Lies & Deceptive Text Detection

Master Thesis

To obtain the degree Master of Science (M.Sc.)
in the program Web and Data Science

Submitted by
Hisham Parveez

First examinier: Prof. Dr. Martin Prause

Second examinier: Prof. Dr. Martin Friskle

Koblenz, in October 2023

The technology you use impresses no one. The experience you create with it is everything

Sean Gerety

Contents

Figures

Abbreviations

NLP Natural language Processing
AI Artificial Intelligence
ML Machine Learning
DL Deep learning
RL Reinforcement learning
tf-idf Inverse Document frequency
tf Term Frequency
GloVe Global Vectors
NN Neural Networks
ANN Artificial neural networks
BP Backpropogation
SGD Stochastic Gradient Descent
LSTM Long Short-Term Memory
BiLSTM bidirectional Long Short-Term Memory
GRU Gated Recurrent Unit
CBOW Continous bag of words
ELMo Embeddings from Language Models
QnA Question and Answer
BERT Bidirectional Encoder Representations from Transformers
GPT Generative Pre-trained Transformer
RoBERTa Robustly optimized Bidirectional Encoder Representations from Transformers
XLM Cross-lingual Language Model
CG Computer Generated
ASR Automatic Speech Recognition
ST Speech Translation
TTS Text-to-speech

Zusammenfassung

Soziale Medien spielen eine entscheidende Rolle dabei, Menschen weltweit zu verbinden und zwischenmenschliche Verbindungen zu fördern. Die enorme Reichweite von Social-Media-Plattformen bedeutet, dass alle über sie verbreiteten Informationen das Potenzial haben, ein beträchtliches globales Publikum zu beeinflussen. Folglich finden sich Einzelpersonen häufig wieder auf potenziell irreführende Aussagen stoßen, sei es in Form erfundener Nachrichten, voreingenommener Produktbewertungen oder Lügen über verschiedene Ereignisse und Aktivitäten. Um dieser Herausforderung zu begegnen, haben sich mehrere Studien auf den Bereich der automatisierten textbasierten Täuschungserkennung gewagt und dabei die Fähigkeiten von genutzt Techniken der Verarbeitung natürlicher Sprache (NLP). Ein wesentliches Hindernis bei diesem Unterfangen ist jedoch die inhärente Schwierigkeit, die zugrunde liegenden Gründe für irreführende Aussagen zu entschlüsseln und zu verstehen.
Jüngste Durchbrüche in der Verarbeitung natürlicher Sprache (NLP), insbesondere die Weiterentwicklung transformatorbasierter Modelle, haben neue Möglichkeiten eröffnet. Angesichts dieser Entwicklungen geht es in dieser Arbeit darum, die Kraft der Feinabstimmung zu nutzen Transformer-basierte Modelle für gezielte Ziele, beispielsweise die Erkennung von irreführenden Aussagen. Um dieses Ziel zu erreichen, untersucht die Arbeit eine vielschichtige Ansatz. Es schlägt die Verwendung einer Klassifizierungsarchitektur für überwachte Systeme vor Lernen, eine auf Generative Adversarial Network (GAN) basierende Architektur für halbüberwachtes Lernen und eine chatbasierte Frage-und-Antwort-Architektur (QnA). um Probleme im Zusammenhang mit Lügen und Fehlinformationen wirksam anzugehen. Diese vielschichtige Strategie nutzt die Vielseitigkeit und Robustheit von Transformer Modelle, die eine umfassende Erforschung ihres Adressierungspotenzials ermöglichen verschiedene Herausforderungen im Zusammenhang mit irreführenden Informationen.

Abstract

Social media plays a vital role in connecting people worldwide and fostering interpersonal connections. The vast reach of social media platforms means that any information circulated through them carries the potential to influence a substantial global audience. Consequently, Individuals find themselves frequently encountering potentially deceptive content, Whether in the form of fabricated news, biased product appraisals, or lies regarding various events and activities. While addressing this challenge, Several studies have ventured into automated text-based deception detection using traditional Machine learning algorithms, Recurrent neural networks and recently introduced Transformer models [35].

Recent breakthroughs in Natural language processing have introduced more deceptive content which is even undetectable at human level [50]. The significant obstacle in this scenario is the inherent difficulty in deciphering and comprehending the changing underlying logic behind deceptive context. This thesis will focus on utilizing various Transformer models and fine-tuning them for the target objective of detecting misinformation and deceptive content. The thesis will explore different scenarios with utilization of Linear classifier for supervised learning and Generative Adversarial Network classifier for semi-supervised learning for deceptive statements and finally Question and Answer classifier to tackle lies and misinformation content. These approaches will capitalize on the versatility and robustness of Transformer models, Enabling a comprehensive exploration of their potential in addressing various challenges related to deceptive content.

1 Introduction

1.1 Motivation

The paper [50] analyses content purchased by different private groups mainly related to product reviews. The paper concludes that the reviews were mainly used to boost the rating manipulation and was mostly used by low-quality products. While the reviews were human generated. However with advancements in NLP, The computer generated deceptive content is easily available and becoming popular in the form of fake news, product reviews etc. it is crucial to understand and explore different learning scenarios and achieve a generalized adaptive learning for deceptive content.
The conventional practice of employing RNN, particularly the LSTM architecture, remains widely favored to tackle various problems in time series, NLP, computer vision etc. while Transformer model which are predominately used for NLP tasks such as text summarizing, question and answering, text generation showcased recently by ChatGPT which uses a variant of transformer model called GPT-3.5 (Generative Pre-trained Transformer). Nevertheless, recent findings suggest that transformer based models offer superior efficiency, especially when trained on extensive datasets [1]. However, The predominant advantage of Transformer-based models like BERT over LSTMs lies in their unparalleled ability to grasp the contextual semantics of words.

Consider, for instance, two sentences that both contain the word "fine." Despite the shared word, the context in which "fine" is applied in each sentence is interpreted different.

I have fine hair
I am fine

Traditional LSTM essentially learn from left to right contexts which could result in loss of true contextual meaning, While a transformer based language model BERT uses the concept of bidirectional where it captures meaning of the input sequence from both directions ensuring that the genuine context is retained.
It's important to distinguish between static word embeddings, such as those used in word2vec and GloVe, and the dynamic word embeddings employed by BERT. In the former, each word is associated with a single vector representation, which remains constant regardless of context. While BERT utilizes dynamic word embeddings, where the input is an entire sentence rather than an isolated word. BERT's ability to consider the context of surrounding words before generating a word vector is a crucial advantage in detecting deceptive content [2].

1.2 Research Questions and Objectives

1.2.1 Performance of Transformer Models

One of the objective of this thesis revolves around the utilization of transformer models that are subsequently fine-tuned for specific tasks, such as the classification of computer-generated deceptive reviews. The primary aim here is to harness the potential of these models, with the expectation that they can deliver enhanced accuracy when compared to some of the traditional ML algorithms and RNNs.
Since the transformer models used for the thesis are compact or smaller models, This would also showcase the importance of fine-tuning which is one of the important step in language model's performance and also showcase that smaller models can be resource efficient and provide better results in some scenarios in comparison to some of the large models available.

1.2.2 Transformer Models in semi-supervised learning

The area of interest is mainly related to Transformer models capabilities to detect computer generated deceptive content. While this is main concern, However the lack of good quality datasets related to computer generated text is scare. The other major reason is the labeling of dataset, Which is a tedious task.
Hence the research into fine-tuning Transformer models and measuring their performance in a semi-supervised learning scenario, Where the models are trained on small percentage of labelled dataset and large corpus of unlabelled dataset. The research also aims to establish the dynamics between the labelled data used for training against it's performance for each individual models.

1.2.3 Language Inference capabilities of Transformer Models

With the emergence of ChatGPT and Question-Answering (Q&A) systems, There has been a shift in generative capabilities of the language models. However it also opens up multitude of possibilities for applications ranging from fact-checking to information validation, Enabling AI to play a more discerning role in our interactions with any textual content. Hence this research question explores the language inference capabilities of the Transformer models where for the given context, The model tries to classify as the asked question is truthful or lie.

1.3 Course of Investigation

The Entire thesis consists of six main chapter. The introduction (Chapter 1) followed by Theoretical background (Chapter 2) which provides overview on NLP, deep Learning, various types of language models and delves deep into transformer architectures and the types of word embeddings. Chapter 3 focuses on the related research done in the area of deceptive content and their comparisons with transformer architectures. Chapter 4 gives an Overview about the Approaches implemented to answer research question and in depth analysis of the datasets used. Chapter 5 provides results and discussions of the outcome and Finally Chapter 6 concludes this thesis.

2 Theoretical Background

2.1 Natural Language Processing

Natural language processing(NLP) refers to the branch of AI which is specifically dedicated for improving computer's ability to understand textual data and spoken language words in a way humans can interpret and understand. NLP combines various interdisciplinary areas of computational linguistics mainly for modelling of human language along with probability and statistical approaches, with machine learning and deep learning models [3]. Combined together, These technologies aid machines in processing natural human language in the form of textual or audio format, facilitating their understanding of not only the literal content but also discerning the underlying meaning, intent, and sentiment conveyed by the writer or speaker.

2.1.1 Overview

The visualization of the key domains like AI, RL, ML and DL is critical in understanding the relation of NLP with specified domains. AI is a broader term that confine systems with a capability to capture the cognitive thinking. There are many different interpretations the AI that encapsulates the above domains. For the thesis, The interpretation of [4] is used as a reference of positioning NLP in the relation to AI. AI majorly consists of two areas, namely ML and DL, NLP is considered to be part of DL as shown in the Figure 2.1.

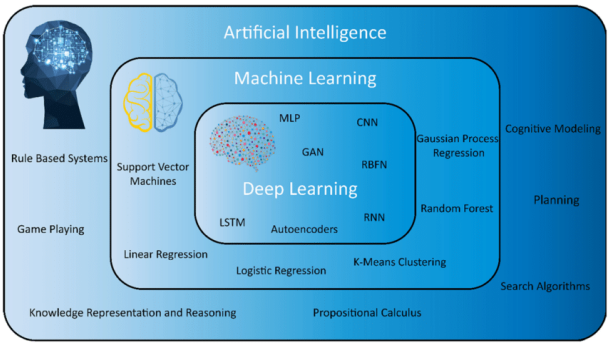

Figure 2.1 : AI Architecture with reference to ML and NLP, source:[4]

The Interpretation can have many different applications in understanding human understandable natural language. The areas of applications consists of speech recognition, machine translation, text classification, information retrieval etc.
The domain of NLP combines all the computer and human interactions by the means of

written and spoken natural language. It is a research area, which is concerned with the transformation and understanding of human natural languages. The processing of human language is based on understanding the intended meaning of the text, which would be challenging even for humans to understand and interpret. For instance, when the word "fine" is used in a sentence, All the components of the natural language such as syntax, pragmatics, semantics and morphology must be considered for interpreting and understanding the intended meaning of the sentence.
Semantics is focused in understanding the meaning of the sentences, each sentences are built using particular grammatical structure or syntax for the correctness of the sentences and morphological form of words. Morphology is about the architecture and the meaning of the words. While Pragmatics is used to get the intended meaning of the given sentence based on the overall context.
One famous definition of Machine Learning (ML) is based on the idea of experience and illustrates the learning part in ML:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."[6]

If we consider our system to be a medical testing application that predicts whether the person suffers from certain disease or not (T). The model used for the system would be a classifier, as the possible outputs would be "Yes"(1) or "No"(0) and its performance will be measured by its Accuracy or F1-score (P), which are some of the performance metrics. It learns from historic medical data (E) to predict the right outcome.

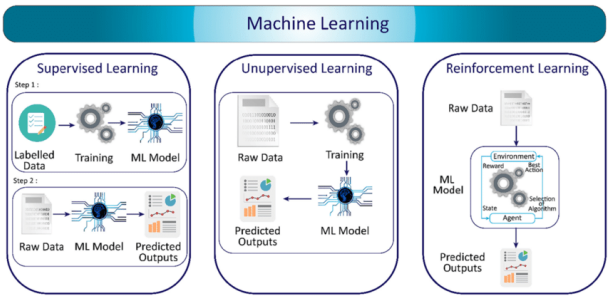

Figure 2.2 : Categorization of ML, source:[4]

Machine learning is considered to one of the central area of AI. It can be further classified into four sub domains namely, Unsupervised learning, Supervised learning and Reinforcement learning. Supervised and Unsupervised learning are majorly two types of

data mining problems. The difference lies in the structure of the training data. In supervised learning, the training data contains the target variable for each specific input and the model learning is based on inputs and target variables. While unsupervised learning do not contain target variable and the training data mainly consists of unlabelled data, the model's learning is based on identifying groups and patterns.
Reinforcement learning is similar to supervised learning, but the learning is not limited to the specific dataset. RL is lately being implemented in autonomous systems like self-driving cars. RL algorithms use trial and error to learn and obtain the objective for the given problem. Environment plays a crucial role in RL algorithms, all agent interactions with the environment is used as a feedback to improve the learning and experience of the agent.[7]

2.1.2 Feature Selection and Preprocessing

Both feature selection and preprocessing are crucial steps in AI, specifically in the context of NLP. These tasks hold a greater importance in data preparation phase as they control the effectiveness of the outcomes of the text analysis tasks like classification, sequence prediction etc [3]. The primary challenge is to understand unstructured and nature of natural language text. Machines cannot understand natural language and require structure and numerical data for processing. Hence the concept of word embeddings such as vector space, Glove were introduced to address the challenge [8].
The following sections provides an overview of different preprocessing and feature selection techniques for an english sentence "The best ship is sinking" as an example to illustrate the application of text preprocessing.

"The best ship is sinking"

2.1.3 Tokenization

One of the initial step for processing natural language text is dividing the sentence into smaller components as tokens. Traditionally, each token correspond to a basic word, which are the fundamental units of natural language. Typically a token can consist of idioms or special characters like hyphens. Tokenization is a process of converting continuous text into discrete text units and serves as an initial text preprocessing step.[9]
Apart from dividing a sentence into a smaller words or tokens, Tokenization depends upon the overall problem, for instance tokenization can be implemented to divide a context of natural language into entire sentences as tokens rather than a traditional approach

of tokenizing words by splitting based on whitespaces. However the straightforward approach has drawbacks, primarily because it doesn't distinguish tokens that are semantically linked [10]. Considering simple word tokenizer applied to the example sentence divides the sentence based on white spaces into five tokens.

"The", "best", "ship", "is", "sinking"

2.1.4 Stop Word Removal

Stop words in a natural language are considered insignificant and hence are removed for NLP preprocessing to minimize the given input and improve model's learning. Stops words in a natural language occur quite frequently and provide minimal information about the overall context of the sentence.[12] As a result they are removed for further analysis. Some of the common occurring stop words are "the", "are", "a" etc. Each natural language has a standardized set of stop words which are referenced for removal process from the given text context. One of the example for the NLP software package nltk consists of stop words library for common english language stop words [11].
Sometimes stop words removal could be inefficient, especially considering sentiment analysis where the objective is to understand the underlying intent of the speaker and classifying text as positive, neutral or negative. Here "not" plays a critical role in understanding sentiment, so for "not good" and "good" after stop word elimination might result in the same intent, hence misclassifying the sentiment of the speaker. for our example the tokens "The" and "is" are removed as a part of stop word removal.

"best", "ship", "sinking"

2.1.5 Stemming

Stemming is a method of mapping words to their word stem, or base root form which is also crucial in reducing input dimensions. The aim of stemming is to capture the meaning of the word and map it to to the stem word, which aids machines in better understanding of natural language text. For instance the words "fishy","fished","fisher" have the same stem word "fish". [14]
The initial stemming algorithm introduced in 1968 [13] focused on on shortening the longest suffixes while also accommodating spelling exceptions. However the recent approach strips suffixes of the words and retains the stem word. The approach works well for english language but is inefficient when applied to other languages like German as suffixes

are not added at the end of the word. For our example sentence, upon applying stemming for the tokens "best","ship" and "sinking" are assigned to the following words.[15]

$$best \rightarrow best, ship \rightarrow ship, sinking \rightarrow sink$$

2.1.6 Lemmatization

lemmatization is a process of determining the lemma of the word based on their intended meaning. Unlike stemming which truncates the prefixes for English words, lemmatization is superior and focuses on identifying the intended part of speech and understanding the meaning of the word for the given sentence as well as within the larger context surrounding the sentence [15].The process is based on morphological analysis and often uses a dictionary containing lemma of every modified word form which could be retrieved. As a result, the input dimension are reduced and aids machines in understanding the meaning of words better in complete context. When lemmatization is applied for our example sentence, we map the word "best" to its lemma, which is "good".[17]

$$best \rightarrow good, ship \rightarrow ship, sinking \rightarrow sink$$

2.1.7 Encoding

As machines do not posses capabilities to understand natural language, it is crucial to have a generic machine-understandable format to represent natural language text in numerical representation. There are several methods which are used to translate text to numerical representation, where statistical attributes of the words are critical and are used to represent the word.
The vector space model is an approach that translates a text into a vector, based on one-hot encoding. It was most commonly used in early information retrieval systems, where for a given textual context, a vocabulary vector of length N is created. One-hot-encoded vector represents each of the word which is unique for the context. For instance one-hot encoding of the word "ship" which is unique word in the corpus present in the i-th position, The vector representing "ship" would consists of 0 for all position except for i-th position which contains 1 [16].

$$one\text{-}hot\ (ship) = (0,.....,1,.....,0) \in N \qquad (2.1)$$

The vector space model is also used for documents for information retrieval, Here each document d is represented with a vector representation. The ordering of the n distinct words from w1...wn represents the value of the vector.

$$Q : d \rightarrow Q(d) = (tf(w1,d), tf(w2,d), ...tf(wn,d)) \in R \qquad (2.2)$$

Q counts the term frequency (tf) for each word in the vocabulary per document[16]. Hence the document vector computed can have multiple values 1 unlike one-hot encoding. The term frequency tf(w1,d) computes frequency of the word w1 for the document d. Apart from just focusing on term-frequency and returning d with highest count, each w1....wn in vocabulary can be given a weight, based on their relative appearance. Inverse Document Frequency (tf-idf) is divided using their respective term frequency, so the retrieval is based on word weight rather than just frequency. The tf-idf of a word in document is calculated using normalized term frequency. Eqn 2.3 is the normalized term frequency equation

$$ntfrel(wi, d) = \frac{tf(wi, d)}{\sum_{wm} tf(wm, d)} \qquad (2.3)$$

where each word frequency is calculated for i-th word wn in document d and is divided by sum of other word frequencies for same document d.
However the major drawbacks of Vector space model is the space complexity, where document vectors are computed where the size of vector would be N-length(d) where N is the vocabulary size. Another drawback which is related to dimensionality, where the distances between two document vectors might be very low, which could result in inefficiency when grouping similar documents.

GloVe is one of the recent word representations which is commonly used in NLP tasks. As the name suggests the main objective of the GloVe was to take in consideration the global knowledge of the corpus, where word co-occurrences is with respect to the entire corpus unlike previous representations where the context of the word is limited. It uses vectors to represent words and uses concepts of distances and directions which are computed for the vectors which provide better understanding of semantics and contextual relationships between the words [8].
Glove creates a co-occurrence matrix which takes in all the words in the corpus. X_{ij} is

computed for words wi and wj which indicates the the frequency of wi with respect to wj at global level and frequency of the word in relation to other i is given by X_i and the likelihood is given by the eqn 2.4.

$$P_{ij} = P(\frac{j}{i}) = \frac{X_{ij}}{X_i} \qquad (2.4) \quad [8]$$

2.2 Neural Networks

Neural Networks (NN) or Feedforward neural networks are the central concept of DL. NN are being implemented to solve problems in the area of computer vision, NLP, Time series etc. NN have been successful in providing good results due to their versatility to integrate for the given problem, As they are able to process structured and unstructured data such as audio, video etc. NN have been utilized for ML learning techniques of supervised and unsupervised tasks, as well as semi-supervised learning where training data consists of small percentage of labelled data and remaining being unlabelled [18].
The advent of backpropagation algorithm in the 1980's and it's application into NN, where BP based NN became prominent in the recent times with combination of optimization techniques, like SGD. Hence contributing to advancement in computing power of computers and NN implementations in different fields.[19]

A simple fully-connected neural network is mainly composed of an Input layer, hidden layers and output layer. A NN can have multiple hidden layers and each node is termed as neuron. Each of the neurons are connected to neurons of consecutive layer, each connection is assigned a weight. The connections are the critical components in enabling model's learning.[19]

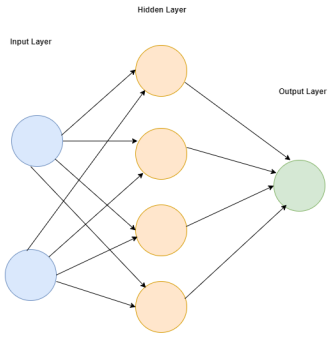

Figure 2.3: Simple fully-connected neural network architecture.

In the figure 2.3, blue neurons represent the input layer, where each input neuron is fully-connected to neurons of hidden layers represented by orange. In case of multiple hidden layers, The outputs of 1st hidden layer would be input to 2nd hidden layer. The green neuron represent the output layer. Each input is multiplied with the assigned weight W and a bias b is added at the end. The mathematical equation for it is given by $W^T x + b$

As discussed above, NN takes input and utilizes multiple hidden layers and activation functions are applied to each neuron, which generates a prediction for the given input. The training is done in iterations and multiple metrics are used to compute the error and evaluate the learning. NN are modelled similar to a human brain, and hence the intelligence gained for predictions is mainly identifying and recognizing patterns based on learning.

2.2.1 Hidden Layers

An NN consists of multiple layers with different usages. The input layer consists of the data inputs from the training data, while hidden layers are placed between the input and output layer as discussed above. However it critical to identify the no of hidden layers for learning, too simple or too complex NN can overfit or underfit the data. Overfitting is a scenario where the networks designed are too complex which results in high variance, Hence the learning of the network focused to solve a problem diminishes and network starts to to learn from random regularity in training data.[20]. Conversely Underfitting is a scenario where the model is too simple which causes low variance.

2.2.2 Weights

A strong connection between any two neurons is determined by the weights. As seen in the equation $W^T + b$, The inputs undergo multiplication by these weights thereby influencing the output of the receiving neuron. Subsequently activation functions are employed which adjust the data before transmission to subsequent layer. we will look into activation functions and it's types in depth in the following section.

2.2.3 Activation functions

Activation functions are crucial in understanding the network's complexity and learning features of the training data. As mentioned in [21], Some of the tasks of the activation functions are to improve network training, distribution of data during training and making

sure not to hinder gradient flow. The activation functions can be linear or non-linear, However their application is highly critical for the last layer of the network(output layer) which produces the end output. When we consider a binary classification tasks, Which consists of two outputs (0,1) The functions responsibility is to map the suitable output value in the range of 0 and 1.

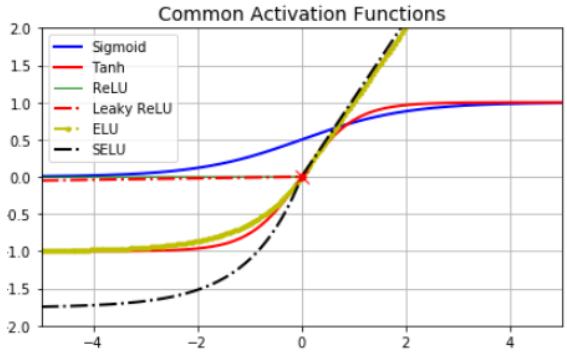

Figure 2.4 illustrates Multiple distinct functions [22]

The simplest activation function is the linear or identity function because the input z is multiplied by 1. The output is a continuous value:

$$Linear(z) = z \qquad (2.5)$$

The sigmoid function also known as "S"-shaped curve, Is one of earlier activation function which uses logistic function, The formula for sigmoid activation function is given in eqn (2.6). Sigmoid is a differential, and the value return is in range between 0 and 1. It's often implemented in classification tasks, However one of the major drawback of sigmoid is it's prone to vanishing gradient problem [23]. Where the learning is stagnant, as a result different activation functions were introduced to tackle the above problem.

$$sigmoid(z) = \frac{1}{1 + e^{-}z} \qquad (2.6)$$

hyperbolic tangent or tanh activation function is similar to sigmoid where the resultant graph is an "S"-shaped, However major difference lies in their value range. The tanh function returns value between -1 and 1. It is considered superior to sigmoid in NLP tasks, However similar to sigmoid it suffers from vanishing gradient problem [21]. tanh uses tangent hyperbolic function and the formula for tanh is given in eqn (2.7):

$$tanh(z) = \frac{e^{z} - e^{-}z}{e^{z} + e{-}^{z}} \qquad (2.7)$$

Softmax Activation function is a probabilistic function, where the output of each neuron is highly dependent on other neurons present in the same layer. As a result the sum of all the output values for neurons in the same layer is 1. Hence they are efficient for classification tasks involving multiple classes [23]. The formula for softmax function is given in eqn 2.8

$$Softmax(zi) = \frac{e^{zi}}{\sum_1^j e^{zj}} \qquad (2.8)$$

Rectified Linear Unit (ReLU) is one of the widely used activation function, mainly when implementing Deep NN because their efficiency in learning tasks contributing to faster learning. Unlike previous activation functions, ReLU is robust to vanishing gradient problem and is superior to previous functions. The concept of ReLU is quite simple, Non-negative output of neuron is returned. In case of negative returns, it is recast to 0. The formula for ReLU is given in eqn(2.9)

$$ReLU(z) = max(0, z) \qquad (2.9)$$

Leaky Rectified Linear Unit (Leaky ReLU) is an enhanced variant of simple ReLU, Unlike ReLU which returns 0 for negative values which can sometimes cause dying ReLU scenario because the neurons returning negative values are completed deactivated. However to tackle this problem Leaky ReLU multiplies a small value usually 0.01 to input and maximum of the values is returned [23]. The formula for Leaky ReLU is given in eqn 2.10

$$LeakyReLU(z) = max(0.01 * z, z) \qquad (2.10)$$

The above mentioned are some of the major activation functions, However there are multiple variants of ReLU and others which are used for different purpose.

2.2.4 Recurrent Neural Networks

RNN represent a category of NN applied across various domains that involve sequential data and poss capabilities to process sequences of data of varying lengths which is quite convenient when the problems are related to NLP. RNN unlike traditional NN which takes an input and produces an output, Each neuron in RNN gets two inputs. One being the actual input from previous layer and the other being the output of the neuron itself as showcased in the fig 2.5, The approach is quite similar to a recursion scenario in programming. Hence previous outputs of the neuron is crucial in determining the consecutive outputs. As previous states play a major role, RNN posses a resemblance of memory.[24]. However some of the major drawback is it's ability to store and form pattern for inputs which

were processed a while ago. For instance The sentences (a) "Pizza is an Italian food" and (b) "Pizza having pineapple as toppings isn't great, As normal Italian pizza don't have it". RNN might not capture from sentence(b) that Pizza is an italian food.

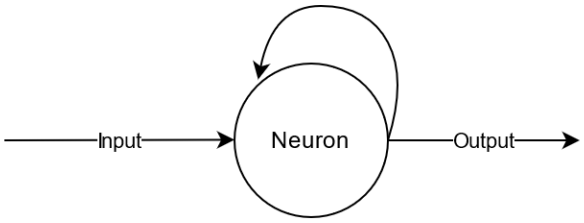

Figure 2.5 A Basic RNN Neuron Structure.

Long Short-Term Memory (LSTM) is a type of RNN but unlike traditional RNN, It uses a gated structure which enables LSTM to store information and recall from memory even after extend period of time, Hence it is considered to be more superior in comparison with traditional RNN. The unit of LSTM which acts as a memory is a logical gated structure consisting of mainly 3 gates. Input gate calculates the amount of information to be stored which it received from last input while forget gate as name indicates, Decides how much information from the previous learnings are considered to be irrelevant and finally the output gate decides the amount of information which is considered to be relevant and be sent to the next consecutive layer
Unlike traditional RNN which suffers from problems related to exploding and vanishing gradients, where the learning gradients either become too big or too small respectively affecting network's learning, LSTM's gated structure is robust to the gradient problems.

2.3 Language Models

Pictorial text data is used by CNN in processing text as shown in [25] a hybrid of CNN and RNN, However traditional CNN are used in a system which process pixel data to identify patterns from the training data and are specifically used in image processing while NN in NLP use raw natural language text as an input and RNN are better suited for these tasks. As discussed in section 2.1.7 (Encoding), Probabilistic approaches are implemented for processing natural language text. The main idea is to compute probability for a word wi with context to other words in a sequence w1....wn where the probability of word wi is highest to appear in context to other words on a sequence.

Hence numerical representation and probability computed for the words plays a major role in NLP, The vector representations of the tokens or words in the sequence are called

word embeddings. GloVe embedding [8] as showcased in the section 2.1.7 is an example of a word embedding. Variants of LSTM and GRU, GRU which is considered to be more superior in comparison to LSTM, were used to tackle the vanishing gradient scenario. However recent advancement in NLP gave rise to pretrained words embedding and language models followed by introduction of Transformers architecture which uses attention mechanism, which will discussed in detail in upcoming sections. Bidirectional Encoder Representations or simply BERT [26] was first of it's kind where the learning the context of words for a sequence is done from both directions simultaneously.

2.3.1 Masked Language Models

Masked language model (MLM) expertise lies in solving text classification problems which can be using supervised or semi-supervised learning. The MLM objective is to classify statements based on the labels. As the training data contains labels for the input text, The language model tries to identify and learn pattern between the labels and input by using the concept of masking. Masking is a technique where a token in each input is masked so that it improves model's capability to understand data better.
All the approaches used in this thesis tries to solve the classification problem of deceptive texts. Some of the MLM are opensource and hence the pretrained models are implemented to solve the problem by adding a custom layers and a output layer that in inline with no of labels present in the training data and the model is fine-tuned for the specific data set. However these models do not perform well in generative tasks like QnA, text summarization etc. BERT and RoBERTa are examples of masked Language models.[27]

2.3.2 Autoregressive Language Models

Autoregressive language model are more robust in comparison with MLM for mainly tasks which requires textual data to be generated such as Question Answering, text summarization etc. Unlike MLM which are used very efficient for classification tasks, Autoregressive language models perform well in sequence or word prediction for the given context. Autoregressive process makes use of Markov chains where the token or sequence to be predicted completely depends upon previous token [28].
However unlike BERT, where the learning in done bi-directional to understand the word in the context better, The autoregressive language models learns uni-directional mainly from left to right which is why the generated output text is human like however for tasks like classification would be inefficient [28]. Some of the examples of Autoregressive language models are GPT and XLNet

2.3.3 Classification Model Evaluation Metrics

Given that the tasks undertaken in this thesis will involve classification, it is fitting to delve into metrics commonly employed in NLP for text classification. Some of the commonly used evaluation metrics can be derived and explained in depth using fig 2.6 of contingency table.

The 2X2 contingency table as shown in fig 2.6 consists of 4 sections, The prediction done by the model will be classified to any of the 4 sections based on Actual and predicted value

TN (True Negative) contains all the correct predictions of the model which were negative(0)

TP (True Positive) contains all the correct predictions of the model which were positive(1)

FN (False Negative) contains all the incorrect predictions of the model which were negative(0 but 1 predicted)

FP (False Positive) contains all the incorrect predictions of the model which were Positive(1 but 0 predicted)

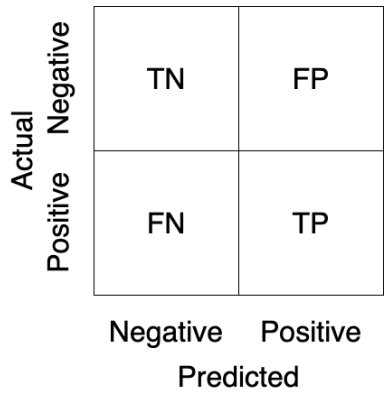

Figure 2.6: 2X2 contingency table

Confusion Matrix

Confusion matrix is a numerical representation of the contingency matrix mentioned in fig 2.6. In Binary classification the confusion matrix would be 2 X 2 as there are 2 labels for classification. Models predictions will be classified to any of 4 categories TP,TN, FP, FN based on ground truth and predicted value. Final confusion matrix will contain count of each category. Confusion matrix would be crucial in understanding the learning, when the dataset is imbalanced and If model predicts the majority class it could have high accuracy but in reality is inaccurate.

Accuracy

Accuracy is a perfect metric when the dataset is balanced, It checks for correct predictions. The correction would be TP and TN from the contingency matrix.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.11)$$

Precision

Precision is the ratio of true positives TP to all positive model predictions which is the sum of TP and FP.

$$Precision = \frac{TP}{TP + FP} \quad (2.12)$$

Recall

The recall is the ratio of true positive TP to actual positive labels which is the sum of TP and FN

$$Recall = \frac{TP}{TP + FN} \quad (2.13)$$

F1-score

The values of precision and recall are used to compute the f1-score, Hence F1-score is more accurate when the dataset is imbalanced in comparison with accuracy. The eqn 2.14 represents formula with precision and recall while eqn 2.15 represents formula with context to positives and negatives which are TP, FP and FN.

$$F1 = \frac{2Xprecisionxrecall}{precision + recall} \quad (2.14)$$

$$F1 = \frac{TP}{TP + 0.5(FP + FN)} \quad (2.15)$$

2.4 Word Embeddings

Word embeddings are the pivotal components of language models, As their outcomes majorly depends on them for NLP tasks like text classification, sentiment analysis etc. The main function of the word embeddings is to represent words or sometimes documents in a numerical format and identify or capture meaning or relationships for the given context. As discussed in section 2.1.7 , Encoding is a simple numerical vector representation of a word consisting of 0 and 1, But word embedding are multi-dimensional vectors which are not just limited to 0 and 1 [29] . The core idea behind the embeddings is that for given

context, The words which have similar meaning would be used in a similar context and hence would have a similar representations. For instance the words "Fast" and "Quick" would have similar representations as they would be used in a similar context for the given data, while "Happy" would have a completely different representation.

Selecting a word embedding is a pivotal decision in NLP as the performance of these embeddings is crucial in determining both their quality and relevance for the given task.Certain word embeddings can prove effective in capturing specific patterns tailored to a particular NLP problem, rendering them well-suited for the chosen task [30]. The main considerations for selecting word embeddings can depend on the characterstics of dataset, Nature of task, Learning type (supervised or unsupervised) and sometimes even on the computational resources available as some embeddings especially the newer pretrained embeddings demand substantial computational resources. The word embeddings can be broadly classified as static and dynamic, which are discussed below

2.4.1 Static Embeddings

According to [29] static embedding could be classified as prediction based and count based model embeddings. Prediction based embeddings puts emphasis on context of a word with respect to it's current sequence, emphasizing the relationships within the immediate context. One other hand count-based embeddings focuses on word frequencies but considers the entire corpus, hence the word frequency computed will be with respect to the entire given context which provides broader perspective in identifying patterns for given corpus.

Prediction Based Embeddings

Continous bag of words (CBOW) and Skip-gram model introduced by [31] are used in the word2Vec model [30]. Both CBOW and Skip-gram objectives are to understand the semantics and contextual relationships for the given words. Figure 2.7 showcases a general representation of CBOW and Skip-gram, CBOW is trained to predict a word w(t) based on the context of the surrounding words in a sequence which are w(t-2), w(t-1)....w(t+2). The sum calculated is the average of the weights of the vectors of the surrounding words [31] which is a log-linear function and the sequences selected for predictions are short and hence their knowledge domain is confined to local context.

Skip-Gram is trained in a manner that complements CBOW, Skip-gram objective is to predict the sequence of words w(t-2),w(t-1)....w(t+2) based on an input word w(t). The projection calculated in skip-gram with log-linear functionality and the sequence of words predicted have close associated with the input word. However since the context is confined locally, the distance words in most scenarios are less related to the current input word. For instance w(t-2) and w(t-1) might be closely related w(t) when compared to w(t+2). Even though both CBOW and skip-gram word embeddings are prediction based and their context is limited, They can still be used in NLP tasks depending upon criteria such as size of dataset, desired level of word representation and available computational resources.[30]

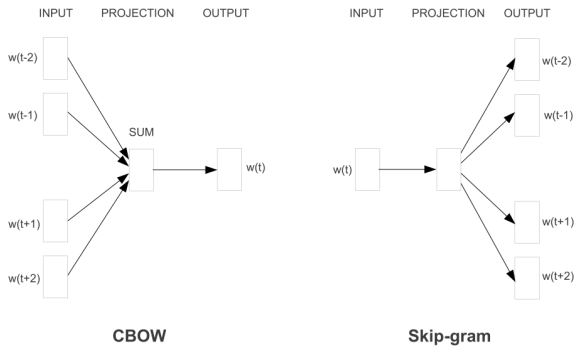

Fig. 2.7 CBOW and Skip-gram model source:[31]

Count Based Embeddings

Unlike prediction based embedding, Count based embeddings takes in consideration the entire corpus for computing frequency of the word [29] . Hence their scope is global and GloVe word embedding is classified as count-based embedding.

GloVe is another model of word representation which is more probabilistic approach. A matrix of word occurrence is built where all the words in vocabulary are implemented and and their frequency X_{ij} is computed, which denotes the frequency of word j with respect to word i in the entire context [8] . Similarly $Xi = \sum_{i=1}^{k} X_{i,k}$ be the frequency any word k with respect to word i. Finally the probability of word j is computed by $P_{ij} = X_{ij}/X_i$
The meaning of the words in the context completely depends upon the computed co-occurrence probabilities. for two words i and j, the relationship of these words is determined by examining the ratio of their co-occurrence probabilities with respect to various words k. For word k related i but not j, the P_{ik}/P_{jk} will be big, similarly for other case

where k is related to j, the value would be small. Since the co-occurrence probabilities depend upon 3 words i,j and k [8]. The general formula is given by

$$F(wi, wj, wk) = P_ik/P_jk \qquad (2.16)$$

2.4.2 Dynamic Embeddings

Dynamic word embeddings unlike static word embeddings adapt and change over time during the NLP task. The embeddings are not fixed for the dataset, They are updated based on changing contexts or whenver new information is available . Hence dynamic word embeddings are efficient where the used language is dynamic and meaning of the words are context-dependent rather than pre-defined. For instance the use of word "Bug" could refer to a computer bug in terms of technical terminology or can refer to an insect in general context. The ability of dynamic word embeddings to capture the context-specific meanings makes them valuable in NLP. There are multiple approaches to achieve dynamic word embedding and one of the common approach is the use of contextualized Word embeddings

Contextualized Word Embeddings

Contextual word embeddings are context-sensitive in which a word appears and same word can have multiple representations depending upon various words in the sequence. As the complexity of the represents has advancement drastically in comparison to static embeddings, Most of the contextual word embeddings or contextual models are pretrained on a large dataset to learn and understand the general language patterns. some of the dynamic word embeddings are mentioned below

ELMo

ELMo word Embedding was one of the early contextual word embeddings introduced in 2018 which was bidirectional. The concept of bidirectional indicates that the context of the given word is understood from both the directions. For this to achieve ELMo uses Bidirectional LSTM network in it's architecture, The contextual information is captured at various layers of LSTM network. ELMo performs well in semi-supervised setup where the Bi-LSTM network is pretrained [33].

The Bi-LSTM framework would consists of 2 networks would compute forward and backward probabilities for a word in a given sequence. For a sequence having N tokens, the forward network would compute the probability for token t_k by making use of all tokens

where the current token t_k is forward or all tokens which are present in a sequence before current token which are tokens from $t_1....t_{k-1}$

$$p(t1, t2....tn) = \prod_{k=1}^{N} P(t_k|t_1....t_{k-1}) \quad (2.17) \quad [33]$$

Similarly the backward network would compute the probability for token t_k which uses all tokens that occur after current token in a sequence which are $t_{k+1}....t_N$

$$p(t1, t2....tn) = \prod_{k=1}^{N} P(t_{k+1}....t_N) \quad (2.18) \quad [33]$$

Bi-LSTM framework uses the overall log liklihood of both forward and backward probabilities.

$$\sum_{k=1}^{N} logp(t_k|t_1...t_{k-1}; \ominus^{\rightarrow}_{LSTM}, \ominus_s) + logp(t_k|t_{k+1}...t_N; \ominus^{\leftarrow}_{LSTM}, \ominus_s) \quad (2.19) \quad [33]$$

$\ominus_s$ indicates that softmax activation function is applied to both forward and backward direction. ELMo was the early model which made use of concept of bidirection and paved a way to BERT and GPT which have significantly improved performance for various NLP tasks.

GPT

GPT introduced in 2018 by OpenAI [34], unlike ELMo which uses Bi-LSTM framework in it's architecture, GPT is a model based on Transformer architecture. Transformer architecture will be discussed in depth in following section 2.5. GPT operates at more comprehensive level with capabilities to generate sequences of texts rather than just focused on word embeddings. GPT uses a concept of attention which is a core concept of any transformer-based models which is a mechanism that identifies the significance of each word in the context of a given input sequence. unlike ELMo the GPT is unidirectional, As a result it is trained to predict word based on preceding words in a sequence and hence is categorized as an autoregressive model.

The GPT model was trained in two stages, The first stage was done using unsupervised learning over a large corpus of data with an aim to understand the natural language text and second stage was to use labelled data to fine-tune the model in a supervised learning environment.

In unsupervised learning for n tokens $u_1.....u_n$ log-liklihood is computed for token u_i with k as context size and P being the conditional probability which is modelled based on NN having parameters $\ominus$

$$L_1(U) = \sum_i logP(u_i|u_{i-k}, ...u_{i-1}; \ominus) \quad (2.21) \quad [34]$$

The concept of self-attention as discussed earlier is applied for the tokens accompanied by softmax activation which would compute output distribution for n layers with an embedding matrix W_e and positional embedding matrix denoted by W_p.

$$h_0 = UW_e + W_p \quad (2.20) \quad [34]$$

$$h_1 = transformer\_block(h_{l-1}) \forall i \epsilon [1, n] \quad (2.22) \quad [34]$$

$$P(u) = softmax(h_n W_e^T) \quad (2.23) \quad [34]$$

In supervised learning, The main objective is to fine-tune the model based on a labelled dataset C consists of sequences $x = x^1...x^m$ with corresponding labels y. The dataset is fed to pre-trained model from eqn 2.22 to get h_l^m which is a transformer block. W_y is a linear output layer with y labels. Both $h_l^m and W_y$ is used compute probability of label y with respect the input sequence. eqn 2.25 log applied to maximize

$$P(y|x^1...x^m) = softmax(h_l^m W_y) \quad (2.24) \quad [34]$$

$$L_2(C) = \sum_{(x,y)} logP(y|x^1...x^m) \quad (2.25) \quad [34]$$

GPT is known for its impressive language understanding and generation capabilities. While it doesn't generate word embeddings in the traditional sense, its output can be leveraged for various NLP tasks. However since the learning is unidirectional it's better suited for generative tasks like text completion,summarizing rather than classification and sentiment analysis. GPT-3.5 is currently used for ChatGPT with largest variant trained on 175 billion parameters.

2.5 Transformer

Transformer architectures proposed by [35] are simple and more efficient in comparison to the complex CNN or RNN architectures consisting of encoder and decoder blocks. Transformers use a core concept of attention mechanism is eliminates the necessity of using an RNN or CNN in the architecture. They are particularly quite efficient when handling sequential data and the architecture consists of 2 major blocks namely encoder and decoder.

For a given input sequence $x = (x_1....x_n)$ the main objective of the encoder is to map the input sequence to $z = (z_1...z_n)$ which is a sequence of continuous token representations. Similarly for z, the decoder's objective is to generate a corresponding output representation $y = (y_1....y_m)$. The architecture follows auto-regressive approach which is similar

to RNN, where previously generated output will be used for generating the succeeding output. fig 2.8 showcases the layers for encoder and decoder blocks where right block represents decoder and left representing the encoder block.

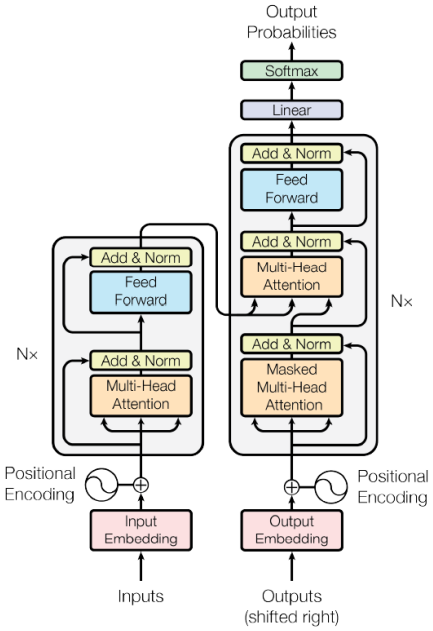

Figure 2.8 Transformer Architecture source: [35]

Encoder

The encoder consists of N identical layers, where each layer consists of two sub layers. one of the sub layer is used for multi-head self attention while other sub-layer is a position-specific feed forward NN. Residual connection is applied around the two of the mentioned sub-layers whose objective is to tackle the vanishing gradient problem. As seen on the left side of the fig 2.8, each of the sub-layers are applied with a layer of normalization. normalization layer objective is to provide stable and faster training, it is given by the formula

$$LayerNormalization(x + sublayer(x)) \quad (2.26) \quad [35]$$

The function sublayer(x) is implemented for each sub-layer. Both the layer normalization and residual connections contribute the encoder output with dimensions of 512.

Decoder

Similar to encoder, Decoder is a block consisting of N layers. However unlike each layer of encoder which has 2 sub-layers, In decoder each layers consists of 3 sub-layers. The additional sub-layer is added in between the other two layer and the objective is to perform the multi-head attention mechanism to the output of the encoder block. while the

sub-layer with feed-forward NN is similar to encoder, The sub-layer consisting of multi-head attention is replaced with masked multi-head attention. This sub-layer along with output embeddings as shown in fig 2.8, makes predictions for i making based on values which are less than i [35]. In the decoder, layer normalization and residual connections are employed with the same purpose as in the encoder.

The major architectural difference between an encoder and decoder of the transformer is the inclusion of masked multi-head attention mechanism in decoder as mentioned above and the output is fed to a linear layer and finally to a softmax layer which would compute the probabilities for the words in the sequence. Hence decoder block is crucial for generative tasks like text generation as output of the model is natural language. However for tasks related to classification, we can just make use of encoder structure. One of the strengths of the transformers is the flexibility to use encoder and decoder separately as independent models. Given that the primary tasks in this thesis revolve around text classification, models based on encoders will be employed.

Positional Embedding

Since the transformer architecture does not understand the sequence order of words, positional encodings are added to the embeddings of both the transformer blocks. They provide information about the position of each word in a sequence which helps the model to understand the sequential relationships. The positional encodings dimensions are same as the model which is d_model so that the sum can be computed easily. Positional encodings are computed using trigonometric functions given by eqn 2.27 using sine and eqn 2.28 using cosine function

$$PE_{pos,2i} = sin(\frac{pos}{1000^{2i/dmodel}}) \quad (2.27) \quad [35]$$

$$PE_{pos,2i+1} = cos(\frac{pos}{1000^{2i/dmodel}}) \quad (2.28) \quad [35]$$

Each positional encodings is represented by it's position pos and has dimension i. Both of the above functions are used because they enable the model to efficiently learn the relative positions. If k represents a fixed offset, PE_{pos+k} is a linear function of PE_{pos}

2.5.1 Attention

The concept of attention as introduced in [35] is a core concept of transformer architecture which enable model in effectively learning different parts of input sequence when producing output sequence for each element.
Attention is calculated using 3 major parameter input vectors which are query Q, key K and value V. The objective of attention function is to map key K and value V pairs along query Q to an output . The output is an vector which is computed using weighted sum of values [35]. Here the weight assigned to each value is generated using a compatibility function which takes 2 parameters, the relevant key of the value and the associated query. While numerous attention mechanisms exist, this thesis delves deeper into the examination of Scaled Dot-Product and Multi-head Attention mechanisms, Their architectures are showcased on Left and right side of fig 2.9 respectively.

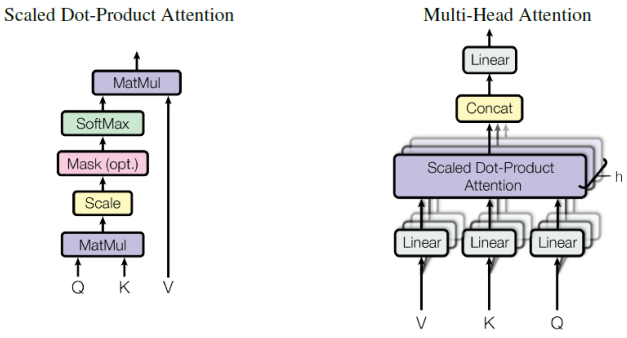

Figure 2.9 Scaled Dot-Product Attention(left) and Multi-Head Attention (right) [35]

Scaled Dot-Product Attention
Scaled Dot-Product Attention calculates the weights using the dot product of query and key matrices. If we assume both queries and keys having dimensions d_k, while d_v represents the dimension of values. Dot products of all keys with a query is computed and is divided by $\sqrt{d_k}$, Finally softmax activation is applied to the result to obtain series of weights for each values.
However In practical implementation applying attention to each query would be tedious, As a result matrix Q containing series of queries is passed as an input. Similarly keys and values are contained in matrices K and V respectively. Eqn 2.29 represents the attention eqn

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (2.29) \quad [35]$$

$\frac{1}{\sqrt{d_k}}$ is the scaling factor, which regulates dot product of the key and query matrices from becoming too big which can cause instability during model training. Scaling factor stabilize the gradients to avoid vanishing or exploding gradient scenarios. It also the magnitude of the computed scores by the attention function. $\frac{1}{\sqrt{d_k}}$ has provided better results in practical implementation in terms of space-efficiency and better execution times.[35]

Multi-Head Attention

A single attention function as in eqn 2.29, returns output vector of weights with dimensions d_v corresponding to each values. On other hand Multi-head approach is an extension of attention mechanism introduced in [35], which enhances the capturing capabilities by allowing model to jointly attend various positions in the input sequence, As a result different relationships are acquired for the provided sequence.

Multiple head refers to the implementation of multiple sets of keys,values and queries rather than single set (single head). Each head computes the attention independently and are later concatenated to produce a set of outputs, which allows model to capture diverse patterns and relationships in parallel. Considering queries, keys and values of dimensions d_k, d_k, d_v respectively, The output computed using softmax layer corresponding to each value will be of dimension d_v. However the output is generated h no of times with different learned patterns. Each output of the head is concatenated and final values are computed. Eqn 2.30 represents the multi-head attention for h times.

$$MultiHead(Q, K, V) = concat(head_1, ...head_h)W^O \quad (2.30) \quad [35]$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad (2.31) \quad [35]$$

W^O is a learned linear transformation applied to the concatenated outputs, In eqn 2.31, Following are the projections for the parameter matrices $W_i^Q \epsilon \mathbb{R}^{d_{model} X d_k}$, $W_i^K \epsilon \mathbb{R}^{d_{model} X d_k}$, $W_i^V \epsilon \mathbb{R}^{d_{model} X d_v}$ and $W^O \epsilon \mathbb{R}^{d_{model} X h d_v}$

2.5.2 BERT

BERT transformer model introduced in [36], BERT uses the concept of directional where previous models only considered left or right context. Similar to ELMo, BERT captures the bidirectional relationships of tokens in the given sequence. However unlike ELMo as discussed in section 2.4.2 which uses bi-LSTM framework to capture both contexts, BERT is a transformer architecture which uses concept of masking. It is pre-trained on a large corpus of data using masked language modelling and hence the model can be fine-tuned with additional layers and an output layer for various NLP tasks like QnA, classification etc.

The implementation of BERT was done in two stages:
Pre-training : The model is pre-trained in a unsupervised manner over large corpus for different tasks.
Fine-tuning: The parameters of the model are fine-tuned in a supervised learning structure for different tasks.
fig 2.10 Provides an overview architectures of both stages of BERT Implementation, Pre-training (right) and Fine-tuning (Left)

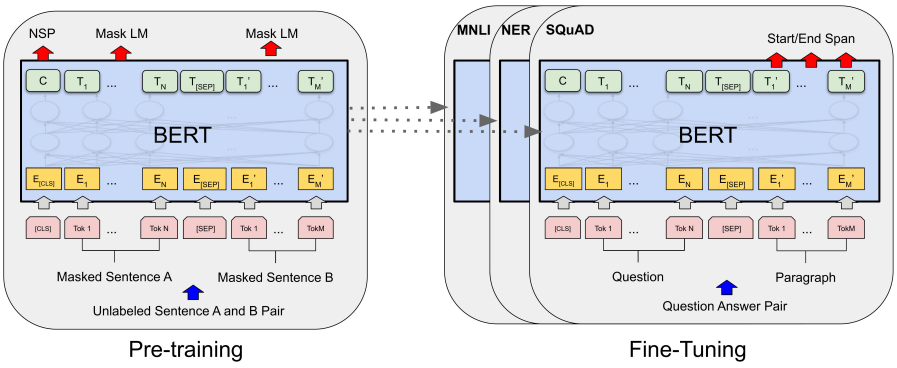

Figure 2.10 Pre-training and Fine-tuning stages of BERT source: [36]

Two variants of BERT models are proposed in the paper [36] namely $BERT_{Base}$ a base variant and $BERT_{Large}$ a large variant. These 2 variants differ in the architecture, as a result large variant has more computational capacity in comparision to base variant. Base variant consists of 12 layers of transformer blocks while large variant has 24 layers. The output dimensions of large variant is 1024 while the base variant has 768. As discussed in section 2.5.1, The attention heads of base variant are 12 and large variant consists of 16. However the major difference is the no of parameters the models are trained on, base variant is trained on 110M parameters while large variant is trained on 340M parameters.

BERT uses WordPiece embeddings which is a type of dynamic word embedding. WordPiece embeddings introduced in the paper [37] has a vocabulary size of 30,000. The BERT embeddings adds specials tokens for the given input sequence. The token [CLS] is added at the start of every input sequence, [CLS] is an abbreviation for classification because it acts as an aggregate sequence for NLP classification tasks. The other token [SEP] is added for a pair of sequences. To differentiate the sequences in the pair, [SEP] acts as an separator which distinguishes two sequences. Figure 2.11 showcases the BERT embedding where special tokens are appended to input and word embedding is computed as sum of token, segment and position embeddings.

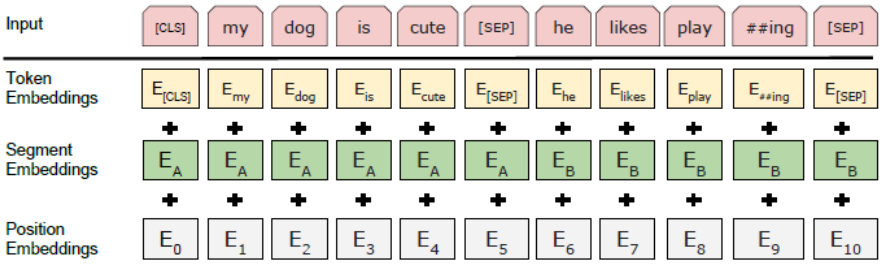

figure 2.11 Representation of BERT's input embeddings source: [36]

The two major concepts used in Pre-training of BERT are Masked Language Modelling and Next Sentence Prediction.

Masked Language Modelling

For any given input sequence, certain percentage of the tokens are masked at random and the objective is to predict the masked tokens accurately which will ultimately contribute to model's ability to learn the sequence in a bidirectional manner. In the paper [36] about 15% of input tokens are masked with token [MASK] which will be used for prediction. For instance for the input sequence "Pizza is an Italian food" after masking would be "Pizza is an [MASK] food". Now the model tries to predict which token would replace [MASK] considering information from both directions in the input sequence.

Next Sentence Prediction

Apart from masking, BERT pre-training stage also include next sentence prediction. The objective of the task is to understand the relationship between the given two input sequences because it is critical for various NLP tasks like QnA or language Inference. The model is trained in a binarized way such that it takes input pairs sequences and provides predictions on whether they are consecutive sentences or not. For sequences A:"Pizza is

an Italian food" and B :"Taj Mahal is made of white marble" , The model's prediction would be Not true as consecutive sentences.

For both Masked language modelling and NSP, Cross-entropy loss or log loss methods are implemented to compute losses. The overall loss is a linear combination of matrices or sum of the individual losses.

2.5.3 RoBERTa

RoBERTa transformer model introduced in [38], Robustly optimized BERT is built and optimized using BERT as a reference. Since RoBERTa is based on BERT, It uses masking for learning patterns and capture bidirectional relationships. However the type of masking differs, While BERT masks 15% of the tokens in a sequence in randomly, RoBERTa uses dynamic masking where a new masking pattern is generated for every input sequence. The other major change is the exclusion of Next sentence prediction which is critical concept of BERT. RoBERTa relies on masked language modeling. The author in [38] experiments using full sentences having max length of 512 tokens where separator is added when sampling begins from new document and performance noted was similar or better than $BERT_{base}$ which uses next sentence prediction. Similar to BERT, RoBERTa uses special tokens [CLS] and [SEP] for input sequence and [MASK] for masking. figure 2.12 shows pair of sentences embedded by RoBERTa.

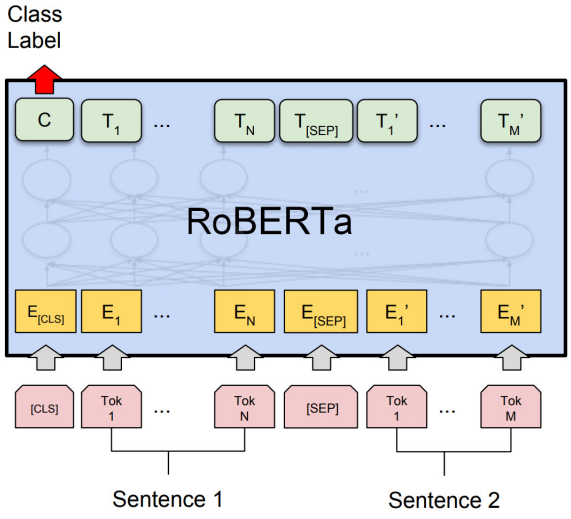

Figure 2.12 Overview of RoBERTa source:[39]

Unlike BERT which uses WordPiece for embedding, RoBERTa makes use of Byte-pair encoding which was introduced in [40]. The unique characters are initially treated as uni-

gram and the process is recursive where for every succession most frequent pair is merged as a bi-gram which will eventually result in a n-gram.

RoBERTa achieves enhanced efficiency and training speed by being trained on a larger batch size when compared to BERT. It is trained on a large corpus of data for a longer duration of time, which significantly improves the model's ability to understand and capture distinct patterns and also benefits from extensive hyperparameter tuning. Overall this contributes to the robust performance across various NLP tasks and is considered a strong alternative to BERT.

2.5.4 XLM-RoBERTa

XLM transformer architecture proposed in [42] extends the existing transformer architecture specifically for cross-lingual NLP tasks. Hence the basic transformer architecture proposed in [35] acts as a foundational model because of it's efficiency in capturing long-range patterns and relationships for sequential data. It uses attention mechanism to compute weights for different values or words in a given sequence based their significance by understanding their contextual relationships. As discussed in section 2.5.1 Multi-head attention is also utilized to identify different aspects of input sequence in parallel enhancing model's understanding of data.

In [42], The authors propose cross-lingual understanding framework for XLM model pretraining with the objectives of better cross-lingual representations using unsupervised learnings, a supervised learning approach for parallel data computation during pre-training, Introducing supervised and unsupervised machine translation methods which significantly outperform previous cross-lingual models. Some of the key concepts used for XLM embedding are Masked language modeling and Translation language modeling as shown in the fig 2.13

Casual Language Modeling The concept of CLM is implemented in XLM architecture whose objective is predict the consecutive word in a given input sequence based on the probabilities of preceding words. The functionality can be easily achieved using RNN where $P(w_t)$ would be computed as$P(w_t|w_1...w_{t-1})$. However transformers makes use of different approach where previous hidden states are passed to current batch unlike LSTM where backpropogation through time (BPTT) provides the network with hidden states of previous iteration.

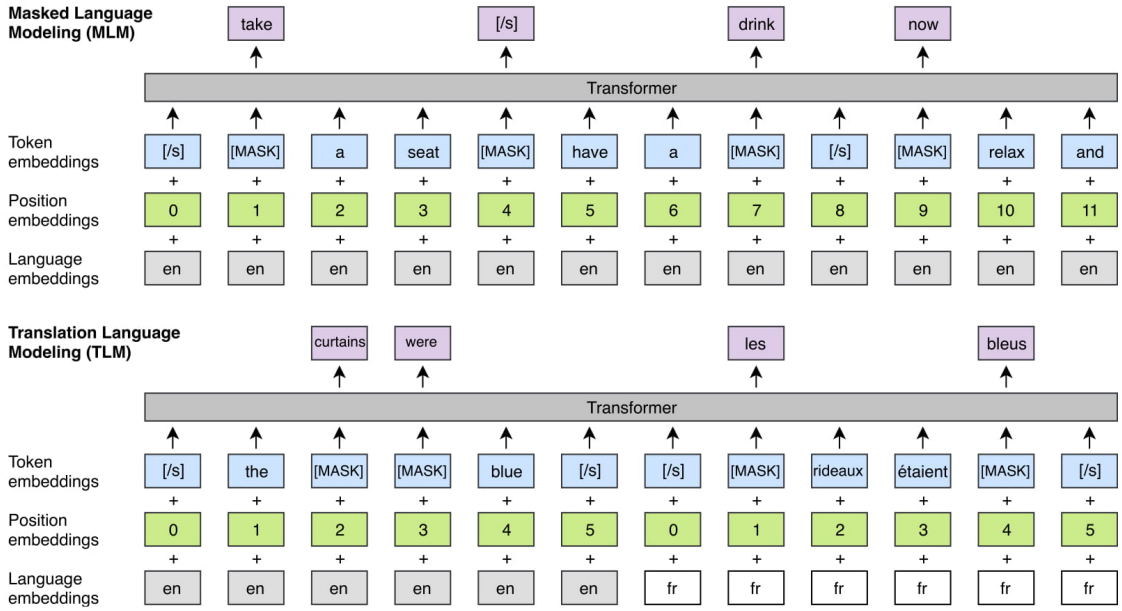

Figure 2.13 XLM model embeddings MLM(top) and TLM(bottom) source:[42]

Masked Language Modeling The objective of MLM is to make predictions by masking certain section of input sequence, so that the model predicts the masked token which will improve the model's ability to learn the sequence in bidirectional manner. Similar to BERT the masking is done for 15% of tokens in the sequence with [MASK], However unlike BERT where a pair of sentences are used, XLM uses text streams of various lengths with maximum length of 256 tokens.

Translation Language Modeling For tasks MLM and CLM the data used is monolingual. However in TLM the objective is to capture patterns and semantics of the given sentences across different languages. In simple terms the MLM task is extend to multilingual sentences where the focus is on parallel computation as showcased in fig 2.13. For instance 2 sentences namely source and target which are in different languages are masked randomly. for source prediction of the masked tokens the model can use the neighboring tokens of the same sentence or can infer the translation sentence(target) which would improve model's understanding of different representations. TLM broader goal is to create language representations that are useful for various cross-lingual NLP tasks. The model captures semantic similarities and differences between languages, allowing it to transfer knowledge effectively for specific downstream NLP tasks in a multilingual environment.

If the model uses same sequence, the objective could be achieved using CLM. As a result XLM models are pre-trained with CLM, MLM or MLM in combination with TLM [42]. The paper by Facebook AI [41] makes use of RoBERTa model and integrates it with

XLM architecture, Hence referred as XLM-RoBERTa. It outperforms mBERT used in [42] by 23% accuracy. It is trained on large no different languages and English , German and French being the main high-resourced languages which are evaluated for sequence and classification tasks.

2.6 General Adverserial Networks

GANs introduced in the paper [44], is a generative framework which consists of two models namely generator and discriminator which are trained simultaneously through adversarial process. The adversarial training procedure involves a zero-sum game between the generator and discriminator where the main objective to the generator is to produce data which is similar to the real input data while the discriminator's goal is to distinguish whether the given data is real input or produced by the generator. Fig 2.14 showcases an overview of a GAN architecture for image dataset

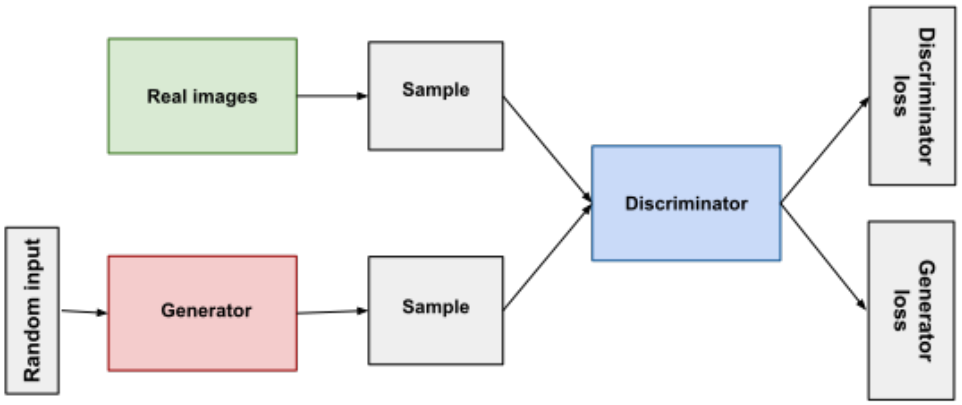

figure 2.14 Overview of GAN-Architecture source:[43]

for data x, The generator learns the data distribution P_g with the help of noise distribution function $P_z(z)$ where z is a random noise vector sampled using prior distribution, The overall generator function is given by $G(z;\theta_g)$ where θ_g is parameters. Discriminator $D(x:\theta_d)$ is a single scalar where D(x) is the probability that x is from the real data rather than P_g. The overall objective function of GAN is given by eqn 2.31.

$$min_G max_D V(D,G) = E_{x\ P_{data(x)}}[logD(x)] + E_{z\ p_z(z)}[log(1 - D(G(z)))] \quad (2.31) \quad [44]$$

2.6.1 Generator

The Generator creates data samples which are similar to real input data fed to the GAN network. The objective function of the generator is to minimize the log probability of the discriminator D assigning the generators produced output G(z) for random noise vector z as fake. The generator objective function is given by eqn 2.32. Generator tries to minimize this objective function by producing outputs which are similar to real data, such that discriminator classifies them as real.

$$min_G E_{z\ p(z)}[-logD(G(z))] \quad (2.32) \quad [44]Theorem1$$

2.6.2 Discriminator

The Discriminator classifies the given input samples as real which is the real input data or fake i,e the generator output. Unlike generator, discriminator tries to maximize it's objective function to enhance it's capability of distinguishing between real and fake. The objective function is given by the eqn 2.33. Discriminator maximizes this function by assigning high probability values to real data samples and low values to generator's outputs.

$$max_D E_{x\ P_data(x)}[logD(x)] + E_{z\ p_z(z)}[log(1 - D(G(z)))] \quad (2.33) \quad [44]Theorem2$$

3 Related work

3.1 LSTM vs Transformers

The most common and highly effective models for NLP data are transformers and RNN. While both models have proven successful in wide range of applications such as speech recognition, language translation etc. LSTM as discussed in section 2.2.4 is a type of RNN which is widely used for NLP applications. The paper [62] studies the performances of Transformers and RNN for various tasks such as automatic speech recognition, text to speech, language translation. In total 15 benchmarks are considered for automatic speech recognition for evaluation. While 2 and 1 benchmarks are considered for text to speech and language translation tasks respectively.

3.1.1 Transformer over RNN

- Transformers makes use of attention mechanism that enables them to process and capture crucial aspects of input data. On a contrary RNN's like LSTM and GRU makes use of gated structure to capture insights of the input data. This allows transformers to process data in parallel, resulting in faster training times compared to sequential approaches of RNNs.
- Transformers are more effective compared to LSTMs when handling long-range dependencies, mainly because of the self-attention mechanism. Transformers are able to weigh the importance of various positions in the input sequence, while LSTMs struggle with retaining information as they use gated structure which are less effective in retaining information of the distance positions in longer sequences.

3.1.2 Experiments Results

- The paper [62] setup 15 benchmarks for ASR tasks which were various datasets and some were multilingual. Both Transformers and RNN architectures were experimented for these benchmarks where Transformers revealed significant performance benefits and outperformed RNN for 13/15 benchmarks. For TTS the performances were similar, However for ST benchmarks again, Transformers outperformed RNNs which struggled with under-fitting.
- The paper [63] experimented using Transformer and LSTM based language models for ASR tasks in a semi-supervised learning scenario, Where about 20%

of the training set was labeled and remaining 80% being unlabeled. There was about 3-6% lower perplexity for transformers language models compared to LSTM where where perplexity measures the uncertainty for unseen data, Hence lower perplexity score contributes to better model generalization. For word error rate or WER evaluation metrics which checks for incorrectly transformed words in the speech recognition system were similar for both language models.

3.2 Bi-LSTM

Bidirectional Long Short-Term Memory (BiLSTM) is a variant of traditional LSTM which is mainly used in NLP tasks. As discussed in section 2.4.2, Unlike LSTM which is uni-directional, Bi-LSTM is bidirectional. Hence Bi-LSTM have better understanding and capabilities to capture semantic relationship and patterns compared to LSTM, Achieving better performance for wide range of NLP applications.
For integrating Bi-LSTM in the network, an extra LSTM layer is introduced that is responsible for reversing the flow of information for the given input sequence in the network which is known as Backward LSTM. While the LSTM layer which is responsible for computing forward flow for the sequence is Forward LSTM as shown in the fig 3.1. As discussed in section 2.4.2 Forward and backward LSTMs compute the Forward and backward probabilities given by eqns 2.17 and 2.18 respectively. Finally the overall likelihood is computed as sum, product or concatenation. In case ELMo, Both the probabilities are summed, which is given by the eqn 2.19.

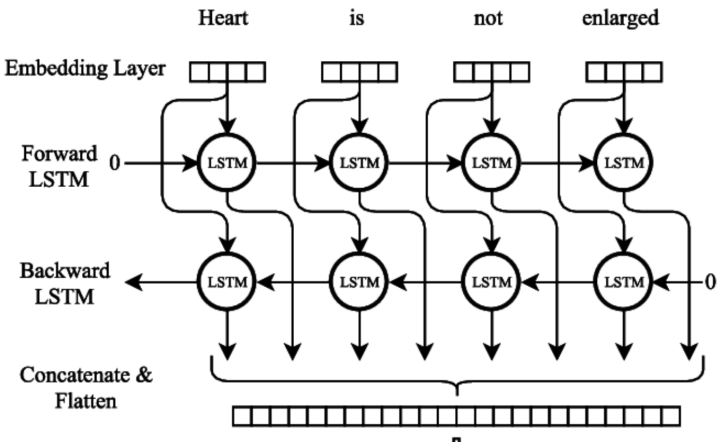

Figure 3.1 Overview of Bi-LSTM [66]

BiLSTM generates distinct outputs for each element, or word, within a given sequence, making it particularly advantageous in various NLP tasks like sentence classification, language translation and entity recognition. Furthermore, its applications extends to diverse

domains such as speech recognition, protein structure prediction, handwritten text recognition and related fields.

3.2.1 Experimental Results

- The paper [64] compares performances of BiLSTM and Transformer architecture for text classification in the context of cyberbullying. To tackle the problem of cyberbullying, The paper introduces a new framework which uses Bi-LSTM as a core component where 4 bidirectional LSTM layers of dimensions 512 are implemented, The performance is compared with a straightforward transformer architecture. The validation accuracy and model generalization is better in transformer compared to BiLSTM. However the major difference lies in the training, where Transformer trained on less than 1 million parameters was able to achieve better evaluation results compared to proposed BiLSTM framework which was trained on 48.82 million parameters.

- The primary goal of the paper [65] is to extract and categorize text in Name Entity Recognition. Various sequence tagging models are used to tag or assign relevant labels to the extracted tokens. The models used are of two distinct architectures, one based on BiLSTM and other on transformer architecture. A comparative evaluation of their performance was conducted where 8 variants of BiLSTM and Transformer models were experimented. The transformer-based models surpassed the limitations observed in Bi-LSTM networks. This advancement was achieved through the incorporation of different input features at the character, sub-word, and word levels. BERT variants were used for transformer architecture models which was able to achieve highest F1-score of 95.95 compared to the F1-score of 91.84 obtained by BiLSTM models.

3.3 SVM

Support Vector Machines or SVM is one of the traditional ML algorithm which is widely used in classification, regression tasks. SVM are less prone to overfitting as they use margin-based classification approach and hence work well in high dimensional spaces [67]. Another important feature of SVM is the support of kernel options, some of the kernel function like linear, polynomial are used to handle various types of data distribution. SVM are very effective for smaller datasets, Hence making them suitable for scenarios where data availability is limited. However training SVM on larger datasets is expensive, As time complexity of is cubic of the size of dataset. They are ideal for binary classifi-

cation but the efficiency decreases with increase in labels. Finally SVM do not provide probability estimates, Even though this can be achieved using cross validation but is computationally costlier.

3.3.1 SVM for Linear Classification

- The paper [68] focuses on detecting deceptive information around Covid-19. Traditional ML algorithms are used where SVM outperforms Random forest, Bayes Theorem etc. Dataset consists of labeled tweet data where linear SVM classifies the tweet text as fake or real. Traditional NLP preprocessing methods like tokenization, stemming, stop word removal are implemented with an aim of extracting relevant information. SVM classifier was able to achieve an F1 score of 0.94.

- The paper [69] compares the performance of transformers pre-trained language models with SVM. For measuring performance, The authors use 3 of the publicly available generic datasets and 1 domain specific private dataset related to IT support tickets. In a supervised learning scenario, The performances of SVM and transformers were similar and the use of transformer language model did not provide any major significant performance compared to SVM.

3.3.2 Transformer over SVM

- The paper [68] was able to achieve an F1-score of 0.94, However the evaluation test as shown in Fig 9 of the paper [68] consists of only 214 samples. The test sample is too small and as discussed SVM perform well on smaller datasets. While Transformer language models achieve better model generalization on larger datasets. However compared to other traditional ML algorithms, SVM is robust and can be use as a baseline for text classification tasks.

- The paper [69] compares the performances of SVM classifier with Transformer language models and concludes both have similar performances. However for SVM classifer, The author uses Tf-idf vectorization which does not consider the context of semantic of the word in the given sequence. Although there exists a semi-supervised learning variant of SVM as proposed in the paper [70], Transformer language models offer greater flexibility for extending the existing framework. They enable the seamless integration of various networks, providing a more versatile approach.

4 Approach

4.1 Datasets

The primary focus of the thesis is to solve challenges related to deceptive text in the area of NLP. Hence the most suitable direction of exploration would be datasets consisting of both authentic and deceptive texts, As a results product reviews datasets would be an optimal choice. The research paper [45] investigates various NLP datasets dedicated to text summarization. The outcomes of the study propose specific criteria essential for the selection of an appropriate dataset. While the focus of the investigation is on text summarization, The datasets under examination are in natural language, aligning with the relevance and applicability to our specific problem. According to [45] , Some of the dataset selection criteria are mentioned below:

Dataset Selection Criteria

Relevance: The selected dataset should be directly relevant to the problem being addressed. The examples contained in the dataset should resemble or mimic the characteristics of the task, making sure that the model is trained and learning is done of representative data.

Size: Dataset size is another major criteria which is crucial in model training. In deep learning the build models often require large amount of data for efficient training. When the model is trained on larger dataset it promotes improved generalization.

Quality: The quality of the dataset is essential for reliable results and meaningful discussions. The dataset should be clean without missing values. In case of labelled dataset, The labels should be well annotated. The quality also contributes to the robustness of the model.

Representative: The dataset should be representative of broader population or domain for better generalization. Unrepresentative or biased datasets can results in models which do not generalize and hence perform poor on unseen data.

Diverse: The model will be able to learn wide range of patterns and features from the datasett which is diverse, consists of different scenarios and variations within the data. For instance in NLP dataset, The wide range of vocabulary used is essential for model

generalization.

Available Datasets

Tripadvisor Dataset: The Dataset developed in the paper [47] is a reviews dataset consisting of truthful and deceptive reviews regarding hotels mainly from Tripadvisor. According to the paper, The truthful reviews collected are from most popular hotels in the Chicago area of United states. The dataset creation takes inspiration from paper [48] which consisted of 42 deceptive and 40 truthful hotel reviews and using standard test and manual comaprision, They tried finding psychological relevant linguistic differences. However in [47] the dataset is balanced with 800 deceptive and truthful reviews, where deceptive reviews were created by crowdsourcing service Amazon Mechanical Turk by Turkers. Turkers fabricated reviews for the hotels, providing false accounts of their experiences for hotels they had never actually visited.
Some of the drawbacks of the dataset are:

- The dataset is balanced with 800 reviews belonging to each category which accouts to a size of 1600 reviews. Despite its substantial size relative to the dataset in [48] , Is still small for today standards for effective model generalization.
- The dataset consists of only hotel reviews, Hence lacks a broader representations of texts in various domains. The reviews collected are specific to hotels in Chicago and the dataset might lack in diversity.
- For truthful reviews, Any reviews less than 150 characters and reviews by any new-user were eliminated and not considered as part of dataset, Hence lacks length representations.

Trustpilot Dataset: The Datasets introduced in [46] are Yelp and Trustpilot datasets. Yelp dataset was created using 57,000 reviews from restaurants in New York. This dataset suffers almost similar drawbacks of Tripadvisor dataset which is limited to certain domain and geographic location. On other hand the Trustpilot dataset consists of 9,000 reviews which was shared with Trustpilot website. According to [46], The dataset is balanced and the representation of the reviews is not limited to hotels but over 130 various businesses
However some of the drawbacks of the dataset are:

- One of the major issue with the Truspilot dataset is its limited range, as it contains only 4-5 star reviews. As a result the model would lack generalization and would lean towards positive reviews.

- Unlike Tripadvisor dataset where the reviews from new users are eliminated, In Trustpilot The reviews collected are only from new users, As a result lacks representations.

4.1.1 Amazon Computer Generated Dataset

Author : Joni Salminen [49]
For this thesis the dataset proposed in the paper [49] would be used for the approaches focused on detecting deceptive statements. The dataset is a balanced set consisting of 40,000 real and computer-generated reviews. Original Amazon product reviews dataset [51] is used as a reference where 20K original reviews are taken directly, while deceptive reviews are computer generated using pre-trained models. It overcomes the problems of all the previous dataset mentioned above. A dataset size of 40,000 is deemed substantial and favorable for effective model generalization. Unlike some of the previous datasets which focused on hotel reviews, Amazon CG dataset consists of reviews from various categories ranging from sports, kitchen, Books, Toys etc. The no of reviews of each of the categories is roughly around 4000. figure 4.1 shows all the reviews categories along with their count in the dataset. The Dataset also has same sentence distribution as the original Amazon dataset [51]. fig 4.2 showcases the sentence length distribution over count of reviews

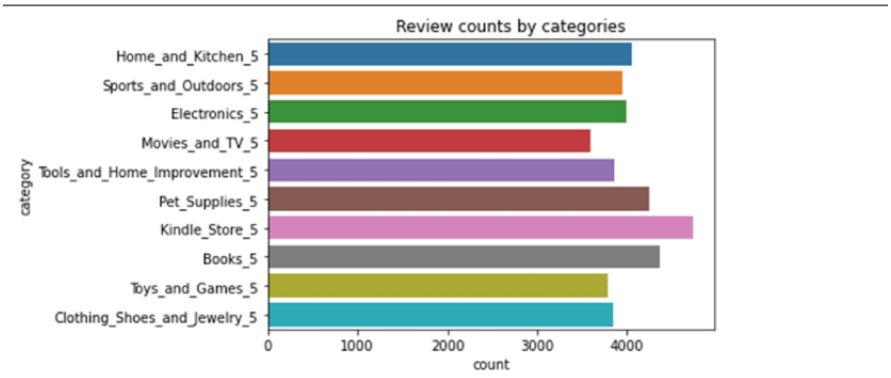

Figure 4.1 Reviews categories by count graph source: [49]

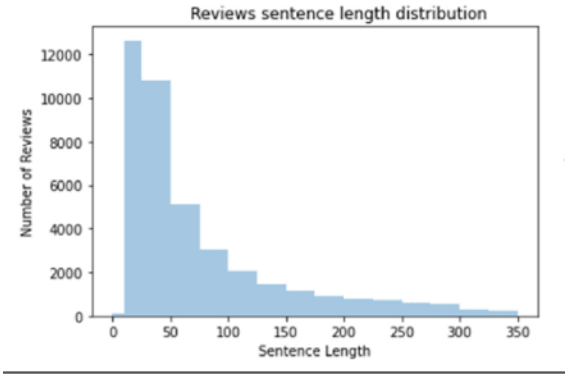

Figure 4.2 Sentence length distribution by count graph source: [49]

For computer-generated reviews, Top 10 categories with most product reviews are choosen from Amazon dataset [51]. The categories are showcased in fig 4.1, The reviews from these categories constitute 88.4% of Amazon dataset [49]. 2 language models are selected by the author which are ULMFiT [52] and GPT-2, They are pretrained on the sample Amazon dataset followed by deceptive reviews generation. ULMFiT being an LSTM based model is outperformed by GPT-2 a transformer based model. Hence for final dataset generation, GPT-2 is selected to generate 20k reviews. The dataset is evaluated by recruiting crowd workers and a trained GPT-2 classifier, where crowd workers annotated the sample set as deceptive or real and GPT-2 classifier was trained to detect deceptive reviews. Both human and GPT-2 accuracy are compared through statistical testing.

Figure 4.3 shows the sentence length distribution with mean of 305.6 and standard deviation of 307. The reviews having more than 350 words drastically reduces which is similar to original dataset and fig 4.4 shows sentence length to the reviews rating, From the graph it's clear there is no correlation between the reviews rating and sentence length.

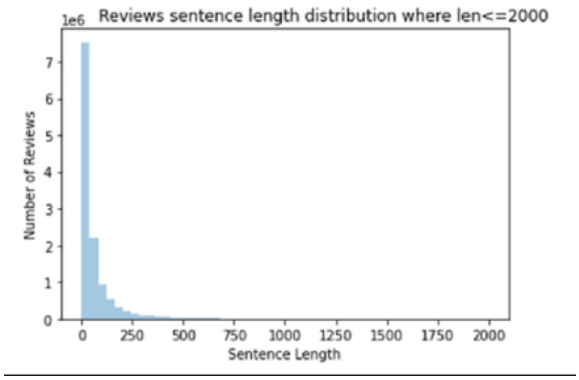

Figure 4.3 Sentence length to reviews count graph : [49]

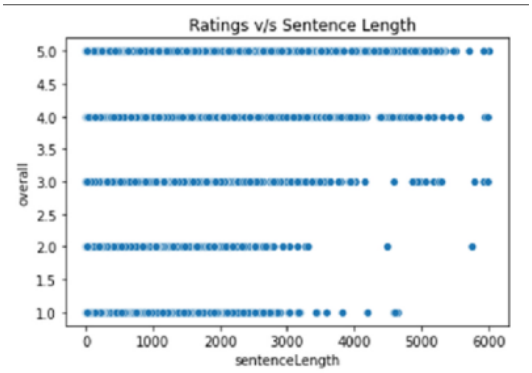

Figure 4.4 Sentence length and ratings distribution source: [49]

Available Conversational Datasets

CoQA: The **C**onversational **Q**uestion **A**nswering dataset proposed in the paper [54] consists of 127k questions along with their answers which are obtained from 8k conversations across social media sites like wikipedia reddit, news etc. However the aim of the dataset is training the model to have a suitable continuous conversation where for given n pairs of (Q,A), The answer A_n for question Q_n would depend on conversation history of $(Q_1, A_1)....(Q_{n-1}, A_{n-1})$. For each conversation in the dataset, Two annotators were employed for question and answer respectively when compared one annotator to have natural dialog flow in conversation. However some of the drawbacks are:

- The major drawback is that the answer generated to the question is in natural language rather than simple yes or no statement.
- The dataset is suitable for chatbot applications, where the questions asked starts with "when", "what", "where", rather than classifying the given statement as lie or truth from given context.
- Another major drawback is the annotations done by questioner and answerer, both have access to previous conversations as a result the questions asked are not naturally occurring.

QuAC: **Q**uestion **A**nswering in **C**ontext dataset proposed in the paper [55] contains 100K QA dialogs obtained mostly from wikipedia domain and around 14K from crowdsourced services. Unlike CoQA where the answer is a natural language with relevant answer, The answers in QuAC starts with yes or no for the relevant question followed by it's justification. The questioners doesn't have access to the context as a result the questions asked in the dataset are naturally occurring and is effective for model learning. Some of the drawbacks of the QuAC dataset are:

- Even though the answers in the dataset starts with yes or no, It needs to preprocessed for all QA samples such as the resultant dataset is simple yes or no which ideal for classification task.
- The dataset is highly imbalanced where 80% of classes are yes, which could result in model leaning towards yes and hindering model generalization. [53]
- Some of the answers for the questions are also labelled as "No Answer", which is suitable for a conversation application rather than classifying statement yes or no using probabilistic approach.

4.1.2 BoolQ Dataset

Author : GoogleAI [53]
The second dataset used for this thesis is proposed by Google AI team in paper [53] , This dataset would be by the approach focused on lies and misinformation scenarios. The dataset consists of pairs of (Context,Question,Answers) collected mainly from Wikipedia domain. However unlike the previous datasets mentioned, The answers for the relevant questions are either True(yes) or False(no). The training and validation sets consists of 9.4k and 3.2k samples. The questions are short which are average of 9 tokens while the contexts are on the average of 108 tokens.

Annotation is done using three authors, In case of disagreement the best answer upon discussion is finalized. The annotation quality in comparison with gold-standard was about 90%. The questions covers wide range of topics from sports, music, history etc. The questions which return a Wikipedia page as one of top 5 results are part of the dataset. The Wikipedia page acts as an reference for annotators for further examination. The data collection and annotation is done is mainly done using 3 steps:

- The questions are considered good to be included in the dataset if they are clear in the meaning and request authentic answer and are not subjective. This is validated if the question returns a Wikipedia page.
- For the question asked, The annotator selects a contextual text which is sufficient to answer the question from the Wikipedia page.
- For the answers the annotators use True for yes and False for no. The overall annotation process is quite expensive as the annotators have to scan entire wikipedia page for relevant context.

The dataset as discussed is a pair of (Context,Question,Answer). For the selected question, The context or paragraph is selected from Wikipedia page. The table 4.5 shows an examples from BoolQ dataset where P is the paragraph contextual information from the page, Q is the asked factual question asked based on above considerations and finally the answer denoted by A.

BoolQ Dataset	
Type	Text
Q	Has the UK been hit by a hurricane?
P	The Great Storm of 1987 was a violent extratropical cyclone which caused casualties in England, France and the Channel Islands . . .
A	Yes. [An example event is given.]
Q	Does France have a Prime Minister and a President?
P	. . . The extent to which those decisions lie with the Prime Minister or President depends upon . . .
A	Yes. [Both are mentioned, so it can be inferred both exist.]
Q	Have the San Jose Sharks won a Stanley Cup?
P	. . . The Sharks have advanced to the Stanley Cup finals once, losing to the Pittsburgh Penguins in 2016. . .
A	No. [They were in the finals once, and lost.]

Table 4.5 Overview of BoolQ dataset [53]

4.2 Linear Classification Approach

The linear approach objective is to classify the given text as deceptive or real. As discussed in the section 2.5, The transformer based models are fine-tuned to specific task by adding suitable extra output layer corresponding to the problem. The paper [58] employs base BERT model with an additional LSTM layer. However for thesis, The utilization of LSTM layer is omitted. Data cleaning is done as an initial preprocessing step aimed at removing any punctuation, special characters, words with numbers etc. For training and testing model, The dataset is dynamically split for every run into 75% and 25% for training and testing respectively as a result contributing to the robustness of the model.

Each model will have a corresponding pretrained tokenizer, The tokenizer processes the natural language text and produces an embeddings and attention masks. The special tokens are added in the text by the tokenizer followed by embedding and attention. As discussed in the section 2.5, Each model will have a special tokens. In case of BERT they are [CLS][SEP]. The pretrained model is loaded and is trained on sequence of embeddings and attention masks to identify patterns and semantic relationships. The output of model is fed to our classifier which consists of a pooling layer to make sure the input characters are retained and doesn't go out of range as it is very critical for classification tasks. A dropout is added to make sure the model doesn't overfit and promotes generalization and finally a output layer with ReLU activation function to produce the final output. Fig 4.6 showcases the linear approach using transformer model and a customized classifier.

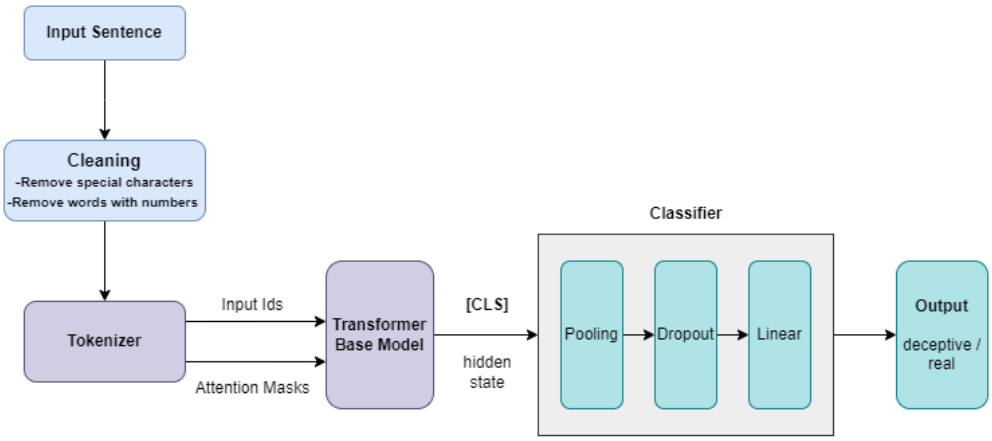

Figure 4.6 Proposed Linear Approach with Pre-trained Tokenizer and Model

The classifier is build from the ground up consisting of custom-built architecture. The first layer of the classifier is the pooling layer followed by a dropout layer, The dropout value of 0.6 is chosen which means about 60% chance that the neurons in the layer would be dropped from pooling layer above. This is to avoid any overfitting as mentioned above. The linear layer consists of (768,2) dimensions which corresponds to 768 hidden dimensions of the selected transformer model and 2 representing the labels which are truthful and deceptive.

4.3 Generative Adversarial Network Approach

The Gan-based transformer approach is similar to linear approach which is used to classify the text as truthful or deceptive, However here we use self-supervised learning with the objective of achieving similar accuracy using smaller percentage of labelled data. The approach is based on the paper [59] which specifically used BERT in a GAN framework and was able to achieve good results with just 50% of labelled data. As discussed in section 4.1, The datasets available for computer generated deceptive statements is scarce and annotation is a tedious process. Hence Gan-based approach will be ideal when there is small labelled data available for training.

As discussed in section 2.6, GAN consists of two networks namely generator and discriminator which play a minimax game. A variant of GAN known as self supervised GAN or SS-GAN proposed in the paper [60] is used for the approach. The concept of self-supervised learning focuses on improving model training such that it generates it's

own supervision labels from the available data rather than relying on externally provided labels in the dataset. The SS-GAN consists of self-supervised classification tasks which are part of main GAN used to improve learnings of Generator G and Discriminator D for classifier C. The eqns 4.1 and 4.2 represent the objective functions of Generator and Discriminator respectively.

$$minV_G(D,C,G) = V(D,G) - \lambda_g \phi(G,C) \quad (4.1) \quad [60]$$

$$maxV_D(D,C,G) = V(D,G) + \lambda_d \psi(G,C) \quad (4.2) \quad [60]$$

$$V(D,G) = E_{x\ P_data(x)}[logD(x)] + E_{z\ p_z(z)}[log(1 - D(G(z)))] \quad (2.31) \quad [44]$$

from eqn 4.1 and 4.2 $\phi(G,C)$ and $\psi(G,C)$ represent self-supervised learning tasks(SS tasks) for generator and discriminator learnings respectively. C is a classifier for SS tasks. $V(D,G)$ for both eqn 4.1 and 4.2 is given by original GAN objective function shown in eqn 2.31 as discussed in section 2.6. The goal of the discriminator is to classify the text as truthful or deceptive and the SS task $\psi(G,C)$ provides additional learning as it's learnings are added in eqn 4.1. The goal of the generator is to generate data based on it's learnings, However generator using SS task $\phi(G,C)$ can maximize it's learning without actually learning from the original data distribution, Hence the learnings of SS task are removed. As a result both the SS tasks play a minmax game to reach a point of saturation. fig 4.7 shows the GAN-based approach using transformer model.

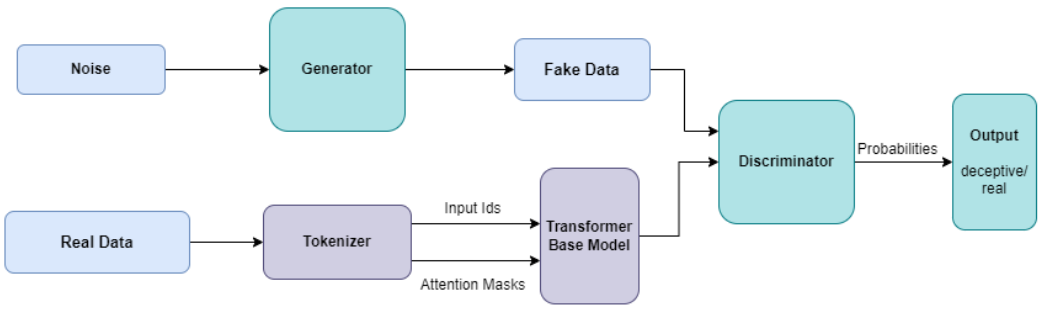

Figure 4.7 General Adversarial Approach with Pre-trained Tokenizer and Base model

In the paper [59] the linear transformer approach is extended by using SS-GAN architecture. The SS-GAN layers enable in Model fine-tuning and semi-supervised learning. The discriminator acts as a classifier and classifies the input text over k labels and the

generator is trained to generate realistic text with a uniform distribution noise between 0 and 1 which is a vector of size 100. The real data is fed to the tokenizer to obtain the embeddings or input ids along with their respective attention masks. These are again fed to the transformer model to obtain the states. The discriminator classifies the outputs of transformer model and generator as real or fake. For evaluation, The trained discriminator is used as the classifier and it's accuracy is measured for the test set. The training batch size is 64 and a dropout rate of 0.6 similar to linear approach.

4.4 Question & Answer Approach

The methodology is based on the advanced capabilities of the transformer-based models to make binary predictions, discerning between "Yes" and "No". This prediction process involves comprehensive understanding of the overall contextual information and the relevance of the query asked in relation to the contextual information. The approach aims to provide accurate answers for the context relevant query.

In the paper [53], The author used BERT and trained the model on MultiNLI dataset and re-trained on the training set to achieve accuracy of 80.4%. MultiNLI dataset proposed in the paper [61] has 433k samples with an objective of model training and evaluation to identify and understand the natural language sentence patterns. Since it is a tedious process, for the thesis we try to explore the large variants of each transformer model and achieve reasonable results by just using training set.

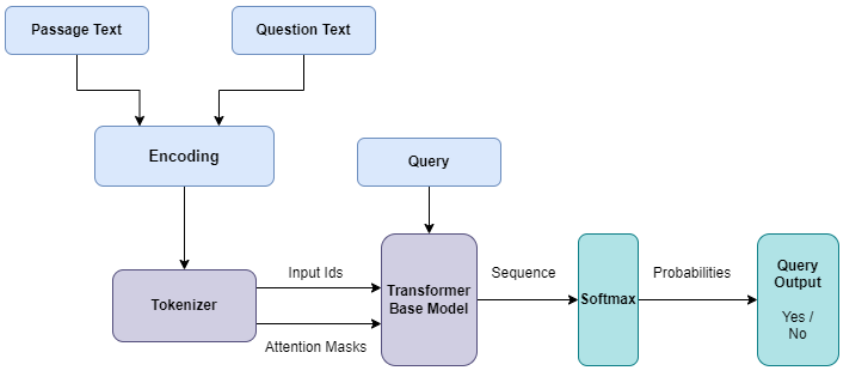

Figure 4.8 Proposed Question & Answer Approach with Pre-trained Tokenizer and Base model

In QnA approach, we utilize the boolQ dataset as our foundation. This dataset consists of structured format comprising of queries, passages or context, and their associated binary responses, specifically "Yes" or "No" as discussed in section 4.1. To adapt our model effectively to this dataset, we initiate a meticulous fine-tuning process.

To create a knowledgeable input representation as shown in fig 4.8, both the query and passage are encoded together. This encoding procedure results in a unified sequence which is fed to the tokenizer to obtain a single attention mask and corresponding IDs. The input embeddings and the attention masks are employed directly in the training process, enabling the model to understand the relationships between queries and passages.
The trained transformer model is evaluated, A softmax layer as discussed in section 2.2.3, Is added to get binary predictions. This allows the model to determine whether the given query and passage combination supports a "Yes" or "No" answer, thus enhancing the model's capacity to deliver accurate responses.

5 Evaluation and Discussion

5.1 Technologies

This section gives an overview of all the technologies and software packages used for the coding implementation for all the 3 approaches discussed in section 4.

5.1.1 PyTorch

Pytorch is a deep learning framework which provides libraries to build and train neural networks. Functionalities like addition of hidden layers, dimension sizes, activation functions can be easily added using inbuilt APIs of Pytorch. The core component of Pytorch are the tensors which are multidimensional arrays consists of values used for storing inputs and outputs. All the calculations are performed using tensors. All the embeddings and the attention masks generated for the input text in NLP are stored in tensors. Additional functionality of Graphics Processing Unit (GPU) is available which makes of available GPU for tensor computations which eventually reduce the training time of the models.

5.1.2 TensorFlow

Tensorflow similar to PyTorch is an deep learning framework which has libraries for building and training neural networks. Tensorflow in comparison to Pytorch provides more functionalities in the areas of visualization and Tensorflow models can be easily deployed using in build serving framework while Pytorch contains no similar framework.

5.1.3 Transformers

Transformers is a framework which provides APIs to easily download pretrained language models for fine-tuning. The major benefit of using pretrained models is that the models are trained on wide range of text and hence time and resources required to train the model from scratch are saved. Transformers provide pretrained models in wide range of application such as NLP, computer vision, Multimodal. Transformers are flexible and easily supported by different deep learning frameworks like PyTorch, TensorFlow etc. Hence a transformer model trained using one of the DL framework can be used by another in the development environment with any DL framework. The models are exported using a format ONNX for production deployment.

5.1.4 Google Colab

Google Colab is cloud-based jupyter notebook environment that allows developer to easily import software packages thus eliminating the need of local installations. It is also advantageous to run heavy graphic oriented DL models as colab provides free GPUs where certain computers may struggle with the computational workload.

5.1.5 NumPy

Numpy is one of the foundational python package for scientific computing as it supports creation of multi dimensional arrays and provides functionalities for wide range of tasks such as random number generation, linear transformations etc. DL packages use numpy to store tensors.

5.1.6 Pandas

Pandas is a software library widely utilized in data science for analytical purposes. Its primary strength lies in the introduction of DataFrames and Series, facilitating the seamless handling of large datasets with ease.

5.1.7 Scikit-learn

Scikit-learn software package consists of data mining and evaluation metrics for ML models. Most traditional ML techniques such as SVM, Decision trees, regression models can be easily implemented from the package. For evaluation scikit-learn provides metrics such as confusion matrix, F1-score, tf-idf.

5.1.8 NLTK

Natural Language Toolkit (NLTK) is toolkit library used primarily for Inputs which are Natural language. A wide range of NLP tasks such as tokenization, stopwords removal, stemming etc. can be easily performed in python using NLTK.

5.2 Fine-Tuning

Fine-tuning of a model refers to the procedure of taking already pre-trained model and futhur training it for a specfic task or on a dataset, As a result the model is able to adapt

or transfer it's previous learning to the new trained task. This concept is also known as transfer learning. BERT family models are pre-trained as discussed in section 2.5.2 on large corpus of data in a unsupervised scenario. However it is also crucial the model is pre-trained on various datasets which enhances the model to learn general patterns and representations from wide range of data. The model is fine-tuned onto the specific dataset that is relevant to a specific task.

One of the concept is the weights adjustments, where the model is fine-tuned onto new dataset but keeping the weights of some layers frozen. The layers which are frozen do not change and retain the knowledge learned from the original dataset. while the layers, usually final layers are adjusted to fit the new task. Another important concept is hyper-parameters tuning, where during fine-tuning the hyperparameters such as learning rate, dimensions, dropout, hidden layers, batch size are experimented for better optimization such that the model graph converges to optimal solution.

5.3 Baseline

Given that the training dataset is a balanced distribution of classes, our evaluation will focus on accuracy as the primary metric in contrast to F1-score which is more robust assessment of model performance when dealing with imbalanced classes.To assess performance, the Linear and GAN classifiers are benchmarked against LSTM and SVM baselines.

5.3.1 LSTM

To facilitate a comprehensive comparison, An LSTM classifier is built from the ground up, which will serve as a benchmark for evaluating fine-tuned models. This LSTM classifier is trained using a set of standardized parameters, including a batch size of 8, a dropout rate of 0.6, ReLU activation functions, and a learning rate set at 1e-3. The training process spans over 5 epochs, The model is trained on about 100k parameters. The learning rate is optimized accordingly for best convergence for LSTM. LSTM classifier was able to achieve 81.2% accuracy on validation set with a loss of 0.490 and a training loss of 0.547.

5.3.2 SVM

In recent research papers as discussed in section 3.3, Support Vector Machines (SVMs) have emerged as a noteworthy choice for text classification tasks, particularly in the areas

of deceptive reviews and fake news detection. These studies have revealed that SVMs tend to exhibit superior performance in such contexts. The primary approach employed in these investigations leverages the "tf-idf" method, where a maximum of 10,000 features are selected to represent the textual data. The linear kernal function is selected as the problem is a binary classification. SVM was able to achieve better optimization compared to LSTM with an accuracy of 86.98% on validation set

5.3.3 $BERT_{Uncased}$

For the BoolQ dataset, Noteworthy results were achieved in the Boolq paper[53], showcasing an impressive 80% accuracy when employing BERT which is pretrained on MultiNLI dataset [61]. Since MultiNLI is large set and requires more computational resources, the models are directly fine-tuned to the BoolQ dataset. However for the thesis $BERT_{Uncased}$ base model is chosen as a benchmark for evaluation of other models. Since the QnA approaches focuses more on language inference, large variants of the models are also evaluated against the benchmark.$BERT_{Uncased}$ was able to achieve an accuracy of 70.2% where the training process consisted of 3 epochs and validation loss of 0.087.

5.4 Results and discussions

For evaluation, three prominent transformer-based models are chosen. Each with unique strengths and characteristics which provide comprehensive assessment of their performance in the context of our specific tasks, offering insights into the suitability of each model for various NLP challenges.

BERT (Bidirectional Encoder Representations from Transformers): BERT serves as one of the foundational large language models, playing a pivotal role in shaping the landscape of natural language processing. It not only provides a strong baseline but also serves as a reference point for future enhancements and developments in the field.

RoBERTa: RoBERTa introduced by Facebook, has garnered significant attention due to its innovative approach of dynamic masking. In recent times, RoBERTa has demonstrated superior performance compared to BERT, making it a compelling choice for our evaluation. Its dynamic masking strategy has proven highly effective in language understanding tasks.

XLM-RoBERTa (Cross-lingual Language Model RoBERTa): XLM-RoBERTa represents the multilingual variant of RoBERTa, having undergone training on a multitude of languages. Recent studies [71] and [72] have highlighted the advantages of multilingual models, showcasing their enhanced understanding of semantic patterns in areas of hate speech and cyberbullying. Therefore, XLM-RoBERTa has been selected for our evaluation to harness its multilingual capabilities.

5.4.1 Linear Approach

A total of six experiments were conducted, employing three distinct models: BERT, RoBERTa, and XLM-RoBERTa. These models were fine-tuned using varying batch sizes, specifically 8 and 16. It's noteworthy that the choice of a higher batch size was constrained by memory limitations of Google Colab.
Across all six experiments, uniform hyperparameters were maintained. These hyperparameters include a learning rate set at 1e-6, a commonly employed value for fine-tuning tasks. A dropout rate of 0.6 was integrated to mitigate the risk of overfitting within the models. The training phase for all experiments spanned two epochs.

Linear classifier				
Model	Batch-size	t-loss	v-loss	Accuracy
Baseline(SVM)		-	-	86.58
Baseline(LSTM)	8	0.547	0.490	81.2
BERT		0.015	0.016	95.6
RoBERTa	8	0.011	0.012	96.4
XLM-RoBERTa		0.012	0.014	95.9
BERT		0.025	0.014	94.5
RoBERTa	16	0.016	0.008	95.7
XLM-RoBERTa		0.018	0.009	95

Table 5.1 illustrates the performance metrics and losses associated with various batch sizes for the Linear classifier.

The transformer based models clearly outperform both LSTM and SVM baselines. It was expected that models with small batch size would perform well as the model would learn from each individual examples. While higher batch size would promote robustness and more model generalization. The evaluation accuracy of all 3 models are similar, However RoBERTa slightly performer better in comparison to BERT and XLM-RoBERTa for both

batch sizes. The result is in accordance with the paper [73] which evaluated performances of various language models such as BERT, DistillBERT, RoBERTa, XLNet and ELECTRA in the area of emotion recognition NLP task where RoBERTa outperformed other models. Surprisingly XLM-RoBERTa being trained on multi lingual dataset was not able to achieve performance above RoBERTa.

5.4.2 Generative Adversarial Approach

A comprehensive set of 15 experiments were conducted, with each experiment majorly involving the selection of transformer models using distinct proportions of the original labeled training data. The primary focus of these experiments was to assess the accuracy performance across a spectrum of training data percentages.
Throughout these experiments, uniformity in certain hyperparameters were maintained. Specifically, a batch size of 64 was consistently employed, ensuring stable training conditions. Moreover, both the generator and discriminator were equipped with a learning rate set at 1e-6 for better optimization. To further enhance model generalization, a dropout rate of 0.6 was incorporated to mitigate the risk of overfitting.

GAN classifier					
Model	Training-data%	D-loss	G-loss	V-loss	Accuracy
Baseline(LSTM)	100	-	-	0.490	81.2
Baseline(SVM)	100	-	-	-	86.58
BERT	35	0.945	0.751	0.281	90
RoBERTa		1.070	0.722	0.394	85
XLM-RoBERTa		1.071	0.723	0.395	85
BERT	15	0.912	0.751	0.234	91.9
RoBERTa		0.944	0.723	0.399	86
XLM-RoBERTa		1.086	0.724	0.406	84.4
BERT	5	0.894	0.751	0.326	89
RoBERTa		0.965	0.723	0.491	82.4
XLM-RoBERTa		1.053	0.723	0.373	86.1
BERT	1	1.250	0.745	0.481	79.6
RoBERTa		1.173	0.718	0.574	76.1
XLM-RoBERTa		1.420	0.715	0.579	71.8
BERT	0.5	1.402	0.745	0.671	65.2
RoBERTa		1.259	0.772	0.682	66.6
XLM-RoBERTa		1.478	0.714	0.698	65.96

Table 5.2 illustrates the performance metrics and losses associated with various training data sizes for the GAN classifier.

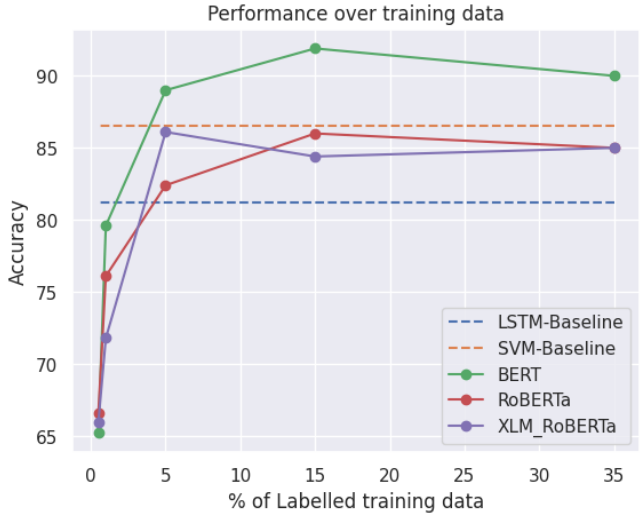

Figure 5.3: Performance accuracy against training data

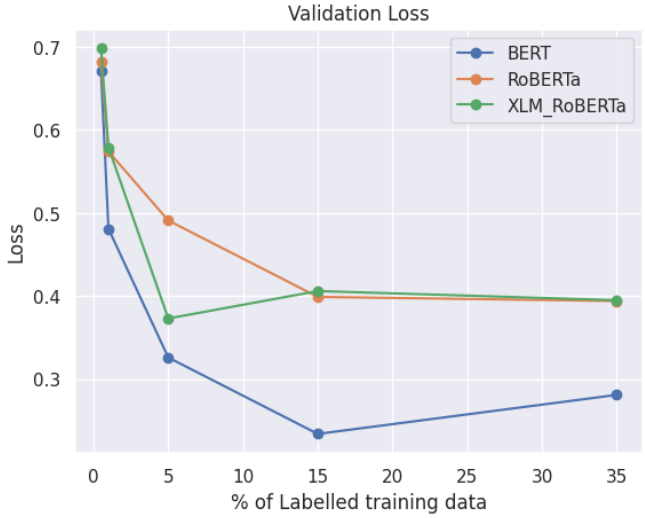

Figure 5.4: Validation loss across training data

Figure 5.5: Generator loss across training data

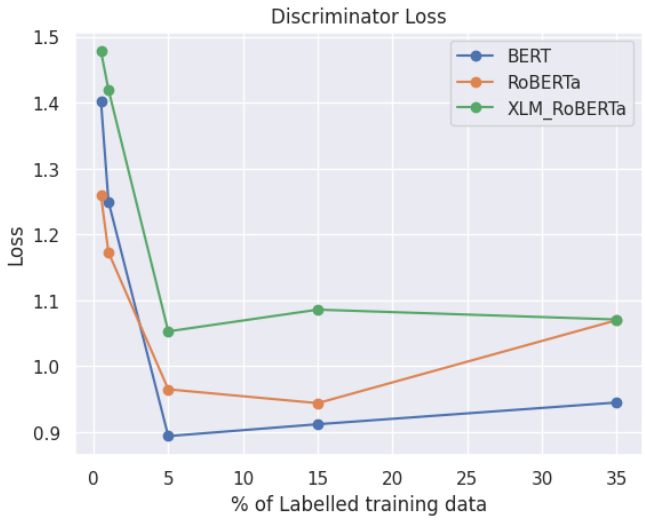

Figure 5.6: Discriminator loss across training data

For semi-supervised learning scenario, The training percentage of data is kept unchanged while the remaining data is masked in the training set. For instance from table 5.2, Training data of 35% suggests 35% of the training data is unchanged while the remaining training data is masked. The transformer models perform well in a semi-supervised scenario where the accuracy's and model generalization is stable with 5% of labelled data, However the models become unstable below 5% of labelled data as shown in fig 5.3 and fig 5.4. which showcases accuracy over training data and validation loss.
Surprisingly BERT outperforms both RoBERTa and XLM-RoBERTa and both the Baselines up till 5% labelled training data. As shown in fig 5.4 and fig 5.5 which provides generator and discriminator losses, BERT achieves point of equilibrium for generator and discriminator, while BERT generator loss is similar to other models, BERT discriminator achieves better generalization shown in fig 5.6. RoBERTa and XLM-RoBERTa achieves accuracy's similar to SVM baseline without any significant difference up till 5% labelled data.

5.4.3 Question & Answer Approach

For QnA classifier six experiments were conducted to explore the accuracy of models in classifying the query as yes or no based on a context. As QnA approach also focuses on language inference capabilities of the model, The large variants of the transformer models are also evaluated. However the batch size of 8 is fixed as the large variants of the models require more computational resources.
Similar to previous evaluations, Apart from base models of BERT, RoBERTa, XLM-RoBERTa, Their respective large variants are employed to measure performances. For the

fine-tuning process, a dropout rate of 0.6 is selected to main consistency and to mitigate overfitting, while a learning rate of 1e-5 is used for training models for total of 3 epochs.

Q & A classifier			
Model	Batch-size	v-loss	Accuracy
Baseline($BERT_{Uncased}$)	8	0.087	70.2
$BERT_{cased}$	8	0.116	71.3
$BERT_{Large}$		0.115	72.21
$RoBERTa_{base}$		0.104	77.46
$RoBERTa_{Large}$		0.082	84.55
$Xlm-RoBERTa_{base}$		0.086	72.62
$Xlm-RoBERTa_{Large}$		0.165	62.20

Table 5.7 illustrates the performance metrics and losses associated for the Q&A classifier.

A planet is a large, rounded astronomical body that is neither a star nor its remnant. The best available theory of planet formation is the nebular hypothesis, which posits that an interstellar cloud collapses out of a nebula to create a young protostar orbited by a protoplanetary disk. Planets grow in this disk by the gradual accumulation of material driven by gravity, a process called accretion.
The Solar System has at least eight planets: the terrestrial planets Mercury, Venus, Earth and Mars, and the giant planets Jupiter, Saturn, Uranus and Neptune. These planets each rotate around an axis tilted with respect to its orbital pole. All planets of the Solar System other than Mercury possess a considerable atmosphere, and some share such features as ice caps, seasons, volcanism, hurricanes, tectonics, and even hydrology. Apart from Venus and Mars, the Solar System planets generate magnetic fields, and all except Venus and Mercury have natural satellites. The giant planets bear planetary rings, the most prominent being those of Saturn

Figure 5.8 : Context from Wikipedia page about Planet.

Question: Mercury is a planet, Yes: 0.97, No: 0.03

Question: Planet is a star, Yes: 0.02, No: 0.98

Question: Mercury has a no atmosphere, Yes: 0.23, No: 0.77

Question: There are eight planets in solar system, Yes: 0.98, No: 0.02

Question: There are hundred planets in solar system, Yes: 0.15, No: 0.85

Question: Venus is closer to sun compared to mercury, Yes: 0.25, No: 0.75

Figure 5.9 : Relevant Questions asked to $RoBERTa_{large}$

$BERT_{uncased}$ model is the one of the basic Transformer model which is used as a baseline as discussed in section 5.3. The other models and variants performing better was expected as shown in Table 5.7. However the performances for all the BERT variants are similar which was unpredicted as it's expected that large variants perform well in comparison to their respective base models. Both variants of RoBERTa outperforms other models. As expected the significant difference is seen between the base and large variant of RoBERTa. $RoBERTa_{large}$ even outperforms the BERT trained on MultiNLI set which achieved accuracy of 80% as discussed in the paper [53]. Another unexpected outcome were the variants of XLM-RoBERTa where base version had similar accuracy to $BERT_{large}$, However $XLM - RoBERTa_{large}$ which was one of the largest model among the group was not able to match even the baseline accuracy.

Since $RoBERTa_{large}$ achieved good accuracy, A random context regarding planets consisting of about 200 words is fed to the fine-tuned $RoBERTa_{large}$. A series of queries are asked to validate whether the given query is yes(truth) or No(Lie) according to the given context. Fig 5.8 and Fig 5.9 shows the context and the queries asked for validation respectively. For some of the straightforward queries like "Mercury is a planet" and "There are eight planets in solar system" are answered with most precision, However even for unseen queries such as "Venus is closer to sun compared to Mercury" and "Mercury has no atmosphere" the model performs well and the answers are in line with the given context.

6 Conclusion and Future Work

The Fine-tuned transformer models were able to outperform the baselines of LSTM and SVM for the Linear approach. The performance of the 3 models were similar and exhibits the capabilities of self attention and dynamic word embeddings in NLP tasks. However for GAN classifier approach, BERT outperforms all the other models. The major difference lies in the type of masking where BERT uses static, While both XLM-RoBERTa and RoBERTa use dynamic masking. The models are used in combination with GAN which are difficult to train for NLP tasks as shown in the paper [74] where GANs are likely to overshoot for NLP when compared to computer vision tasks. However the models achieved generalization with good accuracy with 5-10% of the labelled data. For QnA classifier, RoBERTa outperforms the other models and exhibits the superior language inference capabilities. However the large variant of XLM-RoBERTa was unstable for the language inference task which could be due to XLM architecture being trained on maximum sequence size of 256 tokens for MLM and CLM tasks discussed in section 2.5.4 [42] while both BERT and RoBERTa are trained on 512 token size which could promote better inference capabilities.

For Future work, we can use the approaches as a building blocks for a complex architecture which can process and provide a deceptive score for the content. The paper [75] proposes a concept of Moral direction, where a series of questions are asked to trained BERT, GPT-3 and other models to check their moral compass based on their answers. Similarly for the given context , A trained model asks a series of queries based on the context to a trained validator model which provides a overall deceptive score for the content.

Bibliography

[1] Albert Zeyer, Parnia Bahar , Kazuki Irie , Ralf Schluter , Hermann Ney,"A comparison of transformer and LSTM encoder decoder models for ASR", IEEE, Feb 2020, Aachen, Germany, RWTH Aachen University,10.1109/ASRU46091.2019.9004025.

[2] Valentin Hofmann , Janet B. Pierrehumbert , Hinrich Schütze, "Dynamic Contextualized Word Embeddings, Association for Computational Linguistics", August 2021, Faculty of Linguistics, University of Oxford, 10.18653/v1/2021.acl-long.542.

[3] Diskha Khurana, Aditya koli, Kiran khatter, sukhdev singh,"NLP: current trends and challenges". In(2022). url: https://doi.org/10.1007/s11042-022-13428-4

[4] Shanaka kristombu, Sandeep, Jude, Mehrad,"Artificial Intelligence and Smart Vision for Building and Construction 4.0: Machine and Deep Learning Methods and Applications".In(June 2022), Australia, University of Melbourne, Article in Automation in Construction. url : https://www.researchgate.net/publication/361501987

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

[6] Tom M. Mitchell. Machine learning. 1997. isbn: 0070428077.

[7] Ahmad Hammoudeh, "A Concise Introduction to Reinforcement Learning", In Feb 2018, DOI: 10.13140/RG.2.2.31027.53285, url: https://www.researchgate.net/publication/323178749

[8] Jeffrey Pennington, Richard Socher, Christopher D manning, "Glove: Global Vectors for Word Representation" In Jan 2014, DOI: 10.3115/v1/D14-1162, url: https://www.researchgate.net/publication/284576917

[9] Nitin INDURKHYA and Fred J Damerau. HANDBOOK OF NATURAL LANGUAGE PROCESSING. Chapman & Hall/CRC, 2010. isbn: 978-1-4200-8593-8. url: http://portal.acm.org/citation.cfm?doid=1220835.1220877.

[10] Jonathan J. Webster and Chunyu Kit. “Tokenization as the initial phase in NLP”. In: Proceedings of the 14th conference on Computational linguistics -. Vol. 4. Morristown, NJ, USA: Association for Computational Linguistics, 1992, p. 1106. doi: 10.3115/992424.992434.

[11] Matthew James Denny and Arthur Spirling. "Text Preprocessing For Unsupervised Learning: Why It Matters, When It Misleads, And What To Do About It". In: SSRN Electronic Journal 26.02 (Apr. 2017), pp. 168–189. issn: 1556-5068. doi: 10 . 2139 / ssrn . 2849145.

[12] "Stop Word",Wikipedia, Wikimedia Foundation, last modified: 27 september 2023, url:https://en.wikipedia.org/wiki/Stop_word

[13] Julie Beth Lovins. “Development of a stemming algorithm”. In: Mechanical Translation and Computational Linguistics 11.June (1968).

[14] "Stemming",Wikipedia, Wikimedia Foundation, last modified: 16 April 2023, url:https://en.wikipedia.org/wiki/Stemming

[15] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008. ISBN: 0521865719. HTML-edition:https://nlp.stanford.edu/IR-book/html/htmledition/irbook.html

[16] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, Introduction to Information Retrieval, Cambridge University Press. 2008. Chapter-6 "Vector space classification", ISBN: 0521865719. HTML-edition:https://nlp.stanford.edu/IR-book/html/htmledition/irbook.html

[17] "Lemmatization",Wikipedia, Wikimedia Foundation, last modified: 7 september 2023, url:https://en.wikipedia.org/wiki/Lemmatization

[18] Jurgen Schmidhuber, "Deep Learning in Neural Networks: An Overview", In Oct 2014, University of Lugano & SUPSI, Switzerland, arXiv:1404.7828 v4

[19] Enzo Grossi,Massimo Buscema,"Introduction to artificial neural networks",In jan 2008, DOI:10.1097/MEG.0b013e3282f198a0, url:https://www.researchgate.net/publication/5847739

[20] Haider Khalaf Allamy, "METHODS TO AVOID OVER-FITTING AND UNDER-FITTING IN SUPERVISED MACHINE LEARNING (COMPARATIVE STUDY)" In Dec 2014, University of Misan, url:at: https://www.researchgate.net/publication/295198699

[21] Shiv ram dubey, satish kumar singh, bidyut baran chaudhuri, "Activation functions in Deep Learning: A comprehensive survey and benchmark" In Jun 2022, IIT Allahabad India, arXiv:2109.14545v3

[22] Kshitij Khurana, "Activation Functions in Neural Networks", posted: Nov 2019 medium.com, url: https://medium.com/@kshitijkhurana3010/activation-functions-in-neural-networks-ed88c56b611b

[23] Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall,"Activation Functions: Comparison of Trends in Practice and Research for Deep Learning", In Nov 2018, arXiv:1811.03378v1

[24] Alex Sherstinsky, "Fundamentals of Recurrent Neural Network(RNN) and Long Short-Term Memory(LSTM) Network", In July 2023, Elsevier journal "Physica D: Nonlinear Phenomena", Volume 404, March 2020, arXiv:1808.03314v10

[25] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Théophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, Timothy Lillicrap,"Relational recurrent neural networks", In June 2018, University College,London, United kingdom,arXiv:1806.01822v2

[26] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. CoRR, abs/1810.04805.

[27] Hasin Rehana, Nur Bengisu Çam, Mert Basmaci, Yongqun He, Arzucan Özgür, Junguk Hur,"Evaluation of GPT and BERT-based models on identifying protein-protein interactions in biomedical text",In March 2023,arXiv:2303.17728 , url:https://doi.org/10.48550/arXiv.2303.17728

[28] Yi Liao, Xin Jiang, Qun Liu, "Probabilistically Masked Language Model Capable of Autoregressive Generation in Arbitrary Word Order",In Apr 2020, arXiv:2004.11579, url:https://doi.org/10.48550/arXiv.2004.11579

[29] Felipe Almeida,Geraldo Xexeo,"Word Embeddings: A Survey",In May 2023, Computer and Systems Engineering Program ,Federal University of Rio de Janeiro, Brazil, arXiv:1901.09069v2

[30] Deepak Suresh Asudani, Naresh Kumar Nagwani,Pradeep Singh,"Impact of word embedding models on text analytics in deep learning environment: a review",In Feb 2023, National Institute of Technology, Raipur,India, url: https://doi.org/10.1007/s10462-023-10419-1

[31] Tomas Mikolov,Greg Corrado,Kai Chen,Jeffrey Dean "Efficient Estimation of Word Representations in Vector Space",In Sep 2013, Google Inc, arXiv:1301.3781v3

[32] Valentin Hofmann, Janet B. Pierrehumberty, Hinrich Schutze,"Dynamic ContextualizedWord Embeddings", In Aug 2021,Faculty of Linguistics, University of Oxford,DOI:10.18653/v1/2021.acl-long.542,url: https://aclanthology.org/2021.acl-long.542

[33] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner,Christopher Clark, Kenton Lee, Luke Zettlemoyer,"Deep contextualized word representations",In Mar 2018,Allen Institute for Artificial Intelligence,arXiv:1802.05365v2

[34] Alec Radford, Karthik Narasimhan,Tim Salimans,Ilya Sutskever,"Improving Language Understanding by Generative Pre-Training" In 2018, OpenAI.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin,"Attention Is All You Need", In Aug 2023, arXiv:1706.03762v7, url: https://doi.org/10.48550/arXiv.1706.03762

[36] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova,"BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", In May 2019, arXiv:1810.04805v2,url: https://doi.org/10.48550/arXiv.1810.04805

[37] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean,"Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation",In Sep 2016, arXiv:1609.08144, url: https://doi.org/10.48550/arXiv.1609.08144

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov,"RoBERTa: A Robustly Optimized BERT Pretraining Approach",In July 2019, Facebook AI, arXiv:1907.11692, url: https://doi.org/10.48550/arXiv.1907.11692

[39] "Fine Grained Named Entity Recognition with Transformer", AllenNLP, Domain: Papers with code, Last modified 15 march 2021, url: https://paperswithcode.com/lib/allennlp/fine-grained-named-entity-recognition-with

[40] Rico Sennrich, Barry Haddow,Alexandra Birch,"Neural Machine Translation of Rare Words with Subword Units" In June 2016, arXiv:1508.07909, url https://doi.org/10.48550/arXiv.1508.07909

[41] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, Veselin Stoyanov,"Unsupervised Cross-lingual Representation Learning at Scale",In April 2020, arXiv:1911.02116, url: https://doi.org/10.48550/arXiv.1911.02116

[42] Guillaume Lample,Alexis Conneau,"Cross-lingual Language Model Pretraining",In Jan 2019, Facebook AI, arXiv:1901.07291v1,url: https://doi.org/10.48550/arXiv.1901.07291

[43] "Overview of GAN Structure",developers.google.com, Last modified: 18-07-2022,Last accessed 10-10-2023,url: https://developers.google.com/machine-learning/gan/gan_structure

[44] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio,"Generative Adversarial Networks",In Jun 2014,arXiv:1406.2661,url: https://doi.org/10.48550/arXiv.1406.2661

[45] Laura Koesten, Elena Simperl, Emilia Kacprzak,Tom Blount, Jeni Tennison,"Everything you always wanted to know about a dataset: studies in data summarisation",In Oct 2018,arXiv:1810.12423v1,url: https://doi.org/10.48550/arXiv.1810.12423

[46] Vlad Sandulescu,Martin Ester,"Detecting Singleton Review Spammers Using Semantic Similarity",In Sep 2016, arXiv:1609.02727v1, url: https://doi.org/10.48550/arXiv.1609.02727

[47] Myle Ott,Yejin Choi,Claire Cardie,Jeffrey T. Hancock,"Finding Deceptive Opinion Spam by Any Stretch of the Imagination",In Jul 2011,Cornell University,USA,arXiv:1107.4557v1,url: https://arxiv.org/pdf/1107.4557.pdf

[48] Kyung-Hyan Yooa, Ulrike Gretzela,"Comparison of Deceptive and Truthful Travel Reviews", In Jan 2009,Texas A&M University, USA,DOI:10.1007/978-3-211-93971-0_4, url: https://www.researchgate.net/publication/221357520

[49] Joni Salminen, Chandrashekhar Kandpal, Ahmed Mohamed Kamel, Soon-gyo Jung, Bernard J. Jansen,"Creating and detecting fake reviews of online products ",In Jan 2022, Journal of Retailing and Consumer Services 64:102771, DOI:10.1016/j.jretconser.2021.102771

[50] Sherry He, Brett Hollenbeck, Davide Proserpio, "The Market for Fake Reviews", In Oct 2022, Marketing Science, 2022,Vol.41(5),p.896-921, url: https://doi.org/10.1287/mksc.2022.1353

[51] Nitin Jindal, Bing Liu, "Opinion Spam and Analysis ",In 2008, University of Illinois, USA, url: https://www.cs.uic.edu/ liub/FBS/opinion-spam-WSDM-08.pdf

[52] Jeremy Howard, Sebastian Ruder,"Universal Language Model Fine-tuning for Text Classification",In July 2018,Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),Melbourne, Australia,328–339,url: https://aclanthology.org/P18-1031

[53] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins , Kristina Toutanova, "BoolQ: Exploring the Surprising

Difficulty of Natural Yes/No Questions",In May 2019, Google AI,Paul G. Allen School of CSE, University of Washington, arXiv:1905.10044, url : https://doi.org/10.48550/arXiv.1905.10044

[54] Siva Reddy, Danqi Chen, Christopher D. Manning, "CoQA: A Conversational Question Answering Challenge",In Mar 2019, Stanford University, arXiv:1808.07042, url: https://doi.org/10.48550/arXiv.1808.07042

[55] Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wen-tau Yih, Yejin Choi, Percy Liang, Luke Zettlemoyer,"QuAC : Question Answering in Context",In Aug 2018, Allen Institute for Artificial Intelligence,Stanford University, arXiv:1808.07036, url: https://doi.org/10.48550/arXiv.1808.07036

[56] Zhilin Yang, Peng Qi, Saizheng Zhang,Yoshua Bengio,William W. Cohen,Ruslan Salakhutdinov,Christopher D. Manning,"HOTPOTQA: A Dataset for Diverse, Explainable Multi-hop Question Answering",In Sep 2018, arXiv:1809.09600v1, url: https://doi.org/10.48550/arXiv.1809.09600

[57] Marzieh Saeidi, Max Bartolo, Patrick Lewis, Sameer Singh,Tim Rocktaschel, Mike Sheldon, Guillaume Bouchard ,Sebastian Riede,"Interpretation of Natural Language Rules in Conversational Machine Reading", In Aug 2018, Bloomsbury AI, arXiv:1809.01494v1, url: https://doi.org/10.48550/arXiv.1809.01494

[58] Nishant Rai, Deepika Kumar , Naman Kaushik, Chandan Raj, Ahad Ali,"Fake News Classification using transformer based enhanced LSTM and BERT",In Mar 2022, International Journal of Cognitive Computing in Engineering, DOI:10.1016/j.ijcce.2022.03.003

[59] Danilo Croce,Giuseppe Castellucci,Roberto Basili,"GAN-BERT: Generative Adversarial Learning for Robust Text Classification with a Bunch of Labeled Examples",In July 2020,Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics,DOI:10.18653/v1/2020.acl-main.191,url: https://aclanthology.org/2020.acl-main.191

[60] Ngoc-Trung Tran, Viet-Hung Tran, Ngoc-Bao Nguyen, Linxiao Yang, Ngai-Man Cheung,"Self-supervised GAN: Analysis and Improvement with Multi-class Minimax Game",Singapore University of Technology and Design,In Jan 2020, arXiv:1911.06997v2, url: https://doi.org/10.48550/arXiv.1911.06997

[61] Adina Williams,Nikita Nangia,Samuel R. Bowman,"A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference",In Feb 2018, arXiv:1704.05426v4,url: https://doi.org/10.48550/arXiv.1704.05426

[62] Shigeki Karita,Nanxin Chen, Tomoki Hayashi,"A COMPARATIVE STUDY ON TRANSFORMER VS RNN IN SPEECH APPLICATIONS", In Sep 2019, arXiv:1909.06317v2, url: https://doi.org/10.48550/arXiv.1909.06317

[63] Imran Sheikh, Emmanuel Vincent, Irina Illina,"Transformer versus LSTM Language Models trained on Uncertain ASR Hypotheses in Limited Data Scenarios",In Jun 2022, Proceedings of the Thirteenth Language Resources and Evaluation Conference,European Language Resources Association,France, P 393-399, url: https://aclanthology.org/2022.lrec-1.41

[64] Raunak Joshi, Abhishek Gupta,"PERFORMANCE COMPARISON OF SIMPLE TRANSFORMER AND RES-CNN-BILSTM FOR CYBER-BULLYING CLASSIFICATION",In Jun 2022, arXiv:2206.02206v1,url: https://doi.org/10.48550/arXiv.2206.02206

[65] Gizem Arasa, Didem Makaroglua, Seniz Demirc, Altan Cakir, "An Evaluation of Recent Neural Sequence Tagging Models in Turkish Named Entity Recognition",In May 2020, arXiv:2005.07692v2, url: https://doi.org/10.48550/arXiv.2005.07692

[66] Savelie Cornegruta, Robert Bakewell, Samuel Withey,Giovanni Montana,"Modelling Radiological Language with Bidirectional Long Short-Term Memory Networks",In Sep 2016, King's College London, UK, arXiv:1609.08409v1, url : https://doi.org/10.48550/arXiv.1609.08409

[67] Theodoros Evgeniou, Massimiliano Pontil,"Support Vector Machines: Theory and Applications",In Sep 2001, DOI: 10.1007/3-540-44673-7_12,url: https://www.researchgate.net/publication/221621494

[68] Amisha Sinha, Mohnish Raval, S Sindhu,"Machine Learning Based Detection of Deceptive Tweets on Covid-19",In Jun 2021, International Journal of Engineering and Advanced Technology,DOI:10.35940/ijeat.E2831.0610521, url: https://doi.org/10.35940/ijeat.e2831.0610521

[69] Yasmen Wahba, Nazim Madhavji, John Steinbacher,"A Comparison of SVM against Pre-trained Language Models (PLMs) for Text Classification Tasks",In Nov 2022, arXiv:2211.02563, url: https://doi.org/10.48550/arXiv.2211.02563

[70] Yu-Feng Li, Zhi-Hua Zhou,"Improving Semi-Supervised Support Vector Machines Through Unlabeled Instances Selection",In May 2011, arXiv:1005.1545, url: https://doi.org/10.48550/arXiv.1005.1545

[71] Li Xu, Jun Zeng, Shi Chen, "yasuo at HASOC2020: Fine-tune XML-RoBERTa for Hate Speech Identification",In Dec 2020, FIRE 20, Forum for Information Retrieval Evaluation,Yunnan University, Kunming, P.R. China.

[72] Tharindu Ranasinghe,Marcos Zampieri,"Multilingual Offensive Language Identification with Cross-lingual Embeddings",In Nov 2020, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP),DOI:10.18653/v1/2020.emnlp-main.470, url: https://www.researchgate.net/publication/347236560

[73] Diogo Cortiz, "Exploring Transformers in Emotion Recognition: a comparison of BERT, DistillBERT, RoBERTa, XLNet and ELECTRA",In Apr 2021,arXiv:2104.02041, url: https://doi.org/10.48550/arXiv.2104.02041

[74] David Alvarez-Melis, Vikas Garg, Adam Tauman Kalai, "Why GANs are overkill for NLP", In May 2022, arXiv:2205.09838, url: https://doi.org/10.48550/arXiv.2205.09838

[75] Patrick Schramowski, Cigdem Turan, Nico Andersen, Constantin A. Rothkopf, Kristian Kersting,"Large Pre-trained Language Models Contain Human-like Biases of What is Right and Wrong to Do", In Feb 2022, arXiv:2103.11790v3, url : https://doi.org/10.48550/arXiv.2103.11790

1

[1] https://github.com/hishamp3/MasterThesis-Lies-DeceptiveText

Appendix

A SVM Baseline

[2]

```
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
import matplotlib.pyplot as plt
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
import pickle
import time
import re

import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

import pandas as pd
df = pd.read_csv("./sample_data/fake reviews dataset.csv",usecols=["
    text_","label"])

import re
import string
def clean_text(text):
    # to lower case
    text = text.lower()
    # remove links
    text = re.sub('https:\/\/\S+', '', text)
    # remove punctuation
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    # remove next line
    text = re.sub(r'[^ \w\.]', '', text)
    # remove words containing numbers
    text = re.sub('\w*\d\w*', '', text)

    return text
df['text'] = df.text_.apply(lambda x: clean_text(x))
df['fake']=df['label'].apply(lambda x: 1 if x=='CG' else 0)
cleanedData = []

lemma = WordNetLemmatizer()
swords = stopwords.words("english")
for text in df["text"]:

    # Cleaning links
    text = re.sub(r'http\S+', '', text)

    # Cleaning everything except alphabetical and numerical characters
```

[2] https://github.com/hishamp3/MasterThesis-Lies-DeceptiveText/blob/main/SVM_baseline.ipynb

```
47      text = re.sub("[^a-zA-Z0-9]"," ",text)
48
49      # Tokenizing and lemmatizing
50      text = nltk.word_tokenize(text.lower())
51      text = [lemma.lemmatize(word) for word in text]
52
53      # Removing stopwords
54      text = [word for word in text if word not in swords]
55
56      # Joining
57      text = " ".join(text)
58
59      cleanedData.append(text)
60
61 vectorizer = CountVectorizer(max_features=10000)
62 BOW = vectorizer.fit_transform(cleanedData)
63 from sklearn.model_selection import train_test_split
64 x_train,x_test,y_train,y_test = train_test_split(BOW,np.asarray(df["
       fake"]))
65
66 from sklearn.svm import SVC
67 start_time = time.time()
68
69 model = SVC(kernel='linear')
70 model.fit(x_train,y_train)
71
72 end_time = time.time()
73 process_time = round(end_time-start_time,2)
74 print("Fitting SVC took {} seconds".format(process_time))
75 predictions = model.predict(x_test)
76
77 from sklearn.metrics import accuracy_score,confusion_matrix
78 print("Accuracy of model is {}%".format(accuracy_score(y_test,
       predictions) * 100))
```

B LSTM Baseline

[3]

```
1 !pip install -q -U "tensorflow-text"
2 import numpy as np
3 import tensorflow as tf
4 import matplotlib.pyplot as plt
5 from keras.layers import Embedding
6
7 import pandas as pd
8 df = pd.read_csv("./sample_data/fake reviews dataset.csv",usecols=["
      text_","label"])
9
10 import re
11 import string
12 def clean_text(text):
13     # to lower case
14     text = text.lower()
```

[3] https://github.com/hishamp3/MasterThesis-Lies-DeceptiveText/blob/main/LSTM_baseline.ipynb

```
    # remove links
    text = re.sub('https:\/\/\S+', '', text)
    # remove punctuation
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    # remove next line
    text = re.sub(r'[^ \w\.]', '', text)
    # remove words containing numbers
    text = re.sub('\w*\d\w*', '', text)

    return text

df['text'] = df.text_.apply(lambda x: clean_text(x))
df['fake']=df['label'].apply(lambda x: 1 if x=='CG' else 0)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df['text'],df['fake
    '], stratify=df['fake'])
X_train.head(4)
train_dataset = X_train.to_frame().join(y_train)

# Defining pre-processing parameters
max_len = 10
trunc_type = 'post'
padding_type = 'post'
oov_tok = '<OOV>' # out of vocabulary token
vocab_size = 500

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
tokenizer = Tokenizer(num_words = vocab_size,
                      char_level = False,
                      oov_token = oov_tok)
tokenizer.fit_on_texts(X_train)
word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(X_train)
training_padded = pad_sequences(training_sequences,
                                maxlen = max_len,
                                padding = padding_type,
                                truncating = trunc_type)

testing_sequences = tokenizer.texts_to_sequences(X_test)
testing_padded = pad_sequences(testing_sequences,
                               maxlen = max_len,
                               padding = padding_type,
                               truncating = trunc_type)

print('Shape of training tensor: ', training_padded.shape)
print('Shape of testing tensor: ', testing_padded.shape)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense, Embedding,
    Dropout, GlobalAveragePooling1D, Flatten, SpatialDropout1D,
    Bidirectional

vocab_size = 500
embedding_dim = 16
drop_value = 0.6
n_dense = 24
```

```
70
71 # Define parameter
72 n_lstm = 128
73 drop_lstm = 0.6
74 # Define LSTM Model
75 model1 = Sequential()
76 model1.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
77 model1.add(SpatialDropout1D(drop_lstm))
78 model1.add(LSTM(n_lstm, return_sequences=False))
79 model1.add(Dropout(drop_lstm))
80 model1.add(Dense(1, activation='ReLU'))
81 model1.summary()
82
83 from tensorflow.keras.optimizers import Adam
84 adamOpti = Adam(learning_rate = 1e-3)
85 model1.compile(loss = 'binary_crossentropy',
86                optimizer = adamOpti,
87                metrics = ['accuracy'])
88 num_epochs = 5
89 early_stop = EarlyStopping(monitor='val_loss', patience=2)
90 history = model1.fit(training_padded,
91                      y_train,
92                      batch_size=8,
93                      epochs=num_epochs)
94
95 loss, accuracy = model1.evaluate(testing_padded,y_test,batch_size=8)
96
97 print(f'Loss: {loss}')
98 print(f'Accuracy: {accuracy}')
```

[4]

C Linear Classification

```
1 !pip install transformers
2 !pip install sentencepiece
3 import pandas as pd
4 import re
5 import string
6
7 def clean_text(text):
8     # to lower case
9     text = text.lower()
10    # remove links
11    text = re.sub('https:\/\/\S+', '', text)
12    # remove punctuation
13    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
14    # remove next line
15    text = re.sub(r'[^ \w\.]', '', text)
16    # remove words containing numbers
17    text = re.sub('\w*\d\w*', '', text)
18
19    return text
20
```

[4] https://github.com/hishamp3/MasterThesis-Lies-DeceptiveText/blob/main/Linear_Classifier.ipynb

```
df = pd.read_csv("./sample_data/fake reviews dataset.csv",usecols=["text_","label"])
df['text'] = df.text_.apply(lambda x: clean_text(x))

model_name = 'xlm-roberta-base'
#model_name = 'roberta-base'
#model_name = 'bert-base-uncased'

from transformers import BertTokenizer
from transformers import RobertaTokenizer, TFRobertaModel
from transformers import XLNetTokenizer, XLNetModel
from transformers import AutoTokenizer,XLMRobertaForMaskedLM
import torch
import numpy as np
from transformers import BertTokenizer

tokenizer = AutoTokenizer.from_pretrained(model_name)
labels = {'OR':0,
          'CG':1
          }

class Dataset(torch.utils.data.Dataset):

    def __init__(self, df):

        self.labels = [labels[label] for label in df['label']]
        self.texts = [tokenizer(text,
                                padding='max_length', max_length = 512, truncation=True,
                                return_tensors="pt") for text in df['text']]

    def classes(self):
        return self.labels

    def __len__(self):
        return len(self.labels)

    def get_batch_labels(self, idx):
        # Fetch a batch of labels
        return np.array(self.labels[idx])

    def get_batch_texts(self, idx):
        # Fetch a batch of inputs
        return self.texts[idx]

    def __getitem__(self, idx):

        batch_texts = self.get_batch_texts(idx)
        batch_y = self.get_batch_labels(idx)

        return batch_texts, batch_y

np.random.seed(112)
df_train, df_val, df_test = np.split(df.sample(frac=1, random_state=42),
                                     [int(.8*len(df)), int(.9*len(df))])
```

```
74
75 print(len(df_train),len(df_val), len(df_test))
76
77 from torch import nn
78 from transformers import BertModel
79 from transformers import RobertaTokenizer, RobertaModel
80 from transformers import XLMRobertaForCausalLM, AutoConfig
81 from transformers import AutoModelForMaskedLM
82 from transformers import AutoModel
83
84 class Classifier(nn.Module):
85
86     def __init__(self, dropout=0.6):
87
88         super(Classifier, self).__init__()
89         self.model = AutoModel.from_pretrained(model_name)
90         self.dropout = nn.Dropout(dropout)
91         self.linear = nn.Linear(768, 2)
92         self.relu = nn.ReLU()
93
94     def forward(self, input_id, mask):
95
96         _, pooled_output = self.model(input_ids= input_id,
               attention_mask=mask,return_dict=False)
97         dropout_output = self.dropout(pooled_output)
98         linear_output = self.linear(dropout_output)
99         final_layer = self.relu(linear_output)
100
101         return final_layer
102
103 from torch.optim import Adam
104 from tqdm import tqdm
105
106 def train(model, train_data,learning_rate, epochs):
107
108     train = Dataset(train_data)
109
110     train_dataloader = torch.utils.data.DataLoader(train, batch_size=8,
          shuffle=True)
111
112     use_cuda = torch.cuda.is_available()
113     device = torch.device("cuda" if use_cuda else "cpu")
114
115     criterion = nn.CrossEntropyLoss()
116     optimizer = Adam(model.parameters(), lr= learning_rate)
117
118     if use_cuda:
119
120             model = model.cuda()
121             criterion = criterion.cuda()
122
123     for epoch_num in range(epochs):
124
125             total_acc_train = 0
126             total_loss_train = 0
127
128             for train_input, train_label in tqdm(train_dataloader):
129
```

```
130                 train_label = train_label.to(device)
131                 mask = train_input['attention_mask'].to(device)
132                 input_id = train_input['input_ids'].squeeze(1).to(
                        device)
133
134                 output = model(input_id, mask)
135
136                 batch_loss = criterion(output, train_label.long())
137                 total_loss_train += batch_loss.item()
138
139                 acc = (output.argmax(dim=1) == train_label).sum().item
                        ()
140                 total_acc_train += acc
141
142                 model.zero_grad()
143                 batch_loss.backward()
144                 optimizer.step()
145
146             print(
147                 f'Epochs: {epoch_num + 1} | Train Loss: {
                        total_loss_train / len(train_data): .3f} \
148                 | Train Accuracy: {total_acc_train / len(train_data):
                        .3f}'
149                 )
150
151 import gc
152 gc.collect()
153 torch.cuda.empty_cache()
154 # Training
155 EPOCHS = 2
156 model = Classifier()
157 LR = 1e-6
158 train(model, df_train, LR, EPOCHS)
159
160 from sklearn.metrics import confusion_matrix
161 from sklearn.metrics import f1_score
162
163 def evaluate(model, test_data):
164
165     test = Dataset(test_data)
166
167     test_dataloader = torch.utils.data.DataLoader(test, batch_size=8)
168
169     use_cuda = torch.cuda.is_available()
170     device = torch.device("cuda" if use_cuda else "cpu")
171
172     criterion = nn.CrossEntropyLoss()
173     if use_cuda:
174
175         model = model.cuda()
176         criterion = criterion.cuda()
177
178     total_loss_test = 0
179     total_acc_test = 0
180     y_pred = []
181     y_true = []
182     with torch.no_grad():
183
```

```
184         for test_input, test_label in test_dataloader:
185
186               test_label = test_label.to(device)
187               mask = test_input['attention_mask'].to(device)
188               input_id = test_input['input_ids'].squeeze(1).to(device)
189
190               output = model(input_id, mask)
191
192               batch_loss = criterion(output, test_label.long())
193               total_loss_test += batch_loss.item()
194
195               acc = (output.argmax(dim=1) == test_label).sum().item()
196               total_acc_test += acc
197
198               y_pred.extend((torch.max(torch.exp(output), 1)[1]).data.
                      cpu().numpy())
199               y_true.extend(test_label.data.cpu().numpy())
200               cf_matrix = confusion_matrix(y_true, y_pred)
201               score_f1 = f1_score(y_true, y_pred)
202
203     print(f'Test Accuracy: {total_acc_test / len(test_data): .3f}|Val
          Loss: {total_loss_test / len(test_data): .3f}')
204     print(f'Confusion Matrix: {cf_matrix}')
205     print(f'F1 score: {score_f1}')
206
207 frames = [df_val, df_test]
208 test_set = pd.concat(frames)
209 # Evaluation
210 evaluate(model, test_set)
```

D GAN classifer

[5]

```
1  !pip install transformers
2  import random
3  import numpy as np
4  import time
5  import math
6  import datetime
7  import torch
8  import io
9  import torch.nn.functional as F
10 import torch.nn as nn
11 from transformers import AutoModel,AutoTokenizer,AutoConfig
12 from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
       SequentialSampler
13
14 ##Set random values
15 seed_val = 42
16 random.seed(seed_val)
17 np.random.seed(seed_val)
18 torch.manual_seed(seed_val)
19 if torch.cuda.is_available():
```

[5] https://github.com/hishamp3/MasterThesis-Lies-DeceptiveText/blob/main/Gan_Classifier.ipynb

```
  torch.cuda.manual_seed_all(seed_val)
# If there's a GPU available...
if torch.cuda.is_available():
    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")
    print('There are %d GPU(s) available.' % torch.cuda.device_count())
    print('We will use the GPU:', torch.cuda.get_device_name(0))
# If not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")

#-------------------------------
#  Transformer parameters
#-------------------------------
max_seq_length = 64
batch_size = 64

#-------------------------------
#  GAN-BERT specific parameters
#-------------------------------
# number of hidden layers in the generator,
# each of the size of the output space
num_hidden_layers_g = 1;
# number of hidden layers in the discriminator,
# each of the size of the input space
num_hidden_layers_d = 1;
# size of the generator's input noisy vectors
noise_size = 100
# dropout to be applied to discriminator's input vectors
out_dropout_rate = 0.6

# Replicate labeled data to balance poorly represented datasets,
# e.g., less than 1% of labeled material
apply_balance = True

#-------------------------------
#  Optimization parameters
#-------------------------------
learning_rate_discriminator = 1e-6
learning_rate_generator = 1e-6
epsilon = 1e-8
num_train_epochs = 5
multi_gpu = True
# Scheduler
apply_scheduler = False
warmup_proportion = 0.1
# Print
print_each_n_step = 10
label_percentage = 15

#-------------------------------
#  Adopted Tranformer model
#-------------------------------
#model_name = "bert-base-cased"
#model_name = "bert-base-uncased"
#model_name = "roberta-base"
model_name = "xlm-roberta-base"
```

```
78
79 #--------------------------------
80 #  Retrieve the TREC QC Dataset
81 #--------------------------------
82 label_list = ["UNK","OR","CG"]
83 transformer = AutoModel.from_pretrained(model_name)
84 tokenizer = AutoTokenizer.from_pretrained(model_name)
85 import pandas as pd
86 df = pd.read_csv("./sample_data/fake reviews dataset.csv",usecols=["
       text_","label"])
87
88 def series_index_reset(series):
89   series = series.reset_index(drop=True)
90   return series
91 from sklearn.model_selection import train_test_split
92 X_train, X_test, y_train, y_test = train_test_split(df['text_'],df['
       label'], stratify=df['label'])
93
94 # train data
95 X_train = series_index_reset(X_train)
96 y_train = series_index_reset(y_train)
97
98 #test data
99 X_test = series_index_reset(X_test)
100 y_test = series_index_reset(y_test)
101
102 def create_tuple_list(x,y):
103   lst = []
104   for i in range(0,len(x)):
105     lst.append((x[i],y[i]))
106   return lst
107
108 def create_unlabelled_tuple_list(x):
109   lst = []
110   for i in range(0,len(x)):
111     lst.append((x[i],"UNK"))
112   return lst
113
114 LABELLED_DATA_COUNT = int(len(X_train) * (label_percentage/100))
115 print(LABELLED_DATA_COUNT)
116
117 labelled_x = X_train[:LABELLED_DATA_COUNT]
118 unlabelled_x = X_train[LABELLED_DATA_COUNT:]
119 labelled_y = y_train[:LABELLED_DATA_COUNT]
120 unlabelled_y = y_train[LABELLED_DATA_COUNT:]
121
122 #labelled data
123 labelled_x = series_index_reset(labelled_x)
124 labelled_y = series_index_reset(labelled_y)
125 labelled_set = create_tuple_list(labelled_x,labelled_y)
126
127 #unlabelled data
128 unlabelled_x = series_index_reset(unlabelled_x)
129 unlabelled_y = series_index_reset(unlabelled_y)
130 unlabelled_set = create_unlabelled_tuple_list(unlabelled_x)
131
132 # test data
133 test_set = create_tuple_list(X_test,y_test)
```

```
134
135 #Load the examples
136 labeled_examples = labelled_set
137 unlabeled_examples = unlabelled_set
138 test_examples = test_set
139
140 def generate_data_loader(input_examples, label_masks, label_map,
      do_shuffle = False, balance_label_examples = False):
141   '''
142   Generate a Dataloader given the input examples, eventually masked if
        they are
143   to be considered NOT labeled.
144   '''
145   examples = []
146
147   # Count the percentage of labeled examples
148   num_labeled_examples = 0
149   for label_mask in label_masks:
150     if label_mask:
151       num_labeled_examples += 1
152   label_mask_rate = num_labeled_examples/len(input_examples)
153
154   # if required it applies the balance
155   for index, ex in enumerate(input_examples):
156     if label_mask_rate == 1 or not balance_label_examples:
157       examples.append((ex, label_masks[index]))
158     else:
159       # IT SIMULATE A LABELED EXAMPLE
160       if label_masks[index]:
161         balance = int(1/label_mask_rate)
162         balance = int(math.log(balance,2))
163         if balance < 1:
164           balance = 1
165         for b in range(0, int(balance)):
166           examples.append((ex, label_masks[index]))
167       else:
168         examples.append((ex, label_masks[index]))
169
170   #-----------------------------------------------
171   # Generate input examples to the Transformer
172   #-----------------------------------------------
173   input_ids = []
174   input_mask_array = []
175   label_mask_array = []
176   label_id_array = []
177
178   # Tokenization
179   for (text, label_mask) in examples:
180     encoded_sent = tokenizer.encode(text[0], add_special_tokens=True,
          max_length=max_seq_length, padding="max_length", truncation=
          True)
181     input_ids.append(encoded_sent)
182     label_id_array.append(label_map[text[1]])
183     label_mask_array.append(label_mask)
184
185   # Attention to token (to ignore padded input wordpieces)
186   for sent in input_ids:
187     att_mask = [int(token_id > 0) for token_id in sent]
```

```
188     input_mask_array.append(att_mask)
189   # Convertion to Tensor
190   input_ids = torch.tensor(input_ids)
191   input_mask_array = torch.tensor(input_mask_array)
192   label_id_array = torch.tensor(label_id_array, dtype=torch.long)
193   label_mask_array = torch.tensor(label_mask_array)
194
195   # Building the TensorDataset
196   dataset = TensorDataset(input_ids, input_mask_array, label_id_array,
          label_mask_array)
197
198   if do_shuffle:
199     sampler = RandomSampler
200   else:
201     sampler = SequentialSampler
202
203   # Building the DataLoader
204   return DataLoader(
205               dataset,  # The training samples.
206               sampler = sampler(dataset),
207               batch_size = batch_size) # Trains with this batch size.
208
209 def format_time(elapsed):
210     '''
211     Takes a time in seconds and returns a string hh:mm:ss
212     '''
213     # Round to the nearest second.
214     elapsed_rounded = int(round((elapsed)))
215     # Format as hh:mm:ss
216     return str(datetime.timedelta(seconds=elapsed_rounded))
217
218 label_map = {}
219 for (i, label) in enumerate(label_list):
220   label_map[label] = i
221 #------------------------------
222 #   Load the train dataset
223 #------------------------------
224 train_examples = labeled_examples
225 #The labeled (train) dataset is assigned with a mask set to True
226 train_label_masks = np.ones(len(labeled_examples), dtype=bool)
227 #If unlabel examples are available
228 if unlabeled_examples:
229   train_examples = train_examples + unlabeled_examples
230   #The unlabeled (train) dataset is assigned with a mask set to False
231   tmp_masks = np.zeros(len(unlabeled_examples), dtype=bool)
232   train_label_masks = np.concatenate([train_label_masks,tmp_masks])
233
234 train_dataloader = generate_data_loader(train_examples,
       train_label_masks, label_map, do_shuffle = True,
       balance_label_examples = apply_balance)
235
236 #------------------------------
237 #   Load the test dataset
238 #------------------------------
239 #The labeled (test) dataset is assigned with a mask set to True
240 test_label_masks = np.ones(len(test_examples), dtype=bool)
241
```

```
242 test_dataloader = generate_data_loader(test_examples, test_label_masks,
        label_map, do_shuffle = False, balance_label_examples = False)
243
244 #-----------------------------
245 #   The Generator as in
246 #   https://www.aclweb.org/anthology/2020.acl-main.191/
247 #   https://github.com/crux82/ganbert
248 #-----------------------------
249 class Generator(nn.Module):
250     def __init__(self, noise_size=100, output_size=512, hidden_sizes
          =[512], dropout_rate=0.1):
251         super(Generator, self).__init__()
252         layers = []
253         hidden_sizes = [noise_size] + hidden_sizes
254         for i in range(len(hidden_sizes)-1):
255             layers.extend([nn.Linear(hidden_sizes[i], hidden_sizes[i
                +1]), nn.LeakyReLU(0.2, inplace=True), nn.Dropout(
                dropout_rate)])
256
257         layers.append(nn.Linear(hidden_sizes[-1],output_size))
258         self.layers = nn.Sequential(*layers)
259
260     def forward(self, noise):
261         output_rep = self.layers(noise)
262         return output_rep
263
264 #-----------------------------
265 #   The Discriminator
266 #   https://www.aclweb.org/anthology/2020.acl-main.191/
267 #   https://github.com/crux82/ganbert
268 #-----------------------------
269 class Discriminator(nn.Module):
270     def __init__(self, input_size=512, hidden_sizes=[512], num_labels
          =2, dropout_rate=0.1):
271         super(Discriminator, self).__init__()
272         self.input_dropout = nn.Dropout(p=dropout_rate)
273         layers = []
274         hidden_sizes = [input_size] + hidden_sizes
275         for i in range(len(hidden_sizes)-1):
276             layers.extend([nn.Linear(hidden_sizes[i], hidden_sizes[i
                +1]), nn.LeakyReLU(0.2, inplace=True), nn.Dropout(
                dropout_rate)])
277
278         self.layers = nn.Sequential(*layers) #per il flatten
279         self.logit = nn.Linear(hidden_sizes[-1],num_labels+1) # +1 for
             the probability of this sample being fake/real.
280         self.softmax = nn.Softmax(dim=-1)
281
282     def forward(self, input_rep):
283         input_rep = self.input_dropout(input_rep)
284         last_rep = self.layers(input_rep)
285         logits = self.logit(last_rep)
286         probs = self.softmax(logits)
287         return last_rep, logits, probs
288
289 # The config file is required to get the dimension of the vector
      produced by
290 # the underlying transformer
```

```
config = AutoConfig.from_pretrained(model_name)
hidden_size = int(config.hidden_size)
# Define the number and width of hidden layers
hidden_levels_g = [hidden_size for i in range(0, num_hidden_layers_g)]
hidden_levels_d = [hidden_size for i in range(0, num_hidden_layers_d)]

#------------------------------------------------
#   Instantiate the Generator and Discriminator
#------------------------------------------------
generator = Generator(noise_size=noise_size, output_size=hidden_size,
    hidden_sizes=hidden_levels_g, dropout_rate=out_dropout_rate)
discriminator = Discriminator(input_size=hidden_size, hidden_sizes=
    hidden_levels_d, num_labels=len(label_list), dropout_rate=
    out_dropout_rate)

# Put everything in the GPU if available
if torch.cuda.is_available():
  generator.cuda()
  discriminator.cuda()
  transformer.cuda()
  if multi_gpu:
    transformer = torch.nn.DataParallel(transformer)

# print(config)

training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

#models parameters
transformer_vars = [i for i in transformer.parameters()]
d_vars = transformer_vars + [v for v in discriminator.parameters()]
g_vars = [v for v in generator.parameters()]

#optimizer
dis_optimizer = torch.optim.AdamW(d_vars, lr=
    learning_rate_discriminator)
gen_optimizer = torch.optim.AdamW(g_vars, lr=learning_rate_generator)

#scheduler
if apply_scheduler:
  num_train_examples = len(train_examples)
  num_train_steps = int(num_train_examples / batch_size *
      num_train_epochs)
  num_warmup_steps = int(num_train_steps * warmup_proportion)

  scheduler_d = get_constant_schedule_with_warmup(dis_optimizer,
                                          num_warmup_steps =
                                              num_warmup_steps)
  scheduler_g = get_constant_schedule_with_warmup(gen_optimizer,
                                          num_warmup_steps =
                                              num_warmup_steps)

# For each epoch...
for epoch_i in range(0, num_train_epochs):
    # ========================================
    #               Training
```

```
342     # ========================================
343     # Perform one full pass over the training set.
344     print("")
345     print('======== Epoch {:} / {:} ========'.format(epoch_i + 1,
            num_train_epochs))
346     print('Training...')
347
348     # Measure how long the training epoch takes.
349     t0 = time.time()
350
351     # Reset the total loss for this epoch.
352     tr_g_loss = 0
353     tr_d_loss = 0
354
355     # Put the model into training mode.
356     transformer.train()
357     generator.train()
358     discriminator.train()
359
360     # For each batch of training data...
361     for step, batch in enumerate(train_dataloader):
362
363         # Progress update every print_each_n_step batches.
364         if step % print_each_n_step == 0 and not step == 0:
365             # Calculate elapsed time in minutes.
366             elapsed = format_time(time.time() - t0)
367
368             # Report progress.
369             print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format
                    (step, len(train_dataloader), elapsed))
370
371         # Unpack this training batch from our dataloader.
372         b_input_ids = batch[0].to(device)
373         b_input_mask = batch[1].to(device)
374         b_labels = batch[2].to(device)
375         b_label_mask = batch[3].to(device)
376
377         real_batch_size = b_input_ids.shape[0]
378
379         # Encode real data in the Transformer
380         model_outputs = transformer(b_input_ids, attention_mask=
                b_input_mask)
381         hidden_states = model_outputs[-1]
382
383         # Generate fake data that should have the same distribution of
                the ones
384         # encoded by the transformer.
385         # First noisy input are used in input to the Generator
386         noise = torch.zeros(real_batch_size, noise_size, device=device)
                .uniform_(0, 1)
387         # Gnerate Fake data
388         gen_rep = generator(noise)
389
390         # Generate the output of the Discriminator for real and fake
                data.
391         # First, we put together the output of the tranformer and the
                generator
392         disciminator_input = torch.cat([hidden_states, gen_rep], dim=0)
```

```
393         # Then, we select the output of the disciminator
394         features, logits, probs = discriminator(disciminator_input)
395
396         # Finally, we separate the discriminator's output for the real
                and fake
397         # data
398         features_list = torch.split(features, real_batch_size)
399         D_real_features = features_list[0]
400         D_fake_features = features_list[1]
401
402         logits_list = torch.split(logits, real_batch_size)
403         D_real_logits = logits_list[0]
404         D_fake_logits = logits_list[1]
405
406         probs_list = torch.split(probs, real_batch_size)
407         D_real_probs = probs_list[0]
408         D_fake_probs = probs_list[1]
409
410         #---------------------------------
411         #  LOSS evaluation
412         #---------------------------------
413         # Generator's LOSS estimation
414         g_loss_d = -1 * torch.mean(torch.log(1 - D_fake_probs[:,-1] +
                epsilon))
415         g_feat_reg = torch.mean(torch.pow(torch.mean(D_real_features,
                dim=0) - torch.mean(D_fake_features, dim=0), 2))
416         g_loss = g_loss_d + g_feat_reg
417
418         # Disciminator's LOSS estimation
419         logits = D_real_logits[:,0:-1]
420         log_probs = F.log_softmax(logits, dim=-1)
421         # The discriminator provides an output for labeled and
                unlabeled real data
422         # so the loss evaluated for unlabeled data is ignored (masked)
423         label2one_hot = torch.nn.functional.one_hot(b_labels, len(
                label_list))
424         per_example_loss = -torch.sum(label2one_hot * log_probs, dim
                =-1)
425         per_example_loss = torch.masked_select(per_example_loss,
                b_label_mask.to(device))
426         labeled_example_count = per_example_loss.type(torch.float32).
                numel()
427
428         # It may be the case that a batch does not contain labeled
                examples,
429         # so the "supervised loss" in this case is not evaluated
430         if labeled_example_count == 0:
431           D_L_Supervised = 0
432         else:
433           D_L_Supervised = torch.div(torch.sum(per_example_loss.to(
                  device)), labeled_example_count)
434
435         D_L_unsupervised1U = -1 * torch.mean(torch.log(1 - D_real_probs
                [:, -1] + epsilon))
436         D_L_unsupervised2U = -1 * torch.mean(torch.log(D_fake_probs[:,
                -1] + epsilon))
437         d_loss = D_L_Supervised + D_L_unsupervised1U +
                D_L_unsupervised2U
```

```
438
439         #---------------------------------
440         #  OPTIMIZATION
441         #---------------------------------
442         # Avoid gradient accumulation
443         gen_optimizer.zero_grad()
444         dis_optimizer.zero_grad()
445
446         # Calculate weigth updates
447         # retain_graph=True is required since the underlying graph will
                be deleted after backward
448         g_loss.backward(retain_graph=True)
449         d_loss.backward()
450
451         # Apply modifications
452         gen_optimizer.step()
453         dis_optimizer.step()
454
455         # A detail log of the individual losses
456         #print("{0:.4f}\t{1:.4f}\t{2:.4f}\t{3:.4f}\t{4:.4f}".
457         #      format(D_L_Supervised, D_L_unsupervised1U,
                   D_L_unsupervised2U,
458         #             g_loss_d, g_feat_reg))
459
460         # Save the losses to print them later
461         tr_g_loss += g_loss.item()
462         tr_d_loss += d_loss.item()
463
464         # Update the learning rate with the scheduler
465         if apply_scheduler:
466           scheduler_d.step()
467           scheduler_g.step()
468
469     # Calculate the average loss over all of the batches.
470     avg_train_loss_g = tr_g_loss / len(train_dataloader)
471     avg_train_loss_d = tr_d_loss / len(train_dataloader)
472
473     # Measure how long this epoch took.
474     training_time = format_time(time.time() - t0)
475
476     print("")
477     print("  Average training loss generetor: {0:.3f}".format(
           avg_train_loss_g))
478     print("  Average training loss discriminator: {0:.3f}".format(
           avg_train_loss_d))
479     print("  Training epcoh took: {:}".format(training_time))
480
481     # ========================================
482     #     TEST ON THE EVALUATION DATASET
483     # ========================================
484     # After the completion of each training epoch, measure our
           performance on
485     # our test set.
486     print("")
487     print("Running Test...")
488
489     t0 = time.time()
490
```

```
491     # Put the model in evaluation mode--the dropout layers behave
            differently
492     # during evaluation.
493     transformer.eval() #maybe redundant
494     discriminator.eval()
495     generator.eval()
496
497     # Tracking variables
498     total_test_accuracy = 0
499
500     total_test_loss = 0
501     nb_test_steps = 0
502
503     all_preds = []
504     all_labels_ids = []
505
506     #loss
507     nll_loss = torch.nn.CrossEntropyLoss(ignore_index=-1)
508
509     # Evaluate data for one epoch
510     for batch in test_dataloader:
511
512         # Unpack this training batch from our dataloader.
513         b_input_ids = batch[0].to(device)
514         b_input_mask = batch[1].to(device)
515         b_labels = batch[2].to(device)
516
517         # Tell pytorch not to bother with constructing the compute
                graph during
518         # the forward pass, since this is only needed for backprop (
                training).
519         with torch.no_grad():
520             model_outputs = transformer(b_input_ids, attention_mask=
                    b_input_mask)
521             hidden_states = model_outputs[-1]
522             _, logits, probs = discriminator(hidden_states)
523             ###log_probs = F.log_softmax(probs[:,1:], dim=-1)
524             filtered_logits = logits[:,0:-1]
525             # Accumulate the test loss.
526             total_test_loss += nll_loss(filtered_logits, b_labels)
527
528         # Accumulate the predictions and the input labels
529         _, preds = torch.max(filtered_logits, 1)
530         all_preds += preds.detach().cpu()
531         all_labels_ids += b_labels.detach().cpu()
532
533     # Report the final accuracy for this validation run.
534     all_preds = torch.stack(all_preds).numpy()
535     all_labels_ids = torch.stack(all_labels_ids).numpy()
536     test_accuracy = np.sum(all_preds == all_labels_ids) / len(all_preds
            )
537     print("  Accuracy: {0:.3f}".format(test_accuracy))
538
539     # Calculate the average loss over all of the batches.
540     avg_test_loss = total_test_loss / len(test_dataloader)
541     avg_test_loss = avg_test_loss.item()
542
543     # Measure how long the validation run took.
```

```
    test_time = format_time(time.time() - t0)

    print("  Test Loss: {0:.3f}".format(avg_test_loss))
    print("  Test took: {:}".format(test_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss generator': avg_train_loss_g,
            'Training Loss discriminator': avg_train_loss_d,
            'Valid. Loss': avg_test_loss,
            'Valid. Accur.': test_accuracy,
            'Training Time': training_time,
            'Test Time': test_time
        }
    )

for stat in training_stats:
  print(stat)

print("\nTraining complete!")

print("Total training took {:} (h:mm:ss)".format(format_time(time.time
    ()-total_t0)))
```

E QnA classifer

[6]

```
# Install Transformers
!pip install transformers==4.20.0
!pip install torch torchvision
!pip install pandas
!pip install numpy
!pip install sentencepiece

# loading data
import pandas as pd
train_data_df = pd.read_json(path_or_buf="./sample_data/train.jsonl",
    lines=True,orient='records')
dev_data_df = pd.read_json(path_or_buf="./sample_data/dev.jsonl", lines
    =True,orient='records')
print(train_data_df.head(5))

import random
import torch
import numpy as np
import pandas as pd
from tqdm import tqdm
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,
    SequentialSampler
from transformers import AutoTokenizer, AutoModel, AdamW,
    AutoModelForSequenceClassification
```

[6] https://github.com/hishamp3/MasterThesis-Lies-DeceptiveText/blob/main/QnA_Classifier.ipynb

```
21 !pip install torch torchvision torchaudio --extra-index-url https://
       download.pytorch.org/whl/cpu
22
23 import gc
24 gc.collect()
25 torch.cuda.empty_cache()
26
27 # use GPU if available
28 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
29
30 #setting seeds
31 random.seed(26)
32 np.random.seed(26)
33 torch.manual_seed(26)
34 #model_name = 'roberta-base'
35 #model_name = 'bert-large-cased'
36 #model_name = 'roberta-base-large'
37 #tokenizer = AutoTokenizer.from_pretrained("bigscience/bloom-560m")
38 #model = AutoModelForCausalLM.from_pretrained("bigscience/bloom-560m")
39
40 from transformers import AutoTokenizer,
       XLMRobertaForSequenceClassification
41
42 #tokenizer = AutoTokenizer.from_pretrained("cardiffnlp/twitter-roberta-
       base-emotion")
43 #model = XLMRobertaForSequenceClassification.from_pretrained("
       cardiffnlp/twitter-roberta-base-emotion")
44
45 from transformers import AutoTokenizer,
       XLMRobertaXLForSequenceClassification
46
47 #tokenizer = AutoTokenizer.from_pretrained("xlm-roberta-large")
48 #model = XLMRobertaXLForSequenceClassification.from_pretrained("xlm-
       roberta-large", num_labels=2)
49
50 #tokenizer = AutoTokenizer.from_pretrained('bert-large-cased')
51 #model = AutoModelForSequenceClassification.from_pretrained('bert-large
       -cased')
52 #model = AutoModelForSequenceClassification.from_pretrained(model_name)
53 #tokenizer = AutoTokenizer.from_pretrained(model_name)
54
55 tokenizer = AutoTokenizer.from_pretrained('roberta-base')
56 model = AutoModelForSequenceClassification.from_pretrained('roberta-
       base')
57 #tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased')
58 #model = XLNetModel.from_pretrained('xlnet-base-cased')
59
60 model.to(device)
61 learning_rate = 1e-5
62 optimizer = AdamW(model.parameters(),lr=learning_rate,eps=1e-8)
63
64 def encode_data(tokenizer, questions, passages, max_length):
65     """Encode the question/passage pairs into features than can be fed
          to the model"""
66     input_ids = []
67     attention_masks = []
68
69     for question,passage in zip(questions,passages):
```

```
70          encoded_data = tokenizer.encode_plus(question, passage,
                max_length=max_length, pad_to_max_length=True,
                truncation_strategy="longest_first")
71          encoded_pair = encoded_data["input_ids"]
72          attention_mask = encoded_data["attention_mask"]
73
74          input_ids.append(encoded_pair)
75          attention_masks.append(attention_mask)
76
77      return np.array(input_ids),np.array(attention_masks)
78
79 passages_train = train_data_df.passage.values
80 questions_train = train_data_df.question.values
81 answers_train = train_data_df.answer.values.astype(int)
82 passages_dev = dev_data_df.passage.values
83 questions_dev = dev_data_df.question.values
84 answers_dev = dev_data_df.answer.values.astype(int)
85
86 max_seq_length = 512
87 input_ids_train,attention_masks_train = encode_data(tokenizer,
       questions_train, passages_train, max_seq_length)
88 input_ids_dev,attention_masks_dev = encode_data(tokenizer,
       questions_dev, passages_dev, max_seq_length)
89
90 train_features = (input_ids_train,attention_masks_train,answers_train)
91 dev_features = (input_ids_dev,attention_masks_dev,answers_dev)
92
93 # building dataloaders
94 batch_size = 16
95 train_features_tensors = [torch.tensor(feature, dtype=torch.long) for
       feature in train_features]
96 dev_features_tensors = [torch.tensor(feature, dtype=torch.long) for
       feature in dev_features]
97
98 train_dataset = TensorDataset(*train_features_tensors)
99 dev_dataset = TensorDataset(*dev_features_tensors)
100
101 train_sampler = RandomSampler(train_dataset)
102 dev_sampler = SequentialSampler(dev_dataset)
103
104 train_dataloader = DataLoader(train_dataset, sampler=train_sampler,
       batch_size=batch_size)
105 dev_dataloader = DataLoader(dev_dataset, sampler=dev_sampler,
       batch_size=batch_size)
106
107 torch.cuda.empty_cache()
108 def cache_clear():
109     gc.collect()
110     torch.cuda.empty_cache()
111 from sklearn.metrics import confusion_matrix
112 from sklearn.metrics import f1_score
113
114 epochs = 3
115 grad_acc_steps = 4
116 train_loss_values = []
117 dev_acc_values = []
118
119 for _ in tqdm(range(epochs), desc="Epoch"):
```

```
120
121     #training
122     epoch_train_loss = 0
123     model.train()
124     model.zero_grad()
125
126     for step,batch in enumerate(train_dataloader):
127         inputs_ids = batch[0].to(device)
128         attention_masks = batch[1].to(device)
129         labels = batch[2].to(device)
130
131         # cache_clear()
132         outputs = model(inputs_ids, token_type_ids=None, attention_mask
                =attention_masks, labels=labels)
133         #outputs = model(inputs_ids, attention_mask=attention_masks)
134
135         loss = outputs[0]
136         loss = loss / grad_acc_steps
137         epoch_train_loss += loss.item()
138
139         loss.backward()
140
141         if(step+1) % grad_acc_steps == 0:
142             torch.nn.utils.clip_grad_norm_(model.parameters(),1.0)
143             optimizer.step()
144             model.zero_grad()
145
146     epoch_train_loss = epoch_train_loss / len(train_dataloader)
147     train_loss_values.append(epoch_train_loss)
148
149     #Evaluation
150     epoch_dev_accuracy = 0
151     model.eval()
152
153     y_pred = []
154     y_true = []
155     for batch in dev_dataloader:
156
157         input_ids = batch[0].to(device)
158         attention_masks = batch[1].to(device)
159         labels = batch[2]
160
161         with torch.no_grad():
162             outputs = model(input_ids, token_type_ids=None,
                    attention_mask = attention_masks)
163             # outputs = model(inputs_ids, attention_mask=
                    attention_masks, labels=labels)
164
165         logits = outputs[0]
166         logits = logits.detach().cpu().numpy()
167
168         predictions = np.argmax(logits,axis=1).flatten()
169         labels = labels.numpy().flatten()
170
171         y_pred.extend(predictions)
172         y_true.extend(labels)
173         epoch_dev_accuracy += np.sum(predictions==labels)/len(labels)
174
```

```
175
176     cf_matrix = confusion_matrix(y_true, y_pred)
177     score_f1 = f1_score(y_true, y_pred)
178     epoch_dev_accuracy = epoch_dev_accuracy / len(dev_dataloader)
179     dev_acc_values.append(epoch_dev_accuracy)
180
181 print(cf_matrix)
182 from sklearn.metrics import classification_report
183 target_names = ['No', 'Yes']
184 print(classification_report(y_true, y_pred,target_names=target_names))
185 print("Training Losses :",train_loss_values)
186 print("Accuracy :",dev_acc_values)
187
188 def predict(question, passage):
189   sequence = tokenizer.encode_plus(question, passage, return_tensors="
        pt")['input_ids'].to(device)
190
191   logits = model(sequence)[0]
192   probabilities = torch.softmax(logits, dim=1).detach().cpu().tolist()
        [0]
193   proba_yes = round(probabilities[1], 2)
194   proba_no = round(probabilities[0], 2)
195
196   print(f"Question: {question}, Yes: {proba_yes}, No: {proba_no}")
197
198 passage_planets = '''A planet is a large, rounded astronomical body
      that is neither a star nor its remnant. The best available theory
      of planet formation is the nebular hypothesis, which posits that an
       interstellar cloud collapses out of a nebula to create a young
      protostar orbited by a protoplanetary disk. Planets grow in this
      disk by the gradual accumulation of material driven by gravity, a
      process called accretion. The Solar System has at least eight
      planets: the terrestrial planets Mercury, Venus, Earth and Mars,
      and the giant planets Jupiter, Saturn, Uranus and Neptune. These
      planets each rotate around an axis tilted with respect to its
      orbital pole. All planets of the Solar System other than Mercury
      possess a considerable atmosphere, and some share such features as
      ice caps, seasons, volcanism, hurricanes, tectonics, and even
      hydrology. Apart from Venus and Mars, the Solar System planets
      generate magnetic fields, and all except Venus and Mercury have
      natural satellites. The giant planets bear planetary rings, the
      most prominent being those of Saturn.'''
199 planets_questions = [
200     "Mercury is a planet",
201     "Planet is a star",
202     "Mercury has a no atmosphere",
203     "There are eight planets in solar system",
204     "There are hundred planets in solar system",
205     "Venus is closer to sun compared to mercury"
206
207 ]
208
209 for s_question in planets_questions:
210   predict(s_question, passage_planets)
```

Declaration of Authorship

I hereby declare that, to the best of my knowledge and belief, this Master Thesis titled " Lies & Deceptive Text Detection" is my own work. I confirm that each significant contribution to and quotation in this thesis that originates from the work or works of others is indicated by proper use of citation and references.

Koblenz, 21st October 2023

Hisham Parveez

Consent Form

for the use of plagiarism detection software to check my thesis

Last name: Parveez **First name**: Hisham
Student number: 221202741 **Course of study**: Web and Data Science
Address: Emser str, 139, 56076 Koblenz
Title of the thesis: " Lies & Deceptive Text Detection"

What is plagiarism? Plagiarism is defined as submitting someone else's work or ideas as your own without a complete indication of the source. It is hereby irrelevant whether the work of others is copied word by word without acknowledgment of the source, text structures (e.g. line of argumentation or outline) are borrowed or texts are translated from a foreign language.

Use of plagiarism detection software The examination office uses plagiarism software to check each submitted bachelor and master thesis for plagiarism. For that purpose the thesis is electronically forwarded to a software service provider where the software checks for potential matches between the submitted work and work from other sources. For future comparisons with other theses, your thesis will be permanently stored in a database. The student agrees that his or her thesis may be stored and reproduced only for the purpose of plagiarism assessment. The examiners of the thesis will be advised on the outcome of the plagiarism assessment.

Sanctions Each case of plagiarism constitutes an attempt to deceive in terms of the examination regulations and will lead to the thesis being graded as "failed". This will be communicated to the examination office where your case will be documented. In the event of a serious case of deception the examinee can be generally excluded from any further examination. This can lead to the exmatriculation of the student. Even after completion of the examination procedure and graduation from university, plagiarism can result in a withdrawal of the awarded academic degree.

I confirm that I have read and understood the information in this document. I agree to the outlined procedure for plagiarism assessment and potential sanctioning.

Koblenz, 21st October 2023

Hisham Parveez