Bridging the Gap between Pre-Training and Fine-Tuning for Improved Text Classification

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computational Linguistics by Research

by

Tanish Lad
2018114005
tanish.lad@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2023

Copyright © Tanish Lad, 2023
All Rights Reserved

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Bridging the Gap between Pre-Training and Fine-Tuning for Improved Text Classification" by Tanish Lad, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Prof. Radhika Mamidi

To my parents, Amita and Nitin,
my brother, Tejas,
and my grandfather, Navin.

Acknowledgments

As I look back on my thesis journey, I am filled with immense gratitude towards the multitude of individuals who have been instrumental in my achievements.

First and foremost, I cannot express how deeply I am grateful to my advisor, Prof. Radhika Mamidi, whose unwavering guidance, feedback, and support have been invaluable. Your expertise and mentorship have been instrumental in shaping both my research and professional development. I will miss attending your classes, and sitting in LTRC pretending to work!

I am also deeply grateful to my former advisor, Prof. Vasudeva Varma, whose expertise and experience have played a pivotal role in shaping my research direction and career path. I am also indebted to my mentor/buddy/friend Himanshu during that time for his guidance, support, and insights. His experiences have been a source of inspiration and have helped shape my work.

How can I ever forget my friends? They were the most special and memorable part of my college journey. Manish, Akshat, Vasu, and Dhruv, you are the most amazing people, teammates, buddies, and friends I have ever met. Thank you for being there for me when I needed you, for putting up with my rants, and for making college life unforgettable. Words fail to do justice to your incredible personalities and the impact you have had on me. Jashn, your support and patience has been indispensable. You have always been there to answer my most foolish questions and to offer words of encouragement. Prabhav, you are not just the best roommate I could have ever asked for, but also a dear friend. I am grateful for the memories we have created together. Siddharth, Kartik, Raghav, Daksh, Parv, Priyanshu, Aniketh, Nikunj, and tens of many others, you have all played a special role in my life. You were there to listen, to offer a helping hand, and to make college life an adventure. Your impact on me will be felt for years to come. A special thanks to all the wonderful seniors, Gaurav, Sayantan, Tanuj, Arnav, and Puru, who have guided and supported me on various occasions.

Finally, I am grateful for my loving family and all their support. They are the cornerstone of my life. Thank you for taking a chance and letting me come to IIIT.

To all of you folks, I owe a debt of gratitude that cannot be expressed in words. Thank you for being an integral part of my research journey and for shaping me into the researcher and person that I am today. Let's raise a toast to a promising future!

Abstract

Pre-training language models like BERT and then fine-tuning them on downstream tasks has demonstrated state-of-the-art results in various natural language processing tasks. However, pre-training is usually independent of the downstream task, and previous works have shown that this pre-training alone might not be sufficient to capture the task-specific nuances. We propose a novel way to tailor a pre-trained BERT model for the downstream task via task-specific masking prior to the standard supervised fine-tuning.

Our approach involves first creating a word list specific to the task at hand. For example, if the task is sentiment analysis, we gather a small set of words that represent positive and negative sentiments. If the task is hate speech detection, then we gather a small set of words that represent hate words. If the task is humor detection, then we gather a small set of words that represent humorous words. Next, we use word embeddings to measure the importance of each word in the task using the word list, which we call the word's task score. Based on the task score, we assign a probability of masking to each word. This probability reflects the likelihood of masking the word during the fine-tuning process.

We experiment with different masking functions, including the step function, the linear function, and the exponential function, to determine the best approach for assigning the probability of masking based on the word's task score. We then use this selective masking strategy to train the BERT model on the masked language modeling (MLM) objective. During MLM training, rather than randomly masking 20% of the input tokens, we selectively mask these input tokens based on their assigned probability of masking, calculated using the word's task score.

Finally, we fine-tune the BERT model on different downstream binary and multi-class classification tasks, such as sentiment analysis, hate speech detection, formality style detection, named entity recognition, and humor detection. Our experiments show that our selective masking strategy outperforms random masking, indicating its effectiveness in fine-tuning the pre-trained BERT model for specific tasks.

Overall, our approach provides a more targeted and effective way of fine-tuning pre-trained language models for specific tasks, by incorporating task-specific knowledge between the pre-training and the fine-tuning stages. By selectively masking input tokens based on their importance, we are able to better capture the nuances of a particular task, leading to improved performance on various downstream classification tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 NLP in our day-to-day lives

Nowadays, everyone is talking about Natural Language Processing, *aka NLP*. It has become a buzzword. But what's so fascinating about NLP? Why does everyone keep talking about it? Let's take a closer look.

NLP is used almost everywhere in our daily life. Below are a few examples where we encounter NLP in our day-to-day life. Phrases marked in bold are some of the core high-level problems that researchers are constantly trying to solve.

We use **language modeling** tools such as keyboard **auto-complete**, email auto-complete, and candidate suggestions for google search completion (refer figure 1.1) hundreds of times daily in our lives. These tools hugely boost our productivity.

Virtual assistants like Alexa, Siri, and Google Assistant use NLP to understand human commands and respond appropriately. These virtual assistants use **speech recognition** technology to convert spoken words into text, which is then analyzed by **NLP algorithms** to understand the intent behind the user's command.

Machine translation tools like Google Translate use NLP to **translate text** from one language to another. These algorithms analyze the structure of sentences and the meaning of words in order to produce accurate translations.

Chatbots or **Dialogue Systems** are computer programs that use NLP to **simulate human conversation**. These bots can be used to provide customer support, answer frequently asked questions, or even carry out simple transactions like booking a flight or ordering food. The domain of Chatbot research has been further bolstered by the arrival of ChatGPT. What a fascinating tool! Tools such as ChatGPT, YouChat, Google's LaMBDA, and Bing's integration of a powerful ChatGPT-like model in it's search engine are taking over. This is our future. These tools will be our copilots for the web!

Sentiment Analysis: NLP can be used to **analyze the sentiment** of large volumes of text data, such as social media posts, product reviews, and customer feedback. Sentiment analysis tools can identify

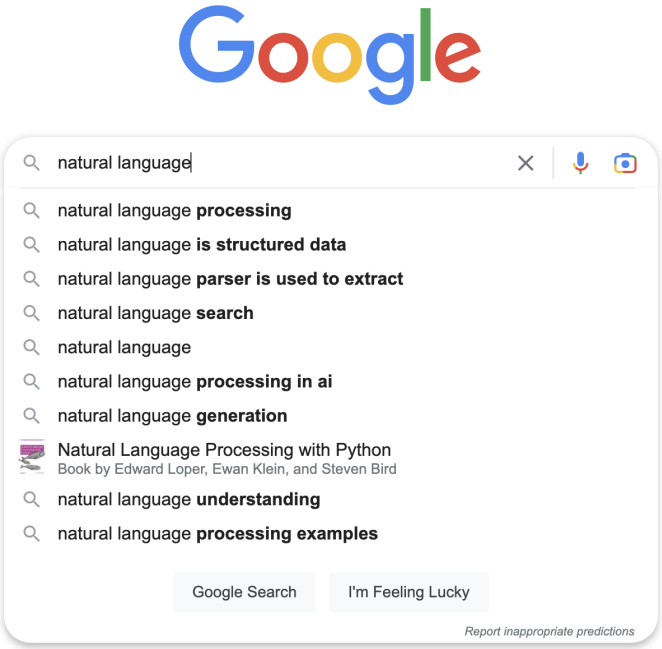

Figure 1.1 Google Search shows top-ranked candidates to complete the phrase "natural language".

the overall sentiment of a piece of text as positive, negative, or neutral, as well as specific emotions like happiness, anger, and sadness.

Information Extraction: NLP algorithms can be used to **extract structured information** from unstructured text data. For example, they can identify key entities like people, places, and organizations, as well as relationships between them

NLP is an active area of research and has significant potential for innovation and transformation across many industries, including healthcare, finance, education, and many more.

Ever wondered what will you do without these tools? We did too. Now that we know NLP is useful and something without which we cannot live, let's start from understanding the roots of NLP.

1.2 Introduction to NLP

The phrase Natural Language Processing comes from two phrases: *Natural Language* and *Processing*.

- Natural Languages are languages that evolved naturally through human use for e.g. Hindi, Telugu, English, etc.
- Processing means to do anything computationally with text such as rule-based approaches, supervised learning, unsupervised learning, and text generation, among others.

Talking about the processing side, Supervised learning involves learning a mapping from a given text X to a label Y using a given collection of labelled examples. For example, sentiment analysis is a task that requires supervised learning, where the goal is to predict the sentiment (Y) of a given text (X). However, a major downside of supervised learning is that a model trained on one task cannot be used for other tasks. A model trained on sentiment analysis cannot be used for machine translation or named-entity recognition. It is also challenging to collect labelled examples for every task.

Driven by the limitations of the supervised learning paradigm, self-supervised learning (or pre-training) has gained traction. The task in self-supervised learning is to train a powerful NLP model without relying on large amounts of labelled X-Y pairs. Instead, a collection of just text, without extra labels, is used to create labels and pre-train a model that has a general understanding of human language.

Labelled pairs may not be available for supervised learning but we have the whole internet available for pre-training. The internet contains a lot of questions and answers, and by training a model on such data, it can learn to pick up various tasks such as question answering without requiring us to curate the data ourselves.

One of the core tasks in self-supervised learning is language modelling, which involves predicting the next word in a sentence. Language modelling is an important task because it requires the model to have a certain level of semantic understanding about the world and the language being used. For instance, if the task is to predict the word that goes in the blank ”Tanish went to a restaurant and ordered a BLANK”, the model needs to know that the word should be a food item that can be ordered in a restaurant. Additionally, the model should understand that when a sentence includes the phrase ”ordered a BLANK”, the word that fills the blank is most likely to be a noun and not a preposition or any other part of speech.

Self-supervised learning involves pre-training a model on unlabelled text, but our intent is to solve more specialized NLP tasks such as sentiment analysis or question answering. Transfer learning was introduced as a solution to this challenge. It involves pre-training a large self-supervised model and then fine-tuning it on a small labeled dataset using supervised learning.

The reason transfer learning is preferred over just doing supervised learning is that if we rely solely on supervised learning, the model starts from scratch and needs to learn the entire language from just a few thousand examples. With transfer learning, we start with a model that's already trained to predict the next word and, therefore, has a reasonable understanding of language. We then repurpose this pre-trained model to solve the specific task in hand, leveraging the labeled data available for that task.

In recent times, prompt-based learning has emerged as a powerful alternative to transfer learning. The fundamental idea behind prompt-based learning is to pre-train a large self-supervised model on a massive corpus of text data and then prompt it with natural language to solve specific tasks without any further training. For instance, let's consider the task of sentiment analysis. With prompt-based learning, we first pre-train the model on billions or trillions of words, and then feed it a prompt in natural language, such as ”what is the sentiment of the sentence: [insert sentence],” to make it solve sentiment analysis.

The key advantage of prompt-based learning over transfer learning is that a single model can solve a wide range of tasks, whereas with transfer learning, we need to fine-tune the model for each task separately. This makes prompt-based learning highly efficient and cost-effective.

The most prominent example of prompt-based learning one can think of is ChatGPT, where a large language model is pre-trained on a massive amount of text data. This model can then be prompted with natural language to perform various tasks. Similarly, OpenAI's Codex is a new AI system that is pre-trained on extensive amounts of code. It can be prompted with natural language to write code in different programming languages.

1.3 Thesis Motivation

The conventional pre-training and then fine-tuning paradigm in NLP models has always left us feeling disconnected and searching for a better solution. It seemed counterintuitive to us that pre-training and downstream tasks should be independent of each other. We believed that there had to be a more effective approach that bridged this gap and allowed the model to learn better.

We always thought there must be something that bridges the gap between pre-training and fine-tuning. And that's how we knew what problem we were going to solve. Our research involved a thorough review of literature and resources related to the pre-training and then fine-tuning paradigm, with a focus on identifying their shortcomings and exploring various approaches to address the problem.

After years of dedicated research, we are proud to present our findings on a promising new solution: a framework using selective masking, word lists, and masking functions, which has been a major focus of our work, and we have provided detailed insights on this approach in this thesis.

1.4 Major Contributions

- Bridge the gap between pre-training and fine-tuning for text classification tasks.
- Introduce the concept of word lists as the bridging gap between pre-training and fine-tuning.
- Introduce the concept of Masking Functions and how masking functions can be used to selectively mask the *important* words in a dataset.
- Propose an architecture using word lists, masking functions, and selective masking which achieves state-of-the-art results in multiple tasks including Sentiment Analysis, Hate Speech Detection, Formality Detection, and Named-Entity Recognition.

The results demonstrate the effectiveness of this approach in addressing the challenges of pre-training and fine-tuning in text classification, and its potential for improving the accuracy and efficiency of various models.

1.5 Thesis Workflow

This rest of this thesis is organized in the following way:

- **Chapter 2** contains the background and relevant literature to understand the actual work. We start from the basics: word representations to the actual topic in hand: selective masking. We look at how word representations evolved to RNNs which evolved to transformers which finally evolved to selective masking.
- **Chapter 3** takes a look at the binary text classification problem. We take three tasks: sentiment analysis, hate speech detection, and formality detection, and show how selective masking can be applied to existing works to achieve much better results. We also talk about word lists, and masking functions.
- **Chapter 4** takes a look at the multi-class text classification problem. We summarize the difference between multi-class and binary-text classification problems and present the techniques needed to be modified to make selective masking work for multi-class classification problems, and show that selective masking can be applied to achieve much better results by taking the example of Named-Entity Recognition.
- **Chapter 5** takes a look at the importance of word lists in our work. We take the example of the task of Humor Detection where we deliberately chose an *incorrect* word list and demonstrate that there are no improvements in the results when the word list doesn't synchronize with the fine-tuning dataset.
- Finally, **Chapter 6** summarizes the main findings of the works described in the thesis and puts forward the limitations and possible future directions of our work.

Chapter 2

Background and Literature: From Language Modelling to Selective Masking

2.1 Language Modelling

Language Modelling aims to predict the next word in a sequence given a context.

What do we want to do?

Predict the likelihood of a sequence of words or phrases based on the information available in the training corpus.

How do we do it?

Mathematically, it translates to estimating the probability distribution of sequences of words, and the model is trained to maximize the likelihood of the correct word sequence.

Why is Language Modelling important to discuss?

We use language models every day in our lives. An application of the conditional form of language modelling is to predict what is the probability of the next word in a sentence given a context (i.e. previous few words in the sentence). This form is heavily used during email auto-complete and finding good candidates for Google search auto-completion.

Generally, we talk in terms of conditional probabilities i.e. the probability of a word given a context, but it is still useful to talk about the probability of a sentence as a whole, For example: suppose we have a machine translation model that gives us multiple candidates for a translation, and we need to rank them, then we can do so by ranking them in order of their probabilities for e.g.

$$P(\text{I flew to the movies}) <<< P(\text{I went to the movies})$$

Another example: consider the case of speech recognition. The following are two valid English sentences. Both of them sound very similar. Humans can decide which is the obvious correct choice but how can a speech recognition model decide which is the valid one: because one is more probable than the other:

$$P(\text{eyes awe of an}) <<< P(\text{I saw a van})$$

With selective masking, the approach to language modelling becomes a form of masked language modelling, where a portion of the input sequence is masked or replaced with a special token [MASK]. The model is then trained to predict the masked tokens based on the context provided by the rest of the sequence. We will describe Selective Masking in more detail towards the end of the chapter.

Traditionally, n-gram Language Models were used for Language Modelling. The probability of a word given a context was some or the other variation of the following formula:

$$P(\text{thesis } | \text{ Tanish is writing his}) = \frac{\text{count(Tanish his writing his thesis)}}{\text{count(Tanish his writing his)}} \tag{2.1}$$

One problem with such an approach was that if the phrase "Tanish is writing his thesis" never occurred in the data, then the probability of the phrase is 0. This is called the sparsity problem. A partial solution is to add a small δ in the count of every word in the probability. This is called smoothing.

Another problem of n-gram Language Models was the storage cost. We need to store the counts for all possible n-grams implying that the storage is exponential in the vocabulary size of the model.

Another problem is that we treat all the prefixes/words independently of each other. We should ideally share information across semantically similar prefixes. So if "A student is writing their thesis" occurs in the data but "Tanish is writing his thesis" does not, then we should somehow use the earlier phrase to account for the probabilities of the latter because both phrases are semantically similar.

Let's move on to Word Representations in order to understand how we can solve these problems.

2.2 Word Representations

Word representations are an essential component of NLP, allowing computers to work with text data in such a way that they can understand the semantics of the language. For example, word embeddings can be used to represent the meaning of words in a vector space, where words that are semantically related are located near each other. Word embeddings are low-dimensional, continuous-valued vector representations of words. By representing words as numerical vectors, computers can perform mathematical operations on words, such as addition, subtraction, and cosine similarity, to understand relationships between words and phrases.

These embeddings can be learned from large amounts of text data using unsupervised learning methods, such as word2vec or GloVe. Both of them, and a few other techniques, will be discussed in this section.

2.2.1 One-hot Representations

Traditionally, one-hot encoding was a very common way to represent words. In one-hot encoding, we represent each word in a vocabulary as a binary vector, where the vector has a length equal to the size of the vocabulary, and only one element is set to 1 (i.e., "hot") while all others are set to 0.

For example, if our vocabulary consists of three words: cat, dog, and book, then the word cat would be represented as the vector $[1, 0, 0]$, dog as $[0, 1, 0]$, and book as $[0, 0, 1]$.

Formally, let's say we have a vocabulary V of size $|V|$, where each word $v_i \in V$ is assigned a unique index i. The one-hot representation of word v_i is a vector $\boldsymbol{x}_i \in \{0, 1\}^{|V|}$, where the i-th component of the vector is 1, and all other components are 0:

$$\boldsymbol{x}_i = \begin{bmatrix} 0 & 0 & \dots & 1 & \dots & 0 \end{bmatrix}^T,$$

where the 1 is in the i^{th} position of the vector. This form of encoding has several advantages, including simplicity, interpretability, and efficiency.

However, obviously, they suffer from many problems:

1. The dimension of the one-hot encoding vector is the size of the vocabulary, and vocabulary generally tends to be very large, which can lead to sparsity and computational inefficiencies.

2. The inability to capture the semantic relationships between words and the high dimensionality of the resulting vectors: there is no notion of similarity. Cat and dog should be closer to each other (since they are animals) and farther from book. But the euclidean distance between any two vectors is $\sqrt{2}$ and the cosine similarity between any two vectors is 0.

Hence, there is a need to look out for distributed representation of words.

2.2.2 TF-IDF

TF-IDF stands for Term Frequency-Inverse Document Frequency. That is a bit of a mouthful, no? It is a method for generating word representations that aim to capture the importance of a word in a document or a corpus [89]. It considers both the frequency of the word in the document and the inverse frequency of the word across the corpus.

Formally, the TF-IDF weight of a word w in a document d is defined as the product of its term frequency (TF) and inverse document frequency (IDF):

$$\text{TF-IDF}(w, d) = \text{tf}(w, d) \quad \text{idf}(w),$$

where $\text{tf}(w, d)$ is the number of times the word w occurs in the document d, and $\text{idf}(w)$ is the inverse document frequency of the word w, defined as:

$$\text{idf}(w) = \log \frac{N}{df(w)},$$

where N is the total number of documents in the corpus, and $df(w)$ is the number of documents in the corpus that contain the word w.

The logarithmic scaling of IDF ensures that rare words have a higher weight than common words, and the product of the TF and IDF weights gives a higher weight to words that are frequent in the document but rare in the corpus.

TF-IDF has several advantages over simpler methods like one-hot encoding, including its ability to capture the importance of words in a document, handle noisy and irrelevant terms, and relatively low computational complexity. TF-IDF does capture some semantic information, however, it still suffers from the same limitations as one-hot encoding since it does not capture any contextual relationships between words.

2.2.3 Using SVD for learning word embeddings

There is a famous saying by John Rupert Firth: "*You shall know a word by the company it keeps*".

This saying leads us to the idea of using co-occurrence matrices to represent the relationships between words in the dataset. The matrix is constructed by counting the number of times each word appears in close proximity to every other word in the corpus. The resulting matrix is a square matrix, with the rows and columns representing words, and the entries representing the number of times a pair of words appear together.

But, the flaws are similar to the ones in one-hot representations:

1. The size of the matrix is proportional to the square of the vocabulary.

2. The matrix will be sparse: a lot of entries will be zero, as most words do not co-occur with every other word.

SVD (Singular Value Decomposition) is a mathematical technique that is used to reduce the dimensionality of a matrix while preserving its most important information. SVD can be used to reduce the co-occurrence matrix to a low-dimensional matrix where each row represents a word and each column represents a latent semantic dimension [45]. These reduced dimensions can be used to represent the relationships between words in a more compact and interpretable form.

The only major problem with SVD is that it is computationally very expensive and hence this technique is not used very frequently to capture word representations. Let us move on to word2vec, which is, arguably, one of the most important advancements in the field of NLP. There is a work that proves that in many cases, SVD's embeddings are as good as word2vec [44].

2.2.4 Word2Vec

Word2vec is a neural network-based method for learning word embeddings . In the case of word2vec, the word embeddings generated are dense vector representations of words and are able to capture the semantic and contextual relationships between them. There are two types of word2vec models: CBOW and Skip-Gram, both introduced in one of the most influential papers regarding word representations [55].

2.2.4.1 CBOW

The objective of the Continuous Bag-of-Words (CBOW) model is to predict the target word given a set of context words (the words that appear before and after the target word in a given window), i.e., given a set of context words, the CBOW model predicts the probability distribution of the target word. The goal is to minimize the difference between the predicted vector and the true vector representation of the target word.

The CBOW model consists of an input layer, an embedding layer, and an output layer. The input layer of CBOW takes a bag-of-words representation of the context words as input, i.e., a concatenated vector of one-hot encoded vectors representing the context words. This input vector is then multiplied with the weight matrix of the embedding layer to produce the dense vector representation of the context words. The output layer uses this dense vector representation to predict the probability distribution of the target word. The hidden layer contains the learned word embeddings, which are used to map the context words to the target word.

The objective of the CBOW model is to maximize the probability of predicting the target word given the context words. This can be achieved by minimizing the negative log-likelihood of the predicted target word.

The context needs to be from both directions (previous and next few words) and not only from the left (previous few words) because it is possible that the current word to be predicted is an adjective to a noun (noun is the next word in the sentenec), but since the model won't know that (it only sees the previous words), it might predict the noun to be the target word rather than the desired adjective.

SGD (Stochastic gradient descent) is used to update the model parameters iteratively by backpropagating the error signal from the output layer to the hidden layer to train the CBOW model.

2.2.4.2 Skip-Gram

In addition to the CBOW model, word2vec also includes another model known as skip-gram, which has a similar architecture but with the roles of the input and output reversed. It is a shallow neural network with one hidden layer and is the most common form of word2vec.

In skip-gram, the objective is to predict the context words given a target word, i.e., given a target word in a sentence, the skip-gram model predicts the probability distribution of the words that appear around the target word.

The skip-gram model consists of an input layer, an embedding layer, and an output layer. The input layer takes a one-hot encoded vector representing the target word as input. This input vector is then multiplied with the weight matrix of the embedding layer to produce the dense word vector representation. The output layer uses this dense vector representation to predict the probability distribution of the context words. The hidden layer contains the learned word embeddings and continuous, low-dimensional vectors representing each word in the vocabulary.

The objective of the skip-gram model is to maximize the probability of predicting the context words given the target word. This can be achieved by minimizing the negative log-likelihood of the predicted context words.

The skip-gram model can also be trained using SGD to update the model parameters iteratively by backpropagating the error signal from the output layer to the hidden layer.

The softmax-based approach used to calculate the probability of all possible words in the vocabulary can become computationally infeasible when the vocabulary is very large. In addition, for rare words, the softmax-based approach can result in poor model performance, as the model may not have sufficient training examples to learn their representations accurately. Negative sampling [57] is a technique used to address these challenges. Instead of trying to predict the probability of each possible word in the vocabulary as the output of the model, negative sampling only samples a small number of negative examples to train the model. The idea behind negative sampling is to randomly select a few words that are not the actual context of the target word and consider them as negative samples. The number of words considered during training is reduced and the model is enabled to focus on learning the difference between positive and negative examples, rather than trying to learn the probabilities of all possible words.

The learned word embeddings in the hidden layer of the model can be used as word representations for downstream NLP tasks [9]. These embeddings have been shown to capture semantic and syntactic relationships between words, such as word similarity (e.g., ”dog” and ”cat” are more similar than ”dog” and ”computer”) and analogies. One of the most famous examples (”man” is to ”king” as ”woman” is to ”queen”) depicting the analogical prowess of word2vec is shown in figure 2.1. It has been reproduced from [1].

In practice, both the CBOW and skip-gram models are used to learn word embeddings. The CBOW model is generally faster to train and works better for frequent words, while the Skip-gram model is better suited for infrequent words and capturing more nuanced relationships between words. However, both of them have been shown to suffer from gender biasing [11].

Not only are the word embeddings learned on text, but also on other various kinds of data. Airbnb considered the click stream of a user as a sentence and trained a skip-gram model on millions of those click sessions and achieved around 20% increase in the similar listings click-through rate (CTR) [33]. Alibaba created a graph on the click-streams of a user and created many such clickstreams by doing random walks on these graphs to power their recommendation system [100]. ASOS has successfully used embeddings to predict their customer lifetime value [16].

2.2.5 GloVe

GloVe (Global Vectors for Word Representation) is another popular method for learning word embeddings, developed by Pennington et al. [74] that combines the advantages of both count-based methods like TF-IDF and prediction-based methods like word2vec. GloVe aims to learn word embeddings

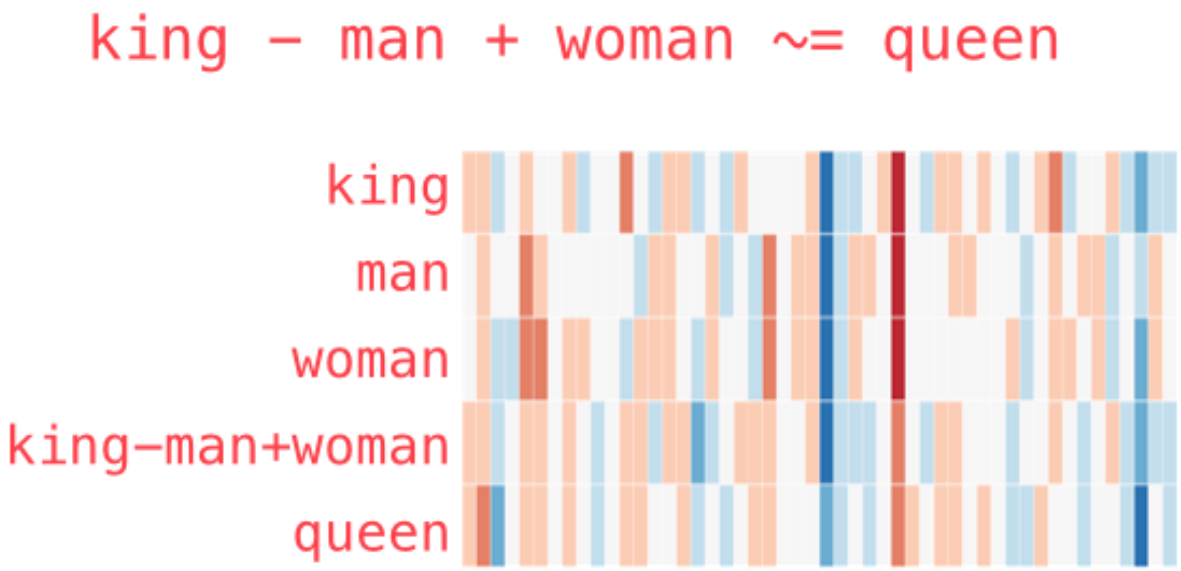

Figure 2.1 The vector obtained from the operation ”king-man+woman” is not exactly the same as the vector for ”queen,” but ”queen” is the word that most closely matches it in the vocabulary.

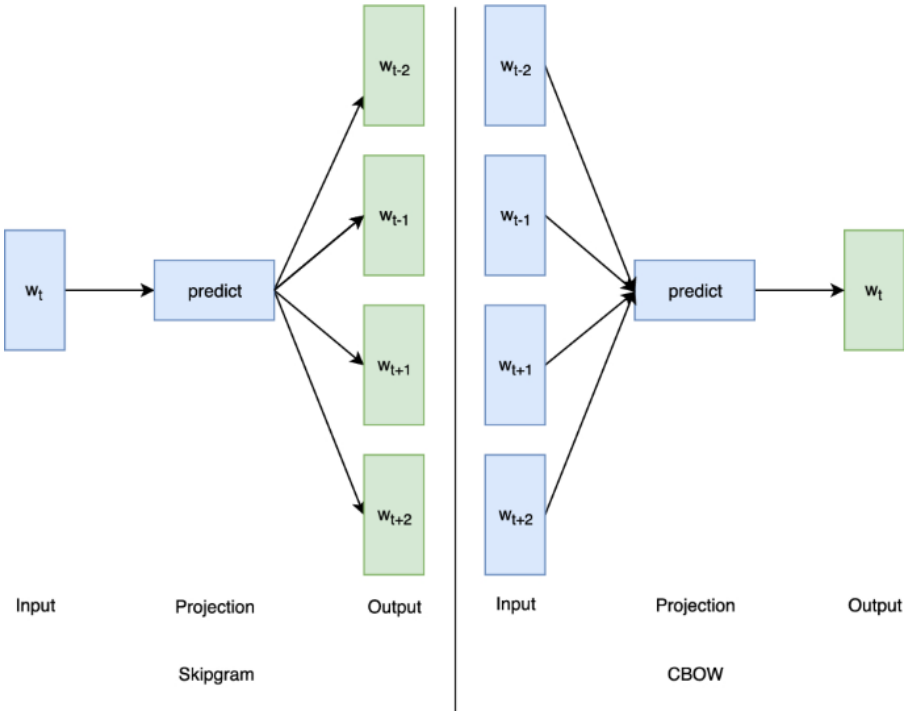

Figure 2.2 High-level overview of the difference between the CBOW and skip-gram models.

by modelling the co-occurrence statistics of words in a corpus, focusing on capturing the global context of words.

The basic idea of GloVe is to use matrix factorization techniques to decompose the co-occurrence matrix of words into two lower-dimensional matrices, one for the word embeddings and one for the context embeddings. The elements of the resulting word and context embeddings are learned by minimizing a weighted least squares objective function, with weights that are determined based on the frequency of co-occurrence of words.

More specifically, the objective of GloVe is to learn a set of word embeddings and a set of context embeddings such that the dot product between them gives the logarithm of the co-occurrence probability of the corresponding word-context pairs.

The GloVe model is then trained using SGD to update the word and context embeddings iteratively by backpropagating the error signal from the objective function.

One of the main advantages of GloVe over word2vec is that it takes into account the global co-occurrence statistics of words, which helps to capture semantic relationships that are not easily captured by local context windows. Another advantage is that GloVe is less sensitive to the choice of hyperparameters compared to word2vec, which can make it easier to train and tune. However, one limitation of GloVe is that it is computationally more expensive than word2vec since it requires the computation of the co-occurrence matrix for all pairs of words in the vocabulary.

How did these distributed word vectors solve the problems in n-gram Language Models?

- Sparsity
  We have dense vector representations. The size of the vectors is no more the size of the vocabulary but a very small number such as 256 or 512 compared to the size of the vocabulary.

- Storage
  Each word is represented using a single vector. Hence storage is linear in the order of vocabulary in the case of word vectors, and not exponential, which is the case for n-grams.

Now let's move on to more advanced ways of language modeling and word representations.

2.3 RNN and Variants

Recurrent Neural Network (RNN) is a neural network architecture specifically designed to handle sequential data, such as text or speech. RNNs have a unique ability to capture the temporal dynamics of a sequence, making them a powerful tool for tasks such as language modeling, machine translation [94], and sentiment analysis.

The learning task of an RNN for language modeling is to predict the probability distribution of the next word in a sequence given the previous words. The RNN takes a sequence of word embeddings as input and produces a hidden state vector at each time step that captures the information about the

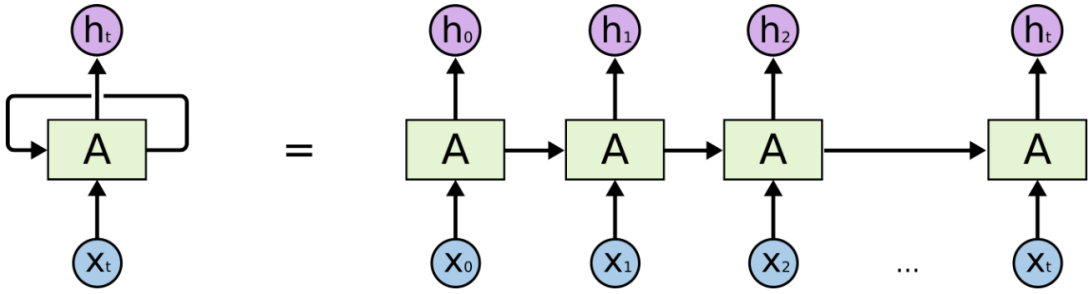

Figure 2.3 An unrolled recurrent neural network. Reproduced from [65].

previous words in the sequence. The output of the RNN at each time step is then passed through a softmax function to generate a probability distribution over the vocabulary of possible next words.

The hidden state h_t is updated at each time step t by combining the previous hidden state $h_{t\ 1}$, the current input x_t, and biases b_h using matrix multiplication and the tanh activation function. The output y_t is then computed from the current hidden state h_t using matrix multiplication, the softmax activation function, and bias b_y.

One of the main advantages of RNNs over methods like word2vec or GloVe is that they can capture more complex relationships between words, such as long-term dependencies or context that spans across sentences. RNNs can also be used to generate text, by feeding in a starting sentence and using the RNN to generate the next word in the sequence, and so on.

2.3.1 Challenges with RNNs and the Vanishing Gradient Problem

RNNs are powerful tools for processing sequential data, but they also come with several challenges. Training RNNs can be computationally expensive due to the need to process sequential data in a time-dependent manner.

One of the main challenges is the vanishing gradient problem, which occurs when the gradients of the loss function with respect to the weights of the network become very small as they propagate through time, leading to slow or even no learning. The gradients of the log-likelihood objective with respect to the network parameters can be computed using backpropagation through time (BPTT), which involves recursively applying the chain rule to compute the gradients at each time step. However, the gradients can become very small as they are backpropagated, especially when the network is deep, and the gradients are multiplied by the same weight matrix at each time step. This makes it difficult for the optimizer to update the parameters effectively and can result in slow convergence or even stagnation of the training.

Additionally, the exploding gradient problem can arise during training when the gradients of the loss function with respect to the weights become too large and cause the weights to update significantly,

leading to instability in the training process. The exploding gradient problem can also lead to numerical overflow errors, where the values of the gradients or weights become too large to be represented by the computer's finite numerical precision.

Another challenge is the difficulty in capturing long-term dependencies in the data, as the hidden state of the network may become saturated or forget important information over time. These challenges have led to the development of better RNN architectures, such as LSTMs and GRUs, which are better suited for capturing long-term dependencies and mitigating the vanishing and exploding gradient problems.

2.3.2 LSTM and GRU

Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) are two types of specialized RNN architectures that address the problems of long-term dependencies and vanishing gradients by introducing gating mechanisms.

2.3.2.1 LSTM

LSTM is a type of RNN architecture that was introduced by Hochreiter et al. [37]. LSTM introduces a memory cell and three gating mechanisms: input gate, forget gate, and output gate. The memory cell can selectively add or remove information to or from it, and the three gates control the flow of information into and out of the memory cell.

The forget gate determines which information to discard from the previous time step's memory cell, and the input gate decides which information to add to the current memory cell. The output gate decides which information to output from the memory cell to the next layer or output.

2.3.2.2 GRU

GRU is a newer type of RNN architecture that was introduced by Chung et al. [19]. GRU uses two gating mechanisms: reset gate and update gate. The reset gate determines how to combine the new input with the previous memory, and the update gate decides how much of the previous memory to keep for the current time step.

Figure 2.4 gives a high-level overview of LSTM and GRU, compared to RNN. Adapted from [65].

Solving The Vanishing Gradient Problem:

The key idea behind both LSTM and GRU [20] is to selectively update and forget information over time using specialized units called gates. The gates act as a sort of filter that controls the flow of information through the network, allowing important information to pass through while preventing irrelevant or noisy information from affecting the output. By allowing the network to selectively update and forget information based on the input and previous hidden state, LSTMs and GRUs can solve the vanishing gradient problem by preventing the gradients from becoming too small or too large during backpropa-

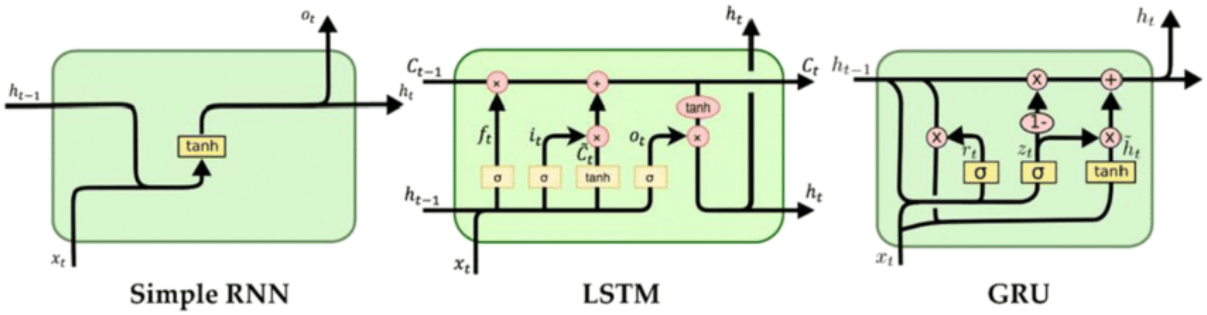

Figure 2.4 Complexities of LSTM and GRU, compared to a vanilla RNN.

gation. This enables the network to effectively learn long-term dependencies and maintain the relevance of past inputs.

2.3.2.3 Challenges with LSTM and GRU

Despite effectively capturing long-term dependencies, LSTMs or GRUs still have major limitations. One major challenge is the difficulty in capturing context-dependent relations between different parts of a sequence. For example, when predicting the next word in a sentence, it may be necessary to look back at different parts of the sentence to capture the relevant context. They rely on the hidden state to capture this context, but the information in the hidden state may not be distributed in a way that enables effective context-dependent processing.

Another major issue is their reliance on recurrent connections to process sequential data. While recurrent connections allow LSTMs and GRUs to capture dependencies between adjacent time steps, they struggle to capture long-term dependencies in the data, as the information passed from one timestep to the next can become diluted or lost over time. This can lead to difficulties in modeling complex sequential data. Additionally, the recurrent nature of LSTMs can make them computationally expensive to train, as each time step must be processed sequentially. These challenges have led to the development of alternative mechanisms and architectures, such as the Attention Mechanism and the Transformers, that use self-attention mechanisms to capture long-term dependencies and process sequential data more efficiently.

2.3.3 The Attention Mechanism

Attention mechanism [5] was developed as a solution to the challenges of LSTMs in capturing long-term dependencies in sequential data. The primary challenge with LSTMs is that they rely on recurrent connections, which can struggle to propagate information over long time intervals, leading to poor performance on tasks requiring long-term memory.

Attention mechanisms were developed to address this challenge by allowing the model to selectively focus on relevant parts of the input sequence at each time step, rather than relying solely on the hidden

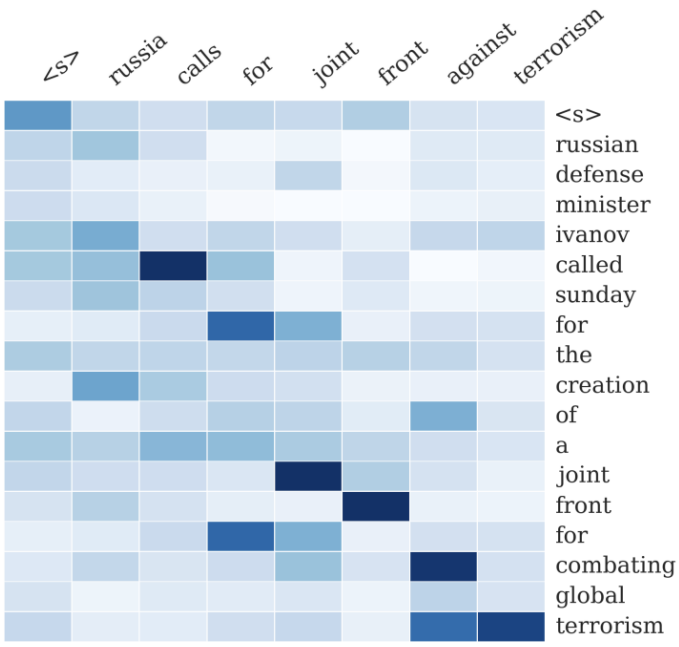

Figure 2.5 "Example output of the attention-based summarization system. The heatmap represents a soft alignment between the input (right) and the generated summary (top). The columns represent the distribution over the input after generating each word." Reproduced from [86].

state from the previous time step. In other words, it enables the network to weigh the importance of different parts of the input when generating the output.

The use of attention mechanisms has been shown to improve the performance of LSTMs and GRUs on a variety of tasks involving long sequences of data, such as machine translation and speech recognition. However, incorporating attention mechanisms into LSTMs and GRUs can also make the models more complex and computationally expensive to train, presenting additional challenges.

However, still one of the main challenges with recurrent-styled architectures is their parallelizability. Recurrent-styled architectures process sequences one element at a time and maintain an internal state that summarizes the previous elements in the sequence. Because the output of each element depends on the previous elements, it is difficult to parallelize the computation across different elements in the sequence. This means that they are typically slower to train and evaluate.

The attention mechanism has been shown to be highly effective for various NLP tasks. Figure 2.5 depicts one such use in summarization. Attention mechanism has also been successfully incorporated in Transformers, which is discussed in the next section.

2.4 Transformers

The Transformer model, introduced in 2017 by Vaswani et al. [99], is one of the most revolutionary ideas in the field of NLP and has become the de facto standard for many state-of-the-art models.

The models that we discussed previously had quite a few limitations, such as the difficulty of parallelizing computation and the inability to capture long-term dependencies. Transformers address these limitations by using an attention mechanism that allows the model to focus on relevant parts of the input sequence, enabling it to capture dependencies between distant tokens.

Following are the key ideas that made transformers what they are today:

- Self-attention

  Self-attention is a mechanism in the Transformer that allows the model to attend to different parts of the input sequence to compute a representation for each token. It allows the model to weigh the importance of different parts of the input sequence when making predictions. Self-attention allows the model to capture long-range dependencies between tokens and has been critical to the success of Transformers.

  In self-attention, the input sequence is transformed into three different representations: queries, keys, and values. The queries, keys, and values are all vectors that are learned by the model during training. The queries are used to compute a score for each key, which represents the relevance of that key to the current query. The scores are then used to weigh the corresponding values, which are combined to form a context vector that represents the importance of each position in the input sequence.

- Multi-head attention

  Multi-head attention is an extension of self-attention that allows the model to attend to different representation subspaces at different positions using multiple parallel attention mechanisms. This allows the model to capture different aspects of the input sequence and has been shown to improve performance on NLP tasks.

  In multi-head attention, the queries, keys, and values are linearly projected, say, h times using different learned weight matrices. Each set of projected queries, keys, and values is then processed by a separate self-attention mechanism, resulting in h different output sequences. The output sequences are concatenated and linearly transformed again to produce the final output sequence.

- Positional encoding

  Positional encoding is used to inject positional information into the input sequence of a model. This is important because self-attention does not encode the position of the tokens, and positional encoding enables the model to distinguish between tokens in different positions.

  The basic idea behind positional encoding is to add a fixed vector to the embedding of each token in the sequence. The positional encoding is designed to represent the position of the token in

the sequence by encoding information about its relative position to other tokens. The positional encoding is added element-wise to the token embedding, and the resulting vector is used as the input to the self-attention layer. To design the positional encoding vector, the approach used is to use a combination of sine and cosine functions with different frequencies which allow the model to learn different patterns of relative position information at different scales.

- Encoder-decoder architecture

  The Transformer uses an encoder-decoder architecture to generate outputs for NLP tasks. The encoder processes the input sequence, while the decoder generates the output sequence.

  The encoder takes an input sequence of tokens and generates a sequence of contextualized embeddings, which capture the meaning of the input sequence. The encoder comprises a stack of N identical layers, where each layer has two sub-layers: a multi-head self-attention layer and a position-wise feed-forward layer. The input sequence is first embedded into a continuous vector space using an embedding layer and then fed into the encoder layers. Each encoder layer takes the previous layer's output as input and generates a new output, which is passed to the next layer.

  On the other hand, the decoder takes the output sequence generated by the encoder and generates a sequence of tokens. The decoder is also composed of a stack of N identical layers, where each layer has three sub-layers: a masked multi-head self-attention layer, a multi-head attention layer, and a position-wise feed-forward layer. The masked multi-head self-attention layer allows the decoder to attend to all previously generated tokens, but not future tokens, to ensure that the model does not cheat by looking at future tokens when generating a token.

Figure 2.6 shows the architecture of the transformer model including the components we discussed above. The figure has been reproduced from the original paper [99].

The attention mechanism in transformers enables the model to focus on the most relevant parts of the input sequence when making predictions. By using multiple parallel attention mechanisms, the model can attend to different subspaces of the input sequence simultaneously, allowing it to capture more complex patterns in the data. Additionally, the attention mechanism allows the model to handle variable-length input sequences without the need for expensive recurrent computations, making it more efficient than traditional RNNs.

The model is trained to predict the most probable sequence of output words given an input sequence of words. This requires the model to capture both short-term dependencies between adjacent words and long-term dependencies between distant words, which is achieved using self-attention.

Additionally, the self-supervised pre-training paradigm used in transformers has made it possible to transfer knowledge across different NLP tasks, significantly reducing the amount of labeled data required for training. This has made NLP more accessible to researchers and has enabled the development of applications that were previously not possible.

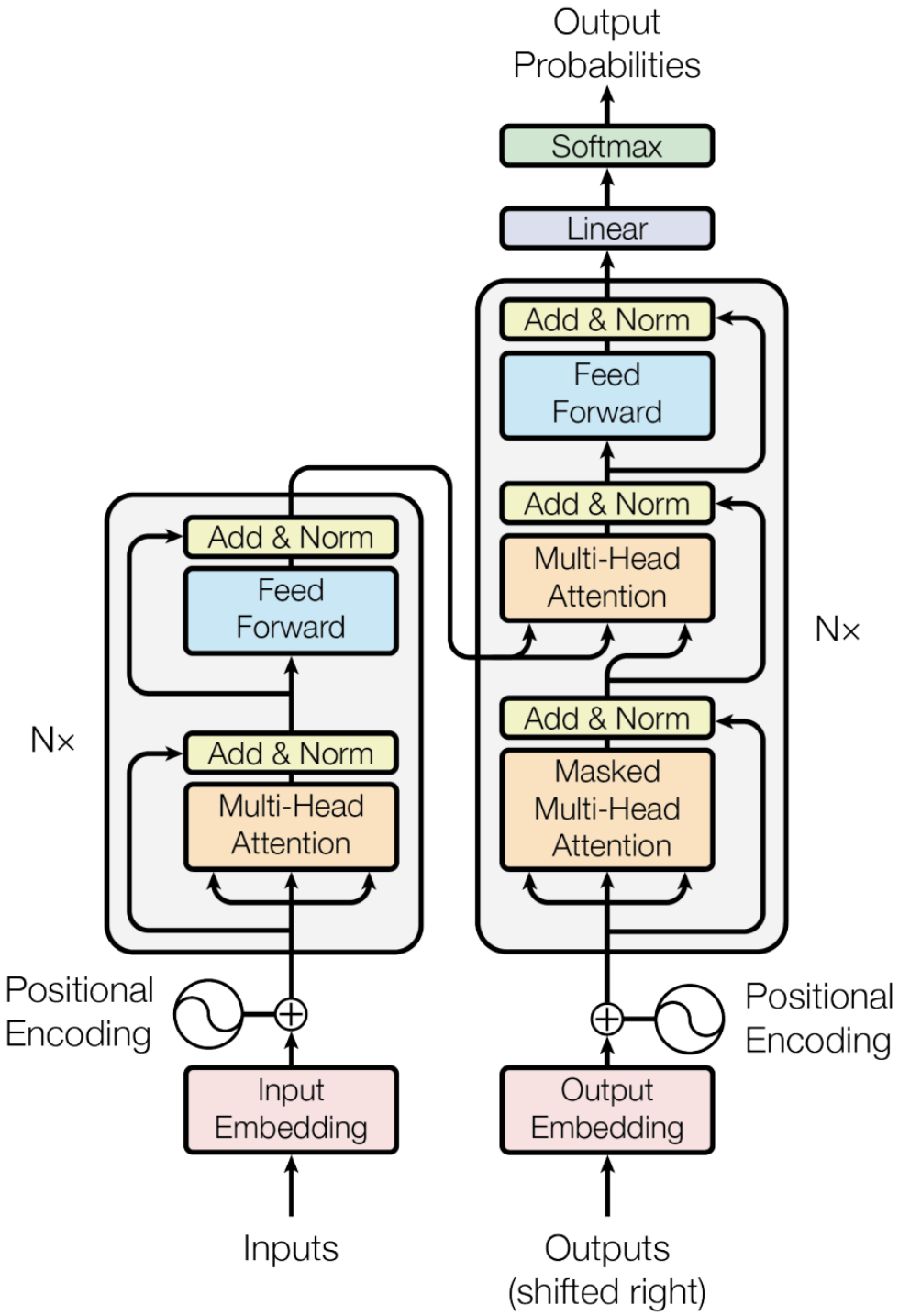

Figure 2.6 The Transformer model architecture.

Overall, the impact of transformers on NLP has been enormous, enabling the development of powerful and efficient models that can learn from large amounts of unstructured text data and achieve state-of-the-art performance on a wide range of tasks.

2.4.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a state-of-the-art pre-trained language model based on the transformer encoder [26]. It learns contextualized word embeddings and has achieved state-of-the-art results on many tasks such as question-answering, sentiment analysis, and named entity recognition.

BERT is pre-trained using two unsupervised tasks:

- **Masked Language Model (MLM):**

  BERT randomly masks some of the tokens in the input text and trains the model to predict the original masked tokens based on the context provided by the other tokens in the sentence. This task is similar to a fill-in-the-blank exercise.

- **Next Sentence Prediction (NSP)**

  BERT is trained on a binary classification task where the model predicts whether two input sentences are adjacent to each other or not in the original text.

The combination of these two tasks allows BERT to learn rich contextualized representations of words that are useful for a wide range of downstream tasks.

The self-attention layers are modified to perform bidirectional attention, allowing the model to use information from both left and right contexts when encoding a word. This is in contrast to the original Transformer architecture, which is unidirectional.

BERT is trained using a large corpus of text data, typically on the order of billions of words. The resulting model contains a large number of parameters, typically in the range of tens to hundreds of millions.

The input to BERT is a sequence of tokens, where each token is a word or a subword. The tokens are then padded or truncated to a fixed length, and special tokens [CLS] and [SEP] are inserted at the beginning and end of the input sequence, respectively.

The [CLS] token represents the classification token, which is used as the first token of the input sequence. This token is used to represent the entire input sequence in the classification task. The [SEP] token is used as a separator between two segments of the input sequence. In the case of sentence classification or question answering, the input sequence is divided into two segments: the first segment contains the input text, and the second segment contains the question or prompt.

The output of the BERT model is a sequence of hidden states for each token in the input sequence. These hidden states are generated by passing the input sequence through multiple layers of self-attention and feed-forward networks. The final hidden state corresponding to the [CLS] token is used as the

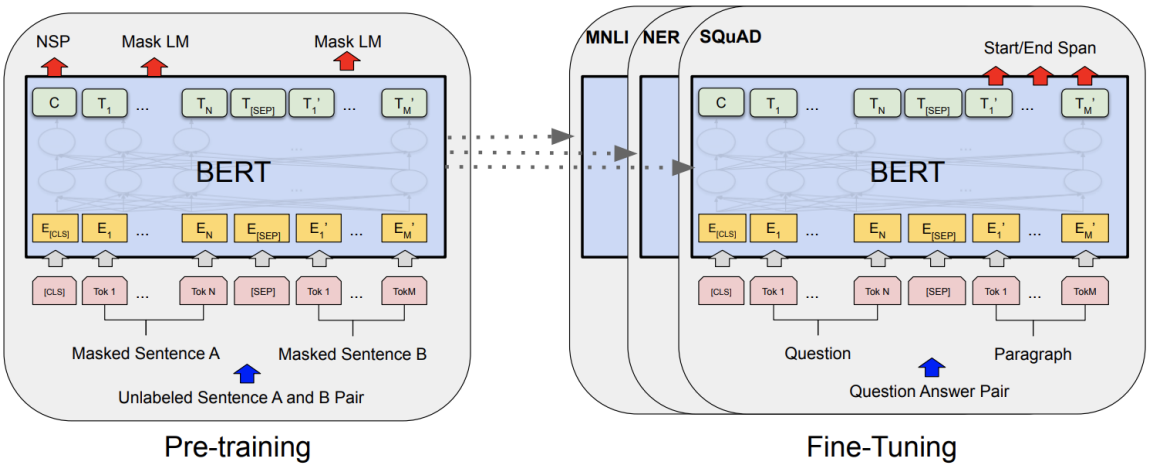

Figure 2.7 Pre-training and fine-tuning procedures for BERT.

representation of the input sequence for classification tasks. For sequence labeling tasks, such as named entity recognition or part-of-speech tagging, the hidden states for each token are used as representations.

Figure 2.7 shows the architecture of BERT including the components we discussed above. The figure has been reproduced from [26].

Overall, BERT has been a major breakthrough in the field of NLP, and its key ideas of pre-training with masked language modeling, fine-tuning for downstream tasks, sentence-level and token-level representations, and large-scale pre-training have influenced the development of many subsequent language models.

2.4.2 GPT

Generative Pre-trained Transformer (GPT) is a type of language model developed by OpenAI [79], which uses an autoregressive approach to generate text. It is based on the decoder side of the transformer.

GPT was initially introduced in 2018 with the GPT-1 model [78] and later updated with GPT-2 and GPT-3, which have shown impressive performance on various language tasks. The emergence of ChatGPT, which is based on GPT 3.5, has caused a lot of excitement and anticipation globally. Similar chatbots such as YouChat, Google's Bard, and Bing AI based on ChatGPT are transforming how we search the web.

The focus of this thesis is on Selective Masking, which is related to BERT and the encoder side of the transformer. Hence, we won't dive into the details of the architecture of GPT.

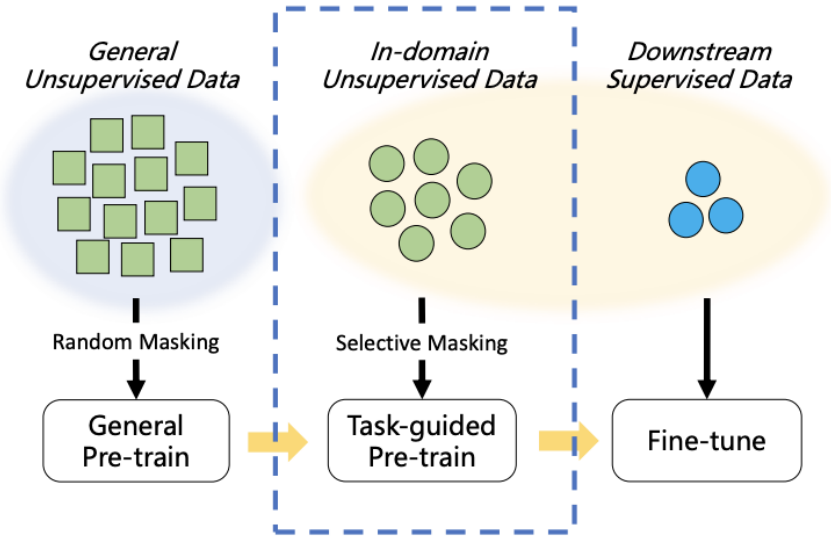

Figure 2.8 "The overall three-stage framework as described in [34]. They add task-guided pre-training between general pre-training and fine-tuning to efficiently and effectively learn the domain-specific and task-specific language patterns."

2.5 Selective Masking

2.5.1 What is Selective Masking?

In traditional pre-training methods, the input text is randomly masked by replacing certain tokens with a special mask token, which forces the model to predict the masked tokens based on the context of the surrounding tokens. However, this approach has limitations in that it can lead to the model memorizing the masked tokens and producing outputs that are too similar to the input.

Selective masking addresses this issue by masking only a subset of the tokens in the input text that are most relevant to the pre-training task. This is done by identifying key phrases or entities in the text and masking only those while leaving the other tokens unchanged.

For example, in a pre-training task focused on named entity recognition, the model might selectively mask only the entities in the text, such as people's names or locations, while leaving other words unchanged. This forces the model to focus its attention on the important parts of the input, rather than potentially memorizing irrelevant or unimportant tokens.

Gu et al. [34] suggest selective masking to learn task-specific patterns. They argue that certain tokens carry more weight and relevance for specific tasks and that these tokens may differ across different tasks. Thus, they suggest a masking strategy that selectively masks these important tokens. Specifically, they introduce task-guided selective masking pre-training, which is performed between the general pre-

training and fine-tuning stages, to learn the domain-specific and task-specific language patterns efficiently. However, this strategy requires access to a small in-domain dataset during pre-training, which may not always be available for some tasks.

2.5.2 Why is Selective Masking beneficial?

Pre-trained language models have been a driving force behind the progress in many NLP tasks [106, 8, 49]. Most of these works follow the paradigm of unsupervised pre-training on large general-domain corpora followed by fine-tuning for downstream tasks, following the success of BERT [27].

BERT uses the Cloze task [96] for its masked language modeling (MLM) pre-training procedure, where 15% of input tokens are randomly masked from each sequence. However, alternative pre-training tasks have been proposed to address the inefficiencies of random masking. Clark et al. [21] suggest a "replaced token detection" pre-training task, while Gu et al. [34] propose a masked language model with selective masking.

While the pre-training and fine-tuning paradigm has yielded remarkable results, Gururangan et al. [35] have demonstrated that even large LMs struggle to capture the complexity of a single textual domain. Thus, they suggest domain-specific pre-training on task-relevant corpora for better downstream performance in that domain. Other works show improvements by pre-training BERT-like models on huge in-domain corpora relevant to the downstream tasks [8, 15, 38, 14, 61].

To address this issue, we propose a novel way to tailor the BERT model for the downstream task via task-specific masking before the standard supervised fine-tuning. We take a pre-trained BERT model and tailor it for downstream tasks using task-specific selective masking on a small chunk of BookCorpus [113]. This tailored model is then fine-tuned for downstream tasks. Our method includes a novel approach to find tokens important for the downstream task using a list of seed words relevant to the downstream task. The word list and word embeddings are used to compute a 'task' score for each word, which is used to calculate the masking probability of the word. We experiment with different masking functions and show considerable value in such selective masking.

Unlike previous approaches, we use selective masking to train our model and only require word-lists instead of in-domain data. Since BERT also uses BookCorpus, we do not use any new corpus for training. Collecting a word list is an easier alternative when in-domain datasets are scarce. Our methodology is effective and generalizable, with experimental results showing improved performance in downstream tasks such as sentiment analysis, hate speech classification, formality detection, and NER on informal text.

Chapter 3

Selective Masking for Binary Text Classification Tasks

3.1 Introduction

Text classification is a supervised machine learning task that involves categorizing a piece of text into one or more predefined classes or categories based on its content. It is used to classify text according to its sentiment (positive or negative), topic (politics, sports, technology), or any other criteria. Text classification is also referred to as text categorization.

Text classification can be either binary (two classes) or multi-class (multiple classes). Binary text classification involves assigning a text to one of two classes, such as positive or negative sentiment. For example, in sentiment analysis, the goal is to classify a given text as either positive or negative. In spam detection, the goal is to classify an email or message as either spam or not spam. Binary text classification is also used in tasks such as hate speech detection, fake news detection, formality analysis, and gender classification. In multi-class text classification, the text can be assigned to one of the multiple classes. In this chapter, we will discuss binary text classification. We will talk about multi-class text classification in the next chapter.

Most of the techniques for binary text classification involve some or the variation of the following steps:

- Data preprocessing

  In this step, the raw text data is cleaned, normalized, and transformed into a structured format that can be used for classification. This can involve removing stopwords, stemming or lemmatizing the text, and converting the text into numerical vectors.

- Feature extraction

  In this step, relevant features are extracted from the preprocessed text data, which can be used for classification. We have discussed a few relevant techniques in the previous chapter such as bag-of-words, TF-IDF, and word embeddings.

- Model training

In this step, a model is trained on the preprocessed and feature-extracted text data using a binary classification objective function such as binary cross-entropy. Common techniques used in the olden days include logistic regression, support vector machines (SVMs), and Naive Bayes. Nowadays, transformer-based architectures are very common and popular. Transformers can capture long-range dependencies, contextual information, and semantic relations between words, which makes them particularly effective for classification tasks.

- Model evaluation

  In this step, the trained model is evaluated on a separate test set to measure its performance. Common evaluation metrics for binary text classification include accuracy, precision, recall, and F1-score.

  In order to obtain good results, it is important to choose an appropriate feature representation and algorithm for the task. Furthermore, it is important to ensure that the data used for training and testing is representative of the data that the model will be used on in production. Finally, it is also important to evaluate the model's performance on a test set to ensure that it is able to accurately classify the text documents.

3.2 Text Classification Tasks

3.2.1 Sentiment Analysis

Sentiment Analysis aims to identify and extract the sentiment or emotion expressed in a piece of text. It is a classification task that involves determining the sentiment of a given text as either positive or negative (and sometimes as neutral). This task is often used to assess the sentiment of reviews, comments, and other types of text. The task consists of assigning a sentiment label to each text document. This can be done manually or by using an automated algorithm.

One of the challenges in sentiment analysis is the complexity and nuance of language. Text can express a range of emotions and attitudes, and even seemingly straightforward expressions can have different meanings depending on the context and tone. Additionally, language can be ambiguous, sarcastic, or figurative, making it difficult for algorithms to accurately identify the sentiment.

Despite the challenges, sentiment analysis has many practical applications, such as social media monitoring, brand reputation management, customer feedback analysis, and market research. It can provide valuable insights into how people perceive and react to products, services, and events, and help organizations make data-driven decisions.

3.2.1.1 Previous Works

The history of sentiment analysis can be traced back to the early 2000s when researchers started exploring the use of ML and NLP techniques to automatically classify the sentiment of text [68, 47].

One of the seminal works in this field is Turney and Littman's paper "Measuring praise and criticism: Inference of semantic orientation from association" [97]. In this paper, the authors proposed a method for automatically classifying the sentiment of text based on the semantic orientation of words.

Another important work in sentiment analysis is Pang and Lee's paper "Opinion mining and sentiment analysis" [67]. This paper provides an overview of the state-of-the-art techniques for sentiment analysis at the time, including rule-based methods, machine-learning methods, and hybrid approaches. The authors also introduced a Polarity Dataset of movie reviews that had become a standard benchmark dataset for sentiment analysis research then.

In recent years, deep learning models have become the dominant approach for sentiment analysis. One of the most influential papers in this area is Kim's "Convolutional neural networks for sentence classification" [41]. This paper proposed a CNN architecture for sentence classification that achieved state-of-the-art results on several benchmark datasets for sentiment analysis.

In addition to these seminal works, there has been a wealth of research on sentiment analysis in recent years [39, 105, 58, 103], including work on aspect-based sentiment analysis [108], sentiment analysis in social media [42], and sentiment analysis in multilingual and cross-lingual settings [6].

3.2.1.2 Datasets

For the task of sentiment analysis, we use the following datasets:

- Amazon review dataset

  The Amazon review dataset [36] consists of product reviews from Amazon.com. The dataset includes reviews for a variety of product categories, each with a star rating (1-5) and review text.

  The dataset contains reviews such as "Great book for travelling Europe: I currently live in Europe, and this is the book I recommend for my visitors. It covers many countries, colour pictures, and is a nice starter for before you go, and once you are there." and "Dont like it: This product smells when you open the package and it appears one of the gel cushions leaks so I spent $10 and havent worn then. I wouldnt buy this product again. Dont waste your money on something you may not be able to use."

- Yelp dataset

  The Yelp dataset [109] consists of reviews of businesses from the Yelp website, labeled as positive or negative. The dataset includes reviews for restaurants, bars, and other businesses, each with a star rating (1-5) and the review text.

  Amazon review and Yelp datasets are multi-class classifications, while the movie review dataset is a binary classification. While we talk about multi-class classification in the next chapter, we are keeping the results of the Amazon review and Yelp datasets in this chapter to maintain synchronization.

The Yelp dataset contains reviews such as ”Very decent fried chicken” and ”This place is terrible. The food was cold and tasted awful, and the service was rude and unhelpful.”

- Movie reviews dataset

  The movie reviews dataset [66] is also known as the Polarity Dataset. It consists of positive and negative movie reviews, manually annotated for sentiment polarity.

  The movie reviews dataset contains movie reviews such as ”an engaging overview of johnson’s eccentric career. ” and ”interesting, but not compelling.”

These examples illustrate the different sentiments that can be expressed in reviews, and demonstrate the types of data that are included in these datasets.

Dataset	**Train**	**Test**
Amazon Review	40,000	5,000
Yelp	40,000	5,000
Movie Review	8,534	2,128

Table 3.1 Dataset Statistics: Sentiment Analysis

3.2.2 Hate Speech Detection

Hate speech detection involves using various techniques to automatically identify and classify text that expresses hostility or prejudice towards individuals or groups based on certain protected characteristics, such as race, ethnicity, religion, gender, sexual orientation, and disability. The goal of hate speech detection is to automatically flag and remove such content from online platforms and social media, and to prevent its spread.

One of the major challenges in detecting hate speech is the diversity and complexity of language. Hate speech can take many forms, including direct and indirect, explicit and implicit, and coded or subtle language, making it difficult to define and identify. Additionally, hate speech can evolve and adapt to changes in social norms and context.

Despite the challenges, hate speech detection has many potential applications, such as online moderation, content filtering, and hate crime prevention. It can help platforms and organizations identify and remove harmful content, and protect vulnerable communities from harassment and discrimination.

3.2.2.1 Previous Works

There have been several influential works in the field of hate speech detection [28, 91, 23]. One important early paper was ”Hateful symbols or hateful people? Predictive features for hate speech detection on Twitter” by Davidson et al [95]. This paper proposed a machine learning approach for hate

speech detection on Twitter, and introduced a dataset of annotated tweets that contained hate speech or offensive language. The authors identified several features that were predictive of hate speech, including the use of certain words and phrases, and the presence of certain symbols and emojis.

Another important paper in the field is "Detecting Hate Speech on Twitter Using a Convolution-GRU Based Deep Neural Network" by Zhang et al. [111]. This paper proposed a deep learning approach for hate speech detection, using a combination of convolutional and gated recurrent neural networks. The authors evaluated their model on a dataset of tweets annotated for hate speech, and achieved high accuracy and F1-score.

"Deep Learning for Hate Speech Detection in Tweets" by Badjatiya et al. [4] is another important paper in the field of hate speech detection. In this paper, the authors proposed a deep learning approach for hate speech detection in tweets, using a combination of convolutional neural networks (CNNs) and gated recurrent units (GRUs). They also introduced a new dataset of tweets annotated for hate speech, and evaluated their model on this dataset and several other existing datasets.

In addition to these influential works, there has been a wealth of research on hate speech detection in recent years including [60, 75, 52, 90].

3.2.2.2 Datasets

For the task of hate speech detection, we use data from Gab and Reddit [76] in a binary classification setup. Platforms like Gab and Reddit have faced criticism for not doing enough to address hate speech on their platforms. While they have policies and guidelines that prohibit hate speech and harassment, enforcement can be inconsistent and subjective. Automated systems may miss certain forms of hate speech or produce false positives, while human moderators may have biases and make mistakes.

The datasets were manually annotated by human judges with labels that indicate whether they contain hate speech or not, and if so, what type of hate speech (e.g. racism, sexism, homophobia).

The Gab data contains samples such as "i joined gab to remind myself how retarded jew haters are. You wouldn't be typing on your abacus without them you retard." which resulted in this comment: "Using words that insult one group while defending another group doesn't come across as helpful, You can make the same point more effectively without the use of hateful terminology. Use of the r-word is unacceptable in our discourse as it demeans and insults people with mental disabilities."

The Reddit data contains samples such as "Gotta love that cultural diversity huh liberals? In an ironic twist, dumb fucking hillbilly white people like yourself have become the country's lowest common denominator. The most amusing part is you're too stupid to even hide it anymore. Wait how do you know his race and location ?" which resulted in this feedback from the moderation team: "Its inappropriate to use that attack one based on their race and nationality such as 'dumb fucking hillbilly white people'. Insulting someone because of their race is hate speech. Do not use that speech here."

Dataset	Train	Test
Gab	28,776	5,000
Reddit	17,314	5,000

Table 3.2 Dataset Statistics: Hate Speech Detection

3.2.3 Formality Style Detection

Formality style detection, also known as formality analysis, aims to automatically determine the level of formality in a given text. Formality is a linguistic feature that refers to the degree of adherence to a set of conventions or standards, such as those related to grammar, vocabulary, and style, that are typically associated with formal contexts, such as academic writing, legal documents, or business correspondence. In contrast, informal language is characterized by a more relaxed, casual, and conversational style, and is typically used in informal settings, such as social media, texting, or personal communication. For our work, we make the simplifying but intuitive assumption that there can be only two degrees of formalism: formal and informal.

One of the challenges in formality detection is defining and measuring formality. Formality can be influenced by various factors, such as the audience, the purpose, the domain, the genre, and the style of the text. Additionally, formality can be subjective and context-dependent, making it difficult to establish a universal standard.

Despite the challenges, formality detection has many potential applications, such as improving machine translation and text-to-speech synthesis systems, detecting plagiarism, automated proofreading, style checking, text simplification, and analyzing social media content. It can help writers and editors improve the readability and accessibility of their texts, and assist non-native speakers and people with cognitive disabilities in understanding complex information. In machine translation, the level of formality of the source text can influence the choice of appropriate target language style, while in text classification, formality can be a useful feature to distinguish between different genres or domains of text.

3.2.3.1 Previous Works

Formality style detection is a challenging NLP task that has received significant attention in recent years. In this section, we will review some of the related works that have been published on this topic.

One of the earliest works in this field is the paper ”Automatic Detection of Text Genre” by Kessler et al. [40] which presents an approach for automatically detecting the genre of text based on a set of linguistic features. This work is relevant to formality detection, as genre is often correlated with formality, and the features used for genre detection can be adapted for formality detection.

Another important work is paper ”An empirical analysis of formality in online communication” by Pavlick et al. [70]. The study analyzed online communication to investigate how formality varies by

context, user demographics, and social status. The authors found that formality was influenced by the topic, platform, and user characteristics. The study highlights the need for nuanced formality detection.

In recent years, machine learning algorithms have been used extensively for formality detection [92, 72]. Abubakar et al. [98] compared the performance of several machine learning algorithms for classifying the formality and sentiment of tweets, and analyzed the influence of feature selection and preprocessing.

More recently, deep learning models have been applied to formality detection. For example, the paper by Banarescu et al. [7] proposed a neural network architecture that uses character-level embeddings and a gated recurrent unit to predict the formality of text. The model was evaluated on a dataset of academic abstracts and achieved state-of-the-art performance.

In spite of significant progress in recent years [110, 93, 12], the task of formality detection remains challenging, particularly in cases where the formality of text is ambiguous or context-dependent.

3.2.3.2 Datasets

For the task of formality style detection, we use the following datasets:

- GYAFC dataset

  The GYAFC (Grammarly's Yahoo Answers Formality Corpus) dataset [81] is a collection of text data from Yahoo Answers, a question-and-answer platform where users can ask and answer questions on a wide range of topics. The dataset contains over 20 million question and answer pairs, with each pair consisting of a "formal" version and an "informal" version of the same question or answer.

  One unique aspect of the GYAFC dataset is that it includes a wide range of topics and domains, including science, history, cooking, and relationships. This makes it a useful resource for studying formality across different subject areas.

  The dataset contains examples such as "Gotta see both sides of the story" (informal), "You have to consider both sides of the story." (formal), and "I'd say it is punk though." (informal), "However, I do believe it to be punk" (formal).

- OC data

  The OC (Online Communication) dataset [71] was created after analyzing a large dataset of messages from a range of online forums and social media platforms to investigate how formality varies across different contexts and user demographics. To ensure that the dataset was representative of online communication as a whole, the authors sampled messages from a diverse range of contexts, including discussions of politics, technology, sports, and entertainment. They also included messages from users of different ages, genders, and geographic locations.

  Some examples from the OC data include formal sentences such as "I will contact Eli this week concerning the necessary processes and I look forward to talking with you personally soon."

and "However, I understand political realities and I know that I must contact you concerning his replacement", and informal sentences such as "You guessed it - a speeding driver" and "So , my family is doing their best to keep the test scores up".

Dataset	Train	Test
GYAFC - Entertainment	105,190	2,832
GYAFC - Music	103,934	2,664
OC	8,730	1,400

Table 3.3 Dataset Statistics: Formality Detection

3.3 Methodology and Experiments

3.3.1 Task Specific Masking

In this section, we first describe the method to calculate the task score of the word. Following that, we describe different masking functions that use this task score to assign a probability to each word. The word is then masked with this probability.

3.3.1.1 Calculating Task Specific Score of a Word

We leverage the classification framework in [63], who calculated the formality score of a word for machine translation. We begin with a set of seed words indicative of different classes in downstream tasks. In a binary classification task like sentiment analysis, we choose negative-sentiment and positive-sentiment words as two sets of word-lists.

Following [63], an SVM model is trained by assigning scores of 0 and 10 for the two sets of words, and a separating hyperplane is learned between Word2Vec [56] vector space representations of the two classes of seed words. [63] reported the best performance using an SVM model with Word2Vec representations. We did some experiments with GloVe embeddings [73] and SVM as well but had to revert to word2vec as it gave better results. We persist with using SVM as our classification model due to the following reasons:

- We have a small set of seed words. When the data is fewer, SVMs work better as compared to neural networks because neural networks have many more parameters than SVMs, which also result in longer training times.
- The distance from the hyperplane also serves as a good measurement for getting the masking probability, as described in the next subsection.

Once the SVM model is trained, Euclidean distance to this hyperplane is used to measure the given word's task-specific score.

Finding an effective word list is important. The word list chosen should represent the fine-tuning dataset. It would be an added bonus if a small sample of words could cover a large part of the dataset. Fortunately, a good set of seed words for many classification tasks can simply be found on the web which can be curated with minimum effort to represent the dataset.

3.3.1.2 Masking Functions

As described earlier, our approach uses a masking function to compute the probability of masking a given word based on its task score (obtained from the SVM model described earlier). In the downstream task of binary classification, then we mask both the extremes. For example, if the task is sentiment analysis, the language model is required to understand both extremes, viz. positive sentiments and negative sentiments. So we assign a high probability to both extremes.

In the following functions, s is the euclidean distance of the word from the SVM hyperplane. α, β, and γ are constants that were adjusted to make sure that approximately 15% of the total tokens are masked.

Masking Function 1: Step Function

$$f(s) = \begin{cases} 1 & s \quad \alpha \; or \; s \quad 10 \quad \alpha \\ 0 & otherwise \end{cases}$$

This function only masks words at the extreme and completely discards other words. However, [46] has argued that the appearance of an opinion word in a sentence does not necessarily mean that the sentence expresses a positive or negative opinion. Similarly, [71] has argued that formality is not only expressed by the use of formal/informal words but also "neutral" words, punctuations, capitalization, paraphrasing, etc. So in the following functions, we assign a non-zero probability to non-extreme words.

Masking Function 2: Linear Function

$$f(s) = \frac{max(s, 10 \quad s) \quad \alpha}{\beta}$$

This function assigns each word a probability linearly proportional to its importance.

Masking Function 3: Concave Up (exponential)

$$f(s) = \frac{e^{\;:max(s;10 \quad s)} \quad e}{\gamma}$$

The exponential function glorifies the values which are towards the extreme and exponentially decreases the values as we move towards the center. This function assigns each word a probability exponentially proportional to its importance.

3.3.2 Experiments

We take a pre-trained BERT model (bert-base-uncased) released by [27] and further train it using different masking functions discussed above. We select α, β and, γ such that approximately 15% of the tokens are masked. Like the original BERT framework, we mask the chosen word 80% of the time, replace it with a random word 10% of the time, and leave it unchanged 10% of the time. The original BERT model uses WordPiece tokenization [102] and masks only tokens. Later whole word masking was introduced, where all the tokens associated with a word are masked. Since our task demands whole word masking, and for a fair comparison, we compare our results with both variants of random masking (whole word masking (WWM) and token masking (TM))

We use a small chunk (100 Mb) of BookCorpus [113] and train the BERT model on it for 20k steps using the above masking strategy. For a fair comparison, we also pre-train another BERT on this corpus using random masking (both token masking and whole word masking). To compare our results with [34], we also train BERT on the Amazon Reviews dataset using our selective masking and finetune it on the Movie Reviews dataset (discussed in the previous sections of this chapter). We test our approach on a variety of downstream tasks. We finetune for ten epochs and report the best result.

Tables 3.1, 3.2, and 3.3 provide statistics about each of these datasets. Note that we report the combined test and validation dataset as our test dataset.

For the sentiment analysis task, we use the word list provided by [48]. We had 1,856 words in each class. It is also possible to use various sentiment lexicons such as AFINN [62] or SentiWordNet [3].

Masking Function	Amazon	Yelp	MR	MR+A.
Random (TM)	59.46	62.96	85.66	87.43
Random (WWM)	59.79	62.64	85.47	87.05
MF 1	60.20	63.20	85.66	87.52
MF 2	**61.42**	63.42	86.58	88.34
MF 3	60.53	**64.40**	**87.60**	**89.65**

Table 3.4 Results on the downstream task of Sentiment Analysis. We report classification accuracy in percentage. MR+A. means Movie Reviews with Selective Masking on Amazon Reviews data, and M.F. means Masking Function.

For hate speech detection, we used the list of hate speech words from here[1]. For non-hate words, we used POS-tagging to get positive adjectives and adverbs. We also collect neutral words from Wikipedia. There were 426 words in each class.

For formality style detection, we used the list provided by Julian Brooke[2] along with words collected from the internet to get a final list with 620 words in each class.

[1] https://www.cs.cmu.edu/~biglou/resources/bad-words.txt
[2] http://www.cs.toronto.edu/~jbrooke/Formality_Word_Lists.zip

Masking Function	Gab	Reddit
Random (TM)	92.22	88.92
Random (WWM)	92.26	87.88
MF 1	92.22	88.44
MF 2	93.14	**89.99**
MF 3	**93.20**	89.34

Table 3.5 Results on the downstream task of Hate Speech Detection. We report classification accuracy in percentage. M.F. means Masking Function.

Masking Function	OC	Family	Entertainment
Random (TM)	85.21	87.88	88.03
Random (WWM)	84.86	88.25	87.92
MF 1	85.43	87.95	87.01
MF 2	86.57	**89.18**	**89.10**
MF 3	**87.43**	89.03	89.07

Table 3.6 Results on the downstream task of Formality Style Detection. We report classification accuracy in percentage. M.F. means Masking Function.

3.4 Results and Discussion

Tables 3.4, 3.5, and 3.6 show the results for different downstream tasks. We see that tailoring the BERT model using selective masking methodology helps and it outperforms BERT's random masking for all the tasks. Due to architectural limitations, we could not train BERT from scratch using the proposed methodology or try Gu et al. [34] like three-stage training. However, with just 20k steps, we are able to achieve improvements over random masking, showing the strength of the approach.

We notice that no single masking function performs the best for all the tasks. Masking Functions 2 and 3 outperform Random Masking almost all the time. Masking Function 3 achieves the best results more than 50% of the time, suggesting the superiority of the exponential functions in capturing the relationship between masking probability and score. For no task, Masking Function 1 performed the best. This confirms the hypothesis of Lui et al. [46] and Pavlick et al. [71]. They argued that sentiments or formality does not always depend on extreme words, and we show it experimentally.

The gain in performance is more significant for smaller datasets (training sample size $<$ 10k) than larger datasets. Also, for smaller datasets, exponential functions provide the best results. These observations suggest two things - First, exponential functions better capture the relationship between task and masking probability for smaller datasets. Secondly, it is known that the more the data, the better the performance. However, when supervised downstream data is scarce, the model cannot always capture the domain-specific and task-specific patterns. To this end, our approach of unsupervised training using word lists and embeddings helps in capturing the domain-specific and task-specific patterns. Which masking function will perform the best depends upon the choice of the word list and downstream task.

Comparing our results with Gu et al. [34], they report a gain of 2.14% (compared to the BERT model) on the Movie Reviews dataset using selective masking and Amazon reviews dataset (in-domain data). We have achieved a gain of 1.94% (compared to the BERT model) in the accuracy using selective masking alone. When we use the Amazon Reviews dataset (in-domain data), we achieve a gain of 3.99% (compared to the BERT model). Thus our strategy is effective.

Chapter 4

Selective Masking for Multi-class Text Classification Tasks

4.1 Introduction

In the previous chapter, we looked at some binary text classification tasks. But all text classification tasks need not be binary. In this chapter, we will look at the other end of the spectrum: multi-class text classification tasks.

Multi-class text classification task involve classifying a given text into one of several predefined classes. In multi-class classification, there are more than two possible classes, and each document or text sample can be assigned to only one class.

Some common examples of multi-class text classification tasks include sentiment analysis (including the neutral label), named-entity recognition (NER), topic classification, language detection, and spam filtering. For example, topic classification is the task of assigning a document to a specific topic or category, such as politics, sports, or entertainment. NER involves identifying and classifying named entities in text, such as people, organizations, locations, dates, and so on.

Previously, multi-class text classification was performed using a variety of ML algorithms, such as logistic regression, decision trees, and SVMs. But today, deep learning and transformer architectures have taken over. Typically, the text data is first preprocessed and transformed into a numerical format using techniques such as tokenization, stemming, and vectorization. Then, a model is trained on a labeled dataset using the transformed text data and corresponding class labels.

Multi-class text classification tasks have a wide range of applications in areas for e.g. social media analysis, customer feedback analysis, news categorization, and content classification, just to name a few.

4.2 Related Works

Several works have been proposed to tackle the problem of multi-class classification. Multi-class classification is generally predominant for image problems rather than text and a lot of research has been done to solve multi-class image classification problems. But the works cited here are generalizable and can be applied to text classification problems. A few works have proposed a multi-class classification

method based on a combination of multiple binary classifiers [32, 50]. Mayoraz et al. proposed a multi-class classification method based on SVMs [54]. Noria et al. proposed a genetic algorithm-based feature selection approach for multi-class classification [64], which aimed to identify the most relevant features for classification. Finally, Yang et al. proposed a multi-class classification method based on a hierarchical attention network [107].

While there exist various multi-class text classification tasks, in this work, we specifically look at the problem of named-entity recognition. We already looked at sentiment analysis in the previous chapter.

4.3 Named-Entity Recognition

NER (Named-Entity Recognition) involves identifying and extracting named entities from text. Named entities are specific types of words or phrases that refer to real-world objects, such as people, places, organizations, dates, times, products, and other entities.

One of the challenges in NER is the ambiguity and variability of named entities. Named entities can have different forms, spellings, and aliases, and they can refer to different entities depending on the context and the domain. Additionally, named entities can be nested, overlapping, or composed of multiple parts, making it difficult to identify and classify them accurately.

Another challenge is the diversity and complexity of languages. Named entities can have different linguistic and cultural patterns and conventions, and they can vary in their frequency and salience in different languages and contexts. Additionally, languages can have different writing systems, morphology, and syntax, which can affect the performance of NER models.

Despite these challenges, NER has many practical applications, such as information extraction, document classification, and sentiment analysis. It can help analysts and researchers extract and organize valuable information from unstructured text, assist in various tasks such as news summarization, chatbots, and virtual assistants, and filter out resumes for finding apt candidates for a job role.

4.3.1 Previous Works

There are various approaches to NER, including rule-based approaches and ML/DL-based approaches. Rule-based approaches rely on a set of predefined rules that are designed to identify named entities based on their characteristics, such as capitalization, context, and position in a sentence. Since these characteristics vary from language to language, almost all the rule-based approaches are specific to a single language. A few works that use rule-based approaches, all specific to a single language, are [30, 83, 17].

Approaches based on machine learning, deep learning, and neural networks, on the other hand, use statistical models trained on labeled data to automatically identify and classify named entities. These models can work for multiple languages if appropriate data is provided to them. This is the area that most research is focused on in recent times. One of the works introduced a novel neural network architecture that combined bidirectional LSTMs and CNNs to achieve state-of-the-art performance in NER

tasks [18]. A work used Conditional Random Fields (CRFs) to do NER [69]. Another work extended the bidirectional LSTM-CNN architecture by adding a CRF layer, resulting in further performance improvements [51].

There also exists a seminal survey paper: "Neural Architectures for Named Entity Recognition" by Lample et al. [43]: This paper presented a survey of different neural network architectures for NER, including feedforward networks, recurrent networks, and convolutional networks.

There also exist hybrid approaches using the advantages of both rule-based and neural network approaches [85, 88]. One work focused on NER in social media texts, specifically tweets, and proposed a hybrid approach that combined rule-based and machine learning-based methods to achieve high accuracy [84].

A different direction is working on low-resource NER [112, 31]. One work proposes a neural network approach for low-resource NER that leverages cross-lingual transfer learning and character-level neural conditional random fields [22].

4.3.2 Datasets

For the task of NER on informal text, we use text from social media as a proxy to informal text.
We use the following datasets:

- Broad Twitter Corpus (referred as BTC)
  The Broad Twitter Corpus (BTC) [24] is a collection of tweets that have been gathered from various locations, time periods, and social contexts. Its purpose is to capture a diverse range of Twitter activities and language usage.

  Some examples from the dataset include: "I'm at StarbucksUK in London, Greater London" (Location: London), "Watching #DemDebate with BernieSanders supporters in #Boise #FeelTheBern" (Person: Bernie Sanders), and "Can't believe I'm going to see Beyonce live tonight at Wembley Stadium!!! #FormationWorldTour" (Person: Beyonce).

- WNUT-17
  The WNUT-17 [25] corpus is a collection of tweets that includes novel and unfamiliar entities within emerging conversations. The dataset comprises multiple tweets that have been labeled with ten different entity types, such as people, places, companies, products, artistic works, groups, fictional characters, events, and others.

  "RT kylegriffin1: Obama releases statement on Paris Agreement." (Location: Paris; Creative Work: Paris Agreement), "I liked a YouTube video from davidpakmanshow. Did Russia Hack Election for Trump?" (Corporation: YouTube; Person: davidpakmanshow; Location: Russia; Person: Trump), and "RT ComplexMusic: .Drake brought out 1future last night in London for their final show. #BoyMeetsWorldTour" (Person: Drake; Person: 1future; Location: London; Event: BoyMeetsWorldTour).

Dataset	Train	Test
BTC	5,551	4,000
WNUT-17	3,394	2,296

Table 4.1 Dataset Statistics: Named-Entity Recognition

4.4 Methodology and Experiments

4.4.1 Task Specific Masking

In this section, we first describe the method to calculate the task score of the word. Following that, we describe different masking functions that use this task score to assign a probability to each word. The word is then masked with this probability.

4.4.1.1 Calculating Task Specific Score of a Word

We leverage the classification framework in Niu et al. [63], who calculated the formality score of a word for machine translation. We begin with a set of seed words indicative of different classes in downstream tasks. For NER on the domain of informal texts: here our first list represents in-domain vocabulary and contains words usually found in informal texts, and our second list contains words not found in that domain, and thus here, we use words found in the formal texts.

Following Niu et al. [63], an SVM model is trained by assigning scores of 0 and 10 for the two sets of words, and a separating hyperplane is learned between Word2Vec [56] vector space representations of the two classes of seed words. Niu et al. [63] reported the best performance using an SVM model with Word2Vec representations. Once the SVM model is trained, Euclidean distance to this hyperplane is used to measure the given word's task-specific score.

4.4.1.2 Masking Functions

As described earlier, our approach uses a masking function to compute the probability of masking a given word based on its task score obtained from the SVM model.

In the following functions, s is the euclidean distance of the word from the SVM hyperplane. α, β, and γ are constants that were adjusted to make sure that approximately 15% of the total tokens are masked. These are the same functions used for binary classification. Copying them here for quick reference. If the task requires the model to learn only one extreme, like NER on the informal dataset, then in that case, we assign a high probability to only one extreme (i.e., the one representing informality).

Masking Function 1: Step Function

$$f(s) = \begin{cases} 1 & s \quad \alpha \; or \; s \quad 10 \quad \alpha \\ 0 & otherwise \end{cases}$$

Masking Function 2: Linear Function

$$f(s) = \frac{max(s, 10 \quad s) \quad \alpha}{\beta}$$

Masking Function 3: Concave Up (exponential)

$$f(s) = \frac{e^{\;:max(s;10 \quad s)} \quad e}{\gamma}$$

4.4.2 Experiments

We take a pre-trained BERT model (bert-base-uncased) released by [27] and further train it using different masking functions discussed above. We select α, β and, γ such that approximately 15% of the tokens are masked. Like the original BERT framework, we mask the chosen word 80% of the time, replace it with a random word 10% of the time, and leave it unchanged 10% of the time. The original BERT model uses WordPiece tokenization [102] and masks only tokens. Later whole word masking was introduced, where all the tokens associated with a word are masked. Since our task demands whole word masking, and for a fair comparison, we compare our results with both variants of random masking (whole word masking (WWM) and token masking (TM))

We use a small chunk (100 Mb) of BookCorpus [113] and train the BERT model on it for 20k steps using the above masking strategy. For a fair comparison, we also pre-train another BERT on this corpus using random masking (both token masking and whole word masking). We finetune for ten epochs and report the best result.

Table 4.1 provides statistics about the datasets used. Note that we report the combined test and validation dataset as our test dataset. For NER, we used the same list as Formality detection[1].

4.5 Results and Discussion

Table 4.2 shows the results for different downstream tasks. We see that tailoring the BERT model using selective masking methodology helps and it outperforms BERT's random masking for both the datasets.

Due to architectural limitations, we could not train BERT from scratch using the proposed methodology or try Gu et al. [34] like three-stage training. However, with just 20k steps, we are able to achieve improvements over random masking, showing the strength of the approach.

[1] http://www.cs.toronto.edu/~jbrooke/Formality_Word_Lists.zip

Masking Function	BTC	WNUT-17
Random (TM)	49.99	18.80
Random (WWM)	49.93	19.25
MF 1	48.33	20.71
MF 2	50.26	21.04
MF 3	**51.78**	**21.96**

Table 4.2 Results on the downstream task of Named-Entity Recognition. We report F1-score. M.F. means Masking Function.

Masking Function 3 achieves the best results for both the datasets, suggesting the superiority of the exponential functions in capturing the relationship between masking probability and score. For no dataset, Masking Function 1 performed the best.

The gain in performance is more significant for smaller datasets (training sample size $<$ 10k) than larger datasets. Also, for smaller datasets, exponential functions provide the best results. These observations suggest two things - First, exponential functions better capture the relationship between task and masking probability for smaller datasets. Secondly, it is known that the more the data better the performance. However, when supervised downstream data is scarce, the model cannot always capture the domain-specific and task-specific patterns. To this end, our approach of unsupervised training using word lists and embeddings helps in capturing the domain-specific and task-specific patterns. Which masking function will perform the best depends upon the choice of the word list and downstream task.

Gu et al. [34] cannot work for non-binary classification tasks. We showed that our approach works for NER as well and thus is more generalizable.

Chapter 5

What's so important about the word list?

5.1 Introduction

Ever wondered why are we making a big deal about the word list? We have been frequently talking about the word list in the previous two chapters.

Finding an effective word list is important. The word list chosen should represent the fine-tuning dataset. It would be an added bonus if a small sample of words could cover a large part of the dataset.

Fortunately, a good set of seed words for many classification tasks can simply be found on the web which can be curated with minimum effort to represent the dataset. But what if you cannot find a suitable word list? We wondered that too, and we designed an experiment to test the actual importance of our word lists. This can be treated like a *Proof of Concept* of our methodology.

We chose a task on which we hadn't experimented before: Humor Detection. It is a binary text classification task. We also found a dataset that is generally used for Humor Detection. We also found the word list. Actually no, we found *a word list*. Upon close inspection, you will find that the word list doesn't actually correspond with the dataset chosen. There's no synchronization amongst the words present in the word list and the words present in the dataset.

Let's dive into the details!

5.2 Humor Detection

Humor detection is the process of using various algorithms to automatically identify and classify text that contains humor or is intended to be humorous.

One of the challenges in humor detection is defining and measuring humor. Humor can take many forms, such as puns, sarcasm, irony, satire, and wit, making it difficult to establish a universal standard. Additionally, humor can be influenced by various factors, such as the audience, the purpose, the domain, the genre, and the style of the text.

Another challenge is the lack of annotated data for training and evaluating models. Humor detection requires large and diverse datasets that are annotated with labels of humor and non-humor, but such datasets may be difficult to obtain and may have subjective and cultural biases.

Despite these challenges, humor detection has many potential applications, such as content recommendation, advertising, and entertainment. It can help platforms and organizations understand the preferences and emotions of their users, and create more engaging and memorable content.

5.2.1 Previous Works

There have been many works related to humor detection. Some examples include:

- Rule-based approaches

  These involve using a set of hand-crafted rules to identify humor in text [82, 87, 80]. For example, one rule might be that puns are often used in jokes.

- Statistical approaches

  These involve using machine learning algorithms to train a model to identify humor [104, 10, 101]. This typically involves providing the model with a dataset of labeled examples (i.e. text labeled as either humorous or not humorous) and having it learn to make predictions based on features of the text.

- Neural network approaches

  These are a type of statistical approach that involve training a neural network to identify humor [59, 13, 53]. This typically involves using a form of deep learning, such as a recurrent neural network, a convolutional neural network, or more recently, the transformers.

- Hybrid approaches

  These involve combining multiple methods, such as rule-based and statistical approaches, to improve performance. One of the works describes a system for automatically detecting humorous captions in the New Yorker's cartoon caption contest [77].

5.3 Dataset

For the task of **humor detection**, we use the dataset by I. Annamoradnejad [2]. To also test whether the gain in results is due to the actual selective masking strategy and not any other reason, we deliberately use a word list whose words mismatch with the type of words present in the dataset for the humor detection task. We use the word list provided by Engelthaler et al. [29] to get 400 words in each class.

Refer to figure 5.1 to get an idea of the kind of words present in the word list. The words present in the word list are the words commonly used in off-color humor (dark sexist, racial, etc. jokes).

In contrast, take a look at a few example sentences from the fine-tuning dataset:

Positive extreme	*Negative extreme*
Booty (4.32)	Rape (1.18)
Tit (4.25)	Torture (1.26)
Booby (4.13)	Torment (1.3)
Hooter (4.13)	Gunshot (1.31)
Nitwit (4.03)	Death (1.32)
Twit (4)	Nightmare (1.33)
Waddle (4)	War (1.33)
Tinkle (3.94)	Trauma (1.35)
Bebop (3.93)	Rapist (1.37)
Egghead (3.92)	Distrust (1.38)
Ass (3.92)	Deathbed (1.39)
Twerp (3.92)	Pain (1.39)

Figure 5.1 A sample list of words present in the word list. Reproduced from [29].

- ”I didn’t text you just to exercise my fingers, i was expecting a reply back...”
- ”Youtube ads youtube can insta load a commercial but my 2 minute video takes 10 minutes to buffer.”
- ”Why do we need to be learned english? hmm.. couldn’t have worded that better myself, luke”
- ”Do you know how to avoid click bait? obviously not...”

The fine-tuning dataset consists of short informal text with no vulgarity, completely opposite of the kind of words present in the word list. We expect that there will be no significant difference in results using selective masking as compared to random masking.

Dataset	**Train**	**Test**
ColBERT: Humor	100,000	10,000

Table 5.1 Dataset Statistics: Humor Detection

5.4 Methodology and Experiments

5.4.1 Task Specific Masking

In this section, we first describe the method to calculate the task score of the word. Following that, we describe different masking functions that use this task score to assign a probability to each word. The word is then masked with this probability.

5.4.1.1 Calculating Task Specific Score of a Word

We leverage the classification framework in [63], who calculated the formality score of a word for machine translation. We begin with a set of seed words indicative of different classes in downstream tasks.

Following [63], an SVM model is trained by assigning scores of 0 and 10 for the two sets of words, and a separating hyperplane is learned between Word2Vec [56] vector space representations of the two classes of seed words. [63] reported the best performance using an SVM model with Word2Vec representations. Once the SVM model is trained, Euclidean distance to this hyperplane is used to measure the given word's task-specific score.

5.4.1.2 Masking Functions

As described earlier, our approach uses a masking function to compute the probability of masking a given word based on its task score obtained from the SVM model.

In the following functions, s is the euclidean distance of the word from the SVM hyperplane. α, β, and γ are constants that were adjusted to make sure that approximately 15% of the total tokens are masked. These are the same functions used in Chapter 3. Copying them here for quick reference. If the downstream task is classification, then we mask both the extremes. So we assign a high probability to both extremes.

Masking Function 1: Step Function

$$f(s) = \begin{cases} 1 & s \quad \alpha \; or \; s \quad 10 \quad \alpha \\ 0 & otherwise \end{cases}$$

Masking Function 2: Linear Function

$$f(s) = \frac{max(s, 10 \quad s) \quad \alpha}{\beta}$$

Masking Function 3: Concave Up (exponential)

$$f(s) = \frac{e^{\;:max(s;10 \quad s)} \quad e}{\gamma}$$

5.4.2 Experiments

We take a pre-trained BERT model (bert-base-uncased) released by [27] and further train it using different masking functions discussed above. We select α, β and, γ such that approximately 15% of the tokens are masked. Like the original BERT framework, we mask the chosen word 80% of the time, replace it with a random word 10% of the time, and leave it unchanged 10% of the time. The original BERT model uses WordPiece tokenization [102] and masks only tokens. Later whole word masking was introduced, where all the tokens associated with a word are masked. Since our task demands whole word masking, and for a fair comparison, we compare our results with both variants of random masking (whole word masking (WWM) and token masking (TM))

We use a small chunk (100 Mb) of BookCorpus [113] and train the BERT model on it for 20k steps using the above masking strategy. For a fair comparison, we also pre-train another BERT on this corpus using random masking (both token masking and whole word masking). We finetune for ten epochs and report the best result.

Table 5.1 provide statistics about the dataset. Note that we report the combined test and validation dataset as our test dataset.

For humor detection, we used the word list provided by Engelthaler et al. [29] to get 400 words in each class.

5.5 Results and Findings

Masking Function	ColBERT
Random (TM)	**98.68**
Random (WWM)	98.45
MF 1	98.57
MF 2	98.64
MF 3	98.61

Table 5.2 Results on the downstream task of Humor Detection. We report classification accuracy in percentage. M.F. means Masking Function.

Table 5.2 shows the results for the Humor Detection task. As expected, there were no gains in results for Humor Detection because we did not use a suitable word list. Thus, this shows that if there is a mismatch in the words found in the word list and the downstream task, our strategy will provide no gains.

Chapter 6

Conclusion and Future Directions

6.1 Summary of the Main Findings

We developed a novel framework for customizing the popular BERT model for a wide range of downstream tasks through selective masking. Unlike the previous methods, which rely on in-domain data for training, our approach used readily available word lists. Furthermore, our approach is adaptable to non-binary classification tasks, increasing its applicability and versatility.

Our experimental results demonstrated the superiority of our selective masking approach over random masking, as it is more effective at identifying domain-specific and task-specific patterns. We tested our approach on several downstream tasks, including sentiment analysis, hate speech detection, formality style detection, and named-entity recognition, and achieved better results compared to existing methods.

Overall, our framework offers a more generalizable and efficient solution for tailoring BERT to various downstream tasks. This innovative framework holds significant promise for improving the performance of BERT in various other downstream tasks, thus contributing to the development of better NLP applications.

6.2 Limitations and Future Work

The approach presented in this work cannot be generalized to all Natural Language Understanding (NLU) tasks. It would be an interesting research problem to explore ways to incorporate selective masking for other NLU tasks such as question understanding or translation, where there are no clear boundaries for categorizing/scoring the relevance of the words for the task at hand.

The proposed approach involves the use of external knowledge, namely, word lists, in the pre-training of Language Models (LMs). Therefore, it is essential to evaluate the pre-trained LM's few-shot or zero-shot learning capabilities on downstream tasks. Additionally, it would be worthwhile to investigate whether this approach could reduce the amount of data required for training.

The methodology was primarily tested in English due to the abundance of data available in this language. However, one of the significant challenges is adapting this approach for Indian languages and creating a multilingual model that can perform equally well across multiple languages.

Overall, further research is needed to explore the generalizability of this approach to different NLU tasks, evaluate its ability to reduce the required amount of training data, and develop a multilingual model that can perform well across different languages.

Related Publications

- **Tanish Lad**, Himanshu Maheshwari, Shreyas Kottukkal, and Radhika Mamidi. "Using Selective Masking as a Bridge between Pre-training and Fine-tuning". In Proceedings of the 2nd Workshop on Efficient Natural Language and Speech Processing, New Orleans, United States. Neural Information Processing Systems, 2022. (NeurIPS 2022)

Bibliography

[1] J. Alammar. The illustrated transformer [blog post]. 2018. 11

[2] I. Annamoradnejad. Colbert: Using bert sentence embedding for humor detection, 2021. 44

[3] S. Baccianella, A. Esuli, and F. Sebastiani. SentiWordNet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, May 2010. European Language Resources Association (ELRA). 34

[4] P. Badjatiya, S. Gupta, M. Gupta, and V. Varma. Deep learning for hate speech detection in tweets. *CoRR*, abs/1706.00188, 2017. 29

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2014. 16

[6] A. Balahur and M. Turchi. Multilingual sentiment analysis using machine translation? In *Proceedings of the 3rd Workshop in Computational Approaches to Subjectivity and Sentiment Analysis*, pages 52–60, Jeju, Korea, July 2012. Association for Computational Linguistics. 27

[7] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider. Abstract meaning representation for sembanking. *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186, 01 2013. 31

[8] I. Beltagy, K. Lo, and A. Cohan. Scibert: A pretrained language model for scientific text, 2019. 24

[9] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155, mar 2003. 11

[10] K. Binsted, A. Nijholt, O. Stock, C. Strapparava, G. Ritchie, R. Manurung, H. Pain, A. Waller, and D. O'Mara. Computational humor. *Intelligent Systems, IEEE*, 21:59– 69, 04 2006. 44

[11] T. Bolukbasi, K.-W. Chang, J. Zou, V. Saligrama, and A. Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings, 2016. 11

[12] E. Briakou, D. Lu, K. Zhang, and J. Tetreault. Olá, bonjour, salve! XFORMAL: A benchmark for multilingual formality style transfer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3199–3216, Online, June 2021. Association for Computational Linguistics. 31

[13] V. Chahar, R. Walia, and S. Sharma. Deephumor: a novel deep learning framework for humor detection. *Multimedia Tools and Applications*, 81:1–16, 05 2022. 44

[14] S. Chakraborty, E. Bisong, S. Bhatt, T. Wagner, R. Elliott, and F. Mosconi. BioMedBERT: A pre-trained biomedical language model for QA and IR. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 669–679, Barcelona, Spain (Online), Dec. 2020. International Committee on Computational Linguistics. 24

[15] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, N. Aletras, and I. Androutsopoulos. LEGAL-BERT: The muppets straight out of law school. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2898–2904, Online, Nov. 2020. Association for Computational Linguistics. 24

[16] B. Chamberlain, Cardoso, C. Liu, R. Pagliari, and M. Deisenroth. Customer lifetime value prediction using embeddings. pages 1753–1762, 08 2017. 11

[17] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, and S. Vaithyanathan. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1002–1012, 2010. 38

[18] J. P. Chiu and E. Nichols. Named entity recognition with bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016. 39

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 15

[20] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. 15

[21] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. Electra: Pre-training text encoders as discriminators rather than generators, 2020. 24

[22] R. Cotterell and K. Duh. Low-resource named entity recognition with cross-lingual, character-level neural conditional random fields. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 91–96, Taipei, Taiwan, Nov. 2017. Asian Federation of Natural Language Processing. 39

[23] T. Davidson, D. Warmsley, M. Macy, and I. Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the international AAAI conference on web and social media*, volume 11, pages 512–515, 2017. 28

[24] L. Derczynski, K. Bontcheva, and I. Roberts. Broad Twitter corpus: A diverse named entity recognition resource. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1169–1179, Osaka, Japan, Dec. 2016. The COLING 2016 Organizing Committee. 39

[25] L. Derczynski, E. Nichols, M. van Erp, and N. Limsopatham. Results of the WNUT2017 shared task on novel and emerging entity recognition. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 140–147, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. 39

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019. 21, 22

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. 24, 34, 41, 47

[28] N. Djuric, J. Zhou, R. Morris, M. Grbovic, V. Radosavljevic, and N. Bhamidipati. Hate speech detection with comment embeddings. In *Proceedings of the 24th international conference on world wide web*, pages 29–30, 2015. 28

[29] T. Engelthaler and T. T. Hills. Humor norms for 4,997 english words. *Behavior Research Methods*, 50(3):1116–1124, Jun 2018. ix, 44, 45, 47

[30] D. Farmakiotou, V. Karkaletsis, J. Koutsias, G. Sigletos, and C. Spyropoulos. Rule-based named entity recognition for greek financial texts. 02 1970. 38

[31] X. Feng, X. Feng, B. Qin, Z. Feng, and T. Liu. Improving low resource named entity recognition using cross-lingual knowledge transfer. In *IJCAI*, volume 1, pages 4071–4077, 2018. 39

[32] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44(8):1761–1776, 2011. 38

[33] M. Grbovic and H. Cheng. Real-time personalization using embeddings for search ranking at airbnb. pages 311–320, 07 2018. 11

[34] Y. Gu, Z. Zhang, X. Wang, Z. Liu, and M. Sun. Train no evil: Selective masking for task-guided pre-training, 2020. ix, 23, 24, 34, 35, 36, 41, 42

[35] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics. 24

[36] R. He and J. J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. *CoRR*, abs/1602.01585, 2016. 27

[37] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 15

[38] K. Huang, A. Singh, S. Chen, E. Moseley, C.-Y. Deng, N. George, and C. Lindvall. Clinical XLNet: Modeling sequential clinical notes and predicting prolonged mechanical ventilation. In *Proceedings of the*

3rd Clinical Natural Language Processing Workshop, pages 94–100, Online, Nov. 2020. Association for Computational Linguistics. 24

[39] X. Ju, D. Zhang, R. Xiao, J. Li, S. Li, M. Zhang, and G. Zhou. Joint multi-modal aspect-sentiment analysis with auxiliary cross-modal relation detection. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4395–4405, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. 27

[40] B. Kessler, G. Nunberg, and H. Schuetze. Automatic detection of text genre. 08 1997. 30

[41] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. 27

[42] S. Kiritchenko and S. Mohammad. Examining gender and race bias in two hundred sentiment analysis systems. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 43–53, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. 27

[43] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360, 2016. 39

[44] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. 9

[45] O. Levy, Y. Goldberg, and I. Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015. 9

[46] B. Liu. *Sentiment analysis and subjectivity*, pages 627–666. 01 2010. 33, 35

[47] B. Liu. Sentiment analysis and opinion mining. volume 5, 05 2012. 26

[48] B. Liu, M. Hu, and J. Cheng. Opinion observer: analyzing and comparing opinions on the web. In A. Ellis and T. Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 342–351. ACM, 2005. 34

[49] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. 24

[50] A. Lorena, A. Carvalho, and J. Gama. A review on the combination of binary classifiers in multiclass problems. *Artificial Intelligence Review*, 30:19–37, 12 2008. 38

[51] X. Ma and E. H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354, 2016. 39

[52] S. MacAvaney, H.-R. Yao, E. Yang, K. Russell, N. Goharian, and O. Frieder. Hate speech detection: Challenges and solutions. *PloS one*, 14(8):e0221152, 2019. 29

[53] J. Mao and W. Liu. A bert-based approach for automatic humor detection and scoring. In *IberLEF@ SEPLN*, pages 197–202, 2019. 44

[54] E. Mayoraz and E. Alpaydin. Support vector machines for multi-class classification. In J. Mira and J. V. Sánchez-Andrés, editors, *Engineering Applications of Bio-Inspired Artificial Neural Networks*, pages 833–842, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. 38

[55] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013. 9

[56] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013. 32, 40, 46

[57] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc. 11

[58] K. Mishev, A. Gjorgjevikj, I. Vodenska, L. T. Chitkushev, and D. Trajanov. Evaluation of sentiment analysis in finance: from lexicons to transformers. *IEEE access*, 8:131662–131682, 2020. 27

[59] J. M. Moran, G. S. Wig, R. B. Adams Jr, P. Janata, and W. M. Kelley. Neural correlates of humor detection and appreciation. *Neuroimage*, 21(3):1055–1060, 2004. 44

[60] M. Mozafari, R. Farahbakhsh, and N. Crespi. A bert-based transfer learning approach for hate speech detection in online social media. In *Complex Networks and Their Applications VIII: Volume 1 Proceedings of the Eighth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2019 8*, pages 928–940. Springer, 2020. 29

[61] M. Müller, M. Salathé, and P. E. Kummervold. Covid-twitter-bert: A natural language processing model to analyse covid-19 content on twitter, 2020. 24

[62] F. A. Nielsen. Afinn, March 2011. 34

[63] X. Niu, M. Martindale, and M. Carpuat. A study of style in machine translation: Controlling the formality of machine translation output. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2814–2819, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics. 32, 40, 46

[64] B. Noria and Z. Elberrichi. Feature selection for text classification using genetic algorithms. pages 806–810, 11 2016. 38

[65] C. Olah. Understanding lstm networks [blog post]. 2015. ix, 14, 15

[66] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. 28

[67] B. Pang and L. Lee. Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, 2:1–135, 01 2008. 27

[68] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 79–86. Association for Computational Linguistics, July 2002. 26

[69] N. Patil, A. Patil, and B. Pawar. Named entity recognition using conditional random fields. *Procedia Computer Science*, 167:1181–1188, 2020. International Conference on Computational Intelligence and Data Science. 39

[70] E. Pavlick and J. Tetreault. An empirical analysis of formality in online communication. *Transactions of the Association for Computational Linguistics*, 4:61–74, 2016. 30

[71] E. Pavlick and J. Tetreault. An empirical analysis of formality in online communication. *Transactions of the Association for Computational Linguistics*, 4:61–74, 2016. 31, 33, 35

[72] P. Pawar and S. Gawande. A comparative study on different types of approaches to text categorization. *International Journal of Machine Learning and Computing*, pages 423–426, 01 2012. 31

[73] J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. 32

[74] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 11

[75] F. Poletto, V. Basile, M. Sanguinetti, C. Bosco, and V. Patti. Resources and benchmark corpora for hate speech detection: a systematic review. *Language Resources and Evaluation*, 55:477–523, 2021. 29

[76] J. Qian, A. Bethke, Y. Liu, E. M. Belding, and W. Y. Wang. A benchmark dataset for learning to intervene in online hate speech. *CoRR*, abs/1909.04251, 2019. 29

[77] D. R. Radev, A. Stent, J. R. Tetreault, A. Pappu, A. Iliakopoulou, A. Chanfreau, P. de Juan, J. Vallmitjana, A. Jaimes, R. Jha, and R. Mankoff. Humor in collective discourse: Unsupervised funniness detection in the new yorker cartoon caption contest. *CoRR*, abs/1506.08126, 2015. 44

[78] A. Radford and K. Narasimhan. Improving language understanding by generative pre-training. 2018. 22

[79] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019. 22

[80] A. Rahaman and M. Kabir. *Sarcasm Detection in Tweets: A Sarcastic Feature Based Approach Using Supervised Machine Learning Model*. PhD thesis, 03 2021. 44

[81] S. Rao and J. R. Tetreault. Dear sir or madam, may I introduce the YAFC corpus: Corpus, benchmarks and metrics for formality style transfer. *CoRR*, abs/1803.06535, 2018. 31

[82] A. Reyes, P. Rosso, and D. Buscaldi. From humor recognition to irony detection: The figurative language of social media. *Data & Knowledge Engineering*, 74:1–12, 2012. 44

[83] K. Riaz. Rule-based named entity recognition in urdu. In *Proceedings of the 2010 named entities workshop*, pages 126–135, 2010. 38

[84] A. Ritter, S. Clark, Mausam, and O. Etzioni. Named entity recognition in tweets: An experimental study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. 39

[85] T. Rocktäschel, M. Weidlich, and U. Leser. Chemspot: a hybrid system for chemical named entity recognition. *Bioinformatics*, 28(12):1633–1640, 2012. 39

[86] A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization, 2015. ix, 17

[87] R. Rzepka, Y. Amaya, M. Yatsu, and K. Araki. Automatic narrative humor recognition method using machine learning and semantic similarity based punchline detection. 07 2015. 44

[88] S. K. Saha, S. Sarkar, and P. Mitra. A hybrid feature set based maximum entropy hindi named entity recognition. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*, 2008. 39

[89] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manag.*, 24:513–523, 1988. 8

[90] M. Sap, D. Card, S. Gabriel, Y. Choi, and N. A. Smith. The risk of racial bias in hate speech detection. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1668–1678, 2019. 29

[91] A. Schmidt and M. Wiegand. A survey on hate speech detection using natural language processing. In *Proceedings of the fifth international workshop on natural language processing for social media*, pages 1–10, 2017. 28

[92] F. Sheikha and D. Inkpen. Automatic classification of documents by formality. pages 1 – 5, 09 2010. 31

[93] C. Sun, X. Qiu, Y. Xu, and X. Huang. How to fine-tune BERT for text classification? *CoRR*, abs/1905.05583, 2019. 31

[94] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks, 2014. 13

[95] Z. Talat and D. Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *North American Chapter of the Association for Computational Linguistics*, 2016. 28

[96] W. L. Taylor. "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433, 1953. 24

[97] P. D. Turney and M. L. Littman. Measuring praise and criticism: Inference of semantic orientation from association. *ACM Trans. Inf. Syst.*, 21(4):315–346, oct 2003. 27

[98] A. UMAR, S. Bashir, M. Abdullahi, and O. S. Adebayo. Comparative study of various machine learning algorithms for tweet classification. *i-manager's Journal on Computer Science*, 6:12, 01 2019. 31

[99] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017. 18, 19

[100] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. 03 2018. 11

[101] O. Weller and K. Seppi. Humor detection: A transformer gets the last laugh. *arXiv preprint arXiv:1909.00252*, 2019. 44

[102] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. 34, 41, 47

[103] G. Xu, Y. Meng, X. Qiu, Z. Yu, and X. Wu. Sentiment analysis of comment texts based on bilstm. *Ieee Access*, 7:51522–51532, 2019. 27

[104] D. Yang, A. Lavie, C. Dyer, and E. Hovy. Humor recognition and humor anchor extraction. pages 2367–2376, 01 2015. 44

[105] L. Yang, Y. Li, J. Wang, and R. S. Sherratt. Sentiment analysis for e-commerce product reviews in chinese based on sentiment lexicon and deep learning. *IEEE access*, 8:23522–23530, 2020. 27

[106] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, 2020. 24

[107] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, June 2016. Association for Computational Linguistics. 38

[108] W. Zhang, X. Li, Y. Deng, L. Bing, and W. Lam. A survey on aspect-based sentiment analysis: Tasks, methods, and challenges, 2022. 27

[109] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015. 27

[110] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015. 31

[111] Z. Zhang, D. Robinson, and J. A. Tepper. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *Extended Semantic Web Conference*, 2018. 29

[112] J. T. Zhou, H. Zhang, D. Jin, H. Zhu, M. Fang, R. S. M. Goh, and K. Kwok. Dual adversarial neural transfer for low-resource named entity recognition. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3461–3471, 2019. 39

[113] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 19–27, 2015. 24, 34, 41, 47