}\left(net_{in_j}(t)\right) \tag{2.1}$$

where

$$net_{out_j}(t) = \sum_u w_{out_j u} y^u(t-1) \tag{2.2}$$

and

$$net_{in_j}(t) = \sum_u w_{in_j u} y^u(t-1) \tag{2.3}$$

A generalised form of this equation can be represented as:

$$net_{c_j}(t) = \sum_u w_{c_j u} y^u(t-1) \tag{2.4}$$

At time t, c_j's output $y^{c_j}(t)$ is computed as:

$$y^{c_j}(t) = y^{out_j}(t) h(s_{c_j}(t)), \tag{2.5}$$

where the "internal state" $s_{c_j}(t)$ is:

$$s_{c_j}(0) = 0, s_{c_j}(t) = s_{c_j}(t-1) + y^{in_j}(t) g\left(net_{c_j}(t)\right) \text{ for } t > 0 \tag{2.6}$$

Where u may stand for input, output or gate units, memory cells or even conventional hidden units, if they are being used.

Figure 2.2 shows the internal workings of how an LSTM network could look like.
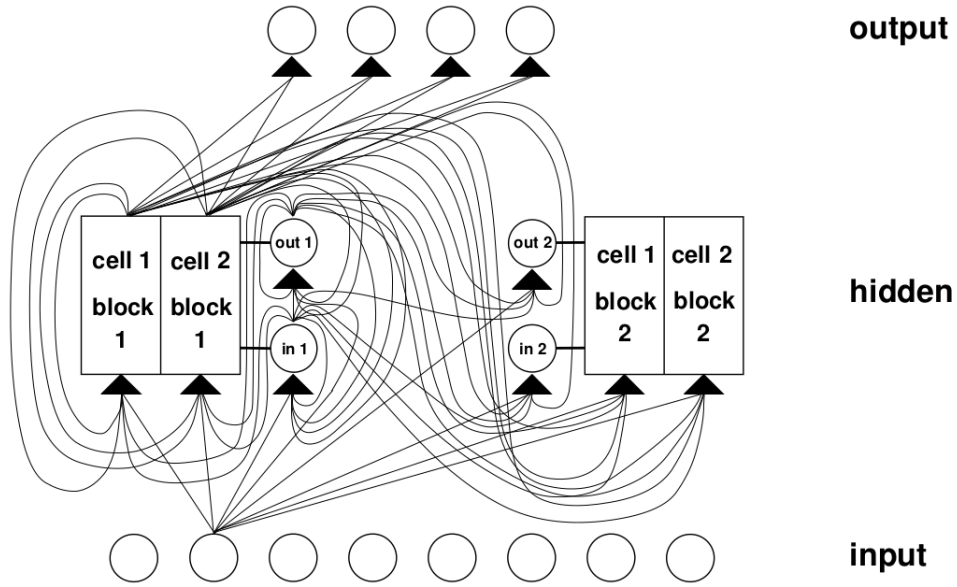
6

Figure 2.2: LSTM network with 8 input cells, 4 output cells and 2 memory cells of block size 2. Here $in1$ and $out1$ represent the input and output gates and $cell/block1$ is the first memory cell. The internal architecture of $cell/block1$ is similar to fig. 2.1. (Schmidhuber and Hochreiter, 1997)

A series of experiments were performed to prove that LSTMs are indeed "state-of-the-art" and provide an improvement over then existing RNN based architectures. Experiments around temporal order, which is defined as a series of events that happen in a particular order, that were previously unsolved by RNNs, were successfully solved by LSTMs. This was possible because of how LSTMs address time lags vs. how RNNs suffer from vanishing gradients. Details of the analysis and experiments conducted are available in section 5 of the paper "Long Short-Term Memory" by (Schmidhuber and Hochreiter, 1997), with experimental summaries in tables 10 and 11 of the same. Being able to solve for temporal order based problems allowed LSTMs to be capable of solving various real world problems that will be discussed later. Temporal orders are key in contextual understanding for NLP based tasks and especially in the case of QA based networks. Let's now look at some advantages and limitations of LSTMs, before moving on to understand how they have been applied in the field of QA.

Advantages of LSTMs

- LSTMs deal with long time lags by bridging the gaps using constant error flow in the memory cells.

- LSTMs respond well to generalization problems, even if the input sequences are widely separated.

- They work well with a variety of braod range hyper-parameters and don't usually require tuning.

- Their time and weight update complexity is the same as BPTT, ie. $O(1)$. The advantage for LSTMs however, is that it is local in both space and time.

7

Limitations of LSTMs

- LSTMs use memory cells which require an additional input and output unit. This *hidden unit* is replaced by 3 units in an LSTM architecture, compared to 9 in an RNN. A fully connected LSTM will have 3^2 connections however.

- LSTMs dont work well when they get the entire input in one go. The architecture is unable to generalize well by random weight guessing in this case.

- They do not have the ability to "natively" count discrete time steps which might be useful in some applications. This isn't a big issue for the case of QA based problems.
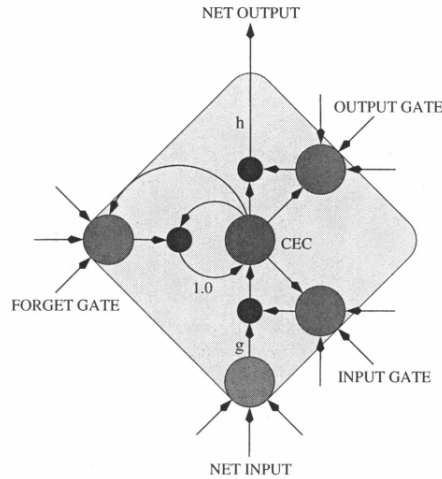
2.1.1 BiLSTMs

Figure 2.3: A memory cell for a biLSTM highlighting a *forget gate*. This forget gate helps in scaling the internal state of a memory cell, for example by resetting the state to 0. We can see that it is different to fig. 2.1, with the implementation of a connection to the forget gate, while still having an internal weight of 1. (Graves and Schmidhuber, 2005)

While LSTMs have found various applications across the field of machine and deep learning, QA seems to be one where being able to achieve consistent results of accuracy as well as having an exact match seems to elusive. Research carried out at IBM by Tan, Santos and co. looked at approaching this problem by using something called Bidirectional LSTMs or BiLSTM, (Graves and Schmidhuber, 2005). Introduced by Graves and Schmidhuber in their paper *Framewise Phoneme Classification with Bidirectional LSTM Networks*, the biLSTM implements a *forget gate* as seen in fig. 2.3. An advantage of biLSTMs over traditional LSTMs is that you feed the data twice, once from the beginning to the end and then from the end to the beginning. This helps the model contextualize and adjust weights better, therefore creating a more robust model that learns and performs better overall when compared to traditional LSTMs and RNNs.

BiLSTMs were introduced as an approach towards phoneme classification for speech recognition applications, however, in their paper "LSTM-BASED DEEP LEARNING MODELS FOR NON -FACTOID ANSWER SELECTION", (Tan et al., 2015) have

	Questions	Answers	Question Word Count
Train	12887	18540	92095
Dev	1000	1454	7158
Test1	1800	2616	12893
Test2	1800	2593	12905

Table 2.1: InsuranceQA Corpus Details: Some questions can have multiple answers, therefore the number of answers is greater than the number of questions (Feng et al., 2015).
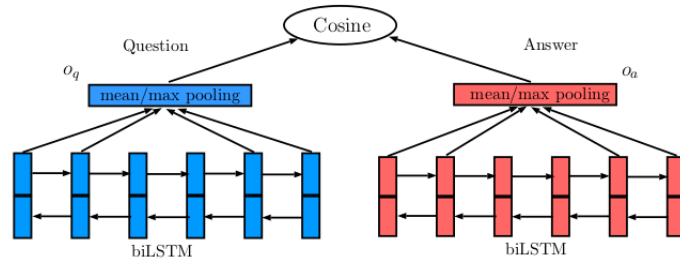

Figure 2.4: Basic QA-LSTM model depicting biLSTM implementation (Tan et al., 2015)

shown that extending the biLSTM model 2 different approaches for QA has its advantages. One approach, using a more composite method of using CNNs for representing questions as well as answers. The other, by implementing an *attention mechanism* that helps in generating an answer representation according to the question context. Their framework is based on developing a biLSTM model for both questions and answers, applying a connective pooling layer and using a cosine similarity metric to calculate the degree of matching answers. They utilized the InsuranceQA(Feng et al., 2015) dataset that was created in 2015 and tested primarily on CNN based networks, details of the dataset can be found in table 2.1.

Pooling layers are known to suffer from an incapability to retain local linguistic information. To combat this the team proposed an additional CNN layer on top of the biLSTM layer. Additionally, they added a simple and efficient attention model to help their biLSTM model better distinguish between candidate answers according to the question context. This study is significant in the field of both deep learning based models and for the field of Question-Answering because it requires little or no feature engineering to achieve significant results. In section 2.2, how biLSTMs are implemented with Transformers to produce an even better model.

In their work (Tan et al., 2015) have shown that biLSTMs are capable of capitalizing long-range sequential context information. This is important as it is often seen that the answer is not directly semantically related to the question but more contextually related. Long-range exploitations help the model understand the question more holistically, therefore helping produce a more relevant answer.

BiLSTMs have an advantage over traditional LSTMs in that they utilize both previous and future contextual information by processing the input in both directions, forward and backward. This leads to the generation of two independent sequences of output vectors form LSTMs. The final output is a concatenation of the outputs from both directions ie. $h_t = \overrightarrow{h_t} || \overleftarrow{h_t}$. The **QA-LSTM** model proposed by (Tan et al., 2015) can be seen in fig. 2.4. It can be seen that it generates a distributed representation
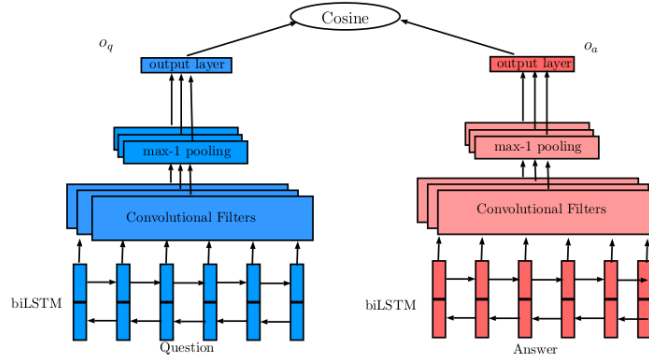
Figure 2.5: QA-LSTM with CNN layer (Tan et al., 2015)

for both questions(blue) and answers(red) before moving on to perform pooling and cosine similarity operations. In fig. 2.5, it can be seen that the CNN layer has been implemented after the biLSTM layers and before the pooling layers. The pooling layers have been modified to be max-k pooling layers, similar to standard CNNs.

The implementation of this convolution layer forces localized interactions between the inputs within a filter size of m. For each window of size m in biLSTM output vectors ie.

$$\mathbf{H}_m(t) = [\mathbf{h}(t), \mathbf{h(t+1)}, ..., \mathbf{h}(t + m - 1)] \tag{2.7}$$

where t is a time step and the convolution filter will generate one value as follows

$$o_F(t) = tanh\left[\left(\sum_{i=0}^{m-1} \mathbf{h}(t+i)^T \mathbf{F}(i)\right) + b\right] \tag{2.8}$$

Using k-MaxPooling,(Tan et al., 2015) emphasized that a maximum of k values will be kept for one filter, thereby indicating the highest degree that a filter matches the input sequence. To end this approach, there are N parallel filters, with different parameter initialization which provide the CNN layer with N-dimension output vectors. This produces two output vectors of dimension kN for each question and answer. Their experiments intuitively took $k = 1$ as anything greater showed no improvements and it helped emphasize aspects of the answer such that this CNN based hybrid biLSTM could differentiate ground truths and incorrect answers efficiently.

The second approach adopted in this paper is actually inspired by another paper (Hermann et al., 2015) where attention based deep LSTM networks were used to enhance the capabilities of reading comprehension in LSTM based models. Here, in (Tan et al., 2015), a modification has been introduced in the form of attention-based connections along with biLSTM layers as depicted in fig. 2.6.

Before conducting a mean/average pooling operation on the output from every biLSTM, the output vectors are multiplied by a softmax weight that is determined by the question embeddings from the biLSTM. It was shown that given an output vector from a biLSTM for an answer at time t, $\mathbf{h}_a(t)$ and the question embedding, o_q, the updated vector $\tilde{\mathbf{h}}_a(t)$ for each answer token can be given as:

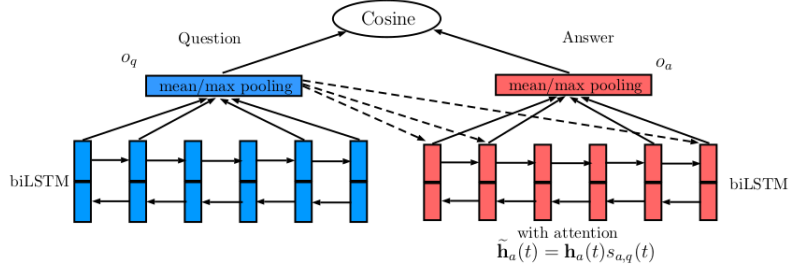$$\mathbf{m}_{a,q}(t) = tanh(\mathbf{W}_{am}\mathbf{h}_a(t) + \mathbf{W}_{qm}o_q) \tag{2.9}$$

10

Figure 2.6: QA-LSTM with Attention layer (Tan et al., 2015)

	Model	Validation	Test1	Test2
A	QA-LSTM basic-model(head/tail)	54.0	53.1	51.2
B	QA-LSTM basic-model(avg pooling)	58.5	58.2	54.0
C	QA-LSTM basic-model(max pooling)	64.3	63.1	58.0
D	QA-LSTM/CNN(fcount=1000)	65.5	65.9	62.3
E	QA-LSTM/CNN(fcount=2000)	64.8	66.8	62.6
F	QA-LSTM/CNN(fcount=4000)	66.2	64.6	62.2
G	QA-LSTM with attention (max pooling)	66.5	63.7	60.3
H	QA-LSTM with attention (avg pooling)	**68**.4	**68**.1	62.2
I	QA-LSTM/CNN (fcount=4000) with attention	67.2	65.7	**63.3**

Table 2.2: Experimental results from (Tan et al., 2015), highlighting how QA-LSTM with attention outperforms its various modifications

$$s_{a,q}(t) \propto exp(\mathbf{w}_{ms}^T \mathbf{m}_{a,q}(t)) \tag{2.10}$$

$$\tilde{\mathbf{h}_a} = \mathbf{h}_a(t)s_{a,q}(t) \tag{2.11}$$

where $\mathbf{W}_{am}, \mathbf{W}_{qm}$ and $\mathbf{w}_m s$ are attention parameters. Attention parameters show somewhat similar behaviour to tf-idf where some words get more attention or weights associated to them. The main difference being that the attention mechanism adjusts its weights according to question information.

Critically, this attention based approach from (Tan et al., 2015) emphasizes on attention-driven representations and uses that to calculate the distances between questions and their respective answers. A combination approach with CNNs and attention based connections helps in localizing various internal connections, correctly scaling the errors and reducing unnecessary noise. From eq. 2.9-2.11, (Tan et al., 2015) one can mathematically prove how the average distances are computed. Experiments conducted on the InsuranceQA dataset showed that the *QA-LSTM/CNN with Attention* model can outperform the taken baseline models, which were based purely on QAs using CNNs.

An improvement to this approach was done in 2018 by (Brahma, 2018) who proposed *Suffix Bidirectional LSTM* or *SuBiLSTM* adding prefixes and suffixes to the inputs over both directions. There are a few obvious differences between the biLSTM and SuBiLSTM like the doubling of parameters, increased time complexity which happens over quadratic time for worst case scenarios compared to linear in biLSTMs. (Brahma, 2018) of this paper has shown that despite higher time complexity, there are

significant gains in accuracy in text encoding and classification in their experiments, of particular note is the Paraphrase Detection experiment which uses GloVe embeddings and shows an 88.2% score in the test set, which is marginally higher than other models implemented. The work presented in this thesis will also implement paraphrase detection and GloVe embeddings to potentially improve model performance and answer detection.

2.1.2 LSTM And Question Answering

In 2016, the SQuAD dataset (Rajpurkar et al., 2016) was released, that generated quite a buzz in the field of QA based NLP tasks due to its nature of being extremely versatile and crowdsourced.

Using the SQuAD dataset (Wang and Jiang, 2016) developed a match-LSTM (Wang and Jiang, 2015) based model that they showed performed significantly better on the dataset than the work done by (Rajpurkar et al., 2016). Their work shows that using a match-LSTM approach combined with a Pointer Net model (Vinyals et al., 2015), that aids token predictions using only input sequences rather than using any form of large fixed vocabulary, therefore allowing for generation of answers with multiple tokens.

The approach by (Wang and Jiang, 2016) was tw-fold. The first approach implemented as a sequence model and the second as a boundary model, which was further extended with a search function.

The match-LSTM model architecture consists of:

- **LSTM Preprocessing Layer**: As can be seen from 2.7, there are 2 LSTM pre-processing layers. This layer is used to integrate contextual information directly into the representation of how each token in the passage and question is represented. This is based directly on the simple LSTM (Schmidhuber and Hochreiter, 1997) and not the BiLSTM.

- **Match-LSTM Layer**: Here, the model treats the question as a premise and the passage as hypothesis. The layer sequentially reads the passage and calculates an attention weight vector $\vec{\alpha}_i \in \mathbb{R}^Q$ as:

$$
\begin{aligned}
\overrightarrow{\mathbf{G}}_i &= \tanh\left(\mathbf{W}^{\mathrm{q}}\mathbf{H}^{\mathrm{q}} + \left(\mathbf{W}^{\mathrm{p}}\mathbf{h}_i^{\mathrm{p}} + \mathbf{W}^{\mathrm{r}}\overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}} + \mathbf{b}^{\mathrm{p}}\right) \otimes \mathbf{e}_Q\right) \\
\vec{\alpha}_i &= \mathrm{softmax}\left(\mathbf{w}^{\top}\overrightarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q\right)
\end{aligned}
\tag{2.12}
$$

where $\mathbf{W}^{\mathrm{q}}, \mathbf{W}^{\mathrm{p}}, \mathbf{W}^{\mathrm{r}} \in \mathbb{R}^{l \times l}, \mathbf{b}^{\mathrm{p}}, \mathbf{w} \in \mathbb{R}^l$ and $b \in \mathbb{R}$ are parameters to be learned, $\overrightarrow{\mathbf{h}}_{i-1}^{\mathrm{r}} \in \mathbb{R}^l$ is the hidden vector of the one-directional match-LSTM at position $i-1$, and the outer product $(\cdot \otimes \mathbf{e}_Q)$ produces a matrix or row vector by repeating the vector or scalar on the left for Q times. To simply summarise, the above equation 2.12 produces $\vec{\alpha}_{i,j}$ that gives the degree of matching between token i in the passage with token j in the question. This is the forward pass representation of the mathc-LSTM, the backward pass, which generates encodings for each token in the passage is given by:

$$
\begin{aligned}
\overleftarrow{\mathbf{G}}_i &= \tanh\left(\mathbf{W}^{\mathrm{q}}\mathbf{H}^{\mathrm{q}} + \left(\mathbf{W}^{\mathrm{p}}\mathbf{h}_i^{\mathrm{p}} + \mathbf{W}^{\mathrm{r}}\overleftarrow{\mathbf{h}}_{i+1}^{\mathrm{r}} + \mathbf{b}^{\mathrm{p}}\right) \otimes \mathbf{e}_Q\right) \\
\overleftarrow{\alpha}_i &= \mathrm{softmax}\left(\mathbf{w}^{\top}\overleftarrow{\mathbf{G}}_i + b \otimes \mathbf{e}_Q\right)
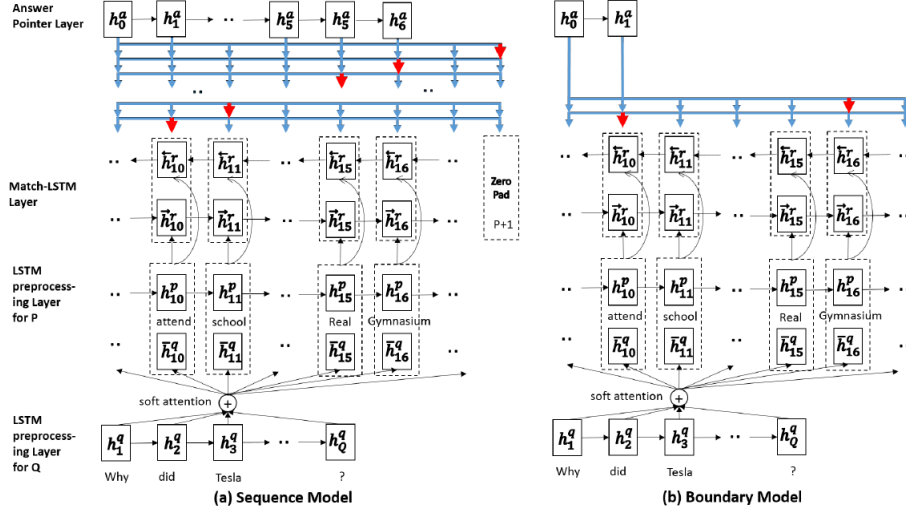\end{aligned}
\tag{2.13}
$$

Figure 2.7: Overview of the 2 approaches by (Wang and Jiang, 2016), showing the Sequence Model on the left and the Boundary Model on the right.

The equations above, 2.12 and 2.13 are presented in (Wang and Jiang, 2016).

- **Answer Pointer Layer**: This layer, the Answer-Pointer layer(Ans-Ptr), introduced by (Vinyals et al., 2015). It uses the sequence $\mathbf{H}^r$ as input.

It is important to highlight that the sequence model represents the answers as integers in a sequence that refer to the position of the selected token in the original passage. The Ans-Ptr layer models these sequentially as well. This thesis will focus on the boundary model from the paper by (Wang and Jiang, 2016). Table 2.3 shows the results are significantly better for SQuAD 1.1 dataset. Further enhancements to this have been made and various other models have come out on top, however this paper pioneered the use of a bidirectional Ans-Ptr approach.

	l	$|\theta|$	Exact Match		F1	
			Dev	Test	Dev	Test
Random Guess	−	0	1.1	1.3	4.1	4.3
Logistic Regression	−	−	40.0	40.4	51.0	51.0
DCR	−	−	62.5	62.5	71.2	71.0
Match-LSTM with Ans-Ptr (Sequence)	150	882 K	54.4	−	68.2	−
Match-LSTM with Ans-Ptr (Boundary)	150	882 K	61.1	−	71.2	−
Match-LSTM with Ans-Ptr (Boundary+Search)	150	882 K	63.0	−	72.7	−
Match-LSTM with Ans-Ptr (Boundary+Search)	300	3.2M	63.1	−	72.7	−
Match-LSTM with Ans-Ptr (Boundary+Search+b)	150	1.1M	63.4	−	73.0	−
Match-LSTM with Bi-Ans-Ptr (Boundary+Search+b)	150	1.4M	**64.1**	**64.7**	**73.9**	**73**.7
Match-LSTM with Ans-Ptr (Boundary+Search+en)	150	882 K	**67.6**	**67.9**	**76.8**	**77.0**

Table 2.3: Experiment results from (Wang and Jiang, 2016) that highlight various configurations implemented. The best being Match-LSTM with Ans-Ptr using a boundary, search and ensemble technique.

Background research done for this thesis has shown that this approach hasn't yet been applied to the SQuAD 2.0 dataset. In chapter 3, section 3.2 a hypothesis is presented for the implementation of this approach along with the use of Transformers, which are covered in the next section.
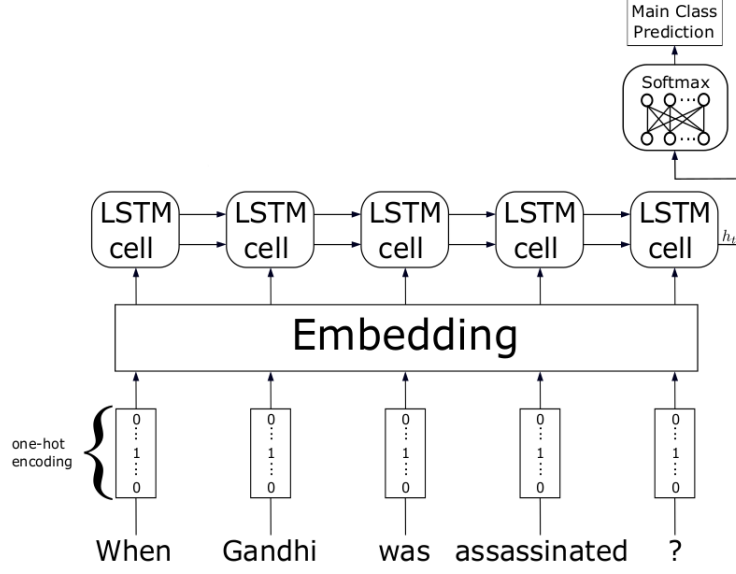
Figure 2.8: Basic classification model using LSTM (Gennaro et al., 2020)

The final piece of research to look at in this section is by (Gennaro et al., 2020) titled *Intent Classification in Question-Answering Using LSTM Architectures.* The approach followed is similar to (Brahma, 2018), using GloVe embeddings, allowing for prefixes to be added and generating word closeness and using a supervised learning technique. In fig. 2.8 it can be observed how applying one-hot encoding, followed by GloVe embedding before reaching the LSTM layer and using a Softmax activation layer at one prediction classes allows the model to generate better predictions on the TREC (Dietz et al., 2017) dataset.

In fig. 2.9 one can see the implementation of a padding layer at the end of each question and adding a subclass prediction strategy at the end of each prediction. This is done to gain better contextual understanding of the question. The subclass acts as a specialization class while the main class acts as a generalization class, therefore narrowing the context while maintaining global outlook. To verify their results (Gennaro
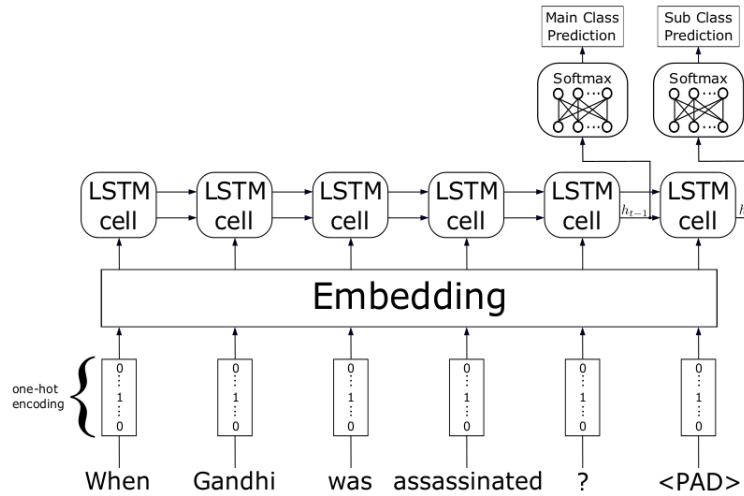

Figure 2.9: Basic classification model using RNNs (Gennaro et al., 2020)

14

et al., 2020) implemented a prototype responder by adding a biLSTM on top of the LSTM layer seen in figs. 2.8 and 2.9. The biLSTM model uses the predictions from the previous layer, both main and subclass predictions, and combines them to form the final output.

2.1.3 Critical Issues in LSTMs

From the literature reviewed so far, it has become apparent that LSTMs generate a view of the world in a very sequential manner. While successful in managing banishing gradients and generating better, contextually relevant results, LSTMs or even biLSTMs aren't entirely correct, with none of the models achieving over 90% accuracy across various datasets consistently. LSTMs suffer from another fatal flaw, their sequential nature makes it extremely difficult to parallelize the work and this leads to extended training times, out of memory errors etc.

2.2 Transformers

The previous section highlighted some key advancements in the field of machine comprehension and LSTMs, however, not a lot changed for almost two decades in the field of NLP based machine comprehension tasks architecturally. Since their introduction in 1997, LSTMs (Schmidhuber and Hochreiter, 1997) have been the base architecture for countless applications. However, as noted in section 2.1.3, LSTMs have some critical drawbacks.

Transformers were introduced in late 2017 by (Vaswani et al., 2017) and have become a very reliable way to implement various NLP based tasks. The basic architecture for which can be seen in fig. 2.10. Recurrent models usually adopt a combination ap-
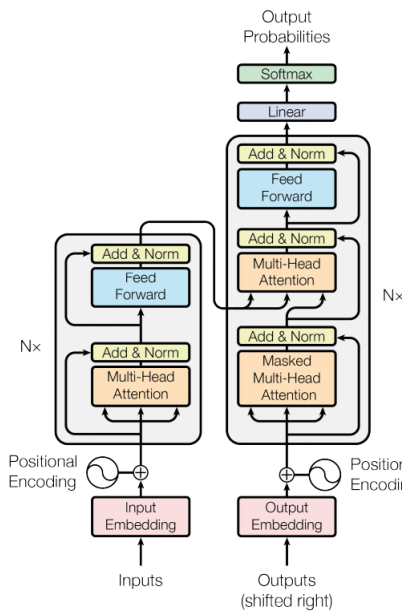

Figure 2.10: Transformer Architecture built by (Vaswani et al., 2017)

proach where they take symbol position of the input and output positions along with the computation. This helps them align positions to steps in computation time and

generate a sequence of hidden states h_t, which is a function of the previous hidden state h_{t-1} and the input function t. This inherently sequential nature of RNNs causes memory issues, leading to reduced batch sizes.

The architecture for a *Transformer* in this paper is outlined as having an encoder that maps input sequences to a continuous representation. The architecture can be seen in Fig. 2.10. This is then decoded into an output sequence of symbols one at a time. Each step is auto-regressive, ie. it consumes the previously generated symbols as additional input when creating the next. This is similar to an ensemble model. Stacks of 6 encoder and 6 decoder layers is used.

Each encoder layer has 2 sub-layers of a multi-head self-attention and the other a simple, position-wise fully connected feed-forward network layer. The output from each encoder layer can be represented as $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the function implemented by the sub-layers in the model. Additionally, supporting residual connections is handled by producing outputs of dimension $d_{model} = 512$.

The decoder layer is similar to the encoder layer and has an additional 3rd sub-layer that performs multi-head attention over the output of the encoders. To ensure that the output layer isn't affected by the output of residual connections from the sub-layers, a normalization layer is implemented at the end. Masking the subsequent positions and offsetting the output by 1 position ensures that the predictions for position i can only depend on outputs from previous positions.

The *Transformer* architecture was the first one to implement a self-attention based transduction model to compute input and output representations without the use of convolution or sequence-aligned networks.

One of the key features in a Transformer architectures is *self-attention*, which is covered in-depth in section 2.2.1.

2.2.1   Self Attention

In their paper "Attention Is All You Need", (Vaswani et al., 2017) highlight that *self-attention* is a key piece of the puzzle. This is due to several reasons such as the ability to parallelize computation, reduce total complexity per layer and the reduced path lengths between various long-range dependencies in the network. It is easier to learn about these dependencies if the distance between them is shorter. Thus, the transformer also computes the length of the longest path between any two input-output positions.

The attention mechanism was designed by (Bahdanau et al., 2014) to solve a critical issue with encoder-decoder architecture dependent LSTM and RNN based models. As has been highlighted before, LSTMs suffer from an inability to retain long-range sequence dependencies. Using a (soft)-search approach allows the model to look for parts that are relevant to predicting an output sequence, rather than forming all the segments of a sentence produces better results than existing fixed-length encoder-decoder models. The (soft)-search is performed after a prediction is made. This prediction is based on a set of positions for the source sentence, where the target word has the highest concentration of context vectors associated with previously generated target words. Another distinguishing feature of the attention mechanism is that it does not try to encode an entire input sentence to a fixed-length vector. Rather, it encodes the input sentence into sequence vectors that can be adaptively decoded during translation. Thus freeing the model from having to squash a lot of information

from the source sentence into a vector of fixed-length.

The work done in (Vaswani et al., 2017) modified that attention mechanism by mapping a query to a set of key-value pairs and developing *self-attention*. To understand self attention take an example sentence like:

"The lion was resting in the shade because it was tired after hunting."

While this is a perfectly normal sentence to a human being, a computer does not know what "it" means and that "it" refers to the "lion" in the second half of this sentence. Self- attention allows a Transformer to understand that the word "it" refers to the "lion" in this context. In fig. 2.11 shows how (Vaswani et al., 2017) modified the attention mechanism.
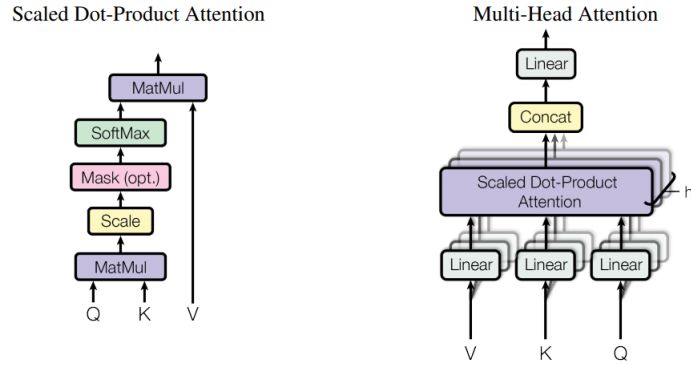

Figure 2.11: Scale Dot and Multi-Head Attention Models (Vaswani et al., 2017)

The "Scaled Dot-Product Attention" layer implemented in the transformer works by computing the dot products of all the input query with all the keys, divide it by the dimension of the keys and finally apply a softmax function to obtain the weights. This can be represented as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.14}$$

where K, V are the matrices containing the keys and values respectively, and Q is the set of queries. The dimension of the keys is given by d_k. The dot product is used because it is more efficient than the additive attention product and is faster as well (Vaswani et al., 2017).

Instead of using single attention, the transformer implements *multi-head* attention. Multi-head attention, which can be seen in fig. 2.11 on the right, allows for the model to simultaneously gather information from various different representations in different subspaces and positions.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$
$$\text{where head}_i = \text{ Attention } \left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{2.15}$$

Matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{mod 1}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$ represent the projection parameters. They also implemented $h = 8$ parallel attention/head layers. The reduced dimensionality of each head with $d_k = d_v = d_{\text{model}}/h =$

64 means that the total cost is similar to a fully connected, single-head attention layer. The evaluations performed on the Wall Street Journal dataset(Marcus et al., 1993), using 40k sentences, showed that even without task-specific tuning the model had better results with a fraction of the training cost.

The advantages of the Transformer architecture are clearly evident over the standard RNN based LSTMs and other architectures. Experiments on the WMT 2014 English to German translation (Bojar et al., 2014) using the big transformer showed a significant jump in BLEU score 28.4 over various other seq2seq models and even combined ensemble models which earlier had a highest score of 26.36 using a ConvS2S Ensemble (Gehring et al., 2017).

2.2.2 Improvements of Transformer Architectures

Transformers have significant advantages as highlighted earlier. Improvements to the basic architecture are discussed below.

BERT

The paper on *Bidirectional Encoder Representations from Transformers* (BERT) (Devlin et al., 2018), introduces a new language model. This model is truly fascinating in many ways. First and foremost it is designed to pre-train deep bidirectional representations using unlabelled data. This is done by jointly conditioning context in all layers to the right and left. This pre-training allows the model to be fine-tuned simply using one additional output layer. These features make this model conceptually simple and very powerful empirically.

BERT employs language pre-training (Dai and Le, 2015, Peters et al., 2018, Radford et al., 2018) which has shown significant advantages in many applications e.g. paraphrasing, language level inference etc. These tasks aim to highlight the relationships between sentences through contextual understanding as well as by using tokenized outputs. BERT improves on the fine-tuning tasks that are important to performing
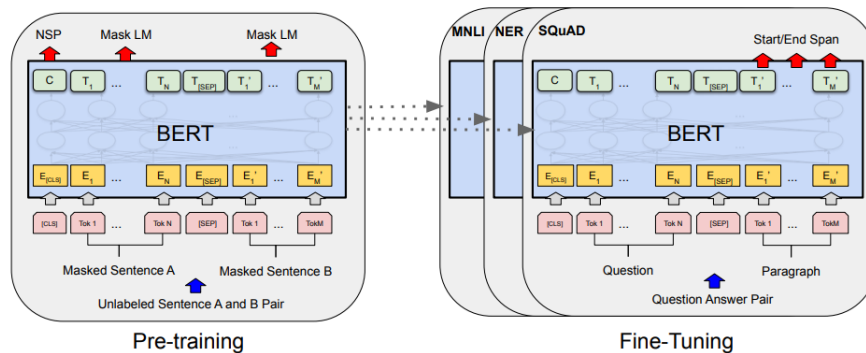

Figure 2.12: Pre-training and Fine Tuning procedures for BERT (Devlin et al., 2018)

BERT was tested on The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) which has a large number of diverse NLU tasks. BERT performed extremely well on the 11 NLP tasks that it ran against. It showed an average accuracy improvement of 4.5% and 7% when compared to the previous state of the art

models. Results of BERT and its significant gains make it one of the best candidate models for NLU tasks.

DistilBERT

The DistilBERT model by (Sanh et al., 2019) was created with a few factors in mind such as model size, complexity, ability to generalize tasks and the high environmental cost of running larger models. Models like RoBerta (Liu et al., 2019), BERT (Devlin et al., 2018) etc. have several hundred million model parameters that take particularly longer to train, even through use of advanced pre-training techniques.

Larger models often tend to exponential increase model computational requirements (Schwartz et al., 2019), these models also fail to generalize. The DistilBERT model shows that by applying a triple loss function on top of a smaller Transformer (Vaswani et al., 2017) that is trained using a knowledge distillation compression technique (Bucilu et al., 2006) can provide similar results in 60% less time.

The work proposes a *"Student-Teacher"* architecture. The DistilBERT model, which is the student architecture, has the same kind of layout as the BERT model but without the tokenized embeddings and the pooler architecture, with a further reduction in the number of layers.

Experiments showed that training a DistilBERT model produced similar results to BERT-base model but in a lot less time than its parent architecture BERT. The model is 0.6% behind BERT on the IMDB and only 3.9% behind BERT on the SQuAD 1 dataset.

These speed and accuracy advantages make this model and architecure worthy of being explored further and will serve as the basis of the improvements suggested in this thesis.

Chapter 3

Research Methodology

3.1 Data Selection

The Stanford Question Answering Dataset (SQuAD) 2.0 has been chosen as the dataset for the experiments proposed. This is as a reading comprehension dataset based on Wikipedia articles. It is based on questions posed by crowd-workers on a set of articles. The answer to every question is a segment of text or span, from the corresponding reading passage, or the question might be unanswerable (Rajpurkar et al., 2018).

The dataset consists of over 150,000 questions. Split into 100,000 answerable and 50,000+ unanswerable question, which were written to look similar to unanswerable questions. The challenge being that a model should be able to correctly answer the answerable questions and abstain from answering the unanswerable ones. The dataset is freely available as a part of the Transformers package in python or it can be downloaded from the SQuAD 2.0 website (Rajpurkar, 2021).

To effectively use this dataset it is important to first take a look at what its contents look like below.

Context: *"The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from Denmark, Iceland and Norway who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia."*
Question: *Who was the Norse leader?*
Answer: *Rollo*

The answer to the aforementioned question is quite simple for humans to comprehend. The challenge is for computational systems is to contextualize this and make it machine-understandable so that the produced model can answer it correctly. The dataset consists of various kinds of English language examples like negation, antonyms, entity swaps, impossible conditions to answer, answerable, etc. making the dataset a well-balanced one. It is important to prepare this dataset appropriately, this will be done

1. Data splitting into separate Question, Answer and Context lists for use for both the training data and the validation data

2. Tokenization of the split data to generate "context-question" pairs

3. Generating indexes for when an answer begins and ends in the dataset

4. Adding answer and context tokens based on their encoded positions

The SQuAD 2.0 Dataset (Rajpurkar et al., 2018), was developed with funding from Facebook to help address some major issues with existing datasets. Most datasets focus on questions that can be easily answered or use of automatically generated, unanswerable questions which are easily identifiable.
The SQuAD 2.0 dataset resolves this by combining the SQuAD dataset along with 50,000 crowd worker generated unanswerable questions. The key feature of these being that the unanswerable questions must look similar to answerable ones. For a model to be successful on this new dataset, it must be able to answer all possibly answerable questions as well as determine when no answers are provided for a question in the given paragraph and abstain from answering. A comparative study was done for a Natural Language Understanding(NLU) task that obtained an 86% score on SQuAD 1.1, only got 66% on the new 2.0 dataset. The dataset helps bridge the gap between true NLU and machine understanding by using the concept of Relevance. Through comparisons with various datasets such as RACE, MCTest, QASENT etc. they have identified the missing links like negative examples, antonyms and helped fill the gap. This dataset forces the models to understand whether a paragraph span has the answer to the question posed.

	SQuAD 1.1	SQuAD 2.0
Train		
Total Examples	87,599	130,319
Negative Examples	0	43,498
Total articles	442	442
Articles with negatives	0	285
Development		
Total Examples	10,570	11,873
Negative Examples	0	5,945
Total articles	48	35
Articles with negatives	0	35
Test		
Total Examples	9,533	8,862
Negative Examples	0	4,332
Total articles	46	28
Articles with negatives	0	28

Table 3.1: Comparison of SQuAD 2.0 to SQuAD 1.1(Rajpurkar et al., 2018).

Table 3.1 shows the difference between SQuAD 1.1 and SQuAD 2.0 datasets.

3.2 Research Hypothesis

The hypothesis of this research is that implementing a self attention based Transformer model with:

- Glove embeddings

- BERT base (Devlin et al., 2018) type Transformer model

- Answer pointer output layer(Wang and Jiang, 2016, Vinyals et al., 2015)

Will help improve the prediction capability of the transformer architecture.

The DistilBERT model (Sanh et al., 2019), which is a lighter, distilled version of the BERT model, and has 40% less parameters at 66 million, instead of 110 million in BERT, as discussed in (Devlin et al., 2018), will be customized with the implementation of the answer pointer layer to generate answers for the questions posed.

3.3   Model Evaluation Metrics

To successfully evaluate the model two metrics will be used. The *F1 - Score* and *Exact Match Score* will be used to measure the success of the model developed. While accuracy is a good score to measure how well a model is performing, the problem with accuracy is that it is prone to overfitting, and underfitting, which might lead to great accuracy scores but real world predictions wont be as accurate.

Accuracy also doesn't work because it is a simplistic probability metric that assumes both false positives and false negatives have an equal weight in predicting the outcome as shown in eq. 3.1.

$$Accuracy = \frac{TruePositive + TrueNegative}{TruePositive + FalsePositive + TrueNegative + FalseNegative} \quad (3.1)$$

Outlined below are some reasons why accuracy is not a good measure:

1. **Class Imbalance**: Class imbalance refers to a situation when the data contains a large number of values that belong to one particular category. This means that the data is skewed in favour of that category and if this data is used, it can lead to results that positively favour the results from that category, thus leading to inaccurate predictions. Since accuracy assigns equal weights to both false positives and false negatives, in case of critical predictions using classifiers, skewed data can lead to disastrous consequences.

2. **Prone to Over and Under-fitting**: Overfitting happens when you model "specializes" on the training dataset so well that it suffers from a poor performance on the testing or validation sets, thus having poor real world performance. Underfitting happens when the model "generalizes" extremely well on the training dataset because it contains lots of different scenarios with overlapping outcomes, thus failing to specialize on a particular context, again leading to poor validation or real world performance. Since the output from the model is skewed in its training regime, its accuracy in either case is very poor on real world applications, thus making accuracy not a good measure.

Having established that accuracy is not a good measure. It is important to understand the following metrics that form the basis for F1 and Exact Match. Precision and Recall are highly used metrics that provide a deeper insights into the dataset and model prediction results. The F1 score helps balance these metrics by combining them into one single metric.

Precision is the probability of identifying a correct positive outcome from all the predicted positive outcomes. Precision is used for cases when the cost of a false positive has a very high negative impact.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{3.2}$$

Recall is defined as the probability of a correctly identifying a positive outcome from all the positive outcomes. Recall is very important in cases where the cost of negative responses or false negatives is high.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{3.3}$$

F1 Score is the harmonic mean of recall and precision. It is used as an evaluation metric in places where the impact of both false positives and negatives is very high.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.4}$$

Exact Match is a binary metric, i.e. it is 1 if the predicted answer matches the expected answer and 0 if it doesn't.

3.4   Research Flow Diagram

To be able to achieve the objectives set out in the previous chapters, it is important to organize the research in such a way that it achieves its objective. This is represented in the research flow diagram 3.1.

Data Preprocessing

- Obtain training and validation datasets
- Split both datasets into questions, answers and context lists

Data Preperation

- Add index positions to both training and validation contexts and answers
- Embed token positions into both sets
- Add dataset lengths to both embeded and tokenized training and validation sets

Benchmarks

- Run prepared dataset through ALBERT model and obtain results
- Run prepared dataset through RoBerta model and obtain results
- Run prepared dataset through DistilBERT model and obtain results

Custom Model Preperation and Evaluation

- Prepare Pointer network with 1 LSTM that has 1 input layer, 8 hidden layers and 1 output layer
- Prepare Pointer Encoders and Decoders
- Prepare DistilBERT based Transformer model
- Modify transformer model with implementation of pointer layer

Results

- Collect results
- Draw conclusions from the results
- Propose future improvements & applications

Figure 3.1: Research Flow Diagram

Chapter 4

Architecture Creation

The literature reviewed in chapter 2 highlights some exceptional work that has been done in the field of NLP, LSTM and Transformer architectures. This chapter lays the groundwork for the improvements proposed in chapter 3. Starting off with a review of the drawbacks of the architectures that have been chosen for benchmarks and comparisons, then the improvements are proposed to a BERT based DistilBERT architecture are mentioned in section 4.2.

4.1 Drawbacks Of Current Architectures

The transformer based architectures reviewed in chapter 2 have shown great advantages. However, they suffer from certain flaws that are outlined below:

- **Fixed Length Strings** - Transformers apply attention mechanism that need the text to be split into smaller, sizeable chunks to be processed.

- **Context Fragmentation** - Since input text is split into smaller chunks, context is often lost because if an input sentence is split down the middle its meaning changes completely.

- **Abstraction of Higher Level Representations Fragmentation** - Each layer in a transformer adds a layer of abstraction and processes them in parallel for a given input sequence. This causes it to lose out on knowledge from previous layers during current computation, leading to recalculation even though it has already been calculated.

- **Feed-Forward Belief State** - Transformers apply an internal belief state that is quickly accessible to the future layers, however, this information isn't accessible to the previous layers. This means that only a limited number of transformations can be applied to the input before it is produced as output. This is also limited by the depth of the model. (Fan et al., 2020)

To address a few of these concerns some improvements are advised in the next section.

4.2 Proposed Architecture Improvements

The proposed architecture is an implementation of the transformer architecture along with a biLSTM based Answer-Pointer network.

The DistilBERT model that is available through the Hugging Face Library (HuggingFace, 2021), will be modified with the implementation of an Answer-Pointer network. The hypothesis is that by passing the results from Answer-Pointer layer to a DistilBERT based architecture will provide better results since it will already have been encoded with output pointer positions.

To achieve this the Answer-Pointer based network is designed initially with 1 LSTM cell containing 8 hidden layers and one output layer. This is described in detail section 4.2.1.

The process begins by serving tokenized inputs that are created as a part of pre-processing the input data and embedding it by using the "distilbert-base-uncased" pre-trained embeddings from Hugging Face libarary. Then the input is then fed to the "DistilBertForQuestionAnsweringC" class, which is the modified class from the Hugging Face Transformers library mentioned before.

The DistilBertForQuestionAnsweringC class takes a pre-trained model config file as input along with the embedded inputs, passes them through a linear layer and a dropout layer. These processed outputs are then processesed through Encoder and Decoder layers that allow the model to process and produce relevant results.

4.2.1 BERT Optimized LSTM Transformer, BOLT

The BOLT model begins with the implementation of a Pointer network that feeds the output to the DistilBERT implementation. To better understand the flow of this model a flow diagram is presented in 4.1.

The Answer-Pointer network consists of 4 main parts, an Encoder layer, an Attention layer, a biLSTM layer and a Decoder layer. The implementation of this is done using PyTorch and inspired by work from (Gur, 2017). This was extremely helpful as Pointer networks aren't readily available from libraries.

The network begins with taking an input sequence of embeddings and passing it through a linear layer that takes input and output dimensions. The embedded inputs are then passed through the Encoder layer.

The Encoder layer consists of LSTMs with dropout layers. The LSTM layers are doubled if implemented in biLSTM mode. The dropuout layer helps reset the input dimensions to 0 before taking the next set of inputs, this helps in preventing overfitting. The dropout layer also controls the rate at which the values are set to 0 and is a value between 0-1, with a default of 0.5. Essentially, the dropout layer retains only a small percentage of the outputs from a hidden layer, thus preventing an already large model from gaining more size.

The LSTM layer processes the input embeddings, encodes them further appropriately and produces encoded outputs for the decoder layer. The Decoder layer is a comprised of several linear layers and an attention layer. The linear layers serve as both input, hidden and output layers, however, the hidden layers are also connected to attention layers that help localize the context of the word in the input question and context.
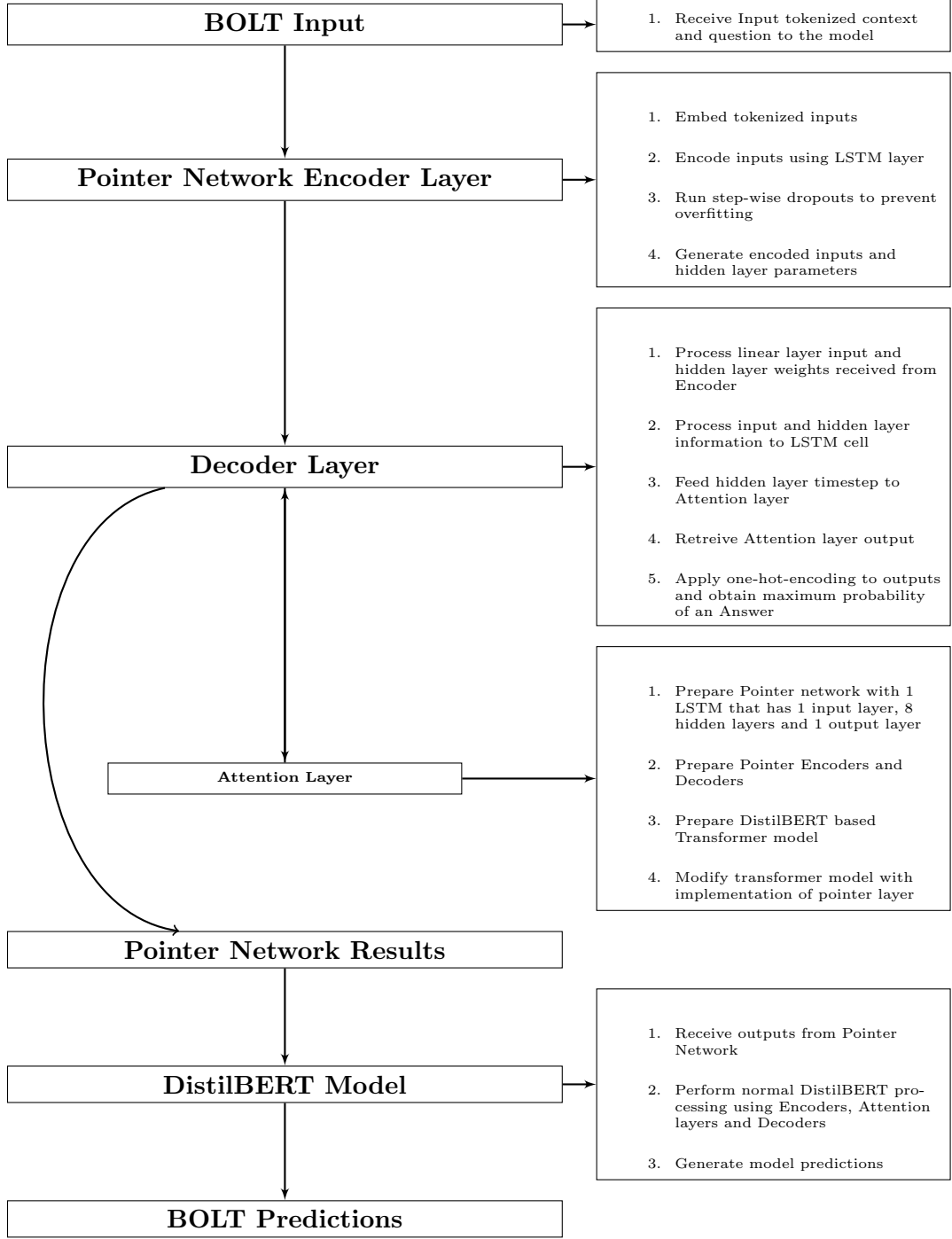
Figure 4.1: BOLT Data Flow Diagram

The decoder layer generates an output using an internal LSTM layer, thus providing a tuple of output and pointer values plus a separate output of hidden layer weights. These outputs and pointer values are then fed back into the linear output layer of the DistilBERT model.

The BOLT model uses the existing implementation of the DistilBERT model from the Hugging Face library (HuggingFace, 2021) and modifies its base class with an implementation of the Answer Pointer layer mentioned earlier in this chapter.

The output from that layer is then fed through to the DistilBERT based transformer where it takens the outputs from the Pointer network as input embeddings, applies an attention mask in case the input sequence is smaller in length than the current batch, before finally producing a set of output sequence tokens that serve as pointers to identify which words are the predicted outputs of the model. The architecture of the DistilBERT model is not modified but used as an extension to the pointer network. This approach allows for the highly optimized DistilBERT model implementation to be retained in its purest form, removing the chances of miscalculated layers and producing incorrect outputs.

Chapter 5

Results

To establish the improvements hypothesized in the previous section it is important to have a baseline metric. The ALBERT (Lan et al., 2019), RoBERTa (Liu et al., 2019) and DistilBERT (Sanh et al., 2019) models are available pre-trained through the HuggingFace library (HuggingFace, 2021). The library also provides the SQuAD 2.0 Dataset (Rajpurkar et al., 2018) for use. The dataset is also available for download directly and to aid understanding of the data correctly that approach is implemented.

The code used to prepare the datasets as well as the results gathered are available as examples from the HuggingFace library documentation, some changes have been mde to them to accomodate upgrades from different dependencies.

For the sake of consistency of the results presented, the experiments were conducted using a Google Colab Pro account with Tesla series GPUs and 16GB of VRAM. Google Colab hosts GPUs in the cloud environment, which impacts performance but is the best alternative to investing and buying a GPU given the silicon shortage and exhorbitant prices in the current market.

5.1 Experiment Data Pre-Processing And Analysis

Before performing the experiments on the custom BERT models designed, the SQuAD 2.0 dataset needs to be prepared for use. As outlined chapter 3, the dataset consists of over 130 thousand questions in the training set. Below are some of the steps taken to prepare the dataset and some details from the same:

- **Splitting Dataset**: The dataset consists of 2 files "dev-v2.0.json" and "train-v2.0.json", which serve as validation and training sets. These consist of questions, relevant contexts and answers, respectively. The dataset is prepared by extracting these into separate data structures(lists) for use.

- **Training Context**: Upon reading and splitting the data into training contexts, questions and answers, it can be seen that there are 86,821 contexts provided in the training set and 20,302 contexts in the validation set.

- **Indexing Ends**: The training and validation datasets are then embedded with index position that help identify the answer to the questions in the context layers.

- **Tokenization**: The training and validation datasets are then embedded with token positions that are obtained from the pre-trained DistilBertTokenizer available from the Hugging Face library.

(a) ALBERT Validation F1 Score

(b) ALBERT Validation Exact Match

(c) ALBERT Validation Loss

(d) ALBERT Training Loss
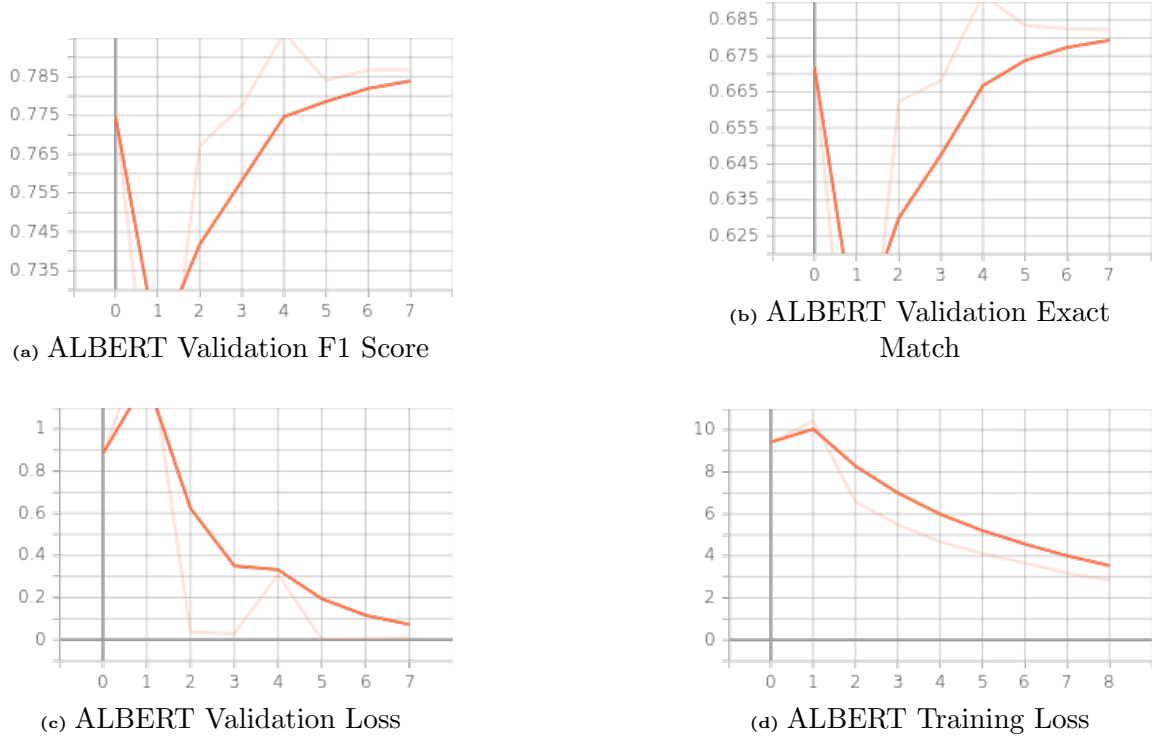
Figure 5.1: ALBERT Model Benchmarks

- **Training and Validation Datasets**: The encoded datasets are then modified to contain the length of each set as that is required by the modified architecture as well as the base architectures.

5.2   Existing Architecture Benchmarks

Each of the three selected models is put through the same process of data preprocessing, as identified in chapter 3, this is to ensure consistency across board with all the model types and even the custom model improvements advised.

The experiments begin with a review of the ALBERT architecture (Lan et al., 2019), where it is observed that at a batch size of 8 with the AdamW optimizer set to a learning rate of 5e-5, the average epoch time for training takes around 1 hour, 24 minutes at 2.15 iterations/second. Further, the vaidation set takes around 7-12 minutes, depending on the availability of the GPU in the cloud based Google Colab Pro environment.

Figure 5.1 shows the various measured parameters across 10 epochs for the ALBERT model. It can be seen from 5.1a that for the "albert-base-uncased" pre-trained model class the ALBERT model still suffers from a significant drop in F1 scores in the initial epochs due to the learning rate being set to a very low value of 5e-5. This behaviour is further confirmed by the 5.1d and 5.1c graphs where the graph peaks to a higher value for both kinds of losses before tapering down over the remaining epochs.

This sudden rise in training and validation loss values that causes poor F1 and exact match scores is purely because of the learning rate. It is seems as though the learning rate is not appropriate for the batch size of 8, however, for the sake of all the experiments to be baseline it will be observed as such.
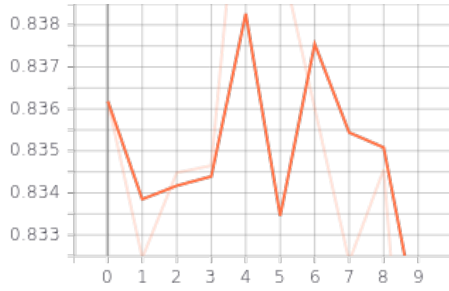
(a) RoBerta Validation F1 Score
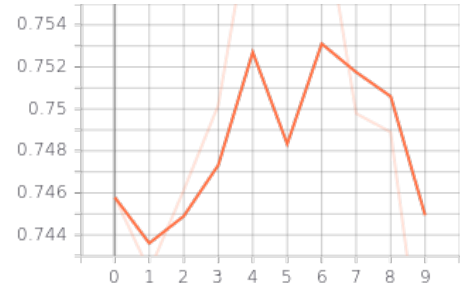
(b) RoBerta Validation Exact Match
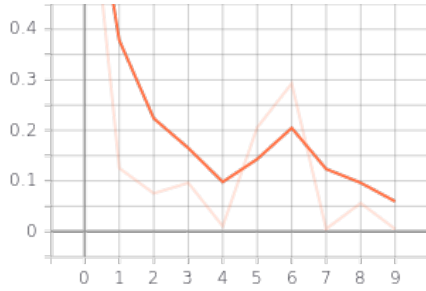
(c) RoBerta Validation Loss
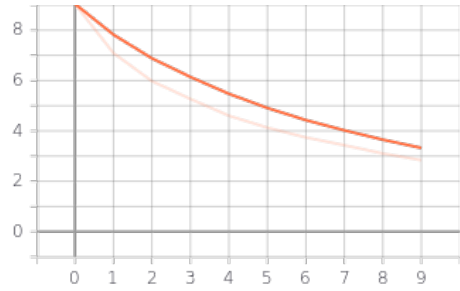
(d) RoBerta Training Loss

Figure 5.2: RoBerta Model Benchmarks

One interesting observation for the ALBERT benchmarks is that the training loss shown in 5.1d shows a consistent, smooth decline while training the model.

The final benchmark scores for the ALBERT model at epoch 10 are:

- **F1 Score**: 77.5%

- **Exact Match Score**: 67.5%

- **Training Loss**: ∼3.7%

- **Validation Loss**: ∼0.04%

Figure 5.2 shows the experimental results for the RoBerta architecutre and model. Unlike the ALBERT benchmarks above, the RoBerta benchmark results seem very erratic in nature. The validation F1 and exact match scores for the RoBerta model, while performing significantly better than ALBERT, remain around 83% and between 74.4 - 75.4% respectively. This is a roughly 10% better performance than the ALBERT model, although the training time for the model is around 2 hours per epoch at 1.21 iterations/second. The validation epoch takes around 21 minutes to complete at 4.06 iterations/second.

This increased training time can be attributed to the almost double number of parameters in the model at 125 million, compared to 66 million in ALBERT when using the "roberta-base" pre-trained model.

Similar to the ALBERT model, a consistent and smooth decline in training loss can be observed, as shown in 5.2d. The RoBerta model is run using the "roberta-base" pre-trained class that is similar to the "albert-base-uncased" class, just optimized by the HuggingFace Team for this model.

The final benchmark scores for the RoBerta model at epoch 10 are:

- **F1 Score**: ∼83.3%

(a) DistilBERT Validation F1 Score

(b) DistilBERT Validation Exact Match

(c) DistilBERT Validation Loss

(d) DistilBERT Training Loss
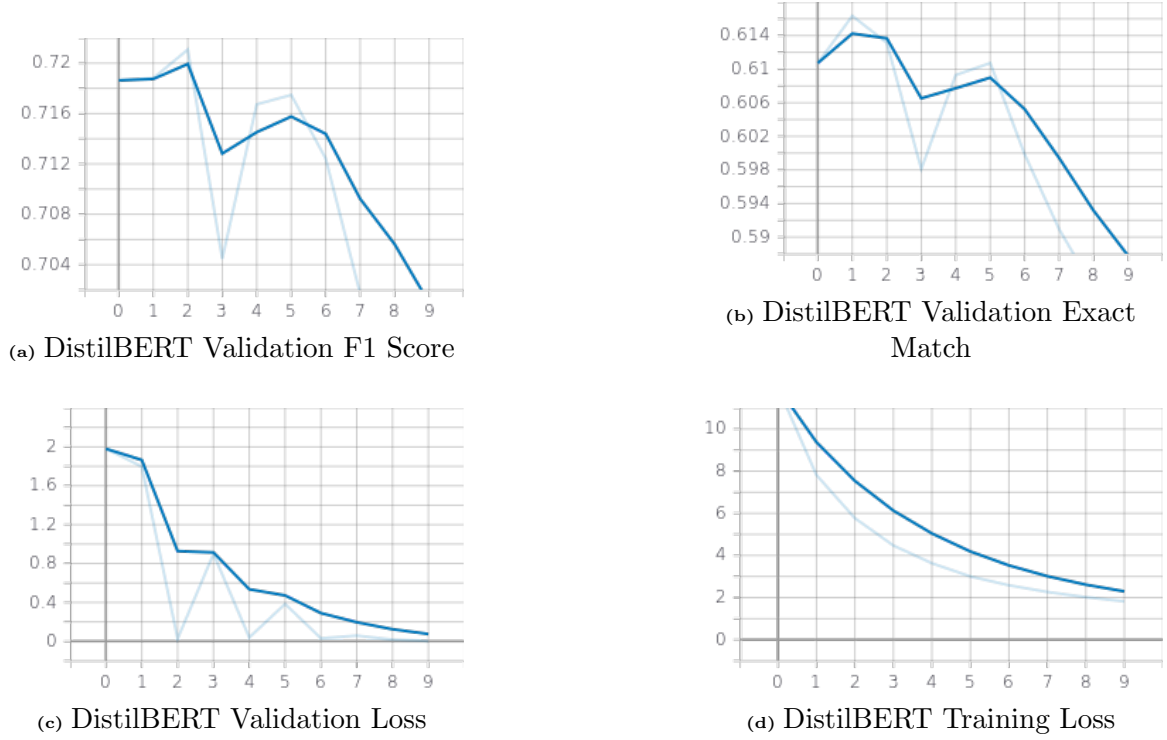
Figure 5.3: DistilBERT Model Benchmarks

- **Exact Match Score**: 74.6%

- **Training Loss**: ∼3.7%

- **Validation Loss**: ∼0.5%

In figure 5.3, the behaviour of the model can be observed to be very different from the other models. The behaviour can be described simply as a combination of the behaviours seen for ALBERT and RoBerta models. Just like in 5.2d, the training loss for the DistilBERT model as seen in 5.3d is extremely consistent and smooth in nature, showing a steady decline till the final epoch. training loss and validation loss, fig. 5.3c are lower compared to both models above, the validation F1 and exact match scores are not exactly comparable.

Using the same training criteria of a batch size of 8 and an AdamW optimizer set to a learning rate of 5e-5, it can be seen that the DistilBERT model with performs slightly poorly when compared to ALBERT and RoBerta. However, the time taken to train a DistilBERT model is significanty less than that of RoBerta and ALBERT. One epoch for a DistilBERT model takes approximately 37 minutes to complete, almost half that of ALBERT and 3x less than RoBerta. The validation set takes around 5 minutes as well.

The final benchmark scores for the DistilBERT model at epoch 10 are:

- **F1 Score**: ∼70.4%

- **Exact Match Score**: ∼58.5%

- **Training Loss**: ∼2.1%

- **Validation Loss**: ∼0.1%

While the results for the DistilBERT model aren't exactly perfect, it does allow for quick prototyping and achieves almost near accurate results.

5.3   BOLT Results

As discussed in chapter 4, the modifications have been applied to the DistilBERT model, specifically the DistilBertModel class offered by the HuggingFace Transformers library (HuggingFace, 2021) and a modified implemetation of an Answer Pointer Network designed using PyTorch by (Gur, 2017) is used to generate the answer pointer layer results for these custom models. All the custom models utilised the same pre-trained "DistilBERT-base-uncased" class from the Hugging Face Transformers library.
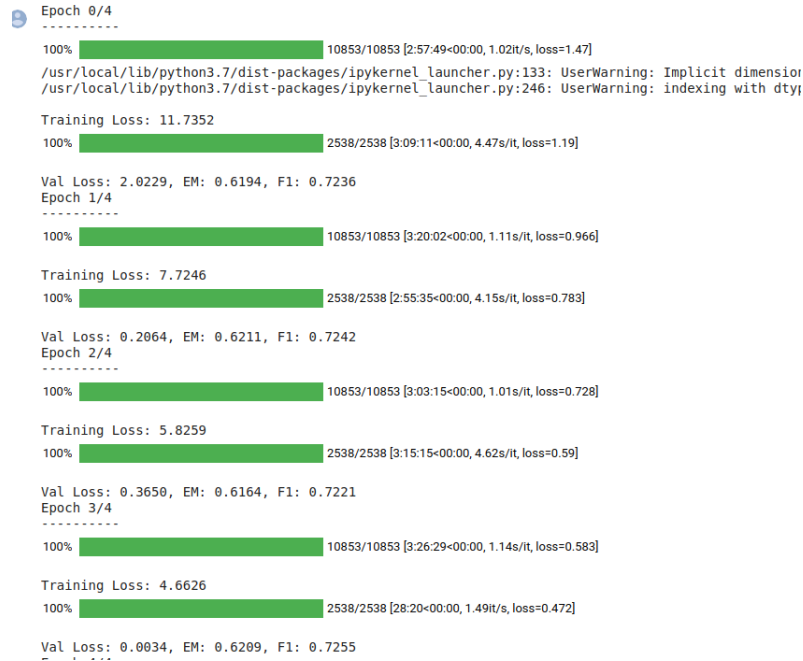
Figure 5.4: BOLT with AdamW Optimizer at 5e-5 rate

The first figure 5.4 shows the various epochs for the modified DistilBERT model with the Answer Pointer network implemented with a batch size of 8 and the AdamW optimizer with a learning rate of 5e-5, which are the same parameters as the benchmark models.

In figure 5.4 it can be observed that the model takes significantly longer to train than a regular DistilBERT model which takes around 37 minutes. Though the training and validation times are higher than existing models, the training loss values track consistently down and are comparable to the ones seen in 5.3d. The increased training and validation set times can however be less than the ones shown as there is a significant decrease in the validation set running time towards the 5th epoch. This is quite peculiar and goes to highlight the distributed nature of the cloud based training environment that is Google Colab.

The F1 scores for the model continuously hover around the 72% mark, which is a slight improvement over the 70% mark observed for the benchmarked DistilBERT model as seen in 5.3a. The validation loss for the custom model hovers lower than the values in the DistilBERT model as well.

In the next experiment, shown in fig. 5.5, a different learning rate is used with the same AdamW optimizer. Using a learning rate of 3e-5 or 0.00003 there is a slight increase in F1 and exact match scores, reaching almost 75% and 65% values respectively. This suggests that the custom model responds well to lower learning rates.

The training times remain roughly the same with a sudden drop in the validation time seen again towards the end.
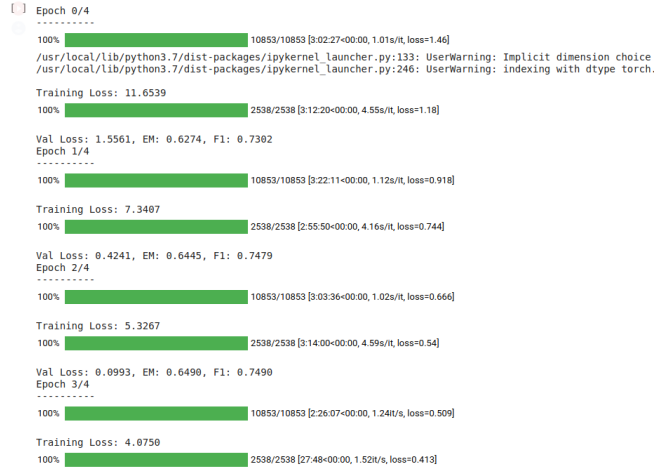
Figure 5.5: BOLT with AdamW Optimizer at 3e-5 rate
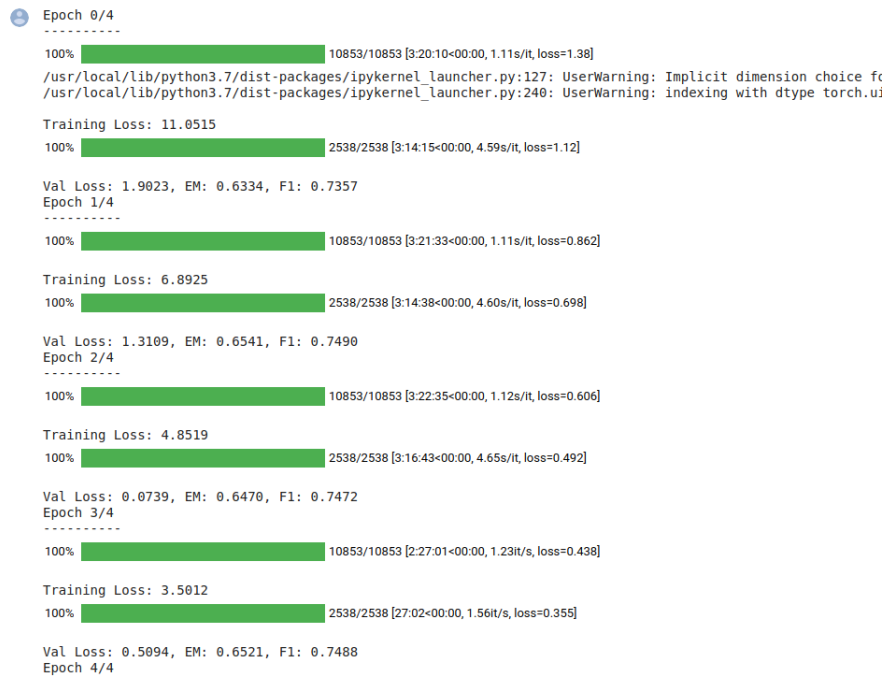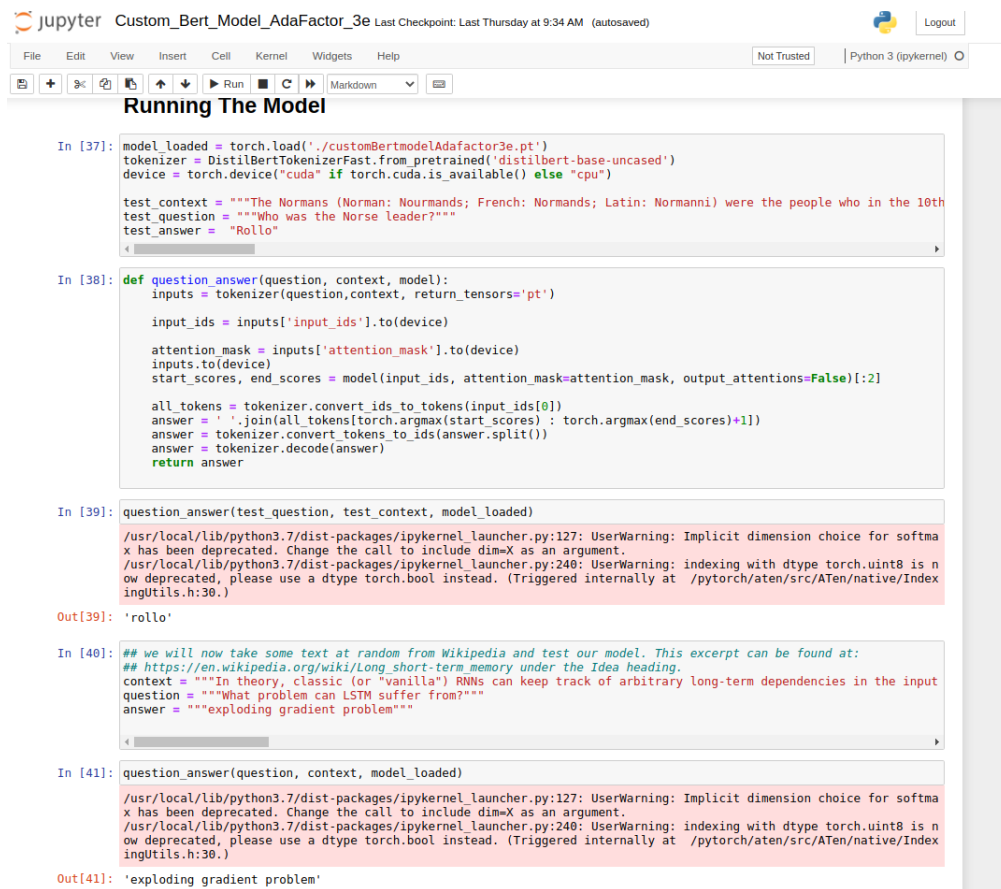
Figure 5.6: BOLT with AdaFactor Optimizer at 3e-5 rate

In the final experiment, that is shown in fig. 5.6, the custom model is run with a modified AdaFactor optimizer with a learning rate of 3e-5, this again presents similar results to the results in the previous experiment where the lower value of the learning rate helped the custom model learn better. The training loss for this run also falls similar to the loss in the previous custom model experiments as well as the DistilBERT

benchmark, while maintaing an F1 score that is higher than the DistilBERT benchmark but roughly similar to the custom model with AdamW optimizer at a learning rate of 3e-5.

Figure 5.7 shows the DistilBERT custom model producing correct answers to a few question asked of it. This confirms that the model is able to produce answers to some questions and can give accurate results as well. The custom model is more accurate than its base DistilBERT model, who's output can be seen in figure 5.8. While the DistilBERT model answers the first 2 questions correctly, it fails to answer the final question, "What problem can LSTM suffer from?", with 100% accuracy and outputs "vanishing gradient problem" instead of the correct answer "exploding gradient problem".

Figure 5.7: BOLT with AdaFactor Optimizer at 3e-5 rate outputs to questions asked

Figure 5.8: DistilBERT model outputs to questions asked

Chapter 6

Conclusions And Recommendations

Through this work it has been attempted to highlight the advancements and propose a unique modification to the existing transformer architecture using an Answer-Pointer network.

The results obtained from experiments in section 5.3 show that the models performance is at par with the DistilBERT model. While this result isn't graphically present due to timeout issues in Google Colab, visually the various scores printed in figures 5.4, 5.5 and 5.6, the model does perform well initially.

The experiments conducted in this thesis highlight that a properly implemented LSTM based Answer Pointer network layer that is incorporated with the Transformer architecture can perform at par with a current state of the art model. While the custom model presents an opportunity for improvement in terms of training times, the results and the metrics are at par with the benchmark models.

The benchmarked models are current state of the art approaches that are used industry wide for various applications and readily available via libraries like PyTorch, Hugging Face and TensorFlow. The model presented here has shown potential through the limited amount of testing and resources available.

It is highly probable that given the right resources and time to optimize the model, it will be able to perform better than the DistilBERT model at least.

6.1   Recommendations

This work provides a unique opportunity to introduce improvements to an already state of the art architecture by adding a layer of LSTM based pointers. Highlighted below are some recommendations that can improve the architecture further.

- **Training Checkpoints**: Checkpointing and training the models across various checkpoints can help run the model for more number of epochs by saving and resuming trainging from a particular checkpoint. This is especially critical in the case of using online resources such as Google Colab because they disconnect runtimes after 24 hours and wipe all the data.

- **Increased Batch-Size**: The experiments presented in this work are limited by a relatively small batch size of 8. Systemic increase of batch size can allow for reduced training times, possibly better performance overall and an opportunity to test this model against various other NLP applications such as Sequence Classification, Token Classification, Sentence Predictions, etc.

- **Better Hardware**: One of the key limiting factors for the work presented in this thesis is the limiting amount of VRAM, The current hardware used only had 8GB of VRAM available on the EVGA 2070 Super GPU, which was insufficient for the benchmark batch size of 8. Reducing the batch size any further led to increased training times per epoch, upwards of 3 hours per epoch at a batch size of 6. This problem was slightly remediated by the use of Google Colab Pro, which is a subscriptions service that allows you to run experiments using better GPUs such as Tesla P100 GPU with 16GB VRAM. This service is time limited and thus not suitable for running long experiments. To effectively quantify the proposed architecture in this thesis it is recommended that it be run for at least 10 epochs on a multi-GPU setup that allows for mini-batches. This approach is discussed below.

- **Mini-Batches Across Multiple GPUs**: Creating a multi-GPU setup, with 2 or more gpus, allows for accelerated training (Pal et al., 2019). A simple guide on how to set this up for PyTorch has been created on the Towards Data Science Medium page (Giacaglia, 2020). Having multiple smaller batches train across a multi-GPU setup can allow the model to not only train faster but also perfrom slightly better than just on a single GPU.

- **Increased LSTM Model Complexity**: One of the biggest challenges faced was increasing model complexity from implementing a single LSTM model with 8 hidden layers to anything more or even bidirectional in nature. The experiments conducted using more number of LSTM layers or hidden layers led to very high GPU memory usage, often causing out of memory errors, hence failing model training. If the above recommendations are provided then it might be possible to increase the LSTM model complexity which could yeild better F1 and Exact Match scores for predictions by the model.

Appendices

Chapter A
Gantt Chart
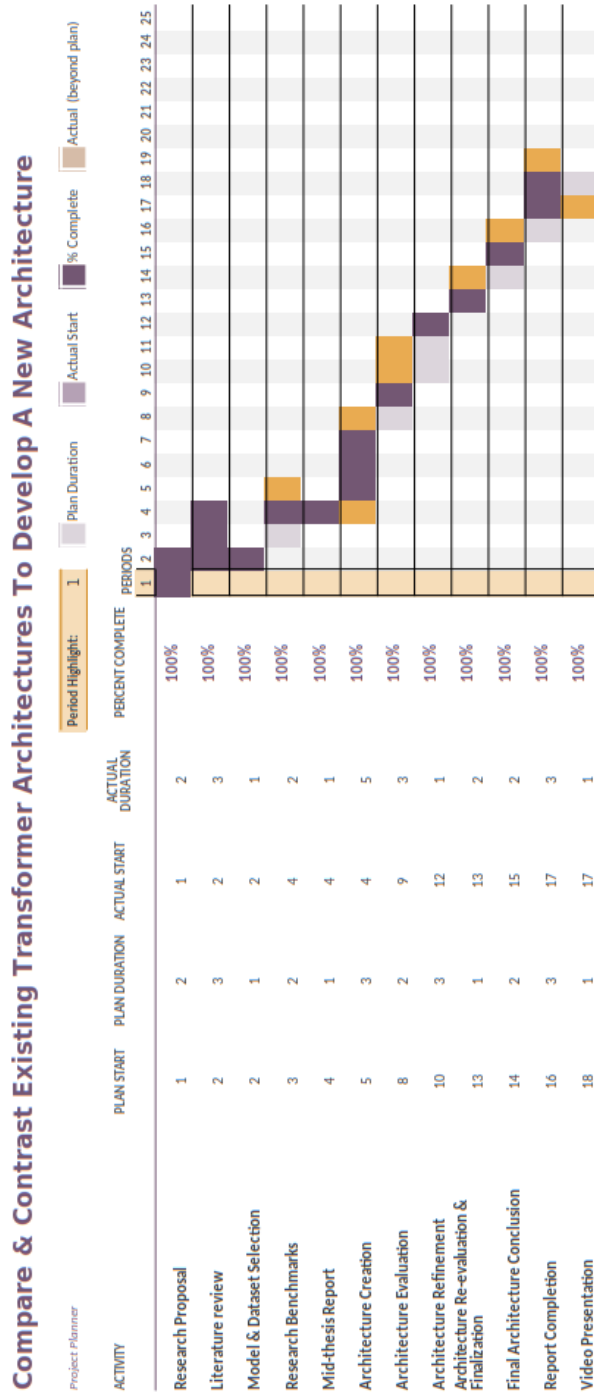
Figure A.1: Thesis Gantt Chart

Chapter B
Research Proposal

Research Proposal

Nirbhay P. Tandon
MSc. Data Science
Research Title:
Develop New Transformer Architecture For
Question and Answering(QandA)

Abstract

Attention-based Transformer architectures have become the norm of current Natural Language Processing applications. Google began this trend back in 2017 with their paper *Attention Is All You Need*, by introducing the Transformer architecture that works solely on attention mechanisms. The purpose of our work will be to create a new kind of Transformer architecture. Compare and contrast its performance against other architectures such as BERT, DistilBERT, ALBERT etc. using the SQuAD 2.0 Dataset. Through our research, we aim to produce a new architecture that has a better performance than existing models specifically for Question Answering based applications.

Contents

List of Figures

1 Introduction

Question-Answering based systems have gained a lot of popularity, especially in the form of "chatbots". These systems depend highly on contextual understanding of the input, the training corpus and the question asked. They use this knowledge to output an answer that can help the user with whatever their query is. Recurrent neural networks and architectures based on them, have been able to provide great advancements in the field of Question-answering and chatbots in general. However, there is a behaviour of over-fitting and lack of contextual understanding of the question. This, coupled with long training times and extremely complex mathematical model designs, have often kept the field of Natural Language Processing slightly obscured from the masses.

We wish to change that. Through our work, we would like to aim at creating a sustainable, fast, easy to understand Transformer architecture for Question Answering. An advancement on the work done by the team at Google (Vaswani et al., 2017). To be able to do so, let us first understand what a *Transformer* is. A Transformer is a form of transduction model that relies solely on self-attention to figure out how to represent its inputs and outputs. It does so without the use of any sequence aligned recurrent neural networks(RNNs) or convolutions.

Through our research proposal we wish to highlight why we are going to be performing this research and putting in the effort to devise a new architecture. We have divided our research proposal into 7 sections. In Section 1, we shall take a look at briefly introducing the concept and why our work is necessary. Next, in Section 2, we outline the background work that has already been done in this field and how some of the papers relate to the work that has been done. We use this as an opportunity to highlight some of the shortcomings in current architectures and modelling techniques. In Section 3, we briefly outline the aim of our proposed research. In Section 4, we define in some detail the work that we will do to establish our research and how we plan to quantify the work that shall be done. Section 5, highlights our goal, which is to produce a new transformer architecture that performs better at Question Answering based tasks. In Section 6, we have outlined the minimum hardware requirements along with the resources available to the author that will be used to conduct this research. Finally, in Section 7, we submit a Gantt Chart to outline our plan against the number of weeks.

2 Background and Related Research

In this section, we shall highlight what has led us this far and some of the interesting challenges that it poses. In 2.1, we briefly look at the history of Natural Language Processing and how some of the challenges were addressed. In 2.2, we look at the latest research that has gone into creating the Transformer architecture, identify some of the common patterns and use that information to strategise our model in later sections.

2.1 Background

The area of Natural language Processing has taken significant leaps in the last two decades. Work done towards improving the ability of machine learning models to first recognize words, then sentences, followed by contextual understanding has led to several interesting and novel approaches in the field. From early on neural networks to creating Long Short-Term Memory architectures (Schmidhuber and Hochreiter, 1997) by Sepp Hochreiter and Jurgen Schmidhuber in the mid-'90s that resolved the vanishing gradient problem of classical neural networks, we have come a long way.

The latest advancements in this field come from Google's research lab in the form of *Transformers*. We look into this in a bit more detail later. However, no model can be successful without a good dataset to train on. This is where the SQuAD 2.0 dataset (Rajpurkar et al., 2018) comes in. This dataset is what forces the machine learning models to do contextual understanding. One might even say that it forces the models to "think" for themselves before answering a task.

Let us now look at some of the key research that has been done in this regard.

2.2 Related Research

Outlined below are some of the most important pieces of research that relate directly to the work done for Transformers and Q&A based systems.

1. The SQuAD 2.0 Dataset (Rajpurkar et al., 2018), was developed with funding from Facebook to help address some major issues with existing datasets. Most datasets focus on questions that can be easily answered or use of automatically generated, unanswerable questions which are easily identifiable.
   The SQuAD 2.0 dataset resolves this by combining the SQuAD dataset along with 50,000 crowd worker generated unanswerable questions.

The key feature of these being that the unanswerable questions must look similar to answerable ones. For a model to be successful on this new dataset, it must be able to answer all possibly answerable questions as well as determine when no answers are provided for a question in the given paragraph and abstain from answering. A comparative study was done for a Natural Language Understanding(NLU) task that obtained an 86% score on SQuAD 1.1, only got 66% on the new 2.0 dataset. The dataset helps bridge the gap between true NLU and machine understanding by using the concept of Relevance. Through comparisons with various datasets such as RACE, MCTest, QASENT etc. they have identified the missing links like negative examples, antonyms and helped fill the gap. This dataset forces the models to understand whether a paragraph span has the answer to the question posed.

2. Work done in the field of Long short-term and gated recurrent (Hochreiter et al., 2001) and (Zhou et al., 2016) neural networks, in particular, has been established as a state of the art approach in sequence modelling, transduction problems such as language modelling and machine translation. In their paper Attention Is All You Need,(Vaswani et al., 2017) the team set out to resolve problems in the parallelization and increased compute times of recurrent models. The inherently sequential nature of RNNs causes issues in memory constraints, leading to reduced batch sizes.

   The architecture for a *Transformer* in this paper is outlined as having
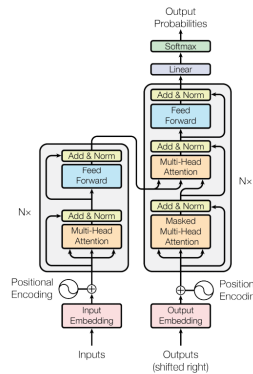
Figure 1: Transformer Architecture built by
(Vaswani et al., 2017)

an encoder that maps input sequences to a continuous representation. The architecture can be seen above in Figure 1. This is then decoded into an output sequence of symbols one at a time. Each step is auto-regressive, i.e. it consumes the previously generated symbols as additional input when creating the next. This is similar to an ensemble model. Stacks of 6 encoder layers and 6 decoder layers is used.

The encoder layers each have 2 sub-layers of a multi-head self-attention and the other a simple, position-wise fully connected feed-forward network layer.

The decoder layer is similar to the encoder layer and has an additional 3rd sub-layer that performs multi-head attention over the output of the encoders. There is also normalization and the outputs are prevented from attending to subsequent positions.

The attention mechanism can be described as mapping a query to a set of key-value pairs. This can be seen from Figure 2, below.

The evaluations performed on the Wall Street Journal dataset(Marcus et al., 1993), using 40k sentences, showed that even without task-specific tuning the model had better results with a fraction of the training cost.
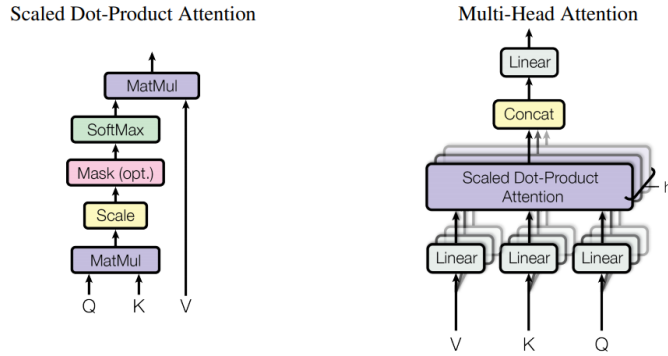
Figure 2: Scale Dot and Multihead Attention
Models Vaswani et al. (2017)

3. The paper on BERT, which is *Bidirectional Encoder Representations from Transformers* (Devlin et al., 2018), introduces a new language model. This model is truly fascinating in many ways. First and fore-

most it is designed to pre-train deep bidirectional representations using unlabelled data. This is done by jointly conditioning context in all layers to the right and left. This pre-training allows the model to be fine-tuned simply using one additional output layer. These features make this model conceptually simple and very powerful empirically.

BERT employs language pre-training (Dai and Le, 2015) which has shown significant advantages in many applications e.g. paraphrasing, language level inference etc. These tasks aim to highlight the relationships between sentences through contextual understanding as well as by using tokenized outputs.
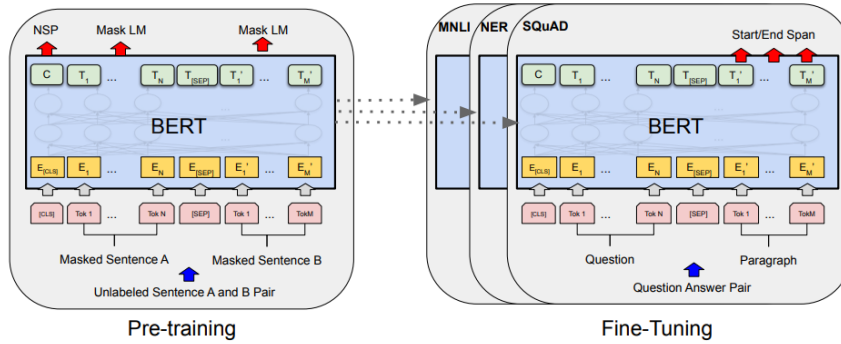
Figure 3: Pre-training and Fine Tuning procedures
for BERT (Devlin et al., 2018)

BERT was tested on The General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2018) which has a large number of diverse NLU tasks. BERT performed extremely well on the 11 NLP tasks that the authors ran it against. it showed an average accuracy improvement of 4.5% and 7% when compared to the previous state of the art models. Results of BERT and its significant gains make it one of the best candidate models for NLU tasks.

Several advancements have been made to the BERT model to make it fast, better at understanding and even performing application-specific tasks such as Q&A.

The SQuAD 2.0 dataset has inspired an LSTM based FastQA (Weissenborn et al., 2017) model architecture. This architecture takes cues from the work done by Hochreiter and Schmidhuber in their work on Long Short-Term Memory Architecture (Hochreiter et al., 2001), to create a model specifically for end-to-end question answering systems.

A common theme with BERT is that it takes a long time to train. Especially in the BERT based RoBERTa architecture Liu et al. (2019). RoBERTa takes on average 4-5 times more time to train than BERT, however, it also shows a maximum of 20% improvement over BERT, depending on the application. ALBERT, which is A Liter BERT, Lan et al. (2019) was developed specifically to deal with memory limitations and reduce training times. ALBERT's XXL implementation has been documented to perform better in models trained on the BOOKCORPUS and Wikipedia ones by at least 2% across multiple applications. In a smaller amount of time.

However, none of these model architectures has been able to address all the problems and serve as more generic solutions to multiple end-to-end sequence encoding problems across various applications.

Our work is primarily focused on building a fast model that allows for higher accuracy in responses specifically with Q&A systems.

3   Aims and Objectives

The main aim of this research is to propose a new transformer architecture that can perform better at Q&A using the SQuAD 2.0 dataset. We shall:

1. Implement the existing models that are available via libraries such as HuggingFace (Team), PyTorch and Tensorflow on the dataset

2. Obtain F1, validation, etc. scores for existing models and treat them as our benchmark scores

3. Identify drawbacks of the current architectures

4. Design our architecture and evaluate its performance

5. Fine-tune the architecture, re-evaluate and report improvements

6. Compare the results of our Transformer model with the benchmark scores.

4 Research Methodology

To implement this research we shall break the project down into 5 phases. These are outlined below.

4.1 Research Dataset

We have selected the Stanford Question Answering Dataset (SQuAD). This is as a reading comprehension dataset based on Wikipedia articles. It is based on questions posed by crowd-workers on a set of articles. The answer to every question is a segment of text or span, from the corresponding reading passage, or the question might be unanswerable (Rajpurkar et al., 2018).

The dataset consists of over 150,000 questions. Split into 100,000 answerable and 50,000+ unanswerable question, which were written to look similar to unanswerable questions. The challenge being that a model should be able to correctly answer the answerable questions and abstain from answering the unanswerable ones. The dataset is freely available as a part of the Transformers package in python or it can be downloaded from the SQuAD 2.0 website (Rajpurkar).

To effectively use this dataset for our purposes, let us first take a look at what its contents look like below.

Context: *"The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from Denmark, Iceland and Norway who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia."*
Question: *Who was the Norse leader?*
Answer: *Rollo*

The answer to the aforementioned question is quite simple for humans to comprehend. The challenge is for us to contextualize this and make it machine-understandable so that our model can answer it correctly.

The dataset consists of various kinds of English language examples like negation, antonyms, entity swaps, impossible conditions to answer, answerable, etc. making the dataset a well-balanced one.

To use this dataset correctly we shall perform the following pre-processing steps on it:

1. Data splitting into separate Question, Answer and Context lists.

2. Splitting the data into separate training and validation sets of question and answers using the 80/20 rule, also known as the Pareto principle. We will have 80% training data and 20% test data.

3. Tokenization of the split data to generate "context-question" pairs

4. Generating indexes for when an answer begins and ends in the dataset

5. Adding answer tokens based on their encoded positions

4.2 Research Benchmarks

Here we shall focus on obtaining benchmark scores for the shortlisted architectures i.e BERT (Devlin et al., 2018), DistilBERT (Sanh et al., 2019) and ALBERT Lan et al. (2019), on the above dataset.

We shall use the F1, Exact Match(EM), Recall and Training Time scores to create a benchmark to compare our architecture against. The Exact Match score will help us identify how many questions were 100% correctly answered by each model.

4.3 Architecture Creation

In this section we will:

1. Mathematically model a new transformer architecture

2. Code the architecture

3. Run sample dataset to identify base benchmarks

4. Run the SQuAD 2.0 dataset to obtain 1st pass performance benchmarks

5. Document architecture performance, identify pros and cons

4.4 Architecture Refinement

In this phase, we will focus on:

1. Reviewing the results from the previous section

2. Identifying the areas of improvement

3. Hypothesise the improvements and implement them in the architecture

4. Run the SQuAD 2.0 dataset to obtain new performance benchmarks

5. Document architecture performance, identify pros and cons

4.5 Model Evaluation

The training shall be carried out using the train-test loss plot to identify the optimal number of epochs for which our model needs to be run. This will also be done for the selected model architectures.

The main parameters we will use for model evaluation are F1, Exact Match(EM), Recall and Training Time.

These metrics will help us reiterate and quantify correctly if our model has improved performance or not.

5 Expected Outcomes

We expect that our created model is at-par, if not better, at performing Q&A than existing models.

6 Requirements and Resources

To successfully deliver on our research we will be utilizing the following hardware:

- EVGA GeForce RTX 2070 SUPER KO GAMING, 08G-P4-2072-KR, 8GB GDDR6, Dual Fans(Evga). This graphics card is based on the Nvidia "Turing" architecture and has 2560 CuDA cores.

- Intel 10700 processor. 8 cores, 16 threads, 16M cache(Intel).

- VENGEANCE® LPX 8GB (1 x 8GB) DDR4 DRAM 2400MHz C14 Memory Kit - Black(Corsair). 8GB x 4, 32 GB total.

- Ubuntu 20.04 Operating System

- We will also be using the latest versions of the following packages: Pandas, NumPy, SciPy, Transformers by HuggingFace, Matplotlib, Tensorflow and PyTorch. In case there are compatibility issues the appropriate versions will be mentioned. We will also mention any other packages that might be required in the course of the research.

The above hardware is available to the author and any changes to the same will be notified/highlighted in the subsequent reports.

7   Research Plan

Shown on the next page is the Gantt Chart highlighting the research stages and timelines.

Compare & Contrast Existing Transformer Architectures To Develop A New Architecture

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Research Proposal	1	2	1	2	100%
Literature review	2	3	2	3	50%
Model & Dataset Selection	2	1	2	1	100%
Research Benchmarks	3	2			0%
Mid-thesis Report	4	1			0%
Architecture Creation	5	3			0%
Architecture Evaluation	8	2			0%
Architecture Refinement	10	3			0%
Architecture Re-evaluation & Finalization	13	1			0%
Final Architecture Conclusion	14	2			0%
Report Completion	16	3			0%
Video Presentation	18	1			0%

References

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

A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. 4:2047–2052 vol. 4, 2005. doi: 10.1109/IJCNN.2005.1556215.

Ming Tan, Bing Xiang, and Bowen Zhou. Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108, 2015. URL `http://arxiv.org/abs/1511.04108`.

Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016. URL `http://arxiv.org/abs/1608.07905`.

Giovanni Di Gennaro, Amedeo Buonanno, Antonio Di Girolamo, Armando Ospedale, and Francesco A. N. Palmieri. Intent classification in question-answering using LSTM architectures. *CoRR*, abs/2001.09330, 2020. URL `https://arxiv.org/abs/2001.09330`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, and Bowen Zhou. Applying deep learning to answer selection: A study and an open task. *CoRR*, abs/1508.01585, 2015. URL `http://arxiv.org/abs/1508.01585`.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL `http://arxiv.org/abs/1806.03822`.

Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224, 1961.

William A Woods and WOODS WA. Lunar rocks in natural english: Explorations in natural language question answering. 1977.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. URL `http://arxiv.org/abs/1910.01108`.

HuggingFace. Transformers, June 2021. URL `https://huggingface.co/transformers/`. Accessed: 2021-06-20.

Pranav Rajpurkar. Squad2.0, 2021. URL `https://rajpurkar.github.io/SQuAD-explorer/`. Accessed: 2021-04-16.

Mark B Ring. Learning sequential tasks by incrementally adding higher orders. *Advances in neural information processing systems*, pages 115–115, 1993.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. URL `http://arxiv.org/abs/1506.03340`.

Siddhartha Brahma. Suffix bidirectional long short-term memory. *CoRR*, abs/1805.07340, 2018. URL `http://arxiv.org/abs/1805.07340`.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL `http://arxiv.org/abs/1606.05250`.

Shuohang Wang and Jing Jiang. Learning natural language inference with LSTM. *CoRR*, abs/1512.08849, 2015. URL `http://arxiv.org/abs/1512.08849`.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

Laura Dietz, Manisha Verma, Filip Radlinski, and Nick Craswell. Trec complex answer retrieval overview. In *TREC*, 2017.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3302. URL `https://www.aclweb.org/anthology/W14-3302`.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017. URL `http://arxiv.org/abs/1705.03122`.

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. *arXiv preprint arXiv:1511.01432*, 2015.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. 2018.

Wei Wang, Ming Yan, and Chen Wu. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. *arXiv preprint arXiv:1811.11934*, 2018.

Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *CoRR*, abs/1907.10597, 2019. URL `http://arxiv.org/abs/1907.10597`.

Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402. 1150464. URL `https://doi.org/10.1145/1150402.1150464`.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020.

Shir Gur. Pointer net, Sep 2017. URL `https://github.com/shirgur/PointerNet/blob/master/PointerNet.py`. Accessed: 2021-07-4.

Saptadeep Pal, Eiman Ebrahimi, Arslan Zulfiqar, Yaosheng Fu, Victor Zhang, Szymon Migacz, David W. Nellans, and Puneet Gupta. Optimizing multi-gpu parallelization strategies for deep learning training. *CoRR*, abs/1907.13257, 2019. URL `http://arxiv.org/abs/1907.13257`.

Giuliano Giacaglia. How to scale training on multiple gpus, Dec 2020. URL `https://towardsdatascience.com/how-to-scale-training-on-multiple-gpus-dae1041f49d2`.