# Sentiment Analysis On Code-Mixed Tweets

---

A Thesis
Presented to
The Division of Mathematical and Natural Sciences
Reed College

---

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

---

Vinay Gopalan

May 2020

Approved for the Division
(Computer Science)

_____

Mark Andrew Hopkins

# Acknowledgements

I would like to thank a few people, without whom this thesis would not be possible and my time at Reed would be incomplete.

To my family, thank you for helping me be who I am today. To **Amma** and **Appa**, thank you for always supporting me in whatever I choose to do and for being there for me during my journey in a foreign country. To **Shruta**, thank you for teaching me to always be creative and to not be afraid to try new things. To **Nani**, thank you for teaching me to always have faith that everything will be okay. To **Malini atthai**, thank you for showing me how to be pragmatic in emotional situations. To **Praveen anna** and **Ranjini aunty**, thank you for helping me navigate through these waters and for giving me a second home away from home.

To my friends, thank you for always making me a better person. To **Hannah**, thank you for showing me what it means to be a true friend. With you around, I always know I have someone to talk to. To **Aiman**, **Alex** and **Edwin**, thank you for always making me smile and teaching me not to take life too seriously. To **Rishabh** and **Abhigyaan**, thank you for showing me that our friendship will never change no matter where we all are.

To my professors at Reed College, thank you for everything you taught me during my time as a student and to **Mark**, thank you for being a great advisor and for giving me the proper guidance while writing this thesis.

# List of Abbreviations

Below is a list of abbreviations used in this thesis:

| | |
|---|---|
| **NLP** | Natural Language Processing |
| **AI** | Artificial Intelligence |
| **NN** | Neural Network |
| **RNN** | Recurrent Neural Network |
| **RNTN** | Recursive Neural Tensor Network |
| **BERT** | Bidirectional Encoder Representations from Transformers |
| **MLM** | Masked Language Model |
| **GLUE** | General Language Understanding Evaluation |
| **BoW** | Bag-of-Words |

# Table of Contents

# List of Tables

# List of Figures

# Abstract

In this thesis, our aim is to explore the task of sentiment analysis on code-mixed Hinglish (Hindi-English) tweets as participants of Task 9 of the SemEval 2020 competition, formally known as the SentiMix task. Though there are several models that have been used to perform sentiment analysis over the years, most of these NLP systems are built for the English language. With a lot of the data on the Internet coming from multilingual and non-native English speakers, it is important to understand and research the task of sentiment analysis performed on code-mixed text. We began by fine-tuning pre-trained models like $BERT_{BASE}$ and $BERT_{LARGE}$ to our target task. Our results suggested that the pre-trained weights were being computed from scratch during training on the target task, and to test this we attempted to recreate the BERT fine-tuning results only using feedforward neural networks that were trained on a Bag-of-Words (BoW) and Bag-of-Ngrams representation. We found that the the results of fine-tuning $BERT_{LARGE}$ (24 layers) could be approximated by a 2-layer BoW feedforward NN in the case of Hinglish text, as the fine-tuning model had a highest accuracy of 63.9% and the BoW model had a highest accuracy of 60.01% during the training phase. In the evaluation phase of the competition, our fine-tuning model had an accuracy of 69.9% without bagging and 71.4% with bagging. Once the official system rankings were published on April 6, 2020[1], our best submission placed 4th out of 62 entries. All the code needed to recreate our results can be found here[2].

[1] see https://competitions.codalab.org/competitions/20654#learn_the_details-results

[2] see https://github.com/gopalanvinay/thesis-vinay-gopalan

# Introduction

The Internet today has a vast collection of data in various forms, including text and images. A big part of this data comes from various social media platforms, which enable users to share thoughts about their daily experiences over the Internet. The millions of tweets, updates, location check-ins, likes/dislikes that users share everyday on different platforms form a large bank of opinionated data that contains information such as political leanings and product preferences. Extracting the sentiment and opinions from this data, though immensely useful, is also challenging, which is why there is a lot of active research in this field known as *sentiment analysis*.

Although several models have been proposed to perform this task over the years, such as those built on top of the Recursive Neural Tensor Network (Socher et al. (2013)) or the more recent BERT model (Devlin et al. (2018)), most of these language technologies are built for the English language. With a lot of the data on the Internet coming from multilingual and non-native English speakers who combine English and other languages when they use social media, it is important to study sentiment analysis for so-called 'code-mixed' social media text.

## 0.1 Code-Mixing

Code-mixing, the phenomenon in which humans interleave multiple languages, is extremely common in multilingual societies. Formally defined, "code-mixing refers to all cases where lexical items and grammatical features from two languages appear in one sentence" (Muysken et al. (2000)). A similar term used in this context is code-switching and sometimes these terms are used interchangeably, but one of the distinctions between them is that term code-switching is done for a particular purpose or in a specific situation whereas code-mixing is done more out of linguistic requirement. For example, native Hindi speakers use English words like *cricket* and *computer* in conversation because hardly anyone

knows the Hindi words for those terms. In this case, a Hindi sentence like "*Aaj mai cricket khelunga*" (Today I'll play cricket) mixes in an English word out of necessity rather than for effect. On the other hand, when a student talking in Hindi with their friends switches to English as soon as their teacher arrives, they code-switch in order to adapt to a more formal situation and not out of necessity.

Code-mixing is different from other language mixing phenomena such as language borrowing because in code-mixing, the speaker is fluent in both the languages being mixed. There are different types of code-mixing, such as intersentential (the language switching occurs between sentences or clauses), intra-sentential (the language switching occurs within a sentence or a clause) and intra-word (the language switching occurs within a word itself), but most of the research and studies focus on the hybrid grammar structures in intra-sentential code-mixing.

Although fairly common as a practice, code-mixing used to be considered an inferior use of language by scholars in the mid-twentieth century (Uriel (1953)). Today, however, code-mixing is a very active area of research and has become more common with the increase in the number of multilingual and non-native English speakers throughout the world. The effect of code-mixing is seen not only in different formal and informal public settings, but also on the content being shared on social media. In a study conducted by the Statista Research Department in 2013 (Statista (2013)) it was seen that more than half the content on Twitter is in a language other than English.

In this thesis, we specifically focus on performing sentiment analysis on a corpus of tweets in Hinglish, a combination of Hindi and English. Hindi is the 4th most spoken language in the world (Ethnologue (2019)), and relies on a phonetic alphabet in the Devanagari script. Since it is phonetic, it becomes easier for several multilingual or non-native English speakers to code-mix by using English-based phonetic typing and inserting anglicisms into their vernacular. Due to intra-sentential and intra-word level switching, this linguistic phenomenon poses a great challenge to conventional NLP systems, which currently rely on monolingual resources to handle the combination of multiple languages.

Examples of Hinglish sentences in English-based phonetic typing are the following:

"We could have met kuch mahine pehle."

Translation: We could have met a few months ago.

"Mere liye **educated** hona kaafi **important** hai."
Translation: Being **educated** is very **important** for me.

Hinglish poses several unseen difficulties to current language technologies. Since most conventional NLP systems rely on monolingual resources even for code-mixed text, the systems aren't adept to handle things like intra-word level identification or English-based phonetic typing of a different language like Hindi. Hinglish tweets also bring up other unseen challenges for current language technologies such as word-level language identification, part-of-speech tagging, dependency parsing, machine translation and semantic processing. All of these challenges form the motivation for studying Hinglish tweets instead of English.

## 0.2 Sentiment Analysis

According to (Liu (2012)), "sentiment analysis, also called opinion mining, is the field of study that analyzes people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities such as products, services, organizations, individuals, issues, events, topics, and their attributes."

Sentiment analysis requires using tools of linguistics and natural language processing (NLP) in order to extract information regarding the emotions implicit in text. Even though NLP and linguistics have had a long history, research on the topic of opinion mining only began in full force around the year 2000, with early papers like (Das & Chen (2001)) and (Pang et al. (2002)) being the first to describe research on sentiments and opinions. With the arrival of the Internet, the platform of the World Wide Web not only gave people the ability to share their opinions, thoughts and ideas, but also provided researchers with a huge volume of opinionated data. Papers on the topic such as *Opinion mining and sentiment analysis* (Pang et al. (2008)) mention how the arrival of the Internet furthered their motivation regarding the topic: "With the growing availability and popularity of opinion-rich resources such as online review sites and personal blogs, new opportunities and challenges arise as people now can, and do, actively use information technologies to seek out and understand the opinions of others."

This field of study has various applications in academia and in the industry, which explains the stark increase in its research activity in recent years.

### 0.2.1 Sentiment Analysis Applications

Our opinions and emotions play an important factor in the day-to-day decisions we make. When it comes to deciding which book to purchase or which political candidate to vote for, our sentiments often decide what we do. Understanding and extracting information from the opinions on the Web has therefore become an important task today. Almost all organizations providing services to people are interested in knowing the opinions of its consumers. Consumers themselves are also interested in product reviews in order to make informed decisions.

Before the advent of the Internet and social media, when businesses needed to learn about public opinions, they needed to conduct focus groups or surveys. With the rapid increase in public opinion on the Internet due to microblogging sites like Twitter and Reddit, however, organizations and businesses often can get away without the expense of surveys due to the abundance of opinionated data already available. However, even though this data is available, gathering it is impractical for a human reader due to the large volume of data to filter through. This task is much better suited for an automated sentiment analysis system.

Due to demands like these, sentiment analysis applications have found a place in many different areas like marketing, healthcare and even politics.

### 0.2.2 Sentiment Analysis Research

Several research papers have been published since early 2000s on the topic of sentiment analysis. A broad overview of the existing work was presented in (Pang et al. (2008)), which surveyed the foundations for opinion-oriented information retrieval and challenges presented by sentiment aware applications. These foundations paved the way for many real-life applications of sentiment analysis, such as models for predicting sales performance (Liu et al. (2007)).

The use of web-blogs as a source for sentiment analysis was presented in (Yang et al. (2007)), which demonstrated the abilities of using microblogging to extract sentiments. Once a baseline for automatically collecting a corpus for sentiment analysis and opinion mining purposes from Twitter was set by (Pak & Paroubek (2010)), tweets became a solid resource for automated systems. The usefulness of Twitter as a resource has been recognized in many papers since then. In a paper by (Tumasjan et al. (2010)), sentiments from Twitter were used to predict election results. In 2010, several papers like (Asur & Huberman (2010)) and (Joshi et al. (2010)) described models that used Twitter data or movie reviews to predict box-office revenues for films, and in (Bollen et al. (2011)), Twitter sentiment was analyzed to predict the stock market.

## 0.3 Neural Networks, Transformers and BERT

### 0.3.1 Neural Networks

The question whether computers will ever be as intelligent as humans or if computers can model the human brain and develop a sort of 'artificial intelligence' has been speculated about for over a century now. In fact, even before the first computer was built when programmable computers were first conceived, people wondered if a machine like that might become intelligent (Menabrea & Lovelace (1842)).

The years of research on the topic have led us to today, where neural networks and fields like machine learning, deep learning and artificial intelligence are ubiquitous, determining whether to recommend cesarean delivery (Mor-Yosef et al. (1990)) or detecting political ideology from text (Werbos (1994)). From health to finance, from sports to politics, neural networks and other AIs are used to solve a host of problems. The particular success of neural networks is largely due to the intense decades of research which can be roughly categorized into three historical waves of neural nets research.

The first wave started in the 1940s and continued through the 1960s, with cybernetics and the development of theories of biological learning, e.g. (McCulloch & Pitts (1943)) and (Hebb (1949)). It also included one of the most groundbreaking papers in the field and the implementation of one of the first models, the *perceptron* (Rosenblatt (1958)) which allowed the training of a single so-called "neuron.".

While the 1970s witnessed a funding freeze due to widespread skepticism about the viability of neural networks (Minsky & Papert (1969)), a second wave occurred between 1980 and 1995, starting with the innovation of back-propagation (Rumelhart et al. (1986)) to train a neural network with multiple hidden layers. The third wave, which is the current deep learning wave, started recently with the unprecedented success of a neural network from the University of Toronto (Krizhevsky et al. (2012)) in the 2012 ImageNet Competition, and continued with other papers like (Nair & Hinton (2010)) and (Bengio et al. (2007)).

Neural network research has become a sprawling field with numerous subfields. In the context of this thesis, there are a few types of neural networks that are particularly important to understand.

**Feedforward Neural Networks**

Feedforward Neural Networks are also often called multi-layer perceptrons (MLPs). They are the simplest example of deep learning models. According to (Bengio et al. (2017)): "The goal of a feedforward network is to approximate some function $f^*$ . For example, for a classifier, $y = f^*(x)$ maps an input $x$ to a category $y$. A feedforward network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters $\theta$ that result in the best function approximation."

The word 'neural' comes from the fact that they are loosely inspired by neuroscience. They are called feedforward because information flows one-way only through the function $f$ to produce output $y$ from input $x$. There are no *feedback connections* in feedforward models, where outputs of the model are fed back as input.

Feedforward networks take in input vectors as parameters and introduce the concept of a hidden layer, which requires us to choose '**activation functions**' that will be used to compute the hidden layer values. The feedforward network uses '**weights**' to discover features about the input data and extract information from it in order to perform different tasks. The **back-propagation** algorithm and its modern generalizations are used to efficiently compute necessary gradients. A depiction of a simple feedforward network is provided in Figure 1.

An example of a very popular activation function is the **Rectified Linear Unit (ReLU)** function. This function can be defined as $f(x) = x^+ = max(0, x)$, i.e., the positive part of its argument. For example, $ReLU(-20, 1, -3, -6, 5) = 0, 1, 0, 0, 5$. Another very important activation function for classifiers is the **Softmax** function, also known as the normalized exponential function, which takes as input a vector of $K$ real numbers, and normalizes it into a probability distribution consisting of $K$ probabilities proportional to the exponentials of the input numbers. Formally,

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for \; i \; = \; 1, \cdots, K \; and \; z = (z_1, \cdots, z_K) \in \mathbb{R}^K \tag{1}$$

The nature of this equation makes it extremely useful for building classifiers, because after applying softmax to a vector, each component will be in the interval (0,1) and the components will add up to 1 so that they can be interpreted as probabilities. For example, if a sentiment classifier for negative, neutral, positive sentences yields a result 0.01, 0.02, 0.97, it would imply that the sentence is positive with a 97% probability according to the classifier. Hence, softmax functions are a popular choice for the final layers in a NN.

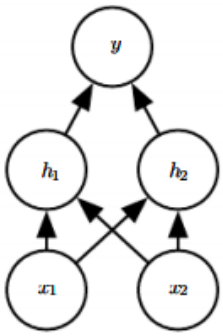


Figure 1: An example of a feedforward network (Bengio et al. (2017)). The network takes input vectors $x_1$ and $x_2$, uses weights **W** to compute hidden layer components $h_1$ and $h_2$. The hidden layers are further used in a linear function with their own weight parameters in order to calculate output $y$.

### Recurrent Neural Networks (RNNs)

When feedforward neural networks are extended to include **feedback connections**, where outputs of the model are fed back into itself, they are called **recurrent neural networks**. RNNs are a family of neural networks that are designed to process sequential data, i.e., "a

recurrent neural network is a neural network that is specialized for processing a sequence of values $x^{(1)}, \ldots, x^{(\tau)}$" (Bengio et al. (2017)) . RNNs use the concept of sharing parameters. Parameter sharing makes it possible to extend and apply the model to examples of different forms and generalize across them.

The ability to process sequential data makes RNNs appropriate to perform various NLP tasks like sentiment analysis (Chen et al. (2017)) and question answering (QA) problems (Kumar et al. (2016)), which need to classify sentencess of arbitrary length. A depiction of an RNN with an explanation of its various parameters is provided in Figure 2.

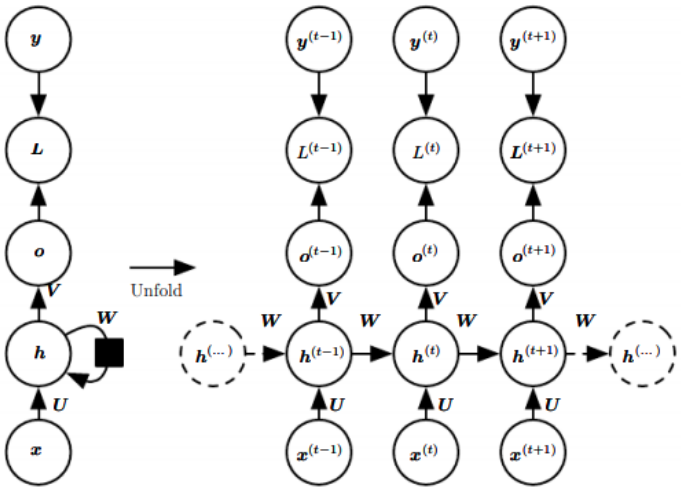


Figure 2: The computational graph to compute the training loss of a recurrent network that maps an input sequence of $x$ values to a corresponding sequence of output $o$ values. A loss $L$ measures how far each $o$ is from the corresponding training target $y$. When using softmax outputs, we assume $o$ is the unnormalized log probabilities. The loss $L$ internally computes $\hat{y} = \mathit{softmax}\,(o)$ and compares this to the target $y$. The RNN has input to hidden connections parametrized by a weight matrix **U**, hidden-to-hidden recurrent connections parametrized by a weight matrix **W**, and hidden-to-output connections parametrized by a weight matrix V (Bengio et al. (2017)).

### Attention Mechanisms

RNNs have been used successfully for many tasks involving sequential data such as machine translation, image captioning and even sentiment analysis. However, due to the sequential nature of the RNN, even the more advanced models have had limitations while working with long data sequences. In most cases, RNNs had to find connections between long input and output sentences composed of dozens of words, and it seemed that the ex-

isting RNN architectures needed to evolve to overcome these issues.

In 2014, (Bahdanau et al. (2014)) introduced Attention for RNNs. Attention was a mechanism that allowed the RNN to focus on certain parts of the input sequence when predicting a certain part of the output sequence, which enabled easier learning of higher quality. Combination of attention mechanisms enabled improved performance in many tasks making it an integral part of modern RNN networks. To demonstrate the usefulness of attention mechanisms, consider the encoder-decoder RNN architecture shown in Figure 3.

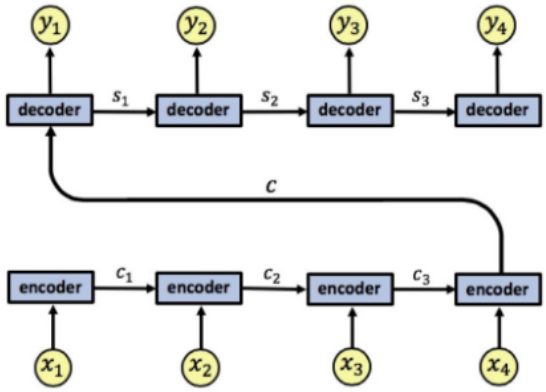


Figure 3: An RNN encoder-decoder architecture (Arbel (2019)).

As seen from the figure, the RNN encoder has an input sequence $x_1, x_2, x_3, x_4$. The encoder states are denoted by $c_1, c_2, c_3$. The encoder outputs a single output vector **c** with a fixed-length, which is passed as input to the decoder. Like the encoder, the decoder is also a single-layered RNN, where the decoder states are denoted by $s_1, s_2, s_3$ and the network's output by $y_1, y_2, y_3, y_4$. A potential issue with this encoder-decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. The encoder needs to compact the entire input sequence $x_1, x_2, x_3, x_4$ as a single vector **c**, while the decoder needs to decipher the passed information from this single vector without any context, which is a feat itself. Hence, this architecture may make it difficult for the neural network to cope with long sentences, especially if those sentences are longer than the ones in its training dataset.

In contrast, let us now consider the encoder-decoder RNN representation which includes an attention mechanism, as shown in Figure 4.

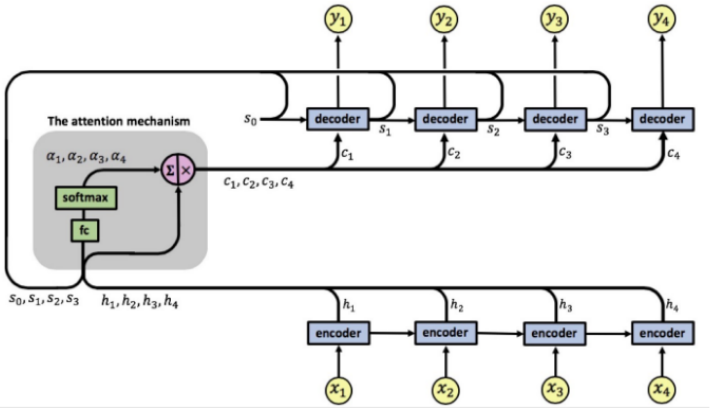


Figure 4: An RNN encoder-decoder architecture which includes an attention mechanism (Arbel (2019)).

The attention mechanism is located between the encoder and the decoder. Its input is composed of the encoder's output vectors $h_1$, $h_2$, $h_3$, $h_4$ and the states of the decoder $s_0$, $s_1$, $s_2$, $s_3$. The output of the attention model is a sequence of vectors called *context vectors* denoted by $c_1, c_2, c_3, c_4$. These context vectors enable the decoder to focus on certain parts of the input when predicting its output, hence providing context to the part being decoded. Each context vector is a weighted sum of the encoder's output vectors, i.e., the vectors $h_1, h_2, h_3, h_4$ are scaled by weights $\alpha_{ij}$ capturing the degree of relevance of input $x_j$ to output $y_i$. More formally,

$$c_i = \sum_{j=1}^{4} \alpha_{ij} \cdot h_j \tag{2}$$

From the equation we can see how each context vector includes all of the outputs of the encoder (as it is a linear combination of the outputs) as well as the degree of relevance of input terms in relation to output terms due to the scaled weights. A large $\alpha_{ij}$ attention weight would hence cause the RNN to focus on input $x_j$ (represented by the encoder's output $h_j$), when predicting the output $y_i$.

In their paper, (Bahdanau et al. (2014)) state "The most important distinguishing feature of this approach from the basic encoder-decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector." Hence, the atten-

tion mechanism allows the decoder to decide which parts of the input sequence to focus on. By implementing this attention mechanism, we relieve the encoder from having to encode all information in the input sequence into a single vector of fixed length. Instead, the information can be spread throughout the sequence $h_1$, $h_2$, $h_3$, $h_4$ which can be selectively retrieved by the decoder. This makes attention mechanisms extremely popular and useful for NLP tasks that contain long input and output sequences.

### 0.3.2 Transformers

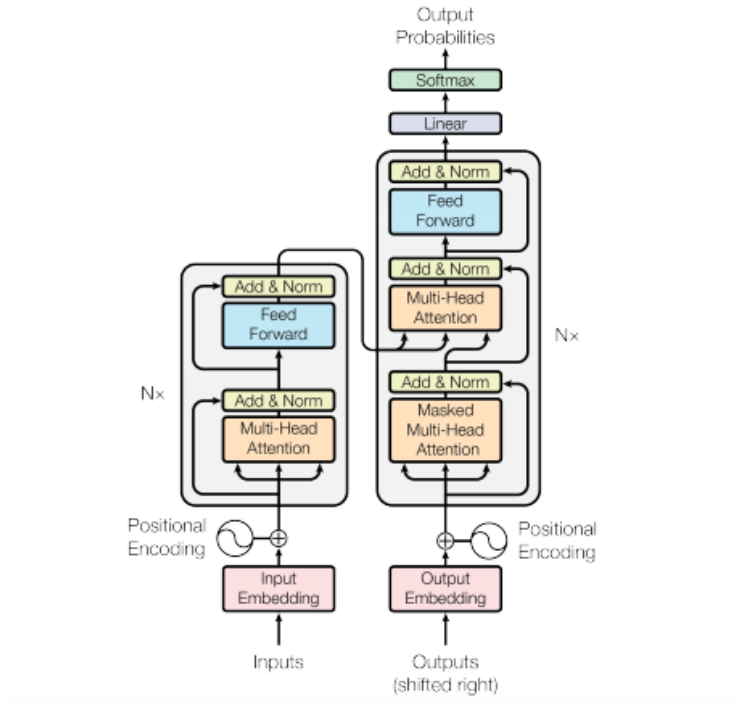


Figure 5: The Transformer - model architecture (Vaswani et al. (2017)).

RNNs have been well established as cutting edge mechanisms for sequence modeling and various NLP tasks. However, the inherently sequential nature of RNNs does not allow for parallelization with training examples, which is not very efficient at higher sequencing lengths. In order to improve on this inefficiency, (Vaswani et al. (2017)) introduced the Transformer in 2017: "We propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality."

An attention function, according to (Vaswani et al. (2017)), can be described as "mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors." Attention mechanisms have been found to be effective for sequence modeling, as seen in papers like (Kim et al. (2017)) and (Bahdanau et al. (2014)). In most of these cases, however, an attention mechanism is used with an RNN.

The particular 'attention mechanism' on which the Transformer relies is the *Scaled Dot-Product Attention*, where the weights of the values are obtained by computing the dot products of the query with all keys, dividing each by the square root of the dimension, $\sqrt{d_k}$, and then finally applying a softmax function, i.e.,

$$\textit{Attention}\,(Q, K, V) = \textit{softmax}\left(\frac{Q \cdot K}{\sqrt{d_k}}\right) V \tag{3}$$

The Scaled Dot-Product Attention algorithm is identical to general dot-product (multiplicative) attention, but also includes a scaling factor of $1/\sqrt{d_k}$. Instead of only applying a single attention function however, (Vaswani et al. (2017)) took it a step further by linearly projecting $Q$, $K$ and $V$ a total of $h$ times, and then performing multiple attention functions in parallel. The outputs of these functions are concatenated and projected as depicted in Figure 5. This extension that performs multiple attention functions is called multi-head attention.

$$\begin{aligned} &\textit{MultiHead}\,(Q, K, V) = \textit{Concat}\,(\textit{head}_1, \ldots, \text{head}_h)W^0 \\ &\text{where}\quad \textit{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{4}$$

The Transformer has two major components — a chained set of encoders and a chained set of decoders — and follows an overall architecture using stacked self-attention and fully connected layers for both the encoder and decoder (left and right halves of Figure 5 respectively). Introducing an attention mechanism in this way allows every position in the encoder and decoder to attend over all positions in the input sequence. Due to this, Transformers, unlike RNNs, do not need to process sequences in order. Hence, in the case of natural language, Transformers do not need to process each sentence linearly, which en-

ables much more parallelization than RNNs during training.

The set of encoders is composed of 6 identical layers, where each layer has two sub-layers: a multi-head self-attention mechanism, and a fully connected feed-forward NN. In self-attention, all of the keys, values and queries come from the same place, which in this case is the output of the previous layer in the encoder. The set of decoders has a similar architecture and also has 6 identical layers, but also includes a third sub-layer that performs a multi-head attention mechanism on the output of the encoder. Self-attention allows the encoder and decoder to attend to all positions in the previous layer.

The architecture described in (Vaswani et al. (2017)) outperformed RNN-based models in various tasks, such as English constituency parsing or machine translation, which makes the Transformer a currently popular model for tackling challenging NLP tasks.

### 0.3.3 Transfer Learning and BERT

In NLP, transfer learning trains a model on one task and then adapts that model to perform related tasks, for which we have less data.

A recently popular example of transfer learning in NLP has been the 'fine-tuning' of pre-trained language models. Fine-tuning models allows people to build robust applications without having to curate extensive training datasets for their specific task of interest, which has made this technique popular in both industry and academic applications.

BERT, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, is a language representation model for tasks in NLP. BERT, like many other models such as the ones described in (Dai & Le (2015)) and (Peters et al. (2018)), relies on the concept of transfer learning.

Pre-trained representations are generally applied to *downstream tasks* (downstream tasks is what the NLP field calls those supervised-learning tasks that utilize a pre-trained model or component) using either a *feature-based approach* or a *fine-tuning approach*. In the feature-based approach, pre-trained vector representations of words are used as the input features for the machine learning model. An example of a model using this strategy is

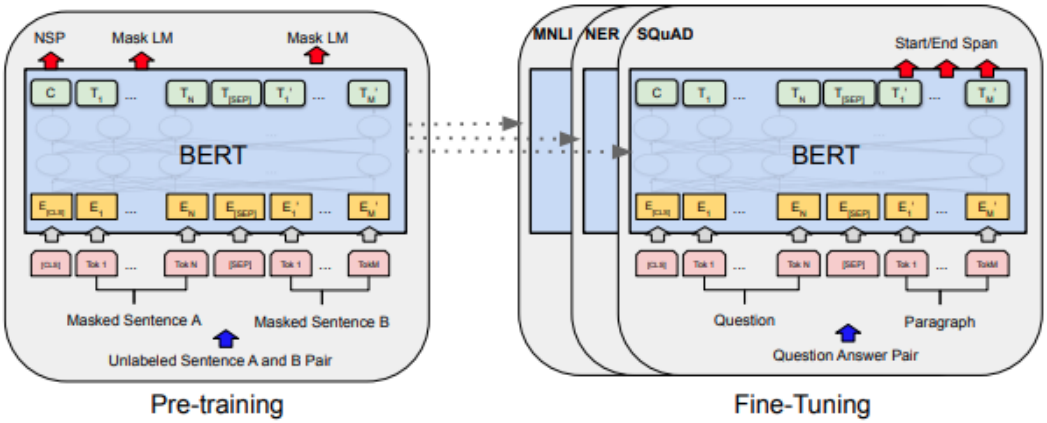


Figure 6: Pre-training and fine-tuning procedures for BERT (Devlin et al. (2018)). The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned.

ELMo described in (Peters et al. (2018)). The fine-tuning approach, on the other hand, retrains the entire pre-trained model on the target task, using the pre-trained parameters as a starting point for optimization. OpenAI GPT (Radford et al. (2018)) is an example of a model that uses this approach.

Though pre-training helped improve the efficiency of standard language models under many NLP tasks, the techniques in existence preceding BERT were limited in their implementation, especially fine-tuning representation models, because standard language models were unidirectional (such as a left-to-right architecture in OpenAI GPT). In 2018 (Devlin et al. (2018)) introduced BERT, which removed the limitation of unidirectionality by implementing a 'masked language model' (MLM) pre-training objective. According to the paper,"the masked language model randomly masks some of the tokens from the input, and the objective is to predict the original vocabulary id of the masked word based only on its context." The MLM is also referred to as a Cloze task (Taylor (1953)). The final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, and in all the experiments 15% of all tokens in each sequence are masked at random. Instead of reconstructing the entire output, only the masked words are predicted.

Although this objective gives us a pre-trained bidirectional Transformer, the downside is that the MLM creates a mismatch between pre-training and fine-tuning because the fine-tuning does not contain the [MASK] token. In order to fix this, not all the masked words

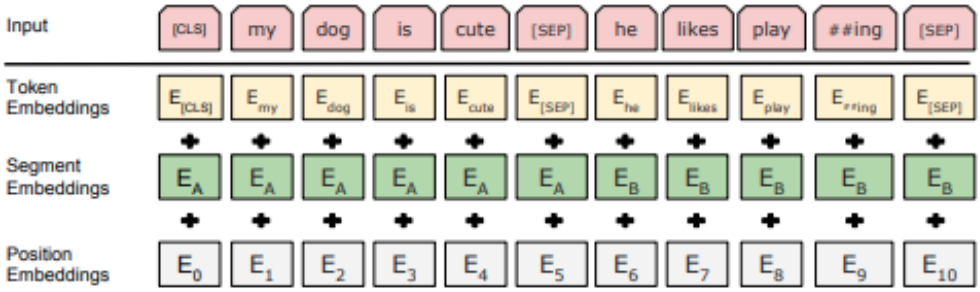


Figure 7: Input embeddings for BERT are the sum of the token embeddings, the segmentation embeddings and the position embeddings (Devlin et al. (2018)).

are replaced with the actual [MASK] token. The training data generator chooses 15% of the token positions at random for prediction, and if the $i^{th}$ token is chosen, it is replaced with either the [MASK] token 80% of the time, a random token 10% of the time or the unchanged $i^{th}$ token 10% of the time. Then, the final hidden vector is used to predict the original token with cross entropy loss.

Google has released several pre-trained BERT models, including an English model pre-trained on English Wikipedia (2500M words) and BooksCorpus (800M words) which is primarily based on two model sizes: $BERT_{BASE}$ (L=12, H=768, A=12, Total Parameters=110M) and $BERT_{LARGE}$ (L=24, H=1024, A=16, Total Parameters=340M), where L is the number of layers, H is the hidden size and A is the number of self-attention heads.

BERT has word vector representations for its input which are created using WordPiece embeddings (Johnson et al. (2017)). Figure 7 briefly depicts how the input token representations for BERT are created by summing up the token embeddings (vector representing the token), the segmentation embeddings (if it is in sentence A or B, used when embedding sentence pairs) and the position embeddings (where it is in the sentence).

The self-attention mechanism of Transformers makes BERT ideal for many downstream tasks, such as General Language Understanding Evaluation (GLUE) and Stanford Question Answering Datasets (SQuAD v1.1/v2.0). BERT advanced the state of the art by outperforming all the pre-existing models in 11 tasks. All results and architectures are described in (Devlin et al. (2018)) and the code and pre-trained models can be found here[3]. In this

[3] see https://github.com/google-research/bert

thesis, we use BERT's architecture to implement the fine-tuning approach for sentiment analysis on code-mixed tweets.

# Chapter 1

# SemEval 2020

SemEval (**Sem**antic **Eval**uation) is an international workshop/competition intended to explore the essence of 'meaning' in language (The term *meaning* in linguistics refers to the information conveyed by a sender in order to communicate with a receiver (Sanchez (2002))). To do this, researchers organize several tasks for computational systems that analyze meaning and set metrics for evaluating those tasks as part of the workshop. The goal of the evaluations is to identify what is essential to compute in meaning. These evaluations are a great source of data for tasks in computational semantics, and they help raise the profile of important but neglected tasks.

The history of SemEval goes back to the early 1990s, when evaluations to understand the current state of word sense disambiguation algorithms were organized. Word sense disambiguation is an open problem in linguistics, and deals with identifying which sense of a word is used in a sentence (Yarowsky (1995)). For example, consider the 2 distinct senses of the word *tire* — it can be interpreted as part of a wheel, as in the sentence *I need to change a flat tire*, or as showing signs of fatigue, as in *Playing all day seems to tire him out*. Finding solutions to this complex problem is essential in improving the coherence and inference abilities of computational systems, which is why the tasks for the first three iterations of the workshop, known as Senseval, were focused primarily on evaluating word sense disambiguation systems. Senseval grew each year not only in the number of participants but also the number of languages being analyzed in the tasks.

By the fourth workshop, the evaluations had evolved to include semantic analysis tasks beyond word sense disambiguation, and the conference came to be known as SemEval instead. Today, SemEval is organized each year under the umbrella of SIGLEX, the Spe-

cial Interest Group on the Lexicon of the Association for Computational Linguistics, and it runs several tasks that focus on different subfields in semantic analysis of text, such as investigating interrelationships among the elements in a sentence (also known as semantic role labeling) and classifying the emotional intent of utterances (also known as sentiment analysis).

### 1.0.1 SentiMix Task

In this thesis, we aim to perform sentiment analysis for code-mixed social media text as participants of Task 9 of the SemEval 2020 competition, formally known as the SentiMix task. A list of the task organizers is provided in Table 1.1.

| Name | Institute | Country |
|---|---|---|
| A. Das | Wipro AI Labs | India |
| T. Chakraborty | IIIT Delhi | India |
| T. Solorio | University of Houston | USA |
| G. Aguilar | University of Houston | USA |
| S. Kar | University of Houston | USA |
| B. Gambäck | NUST | Norway |
| D. Garrette | Google NY | USA |
| P. Srinivas | IIIT Sri City | India |

Table 1.1: A list of the organizers of SentiMix, Task 9 of SemEval 2020

The task was to predict the sentiment of a given code-mixed tweet. Entrants were provided with various datasets throughout the various *phases* of the competition. These datasets were comprised of Hinglish (code-mixed Hindi-English) tweets and their corresponding sentiment labels, which are positive, negative, or neutral. The organizers of SentiMix Hinglish simultaneously organized an analogous sentiment analysis task for Spanglish (code-mixed Spanish-English).

Besides the sentiment labels, the organizers also provided the language labels at the word level. Each word is tagged as English, Hindi, or universal (e.g. symbols, mentions, hashtags). Systems were evaluated in terms of *precision*, *recall* and *F-measures*. In information retrieval and classification, *precision* is the fraction of relevant instances among the retrieved instances, while *recall* is the fraction of the total amount of relevant instances that

were actually retrieved. To understand this more clearly, take as an example a model that classifies 8 tweets out of a set of 12 tweets as positive. Of the 8 identified as positive, 5 are actually positive (true positives), while the rest are either negative or neutral (false positives). The model's precision is then 5/8 while its recall is 5/12.

A measure that combines precision and recall is the harmonic mean of precision and recall, the traditional *F-measure* or balanced F-score, defined as:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{1.1}$$

.

**Phases of the competition**

The first phase of the competition was the trial phase, and it began on July 31, 2019. This phase was meant for the initial setup of your model. During this time, the participants needed to choose/design their computational model, create an experimental workflow and benchmark its accuracy on a provided trial dataset. The participants were provided with initial trial data of 1800 tweets with sentiment labels during this period.

The next phase of the competition was the training phase, which started on September 4, 2019. This was the longest phase of the competition and as the name suggests, it was the period the participants spent to train their models. This was the most crucial stage, as it was the stepping stone to the final evaluation phase where the quality of all computational models was assessed to decide the competition results. The second set of data was already split for the participants, and it contained training data of 14K tweets accompanied with a validation dataset of 3K tweets. Both of these included corresponding sentiment labels. The training phase lasted over five months.

The final evaluation phase began on February 19, 2020. During this phase, the test data was released. This was a dataset of 3K tweets without any labels, and the participants had to submit the predicted labels computed by their trained models. The models were then assessed on precision, recall and F-measures. The evaluation period came to an end on March 8, 2020. The testset labels were posted by the organizers on March 18, 2020, and official system rankings were published on April 6.

As we will describe, our submission ultimately placed 4th out of 62 entries.

**Data Formatting and Cleanup**

All data was provided in tokenized CoNLL format, a commonly used format for NLP data, established by the Conference on Computational Natural Language Learning (CoNLL) [1]. In this format, each tweet is first given a unique ID and a sentiment (positive, negative or neutral). Each tweet is then segmented into word/character tokens and each token is given a language ID, which is either *HIN* for Hindi, *ENG* for English and *O* if the token is in neither language. To summarize, each tweet in the data looks like this:

| | | |
|---|---|---|
| meta | UID | Sentiment |
| token | | LangID |
| token | | LangID |
| token | | LangID |
| ... | ... | ... |

In the aforementioned format, the *meta* keyword prefaces a line which contains auxiliary information about the tweet, namely its Unique ID (UID) and sentiment. The tokens and language IDs are separated by tabs.

The tweets include several tokens, such as URLs and usernames, that we deemed unhelpful for sentiment analysis. To get the most information out of our data, we wanted to remove as much of this as possible. To understand how we did this, consider the example tweet provided in Table 1.2 (in CoNLL format).

Looking at the example, we can see how we might want to remove the URL and username from the tweet in order to only extract relevant information that impacts the sentiment of the overall sentence. Even though we remove URLs and usernames, we still keep emojis as they might be useful for the model in predicting the sentiment. For the purposes of our experiments, the relevant information we want to gather from this datum is the sentence and its corresponding sentiment label. We represent negative, neutral and positive as 0,

[1] see https://www.conll.org/2020

1 and 2 respectively. After this, the above CoNLL tweet is transformed into the format diplayed in Table 1.3.

Table 1.2: An example of a tweet in the ConLL format

| | | |
|---|---|---|
| meta | 508 | positive |
| @ | | O |
| Ahmad | | Hin |
| _ | | O |
| Khan | | Hin |
| To | | Eng |
| all | | Eng |
| muslims | | Eng |
| sab | | Hin |
| e | | Hin |
| qadar | | Hin |
| all | | Eng |
| mubarak | | Hin |
| ♡♡♡ | | O |
| https | | Eng |
| // | | O |
| t | | Eng |
| . | | O |
| co | | Hin |
| / | | O |
| 0NKi2uqqVE | | Hin |

In order to gather such information from the provided CoNLL format we created a custom tokenizer for transforming the CoNLL data into our desired format. The custom tokenizer also identifies both usernames and URLs and removes them from the resulting sentence. The heuristics for identifying usernames is as follows: if the tokenizer encounters an '@' symbol, we build a string only including the token following the '@' symbol (call this token A). However, if the token after token A is an '_', we continue building the string until we encounter the token immediately after the '_' (call this token B). With this methodology, we can identify all usernames by matching the regex patterns of '@TokenA' and '@TokenA_TokenB'.

To catch and remove URLs, we first observed that the URL tokens in a tweet are placed at the end of the tweet. This makes our job much easier, because if the custom tokenizer

encounters a token that starts with '*http*', we keep building a string until the tokenizer encounters a new line, which signifies the end of that tweet and hence that URL. By matching tokens to the regex pattern of 'http[s]*' (starting with http or https followed by anything), we are able to cleanly catch the URL tokens and remove them from our desired resulting sentence.

Table 1.3: Format desired for our experiments

| Sentence | Label |
|---|---|
| To all muslim sab e qadar mubarak ♡♡♡ | 2 |

Using this custom tokenizer we converted the provided CoNLL data into the format of *Sentence* *Label* and saved them in formats more commonly used by NLP models such as JSON or TSV. All the related code talked about in this section can be found at this link [2].

[2] see https://github.com/gopalanvinay/thesis-vinay-gopalan/blob/master/clean_data.py

# Chapter 2

# The BERT Fine-tuning Approach

## 2.1 Transfer Learning

One of the key characteristics of the human race as an evolved species is our inherent ability to transfer knowledge across tasks. The knowledge we acquire while performing one task can also be used to solve another related task. For example, if one knows how to drive an automatic car, one can apply many of these skills when learning to drive a manual car. Similarly, if one knows the discipline involved with learning classical guitar, they can utilize that knowledge to learn other forms of music like jazz or blues (or vice versa).

These examples are specifically chosen because in neither of them do we learn something from scratch. In fact, we leverage knowledge gained from previously performed tasks in order to solve a similar one. This seems like a fairly intuitive principle, but even though deep learning algorithms were originally designed with the idea of a human brain in mind (Rosenblatt (1958)), most of these algorithms were initially designed to work in isolation and were trained to solve specific tasks. If the task is even slightly modified, the models have to be rebuilt from scratch.

To overcome the "training in isolation" methodology, the concept of *transfer learning* came into fruition. According to (Bengio et al. (2017)), "Transfer learning refers to the situation where what has been learned in one setting (i.e., distribution P1) is exploited to improve generalization in another setting (say distribution P2)." Although a lot of advancements have been made recently in the 2010s, the history of transfer learning is believed to go back as far as 1995, when the The Conference on Neural Information Processing Systems (NeurIPS, formerly NIPS) held a workshop called Learning to Learn: Knowledge Consol-

idation and Transfer in Inductive Systems[1], which provided the initial impetus for research in this field.

An important problem that transfer learning addresses in the context of deep learning is that models that solve complex problems need large amounts of data, but in most cases large labeled datasets are hard to find. With transfer learning, we can leverage trained parameters from previous models and utilize them for related tasks for which we have smaller datasets.

In this respect, transfer learning is significantly different from traditional methods of training machine learning models since traditional learning is isolated, which means no knowledge is retained which can be transferred from one model to another. This requires each new task to have a large enough dataset with which to train a model. That is not always feasible, which is why transfer learning has become an important methodology. A visual comparison of traditional vs. transfer learning is shown in Figure 2.1. The figure was taken for this webpage[2]. More information about transfer learning can be found in the seminal paper by (Pan & Yang (2009)).

Before applying transfer learning, it is important to ask certain questions, such as **what** is common between the source and the target, i.e., what knowledge can be transferred from the source to the target in order to *improve* the performance of the target task. Another important question is **when** to use transfer learning, since it does not always help. Used in the wrong situation, we could end up with a transfer that negatively impacts our target task and makes matters worse. We hence need to be careful about when to use pre-trained parameters. Finally, we need to identify **how** to actually transfer knowledge between models, which can be done in several different ways.

Transfer learning methods can be organized into categories such as *inductive transfer learning*, where the source and target domains are the same but the tasks are different from each other (Deng et al. (2014)). These algorithms try to utilize the *inductive biases* of the source domain to help improve the target task. The *inductive biases* are the sets of assumptions related to the distribution of the training data, and these assumptions impact how and what is learned by the model on the given task and domain. Another category is *unsupervised*

[1] see https://nips.cc/Conferences/PastConferences

[2] see https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a

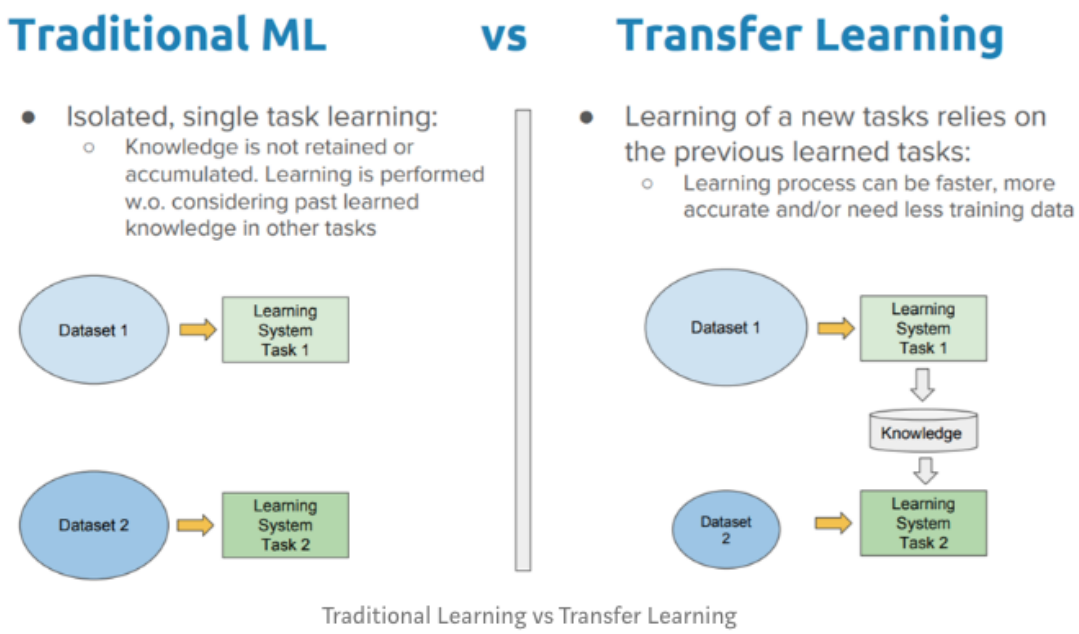


Figure 2.1: A visual representation of the traditional learning vs. transfer learning.

*transfer learning*, which is similar to inductive transfer but focuses on unsupervised tasks in the target domain (Bengio (2012)).

### 2.1.1 Transfer Learning in Deep Learning

There are various deep learning approaches that use transfer learning with state-of-the-art performance that have been developed across fields like computer vision (Szegedy et al. (2015)) and NLP (Devlin et al. (2018)). The two most popular strategies for applying transfer learning to deep learning systems are the feature-based approach and the fine-tuning approach.

In deep learning, models are layered architectures that learn different features at different layers. (as discussed in Section 0.3) These layers are then finally connected to the last layer to get the final output. We can exploit this layered architecture by using a pre-trained network such as Inception (Szegedy et al. (2015)) or BERT (Devlin et al. (2018)) without its final layer. We can use the pre-trained model as a fixed feature extractor and add our own last layer for other tasks such as classification. While doing so, we can choose to either replace the last layer while *freezing* all the pre-trained parameters (weights) as done in ELMo (Peters et al. (2018)), a seminal example of the "feature extraction" approach,

or we can have a more involved technique where we also selectively re-train some of the previous layers while replacing the last layer. In the feature-based approach we focus on the feature extractor and fix weights for all the pre-trained layers while in the fine-tuning approach, we re-train some of the previous hyperparamaters.

A visual description of the difference between the feature-based and fine-tuning approaches is provided in Figure 2.2, taken from (Li & Hoiem (2017)). Note in the figure how in the in the fine-tuning method (left) the entire network is retrained using the trained parameters as an initialization for gradient descent while in the feature-based method (right) only the final layer is removed and then re-trained.

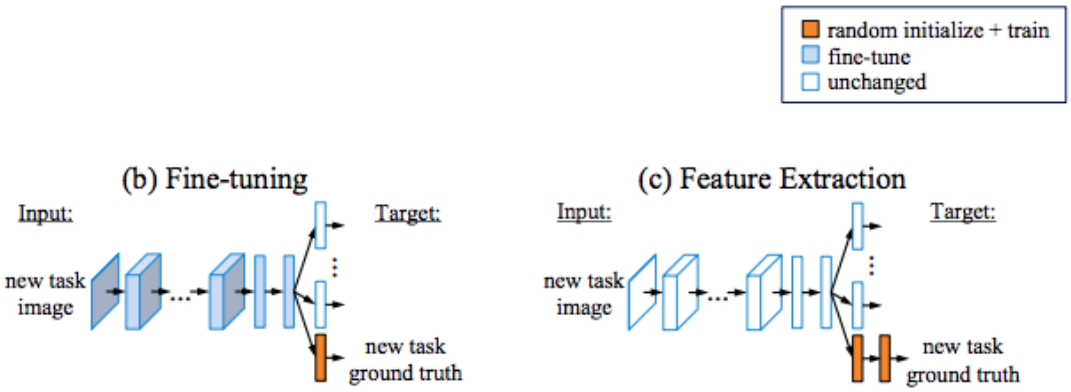


Figure 2.2: A visual representation of the feature-based approach vs. the fine-tuning approach (Li & Hoiem (2017)).

One of the first motivations for fine-tuning was in the field of computer vision, where it was seen that the initial layers of deep learning models seemed to capture generic features, while the later ones focused more on the task at hand. (Szegedy et al. (2015)) was one of the first papers to fix weights for certain layers while retraining and to fine-tune the rest of them for their specific task. Some other important papers related to the Inception model are (Szegedy et al. (2016)) and (Ioffe & Szegedy (2015)). As mentioned in Chapter 0, BERT was one of the first implementations of fine-tuning in the field of NLP. In our experiments, we chose the fine-tuning approach and used BERT to perform sentiment analysis on Hinglish code-mixed tweets.

## 2.2 Experiments and Results

For our experiments, we imported BERT from `transformers`[3]: a state-of-the-art PyTorch module for NLP with over 32 pretrained models in several languages. We based the last layer for our fine-tuning task on the *run_glue* [4] script provided as an example by the creators of the `transformers` module. The General Language Understanding Evaluation (GLUE) benchmark [5] is a collection of nine language understanding tasks for evaluating NLP systems, and the *run_glue* script uses fine-tuning on GLUE tasks for sequence classification, which is what we need in sentiment analysis: to classify a sentence as 0 (negative), 1 (neutral) or 2 (positive).

For fine-tuning, we used several of the pre-trained models provided in the `transformers` module, such as *bert-base-cased, bert-large-cased, bert-chinese, bert-base-multilingual-cased* and *roberta-base* to see which model yielded the highest accuracy. While using these models, we fine-tuned the *learning rate, weight decay, max grad norm* and *Adam epsilon* hyperparameters. Each of these parameters was selected for a reason: 1) The learning rate of a neural network is obviously crucial, as it determines the rate at which the network learns. 2) We experimented with the weight decay to see if we needed to prevent the weights from growing too large or small by multiplying with factors slightly less than 1 or slightly greater than 1 respectively. 3) Since we were not sure if our fine-tuning changes would lead to a numerical overflow in the gradients, we decided to implement gradient clipping by setting and tweaking a *max_grad_norm* parameter. 4) The Adam epsilon parameter helps prevent divide by zero errors in the cases where the gradient is almost zero, and can also help choose the right tradeoff between the weight size (as larger epsilon values lead to smaller weights) and the speed of training.

In our experiments we noticed that different models yield better results with varied tweaks in their parameters. We had several training and validation datasets available to us through the different phases of the competition, so we decided to enter later the phases with models that yielded the best results during the previous phases. In the trial phase, all models had relatively the same performance, with *bert-base-cased*, *bert-large-cased* and *roberta-base* yielding accuracies of 63.5%, 63.9% and 63.8%. To check if the models were computing weights differently based on pre-training, we also used *bert-chinese* in this phase. When

[3] see https://github.com/huggingface/transformers
[4] see https://github.com/huggingface/transformers/blob/master/examples/run_glue.py
[5] see https://gluebenchmark.com/

we observed that *bert-chinese* also had the same performance (63.5%) as *bert-base-cased*, it made us speculate that the weights are not affected due to pre-training and are instead being computed from scratch during training. This was another motivation for us to try and implement a Bag-of-Words model with a feedforward NN that could recreate these results, as described in Chapter 3.

The data released for the training phase was more uniform, which is why it might look like the accuracy of our models decreased from the trial phase, though this has more to do with the spread of the data than the models themselves. For the learning rate experiments, we started with a learning rate of $1 \times 10^{-5}$ for all the pre-trained models. We observed the best result of 63.3% was yielded by *bert-large-cased* at a learning rate of $2 \times 10^{-5}$. On the other hand, *bert-base-cased* required double the learning rate, $4 \times 10^{-5}$, to yield its highest accuracy of 62.7%. The accuracy of *roberta-base* did not improve more than 62%, which is why we decided to not move forward with that model. A summary of our obtained results can be seen in Table 2.1.

In our weight decay experiments, we started with an initial decay of 0. We tweaked this number several times, and found that the best results were yielded by *bert-large-cased* at 63.5% at weight decays of 0 and 0.5. A summary of our obtained results can be seen in Table 2.2. A similar result for *bert-large-cased* was obtained on changing the *max_grad_norm* parameter from 0.5 to 1, although there was no improvement in *bert-base-cased*'s accuracy. The *max_grad_norm* results are given in Table 2.3. Our Adam epsilon experiments began at an initial epsilon of $2 \times 10^{-8}$. We ran trials with relatively larger and smaller values of Adam epsilon to check which values gave us the best tradeoff between weight size and training speed, and observed that an Adam epsilon parameter of $1 \times 10^{-8}$ yields the highest accuracy of 63.8% for *bert-large-cased*. However, *bert-base-cased* needs a higher epsilon parameter of $2.5 \times 10^{-7}$ in order to compute an accuracy of 62.7%. Our obtained results for the Adam epsilon experiments can be seen in Table 2.4.

With our experiments during the training phase, we had a good idea about the settings that would yield the best results on each of our models. Since we were only allowed a maximum of 3 submissions during the testing phase of the competition, we decided to submit the results for the best performing model: *bert-large-cased*. We set the fine-tuned parameters to the values that yielded the best results during our training phase. We selected a *learning rate* of $2 \times 10^{-5}$, a *weight decay* of 0, a *max_grad_norm* parameter of 1 and an

*Adam epsilon* of $1 \times 10^{-8}$. We obtained the best results computed so far during any phase, and received an accuracy score of 69.9%. A summary of all parameters is given in Table 2.5

### 2.2.1 Bagging Experiments

**B**ootstrap **agg**regat**ing**, also known as bagging, is a machine learning procedure designed to improve the accuracy and performance of machine learning systems. According to (Breiman (1996)), "Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets."

In this method, a standard training set **D** of size $k$ is used to generate $m$ new training sets $\boldsymbol{D_i}$, each of size $k'$, by sampling from **D** uniformly and *with replacement*, i.e., an element from **D** may appear multiple times in the sample $\boldsymbol{D_i}$. By sampling with replacement, some observations are repeated in each $\boldsymbol{D_i}$, which forms a sample that has a fraction of unique examples from **D** while the rest are duplicates. This kind of sample is known as a bootstrap sample.

Hence, bagging allows us to create $m$ different sample sets from a single set. We can use these $m$ bootstrap samples to train $m$ different models, and combine their results by averaging the output (for regression) or by voting (for classification). As seen in (Breiman (1996)), bagging can improve accuracy if perturbing the learning set can cause significant changes in the predictor constructed. This was our motivation to implement the bagging method during the testing phase — to see if it could lead to a slight improvement of the accuracy.

In our experiments, we created 10 bootstrap samples from the given sample set. We then trained 10 instances of the *bert-large-cased* model on these bootstrap samples, and computed the results for each model on the validation set provided during the testing phase. The results of all the models were combined by *voting*, i.e., if a majority of the 10 models predicted a particular tweet as positive (negative/neutral), then the tweet's label is deemed as positive (negative/neutral). As can be seen in Table 2.6, the accuracy on the validation set was computed as 71.4%, which was the highest accuracy we obtained. After the testing

phase, the official system rankings were published on April 6, and our submission with the bagging results placed $4^{th}$ out of 62 entries.

Table 2.1: Accuracies computed by fine-tuning the learning rate parameter

| Model Name | Phase | Learning Rate | Accuracy |
|---|---|---|---|
| *bert-base-cased* | Trial Phase | $1 \times 10^{-5}$ | 63.5% |
| *bert-large-cased* | Trial Phase | $1 \times 10^{-5}$ | 63.9% |
| *roberta-base* | Trial Phase | $1 \times 10^{-5}$ | 63.8% |
| *bert-base-cased* | Training Phase | $1 \times 10^{-5}$ | 61.7% |
| *bert-large-cased* | Training Phase | $1 \times 10^{-5}$ | 62.5% |
| *bert-large-cased* | Training Phase | $2 \times 10^{-5}$ | 63.3% |
| *bert-base-cased* | Training Phase | $4 \times 10^{-5}$ | 62.7% |
| *bert-base-cased* | Training Phase | $8 \times 10^{-5}$ | 61.8% |
| *roberta-base* | Training Phase | $2 \times 10^{-5}$ | 61.6% |

Table 2.2: Accuracies computed by fine-tuning the weight decay parameter

| Model Name | Phase | Weight Decay | Accuracy |
|---|---|---|---|
| *bert-base-cased* | Trial Phase | 0 | 63.5% |
| *bert-large-cased* | Trial Phase | 0 | 63.9% |
| *bert-base-cased* | Training Phase | 0.5 | 62.0% |
| *bert-large-cased* | Training Phase | 0.5 | 63.5% |
| *bert-base-cased* | Training Phase | 0.93 | 61.9% |
| *bert-base-cased* | Training Phase | 0.7 | 62.1% |
| *bert-large-cased* | Training Phase | 0.95 | 62.4% |
| *bert-large-cased* | Training Phase | 1.3 | 63.2% |

Table 2.3: Accuracies computed by fine-tuning the max grad norm parameter

| Model Name | Phase | Max Grad Norm | Accuracy |
|---|---|---|---|
| *bert-base-cased* | Trial Phase | 0.5 | 63.5% |
| *bert-large-cased* | Trial Phase | 0.5 | 63.9% |
| *bert-large-cased* | Training Phase | 0.5 | 62.8% |
| *bert-large-cased* | Training Phase | 1 | 63.5% |
| *bert-base-cased* | Training Phase | 0.5 | 61.7% |
| *bert-base-cased* | Training Phase | 1 | 61.7% |

Table 2.4: Accuracies computed by fine-tuning the Adam epsilon parameter

| Model Name | Phase | Adam Epsilon | Accuracy |
|---|---|---|---|
| *bert-base-cased* | Trial Phase | $2 \times 10^{-8}$ | 63.5% |
| *bert-large-cased* | Trial Phase | $2 \times 10^{-8}$ | 63.9% |
| *bert-base-cased* | Training Phase | $2 \times 10^{-8}$ | 62.6% |
| *bert-large-cased* | Training Phase | $2 \times 10^{-8}$ | 62.8% |
| *bert-base-cased* | Training Phase | $2.5 \times 10^{-7}$ | 62.7% |
| *bert-large-cased* | Training Phase | $2.5 \times 10^{-7}$ | 63.0% |
| *bert-base-cased* | Training Phase | $3 \times 10^{-8}$ | 62.0% |

Table 2.5: Accuracy computed during testing phase (without bagging)

| Model Name | Learning Rate | Weight Decay | Max Grad Norm | Adam Epsilon | Accuracy |
|---|---|---|---|---|---|
| *bert-large-cased* | $2 \times 10^{-5}$ | 0 | 1 | $1 \times 10^{-8}$ | 69.9% |

Table 2.6: Accuracy computed during testing phase (with bagging)

| Model Name | Learning Rate | Weight Decay | Max Grad Norm | Adam Epsilon | Accuracy |
|---|---|---|---|---|---|
| *bert-large-cased* | $2 \times 10^{-5}$ | 0 | 1 | $1 \times 10^{-8}$ | 71.4% |

# Chapter 3

# The Bag-of-Words Model

## 3.1 Introduction

In our fine-tuning experiments, *bert-base-chinese* did nearly as well as *bert-base-english* and *bert-base-multilingual*, which suggested that perhaps the fine-tuned classifier might just be classifying using simple word-count heuristics. A bag-of-words model, or **BoW** for short, is a very simple and naive representation of natural language input (text). The term was first coined in the 1950s, by (Harris (1954)). In this model, each "document" (in our case, each document is a tweet) is represented as the multiset of its words, disregarding grammar and word order but keeping track of word frequencies. It is called a "bag" of words, because all structural information about the text is disregarded, and the model is only concerned with whether known words occur in the text, not where those words are in the text. As an example, consider the sentence

*"John likes to eat Chinese food but Mary likes to eat Mexican food."*

which is represented as:

$\{$*'John': 1,'likes': 2, 'to': 2, 'eat': 2, 'Chinese': 1, 'food': 2, 'but': 1, 'Mary': 1, 'Mexican': 1*$\}$

This approach is very simple and easy-to-apply, and can be used in many different ways to extract features from documents. The bag-of-words model is commonly used for document classification, where the frequency of occurrence of each word is used as a feature for training a classifier. Our desired classifier will be for sentiment analysis, i.e., for classify-

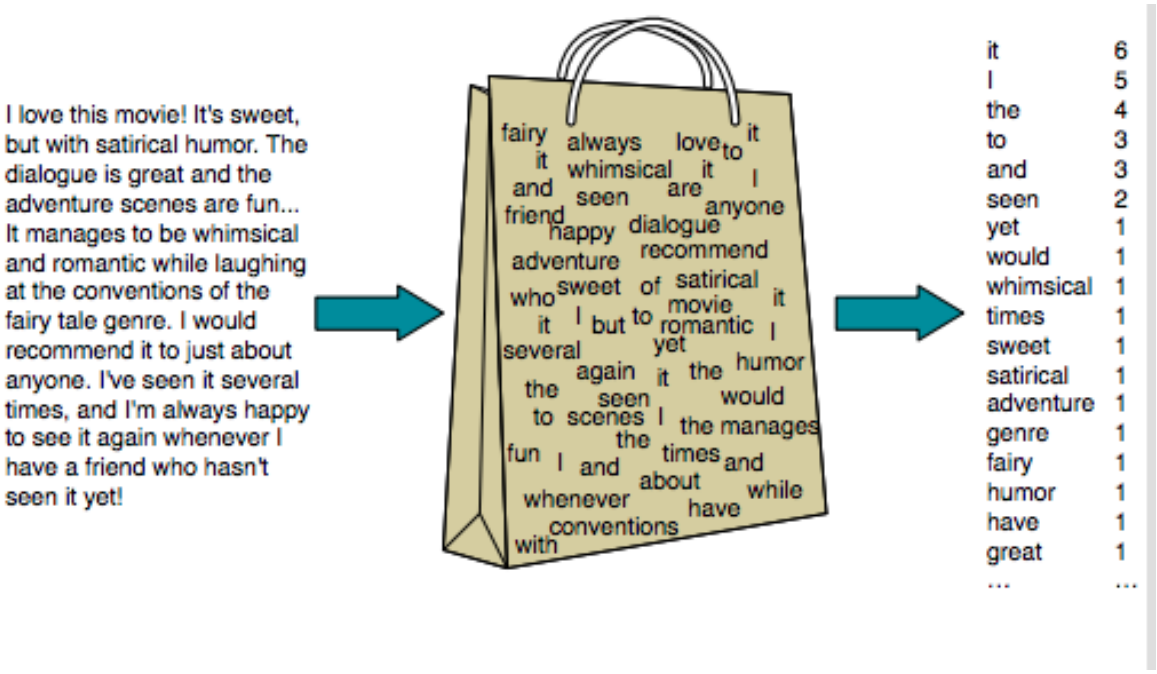


Figure 3.1: A visual representation of the bag-of-words model. The position of the words is ignored (the bag of words assumption) and we make use of the frequency of each word. (Jurafsky & Martin (2016)).

ing a document as positive, negative or neutral. A visual representation of the bag-of-words model is provided in Figure 3.1. Notice in the figure that a classifier training on this representation does not need to have any knowledge of English syntax or structure in order to determine that the document has a positive sentiment. The classifier can simply gather that information from the frequencies of positive words like *fun*, *great*, and *whimsical*.

While the syntactic structure of the document is ignored, some post-hoc optimizations are still applied. For example, *stop* words like 'the' and 'and' — i.e., words that do not contribute meaning without syntactic context — are often deleted. Also, words that are morphological variants —– for example, 'walk', 'walks', 'walking', 'walked' —– are sometimes represented as the base form of the word (in this case, 'walk') through a process known as *lemmatization* (McTear et al. (2016)).

The bag-of-words method has been widely used for information retrieval tasks, where a classifier based on the word frequencies can be used to retrieve documents that are relevant to the user's query. The intuition is that documents are similar if they have similar word histograms.The bag-of-words approach implicitly defines the topic of a document by its

content words. An application of this model, in the context of conversational interfaces, would be in question answering (Antol et al. (2015), Zhou et al. (2015)), where the bag-of-words approach can be used to identify textual sources in which the answer to a user's question can be found.

A major advantage of such a simple approach is that it does not require much linguistic knowledge (processes like lemmatization could be an exception to this rule). The major shortcoming is that often NLP tasks require more precise textual analyses that take into account the order of each sentence. An example from (McTear et al. (2016)) shows how the terms in the two sentences "John chased the dog" and "the dog chased John" would have the same BoW representation but the different orderings of the words have starkly different meanings. In our original example, we lose the ability to determine who likes Chinese food, and who likes Mexican food.

## 3.2 N-gram Representations

***"You shall know a word by the company it keeps."***

*(Firth, J. R. 1957:11)*

In addition to keeping track of the counts of words (frequencies), we can also record spatial context using a slight modification of the BoW model. Consider as an example the sentence "*John likes French food but Mary hates French food*". Here, the bag-of-words representation does not reveal the spatial information tied to the verbs '*likes*' and '*hates*', i.e., how the words always follow a person's name in this text. It also does not know that '*French food*' is something people like or dislike. As an alternative, the n-gram model can store this spatial information. Applying it to the aforementioned sentence, a bigram model will parse the text into the following units and store the term frequency of each unit as before, and including bigrams into our representation gives us

{*'John': 1,'likes': 1, 'French': 2, 'food': 2, 'but': 1, 'Mary': 1, 'hates': 1, 'John likes': 1, 'likes French' : 1, 'French food': 2, 'food but': 1, 'but Mary': 1, 'Mary hates': 1, 'hates French': 1*}

Above, our bag-of-words representation is extended with bigrams, i.e., $n = 2$. In essence, a

simple bag-of-words model is a special case of the n-gram model with $n = 1$. By extending the bag-of-words representation with n-grams, we can gather a lot more information and compact spatial and contextual information from neighboring words. This not only tells us how certain words always follow other words, but depending on how big $n$ is, it can also significantly improve the syntactic and structural understanding of a classifier. With $n = 4$, for example, a 4-gram can store a lot more information such as '*John likes French food*' and '*Mary hates French food*'. However, it is important to find the upper limit for the value of $n$ and performance, as higher values of $n$ require a lot more memory which leads to much slower performance. Furthermore, values greater than the threshold value of $n$ can also cause overfitting.

## 3.3 Experimental Setup

In this section, we describe the various experiments we conducted using feedforward neural networks with varying layers that trained on bag-of-words representations with n-gram extensions. For all our experiments, our training dataset comprises Hinglish tweets with their corresponding labels. A few examples of tweets in our training dataset are

*Saw the episode . Nice one sir . Maan gaye yaar sir aap bahut himmat waale* (positive)

*Hehe. I saw that coming. And it's actually from someone's shaadi* (neutral)

*Koi faida nahi bhaiya yeh automated message hi bhejte rahenge* (negative)

**General setup**

The first step of the process is to set up a baseline which can be tweaked for different kinds of experiments with varying parameters. To do this, we first go through the entire training data and store the frequencies of each word in the corpus. We then create a *vocabulary* $\mathbf{V} = \{v_1, v_2, ..., v_n\}$, which is the set of words that appear with frequency at least $K$ in the training data, for some positive hyperparameter $K \in \mathbb{N}$.

Before moving to the next step, we remove stop words from our vocabulary. We can then use this vocabulary to transform each of the tweets into a vector. We do this by initializing a vector of 0s of size $n$. Then, we set the $j^{th}$ position of this vector to 1 if $v_j \in \mathbf{V}$ from our

vocabulary appears in this tweet.

A good example of implementing this process can be found here [1]. In this example, suppose after implementing our frequency threshold, our vocabulary ends up consisting of 10 words:

$$V = \{\text{'it', 'was', 'the', 'best', 'of', 'times', 'worst', 'age', 'wisdom', 'foolishness'}\}$$

With this vocabulary, a sentence such as *"It was the age of wisdom"* is represented as

$$\text{“}It\ was\ the\ age\ of\ wisdom'' = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]$$

With this vector representation of input tweets, we can now convert our training and validation datasets of sizes 14K and 3K respectively into vector representations, and train simple feedforward NNs to classify the test dataset tweets as positive, negative or neutral.

With the vector representation of our datasets, we can train a simple classifier and compute its accuracy on the validation dataset. For our classifier, we used a standard feedforward NN with $N$ layers and hidden size $H$. We built and trained the NNs using PyTorch (Paszke et al. (2019)). In order to classify tweets into positive, negative and neutral, the final layer applies the softmax function to the 3-dimensional output of the NN, which normalizes the output into a probability distribution over the three possible sentiments of the input tweet. We used a cross entropy loss function for training.

**Count-of-Words Approach**

Instead of just representing whether a vocabulary word appears in a particular tweet as a binary value, we can instead represent the actual *frequency* of that word in that tweet. Using the example vocabulary **V** with 10 members, the sentence "*It was the best of times and it was an age of wisdom*" would be represented as

$$\text{“}It\ was\ the\ best\ of\ times\ and\ it\ was\ an\ age\ of\ wisdom'' = [2, 2, 1, 1, 2, 1, 0, 1, 1, 0]$$

The motivation for this modification is that the classifier might gain insight into the senti-

[1] see https://machinelearningmastery.com/gentle-introduction-bag-words-model/

ment of the overall sentence in relation to the frequency of particular 'positive' or 'negative' vocabulary members. With this modification, we used the same N-layer feedforward classifiers mentioned above, but now they train on the counts of the vocabulary members in order to classify the test dataset tweets as positive, negative or neutral.

**N-grams Extension**

Instead of applying a 1-gram simple bag-of-words representation, we extended our vocabulary by first including BiGrams and then extending that to TriGrams. The motivation behind this extension is fairly clear from the description above regarding n-gram representations, and suggests that extending our representation to include BiGrams and TriGrams should significantly improve the syntactical understanding of our various classifiers. This motivation shall be further strengthened in the results section, where we can see how the extension affects the vocabulary and hence the accuracy on the test datasets.

**Adjusting Parameters**

In all of the different experimental setups mentioned above, there are several parameters that we can tweak in order to sharpen our results. The effect on the accuracy of the classifiers upon tweaking each of these is documented clearly in the results. One of the first parameters we can change in order to directly affect the vocabulary and its size is the frequency threshold parameter, $K$. A lower threshold parameter should lead to a bigger vocabulary, which would certainly affect the accuracy and understanding of our classifier. Another characteristic that would probably affect the classifier's training is the relative *narrowness* (or *wideness*) between the layers of the feedforward NNs.

In terms of the training mechanism employed, an important parameter to tweak is the *batch size*. It is important to experiment with various batch sizes to determine which setup improves the training and hence the testing of the classifier. Based on the computation time due to relative narrowness (or wideness) of layers and large vocabulary size, it is also useful to know what a desirable number of *epochs* is, in order to avoid utilizing more time than required and more importantly to avoid overfitting.

Overfitting models is a common pitfall, and another useful methodology to try is including *Dropout layers* in between linear layers in our N-layer feedforward NNs, and experimenting with their dropout probabilities. These intermediary layers randomly drop some of the

training data and ensure no overfitting occurs, and can be implemented using pyTorch's inbuilt Dropout layers. A methodology following a similar motivation is including *Batch Normalization* in between our linear layers, which also attempts to prevent overfitting.

### 3.3.1 Results

As mentioned earlier, our training dataset and test dataset comprise tweets with their corresponding labels, and are of size 14K and 3K respectively. Using the aforementioned general setup, we used 2-layer, 3-layer and 4-layer feedforward NNs and conducted various runs of each experiment by varying several parameters.

The first important parameter is the threshold frequency, $K$. We observed a larger vocabulary size with a lower frequency threshold. The vocabulary size for a frequency threshold of $K = 20$ is around 1100, while for $K = 10$ it jumps to 2000. For a 2-layer feedforward NN with a hidden layer of $H = 784$, the best accuracy was obtained on lower frequencies of $K = 15$ and $K = 10$. The value of $H$ was chosen experimentally, with the idea of condensing the input vector to at least half its size in order to extract relevant information from the text. This at least confirmed our basic hypothesis that a larger vocabulary size would lead to more information and hence an increased language understanding.

After varying batch sizes (2, 3, 4, 6, 8, 10) and analyzing, we observed a greater accuracy-efficiency tradeoff for a batch size of 6. Experimenting with more layers, we implemented a 3-layer feedforward NN with hidden sizes $H1 = 784$, $H2 = 385$, and a 4-layer feedforward NN ($H3 = 112$). In our experiments, the 2-layer NN had a better performance and yielded a greater accuracy of 58.9% compared to 58% and 57.8% of the 3-layer and 4-layer models respectively. To prevent overfitting and also because our training dataset was fairly large, we kept the number of epochs at 20. We observed a lower accuracy due to overfitting at higher epochs.

The next parameter we tweaked was the relative width between the layers, in order to understand if a more 'narrow' or 'wide' setup was helpful in any way. We obtained higher accuracies upon implementing narrower layers. For example, decreasing hidden size $H$ to 350 (instead of 784) yields an accuracy of 59.3%. This was also observed to a lesser degree using more layers, with the 3-layer feedforward ($H1 = 350,\ H2 = 75$) and the 4-layer feedforward ($H1 = 350,\ H2 = 180,\ H3 = 75$) yielding slightly higher accuracies

of 58.3% and 58%. This only slight increase is because finding the right balance between many layers is tricky, and changing the width of 1 layer can hugely affect the information received by other layers. We performed different variations with the widths, such as narrowing all the layers, widening all the layers, narrowing initial layers and widening later layers, and widening initial layers and narrowing initial layers.

A brief summary of our results using the simple Bag-of-words approach can be found in Table 3.1.

### Count-of-words Approach

After implementing the general setup, we conducted experiments using the count-of-words approach: using the actual count of a vocabulary member in a tweet, i.e., how many times it appears in that sentence instead of only depicting whether it appears in the sentence with a 1 or a 0. Since the vocabularies being generated in the count-of-words approach are the same as the BoW approach (for the same frequency threshold), the behavior is fairly similar to the BoW results, in that we notice a higher accuracy for narrower layers with lower frequency thresholds. We also observe that a lower number of epochs is better to prevent overfitting, and that the best accuracy-efficiency tradeoff is yielded by a batch size of 6. In general, however, the accuracies computed in the count-of-words approach are slightly lower than the simple BoW approach, which implies that using the count of vocabulary members does not pass on more information to the NNs. A brief summary of the different experimental runs can be found in Table 3.2

### N-Grams Extension

In our experiments, we extended the vocabulary of the simple BoW approach, which comprises 1-grams, to include BiGrams (N = 2) and TriGrams (N = 3). From all runs in the previous experiments, we had gotten the best results using a 2-layer feedforward NN. So for all N-gram experiments, we decided to train and test only on the 2-layer NN.

Upon including BiGrams, the size of the vocabulary significantly increased. For example with $K = 15$, we noticed an increase in vocabulary size from about 1800 to about 2600.

This increase in vocabulary and syntactic understanding yielded higher accuracies, and narrowing layers to extract more critical information pushed that number further, giving us an accuracy of 59.96%. Including TriGrams did not have as significant an impact as BiGrams did, mostly because recurring TriGrams are less frequent, and hence including TriGrams didn't really affect the vocabulary by much (an increase of around 10 more vocabulary members). So most of the TriGrams didn't make it past the frequency threshold K. Nevertheless, there were slight increases in accuracy to 60.01%.

Since the inclusion of BiGrams and TriGrams yielded the best results, we further decided to use that representation and experiment with Dropout and Batch Normalization. We tried different probabilities of Dropout in the intermediary layers, as well as different variants of Batch Normalization between layers. We observed that implementing Dropout and Batch Normalization yield lower accuracies, and in the context of this experiment, randomly removing some data ends up being harmful because the model gets a lesser amount of information. A brief summary of our obtained results is provided in Table 3.3

Table 3.1: Accuracies computed using simple Bag-of-Words Approach

| Model | *K* | H1 | H2 | H3 | Accuracy |
|---|---|---|---|---|---|
| 2-layer feedforward | 10 | 784 | N/A | N/A | 58.8% |
| 2-layer feedforward | 15 | 784 | N/A | N/A | 58.6% |
| 2-layer feedforward | 20 | 784 | N/A | N/A | 58.3% |
| 3-layer feedforward | 15 | 784 | 385 | N/A | 58% |
| 4-layer feedforward | 15 | 784 | 385 | 112 | 57.8% |
| 2-layer feedforward | 15 | 350 | N/A | N/A | 59.3% |
| 3-layer feedforward | 15 | 350 | 75 | N/A | 58% |
| 4-layer feedforward | 15 | 350 | 180 | 75 | 57.8% |

Table 3.2: Accuracies computed using Count-of-Words Approach

| Model | $K$ | H1 | H2 | H3 | Batch Size | Epochs | Accuracy |
|---|---|---|---|---|---|---|---|
| 2-layer feedforward | 15 | 784 | N/A | N/A | 6 | 20 | 58.6% |
| 2-layer feedforward | 15 | 784 | N/A | N/A | 4 | 20 | 58.1% |
| 2-layer feedforward | 15 | 784 | N/A | N/A | 4 | 25 | 58.4% |
| 2-layer feedforward | 15 | 784 | N/A | N/A | 6 | 40 | 57.8% |
| 3-layer feedforward | 15 | 784 | 385 | N/A | 6 | 20 | 57.5% |
| 4-layer feedforward | 15 | 784 | 385 | 112 | 6 | 20 | 57% |
| 2-layer feedforward | 15 | 350 | N/A | N/A | 6 | 40 | 58.9% |

Table 3.3: Accuracies computed using N-Grams Extension

| Model | $K$ | H | p1 | p2 | Before ReLU | Before Softmax | Accuracy |
|---|---|---|---|---|---|---|---|
| 2-layer feedforward (BiGrams) | 15 | 350 | 0.0 | 0.0 | No | No | 59.7% |
| 2-layer feedforward (BiGrams) | 15 | 300 | 0.0 | 0.0 | No | No | 59.96% |
| 2-layer feedforward (TriGrams) | 15 | 300 | 0.0 | 0.0 | No | No | 60.01% |
| 2-layer feedforward (TriGrams) | 15 | 300 | 0.2 | 0.5 | No | No | 56.9% |
| 2-layer feedforward (TriGrams) | 15 | 300 | 0.3 | 0.4 | No | No | 58.3% |
| 2-layer feedforward (TriGrams) | 15 | 300 | 0.1 | 0.2 | No | No | 59.5% |
| 2-layer feedforward (TriGrams) | 15 | 300 | 0.0 | 0.0 | Yes | No | 56.2% |
| 2-layer feedforward (TriGrams) | 15 | 300 | 0.0 | 0.0 | No | Yes | 55.6% |
| 2-layer feedforward (TriGrams) | 15 | 300 | 0.0 | 0.0 | Yes | Yes | 56.1% |

# Conclusion

In this thesis, we explored sentiment analysis of code-mixed Hinglish (Hindi-English) tweets as participants of Task 9 of the SemEval 2020 competition, formally known as the SentiMix task. We had two main approaches: 1) Applying transfer learning by fine-tuning pre-trained models like $BERT_{BASE}$ and $BERT_{LARGE}$ and 2) Training feedforward NNs on Bag-of-Words and N-gram representations. We hypothesized during fine-tuning that pre-trained weights were being computed from scratch while training on the target task, which suggested that the language capabilities of the pretrained models were not very helpful for the code-mixed data. To test whether that was the case, we attempted to recreate the BERT fine-tuning results only using feedforward NNs that were trained on a BoW representation. In the training phases of the competition, the fine-tuning model had a highest accuracy of 63.9% and the BoW model had a highest accuracy of 60.01%, which showed us that the results of fine-tuning $BERT_{LARGE}$ (24 layers) could be approximated by a 2-layer BoW feedforward NN in the case of Hinglish text. Since we were only allowed 3 submissions in total during the testing phase, we decided to submit the fine-tuning results as they had higher accuracies. During the testing phase, we obtained an accuracy of 69.9% without bagging and 71.4% with bagging with our fine-tuning model. The official system rankings were published on April 6, 2020 and our submission with the bagging results placed $4^{th}$ out of 62 entries. All the code needed to recreate our results can be found here [2].

[2] see https://github.com/gopalanvinay/thesis-vinay-gopalan

# References


Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., & Parikh, D. (2015). Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, (pp. 2425–2433).

Arbel, N. (2019). *Attention in RNNs*. `https://medium.com/datadriveninvestor/attention-in-rnns-321fbcd64f05`

Asur, S., & Huberman, B. A. (2010). Predicting the future with social media. In *2010 IEEE/WIC/ACM international conference on web intelligence and intelligent agent technology*, vol. 1, (pp. 492–499). IEEE.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, (pp. 17–36).

Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep learning*, vol. 1. MIT press.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, (pp. 153–160).

Bollen, J., Mao, H., & Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of computational science*, *2*(1), 1–8.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Chen, P., Sun, Z., Bing, L., & Yang, W. (2017). Recurrent attention network on memory for aspect sentiment analysis. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, (pp. 452–461).

Dai, A. M., & Le, Q. V. (2015). Semi-supervised sequence learning. In *Advances in neural information processing systems*, (pp. 3079–3087).

Das, S. R., & Chen, M. Y. (2001). Yahoo! for amazon: Sentiment parsing from small talk on the web. *For Amazon: Sentiment Parsing from Small Talk on the Web (August 5, 2001). EFA*.

Deng, Z., Choi, K.-S., Jiang, Y., & Wang, S. (2014). Generalized hidden-mapping ridge regression, knowledge-leveraged inductive transfer learning for neural networks, fuzzy systems and kernel methods. *IEEE transactions on cybernetics*, *44*(12), 2585–2599.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ethnologue (2019). *Summary by Language Size*. `https://www.ethnologue.com/statistics/summary-language-size-19`

Harris, Z. S. (1954). Distributional structure. *Word*, *10*(2-3), 146–162.

Hebb, D. O. (1949). *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., et al. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, *5*, 339–351.

Joshi, M., Das, D., Gimpel, K., & Smith, N. A. (2010). Movie reviews and revenues: An experiment in text regression. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, (pp. 293–296). Association for Computational Linguistics.

Jurafsky, D., & Martin, J. (2016). Naive bayes and sentiment classification. *Speech and Language Processing*, *7*.

Kim, Y., Denton, C., Hoang, L., & Rush, A. M. (2017). Structured attention networks. *arXiv preprint arXiv:1702.00887*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, (pp. 1097–1105).

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., & Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, (pp. 1378–1387).

Li, Z., & Hoiem, D. (2017). Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, *40*(12), 2935–2947.

Liu, B. (2012). Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, *5*(1), 1–167.

Liu, Y., Huang, X., An, A., & Yu, X. (2007). Arsa: a sentiment-aware model for predicting sales performance using blogs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 607–614).

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

McTear, M. F., Callejas, Z., & Griol, D. (2016). *The conversational interface*, vol. 6. Springer.

Menabrea, L. F., & Lovelace, A. (1842). Sketch of the analytical engine invented by charles babbage.

Minsky, M., & Papert, S. (1969). An introduction to computational geometry. *Cambridge tiass., HIT*.

Mor-Yosef, S., Samueloff, A., Modan, B., Navot, D., & Schenker, J. G. (1990). Ranking the risk factors for cesarean: logistic regression analysis of a nationwide study. *Obstetrics and gynecology*, *75*(6), 944–947.

Muysken, P., Muysken, P. C., et al. (2000). *Bilingual speech: A typology of code-mixing*. Cambridge University Press.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, (pp. 807–814).

Pak, A., & Paroubek, P. (2010). Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, vol. 10, (pp. 1320–1326).

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, (pp. 79–86). Association for Computational Linguistics.

Pang, B., Lee, L., et al. (2008). Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, *2*(1–2), 1–135.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Sanchez, N. (2002). Communication process. *Hyperlink [http://www. stfrancis. edu/ba/ghkickul/stuwebs/btopics/works/comproc. html]*, *17*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, (pp. 1631–1642).

Statista (2013). *Statistic on Languages used on Twitter*. https://www.statista.com/statistics/267129/most-used-languages-on-twitter/

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 1–9).

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 2818–2826).

Taylor, W. L. (1953). "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, *30*(4), 415–433.

Tumasjan, A., Sprenger, T. O., Sandner, P. G., & Welpe, I. M. (2010). Predicting elections with twitter: What 140 characters reveal about political sentiment. In *Fourth international AAAI conference on weblogs and social media*.

Uriel, W. (1953). Languages in contact. *The Hague: Mouton*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, (pp. 5998–6008).

Werbos, P. J. (1994). *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, vol. 1. John Wiley & Sons.

Yang, C., Lin, K. H.-Y., & Chen, H.-H. (2007). Emotion classification using web blog corpora. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI'07)*, (pp. 275–278). IEEE.

Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, (pp. 189–196).

Zhou, B., Tian, Y., Sukhbaatar, S., Szlam, A., & Fergus, R. (2015). Simple baseline for visual question answering. *arXiv preprint arXiv:1512.02167*.