Constructing Adversarial Examples in Question-Answering Systems

Vishakha Gautam

University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Part of the Computer Sciences Commons

Recommended Citation
https://scholar.uwindsor.ca/etd/9037

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.
Constructing Adversarial Examples in Question-Answering Systems

By

Vishakha Gautam

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2023

©2023 Vishakha Gautam
Constructing Adversarial Examples in Question-Answering Systems

by

Vishakha Gautam

APPROVED BY:

___________________________
N. Zhang
Department of Electrical and Computer Engineering

___________________________
J. Lu
School of Computer Science

___________________________
J. Chen, Advisor
School of Computer Science

March 16, 2023
DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.
ABSTRACT

Being an NLP pre-training model built with transformer and attention mechanism, BERT has proven to be highly efficient in developing various applications like text classification, machine translation, question-answering systems etc. Despite the recent advancement, however, the generalizability of the models remains a challenging issue. In this thesis, we study the generalizability issues of the prediction models in the question-answering systems, particularly for the unanswerable examples. To gain the insight about where the models do not generalize well, we are interested in constructing adversarial examples that are challenging for the model to predict correctly. The adversarial examples are obtained by pairing each question with a different context in a same dataset. Constructing adversarial examples only, we make sure that the new context does not contain any answer to the question it is paired with. In order to maximally challenge the prediction models, among the large number of candidates of the context to a given question, we select the one with the highest text similarity score to the original context of this question. The proposed method is exercised on SQuAD, a benchmark question answering dataset, with three deep learning models, namely, BERT, LSTM, and GRU, respectively. Our experiment shows that the examples constructed from the proposed method drastically reduce the performance of the models, from a range of 3.19-6.4% to a range of 0.03-0.18%, demonstrating the effectiveness of the method. The experiment also shows that the existing models are capable of learning from the constructed examples, leading to the enhanced performance.

Keywords: BERT, Data augmentation, QA system, Robustness, Adversarial Example
DEDICATION

I would like to dedicate this thesis to my parents

Mrs.Kiran & Mr.Prasenjit

my sister - Vipassna

and

my best friend - Aayush.

I would not have made it here without your unwavering support, trust and endless love.

Thank you for always believing in me.
AKNOWLEDGEMENTS

I would like to sincerely express my most profound gratitude towards my supervi-
sor Dr. Jessica Chen, whose input helped me immensely. With her input, I was able
to look at my research with a different perspective and a more critical eye.

Secondly, I would like to express my gratitude to my thesis committee members
for their beneficial advice and suggestions for my thesis.

I would also like to thank my friends Chandan and Yalda for always encouraging
and supporting me.

I humbly extend my thanks to the School of Computer Science and all concerned
people who helped me in this regard.
# TABLE OF CONTENTS

**DECLARATION OF ORIGINALITY**  
III

**ABSTRACT**  
IV

**DEDICATION**  
V

**ACKNOWLEDGEMENTS**  
VI

**LIST OF TABLES**  
IX

**LIST OF FIGURES**  
X

**LIST OF ABBREVIATIONS**  
XII

1 **Introduction**  
1.1 Problem ........................................... 1  
1.2 Research Question .................................. 2  
1.3 Existing Approaches .............................. 2  
1.4 Relevance to Society and Research .......... 3  
1.5 Thesis Outline .................................. 3

2 **Background**  
2.1 Deep Learning .................................. 4  
2.2 Natural Language Processing .................. 4  
2.3 Long Short-Term Memory Cell (LSTM) ........ 5  
2.3.1 How do LSTMs work? .......................... 5  
2.3.2 The Logic of LSTM: .......................... 7  
2.3.3 Bidirectional LSTM: ......................... 8  
2.4 Gated Recurrent Unit ........................... 9  
2.5 Transformers ................................... 10  
2.5.1 Transformer Architecture Explained ....... 11  
2.5.2 Self-Attention has 3 major elements ...... 12  
2.6 BERT ............................................. 14  
2.6.1 BERT’s mechanism ........................... 14  
2.6.2 BERT’s architecture ......................... 16  
2.7 Question Answering Systems .................. 17  
2.7.1 Applications of QA systems ................. 18  
2.7.2 BERT’s role in building QA systems ...... 18  
2.7.3 BERT’s Input Format: ......................... 18  
2.7.4 Start and End Token Classifiers ............ 20  
2.8 Data Augmentation ............................. 21

3 **Related Work**  

VII
# Method

4.1 Data Acquisition .................................................. 28  
4.2 Models used ....................................................... 35  
4.3 Evaluation Metrics .............................................. 36

# Experiments and Results

5.1 Experiments using Model 1: BERT ................................. 42  
5.2 Experiments using Model 2: Gated Recurrent Units ............ 46  
5.3 Experiments using Model 3: LSTM (Long Short-Term Memory) 51  
5.4 Experiments using Subset of SQuAD 2.0 and adversarial examples on BERT .................................................. 53  
5.5 Model Performance on Test Set: Summary ....................... 55

# Conclusion and Future Steps

6.1 Conclusion .......................................................... 57  
6.2 Future Steps ........................................................ 59

REFERENCES ............................................................ 60

VITA AUCTORIS ........................................................... 63
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.1</td>
<td>Difference between: LSTM and GRU</td>
<td>10</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Example</td>
<td>37</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Adversarial Examples</td>
<td>44</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Models’ Performance: Summary</td>
<td>56</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.3.1 LSTM Memory Cell ............................................. 6
2.3.2 Sigmoid Function [27] ........................................... 6
2.3.3 Bidirectional LSTM model ..................................... 8
2.4.1 GRU Architecture ................................................. 9
2.5.1 Transformers Architecture .................................... 11
2.5.2 Query, Key and Value in Self-Attention .................... 12
2.5.3 Scaled Dot Product Attention [28] ............................ 13
2.6.1 BERT: Process Flow ............................................. 15
2.6.2 BERT Variants[20] ............................................... 16
2.7.1 BERT for QA systems: Input Format ....................... 19
2.7.2 BERT: Start and End Token ................................... 20
4.0.1 Flowchart for Creating Adversarial Examples ............ 28
4.1.1 Vector Embedding Representation for a context from SQuAD 2.0 dataset .......................................................... 30
4.1.2 Code Snippet: Calculating Context Similarity using BERT embeddings .......................................................... 31
4.1.3 Similarity Values for a Context in SQuAD 2.0 .............. 31
4.1.4 2-Dimensional Visualization of Context Embeddings ........ 32
4.1.5 Code Snippet: Construct Adversarial Examples .......... 33
4.1.6 SQuAD 2.0: JSON format ...................................... 34
4.1.7 Answerable and Unanswerable Question for 10 Random Contexts .......................................................... 35
4.3.1 Code Snippet: Computing EM and F1-Score ............... 39
5.1.1 BERT: Training on SQuAD 2.0 .................................. 43
5.1.2 BERT: Training on SQuAD 2.0 + Adversarial Examples .......................................................... 46
5.2.1 GRU: Training on SQuAD 2.0 based on [18] .................. 48
5.2.2 GRU: Training on SQuAD 2.0 .................................. 49
5.2.3 GRU: Training on SQuAD 2.0 + Adversarial Examples .......................................................... 50
5.3.1 LSTM: Training on SQuAD 2.0 .................................. 51
5.3.2 LSTM: Training on SQuAD 2.0 + Adversarial Examples . . . . . . 53
5.4.1 BERT: Training on Subset of SQuAD 2.0 + Adversarial Examples . 54
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Stanford Question Answering System</td>
</tr>
<tr>
<td>QAS</td>
<td>Question Answering System</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>GRU</td>
<td>Gated Recurrent Unit</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long-Short Term Memory</td>
</tr>
<tr>
<td>BERT</td>
<td>Bidirectional Encoder Representations from Transformers</td>
</tr>
<tr>
<td>NLU</td>
<td>Natural Language Understanding</td>
</tr>
<tr>
<td>EM</td>
<td>Exact Match Score</td>
</tr>
</tbody>
</table>
CHAPTER 1

Introduction

1.1 Problem

With the recent advancements in the field of machine learning, question answering systems are gaining a lot of momentum in various applications. Question answering systems when posed with a query; help us retrieve a relevant answer to the query. Therefore, they are being used in popular search engines such as Google, Bing, etc. Question answering systems have also paved their way into modelling chatbots for banking industries, medical sectors, and various others.

Traditionally, question answering systems were being developed with the help of information retrieval and knowledge bases and they were successful enough for being used in plethora of applications [10]. Recently, deep learning has proved to be impactful for question answering system-based applications and its performance has helped researchers develop question answering systems with high accuracy.

There are two basic approaches to question-answering problems. Along with the first approach, the answer to a given question can be generated freely. For the second approach, the generated answer should be part of the given context. In this thesis, we consider the issues in the latter. One major problem in this setting is the lack of data, especially for training the model to answer correctly when the given example is unanswerable.

Although the question-answering models are performing well; they are not robust enough as they lack generalizability [10]. Ample amount of training data along with challenging instances should be generated to improve the robustness of question an-
swering systems, thus, rendering them with extreme power and robustness to be used in complex applications that require highly accurate models.

1.2 Research Question

The question that this thesis considers is: How to construct effective adversarial (negative) examples from the original dataset to improve the robustness of question answering systems?

1.3 Existing Approaches

One popular technique to improve robustness in question answering systems is by data augmentation, where the addition of more data samples to improve the robustness of these systems can be done by paraphrasing the questions or the context available in the original dataset.

Lately, researchers have applied various techniques to augment data for question answering systems. In existing literature, quite a few ways have been adopted to improve the robustness of QA systems by augmenting data, for example,

- Deleting sentences/words from original context [15]
- Adding sentences/words to original context [15]
- Paraphrasing questions and context [15]
- Adding adversarial examples by swapping, replacing, and inserting sentences, words, or passages [15]

In this thesis, we consider augmenting the dataset by adding adversarial examples. These examples are constructed from existing ones by replacing the original context with a context having highest similarity to it. The newly constructed examples are adversarial in the sense that the replacing context does not have the ground truth answer in it. We have evaluated our results on state-of-the-art models including BERT and Transformers.
1.4 Relevance to Society and Research

Improving the robustness of question answering systems can help researchers working with data in finance, medical and other domains to produce highly accurate question answering systems such as chatbots for prescribing medicines when doctors are not available or bots for generating personalized finance suggestions to everyone (e.g.: which stock to buy or not, etc.).

1.5 Thesis Outline

For readers to better understand the methods and models used in this thesis work, some background is summarized in chapter 2. The current literature pertinent to the thesis is introduced in chapter 3. The workflow, implementation specifics, and the evaluation methods of the thesis work are provided in Chapter 4. The findings of the thesis work are detailed in chapter 5. The results are discussed in Chapter 6 along with a summary of the key findings related to the topic being studied.
CHAPTER 2

Background

2.1 Deep Learning

Deep Learning is a sub-field of machine learning that makes use of artificial neural networks which imitate the function and structure of a human brain. Earlier, artificial neural networks were limited as computing power was very limited and thus, they could not be used for complex applications. With the improvement in computer storage and power, neural networks are performing faster than humans for a lot of use cases. Deep Learning uses backpropagation to help improve the learning process. Hence, deep learning has helped build a lot of great applications based on image classification, speech recognition, text classification, language translation and many more.

Deep Learning is used for two major fields – Computer Vision and Natural Language Processing. In our work, we will be using Natural Language Processing for achieving the desired results for the considered application.

2.2 Natural Language Processing

Natural Language Processing (NLP) is a subset of deep learning that helps in processing human language automatically or semi-automatically. NLP is highly efficient in helping computers communicate with humans in natural language such as English. NLP is a multidisciplinary field that is related to linguistics, and it also finds its applications in psychology, philosophy, maths, and cognitive science. NLP is being
used to concoct a wide range of applications such as text classification, text segmentation, summarization, machine translation, speech bots, question answering, sentiment analysis, etc.

Recurrent neural network is a type of neural network architecture that is widely used in the development of NLP based applications. RNNs store the information of both the current feature and feature of the neighbours for prediction purposes. Historical information based on long distance features is maintained in the RNNs memory that enables the model to better predict the output.

2.3 Long Short-Term Memory Cell (LSTM)

RNNs find it difficult to learn long range dependencies and this is where LSTMs play a major role. LSTMs are a special type of RNN that help in understanding the context in addition to learning the dependencies. The architecture of LSTMs is mostly like the conventional RNN, but LSTMs replace the usual hidden layer updates with memory cells. These cells help in learning long range dependencies in data [7].

2.3.1 How do LSTMs work?

Long short-term memory has input connections, which means it can handle large groups of data in addition to single data items like pictures. This finds application in machine translation, speech recognition, and other areas. Long short-term memory RNNs are an excellent type of RNN that performs well on a wide range of problems.

Long short-term memory’s chain-like architecture allows it to store data for longer periods of time, allowing it to tackle testing tasks that traditional recurrent neural networks struggle with or simply cannot resolve.
2. BACKGROUND

The three key components of the LSTM model are as follows:

**Forget gate:** Removes information that is no longer required to complete the task. This process is critical to streamlining the network’s display.

**Input gate:** The input gate is in charge of adding information to the cells.

**Output gate:** The output gate selects and outputs critical information.

Note: Gates in LSTM are sigmoid activation functions, which means they output a value between 0 and 1, which is usually either 0 or 1.
2. BACKGROUND

Equation of Gates

\[ f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \]
\[ i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \]
\[ o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \]

In the above equations, 't' represents current timestamp.

Symbol Representation

\[ i_t \rightarrow \text{Represents input gate.} \]
\[ f_t \rightarrow \text{Represents forget gate.} \]
\[ o_t \rightarrow \text{Represents output gate.} \]
\[ \sigma \rightarrow \text{Represents sigmoid function.} \]
\[ w_x \rightarrow \text{Weight for the respective gate(x) neurons.} \]
\[ h_{t-1} \rightarrow \text{Output of the previous lstm block (at timestamp t-1).} \]
\[ x_t \rightarrow \text{Input at current timestamp.} \]
\[ b_x \rightarrow \text{Biases for the respective gates (x).} \]

2.3.2 The Logic of LSTM:

An LSTM model is focused on a memory cell called a cell state that maintains its state across time. The changes in cell states can be represented graphically in a 2-Dimensional plane by a curve.

Gates control the removal and addition of information from the LSTM cell state. These gates, on the other hand, allow information to flow freely throughout the cell. A pointwise increase activity and a sigmoid indifference net layer help the gates.

The curved tier returns a numerical value between 1 and 0, where 1 means everything should be allowed through and 0 means nothing should be allowed through.
2.3.3 Bidirectional LSTM:

Bidirectional LSTM (BiLSTM) is a recurrent neural network that is mostly used for natural language processing. Unlike traditional LSTM, the input travels in both directions, and it can use information from both sides. It is also an effective tool for simulating the sequential relationships between words and phrases in both directions.

To summarise, BiLSTM adds an additional LSTM layer that reverses the direction of information flow. In a nutshell, it means that the input sequence is reversed in the additional LSTM layer. The outputs of both LSTM layers are then combined in a variety of methods, including average, sum, multiplication, and concatenation.

![Bidirectional LSTM model](Fig_2.3.3_Bidirectional_LSTM_model.png)

Fig. 2.3.3: Bidirectional LSTM model
2.4 Gated Recurrent Unit

The Gated Recurrent Unit, or GRU, has the same workflow as the RNN, but the distinction is in the operation and gates connected with each GRU unit. GRU integrates two gate operating techniques named Update gate and Reset gate to solve a problem encountered by ordinary RNN.

![GRU Architecture](image)

**Fig. 2.4.1: GRU Architecture**

**Update Gate:** The update gate is in charge of determining how much previous information must be passed along to the next state.

**Reset Gate:** The reset gate is employed in the model to determine how much past information may be ignored; in other words, it determines whether the previous cell state is important or not.
Difference between LSTM and GRU

<table>
<thead>
<tr>
<th></th>
<th>LSTM</th>
<th>GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LSTM has three gates – forget, input and output gates</td>
<td>The GRU has two gates – update and reset gates</td>
</tr>
<tr>
<td></td>
<td>In LSTM, forget gate connects the input and output gates, whereas in GRU, the reset gate is applied directly to the prior hidden state. In LSTM, the two gates, input, and output, share the task of the reset gate in GRU</td>
<td>GRU does not have any internal memory and lacks the output gate found in LSTM</td>
</tr>
</tbody>
</table>

Table 2.4.1: Difference between: LSTM and GRU

2.5 Transformers

LSTM faces an issue in learning long range dependencies just like RNNs when the sentences are too long as it becomes very hard for the model to retain content for distant positions in the sentence \[17\]. This issue is highly critical for applications where we need to predict next word based on previous ones.

To solve the problems in RNNs and LSTMs, researchers came up with a technique known as ‘Attention’. Attention is a mechanism that is used to selectively concentrate only on a few relevant details while ignoring the others. Transformer uses its attention mechanism to learn global dependencies between input and output. It relies on self-attention mechanism to compute representations of input and output without getting help from recurrent units or convolutions. Transformers allow parallelization as they don’t use recurrent units, but obtain their features using weighted sums and activations.
2.5.1 Transformer Architecture Explained

Transformer uses an Encoder-Decoder architecture to accomplish its goals. The encoder part is used for extracting features from an input sentence and these features are then used to produce an output using decoder. In [28], multiple blocks of encoder and decoder are used. In Figure 2.5.1, Nx represents multiple blocks.

![Fig. 2.5.1: Transformers Architecture](image)

As mentioned earlier, Transformers make use of Self-Attention mechanism. It is an operation that can be applied to sequence-to-sequence processing tasks that can be defined as a mechanism for relating different positions of a single sequence or sentence to learn clear representations of each sentence. Self-attention uses dot product to calculate weighted average over all input vectors to produce output vector. According to [28], for each word embedding, there should be three different vectors namely Query (Q), Key (K) and Value (V) corresponding to it. Attention function can be defined as mapping a query to a set of key-value pairs. Note that Key, Value, and Query are all vectors.
2.5.2 Self-Attention has 3 major elements

1) **Scaled Dot Product Attention** – In practice, attention is the scaled dot product attention that is calculated by computing the dot product of input queries with all input keys and later uses SoftMax to obtain the weights from the values. The calculated attention scores are used to measure how much focus should be placed on words of input sequence w.r.t the word at a certain position. The scaling factor is used to resolve the issue of the vanishing gradients caused by the incompetency of the SoftMax to work with large values. After applying SoftMax, the value matrix is multiplied by softmax. This is done to retain the values of words that should be given attention as they are pivotal to learn semantic representation of context and the value for irrelevant words should be minimized as they don’t provide much of the contextual details.
2. BACKGROUND

2) **Multi-head attention** – It is used because attention scores only focuses on complete sentences at a time, and this would produce same results even if the word arrangement in a sentence was changed. On the other hand, we want to attend to different segments of words. Combining several attention heads gives self-attention more power of discrimination. Multi-head attention helps expand the model’s ability to focus on different positions and gives the attention layer multiple "representation subspaces”.

3) **Positional Encoding** – Transformers process all the words in a sentence simultaneously, unlike RNNs. Therefore, there is no inherent word-ordering or position information in Transformers. Positional Encoding makes it possible for the model to have knowledge about word order. Position encoding vectors are calculated using equations.

Assuming we have a sentence of length 'L' and we require the position of 'pos' object within this sentence. We can calculate the positional encoding using sine and cosine functions of varying frequencies. In the equations below, sine function represent even positions and cosine represents odd positions [16].
2. BACKGROUND

\[ PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{(2i/d_model)}}\right) \]
\[ PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{(2i/d_model)}}\right) \]

Symbol Representation

\( pos \rightarrow \) Position of an object in the input sequence.

\( d_{model} \rightarrow \) Dimension of the output embedding space.

\( PE_{(pos,x)} \rightarrow \) Position function for mapping a position ‘pos’ in the input sequence

\( i \rightarrow \) Used for mapping to column indices.

Transformers are a game-changer in the field of natural language processing as they helped achieve astonishing results in various NLP tasks.

2.6 BERT

BERT, short for Bidirectional Encoder Representations from Transformers, is an NLP technique for Pre-training. It is a language model that learns representations of language using the encoder part of a transformer. BERT is a language model that has been trained bidirectionally. As opposed to the single-direction language models, we can now perceive the context and flow of language more deeply using BERT. BERT uses unsupervised way for learning representations and it can be then fine-tuned for carrying out downstream tasks using supervised learning.

2.6.1 BERT’s mechanism

Since the objective of BERT is to produce a language representation model, only the encoder portion is used. A series of tokens that are first transformed into vectors and then processed by the neural network make up the input to the BERT encoder. However, BERT requires the input to be modified and embellished with additional
metadata before the process begins:

A. **Token embeddings:** At the start of the first sentence, a [CLS] token is added to the input word tokens, and at the conclusion of each sentence, a [SEP] token is added.

B. **Segment embeddings:** Each token receives a marking designating Sentence A or Sentence B. Because of this, the encoder can tell the sentences apart.

C. **Positional embeddings:** Each token is given a positional embedding to show where it belongs to in the sentence.

![Fig. 2.6.1: BERT: Process Flow](image)

A key point about BERT is that it doesn’t try to predict next word in a sentence. Instead, BERT uses two strategies for training –

A. **Masked Language Model (MLM):** Run the full sequence through the BERT attention-based encoder, then forecast only the masked words based on the context provided by the other non-masked words in the sequence. Randomly mask off 15% of the input words, substituting them with a [MASK] token.
B. **Next Sentence Prediction (NSP):** The BERT training technique also makes use of next sentence prediction to comprehend the relationship between two sentences. When working on tasks like answering questions, a pre-trained model with this kind of expertise is useful. The model learns to predict whether the second sentence is the next one in the original text when given pairs of sentences as input during training.

Masked LM and Next Sentence Prediction are used together to train the model. With the idea that "together is better," this is done to reduce the combined loss function of the two techniques.

### 2.6.2 BERT’s architecture

Depending on the size of the model architecture, there are 2 different pre-trained BERT variants: BERT-Base and BERT-Large.

**Fig. 2.6.2: BERT Variants**

**Fine-Tuning BERT for downstream tasks:** In a wide range of general language understanding tasks, such as sentiment analysis, question answering, paraphrase detection, and linguistic acceptability, BERT exceeded the state-of-the-art.
BERT can be adjusted for a specific task by simply adding a single layer on top of the core model depending on our own dataset.

One well-known use case of BERT is Question-Answering systems. Answering questions is essentially a prediction problem; given a question as input, the objective is to find the appropriate response from a corpus. So, the model predicts a start and an end token from the paragraph that most likely answers the query given a question and a context paragraph. This means that a model for our application can be trained using BERT by learning two additional vectors that indicate the start and end of the response. The question becomes the first sentence in the input sequence, just like in the sentence pair tasks, and the paragraph becomes the second sentence. This time, though, two additional parameters—a start vector and an end vector—are learned through fine-tuning.

The majority of the hyper-parameters in the fine-tuning training remains the same as in the BERT training; [11] provides detailed instructions on the tuning of the hyper-parameters.

2.7 Question Answering Systems

Given a context, question answering systems are models that help us retrieve relevant and reliable answers when a query is posed. These systems are being widely adopted in building smart virtual assistants for customer support, search engines and FAQ bots used in enterprise setups [8].

Question answering systems can be classified in two ways based on their style of generating answers:

- **Extractive QA** – In extractive QA models, the answer is extracted from a context and then supplied to the user directly. This type of model leverage language models such as BERT.

- **Generative QA** – Generative QA models generate answers freely using the context provided to them. These models make use of Text Generation models.
Furthermore, QA systems can also be categorized based on the origin of the answers:

- Open QA – The answer is extracted from a given context.
- Closed QA – An answer is generated by a model and there is no context present.

### 2.7.1 Applications of QA systems

- Customer Support using Bots (Lyft, Mastercard, Door Dash)
- Query Search (Google, Bing)
- Smart Home & Entertainment (Alexa, Google Home)

### 2.7.2 BERT’s role in building QA systems

Question answering systems have become a popular application of BERT. SQuAD is a benchmark dataset that requires BERT to return the text "span" matching to the correct response, given a question and a text passage containing the answer [19].

### 2.7.3 BERT’s Input Format:

To input a QA task into BERT, we must include both the question and the reference text. The unique [SEP] token separates the two pieces of text.

Additionally, "Segment Embeddings" are utilised by BERT to distinguish the inquiry from the reference text. These are essentially two embeddings that BERT learnt (for segments "A" and "B"), which it adds to the token embeddings before passing them to the input layer.
2. BACKGROUND

Fig. 2.7.1: BERT for QA systems: Input Format

Question: What is Covid 19?
Context: The Covid 19 Pandemic, also known as coronavirus disease.
2.7.4 Start and End Token Classifiers

In QA task, BERT’s job is to highlight a ”span” of text that contains the answer and this is done by predicting the tokens that mark the start and end of the answer.

We pass the final embedding of each token in the text to the start token classifier. The start token classifier applies a single set of weights to every word (shown by the
blue "start" rectangle in the figure 2.7.2).

After calculating the dot product between the output embeddings and the start weights, the softmax activation is applied to generate a probability distribution over all of the words. We select the term with the greatest likelihood of being the start token.

This procedure is repeated for the end token, which has its own weight vector.

2.8 Data Augmentation

In data analysis, procedures called "data augmentation" are used to expand the amount of data by adding slightly changed versions of either existing data or brand-new synthetic data that is derived from existing data. It can serve as a regularizer and aid in lowering overfitting when a machine learning mode is being trained \[25].
CHAPTER 3

Related Work

Natural language processing is the most effective method for humans to get information during human-computer interaction. Because of this, Question Answering Systems (QAS) are of great importance. Instead of returning a sorted list of documents like most information retrieval systems do, question answering systems seek to retrieve expected responses to questions. The concept of question-answering systems demonstrates a notable improvement in information retrieval techniques, particularly in its capacity to access knowledge resources naturally by querying and obtaining accurate responses in few sentences [21]. In recent years, a lot of research has been conducted to improve the efficiency, robustness, and accuracy of question-answering systems to help expand the use of question answering systems to challenging domains such as clinics and finance.

Question answering systems have been in existence since 1960’s. In the past, different approaches have been used by QA systems to respond to user questions. Numerous domains, data bases, question kinds, and response structures have been covered by QA systems. In order to respond to inquiries posed in natural language, modern techniques receive and process data from various sources [21].

Beginning in 1960, systems that can answer questions in natural language were being developed. The “Imitation Game,” often known as the “Turing Test,” which enables human and computer communication over an interface, was created by Turing. In 1961, Green et al. built BASEBALL, a QA system that communicates information on American baseball leagues. Dates and places are returned by the system as responses. Woods proposed LUNAR, another well-known QA. Information on soil
3. RELATED WORK

samples was provided by LUNAR. LUNAR was able to answer questions asked by humans with 90% accuracy. In the 1990s, Natural Language Interface to Databases (NLIDB), a QA system developed by Androutsopoulos et al gave users the ability to present queries in natural language and get answers from databases [5]. These systems, which relied on straightforward pattern matching approaches, performed admirably, but they were constrained by the lack of sufficient domain information in their repositories.

With improvements in technology, research based on question answering systems grew and QA systems began performing language analyses on the questions that were being asked to identify the intended inquiry requirements in a natural way. The mapping approach also included the use of semantic and statistical measures [14].

One of the biggest issues in modern information retrieval today is finding accurate answers to inquiries posed in natural language. The development of question-answering systems took diverse paths based on three primary characteristics due to the exponential expansion of digital documents and the ongoing specialisation of subject expertise. The first component has to do with the source of the answers, which might be either organised or unstructured databases. The second dimension relates to the application domain that is being studied, which can be either open (e.g. news, cross-domain encyclopedias) for an overview of the methodologies and applications utilised in restricted domains or specific ones (examples include medicine, sports, and the arts). The complexity of question analysis techniques (such as shallow/deep natural language processing, statistical methods, and semantic methods) is related to the third dimension [1].

As computation power and storage capacities enhanced over time – developing effective question answering systems became easier with deep learning. In 2018, researchers at Google AI concocted a Transformer-based model famously known as BERT. BERT has grown significantly in popularity among academics since its debut. It has been utilised for several NLP applications, including text classification for question answering. Results on BERT’s performance in the Stanford Question Answering Dataset challenge (Rajpurkar et al., 2016), where the system had to forecast
the response span for a particular question in a Wikipedia passage, were included in the original study by Devlin et al. (2018). Yang et al. (2019) took the study a step further by developing a chatbot that can answer questions [31]. In 2021, Alzubi et al were able to achieve 87.3 F1-score for covid question answering system using BERT [4]. Question answering systems have paved their way into various domains with the advent of BERT.

Despite the amelioration of question-answering systems using advanced technologies such as deep learning; they lack of generalizability [23]. Thus, it has been a concern to improve the robustness of question-answering systems. Plethora of research is being carried out to enhance the robustness of question answering systems using paraphrasing. Researchers are paraphrasing questions and context in a dataset to generate adversarial (negative) examples for improving the robustness of question answering systems and to help models generalize well by inserting challenging samples in the dataset. In 2016, Rajpurkar et al created a dataset widely known as SQuAD. This dataset is a reading comprehension dataset that contains more than 100,000 questions based on Wikipedia articles. Each question has its answer present in the corresponding context. SQuAD is used as a benchmark dataset for evaluating the performance of question answering systems [23]. In this thesis, we will be using SQuAD 2.0 for our experiments (more details are discussed later).

To examine how well QA models withstand question paraphrasing, in [12], Gan and Ng created two test sets comprised of paraphrased SQuAD questions. While the paraphrased questions from their first test set are similar to the original questions produced to test QA models’ over-sensitivity, questions from their second test set are paraphrased using context terms close to an incorrect answer option in an effort to confuse QA models. After putting the paraphrased test sets through state-of-the-art question-answering models like BERT, DrQA, and BiDAF, they noticed a decline in performance.

In [24], Rychalska et al have used model explainers to validate the robustness of question answering systems by proposing two question manipulating techniques. In the first technique, they paraphrase the question by replacing the important word in
a question with its semantically correct synonym from Wordnet. In the second tech-
nique, they replace the keyword in a question with its closest word vector produced
in embedding space. To determine the keyword in a question, they use LIME model
(Locally Interpretable Model Agnostic Explanations method). They have tested the
latest question answering models at that time and claimed that these models are
sensitive to the changes in the input.

Geusau and Bloem have used unsupervised learning to explore the capability of the
state-of-the-art question answering models in recognizing two paraphrased questions.
In [3], the authors have constructed an annotated paraphrased evaluation set known
as Para-SQuAD that consists of various paraphrased question pairs from SQuAD
dataset. They compared their results with an existing paraphrased dataset called
Dev-Para and notice that paraphrased dataset tends to confuse question answering
models leading to a decline in their performance.

In 2017, Jia and Liang generated adversarial sentences which are closely related
to the original question present in dataset and some random sentences taken from
contexts in the dataset. Moreover, they appended these sentences at the end of the
original paragraph to confuse the QA system and have tested their technique on
SOTA model such as LSTM and BiDAF. In [15], the authors have given a metrics
for evaluating the accuracy of prediction and quality of dataset containing adversarial
examples (Adv (f)) where ‘A’ is an adversary (negative instance), (p, q) is paragraph-
question pair, ‘a’ is the true answer and ‘f’ is the model and ‘v’ is the F1-score between
true answer ‘a’ and the predicted answer f(p,q).

Adversary Accuracy [M] formula:

\[
\text{Adv}(f) \overset{\text{def}}{=} \frac{1}{|D_{\text{test}}|} \sum_{(p, q, a) \in D_{\text{test}}} v(A(p, q, a, f), f)
\]

While Jia and Liang chose to insert adversaries using random sentences in the
original context, Clark and Gardner established a pipeline to train the models by
sampling multi-paragraphs using TF-IDF. Furthermore, they calculated the confi-
dence ratings for their outputs on the individual paragraphs to provide the output
3. RELATED WORK

that is globally correct. This pipeline was combined with SOTA models for document question answering data. Their work performed better than previous system by achieving an F1 score of 71.3 [9].

Researchers in [32] constructed a new dataset called PAWS (Paraphrase Adversaries from Word Scrambling), which contains 108,463 lexically overlapping pairs of well-formed paraphrases and non-paraphrases. The SOTA language models such as BERT was used to test the dataset and the models attained an accuracy of 91.9 after pre-training and fine-tuning. Pre-training refers to training a model on a task or dataset and later, training another model or task by making use of the previous task’s parameters. On the other hand, fine-tuning refers to adjusting a model’s parameters to fit properly to the required observations.

In [23], the authors developed SQuAD v1.1 that didn’t contain samples with unanswerable questions. To improve the quality of the dataset, Rajpurkar et al constructed a new version of SQuAD (SQuAD v2.0). It contains a combination of previous SQuAD data and 50,000 adversarial unanswerable questions. SQuAD 2.0 is more powerful than version 1.1. In [22], it is shown that SQuAD 2.0 achieves 66% F1 on a strong neural system whereas SQuAD 1.1 attains 86% F1 on the same model.

Inspired by the works in [15] and [30], we will be using the ‘insertion’ technique for generating adversarial examples in the present thesis work. In [30], Wang and Bansal developed an algorithm by producing adversarial examples for improving robustness of QA systems. Their work introduces a method to improve on capabilities that can help the SOTA models learn semantic relationship between sentences, and it shows why random insertions to the original context are better than the insertions at the front and the end of the context.

In this thesis, we will be generating adversarial examples by inserting paragraph having the highest similarity to the original context where the paragraph to be inserted doesn’t contain the answer to the question associated with original context. To do so, we will be using SQuAD 2.0 dataset for creating adversarial examples for data augmentation.
CHAPTER 4

Method

In this thesis, we create adversarial examples by replacing the original context with the most similar context in the dataset; under the condition that the most similar context doesn’t contain the answers to questions corresponding to the original context. To construct these adversarial examples, we use BERT sentence embeddings for transforming the text to vector embeddings and then we apply cosine similarity to measure the similarity scores for all contexts with each other. Furthermore, we compare the robustness of BERT and Recurrent neural networks to this type of negative examples and the potential improvement of their performance when these negative examples are augmented into the dataset.
4. METHOD

Flowchart for constructing Adversarial Examples

4.1 Data Acquisition

SQuAD v2.0

We used Stanford Question Answering Dataset (SQuAD) v2.0 to perform all the experiments in this thesis work. The Stanford Question Answering Dataset (SQuAD)
4. METHOD

SQuAD is a reading comprehension dataset consisting of questions presented by crowd workers on a set of Wikipedia articles, where the response to each question is a span of text from the appropriate reading passage, or the question may be unanswerable. In addition to 100,000 questions from SQuAD v1.1, SQuAD v2.0 contains 50,000 unanswerable questions which were also raised by crowd workers to resemble answerable questions.

The sole purpose of using SQuAD v2.0 over v1.1 for this thesis is that v2.0 contains unanswerable questions and for this work, we construct unanswerable examples too. Also, for a question answering system to be robust, we require that it distinguishes between questions that can be answered and questions for which an answer doesn’t exist. The SQuAD dataset contains 100,000 answerable questions and over 50,000 unanswerable questions. So, the ratio of answerable to unanswerable is 2:1. Thus, it is an imbalanced dataset.

SQuAD v2.0 is available in [22]. The train and dev sets of this dataset can be downloaded in JSON format. An example of the SQuAD 2.0 JSON format is shown in the figure below. The dataset contains contexts with questions and answers corresponding to it. For unanswerable questions, empty answer strings are present along with plausible answers.

In the data we are using, we do not work with questions having multiple answers. We work with questions having single answers only.

In this section, we discuss the construction of Adversarial Examples in detail –

For constructing adversarial examples from SQuAD 2.0 dev set, we used the following steps:

1. **Data Extraction:**

To construct adversarial examples, we make use of the SQuAD v2.0 dev set. There are 20302 examples in the dev set. After the successful extraction of contexts, questions, and ground truth answers from the JSON file; we create a data frame using Pandas module in Python. We stored contexts, questions, and answers under ‘Context’, ‘Question’ and ‘Ground Truth Answer’ respectively.
2. Generating Sentence Embeddings using BERT for all the contexts in SQuAD 2.0 dev set:

Processing textual data is a complex task for a machine, therefore, we construct embeddings from this text that can be easily processed while training language models. In machine learning, embeddings are mappings that help represent textual data in the form of vectors with continuous numbers.

To generate these embeddings, we made use of pre-trained BERT sentence embeddings. Embeddings generated by BERT are semantically rich as they are obtained from a model trained by large corpus. These pre-trained embeddings can be loaded from the NLU module in Python by calling `embed_sentence.bert`. This will help us predict the embeddings for all the contexts present in dev set of SQuAD 2.0. Corresponding to each context, embeddings are stored in a new column in the data frame.

```
[ 0.05948206 -0.11724691 -0.03987756 -0.58164501  0.1342099 -0.48504344
  -0.12577671 -0.34770557 -1.68023694 -0.6439718 -0.19966338  0.41485828
  -0.41493434 -0.37842661  1.51631451 -0.27207646  0.186322 -0.45313516
  -1.20962667  0.22237681 -0.34325457 -0.22372353  0.33708712  1.51820922
  1.89103687 -0.59752589 -0.25214365  1.17584491  0.05004758 -0.47032747
  -0.87842321 -1.43074727 -0.82783014 -0.53107601  1.12466335 -0.92145914
  -0.37473938  1.35449636 -2.89612819 -0.94651949 -0.36360583 -0.44776109
  -0.11425381 -1.43565416  0.06306815 -0.96424925 -0.58502966  0.37327567
   0.4044531  0.10914193 -0.07579678  0.88447177  0.17526278  0.14754315
  -0.57857615 -0.15319192  0.09363051 -0.64697033  0.5833993  1.7074827
  -0.17679542 -0.80782521 -1.47028852 -0.38700616 -0.26878688  0.47988588
  -0.42567852  0.19812949  0.94991219 -0.16556123 -2.02167845  0.67576593
  0.50970919 -0.07131404 -0.34501073  0.55006135 -0.56005394 -0.30937782
  1.5014962  0.31172106 -0.0475575 -0.84212101 -0.41841974  0.58547914
  0.95797873 -0.04951639 -0.19084813  0.93859935  0.43756503 -0.70874411
  2.08336782 -0.02887911 -0.51775587  0.89229226 -0.19037201  0.32678831
  0.9151144 -0.15219748  0.7760092  0.33795741  0.64586985  0.3902588
  0.21050447 -0.89375818  1.30739522 -1.2226882  0.09815544 -0.06141401
  0.22208525 -0.48081326 -0.6013819  1.37125325  1.70034783 -0.6498529
  -0.487351 -0.21600977 -0.18389125 -0.45207754 -1.05887175 -0.06560733
  1.16389835  0.39585364  1.77558792 -1.09955955 -1.84117019 -1.2138021
  -1.53023756  1.23734975]
```

Fig. 4.1.1: Vector Embedding Representation for a context from SQuAD 2.0 dataset

3. Finding similarity scores for all contexts with each other

To find the pairwise similarity scores among all contexts, we use Cosine similarity. Cosine similarity is defined by the dot product of two vectors divided by their mag-
4. METHOD

It takes any value between -1 and +1. A score of -1 means least similar and +1 means highly similar.

\[ \text{similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||} \]

Fig. 4.1.2: Code Snippet: Calculating Context Similarity using BERT embeddings

We calculated cosine similarity between any pair of contexts using their sentence embeddings generated in the previous step. We created new columns with unique contexts as the column headers and stored similarity scores corresponding to each context as shown in figure 4.1.3 below.

Fig. 4.1.3: Similarity Values for a Context in SQuAD 2.0
Now, we have a data frame with contexts, questions, ground truth answers, and similarity scores between each pair of contexts.

In 4.3.1, the Context Embeddings Visualization is shown in 2D space. Each blue dot in the plot corresponds to the context in the dataset. This plot gives a visual understanding of how similar or how dissimilar all contexts are from each other. Practically, the above image is a gif file and if we hover over the dots, we can see which context it is.

4. Constructing adversarial examples

Here, we create adversarial examples by finding out the most similar context [C1] for each context [C] in the data set and then we replace it with the original one only if the similar context [C1] does not contain answers to questions corresponding to original context [C]. If C1 contains the answers, then we find the next most similar context to C and assign that to C1. We repeat the process till we find highly similar contexts with no answers corresponding to original context’s questions. This is done for all unique contexts in the data. We store all the adversarial examples in a data frame.
4. METHOD

Fig. 4.1.5: Code Snippet: Construct Adversarial Examples

5. Converting adversarial examples to SQuAD 2.0’s original JSON format

After successfully generating adversarial examples, we convert the data to SQuAD 2.0’s original JSON format which resembles the notation in the figure below. As all the adversarial examples created are unanswerable; we will keep the ‘if_possible’ variable in the file as ‘True’. The ‘if_possible’ variable is an object in the original SQuAD dataset which tells if a question is answerable or not. So ‘if_possible’ is ‘True’ it means answer exists and if it is ‘False’ then the question is ‘unanswerable’.

```python
# finds the context with highest similarity that does not contain answers
from pandas.compat import numpy

def find_value_column(ans, text):
    if ans in text:
        return True
    else:
        return False

onlyNumbers = inputData.select_dtypes(include=np.number)
scoreGreaterThanZero = onlyNumbers.where(onlyNumbers > 0)
similarity_df['hasNocontext'] = ''
for row in scoreGreaterThanZero.itertuples(index=True):
    answerSpecificToRow = similarity_df.loc[row.Index]["answer_text"]
    # print(answerSpecificToRow)
    descSortedScore = scoreGreaterThanZero.sort_values(by=row.Index, axis=1, ascending=False)
    # print(descSortedScore)
    for col in range(len(scoreGreaterThanZero.columns)):
        # print(col)
        context = descSortedScore.columns[col]
        isNo = find_value_column(answerSpecificToRow, context)
        if(isNo == False):
            similarity_df.loc[row.Index, 'hasNocontext'] = context
            break
```
Fig. 4.1.6: SQuAD 2.0: JSON format

In the figure 4.1.7, we have taken 10 random contexts from the SQuAD 2.0 dataset and displayed the number of contexts that have answers corresponding to each of these 10 context’s questions. Also, the plot displays the number of contexts that don’t contain answers to each of the 10 context’s questions.
4. METHOD

The above plot has the index number of context on x-axis and the number of answerable and unanswerable questions on y-axis.

We can observe from the above plot that for most of the contexts, the dataset contains answers to other context’s questions. Thus, to construct our adversarial examples, we replaced each context with a context that does not have answers to its questions.

4.2 Models used

To train and test our adversarial examples and to evaluate the performance of the state-of-the-art models on the constructed data; we worked with BERT and RNN models. In Chapter 3 we have already discussed these two language models in detail.
4. METHOD

Model 1 – BERT-base-uncased

To evaluate the performance of our adversarial data, we test it on the BERT-base-uncased model that was introduced in [18]. It is a model that is pre-trained on the English language. It uses masked language modelling (MLM) as discussed in Chapter 3. The model is uncased which means it does not differentiate between words like french and French. This model is trained on 110 million parameters.

Model 2- GRU

We made use of Gated Recurrent Units (GRU) to compare the results of BERT base model with a simple recurrent neural network that doesn’t use attention mechanism. GRU has been discussed in Chapter 3.

Model 3 – LSTM

To compare GRU results, we experimented with another RNN i.e, LSTM. LSTM has also been discussed in Chapter 3 in detail.

4.3 Evaluation Metrics

For evaluating the performance of the two models mentioned above; we will be using the following two metrics:

1. **Exact Match Score**: Exact Match is a binary measure (true/false) of whether the system output matches the ground truth answer exactly. This is a strict metric.

2. **F1-Score**: F1 is a less strict metric and is the harmonic mean of precision and recall.

\[
F1 \text{ score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]
4. METHOD

How EM score works in QA systems?

Suppose we have an example as follows:

<table>
<thead>
<tr>
<th>Question</th>
<th>Ground Truth Answer</th>
<th>Predicted Answer</th>
</tr>
</thead>
</table>

Table 4.3.1: Example

In the above example, EM score would be 0 because Ground Truth Answer has only 3 words whereas Predicted Answer has 7 words and according to the definition of EM score - it is a strict metric and all the words in Ground Truth Answer should overlap exactly with words in Predicted Answer. In the assumed example, we see that there are common words in true answer and predicted answer, yet, the EM score will be 0.

The drawback of EM score is that it is very rigorous. If predicted answer is completely same as true answer, only then EM score is 1. Therefore, it becomes necessary to evaluate the performance of QA systems using a metric that is less-strict and resembles human evaluation technique. This is where F1 score comes into the picture.

How F1-score works in QA systems?

The F1 score is a popular measure for classification problems that is commonly used in question answering systems as well. In this QA systems, it is calculated by comparing the individual words in the prediction to those in the True Answer. F1 score is a not as rigorous as the EM score as it calculates the word overlap in ground truth answer and predicted answer.
F1 Score can be calculated as follows for dataset containing both answerable and unanswerable questions -

\[
F1Score(Truth, Predict) = \begin{cases} 
1 & \text{if } Truth = Predict = \emptyset \\
0 & \text{if } Truth = \emptyset ; \text{Predict} \neq \emptyset \text{ Or vice-versa} \\
2 \times \frac{Precision \times Recall}{Precision + Recall} & \text{otherwise}
\end{cases}
\]

Precision is the ratio of words that exist in both ground truth and predicted answer to the total number of words in the predicted answer.

\[
Precision = \frac{TP}{TP + FP}
\]

where, TP = true positive and FP = false positive

Recall is the ratio of words that exist in both ground truth and predicted answer to the total number of words in the ground truth answer.

\[
Recall = \frac{TP}{TP + FN}
\]

where, TP = true positive and FN = false negative

**True positive**: Number of words or tokens that are common between the ground truth answer and the predicted answer.

**False positive**: Number of words or tokens that are present in predicted answer but not in the ground truth answer.

**False Negative**: Number of words or tokens that are present in the ground truth answer but not in the predicted answer.
4. METHOD

Fig. 4.3.1: Code Snippet: Computing EM and F1-Score

```python
def compute_exact_match(prediction, truth):
    return int(normalize_text(prediction) == normalize_text(truth))

def compute_f1(prediction, truth):
    pred_tokens = normalize_text(prediction).split()
    truth_tokens = normalize_text(truth).split()

    # if either the prediction or the truth is no-answer
    # then f1 = 1 if they agree, 0 otherwise
    if len(pred_tokens) == 0 or len(truth_tokens) == 0:
        return int(pred_tokens == truth_tokens)

    common_tokens = set(pred_tokens) & set(truth_tokens)

    # if there are no common tokens then f1 = 0
    if len(common_tokens) == 0:
        return 0

    prec = len(common_tokens) / len(pred_tokens)
    rec = len(common_tokens) / len(truth_tokens)

    return 2 * (prec * rec) / (prec + rec)
```

Understanding the code snippet in Figure 4.3.1 - The `compute_exact_match()` function returns `True` or `1` if text in predicted answer is equal to text in ground truth answer else it returns `False` or `0`.

The `compute_f1()` function splits the text in predicted and ground truth answer into tokens where tokens can be words, sub-words, or characters. For **unanswerable questions**, F1 score is 1 when the predicted answer and ground truth answer are both empty strings or their length is 0 else F1 score is 0.

For **answerable questions**, the function first checks if predicted and ground answer consist of common tokens. If there are no common tokens then F1 score is 0 else it calculates the precision and recall using the number of common tokens that are present in true and predicted answer, number of tokens in predicted answer and number of tokens in true answer. F1 score is returned using the harmonic mean of precision and recall.
In Chapter 5, an in-depth discussion of all the experiments and results we achieved by applying our method to existing models such as BERT, GRU and LSTM can be found.
CHAPTER 5

Experiments and Results

This chapter gives account of the results obtained from various experiments on the models and methods described in chapter 4.

For implementing the methodology and carrying out the experiments, we used the following:

- Programming Language: Python 3
- Jupyter Notebooks: Kaggle, Google Colaboratory
- GPU: Google Colaboratory
- RAM: High-RAM (32 GB)
- Python libraries:
  - Pandas
  - Scitkit-learn
  - Numpy
  - Seaborn
  - Matplotlib
  - NLU
  - PySpark
  - PyTorch
  - Transformers
5.1 Experiments using Model 1: BERT

BERT’s Performance on original SQuAD 2.0 dataset

Hyperparameters

For training the BERT model on various datasets, we used the following hyperparameters and treat them as optimal hyperparameters as we chose these after experimenting with different values:

- Number of Epochs: 3
- Learning Rate: 0.00005
- Optimizer: Adam
- Batch Size: 8

Training

The original SQuAD 2.0 dataset was given as input to BERT for training. The training data and development data were used to train the BERT model for 3 epochs, where each epoch is one full pass of the training data through the model. The performance of BERT was observed using training loss and validation loss. The training loss helps in determining how well the model is fitting on the training data whereas validation loss determines how well the model is fitting the new data. In the figure below we can see how the training and validation loss grow over 3 epochs. It can be observed that train loss is less than validation loss after finishing all the epochs which means the model is underfitting. Underfitting occurs when it becomes hard to accurately model the training data and thus causing large errors [6].

The reason for training BERT only on 3 epochs was due to the limited time and resources available to carry out the experiment. In Section 5.3, we use a subset of SQuAD and train BERT on 50 epochs. With the resources available to us, training a small dataset with more epochs was a feasible task.
5. EXPERIMENTS AND RESULTS

![Train and Validation Losses](image)

**Fig. 5.1.1: BERT: Training on SQuAD 2.0**

### Testing

Table 5.5.1 is a snippet of the data used for testing BERT’s performance on adversarial samples where ‘Replacing Context’ is the adversarial context which was generated by replacing the original context corresponding to the questions under ‘Question’ header. The column ‘answer_text’ has empty arrays as the question should be unanswerable. The empty array here denotes an unanswerable question. This notation is taken from the original SQuAD 2.0 dataset.

After successfully training the data on BERT, we used the model weights obtained after training to test the performance of BERT on unseen test data. The data used for testing was the adversarial data constructed for our work. BERT’s performance on the adversarial data was evaluated using the mean of F1 scores and mean of Exact Match scores.

The Exact Match and F1 scores achieved after testing on 10,421 data points are as follows:

- **Exact Match (EM) Score:** 0.001055608866711448

---

43
5. EXPERIMENTS AND RESULTS

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
<th>Replacing Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>In what country is Normandy located?</td>
<td>[]</td>
<td>The Normans had a profound effect on Irish culture and history after their invasion at Bannow Bay in 1169. Initially the Normans maintained a distinct culture and ethnicity. Yet, with time, they came to be subsumed into Irish culture to the point that it has been said that they became &quot;more Irish than the Irish themselves.&quot; The Normans settled mostly in an area in the east of Ireland, later known as the Pale, and also built many fine castles and settlements, including Trim Castle and Dublin Castle. Both cultures intermixed, borrowing from each other’s language, culture and outlook. Norman descendants today can be recognised by their surnames. Names such as French, (De) Roche, Devereux, D’Arcy, Treacy and Lacy are particularly common in the southeast of Ireland, especially in the southern part of County Wexford where the first Norman settlements were established. Other Norman names such as Furlong predominate there. Another common Norman-Irish name was Morell (Murrell) derived from the French Norman name Morel. Other names beginning with Fitz (from the Norman for son) indicate Norman ancestry. These included Fitzgerald, FitzGibbons (Gibbons) dynasty, Fitzmaurice. Other families bearing such surnames as Barry (de Barra) and De Brca (Burke) are also of Norman extraction.</td>
</tr>
<tr>
<td>When did Herve serve as a Byzantine general?</td>
<td>[]</td>
<td>A few years after the First Crusade, in 1107, the Normans under the command of Bohemond, Robert’s son, landed in Valona and besieged Dyrrachium using the most sophisticated military equipment of the time, but to no avail. Meanwhile, they occupied Petrela, the citadel of Mili at the banks of the river Deabolis, Gllavenica (Ballsh), Kanina and Jericho. This time, the Albanians sided with the Normans, dissatisfied by the heavy taxes the Byzantines had imposed upon them. With their help, the Normans secured the Arbanon passes and opened their way to Dibra. The lack of supplies, disease and Byzantine resistance forced Bohemond to retreat from his campaign and sign a peace treaty with the Byzantines in the city of Deabolis.</td>
</tr>
</tbody>
</table>

Table 5.1.1: Adversarial Examples
F1 Score: 0.027082120218831992

In existing literature, the F1 score and EM score are 76.70 and 73.85 respectively. The performance of BERT has dropped significantly when it’s tested on the adversarial examples. This is due to the fact that the adversarial examples are very challenging.

**BERT’s Performance on original SQuAD 2.0 with adversarially constructed examples**

**Training**

After experimenting with original SQuAD 2.0 dataset, we merged the adversarial examples with the original SQuAD 2.0 samples to augment the dataset. 70% of the total adversarial examples were appended to the train set of SQuAD 2.0 and the remaining 30% of the adversarial examples were added to the dev set of SQuAD 2.0.

The augmented dataset created after merging original and adversarial samples was given as input to BERT for training. The model architecture and hyperparameters were kept same as used for training original SQuAD 2.0 only.

The train and validation losses observed for the BERT model trained on original and adversarial SQuAD 2.0 dataset are shown in figure below. The model tends to underfit the data, same as in the previous model.
5. EXPERIMENTS AND RESULTS

Testing

The model weights obtained after training BERT on original and adversarial data were used to test BERT’s performance on the same set of adversarial examples that were used to test the previous model and the evaluation metrics retrieved are as follows:

Exact Match (EM) Score: 0.11807765754296626

F1 Score: 0.11807765754296626

5.2 Experiments using Model 2: Gated Recurrent Units

Hyperparameters

For training the GRU model on various datasets, we trained the model using combinations of the various values of the hyperparameters. The 2 different optimizers
that were tested are - Adam, RMSprop. RMSprop was used for training LSTM and GRU in [18]. We tried different learning rates as well. We trained using the following values - 0.001, 0.003 and 0.005. In [18], 0.001 was used. For our experiments, we used the following hyperparameters and treat them as optimal hyperparameters as we chose these after experimenting with different values:

- Number of Epochs: 50
- Learning Rate: 0.001
- Optimizer: Adam
- Batch Size: 512

**GRU Performance on original SQuAD 2.0 train and dev sets**

We trained GRU on the hyperparameter setting used in [18].

[18] uses the following hyperparameter setting:

- Optimizer = RMSprop
- Learning rate = 0.001
- Batch Size = 512

The model was trained on SQuAD 2.0’s train and dev set [22]. In the plot below, we can see GRU’s performance on SQuAD 2.0.
In [18], the model's evaluation was done using dev set only whereas in this thesis, we used test sets to evaluate the model's performance. The F1 score and Exact match score achieved using hyperparameter setting in [18] are as follows:

- Exact Match (EM) Score: 0.18164518640256033
- F1 Score: 3.197935497942915

**GRU Performance on original SQuAD 2.0 dataset**

**Training**

Bidirectional GRU was used to train original SQuAD 2.0 dataset. The reason for using RNN to train SQuAD dataset was to compare BERT’s results with a simple bidirectional RNN that does not use attention mechanism like BERT does.

In the plot below, the blue color represents ‘train loss’ and green color represents ‘validation loss’.

---

![Training and validation loss](image)

Fig. 5.2.1: GRU: Training on SQuAD 2.0 based on [18]
Fig. 5.2.2: GRU: Training on SQuAD 2.0

The plot shows that train loss decreased over 50 epochs but validation loss tends to increase during the 50 epochs which is a sign of underfitting.

Testing

The F1 score and Exact match score achieved after testing SQuAD 2.0's adversarial examples using GRU model’s weights are as shown below.

Exact Match (EM) Score: 0.0027679266499437766  
F1 Score: 0.03355265482576688

In existing literature, the F1 score and EM score are 3.19 and 0.18 respectively [18]. The performance of GRU has also dropped significantly when it’s tested on the adversarial examples.
GRU Performance on original SQuAD 2.0 with adversarially constructed examples

Training

Furthermore, we used GRU to train the merged dataset consisting of original SQuAD 2.0 samples and adversarial samples. The same 70:30 ratio was used to split the adversarial samples into train and dev data. The model architectures were same as used for the previous RNN model. From the figure below, it can be observed that the model performs very similar to the situation when trained on original SQuAD data only.

![Training and validation loss graph](image)

Fig. 5.2.3: GRU: Training on SQuAD 2.0 + Adversarial Examples

Testing

The F1 score and Exact match score achieved after testing SQuAD 2.0’s adversarial examples using GRU model’s weights are as shown below.
5. EXPERIMENTS AND RESULTS

Exact Match (EM) Score: 0.0024219358187008044
F1 Score: 0.03519823756536785

5.3 Experiments using Model 3: LSTM (Long Short-Term Memory)

Training

Bidirectional LSTM was used to compare the results of GRU. The hyperparameters were kept same as used in GRU model. From the Epoch v/s Loss plot, we can see that LSTM’s performance is very similar to that of GRU when trained on original SQuAD 2.0 data. The train loss decreases but the model does not perform well on validation set. Hence, there is an increase in validation loss.

Fig. 5.3.1: LSTM: Training on SQuAD 2.0
5. EXPERIMENTS AND RESULTS

Testing

The F1 score and Exact match score achieved after testing SQuAD 2.0’s adversarial examples using LSTM model’s weights are as shown below.

- Exact Match (EM) Score: 0.0032869128968082345
- F1 Score: 0.03542420941865831

In existing literature, the F1 score and EM score are 3.19 and 0.18 respectively [18]. Similar to GRU, the performance of LSTM has declined when it’s tested on the adversarial examples.

LSTM Performance on original SQuAD 2.0 with adversarially constructed examples

Training

Furthermore, we used LSTM to train the merged dataset consisting of original SQuAD 2.0 samples and adversarial samples. The same 70:30 ratio was used to split the adversarial samples into train and dev data (same as GRU). The model architectures were same as used for the previous RNN model. From the figure below, it can be observed that the model performs similarly to GRU.
5. EXPERIMENTS AND RESULTS

5.3.2 LSTM: Training on SQuAD 2.0 + Adversarial Examples

Testing

The F1 score and Exact match score achieved after testing SQuAD 2.0’s adversarial examples using LSTM model’s weights are as shown below.

Exact Match (EM) Score: 0.00259493123432229 F1 Score: 0.036807339830770336

5.4 Experiments using Subset of SQuAD 2.0 and adversarial examples on BERT

SQuAD 2.0 is a large dataset containing more than 150,000 samples. After adding 20,302 adversarial examples, the total count goes up to 170,302 samples which makes it very hard to run models such as BERT for many epochs. Thus, to overcome this problem, we used a subset of original SQuAD 2.0 dataset. We extracted 10,000 samples from the original dataset randomly and used them along with adversarial examples to train BERT for 50 epochs.
5. EXPERIMENTS AND RESULTS

Training

For training BERT on this small dataset, we used the following hyperparameters.

Hyperparameters

Number of Epochs: 50
Learning Rate: 0.00005
Optimizer: Adam
Batch Size: 8

The results attained after training the subset of SQuAD 2.0 and adversarial examples on BERT are shown in the plot below. The training loss becomes consistent after 30 epochs, but the trend followed by validation loss is very irregular and it fluctuates a lot over 50 epochs. These fluctuations are a result of training a huge model like BERT on very little data.

Fig. 5.4.1: BERT: Training on Subset of SQuAD 2.0 + Adversarial Examples
5. EXPERIMENTS AND RESULTS

Testing

The F1 score and Exact match score achieved after testing SQuAD 2.0’s adversarial examples using BERT model’s weights are as shown below.

Exact Match (EM) Score: 0.18125397835773394
F1 Score: 0.18125397835773394

The evaluation metrics are better than when BERT was trained on full SQuAD but the results may vary based on the subset that is used for training. This may be misleading as the training subset and testing samples were randomly chosen. It might be a case where training was done on samples that are also present in testing set, thus, giving better results than other models.

5.5 Model Performance on Test Set: Summary
## 5. EXPERIMENTS AND RESULTS

<table>
<thead>
<tr>
<th>Training Data Used</th>
<th># Of Training Samples in Data</th>
<th>Model Used</th>
<th>F1 Score</th>
<th>Exact Match Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original SQuAD 2.0</td>
<td>150,000</td>
<td>BERT</td>
<td>0.027082120218831992</td>
<td>0.001055608866711448</td>
</tr>
<tr>
<td>SQuAD 2.0 + Adversarial Examples</td>
<td>170,302</td>
<td>BERT</td>
<td>0.11807765754296626</td>
<td>0.11807765754296626</td>
</tr>
<tr>
<td>Original SQuAD 2.0</td>
<td>150,000</td>
<td>GRU</td>
<td>3.197935497942915</td>
<td>0.18164518640256033</td>
</tr>
<tr>
<td>SQuAD 2.0 + Adversarial Examples</td>
<td>170,302</td>
<td>GRU</td>
<td>0.03519823756536785</td>
<td>0.0024219358187008044</td>
</tr>
<tr>
<td>Original SQuAD 2.0</td>
<td>150,000</td>
<td>LSTM</td>
<td>0.03542420941865831</td>
<td>0.0032869128968082345</td>
</tr>
<tr>
<td>SQuAD 2.0 + Adversarial Examples</td>
<td>170,302</td>
<td>LSTM</td>
<td>0.036807339830770336</td>
<td>0.00259493123432229</td>
</tr>
<tr>
<td>Subset of SQuAD 2.0 + Adversarial Examples</td>
<td>30,302</td>
<td>BERT</td>
<td>0.18125397835773394</td>
<td>0.18125397835773394</td>
</tr>
</tbody>
</table>

Table 5.5.1: Models’ Performance: Summary
CHAPTER 6

Conclusion and Future Steps

6.1 Conclusion

In this work, we focused on constructing adversarial (negative) examples using SQuAD. The aim of our work was to improve the robustness of existing Question Answering systems by augmenting the original dataset with challenging and unanswerable samples.

As discussed earlier, SQuAD contains context, and for each context there exist questions and ground truth answers. For creating the adversarial examples, we used the data present in SQuAD 2.0’s dev set as SQuAD 2.0 contains unanswerable questions. SQuAD 2.0 was a reasonable choice as SQuAD 1.1 does not contain unanswerable questions. For our work, we required a dataset containing unanswerable questions as our work aimed to construct unanswerable adversarial examples. We generated the negative samples by replacing the original context ‘C’ in the data with a context that is highly similar to ‘C’ but it does not contain the ground truth answers for questions corresponding to ‘C’. After generating the adversarial examples, we added these samples to original SQuAD 2.0 dataset and performed experiments using 2 models namely BERT and RNN. All the results obtained after carrying out the experiments were reported in Chapter 5.

From the results we obtained, it can be observed that BERT when trained only on original SQuAD 2.0 and tested on adversarial examples only yielded an Exact Match score of 0.001 and F1 score of 0.03. In comparison to the F1 score and Exact Match score obtained in existing literature where the evaluation metrics go up to
maximum value of 0.7 or 0.8, the performance of BERT on unseen adversarial examples dropped drastically. The drop in performance occurred as the training data did not contain samples of highly challenging adversarial examples. To overcome this, we carried out an experiment where we merged the original SQuAD data with the adversarial examples and then re-trained the BERT model to obtain model weights. The F1 score and Exact Match score achieved after re-training on original and adversarial examples are 0.12 and 0.12 respectively.

The results are significantly better than the one when trained only on original SQuAD data. One important point to note here is that the evaluation metrics are extremely low when compared to BERT’s performance on existing data. From this observation, it can be concluded that the adversarial examples constructed in our work are highly challenging, thus, making it hard for BERT to return correct answers.

Moreover, we also investigated how simple bidirectional recurrent neural networks such as LSTM and GRU would perform on the adversarial examples. We experimented with both these RNNs to compare the performance with that of BERT, which is based on attention mechanism unlike bidirectional RNN. To draw conclusion from the results shown in Chapter 5, it can be noticed that RNN model performs even worse than BERT in both the cases i.e., when trained with original SQuAD and with both original and adversarial SQuAD.

Furthermore, it can also be noted that the adversarial examples generated in our work by replacing the original context in dataset with most similar context that does not contain answers corresponding to original context’s questions are extremely challenging instances for both BERT and RNN models. The performance of both models dropped when compared to the results presented in existing literature. The goal of our work was to construct adversarial samples that are challenging enough to enhance the robustness of existing question answering systems.

To sum up, we can state that the adversarial examples constructed in our work are very challenging and such samples should be added to the original question answering datasets to make the systems more robust and help them generalize well on the data rather than overfitting on the data.
6.2 Future Steps

At present, Question answering systems are a popular research topic and ample amount of research is being conducted to make these systems highly efficient and robust. There are various areas with scope of improvements. This research can also be extended further in many ways and one of them is to study the effect of similarity scores on the performance of language models. To elaborate on the previous statement – while constructing adversarial examples, instead of replacing the original context with most similar context, we can try to replace them with less similar context having similarity scores of 0.5 or below. This study can help us examine if contexts with low similarity scores help language models answer the questions with high accuracy or not.
REFERENCES


VITA AUCTORIS

NAME: Vishakha Gautam
PLACE OF BIRTH: New Delhi, India
YEAR OF BIRTH: 1997
EDUCATION:

Bhagwan Parshuram Institute Of Technology, B'Tech in Computer Science, New Delhi, India, 2019

University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2023