NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
MSC. IN DATA SCIENCE AND MACHINE LEARNING

Autonomous Vehicles: Multi-Class Twitter Sentiment Analysis

DIPLOMA THESIS

MARIA-FILIPPA TRIVYZA

Supervisor: George Matsopoulos
Professor, NTUA

Athens, May 2021

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
MSC. IN DATA SCIENCE AND MACHINE LEARNING

Autonomous Vehicles: Multi-Class Twitter Sentiment Analysis

DIPLOMA THESIS

MARIA-FILIPPA TRIVYZA

Supervisor: George Matsopoulos
Professor, NTUA

Approved by the three-member graduation committee on May 18, 2021.

....	....	....
George Matsopoulos	Emmanuel Protonotarios	Nikolaos Doulamis
Professor, NTUA	Professor Emeritus, NTUA	Associate Professor, NTUA

Athens, May 2021

....
Maria-Filippa Trivyza

Copyright © Maria-Filippa Trivyza, 2021
All rights reserved.

The present work is copyrighted and may not be reproduced, stored nor distributed in whole or in part for commercial purposes. Permission is hereby granted to reproduce, store and distribute this work for non-profit, educational and research purposes, provided that the source is acknowledged and the present copyright message is retained Enquiries regarding use for profit should be directed to the author.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the National Technical University of Athens.

Acknowledgements

First of all, I would like to thank my supervisor, Prof. George Matsopoulos, for his guidance and support throughout the period I conducted my thesis.

Secondly, I would like to express my appreciation and gratitude to Prof. Emeritus Emmanuel Protonotarios, for the opportunity he offered me to get involved with a very inspiring topic of machine learning, that combines areas of my great interest, namely deep learning and natural language processing.

I would also like to thank my friends, in and out of NTUA, who have accompanied me so far. Last but not least, a big thank you to my family for their invaluable support and contributions throughout the years.

Marilia Trivyza
May 2021

Abstract

Sentiment analysis (or opinion mining) refers to the use of natural language processing (NLP) and machine learning (ML) to interpret and classify opinions in a piece of human-written text. Sentiment analysis in social media data such as Twitter messages (tweets) present an important topic of research and can refer to the automatic identification of opinions of consumers towards an event, organization, product, brand or person by analyzing their posts. Most studies related to sentiment analysis focus on the binary and ternary classification of these opinions, even though the task of multi-class classification has always been the most interesting, yet most challenging. This thesis researches the task of multi-class Twitter sentiment analysis and aims to explore the views of the general public towards self-driving cars or autonomous vehicles (AVs). We used an imbalanced annotated Twitter data set with captured self-driving car-related tweets, and a data set containing movie reviews used for fine-grained classification. We implemented and experimented with various deep learning models, namely a 2-layer bi-directional long short-term memory (BLSTM) network with self-attention (2-BLSTM+Att), and various state-of-the-art (SOTA) bi-directional encoder representations from transformers (BERT) models. Also, we used a weighted loss function in order to tackle the problem of class imbalance. We evaluated our models using the accuracy and F1 score metrics. To find people's opinion regarding the controversial technology of AVs, self-driving car-related tweets were captured using the Twitter API and classified in a five-scale sentiment polarity from highly negative, negative, neutral, positive to highly positive.

Key Words

Sentiment analysis, Social media, Twitter sentiment analysis, Autonomous vehicles, Multi-class classification, Machine learning, Deep learning, Natural language processing, Transfer learning, Recurrent neural networks, Attention mechanisms, Imbalanced data

Περίληψη

Η ανάλυση συναισθήματος (ή η εξόρυξη γνώμης) αναφέρεται στη χρήση της επεξεργασίας φυσικής γλώσσας (NLP) και της μηχανικής μάθησης (ML) για την ερμηνεία και την ταξινόμηση των συναισθημάτων σε ένα κομμάτι κειμένου γραμμένο από άνθρωπο. Η ανάλυση συναισθήματος σε δεδομένα κοινωνικών μέσων όπως τα Twitter μηνύματα (tweets) παρουσιάζει ένα σημαντικό θέμα έρευνας και αναφέρεται στον αυτόματο προσδιορισμό των απόψεων των καταναλωτών για ένα συμβάν, οργανισμό, προϊόν, επωνυμία ή άτομο αναλύοντας τις δημοσιεύσεις τους. Οι περισσότερες μελέτες που σχετίζονται με την ανάλυση συναισθήματος επικεντρώνονται στη δυαδική και τριμερή ταξινόμηση αυτών των απόψεων, παρόλο που η ταξινόμηση πολλαπλών τάξεων ήταν πάντα η πιο ενδιαφέρουσα, αλλά και η πιο δύσκολη. Αυτή η διατριβή ερευνά την ανάλυση συναισθήματος πολλαπλών κλάσεων από tweets και στοχεύει στη διερεύνηση των απόψεων του κοινού σχετικά με τα αυτοκινούμενα αυτοκίνητα ή τα αυτόνομα οχήματα (AVs). Χρησιμοποιήσαμε ένα μη-ισορροπημένο σχολιασμένο σύνολο δεδομένων από συλλεγμένα tweets που σχετίζονται με τα αυτοκινούμενα αυτοκίνητα και ένα σύνολο δεδομένων που περιέχει κριτικές ταινιών, το οποίο χρησιμοποιείται για ταξινόμηση πολλαπλών κλάσεων. Υλοποιήσαμε και πειραματιστήκαμε με διάφορα μοντέλα βαθιάς μάθησης, όπως ένα δίκτυο δύο στρωμάτων αμφίδρομης μακροπρόθεσμης μνήμης (BLSTM) με αυτο-προσοχή (2-BLSTM+Att) και διάφορα υπερσύγχρονα (SOTA) μοντέλα αμφίδρομων αναπαραστάσεων κωδικοποιητών από τους μετασχηματιστές (BERT). Επίσης, χρησιμοποιήσαμε μια σταθμισμένη συνάρτηση απώλειας για να αντιμετωπίσουμε το πρόβλημα των μη-ισορροπημένων δεδομένων. Για να βρεθεί η γνώμη των ανθρώπων σχετικά με την αμφιλεγόμενη τεχνολογία των AVs, συλλέχθηκαν tweets που σχετίζονται με τα αυτοκινούμενα αυτοκίνητα χρησιμοποιώντας το API του Twitter και ταξινομήθηκαν σε πολικότητα συναισθήματος πέντε κλάσεων από εξαιρετικά αρνητικό, αρνητικό, ουδέτερο, θετικό σε πολύ θετικό.

Λέξεις κλειδιά

Ανάλυση συναισθήματος, Μέσα κοινωνικής δικτύωσης, Twitter ανάλυση συναισθήματος, Αυτόνομα οχήματα, Ταξινόμηση πολλαπλών κλάσεων, Μηχανική μάθηση, Βαθιά μάθηση, Επεξεργασία φυσικής γλώσσας, Μεταφορά μάθησης, Επαναλαμβανόμενα νευρωνικά δίκτυα, Μηχανισμοί προσοχής, Μη-ισορροπημένα δεδομένα

Contents

Acknowledgements 1

Abstract 3

Περίληψη 5

1 Introduction 17
1.1 Motivation . . . 17
1.2 Research Objectives & Contributions . . . 18
1.3 Thesis Structure . . . 19

2 Related Work 20
2.1 Twitter Sentiment Analysis . . . 20
2.2 Acceptance of Autonomous Vehicles . . . 21

3 Machine Learning 23
3.1 Definition of Machine Learning . . . 23
3.2 Types of Machine Learning . . . 23
3.2.1 Supervised Learning . . . 24
3.2.2 Unsupervised Learning . . . 24
3.2.3 Reinforcement Learning . . . 25
3.3 Feature Engineering . . . 25
3.3.1 One-Hot Encoding . . . 26
3.4 Feature Learning . . . 26
3.4.1 Principal Component Analysis . . . 27
3.4.2 t-Distributed Stochastic Neighbor Embedding . . . 27
3.5 Classification Problems . . . 27
3.6 Loss Function . . . 28
3.6.1 Binary Cross-Entropy Loss . . . 28
3.6.2 Categorical Cross-Entropy Loss . . . 30
3.7 Cost Function . . . 30
3.8 Hyper-parameter . . . 31
3.9 Optimization Strategies . . . 31
3.9.1 Gradient Descent . . . 31

3.9.2 Stochastic Gradient Descent 32
3.9.3 Learning Rate 33
3.9.4 Adaptive Moment Estimation 34
3.10 Class Balancing 35
3.11 Evaluation Metrics 36
3.11.1 Confusion Matrix 36
3.11.2 Accuracy 37
3.11.3 Precision 37
3.11.4 Recall 37
3.11.5 Specificity 37
3.11.6 Balanced Accuracy 38
3.11.7 F-Measure 38

4 Deep Learning 39
4.1 Artificial Neural Networks 39
4.1.1 Components 39
4.1.2 Organization 41
4.1.3 Learning 41
4.1.4 Back-propagation 42
4.1.5 Activation Function 42
4.1.5.1 Binary Step Function 43
4.1.5.2 Linear Function 43
4.1.5.3 Sigmoid Function 44
4.1.5.4 Hyperbolic Tangent Function 45
4.1.5.5 Rectified Linear Unit 46
4.1.5.6 Softmax 47
4.1.6 Regularization 47
4.1.6.1 L1 Regularization 48
4.1.6.2 L2 Regularization 48
4.1.6.3 Dropout 48
4.1.6.4 Data Augmentation 49
4.1.6.5 Early Stopping 49
4.2 Definition of Deep Learning 49
4.3 Recurrent Neural Networks 50
4.3.1 Uni-Directional Recurrent Neural Networks 51
4.3.2 Bi-Directional Recurrent Neural Networks 52
4.3.3 Long Short-Term Memory Networks 53
4.4 Attention Mechanisms 54
4.5 Transfer Learning 56

5 Natural Language Processing 58
5.1 Definition of Natural Language Processing 58
5.2 Sentiment Analysis 58
5.3 Text Pre-processing 59
5.3.1 Tokenization 59
5.3.2 Lowercasing 59
5.3.3 Stop Word Removal 60
5.3.4 Stemming 60
5.3.5 Lemmatization 60
5.3.6 Topic Modeling 61
5.3.6.1 Latent Dirichlet Allocation 61
5.4 Language Modeling 62
5.4.1 N-Gram Language Model 62
5.4.2 Neural Language Model 63
5.5 Word Embeddings 65
5.5.1 Term Frequency-Inverse Document Frequency Vector 65
5.5.2 Word2Vec 66
5.5.2.1 Continuous Skip-Gram Model 67
5.5.2.2 Continuous Bag-of-Words Model 68
5.5.3 Global Vectors for Word Representation 68
5.6 Transfer Learning in Natural Language Processing 69
5.6.1 Embeddings from Language Models 69
5.6.2 Bidirectional Encoder Representations from Transformers . . . 70

6 Experiments 74
6.1 Data Sets 74
6.1.1 Twitter Sentiment Analysis Self-Driving Cars 74
6.1.2 Stanford Sentiment Treebank 75
6.2 Explanatory Data Analysis 75
6.2.1 Pre-processing 75
6.2.2 Keyword & Entity Analysis 76
6.2.3 Topic Modeling 78
6.3 Proposed Systems 80
6.3.1 LSTM Based Network 80
6.3.1.1 Pre-processing 80
6.3.1.2 Model Architecture 80
6.3.1.3 Experimental Setup 82
6.3.1.4 Results 82
6.3.2 BERT-Based Networks 83
6.3.2.1 Model Architecture 83
6.3.2.2 Experimental Setup 84

6.3.2.3 Results 85
6.4 Summarized Results 87
6.5 Classifying Unlabelled Tweets about Autonomous Vehicles 88

7 Conclusions 90

Bibliography 93

List of Figures

1.1 Relationship between Artificial Intelligence, Machine Learning, Deep Learning and Natural Language Processing. [1] 18

3.1 Unsupervised vs. Supervised Machine Learning. The left finds logical groupings; the right identifies a boundary between 2 classes. [38] . . . 25
3.2 Example of One-Hot Encoding. [39] 26
3.3 Log Loss when True Label = 1. [42] 29
3.4 Gradient Descent Stuck at Local Minima. [43] 32
3.5 Gradient Descent vs Stochastic Gradient Descent Error Rate During Training. [45] 33
3.6 Choosing a Learning Rate. [46] 34
3.7 Confusion Matrix with 2 Class Labels. [51] 36

4.1 Structure of a typical Biological Neuron. [52] 40
4.2 Computation Model of a Neuron. [53] 40
4.3 Overview of a typical Fully Connected Feed-Forward Artificial Neural Network. [54] 41
4.4 A simple Neural Network Model. [55] 42
4.5 Binary Step Activation Function. [56] 43
4.6 Linear Activation Function. [56] 44
4.7 Sigmoid Activation Function. [56] 45
4.8 Tanh Activation Function. [56] 45
4.9 Rectified Linear Unit (ReLU) Activation Function. [56] 46
4.10 Under-fitting vs. Appropriate-fitting vs. Over-fitting. [57] 47
4.11 Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. [58] 48
4.12 Comparison of Recurrent Neural Networks (on the left) and Feed-forward Neural Networks (on the right). [60] 50

4.13 Each rectangle is a vector and arrows represent functions. Input vectors are in red, output vectors are in blue and green vectors hold the Recurrent Neural Network (RNN)'s state. From left to right: (1) Vanilla mode of processing without Recurrent Neural Network (RNN), from fixed-sized input to fixed-sized output. (2) Sequence output. (3) Sequence input. (4) Sequence input and sequence output. (5) Synced sequence input and output. Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like. [61] 50
4.14 A simple Recurrent Neural Network Unrolled in time. All Recurrent Neural Network (RNN)s have the form of a chain of repeating modules of a neural network (green box). In vanilla Recurrent Neural Network (RNN)s, this repeating module will have a very simple structure, such as a single *tanh* layer. The arrows indicate memory or simply feedback to the next input. [62] 51
4.15 A Bi-directional Recurrent Neural Network. [64] 52
4.16 Long Short-Term Memory Chain. The repeating module in an Long Short-Term Memory (LSTM) contains four interacting layers. [66] . . 53
4.17 The Transformer - model architecture. [70] 55
4.18 Traditional Learning vs. Transfer Learning. [74] 56
4.19 Types of Transfer Learning. [77] 57

5.1 Word Tokenization Example. Tokenization can split a sentence written in English (a segmented language) into individual words by a blank space and also remove punctuation characters at the same time. [78] 59
5.2 Stemming vs. Lemmatization. [79] 61
5.3 Schematic of Topic Modeling. [80] 61
5.4 Exemplary split of a phrase into uni-, bi- and trigrams. [81] 63
5.5 Neural Probabilistic Language Model Architecture. [82] 64
5.6 Word2Vec model architectures: (a) CBOW, (b) Skip-Gram. [85] . . . 67
5.7 The general procedure of sequential transfer learning. [88] 69
5.8 ELMo uses internal representations of multi-layer biLM. [89] 70
5.9 BERT: Masked Language Model. [90] 71
5.10 BERT: Next Sentence Prediction. [90] 71
5.11 BERT: The two steps of how it is developed. [90] 72
5.12 BERT-base vs. BERT-large. [90] 72

6.1 Distribution of Tweets' Character Length in the Twitter Sentiment Analysis Self-Driving Cars Data Set. 76
6.2 Word Cloud of Most Common Words of Twitter Sentiment Analysis Self-Driving Cars Data Set (a) without Lemmatization (b) with Lemmatization. 77
6.3 Intertopic Distance Map (via Multidimensional Scaling). 78

6.4 Topic Modeling: Word Clouds of Top Words (a) Topic #1 (b) Topic #2 (c) Topic #3 (d) Topic #4 (e) Topic #5. 79
6.5 Visualization of the Embeddings using (a) PCA (b) t-SNE. 81
6.6 Confusion Matrix of 2-BLSTM+Att Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set. 83
6.7 Confusion Matrix of BERT (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set. 85
6.8 Confusion Matrix of DistilBERT (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set. 86
6.9 Confusion Matrix of RoBERTa (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set. 86
6.10 Confusion Matrix of BERTweet (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set. 87
6.11 Confidence of each sentiment of sample tweet with (a) id=1, (b) id=2, (c) id=3, (d) id=4, (e) id=5, (f) id=6. 88
6.12 Opinions about Autonomous Vehicles. 89

List of Tables

6.1 A sample of the Twitter Sentiment Analysis Self-Driving Cars Data Set. 74
6.2 Samples distribution among classes in the training, validation and testing data set. 74
6.3 A sample of the Stanford Sentiment Treebank Data Set. 75
6.4 Most Common Words of Twitter Sentiment Analysis Self-Driving Cars Data Set. 76
6.5 Most Common Words of Twitter Sentiment Analysis Self-Driving Cars Data Set. 77
6.6 @-Analysis and Top Hashtags of Twitter Sentiment Analysis Self-Driving Cars Data Set. 78
6.7 Hyper-parameters of the 2-BLSTM+Att Model. 82
6.8 Cost-Sensitive Learning: Class Weights. 82
6.9 Classification Report of 2-BLSTM+Att Model. 83
6.10 Default hyper-parameters' values of BERT. 84
6.11 Hyper-parameters of our BERT based models. 85
6.12 Classification Report of BERT (base) Model. 85
6.13 Classification Report of DistilBERT (base) Model. 86
6.14 Classification Report of RoBERTa (base) Model. 86
6.15 Classification Report of BERTweet (base) Model. 87
6.16 Comparison of the models across the data sets. 87
6.17 A sample of the Unlabelled Tweets about Autonomous Vehicles Data Set. 88
6.18 Top Unigrams of Unlabelled Tweets about Autonomous Vehicles. . . . 89

Chapter 1

Introduction

1.1 Motivation

In everyday life, people make decisions and develop behaviors based on others' thoughts and opinions. With the explosive growth of opinion-rich resources like blogs, online forums, social media, and micro-blogging websites, new opportunities and challenges arise as people can now actively use information technologies to seek out and evaluate the opinions of others. Such resources contain opinionated data about any service, product, topic, event, and idea. Thus, the aforementioned resources can be effectively used for extracting valuable information from them. Businesses and organizations no longer have to conduct surveys, focus groups, and opinion polls to gather consumer or public opinion, as the abundance of online data available makes it much easier to collect all the information needed to understand the social sentiment of their brand, service or product.

Twitter is a popular micro-blogging and social networking service that, in the past decade, has become one of the richest sources of opinionated data. Twitter users create online postings, called "tweets", in order to share opinions, thoughts and sentiments about any topic of their interest. Tweets are short messages, limited to a rather small amount of characters, so they can be viewed as a sentence or a headline rather than a document. The language used in tweets is quite informal, and contains a variety of slang, spelling mistakes, emoticons, punctuation, URLs, creative spelling, and genre-specific terminology and abbreviations, like "re-tweet" (RT), and hashtags (#). Another aspect of tweets is that they include structured information about the users involved in the communication, such as who follows whom.

Mining opinions and sentiment from such online resources is very challenging due to the vast amount of information generated everyday. Each source typically contains a huge volume of text data that is nearly impossible for the average person to decipher and summarize the opinions in them. For that reason, researchers have started investigating and developing systems that can automatically detect, or predict, sentiment in text and effectively mine opinionated information even from a massive amount of data. The field of research in text mining that intends to identify people's opinions, thoughts, and views expressed in a piece of text is called "sentiment analysis". Sentiment analysis (or opinion mining) applies a mix of statistics, Natural Language Processing (NLP), and Machine Learning (ML) to identify and extract subjective information from text files. In particular, Twitter Sentiment

Analysis (TSA) is a technique that tackles the problem of analyzing the tweets in terms of the sentiments they express. In other words, TSA involves dissecting tweets and the content of these expressions.

In this thesis, we focus on Twitter users' opinions regarding one of the most controversial new products, namely self-driving cars. A self-driving car, also known as an Autonomous Vehicle (AV), is a vehicle that is capable of sensing its environment and moving safely with little or no human input. Tech companies like Google and Tesla are competing with each other to attract more people to purchase AVs. However, there is a need to study the impact and consequences of these technologies as they are completely unfamiliar to people or society. Despite the large number of studies in this field, there is limited number of studies on TSA and self-driving cars. Therefore, we aim to contribute to this direction; to study on people's acceptance level and perceptions regarding AVs.

1.2 Research Objectives & Contributions

Most of the State-Of-The-Art (SOTA) approaches on the automatic sentiment analysis of texts collected from social networks and micro-blogging services are mostly focusing on the binary or ternary sentiment classification. In other words, they classify texts into either "positive" and "negative", or into "positive", "negative" and "neutral". In this thesis, we aim to go deeper in the classification of texts collected from Twitter and classify tweets in a five-scale sentiment polarity from highly negative, negative, neutral, positive to highly positive, based on the sentiment towards self-driving cars. For this purpose, we research methods that can help algorithms deal effectively with the task of multi-class sentiment analysis. Our work lies in the intersection of NLP, ML, and Deep Learning (DL), as shown in Figure 1.1.

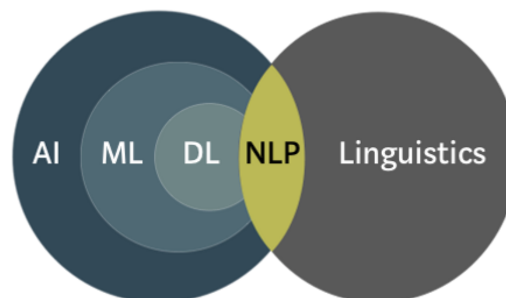

Figure 1.1: Relationship between Artificial Intelligence, Machine Learning, Deep Learning and Natural Language Processing. [1]

More specifically, we employ supervised deep learning models, which get trained to label tweets based on their expressed sentiment. The models are then tested and evaluated for their performance on various evaluation metrics. To train and evaluate the models, we use an imbalanced annotated Twitter data set with AV-related tweets, and a data set containing movie reviews. In order to find people's opinion regarding AVs, we capture and classify AV-related tweets using the Twitter API. Then, we present the results of the sentiment analysis performed on the captured Twitter messages and the current acceptance levels of autonomous mobility.

1.3 Thesis Structure

In chapter 2, we discuss prior research in the area of sentiment analysis, and specifically TSA. In addition, we present studies that focus on the public perception of self-driving cars or AVs.

In chapter 3, we introduce the field of machine learning and provide the knowledge required for the subsequent chapters. In particular, after presenting the types of learning in this field, we focus on classification problems and the basic steps needed to solve and evaluate these kinds of problems.

In chapter 4, we introduce the reader to artificial neural networks and deep learning in general. More specifically, we present the types of deep learning models that are mostly used in the thesis, namely recurrent neural networks, and we explain the concepts of attention mechanisms and transfer learning.

In chapter 5, we present the natural language processing background needed to understand this work. First, we briefly present the task of sentiment analysis. Then, after exploring popular text pre-processing techniques, we present language modeling, initially in the form of a n-gram language model and then as a neural language model. Finally, we explain word embedding and transfer learning methods that are currently used to train natural language processing models.

In chapter 6, our main goal is to present our work on multi-class Twitter sentiment analysis. We start with a description of the data sets that we train our models on, as well as an explanatory data analysis. Then, we describe the models that we used in our experiments and the corresponding results. In addition, we conduct multi-class sentiment analysis on captured unlabelled tweets about AVs.

Finally, chapter 7 contains the conclusion where we summarize our findings and discuss about future directions.

Chapter 2

Related Work

2.1 Twitter Sentiment Analysis

With the growth of social media and micro-blogging services, people began to openly discuss their opinions and thoughts online. The Twitter platform has been the subject of much recent sentiment analysis research, as tweets often express a user's opinion on a topic of interest. Researchers have utilized information derived through TSA to explain and predict product sales [2], stock market movements [3], and the outcomes of political elections [4], [5], [6], [7], [8]. Also, TSA has been used to understand user opinion on diverse business and social issues, like a product brand [9], nuclear power generation [10], and presidential candidate performances in a debate [11].

Some of the research on tweets focus on their characteristics that make TSA challenging, such as the brevity of tweets and resulting compact, novel language with Twitter-specific communication elements [6], [12], a strong sentiment class imbalance [13], [14], and stream-based tweet generation [15], [16]. In addition, some researchers has dealt with the form of the data, the use of slang and how these develop over the time, the use of emoticons, and the nature of tweets themselves [17], [18].

The sentiment polarity classification problem is often modeled as a two-way (binary) or three-way (tenary) sentiment classification of text. Few researches have been conducted on the multi-class sentiment classification task. Note that multi-class classification has conventionally referred to the attribution of one of several sentiment strengths to a text or a tweet. Most of the studies focused on assessing the sentiment strength into different sentiment strength levels (e.g., "very negative", "negative", "neutral", "positive" and "very positive"), or simply give scores ranging from -1 to 1 to the texts, showing at the same time the polarity and the strength of the sentiment [19], [20]. Some researchers have evaluated five-class sentiment models in brand-related TSA to target strong and mild positive and negative sentiments that provide more actionable intelligence to brand management practitioners [9], [12], [21].

There are two commonly used approaches for analyzing the sentiment of texts: the lexicon-based approach and ML methods. The first approach involves the use of a lexicon of opinion-related terms with a scoring method to evaluate sentiment in an unsupervised application [22], [23]. However, the performances of these methods are limited, as they are unable to account for contextual information, nuanced indi-

cators of sentiment expression, or novel vocabulary. The second approach quantify the text based upon a feature representation and apply a ML algorithm to derive the relationship between sentiment and feature values using supervised learning [24], [25]. Models based upon this type of learning require a large training set with sentiment class labels to calibrate model parameters.

Gao et al. [26] proposed an approach that focus in the frequency of sentiment classes in the data set they analyze. The authors concluded that, in contrast to regular classification-oriented algorithms, using a quantification-specific algorithm presents a better frequency estimation. In addition, with the wide adoption of DL as a cutting edge technology, more recent works went "deeper", and new models have been built. For example, the task of multi-class sentiment analysis has been dealt with as well in works such as that of Araque et al. [27].

2.2 Acceptance of Autonomous Vehicles

In recent years, there are a number of studies that focus on the public perception of autonomous mobility and the factors that can influence the likelihood of potential users adopting AVs. In 2016, Zmud et al. [28] conducted an online survey with 556 residents of metropolitan Austin, Texas, to determine potential user intent towards AVs. The researchers classified the respondents into four intent-to-use categories. The "extremely unlikely" category contained 18% of respondents, the "somewhat unlikely" category contained 32% of respondents, 36% were in the "somewhat likely" category, and 14% were in the "extremely likely" category.

In 2017, Greig [29] carried out two surveys to determine the public sentiment towards the adoption of AVs. The study found that only one-fifth of the respondents think AVs are a good idea, and the majority of them remained skeptical about the driver-less cars' promised benefits. Merat et al. [30] focused on the social-psychological factors that can influence people's trust and acceptance of shared SAE Level 4 shared AVs. The study found that the most effective pathway to the acceptance and adoption of AVs is an incremental and iterative stage-by-stage process that provides users with hands-on experience throughout every stage.

In 2018, Greaves et al. [31] conducted an online survey of 455 Australian adults to investigate consumer sentiment toward AVs. The study found that younger, male respondents, less frequent drivers, and people who are open to the idea of sharing their car had more favourable attitudes towards AVs. Negative attitudes were associated more with older, female respondents, those who drive more frequently, and those who are less open to sharing their car. In another study, Liljamo et al. [32] examined the results from a large citizen survey of 2036 people to determine the magnitude and type of concerns that people have in regard to the adoption of AVs. The study found that positive attitudes were associated more with men, highly educated individuals, people living in densely populated area and those living in households without a car. Also, the results indicate that the key perspectives that affect public approval of AVs are traffic safety and ethical perspectives. Nordhoff et al. [33] conducted a survey with 7,755 respondents from 116 countries to determine the relationship between socio-demographic characteristics and people's acceptance of AVs. They concluded that self-reported acceptance of driver-less vehicles is more

strongly determined by domain-specific attitudes than by socio-demographic characteristics.

In 2019, Sener et al. [34] conducted online surveys to examine the factors influencing people's intent to use AVs in the future; the survey included people in various Texas cities, including Dallas, Houston, and Waco. The study found that younger individuals, males and individuals with physical conditions that prohibit them from driving had a higher likelihood of intent to use. Also, the strongest associations with intent to use were observed for attitudes towards self-driving vehicles, performance expectation, perceived safety, and social influence. Moody et al. [35], in an extensive survey of more than 30,000 individuals from more than 50 countries, explore the perceived safety of AVs, the awareness of AVs and the expectations of when the technology would be safe enough for use. They find young, male and highly educated individuals to be more likely to perceive the use of AVs as being safe. Also, they note that individuals who are generally more aware of AVs perceived AVs to be safer.

In a more recent study, Rezaei and Caulfield [36] conducted a survey among 475 Irish people to evaluate their willingness to adopt AVs in their daily commute trends. Also, they analyzed people's acceptance and willingness to pay for AVs compared to manually driven vehicles. The study found that only a fifth of the people expressed a high interest in driving AVs. Most people had concerns about their privacy and were also unsure about the safety and security of AVs' operation. In addition, the results revealed that people were much more interested in purchasing an AV when the cost was not an issue.

Chapter 3

Machine Learning

3.1 Definition of Machine Learning

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that provides systems the ability to automatically learn and improve without following strictly static program instructions. More specifically, machine learning is an approach to data analysis that involves building and adapting mathematical models, which allow programs to improve at problem solving through experience. According to Prof. Tom M. Mitchell, a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E [37].

Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves, automatically, without human intervention or assistance and adjust actions accordingly in order to adapt to new data. The process of learning begins with observations or sample data, known as "training data", in order to look for patterns in data and correctly generalize to new settings, known as "test data", based on the provided examples.

Machine learning algorithms are used in a vast range of applications, such as speech recognition, search engines, traffic prediction, email filtering, medical diagnosis, image processing and self-driving cars, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks. A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers. However, not all machine learning is statistical learning. In its application across business problems, machine learning is also referred to as predictive analytics.

3.2 Types of Machine Learning

Machine learning approaches are traditionally classified into three broad categories, depending on how much feedback they present to the learning system. In some cases, the computer is provided a significant amount of example inputs and their desired outputs, also known as "labelled data", which is called supervised learning. In other cases, no labelled data is provided and this is known as unsupervised learning. Lastly, in reinforcement learning, a computer program interacts with a dynamic environment in which it must perform a certain goal. As it navigates its

problem space, the program is provided feedback that's analogous to rewards, which it tries to maximize. Other types of learning algorithms have been developed which don't fit neatly into this three-fold categorisation, such as semi-supervised learning, in which some labelled training data is provided, but most of the training data is unlabelled.

3.2.1 Supervised Learning

Supervised learning is the most practical and widely adopted form of machine learning. It is the task of learning a mathematical function f that maps input variables X to the preferred output variable Y, also known as a supervisory signal, based on example input-output pairs:

$$Y = f(X) \tag{3.1}$$

A supervised learning algorithm analyzes a large amount of labeled training data and produces an inferred function, which can be used to determine the class labels for new unlabeled examples. Optimally, this function will allow the algorithm to generalize from the training data and accurately predict the output for unseen instances. An algorithm that improves the model performance over time is said to have learned to perform that task and the learning stops when an acceptable level of accuracy is achieved.

Tasks in supervised learning can be categorized further as "classification" or "regression" problems. Classification problems (such as logistic regression) use statistical classification methods to output a class label. On the other hand, regression problems (such as linear regression) use statistical regression analysis to provide a numerical output. Both classification and regression problems may have one or more input variables of any data type, such as numerical or categorical. Popular examples of supervised machine learning algorithms include: decision trees, support vector machines, naive Bayes, and many more.

3.2.2 Unsupervised Learning

Unsupervised learning differs from supervised learning in describing data rather than predicting. In this learning method, also known as self-organization, all input is unlabelled and the algorithm must create structure out of it on its own (without guidance). The goal is to extract relationships or find undetected patterns in a data set with a minimum of human supervision. Unsupervised learning allows for modeling of probability densities over inputs.

One of the main method used in unsupervised learning is cluster analysis. Clustering refers to discovering groupings within the input data in such a way that members of a common group (called a cluster) are similar to each other and different from members of other groups (clusters). In this case, a similarity metric is measured between input data in order to group them into clusters of similar samples. This approach helps detect anomalous data points that do not fit into either cluster. However, it is often difficult to know how many clusters should exist or how they should look. Clustering algorithms can be categorized into a few types, such as connectivity-based, centroid-based, distribution-based, density-based and grid-based clustering.

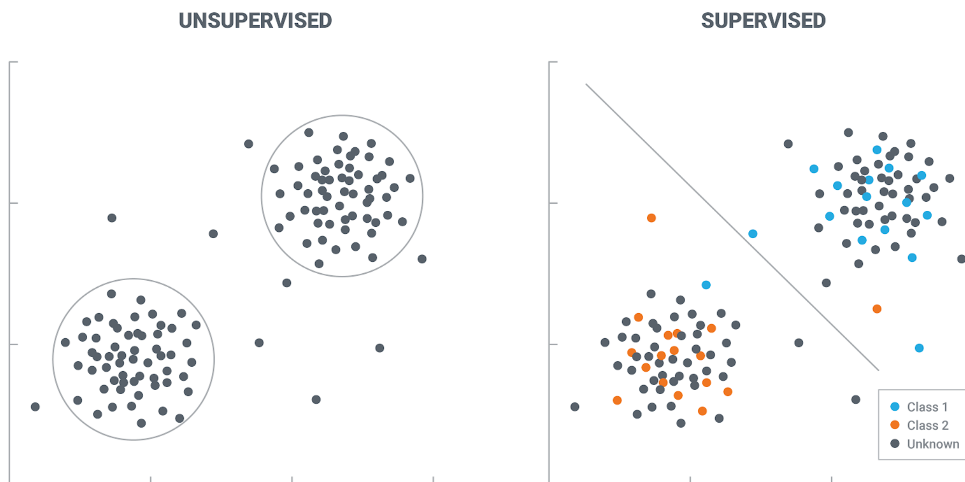

Figure 3.1: Unsupervised vs. Supervised Machine Learning. The left finds logical groupings; the right identifies a boundary between 2 classes. [38]

Another interesting subclass of unsupervised tasks is generative modeling. Generative modeling involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new samples that plausibly could have been drawn from the original data set.

3.2.3 Reinforcement Learning

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment as to maximize some notion of cumulative reward. It focuses on training an algorithm following the trial-and-error approach in order to come up with a solution to the problem. The algorithm (agent) evaluates a current situation (state), takes an action, and receives feedback from the environment after each act. Positive feedback is a reward, negative feedback is a penalty for making a mistake and the goal is to maximize the total reward. An agent basically interacts with the environment and learns from its experience.

The environment is typically represented as a Markov Decision Process (MDP). Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible. Many reinforcement learning algorithms use dynamic programming techniques and are commonly used in autonomous vehicles and in learning to play a game against a human opponent.

3.3 Feature Engineering

Machine learning tasks, such as classification or prediction, often require input that is mathematically and computationally convenient to process. However, real-world data such as images, video, and sensor measurement is usually complex, redundant and high-dimensional, making it harder to find patterns and anomalies. Thus, it is necessary to discover useful features or representations from raw data.

The process of extracting features from raw data by using domain knowledge is called feature engineering. Its purpose is to transform the given data into formats compatible with the ML algorithm at hand, and to improve its performance. Some popular techniques that can be useful in different problems are presented next.

3.3.1 One-Hot Encoding

Categorical variables represent types of data that can divided into a limited number of groups. The representation of categorical data as binary vectors (a numerical format) is called one-hot encoding.

Human-Readable

Pet
Cat
Dog
Turtle
Fish
Cat

Machine-Readable

Cat	Dog	Turtle	Fish
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

Figure 3.2: Example of One-Hot Encoding. [39]

One-hot encoding is one of the most common encoding methods in ML. It creates a vector with length equal to the number of categories in the data set. If a sample belongs to the i-th category, then the i-th component of the vector is assigned a value of 1 and all the other components are assigned the value 0. An example of this method is presented in Figure 3.2.

3.4 Feature Learning

Manual feature engineering often requires expensive human labor, relies on expert knowledge and normally does not generalize well. This motivates the design of efficient feature learning techniques, to automate and generalize this. Feature learning, also called representation learning, is a set of techniques that learn representations of raw data input, typically by transforming it or extracting features from it. This makes it easier for a machine to both learn the features and use them to perform a machine learning task. These techniques allow reconstruction of the inputs coming from the unknown data-generating distribution, while not being necessarily faithful to configurations that are implausible under that distribution.

Feature learning can be divided into two categories: supervised and unsupervised feature learning, analogous to these categories in machine learning generally. In supervised feature learning, features are learned using labeled input data. Examples include artificial neural networks, multilayer perceptrons, and supervised dictionary learning. In unsupervised feature learning, features are learned with unlabeled input data. Examples include principal component analysis, dictionary learning, independent component analysis, auto-encoders and various forms of clustering.

3.4.1 Principal Component Analysis

Principal Component Analysis (PCA) [40] is a linear dimensionality reduction method. It combines the input features in a way that allows the least important features to be dropped while retaining the most valuable parts of all features. In other words, PCA reduces the dimensionality of a multivariate data set, and expresses the important information as a set of few new variables, called principal components, that correspond to a linear combination of the originals. These new variables are all independent of one another and their number is less than or equal to the number of original variables.

The total variation of a given data set contains corresponds to the information in it. PCA performs a linear mapping of the data to a lower-dimensional space in such a way that the variance of the data in the new representation is maximized. It does so by computing the eigenvectors from the covariance matrix. A significant fraction of the variance of the original data is reconstructed by using the eigenvectors that correspond to the largest eigenvalues (the principal components).

3.4.2 t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) [41] is an unsupervised non-linear technique for dimensionality reduction that helps with identifying relevant patterns. It is particularly well suited for visualizing high-dimensional data sets. Specifically, it models each high-dimensional object by a lower-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability. It is applied in image processing, natural language processing, genomic data and speech processing.

In simple terms, t-SNE minimizes the divergence between a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding. In this way, t-SNE maps the multi-dimensional data to a two- or three-dimensional space and attempts to find patterns in the data by identifying observed clusters based on similarity of data points with multiple features.

3.5 Classification Problems

As mentioned in Section 3.2.1, classification problems are a type of supervised machine learning, where the training data is already labeled. They are problems of identifying which category a particular new observation falls into. More specifically, the classification predictive modeling is the task of approximating the mapping function (f) from input variables (X) to discrete output variables (Y), in order to identify which class/label/category the new data will most likely belong in.

The algorithm that implements classification is known as a classifier. Labeled data is used to train a classifier so that the algorithm performs well on data that are not yet labeled. Repeating this process of training a classifier on already labeled data is known as “learning”. In supervised machine learning algorithms, we want to minimize the error for each training example during the learning process. This is

done using some optimization strategies like gradient descent, and this error comes from a loss function.

3.6 Loss Function

In mathematical optimization and decision theory, a loss function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. In statistics, typically a loss function is used for parameter estimation, and the event in question is some function of the difference between estimated and true values for an instance of data. Loss functions play an important role in any statistical model, as they define an objective which the performance of the model is evaluated against. The parameters learned by the model are determined by minimizing a chosen loss function.

In machine learning, loss (or error) functions for classification are computationally feasible loss functions representing the penalty for inaccuracy of predictions in classification problems. It is a method of evaluating how well a specific algorithm models the given data. Formally, a loss function $L(\hat{y}, y)$ assigns a real number to a predicted output $\hat{y}$ given the true expected output y. If the prediction deviates too much from the actual result, the loss function would output a large number. Gradually, with the help of some optimization technique, loss function learns to reduce the error in prediction. The loss function should be bounded from below, with the minimum attained only for cases where the prediction is correct.

There are multiple ways to determine loss. Not all of them fit to the total of possible tasks. Choosing a loss function for a specific problem involves various factors, such as type of machine learning algorithm chosen, ease of calculating the derivatives and to some degree the percentage of outliers in the data set. Broadly, loss functions can be classified into two major categories depending upon the type of learning task we are dealing with: classification and regression loss functions. Some of the most popular loss functions used for training classifiers are presented next.

3.6.1 Binary Cross-Entropy Loss

Generally, the term 'entropy' refers to the disorder or uncertainty in data. The greater the entropy value, the higher the level of uncertainty. For a random discrete variable y with probability distribution $q(y)$, it is measured as:

$$H(q) = -\sum_{c=1}^{C} q(y_c) \log(q(y_c)) \tag{3.2}$$

where C is the number of possible outcomes. The entropy is basically the negative summation of the product of the probability of occurrence of an event with its log over all possible outcomes. So, if the true distribution of a random variable is known, its entropy can be computed. However, the true distribution of the data in most cases is unknown, and we try to approximate it with some other distribution $p(y)$. Cross-entropy measures entropy between two probability distributions. It is the average number of bits required to communicate an event from one distribution

to another distribution. Formally:

$$H_p(q) = -\sum_{c=1}^{C} q(y_c)\log(p(y_c)) \tag{3.3}$$

If the distributions $p(y)$ and $q(y)$ match perfectly, the computed values for both cross-entropy and entropy match as well. Otherwise, cross-entropy will have a bigger value than the entropy computed on the true distribution. The best possible $p(y)$ is the one that minimizes the cross-entropy.

Cross-entropy as a concept is applied in the field of machine learning when algorithms are built to predict from the model build. Model building is based on a comparison of actual results with the predicted results. Cross-entropy signifies reducing entropy or uncertainty for the class to be predicted, and it is suitable as a loss function because the goal is to minimize its value. In a binary classification problem, $C = 2$ and the output label y can take values 0 and 1. In such case, the Binary Cross-Entropy (BCE) per sample is defined as:

$$BCE = -(y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i))) \tag{3.4}$$

where $p(y_i)$ is the predicted probability that the sample i belongs to class 1 (or positive class), and $1 - p(y_i)$ is the predicted probability that the sample i belongs to class 0 (or negative class).

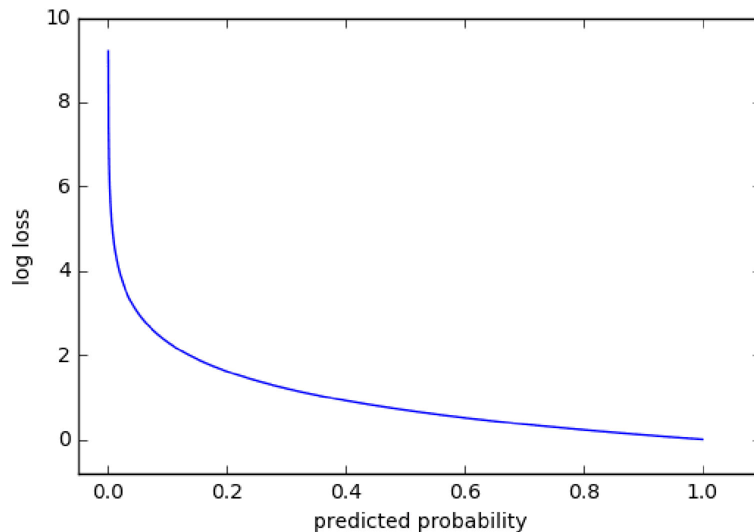

Figure 3.3: Log Loss when True Label = 1. [42]

In summary, binary cross-entropy loss, also called Log Loss, measures the performance of a classification model whose predicted output is a probability value between 0 and 1. Log loss penalizes both types of errors, but especially those predictions that are confident and wrong. Figure 3.3 shows the range of possible loss values given a true observation. As the predicted probability approaches 1, log loss slowly decreases, whereas as the predicted probability decreases, the log loss increases rapidly.

Since probability requires a value in between 0 and 1, the Sigmoid function can be used to calculate p. The sigmoid function, presented in Section 4.1.5.3, is also known as a squashing function because it can squish any real value into the range $[0, 1]$, which makes it suitable for a model where we have probability as our model output.

3.6.2 Categorical Cross-Entropy Loss

The Categorical Cross-Entropy Loss is a loss function that is used in multi-class classification tasks. These are tasks where an observation can only belong to one out of three or more possible categories with probability 1, and to other categories with probability 0. The model must predict one possible class output every time. In the usual case of multi-class classification, the labels are provided in a one-hot representation.

The Categorical Cross-Entropy (CCE) is essentially the binary cross-entropy expanded to multiple classes. It is calculated as a sum of separate loss for each class label per observation. Mathematically, it is given as the following equation:

$$CCE = -\sum_{c=1}^{C} y_{i,c} \log(p(y_{i,c})) \tag{3.5}$$

where C is the number of classes, $y_{i,c}$ is a binary indicator (0 or 1) that indicates whether c is the correct class for the sample i, and $p(y_{i,c})$ denotes the probability of observation i for class c.

When probabilities for multiple classes are considered, one needs to ensure that the sum of all the individual probabilities is equal to 1. Applying sigmoid does not ensure that the sum is always equal to 1, hence there is a need for another function in order to calculate p. The Softmax function, presented in Section 4.1.5.6, ensures that all the output nodes have values between 0–1 and the sum of all output node values equals to 1 always.

3.7 Cost Function

A cost function is a function that measures the performance of a machine learning model for given data. It returns the cost between predicted outcomes compared with the actual outcomes and presents it in the form of a single real number. When the difference between the output and the correct answer is small, the error is low. Depending on the problem, the cost function can be formed in many different ways. While it is possible to define a cost function ad hoc, frequently the choice is determined by the function's desirable properties or because it arises from the model.

Although cost function and loss function are synonymous and used interchangeably, they are different. A loss function is for a single training example. A cost function, on the other hand, is the average loss over the entire training data set. The aim of supervised machine learning is to minimize the overall cost, thus optimizing the correlation of the model to the system that it is attempting to represent.

For example, based on Section 3.6.1, one can determine the binary cross-entropy cost function as follows: during its training, the classifier uses each of the N points in its training set to compute the binary cross-entropy loss (), effectively fitting the distribution $p(y)$. Since the probability of each point is $1/N$, and $y \in \{0, 1\}$, the BCE cost function over the complete set of samples is defined as:

$$J_{BCE} = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i) \log(1 - p(y_i)) \quad (3.6)$$

3.8 Hyper-parameter

In machine learning, the term hyper-parameter is used in order to distinguish from standard model parameters that can be directly learned from the regular training process to fit the data. A hyper-parameter is a constant parameter that is set before the learning process begins and whose value is used to control the learning process. Hyper-parameters can be classified as model hyper-parameters, that cannot be inferred while fitting the machine to the training set because they refer to the model selection task, or algorithm hyper-parameters, that in principle have no influence on the performance of the model but affect the speed and quality of the learning process.

3.9 Optimization Strategies

The main goal of machine learning is to create a model that performs well and gives accurate predictions in a particular set of cases. In order to achieve that, we need machine learning optimization. Optimization is the problem of adjusting the hyper-parameters in order to minimize the defined loss function/cost function by using some optimization technique.

There are perhaps hundreds of popular optimization algorithms, which can make it challenging to know which algorithms to consider for a given optimization problem. The choice, however, of optimization algorithm can make a difference between getting a good accuracy in hours or days. Fundamental optimization methods are typically categorised into first-order, high-order and derivative-free optimization methods. One usually comes across methods that fall into the category of the first-order optimization such as the gradient descent and its variants, which use the first derivative (gradient) to choose the direction to move in the search space.

3.9.1 Gradient Descent

Gradient Descent (GD) is the most popular optimization method that's used when training a machine learning model. It is a first-order iterative optimization algorithm for finding a local minimum of a differentiable objective function $J(\theta)$ parameterized by a model's parameters $\theta \in R^d$. The idea is to iteratively update the parameters in the opposite direction of the gradient (or approximate gradient) of the objective function $\nabla_\theta J(\theta)$ with regard to the parameters, because this is the direction of steepest descent. With every update, this method guides the model to find the target and gradually converge to the optimal value of the objective function.

There are three variants of GD that differ in the total number of samples from a data set (batch) that is used to compute the gradient of the objective function. Depending on the amount of data, there is a trade-off between the accuracy of the parameter update and the time it takes to perform an update.

In order to perform Batch Gradient Descent (BGD), one has to iterate over the entire training data set while readjusting the model. The batch in this case is taken to be the whole data set. More specifically, BGD calculates the error for each example within the training data set, but only after all training examples have been evaluated does the model get updated. This whole process is like a cycle and it's called a training epoch.

To get started, the entire training data set has to be in memory and available to the algorithm, and the initial parameters' values have to be defined. From there, BGD uses calculus to iteratively adjust these values so they minimize the given cost function:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta). \tag{3.7}$$

The learning rate η, presented in Section 3.9.3, determines the size of the step taken at each iteration. When improvement is no longer observed, the optimization is over and a minimum is found. In other words, one follows the direction of the slope of the surface created by the cost function downhill until a valley is reached.

Classical GD is guaranteed to converge to the global minimum for convex error surfaces only. If the surface is non-convex, then it will stop searching when it finds a local minimum. If the cost function is highly non-convex, meaning it consists of many minimum points, then the gradient may settle on any one of the sub-optimal local minima, which depends on the initial parameters and the learning rate.

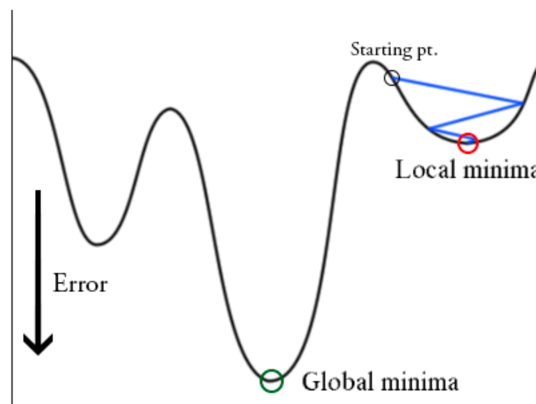

Figure 3.4: Gradient Descent Stuck at Local Minima. [43]

3.9.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) [44] can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual calculation of the gradient from the entire data set by an estimated gradient computed on a randomly selected subset of the data (called a "mini-batch"), which may be as small as a single training sample. SGD performs a parameter update for each training example $x^{(i)}$

and label $y^{(i)}$ as follows:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \quad (3.8)$$

SGD is generally noisier than batch gradient descent as, because of its randomness in its descent, it usually takes a higher number of iterations to reach the minimum. However, it is still computationally much less expensive than BGD, which in large data sets is a significant aspect. Memory usage is also reduced compared to BGD, as SGD does not store values of loss functions. SGD has frequent updates with a high variance that cause a lot of fluctuations in the loss vs iterations graph, as in Figure 3.5.

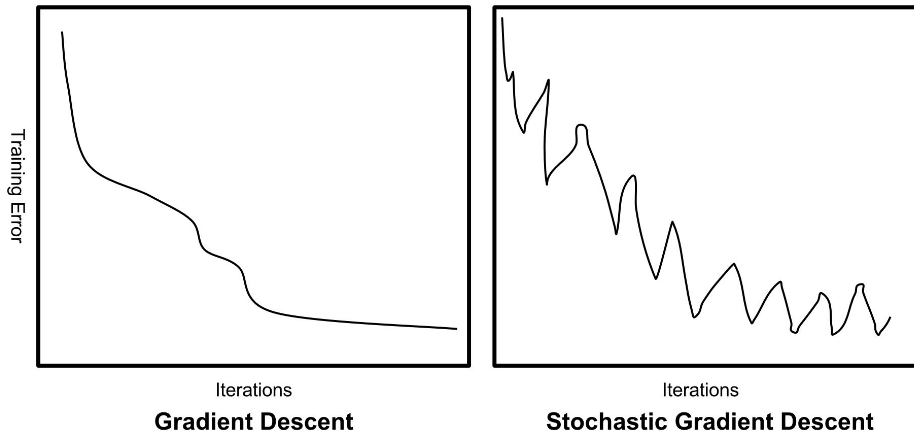

Figure 3.5: Gradient Descent vs Stochastic Gradient Descent Error Rate During Training. [45]

While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation enables it to jump to minima farther away, potentially reaching a better minimum. This ultimately makes it a bit complicated to converge to the exact minimum. However, it has been shown that when the learning rate η decreases with an appropriate rate, SGD shows the same convergence behaviour as BGD, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

3.9.3 Learning Rate

As already mentioned, the learning rate is a tuning parameter in an optimization algorithm that determines the size of the corrective step at each iteration while moving toward a minimum of a loss function. It controls how much to change the model in response to the estimated error each time the model parameters are updated. It metaphorically represents the speed at which a machine learning model "learns".

In setting a learning rate, there is a trade-off between the rate of convergence and overshooting. While the descent direction is usually determined from the gradient of the loss function, the learning rate determines how big a step is taken in that direction. As shown in Figure 3.6, a value too small may result in a long training process that could get stuck in an undesirable local minimum, and a value too large may result in learning a sub-optimal set of parameters too fast or an unstable training process.

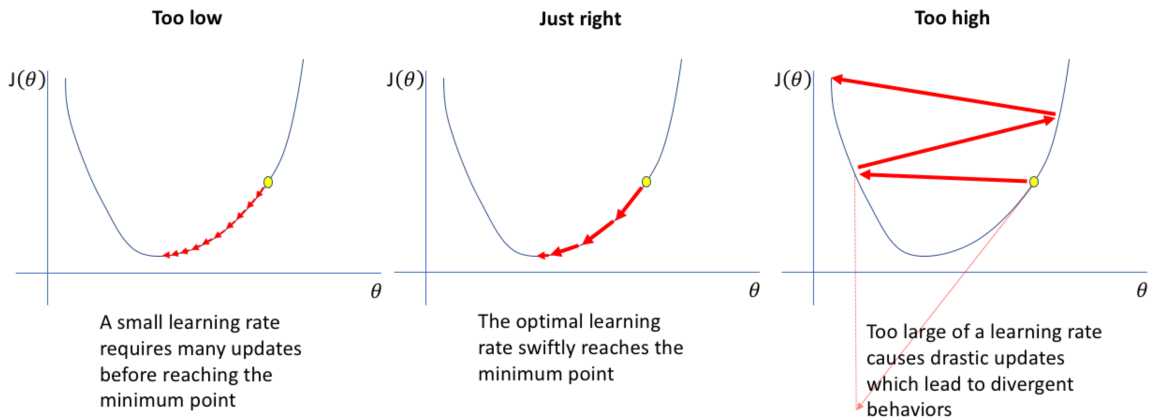

Figure 3.6: Choosing a Learning Rate. [46]

In order to achieve faster convergence, avoid oscillations and getting stuck in undesirable local minima, the learning rate is often varied during training. This can be done either in accordance to a learning rate schedule or by using an adaptive learning rate that increases or decreases as appropriate.

Learning rate schedules seek to adjust the learning rate during training by reducing the learning rate according to a pre-defined schedule. Their hyperparameters have to be defined in advance and they depend heavily on the type of model and problem. Also, the same learning rate is applied to all parameter updates. However, if the data are sparse, one may want to update the parameters in different extent instead. In this case, an adaptive learning rate method can be used to find individual learning rates for different parameters.

3.9.4 Adaptive Moment Estimation

As an improvement to traditional gradient descent algorithms, the adaptive gradient descent optimization algorithms or adaptive learning rate methods can be utilized. Adaptive gradient descent algorithms such as Adagrad [47], which works really well in settings with sparse gradients, but struggles in non-convex optimization, RMSprop [48], which tackles to resolve some of the problems of Adagrad and works really well in on-line setting, and Adam [49], which is almost similar to RMSProp but with momentum [50], provide an alternative to classical SGD. These per-parameter learning rate methods provide heuristic approach without requiring expensive work in tuning hyperparameters for the learning rate schedule manually.

Adaptive Moment Estimation (Adam) is maybe the best optimizer at present. The reason it is called that is because it uses estimations of first (the mean) and second (the uncentered variance) moments of gradient to adapt the learning rate for each parameter. Note, that the n-th moment of a random variable is defined as the expected value of that variable to the power of n. To estimates the moments, Adam utilizes exponentially moving averages, computed on the gradient evaluated on a current mini-batch. Formally:

$$m_t = (1 - \beta_1) g_t + \beta_1 m_{t-1} \tag{3.9}$$

$$v_t = (1 - \beta_2) g_t^2 + \beta_2 v_{t-1} \tag{3.10}$$

where m and v are moving averages, g is gradient on current mini-batch, and β_1, β_2 are new introduced hyper-parameters of the algorithm with default values close to 1. This results in a bias of moment estimates towards 0, which is overcome by first calculating the biased estimates and then calculating bias-corrected estimates:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{3.11}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{3.12}$$

Finally, the parameters of the model are updated similar to the previous optimization methods:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon} \tag{3.13}$$

where η is the learning rate and ε is a small term (10^{-8}) preventing division by 0. Adam is easy to implement, works well on problems with noisy or sparse gradients and with large data sets and large parameters. Also, it is computationally efficient and it requires little memory space.

3.10 Class Balancing

A common problem that appears in machine learning classification tasks is the class imbalance problem. It appears when the occurrence of one or more classes (majority classes) is very high compared to the other classes (minority classes) present. In other words, there are distinct classes with higher number of instances compared to other classes in the data set. This skewness towards the majority class creates problems as most ML algorithms are based on the assumption that the data is balanced, i.e., the data is equally distributed among all of its classes. Therefore, when training a model on an imbalanced data set, the learning becomes biased towards the majority classes.

A simple technique used in order to cope with this problem is having a cost-sensitive loss function. The idea is to weight the loss computed for different samples differently based on whether they belong to the majority or the minority classes. Essentially, a higher weight is assigned to the loss encountered by the samples associated with under-represented classes and lower for the majority ones. The following equation computes the weights of each class:

$$w_c = \frac{n}{Cn_c} \tag{3.14}$$

where w_c is the weight of the class c, n is the number of all observations, C is the total number of classes, and n_c is the number of observations in class c. For example, the weighted log loss function is described by the following formula:

$$BCE_{weighted} = -(w_0(y_i \log(p(y_i)))) + w_1((1-y_i)\log(1-p(y_i))) \tag{3.15}$$

where w_0 is the weight assigned to class 0 and w_0 is the weight assigned to class 1. For multi-class classification, the categorical cross-entropy loss function can be weighted by class to reflect equal error from both majority and minority classes.

3.11 Evaluation Metrics

Evaluating the performance of a trained machine learning model is an essential part of any data science project. How well the model generalizes on the unseen data is what defines adaptive vs. non-adaptive models. By using different evaluation metrics one should be able to improve the overall predictive power of the model. The choice of metrics completely depends on the type of algorithm and influences how the performance of it is measured and compared. An important aspect of evaluation metrics is their capability to discriminate among model results.

3.11.1 Confusion Matrix

Confusion Matrix, or Error Matrix, is the easiest way to measure the performance of a classification problem where the output can be of two or more type of classes. This matrix is an $N \times N$ matrix, where N is the number of classes being predicted. It is a tabular representation of model predictions vs. actual values, where each column and row is dedicated to one class.

For example, suppose some samples in a binary classification problem ($N = 2$) belong to two classes. Also, suppose a binary classifier which predicts a class for a given input sample. On testing the model on a number of test samples for which the true values are known, we get the following table with two dimensions ("True Class" and "Predicted Class"), and sets of classes in both dimensions.

Figure 3.7: Confusion Matrix with 2 Class Labels. [51]

Each prediction can be one of the following outcomes, based on how it matches up to the true class:

- True Positive (TP): The actual class of the data point is True and the predicted is also True.
- True Negative (TN): The actual class of the data point is False and the predicted is also False.
- False Positive (FP): The actual class of the data point is False but the predicted is True.
- False Negative (FN): The actual class of the data point is True but the predicted is False.

Ideally, the model should give 0 False Positives and 0 False Negatives. Based on the type of problem at hand, the minimization of either FP or FN might be considered.

3.11.2 Accuracy

Accuracy (ACC) in classification problems is the number of true predictions (TP and TN) made by the model over all kinds of predictions made (total number of samples). It basically calculates the proportion of the total number of predictions that were correct. Its mathematical formula is:

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{3.16}$$

As mentioned before, accuracy can give wrong impressions about the model, especially in situations where the data set is severe imbalanced, ie. there are many samples of one class and not much of the others. Meaning, if the model is performing well on the class that is dominant in the data set, accuracy will be high, even though the model might not perform well in other cases. Also, note that models that over-fit have an accuracy score of 1.

3.11.3 Precision

Precision is a very useful metric that caries more information than the accuracy metric and it can help when the costs of false positives are high. It is the proportion of positive cases that were correctly identified by the model. Its value can go from 0 to 1 and it is calculated for each class separately with the formula:

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{3.17}$$

3.11.4 Recall

Recall, also known as Sensitivity or True Positive Rate (TPR), can be described as the ability of the classifier to find all the positive samples. It is the proportion of actual positive cases which are correctly identified. It is defined as the fraction of samples from a specific class which are correctly predicted by the model. Mathematically:

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3.18}$$

This metric falls within the 0-1 range, with the 1 being the best value. In contrast to precision, which is a measure of result relevancy, recall is a measure of how many truly relevant results are returned and it can help when the cost of false negatives is high.

3.11.5 Specificity

Specificity, also known as True Negative Rate (TNR) or 1 – False Positive Rate (FPR), can be defined as the proportion of actual negative cases which are correctly identified. Specificity is the exact opposite of recall. It is easily calculated by confusion matrix with the help of the following formula:

$$\mathrm{Specificity} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} \tag{3.19}$$

3.11.6 Balanced Accuracy

Balanced Accuracy (bACC) is a metric that is especially useful when the classes are imbalanced. It is based on two more commonly used metrics: sensitivity (or recall) and specificity. Balanced accuracy is simply the arithmetic mean of the two:

$$\text{bACC} = \frac{\text{Sensitivity} + \text{Specificity}}{2} \tag{3.20}$$

3.11.7 F-Measure

F_β Score is the most popular metric used for multi-class classification or for binary classification where the number of observations is not balanced across the classes. The generalized formula of F-score for classification problems with two or more classes is defined as:

$$F_\beta = \frac{(1+\beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot (\text{Precision} + \text{Recall})} \tag{3.21}$$

where β is a non-negative real number and equal to 1 in binary classification, ie. it represents the number of possible classes reduced by 1. F1 score is the harmonic mean of precision and recall. Its best value is 1, indicating perfect precision and recall, and its worst score is 0, if either the precision or the recall is 0. For a multi-class classification task, in order to combine the F1 score of each class into a single score, the macro or micro F1 metric can be used. The macro F1 score is computed as the arithmetic mean of the N classes' F1 scores as follows:

$$\text{F1}_{macro} = \frac{1}{N} \sum_{i=1}^{N} \text{F1}_i \tag{3.22}$$

where F1_i is the F1 score of the i-th class computed using Equation 3.21 (where $\beta = 1$) in a one vs. rest manner. The micro F1 score treats all predictions on all classes as one vector and then calculates the F1 measure. Note that micro-averaging is biased by class frequency and macro-averaging takes all classes as equally important.

Chapter 4

Deep Learning

4.1 Artificial Neural Networks

Performing machine learning tasks involves creating a model, which is trained on some training data and then can process additional unseen data to make predictions. An Artificial Neural Network (ANN), usually simply called Neural Network (NN), is a computational model vaguely inspired by the structure and functional aspects of biological neural networks that constitute animal brains, like the human brain. ANNs are an attempt to simulate the network of neurons that make up a human brain so that the computer will be able to replicate the way that we humans learn. ANNs are created by programming regular computers to behave as though they are interconnected brain cells.

Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables. They are excellent tools for finding patterns which are far too complex or numerous for a human programmer to extract and teach the machine to recognize. Neural networks are often used in image recognition, language translation, and other common applications today.

4.1.1 Components

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. A neuron, as shown in Figure 4.1, is the basic computational unit of the brain. The human nervous system has more than 86 billion neurons that communicate with each other via synapses. Each neuron has on average 7 thousand synaptic connections to other neurons, and the neurons pass signals to each other via as many as 100 to 500 trillion synapses. Each neuron receives input signals from its dendrites and produces output signals along its (single) axon. The axon eventually branches out and connects via synapses to dendrites of other neurons. The dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon.

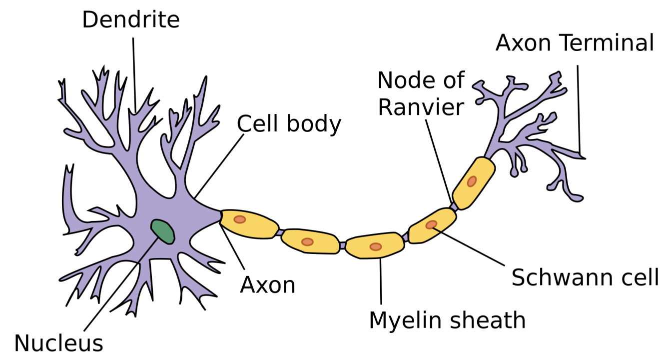

Figure 4.1: Structure of a typical Biological Neuron. [52]

In parallel to the biological structure, a typical example of an artificial neuron is depicted in Figure 4.2. In the computational model, the signals that travel along the "axons" (e.g. x_0) interact multiplicatively (e.g. $w_0 x_0$) with the "dendrites" of the other neuron based on the "synaptic strength" at that "synapse" (e.g. w_0). A "synapse" (connection) between artificial neurons is called an edge and the "synaptic strength" of that edge is called weight. The weight changes during the training procedure and increases or decreases the strength of the signal at the connection. Each connection can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number.

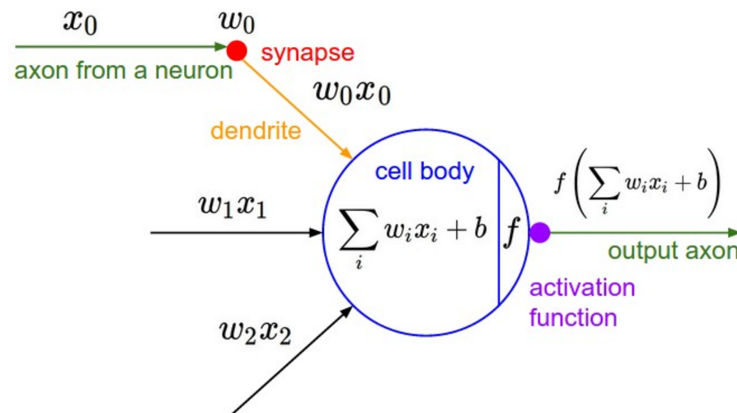

Figure 4.2: Computation Model of a Neuron. [53]

The output of an artificial neuron is computed by summing the multiplication of each input by the corresponding edge weight and then adding a bias term b to this sum. This weighted sum is sometimes called the activation. Neurons may have a threshold such that a signal is sent (the neuron "fires") only if the aggregate signal crosses that threshold. In contrast to the biological model, the precise timings of the "spikes" do not matter, and only the frequency of the "firing" communicates information. Based on this rate code interpretation, the firing rate of the neuron is modeled with a function f, called activation function, which represents the frequency of the spikes along the axon. Simply put, the aggregate signal is passed through an activation function and generates an output. In summary, each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function).

4.1.2 Organization

Neurons are connected to each other in various patterns, to allow the output of some neurons to become the input of others. The network forms a directed, weighted graph. Typically, neural networks consist of an input and an output layer, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use. Different layers may have a different number of neurons and perform a different transformation on their inputs. Signals are fed into the network through the input layer and travel to the output layer, possibly after traversing the layers multiple times.

Between two layers, multiple connection patterns are possible. They can be fully connected, with every neuron in one layer connecting to every neuron in the next layer. They can be pooling, where a group of neurons in one layer connect to a single neuron in the next layer, thereby reducing the number of neurons in that layer. Neurons with only such connections form a directed acyclic graph and are known as Feed-Forward Neural Network (FFNN).

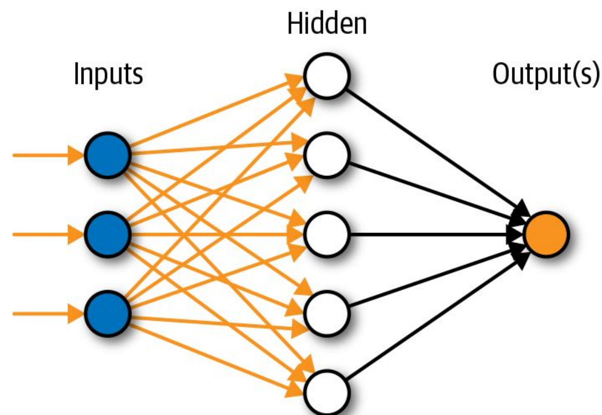

Figure 4.3: Overview of a typical Fully Connected Feed-Forward Artificial Neural Network. [54]

FFNNs are used in general regression and classification problems. The structure of a fully connected FFNN with a hidden layer is shown in Figure 4.3. Alternatively, networks that allow connections between neurons in the same or previous layers are known as recurrent networks. In the case of multiple hidden layers, the network is referred to as a Deep Neural Network (DNN).

4.1.3 Learning

An ANN is a complex adaptive system that can change its internal structure based on the information passing through it and the change in environment. Being a complex adaptive system, learning in ANNs implies the adaptation of the network to better handle a task by considering sample observations. Learning involves adjusting the parameters of the network (weights, biases and optional thresholds) in order to change its input/output behavior and improve the accuracy of the result. This is done by minimizing the observed errors.

Learning is complete when examining additional observations does not usefully reduce the error rate. Even after learning, the error rate typically does not reach

0 and, in case it is too high, the network must be redesigned. Practically this is done by defining a cost function that is evaluated periodically during learning. As mentioned in Section 3.9, an optimization algorithm is used in order to minimize the defined cost function by adjusting the hyper-parameters of the network. Examples of hyper-parameters include the learning rate, the number of epochs, the number of hidden layers and the mini-batch size.

4.1.4 Back-propagation

Back-propagation (backprop) is a widely used algorithm for training (feed-forward) neural networks. Generalizations of backprop exists for other ANNs, and for functions generally. These classes of algorithms are all referred to generically as "back-propagation".

When the neural network is initialized, weights are set for its neurons. Inputs are loaded, they are passed through the network of neurons, and the network provides an output for each one, given the initial weights. Back-propagation helps to adjust the connection weights of the neurons to compensate for each error found during learning. The error amount is effectively divided among the connections. Technically, backprop computes the gradient of the loss function with respect to each weight by the chain rule, , computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule. The weight updates to minimize loss can be done via gradient descent, stochastic gradient descent or other methods.

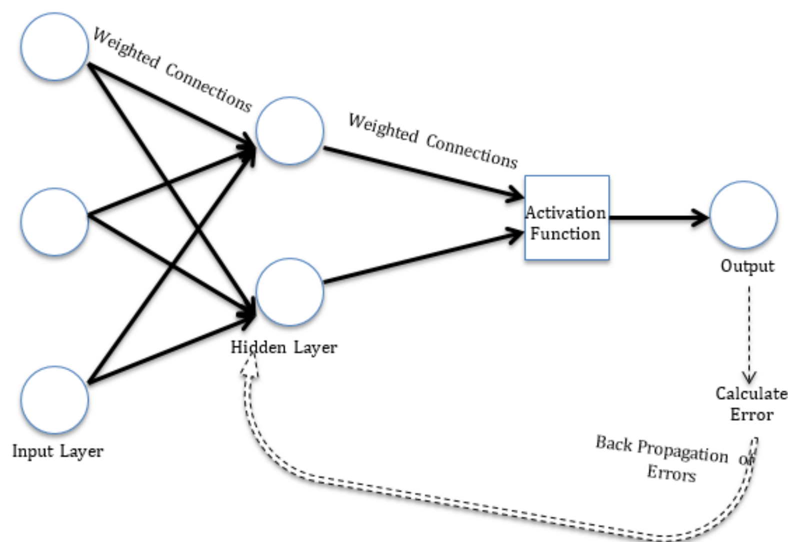

Figure 4.4: A simple Neural Network Model. [55]

4.1.5 Activation Function

In ANNs, the activation function of a node is a mathematical equation that determines the output of that node given an input or set of inputs. The function is attached to each neuron in the network, and determines whether it should be activated or not, based on whether each neuron's input is relevant for the model's prediction. Activation functions also help normalize the output of each neuron to

a range between 1 and 0 or between −1 and 1. Increasingly, NNs use non-linear activation functions, which can help the network learn complex data, compute and learn almost any function representing a question, and provide accurate predictions. An additional aspect of these functions is that they must be computationally efficient because they are calculated across a large number of neurons for each data sample. Some of the most popular activation functions are presented next.

4.1.5.1 Binary Step Function

The binary step function, depicted in Figure 4.5, is a threshold-based classifier, i.e whether or not the neuron should be activated based on the value from the linear transformation. In other words, if the input value is greater than a threshold, then the neuron is activated and the signal is sent to the next layer without any transformation. Otherwise, the neuron is deactivated and its output is not considered for the next layer. The binary step function has the mathematical form:

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{4.1}$$

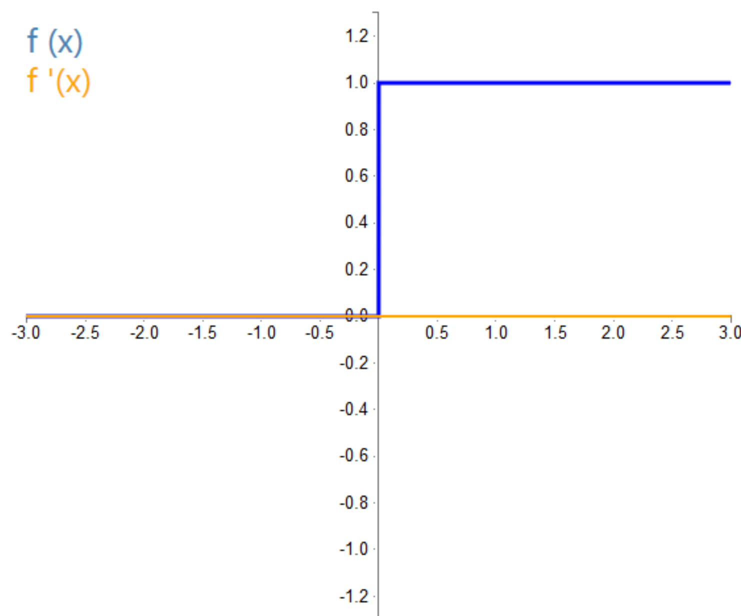

Figure 4.5: Binary Step Activation Function. [56]

One of the limitations of this function is that its output only takes two values, so it cannot support classifying the inputs into one of multiple categories. Moreover, the gradient of the step function is zero which causes a problem in the back propagation process, as the weights and biases don't update.

4.1.5.2 Linear Function

The linear activation function, shown in Figure 4.6, takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. The linear function has the mathematical form:

$$f(x) = x \tag{4.2}$$

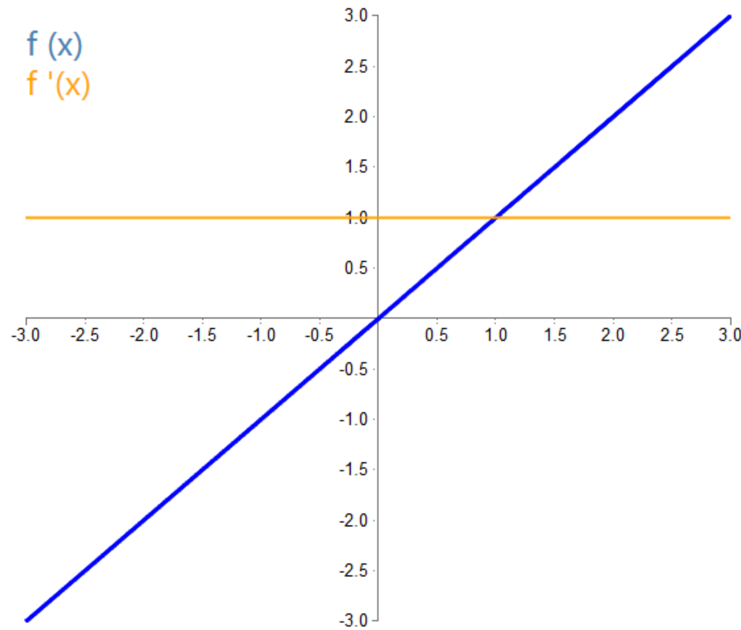

Figure 4.6: Linear Activation Function. [56]

A linear function is better than a step function, because it allows multiple outputs. However, it has two major problems. Firstly, the derivative of the function is a constant, and has no relation to the input, x, so it's not possible to use backprop (gradient descent) to train the model. Secondly, a neural network with a linear activation function is simply a linear regression model. No matter how many layers the network has, the last layer will be a linear function of the first layer. This results in the network having limited power and ability to handle complexity varying parameters of input data.

4.1.5.3 Sigmoid Function

The logistic sigmoid function, shown in Figure 4.7, is a non-linear mathematical function having a characteristic "S"-shaped curve or sigmoid curve. It takes a real-valued input (the signal strength after the sum) and squashes it to range between 0 and 1. Thus, it is used especially when a model has to predict the probability as an output. The sigmoid non-linearity is defined for all real input values by the formula:

$$f(x) = \sigma(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}} \tag{4.3}$$

However, the sigmoid function has some disadvantages. Towards either end of the sigmoid function, for very high or very low values of x, there is almost no change to the prediction Y. The gradient at that region is going to be small or "vanished" (almost zero), causing a problem of "vanishing gradients". This can result in the network being too slow to reach an accurate prediction, or refusing to learn further. Furthermore, the function is computationally expensive, and its outputs are not zero-centered. This is a major drawback, since the data coming into a neuron is always positive, causing the gradient of the weights to either all be positive, or all negative and undesirable zig-zagging dynamics would be introduced to the network.

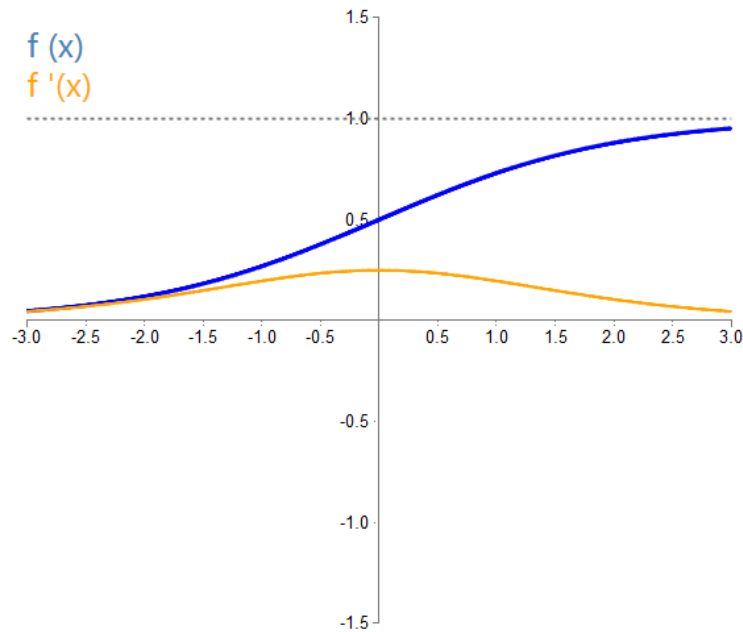

Figure 4.7: Sigmoid Activation Function. [56]

4.1.5.4 Hyperbolic Tangent Function

The hyperbolic tangent (*tanh*) non-linearity is shown in Figure 4.8. As an activation function, tanh it is mostly used to model inputs that have strongly negative and positive values as it is zero-centered. Its outputs range between -1 and 1, making it easier to model inputs that have strongly negative, neutral, and strongly positive values. The zero-centered values help the next neuron during propagating. The *tanh* activation function is continuous and differentiable at all points and follows the function:

$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.4}$$

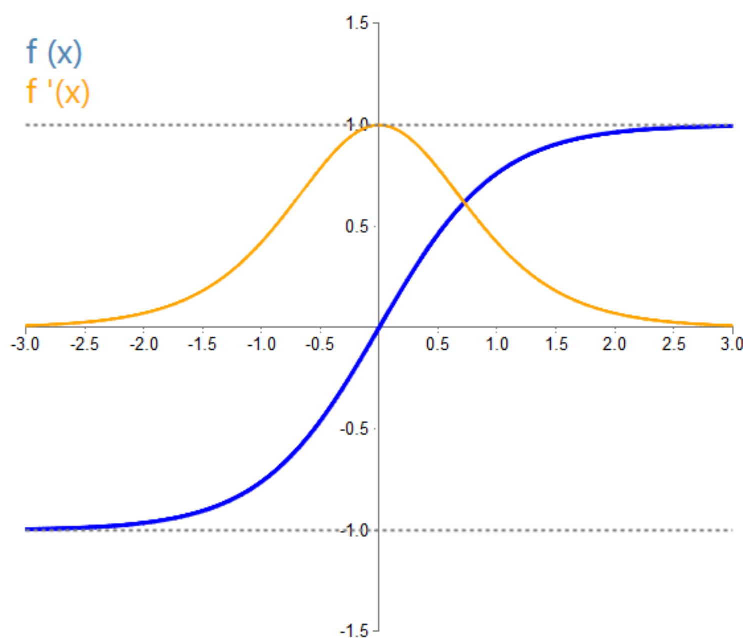

Figure 4.8: Tanh Activation Function. [56]

Tanh function is basically a rescaled sigmoid function, where

$$tanh(x) = 2\sigma(2x) - 1 \tag{4.5}$$

The gradient of the *tanh* function is steeper as compared to the sigmoid function. Deciding between the sigmoid or *tanh* depends on the requirement of gradient strength, but usually *tanh* is preferred over the sigmoid function since it is symmetric around the origin and the gradients are not restricted to move in a certain direction. However, like sigmoid, *tanh* also has the vanishing gradient problem.

4.1.5.5 Rectified Linear Unit

Rectified Linear Unit (ReLU), also known as ramp function, is a non-linear function and one of the most widely used activation functions. It is a simple calculation that returns the value of the input x, or 0 if the input value is negative. ReLU, depicted in Figure 4.9, computes the function:

$$f(x) = x^{+} = max(0, x) \tag{4.6}$$

where x is the input to a neuron. ReLU is less computationally expensive than *tanh* and sigmoid and allows the network to converge very quickly. The main advantage of using the ReLU function however is that it does not activate all the neurons at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.

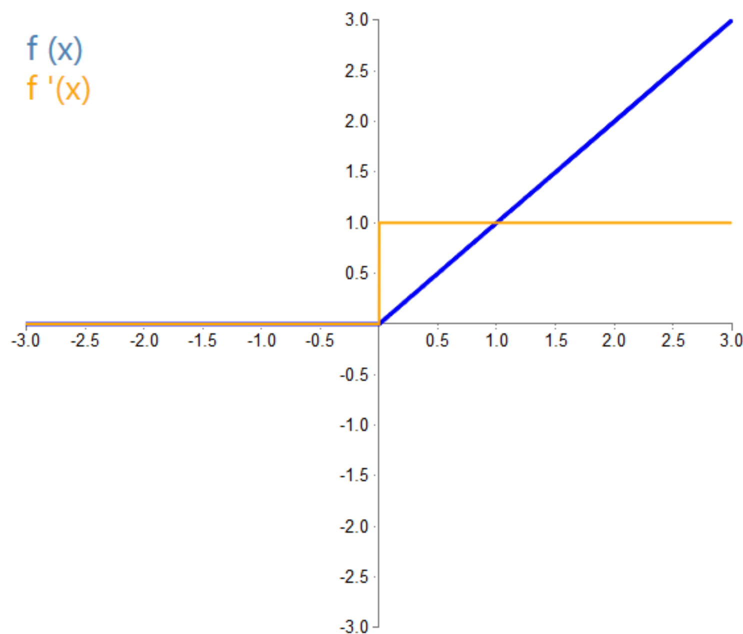

Figure 4.9: ReLU Activation Function. [56]

The range of ReLU is $[0, inf)$, meaning it is not bound, and it can blow up the activation. Also, it is not zero-centered. The main disadvantage, however, of the ReLU function is a problem called "dying ReLU problem", which is a form of the vanishing gradient problem. When inputs approach zero, or are negative, the gradient of the function becomes zero. Due to this reason, during the backprop process, the weights and biases for some neurons are not updated. This can create

"dead" neurons which never get activated, making a substantial part of the network passive. This problem typically arises when the learning rate is set too high. It can be taken care of by the 'Leaky' ReLU function, which simply assigns a small positive slope for $x < 0$, however the performance is reduced. There are other variations too, but the main idea is to let the gradient be non zero and recover during training eventually.

4.1.5.6 Softmax

The softmax function, also known as softargmax function, is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over multiple predicted output classes. The function turns a vector of K real values into a vector of K real values that sum to 1. The input values can be positive, negative, zero, or greater than one, but the softmax transforms them into values between 0 and 1, so that they can be interpreted as probabilities. Basically, it normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class. The standard softmax function is defined by the formula:

$$f(x)_i = \frac{e^{x_i}}{\sum_{n=1}^{K} e^{x_n}} \tag{4.7}$$

4.1.6 Regularization

When training a machine learning model, and especially a neural network, there are two major kinds of problems that may arise: bias and variance problems. The term bias refers to the simplifying assumptions made by a model to make the target function easier to learn. Variance is the amount that the estimate of the target function will change if different training data was used. The goal of any supervised ML algorithm is to achieve low bias and low variance.

The high bias and low variance problems are called under-fitting of the model and the high variance and low bias problems are called over-fitting of the model. In the case of under-fitting, the network is not trained properly and results in high training and test errors. In the case of over-fitting, the network is trained in such a way that it has adapted for the training values but when given unseen data, the model performs poorly and gives errors in the result.

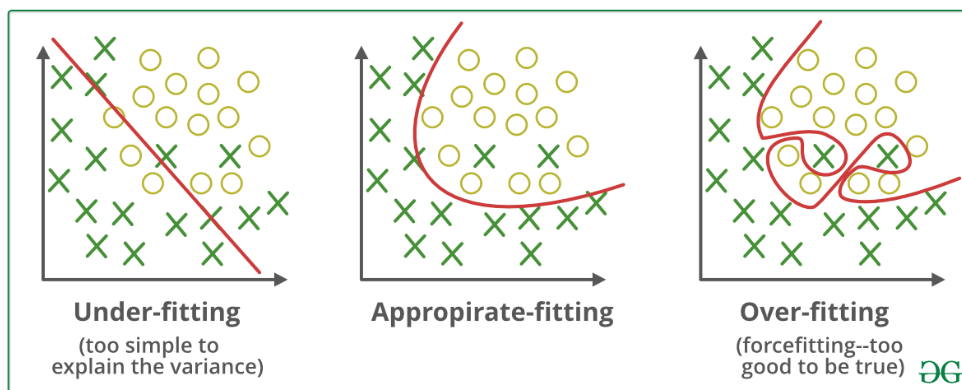

Figure 4.10: Under-fitting vs. Appropriate-fitting vs. Over-fitting. [57]

The collection of techniques used to combat over-fitting and reduce the error on the test set is known as Regularization. Regularization can be defined as any modification made to a learning algorithm that is intended to reduce its generalization error and, ideally, not reduce its training error. This is often achieved by adding an additional penalty term in the error function. Some of the most commonly used regularization techniques are presented next.

4.1.6.1 L1 Regularization

L1 Regularization adds an L1 penalty equal to the absolute value of the magnitude of coefficients. In other words, it limits the size of the coefficients. L1 can yield sparse models, i.e. models with few coefficients. It tends to shrink coefficients to zero and of them can become zero and eliminated. L1 is therefore useful for feature selection, as any variables associated with coefficients that go to zero can be dropped.

4.1.6.2 L2 Regularization

L2 Regularization adds to the loss function an L2 penalty equal to the square of the magnitude of coefficients. It will not yield sparse models and all coefficients are shrunk by the same factor (none are eliminated). L2 is useful when the features are codependent. This technique is also known as weight decay since it reduces the magnitudes of neural network weights during training, resulting in an improved trained model that is not as likely to be over-fitted.

4.1.6.3 Dropout

Dropout [58] is a regularization technique for reducing over-fitting in neural networks by preventing complex co-adaptations on training data. The idea is that it randomly selects a number of neurons which are “dropped-out”, or ignored along with their incoming and outgoing connections during training. This means that, in each weight update cycle, the contribution of the selected ignored nodes to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.

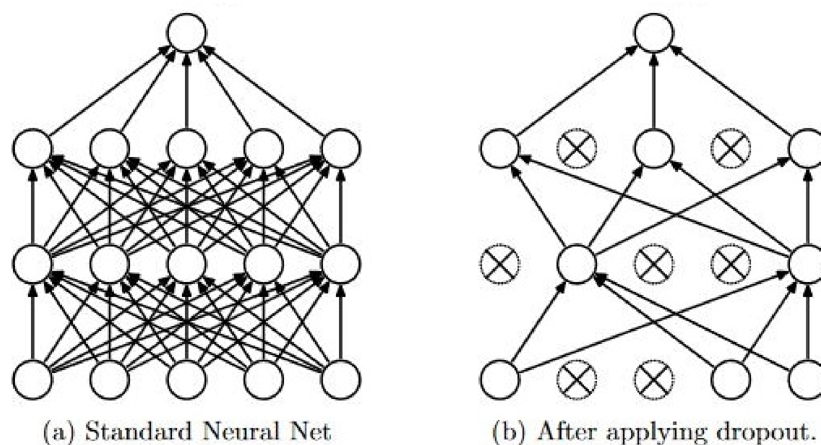

Figure 4.11: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. [58]

Dropout can be easily implemented by randomly selecting nodes to be dropped-out with a given probability (defined by a dropout hyper-parameter) each iteration. This way, each iteration consists of a different set of nodes that produce different outputs. It is only used during the training of the model, and is not used when evaluating the model. A visualization of this regularization technique is shown in Figure 4.11.

4.1.6.4 Data Augmentation

Data Augmentation is an interesting regularization technique that is mostly used when the data set available is small. This technique generates new (fake) training data from the original data set, increasing the number of observations and thus controlling over-fitting. More specifically, it increases the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It is a cheap and easy way to increase the amount of training data.

4.1.6.5 Early Stopping

One of the biggest problem in training neural networks, especially dealing with large data sets, is how long to train the model. Training too little will lead to under-fit in train and test sets and training too much will result in over-fitting the training set and give poor results in test sets. Thus, the challenge is to train the network long enough that it is capable of learning the mapping from inputs to outputs, but not training the model so long that it over-fits the training data.

One technique to prevent over-fitting is to use validation error to decide when to stop the training. The validation error is evaluated on a holdout data set, called validation set, after each epoch while building the model. If the accuracy of the model on the validation set starts to degrade (loss begins to increase or accuracy begins to decrease), then the training process is stopped. This technique is called Early Stopping. In order to denote the number of epochs after which the training will stop if no further improvement is made, a hyper-parameter called patience is defined.

4.2 Definition of Deep Learning

Deep Learning (DL), also known as deep structured learning, is part of a broader sector of ML based on artificial neural networks with representation learning. According to Jeff Dean [59], deep learning algorithms are using very deep neural networks, where "deep" refers to the number of layers in the neural network, or iterations between input and output. DL can be supervised, semi-supervised or unsupervised. DL architectures have been applied to numerous fields of study including computer vision, speech recognition, natural language processing, audio recognition, machine translation, time series prediction and bioinformatics.

4.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a generalization of feed-forward neural networks that has an internal "memory". It is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence, which allows it to exhibit temporal dynamic behavior. While traditional deep neural networks assume that inputs and outputs are independent of each other, RNNs consider the current input and the output that it has learned from the previous input for making a decision. In other words, they are networks with feedback loops that allow information to persist - a trait that is analogous to short-term memory. Unlike a feed-forward network, depending on the preceding inputs the same input may produce different outputs.

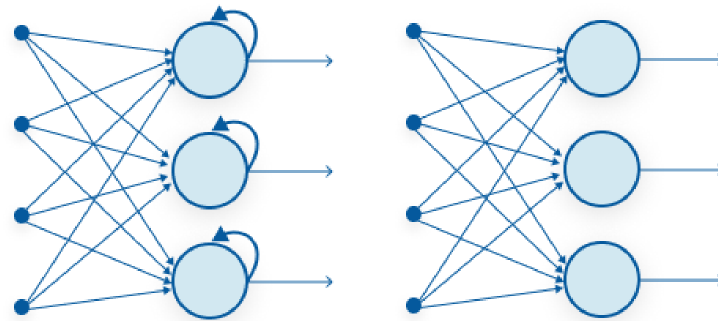

Figure 4.12: Comparison of Recurrent Neural Networks (on the left) and Feed-forward Neural Networks (on the right). [60]

The core reason that RNNs are important is that, because of their internal state (memory), they allow the operation over sequences of vectors: sequences in the input, the output, or in the most general case both. They are perfectly suited for machine learning problems that involve text, audio, video, and time series. The four different types of RNNs are depicted in Figure 4.13.

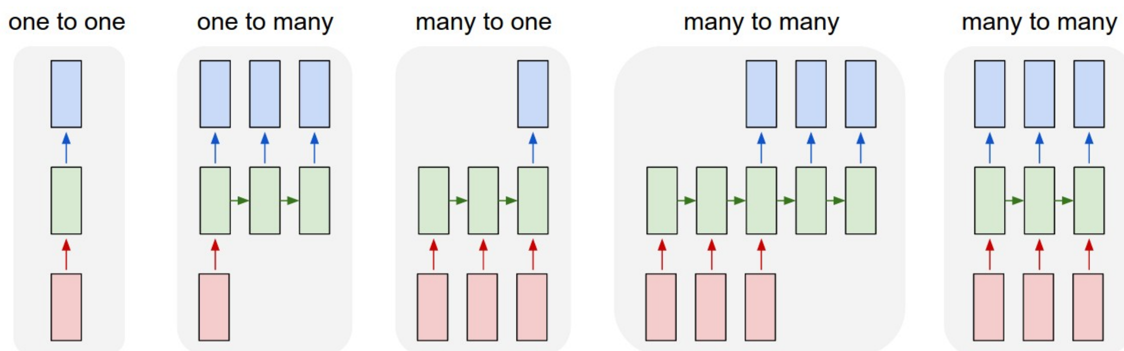

Figure 4.13: Each rectangle is a vector and arrows represent functions. Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state. From left to right: (1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output. (2) Sequence output. (3) Sequence input. (4) Sequence input and sequence output. (5) Synced sequence input and output. Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like. [61]

4.3.1 Uni-Directional Recurrent Neural Networks

Like feed-forward neural networks, recurrent neural networks utilize training data to learn. They are distinguished by their "memory", which is preserved in the recurrent network's hidden state vector and represents the context based on the prior inputs and outputs. While future events would also be helpful in determining the output of a given sequence, uni-directional (vanilla) recurrent neural networks cannot account for these events in their predictions.

An unrolled one-to-many vanilla recurrent neural network is depicted in Figure 4.14. Firstly, during forward propagation, the RNN takes the x_0 from the sequence of input and it produces an output h_0 (hidden state). Note that the production of hidden state is simply a matrix multiplication of input and hidden state by some weights W. In the next time-step, the hidden state h_0 along with the next input x_1 is given as input. Accordingly, h_1 along with x_2 is the input for the next step and so on. So, the "memory" mechanism of RNNs is implemented using an internal hidden layer that produces a hidden state, which remembers all information about what has been calculated. Formally, at each time-step t, the hidden state h_t and the output y_t are calculated as:

$$h_t = f_h(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{4.8}$$

$$y_t = f_y(W_{yh}h_t + b_y) \tag{4.9}$$

where x_t is the input vector, b_h is the bias for h, by is the bias for y and f_x, f_h are the activation functions for x and h respectively. Also, W_{hh} represents the weight at the previous hidden state, W_{hx} the weight at current input state and W_{yh} the weight at the output state.

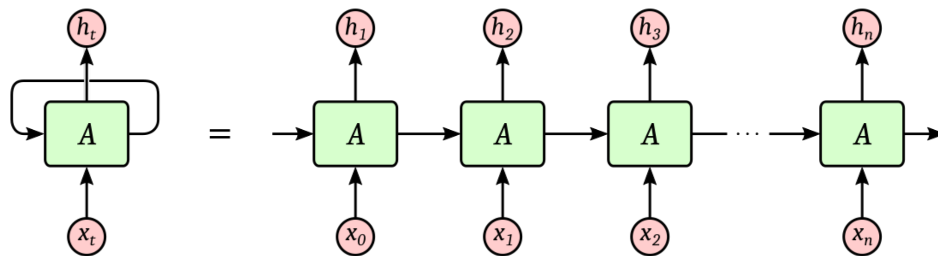

Figure 4.14: A simple Recurrent Neural Network Unrolled in time. All RNNs have the form of a chain of repeating modules of a neural network (green box). In vanilla RNNs, this repeating module will have a very simple structure, such as a single *tanh* layer. The arrows indicate memory or simply feedback to the next input. [62]

RNNs leverage the Back-propagation Through Time (BPTT) algorithm to determine the gradients. This algorithm is different from the traditional backprop described in Section 4.1.4, as it is specific to sequence data. The principles of BPTT are the same as traditional backprop, where the model trains itself by calculating errors from its output layer to its input layer in order to adjust and fit the parameters of the model appropriately. However, BPTT differs because it sums errors at each time-step, whereas feed-forward networks do not need to sum errors as they do not share parameters across each layer.

Through this process, RNNs tend to run into two major problems, known as vanishing gradients and exploding gradients. These issues are defined by the size of the gradient (the slope of the loss function along the error curve). When the gradient is small, it continues to become smaller, updating the weight parameters until they become insignificant. This makes the learning of long data sequences difficult. In contrast, exploding gradients occur when the the slope tends to grow exponentially instead of decaying, creating an unstable model. In this case, the weight parameters will grow too large (approaching infinity) during the training process resulting in poor performance. One solution to these issues is to reduce the number of hidden layers within the RNN, eliminating some of its complexity.

4.3.2 Bi-Directional Recurrent Neural Networks

Bi-directional Recurrent Neural Network (BRNN) [63] is a variant network architecture of RNNs and was introduced in order to increase the amount of input information available to the network. While uni-directional RNNs can only draw information from previous inputs (backwards) to make predictions about the current state, BRNNs pull in future data (forward) simultaneously to improve the accuracy of it.

The structure of a typical BRNN is illustrated in Figure 4.15. BRNNs split the neurons of a regular RNN into two directions, one for positive time direction (forward states), and another for negative time direction (backward states). Those two states' output are not connected to inputs of the opposite direction states. In order to find the hidden state for each time step, BRNNs combine the two hidden layers of opposite directions to the same output.

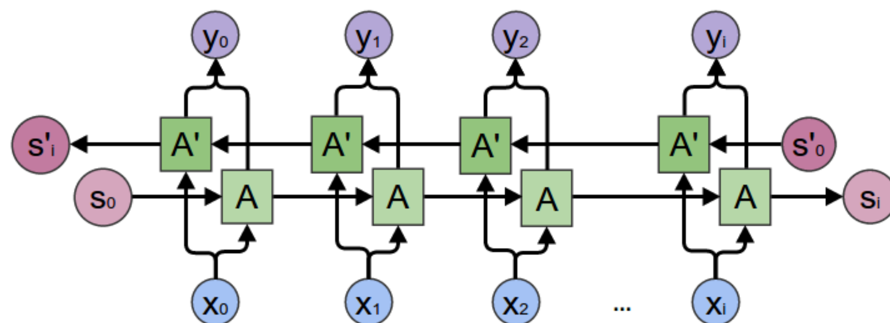

Figure 4.15: A Bi-directional Recurrent Neural Network. [64]

BRNNs can be trained using similar algorithms to RNNs, because the two directional neurons do not have any interactions. However, when BPTT is applied, additional processes are needed because updating input and output layers cannot be done at once. Generally, for forward pass, forward states and backward states are passed first, then output neurons are passed. For backward pass, output neurons are passed first, then forward states and backward states are passed next. After forward and backward passes are done, the weights are updated.

4.3.3 Long Short-Term Memory Networks

The Long Short-Term Memory (LSTM) [65] is a popular RNN architecture used in the field of deep learning. It was introduced as a solution to the vanishing gradient problem in back-propagation, which cause RNNs to be inherently deficient at retaining information over long periods of time. More specifically, if the previous state that is influencing the current prediction is not in the recent past, the RNN model may not be able to accurately predict the current state. LSTMs can overcome this problem by preserving key information, retain it over long periods of time, and then use this information when necessary much later on in the sequence.

The chain like structure of an LSTM is depicted in Figure 4.16. It is composed of "cells" in the hidden layers, which have three gates: a forget gate, an input gate and an output gate. A cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. At every time-step t, each LSTM cell updates its internal state vector c_t and generates an output hidden vector h_t based on the cell state. The equations for the forward pass of an LSTM unit with a forget gate are described next.

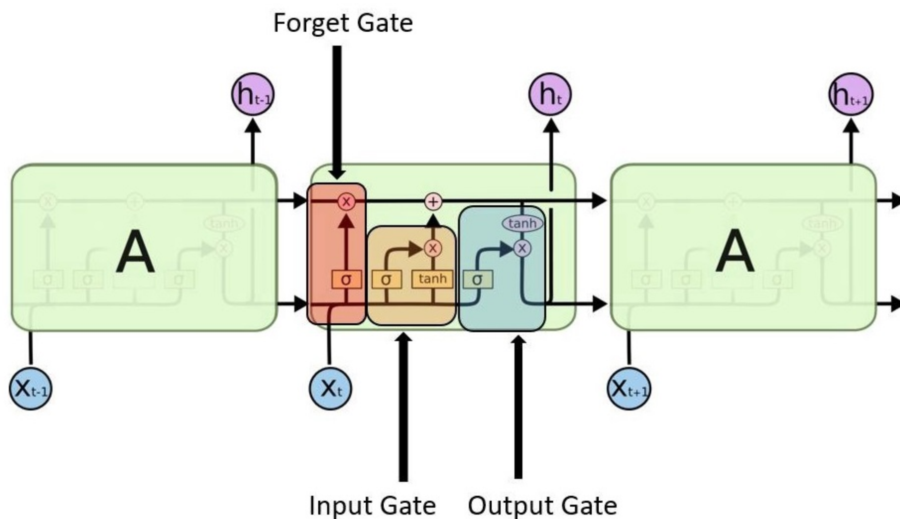

Figure 4.16: Long Short-Term Memory Chain. The repeating module in an LSTM contains four interacting layers. [66]

- Forget gate: It decides which non-important information to delete from the cell state. The current input x_t and the previous hidden state h_{t-1} are passed through the sigmoid function, which outputs a number between 0 (completely forget this) and 1 (completely keep this) for each bit in the cell state c_{t-1}. Formally, the forget gate's activation vector is computed as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{4.10}$$

- Input/Update gate: It controls what information should be stored in the cell state. First, the sigmoid function (called the input gate layer) decides which values to let through by squashing them between 0 and 1 as follows:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{4.11}$$

  where i_t is the input gate's activation vector. Next, the previous hidden state and the current input are also passed to the *tanh* function, which forces the values

to be between −1 and 1, deciding their level of significance. Mathematically, the vector of new candidate values that could be added to the state, called the cell input activation vector, is defined as:

$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{4.12}$$

By element-wise multiplying the *tanh* output with the sigmoid output, the important information of the current input is filtered.

- Cell state: The old state c_{t-1} gets element-wise multiplied by the forget vector in order to forget the information decided by the previous steps. The output of this multiplication is then added to the output vector from the input gate ($i_t \circ \tilde{c}_t$) in order for the state to be updated. The cell state c_t now contains the new values that the neural network considers as relevant.

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \tag{4.13}$$

- Output gate: Finally, this gate decides the output based on the input and the memory of the block. The sigmoid function once again squashes the previous hidden state and the current input between 0 and 1 in order to decide which values to let through. So, the output gate's activation vector is computed as:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{4.14}$$

The updated cell state c_t is passed to the *tanh* function, and its output is element-wise multiplied with the sigmoid output o_t. This multiplication generates the new hidden state h_t which is carried over to the next time-step:

$$h_t = o_t \circ tanh(c_t) \tag{4.15}$$

Note that W, U and b are the weight matrices and bias vector parameters which need to be learned during training. Also, the initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard product (element-wise product).

4.4 Attention Mechanisms

Attention [67] is a mechanism that was developed as a solution to the limitations of the Encoder-Decoder model for RNNs [68] [69] in machine translation, where input sequences differ in length from output sequences. Specifically, it helps memorizing long source sentences in neural machine translation.

Attention is a technique that mimics cognitive attention, and it is one of the most important breakthroughs in DL research in the last decade. It helps models to direct their focus and devote more computing power to the important parts of the input data, while fading out the rest. Which part of the data is more important depends on the context and is learned through training data by GD. Therefore, attention in deep learning can be broadly interpreted as a vector of importance weights. Attention mechanisms are extensively used in Transformer networks [70], illustrated in Figure 4.17, in order for them to achieve their expressive power.

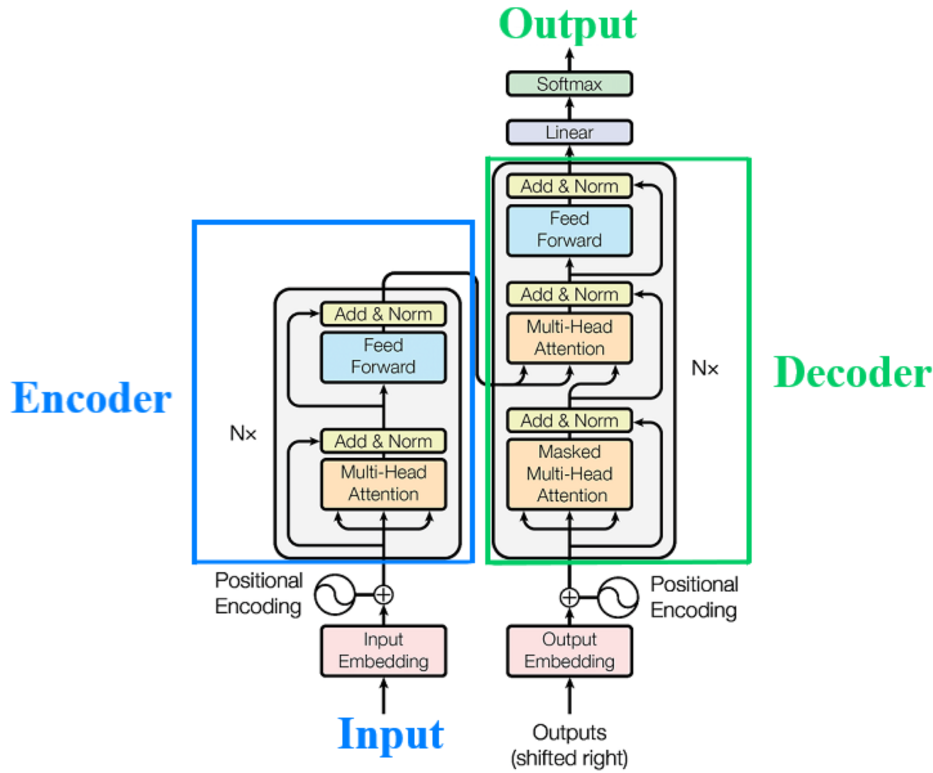

Figure 4.17: The Transformer - model architecture. [70]

The attention mechanism introduced in neural machine translation [67] can be defined as follows. Suppose a source sequence x of length n, $x = [x_1, x_2, \dots, x_n]$, and an output sequence y of length m, $y = [y_1, y_2, \dots, y_m]$. Also, suppose the encoder is a BRNN with both a forward and a backward hidden state. A simple concatenation of the two represents the encoder state h_i, $i = 1, \dots, n$. Both the preceding and following words should be included in the annotation of one word. For the output word at position t, $t = 1, \dots, m$, the decoder network has a hidden state:

$$s_t = f(s_{t-1}, y_{t-1}, c_t) \tag{4.16}$$

where the context vector c_t is a weighted by alignment scores sum of the hidden states of the input sequence:

$$c_t = \sum_{i=1}^{n} a_{t,i} h_i \tag{4.17}$$

$$a_{t,i} = \text{align}(y_t, x_i) = \frac{e^{score(s_{t-1}, h_i)}}{\sum_{i'=1}^{n} e^{score(s_{t-1}, h_{i'})}} \tag{4.18}$$

The alignment model assigns a score $a_{t,i}$ to the pair (y_t, x_i), based on how well they match. The set of $\{a_{t,i}\}$ are weights, and they define how much of each source hidden state should be considered for each output.

Some of the most common attention mechanisms used are the additive [67], the dot-product [71], and the content-based attention [72]. Also, attention mechanisms can be categorized into broader categories, such as self-attention [73]. Self-attention, also known as intra-attention, is a mechanism relating different positions of a single sentence in order to compute a representation of the same sentence. Finally, note that the multi-head attention is a module which runs through an attention mechanism several times in parallel.

4.5 Transfer Learning

Transfer Learning (TL) is a machine learning technique where a model trained on one task is reused as the starting point for a model on a second task. It is a popular approach especially in deep learning because of the vast compute and time resources required to develop neural networks for tasks like computer vision and natural language processing. Also, it allows the development of deep neural network models with comparatively little data. This is really useful as the supervised models which solve complex problems require vast amounts of labeled data that typically cannot be obtained due to the time and effort it takes to label data points. Moreover, most models that are specialized to a specific task suffer a significant performance drop when used in new tasks which might still be similar to the one they were trained on. The goal of TL is to improve target task performance/results by leveraging knowledge from the source task, as illustrated in Figure 4.18.

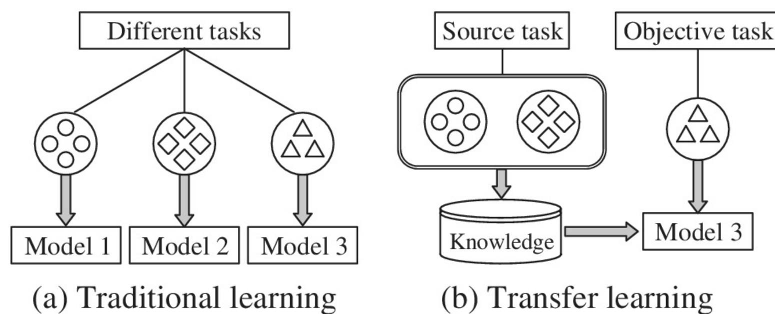

Figure 4.18: Traditional Learning vs. Transfer Learning. [74]

In order to define transfer learning, the terms of domains and tasks are used. According to the survey by Weiss et al. [75], a domain $\mathcal{D}$ consists of two parts: a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. For a given domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task $\mathcal{T}$ is defined by two components: a label space $\mathcal{Y}$ and an objective predictive function $f : \mathcal{X} \rightarrow \mathcal{Y}$, which is learned from the training data consisting of label pairs $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in \mathcal{Y}$. So, $\mathcal{T} = \{\mathcal{Y}, f(x)\}$. Given a source domain $\mathcal{D}_S$ with corresponding task $\mathcal{T}_S$ and a target domain $\mathcal{D}_T$ with learning task $\mathcal{T}_T$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$, transfer learning's objective is to improve the target predictive function $f_T(\cdot)$ in $\mathcal{T}_T$ by using the related information from $\mathcal{D}_S$ and $\mathcal{T}_S$. As both the domain $\mathcal{D}$ and the task $\mathcal{T}$ are defined as tuples, these inequalities give rise to four transfer learning scenarios:

- $\mathcal{X}_S \neq \mathcal{X}_T$: The feature spaces of the source and target domains are not equal.
- $P(X_S) \neq P(X_T)$: The marginal probability distributions of source and target domains are not equal.
- $\mathcal{Y}_S \neq \mathcal{Y}_T$: The label spaces of the source and the target tasks are not equal.
- $P(Y_S|X_S) \neq P(Y_T|X_T)$: The conditional probability distributions of the source and the target tasks are not equal.

Pan and Yang [76] segregate TL mainly into transductive TL, where the source and target task are the same, and inductive TL, where the source and the target task is different. It is further divided into domain adaption (data from different domains), cross-lingual learning (data from different languages), multi-task learning (several tasks are learnt simultaneously) and sequential TL (the source data's general

knowledge is transferred to only one task), as shown in Figure 4.19.

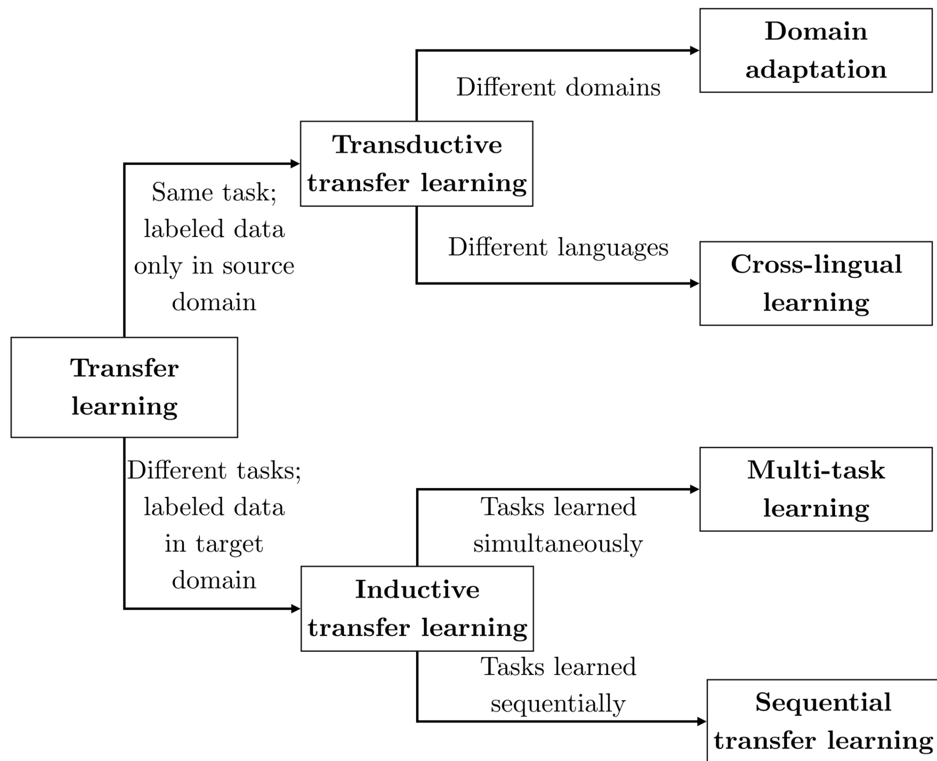

Figure 4.19: Types of Transfer Learning. [77]

Chapter 5

Natural Language Processing

5.1 Definition of Natural Language Processing

Natural Language Processing (NLP) is a branch of AI, computer science, and computational linguistics (rule-based modeling of human language) that is primarily concerned with the interactions between computers and natural (human) languages. In simple terms, NLP represents the automatic processing of natural language like speech (voice data) or text in order for it to be understood, interpreted and manipulated. By utilizing NLP, machines can perform tasks such as automatic text summarization, sentiment analysis, speech recognition, translation, relationship extraction, topic segmentation, named entity recognition, and more.

The main challenge that NLP faces is that human language is filled with ambiguities. Besides the existence of many different languages and dialects, each language include irregularities in grammar and syntax, idioms, sarcasm, metaphors, homonyms, and homophones. So, understanding and manipulating language is extremely complex, which makes it difficult to write software that accurately determines the intended meaning of text or voice data. For this reason, it is common to use algorithms like tokenization, stop words removal, stemming, lemmatization, topic modeling, etc., described in Section 5.3.

5.2 Sentiment Analysis

Sentiment analysis, also called opinion mining or emotion AI, is a type of text research or text mining. It applies a mix of NLP, text analysis, computational linguistics, and biometrics to identify and extract subjective information (people's opinions, sentiments, emotions, evaluations, etc.) towards entities such as products, events, topics, and services. In involves classifying feelings in text into categories such as "positive", "negative" or "neutral". Consequently, this task can be consider as a text classification problem that aims at categorizing a text based on the sentiment that the text contains. Sentiment analysis in text bodies can be applied on the following different levels:

- Document level sentiment analysis: Sentiment is extracted from the whole document, and a whole opinion is classified based on the overall sentiment of the opinion holder. The goal is to classify opinion document into a positive, negative or neutral sentiment.

- Sentence level sentiment analysis: It is associated with a phrase or sentence. It determines whether each sentence expresses a positive, negative or neutral opinion for a product or service.
- Aspect based sentiment analysis: It is the opinion mining and summarization based on aspect/feature in a review. The goal is to identify and extract object features from the source data and determine whether the opinion is positive, negative, or neutral.

5.3 Text Pre-processing

In order to develop a NLP application, usually a huge amount of text data is needed. A large and structured collection of machine-readable texts that have been produced by written or spoken natural language material is called a corpus (plural corpora). Therefore, it is crucial to prepare and transform the text data into a standard format that can be easily processed by algorithms.

5.3.1 Tokenization

Tokenization is a method in which a piece of text is segmented into smaller units, or "tokens". Tokens are the building blocks of natural language, and can be either characters, words, or subwords. Hence, tokenization can be classified into three types: character, word, and subword (n-gram characters) tokenization.

Tokenization can also discard certain unnecessary characters, such as punctuation, easing the path to a proper word segmentation but also triggering possible complications. For example, when dealing with biomedical text domains which contain many punctuation marks, tokenization can be particularly problematic.

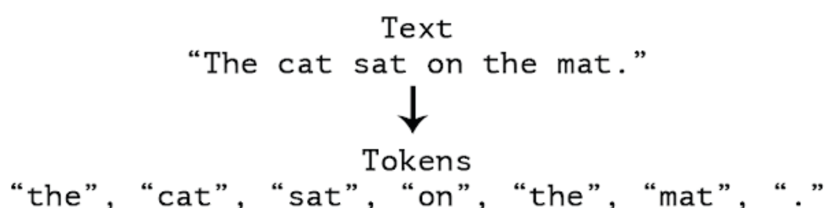

Figure 5.1: Word Tokenization Example. Tokenization can split a sentence written in English (a segmented language) into individual words by a blank space and also remove punctuation characters at the same time. [78]

The most common way of processing raw text happens at the token level so the first step while modeling text data is performing tokenization. In a NLP task, tokenization is performed on the corpus to obtain tokens, which are then used to prepare a vocabulary. Vocabulary refers to the set of unique tokens used in the text corpus, and can be constructed by considering either each unique token in the corpus or the top k frequently occurring words.

5.3.2 Lowercasing

Lowercasing all the text data, or the tokenized words, is one of the simplest and most effective form of text preprocessing. It is applicable to most text mining and

NLP tasks as it helps with consistency of expected output. For example, the words "NLP", "nlp", and "Nlp" are not treated as different words after using the lower casing.

5.3.3 Stop Word Removal

Stop words are the most common words in any natural language (articles, pronouns, prepositions, conjunctions, etc.). They are generally filtered and excluded before processing the text as they do not add much information to it. Examples of stop words in English include “and”, “the”, “to”, “a”, “an”, and “so”. Stop words can be safely removed by performing a lookup in a pre-defined list of keywords, reducing the data set size and improving processing time.

The removal of stop words is not always a good idea and it is highly dependent on the NLP task at hand. In tasks like text classification, stop words are not generally needed as the other words present in the data set are more important and give the general idea of the text. However, sentiment analysis tasks cannot be accomplished properly after the removal of certain stop words.

5.3.4 Stemming

Stemming is the process of reducing inflected (or sometimes derived) words to their word stem or root form through dropping unnecessary characters, usually affixes (lexical additions to the root of the word). Note that affixes that are attached at the beginning of the word are called prefixes and affixes that are attached at the end of the word are called suffixes. Related words are usually mapped to the same root form and thus are treated as synonyms by algorithms. However, the "root" in this case may not be identical to the morphological root of the words. Stemming uses a crude heuristic process that removes the ends of words hopping to correctly transform them into their base forms. For example, the words "trouble", "troubled" and "troubles" would be reduced to "troubl" instead of "trouble". A computer program or subroutine that stems words is called a stemmer. Stemmers are the way to go if speed and performance are important in the NLP model since they are simple to use and perform simple operations.

5.3.5 Lemmatization

Lemmatization is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word’s lemma (plural lemmas). For example, the verb "to walk", which may appear as "walk", "walked", "walks" or "walking", would be reduced to its canonical or dictionary form "walk", called the lemma of the word. Lemmatization is quite similar to stemming. However, unlike a stemmer that operates on a single word without knowledge of the context, lemmatization depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence. Since lemmatization requires more knowledge about the language structure than a stemming approach, it demands more computational power as well.

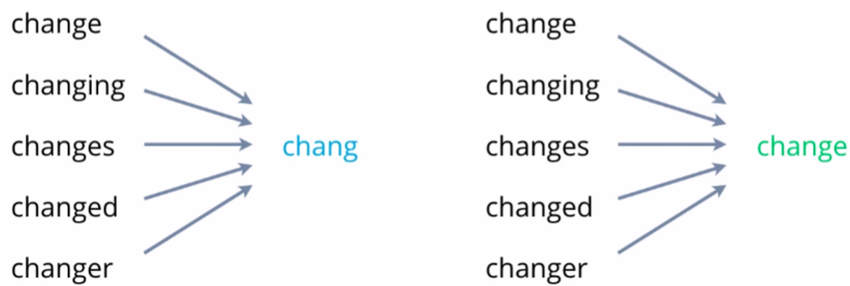

Figure 5.2: Stemming vs. Lemmatization. [79]

5.3.6 Topic Modeling

Topic modeling is a type of statistical model for discovering the hidden structures, or abstract "topics", that occur in a corpus. In essence, it is an unsupervised learning method that scans a set of documents, detects word and phrase patterns within them and groups together (clusters) word groups and similar expressions that best characterize this set.

This technique is based on the assumptions that each document is a mixture of multiple topics in different proportions and that each word's presence is attributable to one of the document's topics. This means that if these hidden topics can be spotted, the meaning of the text will become clear. Topic modeling is extremely useful for classifying texts, building recommendation systems or even detecting trends in online publications. Some popular topic models include Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA) and Term Frequency-Inverse Document Frequency (tf–idf).

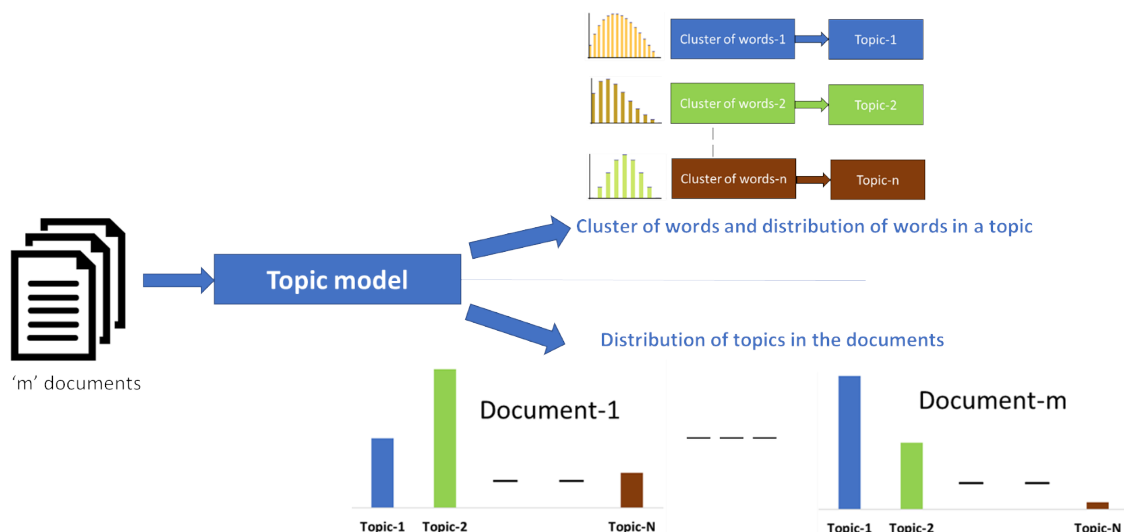

Figure 5.3: Schematic of Topic Modeling. [80]

5.3.6.1 Latent Dirichlet Allocation

The Latent Dirichlet Allocation (LDA) is the most commonly used topic modeling technique. The word 'Latent' indicates that the model identifies hidden topics within the documents. 'Dirichlet' denotes the assumption that the distribution of topics in

a document and the distribution of words in topics are both Dirichlet distributions. Lastly, the word 'Allocation' indicates the distribution of topics in the document.

LDA accepts documents as input and outputs topics. It assumes that each document is produced from a mixture of topics, and each topic from a mixture of words. Also, it assumes that every piece of text that is fed into the algorithm will contain words that are related. In particular, in order for the algorithm to find groups of related words, firstly the number of topics wished to be uncovered is being defined. LDA will assign all documents to the topics in a way that the words in each document are captured by those topics. It then reassigns iteratively each word to a topic by taking into consideration the probability that it belongs to a topic, and the probability that the document will be created by a topic. These probabilities are calculated multiple times, until the algorithm convergences and each document is being assigned to a mixture of topics.

5.4 Language Modeling

Language Modeling (LM) is the use of various techniques in order to predict the probability of occurrence of a sequence of words in a sentence. Formally, given a sequence of T words, $w_1, \ldots, w_T$, a probabilistic language model assigns to the whole sequence the probability:

$$P(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_1, w_2, \ldots, w_{t-1}) \tag{5.1}$$

The goal of LM is to provide adequate probabilistic information so that the likely word sequences have a higher probability. Language models are useful in many NLP-based applications, especially those that generate text as an output.

There are primarily two categories of language models: count-based and continuous-space language models. The count-based methods, such as statistical language models, use traditional statistical techniques like n-grams, Hidden Markov Models (HMM) and certain linguistic rules to develop probabilistic models that are able to predict the next word or character in a document, given a sequence of words that precede it. Continuous-space language models make use of different kinds of neural networks to model language and are often considered as an advanced approach to execute NLP tasks.

5.4.1 N-Gram Language Model

N-grams models are one of the simplest approaches to LM. They create a probability distribution for a sequence of n, where the number n defines the size of the "gram" (or sequence of words being assigned a probability). For example, if $n = 4$, a gram might look like this: "can you help me". There are different types of n-grams such as unigrams ($n = 1$), bigrams ($n = 2$), and trigrams ($n = 3$), as shown in Figure 5.4. Note that the n-gram model is trained on a train corpus and evaluated on a test corpus.

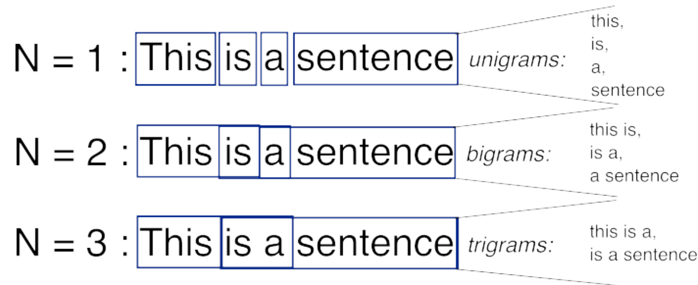

Figure 5.4: Exemplary split of a phrase into uni-, bi- and trigrams. [81]

In order to simplify the problem of estimating the language model from data, the n-gram model makes the assumption that each word depends only on the last $n-1$ words instead of the preceding $t-1$ words. So, the probability of observing the t-th word w_t can be approximated by the probability of observing it in the shortened context history of the preceding $n-1$ words ($(n-1)$-th order Markov property). Therefore, the probability $P(w_1, \ldots, w_T)$ is approximated as:

$$P(w_1, w_2, \ldots, w_T) \approx \prod_{t=1}^{T} P(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1}) \tag{5.2}$$

where the conditional probability can be computed from n-gram model frequency counts:

$$P(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1}) = \frac{\text{count } (w_{t-(n-1)}, \ldots, w_{t-1}, w_t)}{\text{count } (w_{t-(n-1)}, \ldots, w_{t-1})} \tag{5.3}$$

However, for moderately large n's there is a data sparsity problem and the language is not modeled accurately. Data sparsity is a term that describes the phenomenon of many possible word sequences having too few occurrences in a corpus, meaning they are not observed enough during training.

There are several ways to evaluate a language model. Perplexity (PP) is the inverse of the probability that the model assigns to the test corpus normalized by the number of words in the test corpus. The PP score of a test set $W = w_1, w_2, \ldots, w_N$ on an n-gram model is calculated as follows:

$$PP(W) = (P(w_1, w_2, \ldots, w_N))^{-1/N} = \left(\prod_{t=1}^{N} P(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1}) \right)^{-1/N} \tag{5.4}$$

Better language models will have a lower PP score, as it indicates that the probability distribution or the probability model is good at predicting the sample. PP can also be interpreted as the weighted average branching factor of a language in predicting the next word. Note that the branching factor of a language is the number of possible next words that can follow any word.

5.4.2 Neural Language Model

Neural language models use word embeddings, described in Section 5.5, in order to make their predictions. Continuous space embeddings help with reducing the impact of the "curse of dimensionality" in LM, i.e. as the models are trained on larger and larger corpora, the size of the vocabulary (the number of unique words)

and consequently the number of possible sequences increases, resulting in a data sparsity problem. More specifically, neural language models avoid this problem by using non-linear neural networks, such as FFNNs or RNNs, that have the ability to represent words in a distributed way, as non-linear combinations of weights in a network.

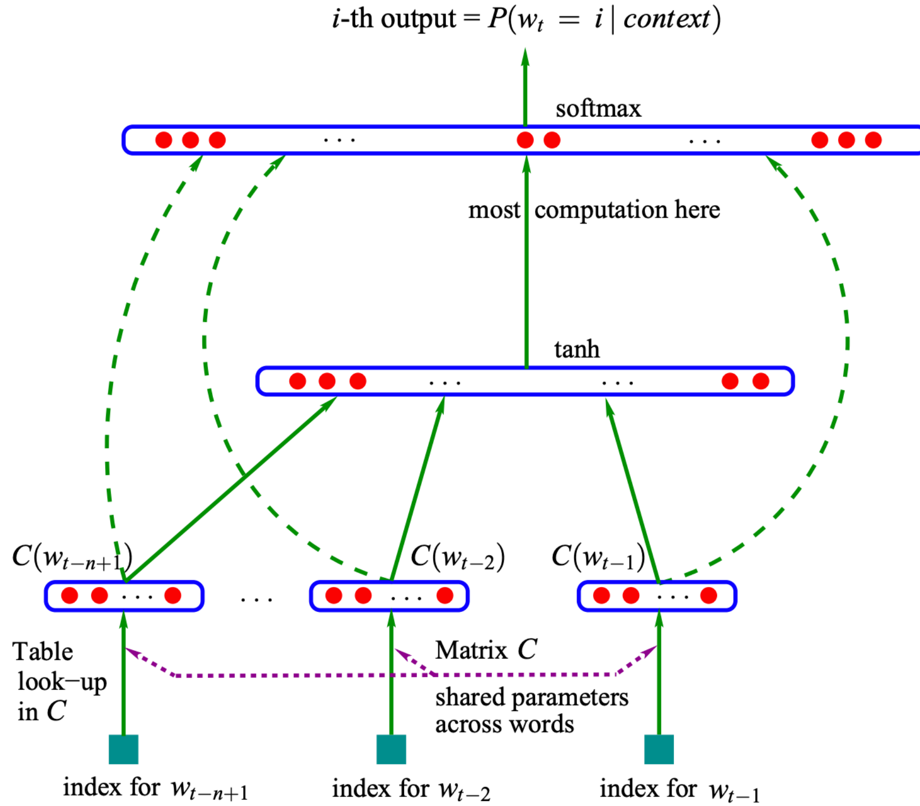

Figure 5.5: Neural Probabilistic Language Model Architecture. [82]

As in n-grams, most probabilistic language models approximate the probability of a sequence of words $P(w_t|w_1, w_2, \ldots, w_{t-1})$ using a fixed context of size $n-1$. The neural probabilistic language model introduced by Bengio et al. [82], uses a FFNN of three layers to obtain this probabilistic prediction. First, each word in the $(n-1)$-word context is mapped into a d-dimensional feature vector $C_{w_{t-i}}$, which is column w_{t-i} of parameter matrix C. The feature vectors (columns of C) are learned simultaneously with the parameters of the NN. The concatenation of these $n-1$ feature vectors, denoted by vector x, is the input to the FFNN:

$$x = (C_{w_{t-(n-1)},1}, \ldots, C_{w_{t-(n-1)},d}, C_{w_{t-(n-2)},1}, \ldots, C_{w_{t-2},d}, C_{w_{t-1},1}, \ldots, C_{w_{t-1},d}) \tag{5.5}$$

The vector x is fed into a hidden layer with *tanh* activation which then feeds into a softmax layer to estimate the probability of the next word k:

$$P(w_t = k|w_{t-(n-1)}, \ldots, w_{t-1}) = \frac{e^{a_k}}{\sum_{l=1}^{N} e^{a_l}} \tag{5.6}$$

where

$$a_k = b_k + \sum_{i=1}^{h} W_{ki} tanh\left(c_i + \sum_{j=1}^{(n-1)d} V_{ij} x_j\right) \tag{5.7}$$

where the vectors b, c and matrices W, V are parameters of the model, and h is the number of hidden units. The NN is trained using a gradient-based optimization algorithm to maximize the training set log-likelihood:

$$L(\theta) = \sum_t \log P(w_t | w_{t-(n-1)}, \ldots, w_{t-1}) \tag{5.8}$$

where θ denotes the concatenation of all the parameters.

5.5 Word Embeddings

In NLP, a word embedding is a learned numerical representation of words for text analysis that can be obtained using a set of feature learning and language modeling techniques. Typically, it is in the form of a real-valued vector in a predefined vector space that encodes the meaning of the word such that the words that have the same meaning have a similar representation. Word embeddings have introduced an efficient way of representing strings and plain text as vectors of real numbers, that can be processed by most ML algorithms. Also, when used as the underlying input representation, they improve the performance of NLP tasks.

The different techniques of word embeddings can be broadly classified into two categories: frequency based embedding techniques and prediction based embedding techniques. Frequency based embeddings, such as count vector, TF-IDF vectorization, and co-occurrence matrix, vectorize the text depending on the frequency of occurrence of the words in the corpus. Prediction based embeddings include methods such as Word2Vec [83], and GloVe [84]. Finally, both PCA and t-SNE can be used in order reduce the dimensionality of the word vector spaces and visualize word embeddings and clusters.

5.5.1 Term Frequency-Inverse Document Frequency Vector

Term Frequency-Inverse Document Frequency (tf–idf) is a term-weighting scheme that reflects how relevant a word is to a document in a collection of documents or corpus. The tf–idf value increases proportionally to the number of occurrences of a word in the document and is offset by the number of documents in the corpus that contain the word. This way, words that are common in every document, such as 'the', 'if', and 'is', rank lower than the words that appear in some documents only, since the former do not contain important information and the latter indicate they might be relevant.

The tf–idf score for the word t in the document d from the document set D is the product of the following two metrics:

- Term Frequency (tf): It refers to the frequency of a word, i.e. how often a given word appears within each document. There are several ways of calculating this metric. The simplest of them is using the raw count $f_{t,d}$ of the instances the term t appears in the document d:

$$\text{tf}(t, d) = f_{t,d} \tag{5.9}$$

Then, there are several ways to adjust it. For example, tf could be adjusted by the length of the document:

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{5.10}$$

or by the augmented frequency in order to prevent a bias towards longer documents:

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{max_{\{t' \in d\}} f_{t',d}} \tag{5.11}$$

The term frequency could also be defined as the logarithmically scaled frequency:

$$\mathrm{tf}(t,d) = \log(1 + f_{t,d}) \tag{5.12}$$

- Inverse Document Frequency (idf): The idf of a word across a set of documents is the count of the number of documents the word appears in. The idf metric measures how significant the word is in the entire corpus. This metric can be computed by dividing the total number of documents in the corpus, $N = |D|$, with the number of documents that contain the specific word t, $|\{d \in D : t \in d\}|$, and then calculating the logarithm of the result. If the word is common in many documents, this number will approach 0. Otherwise, if the word is rare, it will approach 1. Mathematically:

$$\mathrm{idf}(t,D) = \log\left(\frac{N}{1 + |\{d \in D : t \in d\}|}\right) \tag{5.13}$$

Formally, the tf–idf score is calculated as:

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D) \tag{5.14}$$

A high tf in the given document and a low document frequency of the word in the whole corpus causes a high weight in tf–idf to be reached. Thus, the weights tend to filter out common terms. In conclusion, tf–idf provides a way to map documents to word vectors by associating each word in a document with a number that represents how relevant each word is in that document. Documents with similar, relevant words will have similar vectors.

5.5.2 Word2Vec

Word2Vec [83] is a family of model architectures and optimizations that can be used to produce word embeddings. These models are shallow, two-layer neural networks that turn text into a numerical form that deep neural networks can understand. In contrast to techniques like common Bag-of-Words (BOW) and tf–idf, Word2Vec models can capture the meaning or relation of the words once trained. Word2Vec takes as input a text corpus and outputs a vector space, typically of several hundred dimensions. In this space, each unique word in the corpus is being assigned a corresponding vector. The vectors are chosen in such a way that a simple mathematical function, like the cosine similarity between the vectors, indicates the level of semantic similarity between the words represented by those vectors. The aim is to have words that share common contexts in the corpus located close to one another in the vector space.

Word2Vec has two forms, the Skip-Gram model and the Continuous Bag-of-Words (CBOW) model, as illustrated in Figure 5.7. In the first way, a given word is used to predict a target context (surrounding words) while in the second, context is used to predict a target word. Note that both models only have one hidden layer, probably due to the restriction of computation cost at that time these models were proposed.

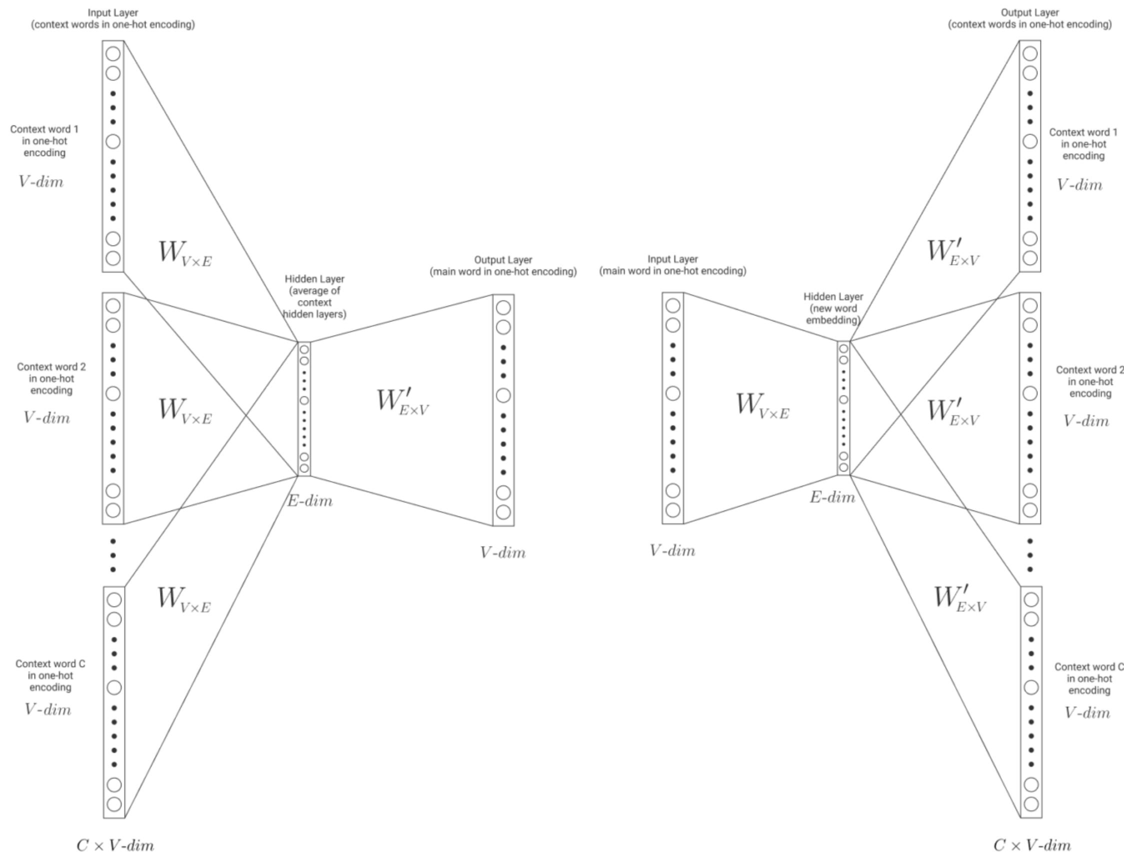

Figure 5.6: Word2Vec model architectures: (a) CBOW, (b) Skip-Gram. [85]

5.5.2.1 Continuous Skip-Gram Model

The Skip-Gram model predicts the surrounding window of context words from the target word. The skip-gram architecture is illustrated in Figure 5.7. The input layer of the model is of size $(1 \times V)$, where V is the number of words in the vocabulary. The input to the network is the one-hot representation of the target word. This input vector is transformed by the weight matrix W of size $(V \times E)$, and goes through a hidden layer of size $(1 \times E)$, where the hyperparameter E is the desired embedding dimensions or features the model is trying to learn. The higher the value of this hyperparameter is, the more information the embeddings will capture, but the harder it will be to learn it. Finally, the weight matrix W' of size $(E \times V)$ transforms the hidden layer into the output layer of size $(1 \times V)$, since the outputs to be predicted are going to be the one-hot encoded context words.

The skip-gram model will learn by training to predict the context given a target word. Once the training is done in the whole vocabulary, a weight matrix $W_{V\times E}$ that connects the input to the hidden layer is going to be produced, and the embeddings can be obtained. This representation, ideally, encapsulates semantics, and similar

words are close to each other in the vector space. Skip-gram performs better with small amount of data and is found to represent infrequent words well.

5.5.2.2 Continuous Bag-of-Words Model

The Continuous Bag-of-Words (CBOW) model looks like the opposite process of the skip-gram model, since it predicts the current word based on surrounding context words. The context consists of a window of words around the current (middle) word, i.e. a few words before and after the center word. The CBOW architecture is illustrated in Figure 5.7. The input layer of size $(1 \times V)$ consists of the context words in one-hot encoding, and for every such word the weight matrix $W_{V \times E}$ results in the hidden layer. Then they are averaged into a single hidden layer, which is passed on to the output layer.

The CBOW model is essentially learning word embeddings by training a model to predict a word given its context. Again, the weight matrix $W_{V \times E}$ is used to generate the word embeddings from the one-hot encodings once the training is done. CBOW can better represent more frequent words and is faster than skip-gram. Note that this architecture is called a bag-of-words model because the order of the context words does not influence prediction.

5.5.3 Global Vectors for Word Representation

Global Vectors (GloVe) [84] is an unsupervised learning algorithm for obtaining vector representations for words. Like most unsupervised algorithms, it is based on measures like word frequency and co-occurence counts. GloVe model trains on global co-occurrence counts of words, which indicate how frequently each pair of words is used in the given corpus, and makes a sufficient use of statistics by minimizing least-squares error. As a result, it produces a meaningful word vector space, where the distance between words is related to semantic similarity. Note that GloVe combines the features of two primary families of learning word vectors, namely the global matrix factorization (e.g. LSA) and local context window methods (e.g. skip-gram).

Formally, GloVe constructs an explicit word-context or word co-occurrence matrix X using statistics across the whole corpus. Each cell X_{ij} represents how often the word i appears in the context of the word j. Usually, for each word in the corpus, context terms are searched within some area defined by a window size before and a window size after the term. Also, more distant words are given less weight. Soft constraints are defined for each word pair:

$$w_i^T w_j + b_i + b_j = \log(X_{ij}) \tag{5.15}$$

where w_i, b_i are the vector and scalar bias for the main word and w_j, b_j are the vector and scalar bias for the context words. GloVe uses a weighted least squares objective that minimizes the difference between the dot product of the vectors of two words and the logarithm of their number of co-occurrences:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log(X_{ij}))^2 \tag{5.16}$$

where f is a weighting function that assigns lower weights to rare and frequent co-occurrences. One class of weighting functions found to work well can be parameterized as:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{X_{max}}\right)^{a} & \text{, if } X_{ij} < X_{max} \\ 1 & \text{, otherwise} \end{cases} \tag{5.17}$$

5.6 Transfer Learning in Natural Language Processing

Most of the tasks in NLP are sequence modeling tasks. In contrast to traditional NNs, recurrent neural networks like RNNs and LSTMs can model sequential information present in the text, so they are commonly used for these types of tasks. However, these networks have some major issues that make the need for transfer learning in NLP obvious. One of them is that, in the case of a text sequence, RNNs or LSTMs can not be parallelized, as they take one token at a time as input. Hence, training such a model on a large data set will take a lot of time.

Transfer learning, as seen in Section 4.5, is a technique where a deep learning model trained on a large labelled data set, called a pre-trained model, is used as a starting point to solve a problem on another, usually smaller, data set. The form of TL that has led to the biggest improvements so far in NLP is sequential TL. It involves transferring knowledge with a sequence of steps, where the source and target task are not necessarily similar. The general practice is to pre-train representations on a large unlabelled text corpus using a method like ELMo [86] or BERT [87], and then to adapt these representations to a supervised target task using labelled data.

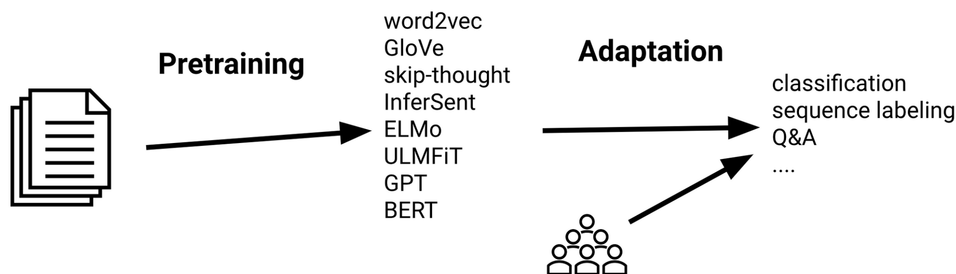

Figure 5.7: The general procedure of sequential transfer learning. [88]

5.6.1 Embeddings from Language Models

Embeddings from Language Models (ELMo) [86] is a type of deep contextualized word representation, that models both complex characteristics of word use, such as syntax, and how these uses vary across linguistic contexts. Unlike earlier vector approaches such as Word2Vec and GloVe, ELMo embeddings are context-sensitive. In other words, they are able to capture the context of the word used in the sentence and can generate different representations for the same word used in a different context in different sentences. ELMo is trained on predicting the next word in a sequence of words, which is a task called language modeling. This model is trained on a massive data set, and then it can be used as a component in other architectures in order to perform specific language tasks.

ELMo uses a particular type of language model called Bi-directional Language Model (biLM), which is a combination of a forward and backward language model, as illustrated in Figure 5.8. Word vectors are learned functions of the internal states of this biLM, which is pre-trained on a large text corpus. More specifically, according to Peters et al. [86], ELMo is a task specific combination of the intermediate layer representations in the biLM. Given that for each token t_k, x_k^{LM} is a context-independent token representation (via token embeddings or a Convolutional Neural Network (CNN) over characters), a L-layer bi-directional LSTM computes a set of $2L+1$ representations

$$R_k = \{x_k^{LM}, \overrightarrow{h}_{k,j}^{LM}, \overleftarrow{h}_{k,j}^{LM} | j = 1, \ldots, L\} = \{h_{k,j}^{LM} | j = 0, \ldots, L\} \tag{5.18}$$

where $h_{k,0}^{LM}$ is the token layer and $h_{k,j}^{LM} = [\overrightarrow{h}_{k,j}^{LM}; \overleftarrow{h}_{k,j}^{LM}]$, for each bi-directional LSTM layer. ELMo embeddings are computed by a task-specific weighting of all biLM layers:

$$ELMo_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM} \tag{5.19}$$

where s^{task} are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to scale the entire ELMo vector.

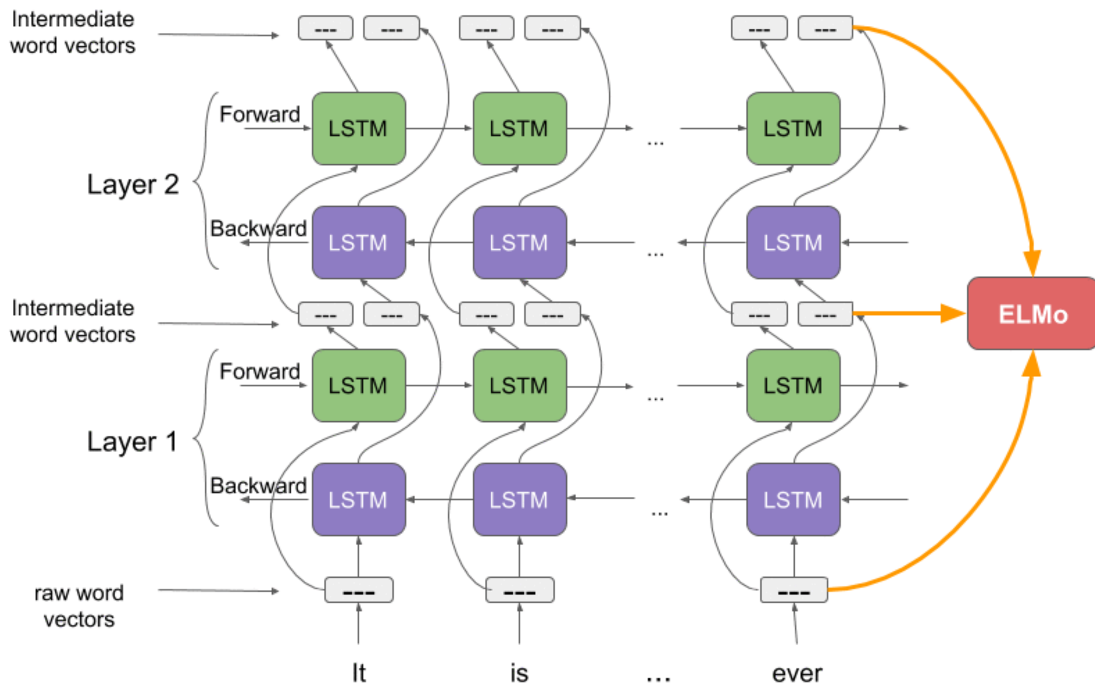

Figure 5.8: ELMo uses internal representations of multi-layer biLM. [89]

5.6.2 Bidirectional Encoder Representations from Transformers

Bi-directional Encoder Representations from Transformers (BERT) [87] is a Transformer-based machine learning technique for NLP pre-training. It improves upon standard Transformers, shown in Figure 4.17, by using a Masked Language Model (MLM) that allows bi-directional training in models which it was previously

impossible. More specifically, BERT is pre-trained with two objectives:: masked language modeling and Next Sentence Prediction (NSP). Taking a sentence, the MLM (Figure 5.9) randomly masks 15% of the words in the input, and then it tries to predict them. Masking means that the model looks in both directions and it uses the full context of the sentence, both left and right surroundings, in order to predict the masked word. Unlike the previous left-to-right language models, it takes both the previous and next tokens into account at the same time, which allows to pre-train a deep bidirectional Transformer. In addition to the MLM, BERT uses a NSP (Figure 5.10) that jointly pre-trains text-pair representations. More precisely, the model concatenates two masked sentences as inputs during pre-training, and then it has to predict whether or not the two sentences were following each other.

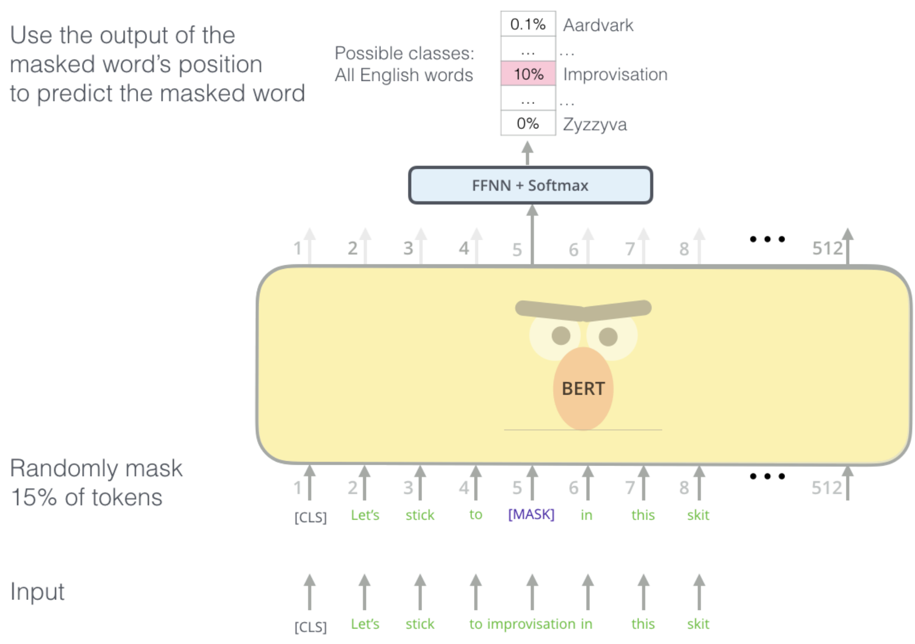

Figure 5.9: BERT: Masked Language Model. [90]

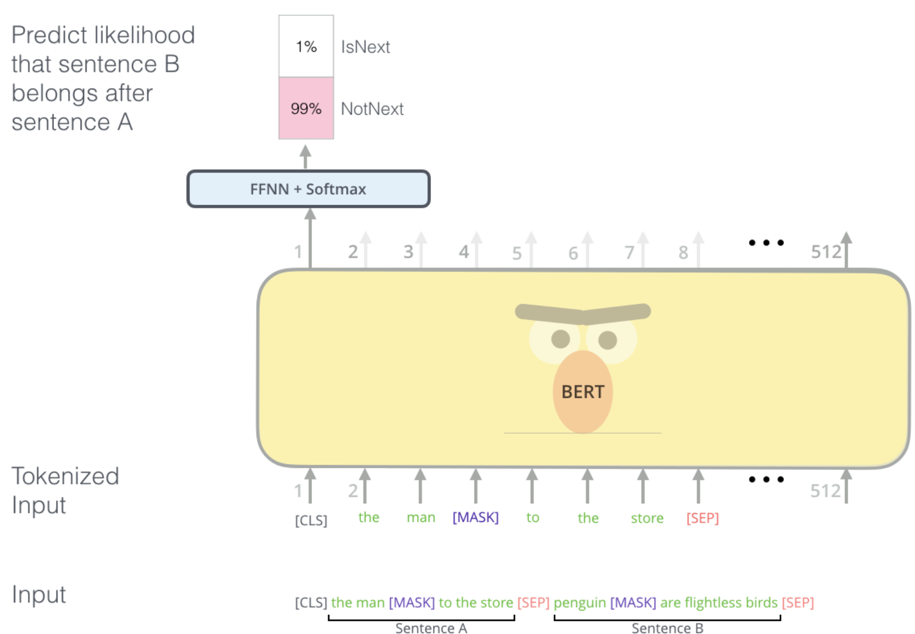

Figure 5.10: BERT: Next Sentence Prediction. [90]

Note that the special tokens [SEP] and [CLS] that appear in the above Figures are special tokens required by BERT. The special token [SEP] is used to differentiate between the different input sentences. The [CLS] token always appears at the start of the text, and is specific to classification tasks.

BERT consists of two steps (Figure 5.11): pre-training and fine-tuning. During the pre-training step, the BERT model is trained on unlabeled text data, over different pre-training tasks. For the fine-tuning step, the model is initialized with the pre-trained parameters, and then all of the parameters are fine-tuned using labeled data from the downstream tasks. Each downstream task, even though they are initialized with the same pre-trained parameters, has separate fine-tuned models.

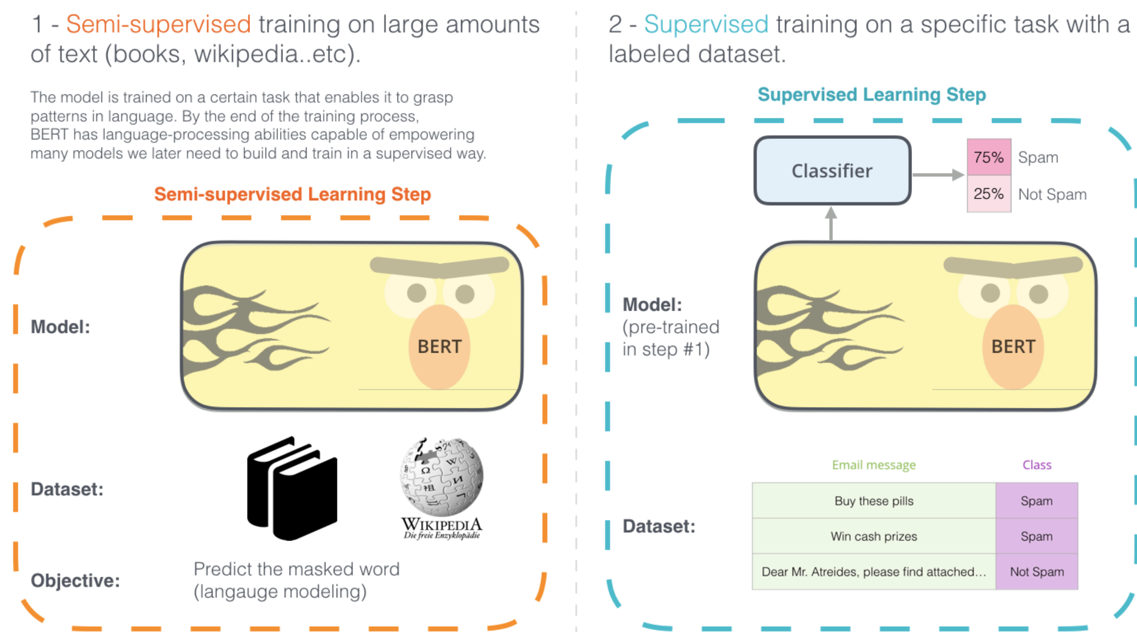

Figure 5.11: BERT: The two steps of how it is developed. [90]

The original English-language BERT has two pre-trained versions depending on the scale of the model architecture:

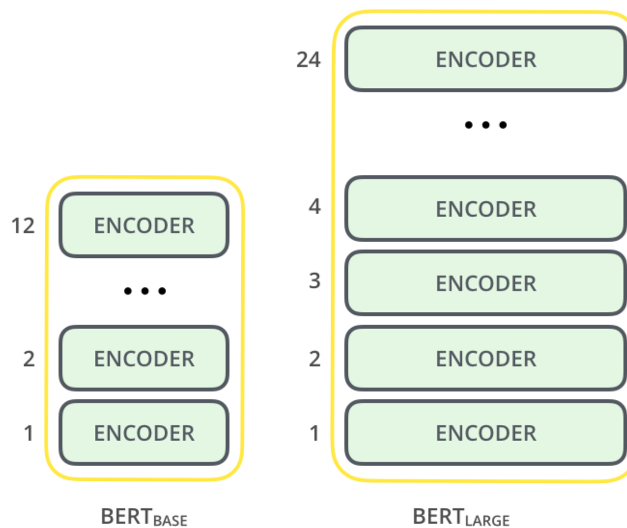

Figure 5.12: BERT-base vs. BERT-large. [90]

- BERT-base: 12 encoder layers (transformer blocks), 12 attention heads, and 110 million parameters.
- BERT-large: 24 encoder layers, 16 attention heads and, 340 million parameters.

Both models, illustrated in Figure 5.12, are pre-trained from unlabeled data extracted from the English Wikipedia and the BooksCorpus in a self-supervised fashion. In other words, they were pre-trained on the raw texts only, with an automatic process to generate inputs and labels from those texts.

Chapter 6

Experiments

6.1 Data Sets

6.1.1 Twitter Sentiment Analysis Self-Driving Cars

We use a data set from CrowdFlower[1] that consists of Twitter messages about self-driving cars. More specifically, contributors read tweets and classified them as "very negative", "negative", "neutral", "positive", or "very positive". Contributors were also asked to mark if the tweet was not relevant to self-driving cars. This data set contains 7,156 observations and 11 variables. In this analysis, we focus on two features, namely the "sentiment", which contains the sentiment of the tweet (denoted by the numbers 1 through 5), and the "text", which contains the text of the tweet. Table 6.1 depicts a sample of the data set.

sentiment	text
5	Two places I'd invest all my money if I could: 3D printing and Self-driving cars!!!
5	Awesome! Google driverless cars will help the blind travel more often; https://t.co/QWuXR0FrBpv
2	Why would anyone need a self driving car?
3	How is an automated vehicle supposed to behave in an emergency? #AutoAuto
4	Driverless cars?! I want one
3	Will driverless cars eventually replace taxi drivers in cities?

Table 6.1: A sample of the Twitter Sentiment Analysis Self-Driving Cars Data Set.

The data set was randomly split into training (85%), validation (7.5%) and testing (7.5%) set. The training set consists of 5,901 samples and the validation set consists of 521 samples. The proposed models were evaluated on the independent testing set consisting of totally 521 samples. Table 6.2 shows the number of samples in the different sets for each class. We notice that the data set is heavily imbalanced.

Sentiment	Training Set	Validation Set	Testing Set
very positive	390	35	34
positive	1227	108	109
neutral	3608	319	318
negative	582	51	52
very negative	94	8	8

Table 6.2: Samples distribution among classes in the training, validation and testing data set.

[1] http://www.crowdflower.com/data-for-everyone

6.1.2 Stanford Sentiment Treebank

The Stanford Sentiment Treebank (SST) [91] is one of the most popular corpora for fine-grained (five-way) sentiment classification. The data set consists of 11,855 single sentences extracted from movie reviews on Rotten Tomatoes. It was also parsed by the Stanford constituency parser [92] and includes a total of 215,154 unique phrases from those parse trees, each annotated by 3 human judges.

Each phrase is labelled as either "very negative" (1), "negative" (2), "neutral" (3), "positive" (4) or "very positive" (5). The corpus with all 5 labels is referred to as SST-5 or SST fine-grained. Models are often evaluated using this data set as a benchmark for fine-grained classification based on accuracy. Table 6.3 shows some sample reviews from the SST-5 data set.

label	text
1	A predictable , manipulative stinker.
4	If you love reading and/or poetry , then by all means check it out.
5	You 'll probably love it.
3	Some movies blend together as they become distant memories.
2	It 's not a great monster movie .
2	The movie generates plot points with a degree of randomness usually achieved only by lottery drawing.

Table 6.3: A sample of the Stanford Sentiment Treebank Data Set.

Note that we are going to use this data set only for evaluation purposes, so we are not going to describe the pre-processing steps or conduct an explanatory analysis as with the Twitter sentiment analysis self-driving cars data set.

6.2 Explanatory Data Analysis

In this section, we describe the pre-processing steps necessary in order to prepare the tweets in the Twitter sentiment analysis self-driving cars data set to a form suitable for a basic explanatory analysis.

6.2.1 Pre-processing

As the Twitter sentiment analysis self-driving cars data set is crawled from the internet, raw tweets may contain a lot of useless or misleading information, such as punctuation and abbreviations. Therefore, we perform a few data cleaning steps, or pre-processing steps, to improve the quality of the raw tweets in data:

1. We remove null values, i.e. the tweets that are not relevant to self-driving cars.
2. We remove the Twitter handles, i.e. the "@username", as they don't give any relevant information about the nature of the tweet.
3. We remove all URL addresses ('www.' pattern). Most of them are short and located at the end of a tweet.
4. We decode HTML to general text.
5. We restore some abbreviations in the tweets, e.g., substituting ”have” for ”ve”.
6. We remove some meaningless numbers, punctuation (e.g. semicolon and colon), and special characters (except hashtags symbol "#").
7. We convert all characters to lower-case.

8. We remove stop words, except some words such as "not", "nor" and "can" that play a significant role in differentiating the sentiments of tweets.
9. We remove small words (length less than three characters), as they do not add significant value for the analysis. After this step, we can visualize the length of the tweets in the data set, as in Figure 6.1. We see that the average tweet consists of 61 characters.
10. We perform tokenization of the tweets.
11. We experiment with lemmatization and stemming.

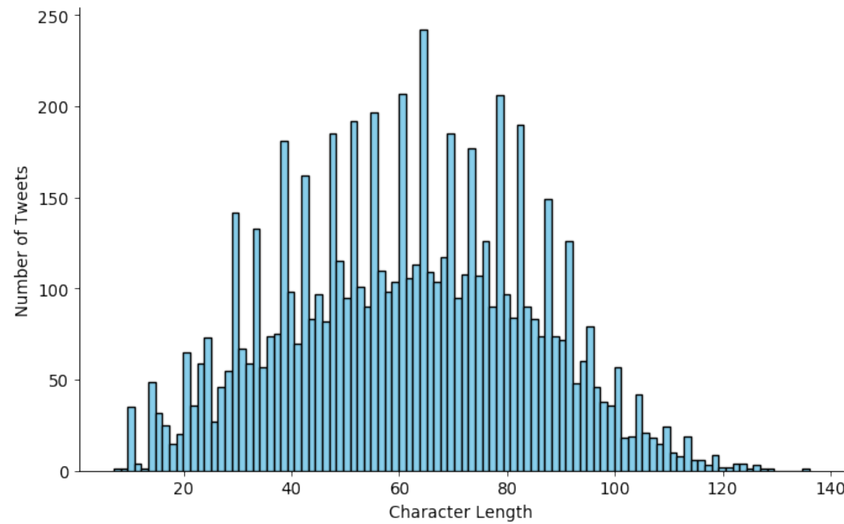

Figure 6.1: Distribution of Tweets' Character Length in the Twitter Sentiment Analysis Self-Driving Cars Data Set.

6.2.2 Keyword & Entity Analysis

First, we find the most frequent words in the data set (Table 6.4). We can also create a word-cloud in order to simply visualize the data and which words are shown in varying sizes depending on how often they appear in the tweets, as shown in Figure 6.2. We present the results before and after performing lemmatization that reduces inflected words to their root words.

Without Lemmatization		With Lemmatization	
Word	**Count**	**Word**	**Count**
driving	4248	car	6921
self	4018	drive	4533
car	3643	self	4040
cars	3209	google	2937
google	2674	driverless	1893
driverless	1810	just	570
not	733	autonomous	398
can	458	vehicle	365
autonomous	369	future	331
future	306	driver	322

Table 6.4: Most Common Words of Twitter Sentiment Analysis Self-Driving Cars Data Set.

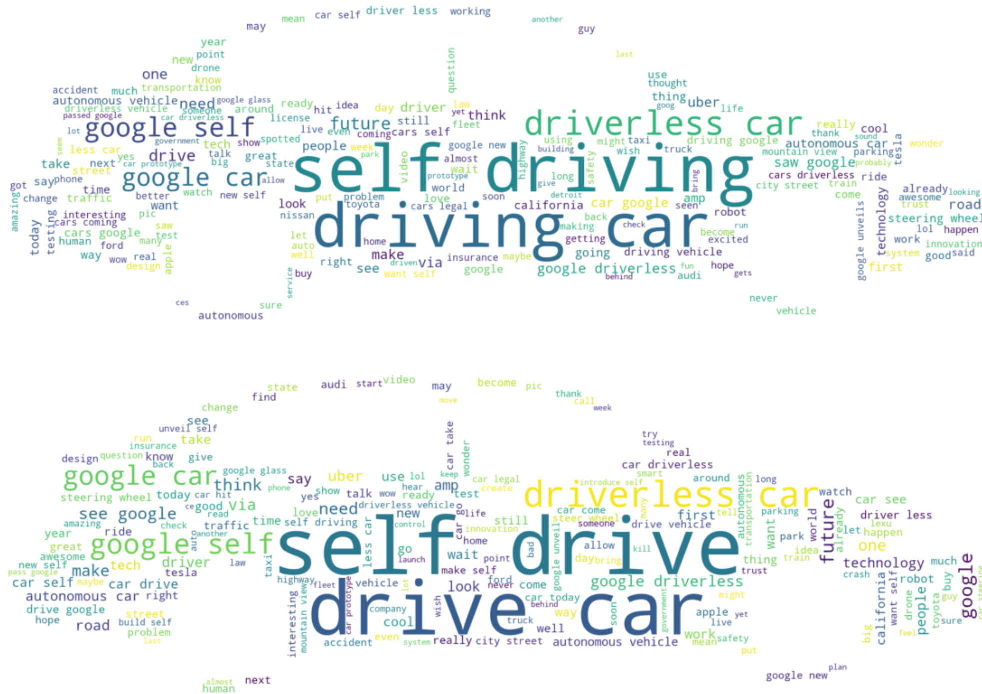

Figure 6.2: Word Cloud of Most Common Words of Twitter Sentiment Analysis Self-Driving Cars Data Set (a) without Lemmatization (b) with Lemmatization.

In addition, we can find the most frequent words for every class in the data set (Table 6.5). We see that the top 10 words of each class are almost the same, so we can not make any assumptions about the data from this information. However, if we visualize the top 100 words of each class, we start to see the differences that lead to the sentiment in question. For example, a "very negative" tweet consists of words like "destroy", "hell" and "dangerous", while a "very positive" tweet about AVs has words like "love", "amazing", and "excited". Note that we do not perform lemmatization in this case.

Very Negative		Negative		Neutral		Positive		Very Positive	
Word	**Count**	**Word**	**Count**	**Word**	**Count**	**Word**	**Count**	**Word**	**Count**
driving	61	driving	458	driving	2439	driving	982	driving	308
car	60	self	428	self	2334	self	911	self	289
cars	56	cars	394	car	2264	car	731	car	253
self	56	car	335	cars	1853	cars	710	cars	196
driverless	36	google	228	google	1716	google	529	google	171
not	30	driverless	200	driverless	1147	driverless	324	driverless	103
google	30	not	166	not	319	not	163	not	55
would	17	like	44	autonomous	255	would	104	awesome	39
think	10	think	36	future	185	cool	80	wait	38
people	9	people	35	saw	177	drive	78	one	36

Table 6.5: Most Common Words of Twitter Sentiment Analysis Self-Driving Cars Data Set.

Entity analysis shows the top entities related to tweets on self-driving cars. For instance, Google cars, Uber, Audi and Tesla Motors. Similarly, by analyzing top hashtags, an interesting linkage of autonomous cars was revealed. For example, #codecon, which was an annual conference for software hackers and technology enthusiasts.

Top Mentions		Top Hashtags	
Twitter Handle	**Count**	**Hashtag**	**Count**
@google	150	#Google	118
@Uber	49	#google	103
@Google	45	#tech	34
@Audi	17	#cars	27
@TeslaMotors	16	#technology	21
@nytimes	15	#googlecar	20
@ComputerHistory	11	#future	19
@USATODAY	11	#car	19
@nvidia	9	#codecon	19
@audi	9	#innovation	18

Table 6.6: @-Analysis and Top Hashtags of Twitter Sentiment Analysis Self-Driving Cars Data Set.

6.2.3 Topic Modeling

We are going to perform topic modeling on the Twitter data set in order to create topics along with the probability distribution for each word in our vocabulary for each topic. Before fitting into LDA model, tf–idf vectorizer feature extraction was done. The number of components is 5, the number of max iterations is taken as 10, and the random state was predefined as 42.

We can visualize the topics as circles in the two-dimensional plane [93], as shown in Figure 6.3. In addition, we create word-clouds (Figure 6.4) that show the top words in the extracted topics. Based on this we see that the topics can somehow differentiate the sentiments regarding AVs. For example, topic #3 contains words such as "problem", "kill", "bad" and "accident", and topic #1 words like "great", "hope" and "pretty". However, we still have a long way to go if we want to classify the tweets correctly.

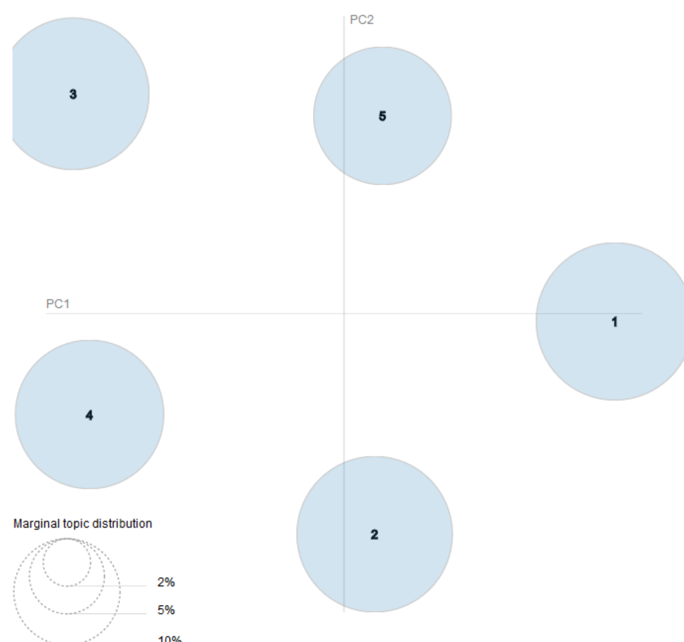

Figure 6.3: Intertopic Distance Map (via Multidimensional Scaling).

Figure 6.4: Topic Modeling: Word Clouds of Top Words (a) Topic #1 (b) Topic #2 (c) Topic #3 (d) Topic #4 (e) Topic #5.

6.3 Proposed Systems

In this section, we describe the LSTM based models and the BERT based models that we used in order to tackle the problem of multi-class Twitter sentiment analysis. In addition, we train and evaluate the models, and present the results in the form of classification reports and confusion matrices.

6.3.1 LSTM Based Network

Initially, LSTM based models were developed, including a bi-directional stacked LSTM neural network with self-attention. Note that the output vectors of this Bi-directional LSTM (BLSTM) model are sent through an attention layer and then a series of dense and dropout layers to a final softmax layer to build a sentence classifier.

6.3.1.1 Pre-processing

First, we first present the preparation of the data to a form suitable for the classifier training. Note that the pre-processing steps described here are different than the ones described in Section 6.2.2. More specifically:

1. We remove null values.
2. We replace the Twitter handles with the string "_MENTION_".
3. We replace all URL addresses with the string "_URL_".
4. We decode HTML to general text.
5. We restore some abbreviations in the texts.
6. We replace numbers with the string "_NUMBER_".
7. We perform tokenization of the texts.

In this case, the average tweet of the data set consists of almost 100 characters.

6.3.1.2 Model Architecture

After applying the different steps of the pre-processing part, we can now focus on the machine learning part. In order to perform fine-grained sentiment analysis for tweets, we experiment with a self-attention-based BLSTM model. The model consists of an embedding layer, two BLSTM layers forming a stacked BLSTM, an attention layer and three dense layers.

- Embedding Layer: The input to the network is a tweet, treated as a sequence of words, $w_1, w_2, \ldots, w_T$. In order to extract the semantic information of tweets, each sequence of words is represented as a sequence of word embeddings. We use an embedding layer of size d to project the words to a low-dimensional vector space and create word embeddings, $x_1, x_2, \ldots, x_T$. In this approach, the embedding layer is initialized with a pre-trained word embedding model that uses Word2Vec technique [83] on tweets. The Word2Vec 400M Twitter model [94] has an embedding dimension of $d = 400$. Note that the weights of the embedding layer does not remain frozen during training.
- BLSTM Layers: After the embedding layer, the sequence of word embeddings is fed into a 2-layer Bi-directional LSTM to achieve another representation of

$h_1, h_2, \ldots, h_T$, where h_t is the hidden state at time-step t. Note that BLSTM networks [95] are sequence processing models that consist of 2 LSTMs: one taking the input in a forward direction, and the other in a backwards direction. The second LSTM is basically a reversed copy of the first one, so that we can take full advantage of both past and future input features for a specific time-step. BLSTM networks are trained using BPTT.

- Attention Layer: The attention mechanism lets the model decide the importance of each word for the prediction task by weighing them when constructing the representation of the text. We use a simple approach presented in Felbo et al. [96], inspired by [67], [97] with a single parameter input channel:

$$e_t = h_t w_a + b_a \tag{6.1}$$

$$a_t = \frac{\exp(e_t)}{\sum_{i=1}^{T} \exp(e_i)} \tag{6.2}$$

$$\upsilon = \sum_{i=1}^{T} a_i h_i \tag{6.3}$$

where h_t is the hidden representation of the word at time-step t, w_a is the weight matrix and b_a the bias term for the attention layer. The attention importance scores, or attention weights, for each time-step, a_t, are obtained by multiplying the hidden representations with the weight matrix and then normalizing to construct a probability distribution over the words. Lastly, the representation vector for the text, υ, is defined as a weighted average of the vectors h_i.

- Dense Layers: The attention layer is followed by 3 dense layers with different sizes of neurons. First, the output of the attention layer is used as input to a dense layer of 128 hidden neurons, with Leaky ReLU (LReLU) activation function. In order to avoid potential over-fitting, a dropout layer is utilized between the first and the second dense layer. Then, the output is fed into the next dense layer of 64 units with LReLU activation function, followed by a dropout layer. Note that we try different dropout rates in order to find the best configurations. Finally, the output is used as input to a softmax layer, which outputs a probability distribution over all classes.

We can visualise the embeddings by using techniques such as PCA and t-SNE, as shown in Figure 6.5.

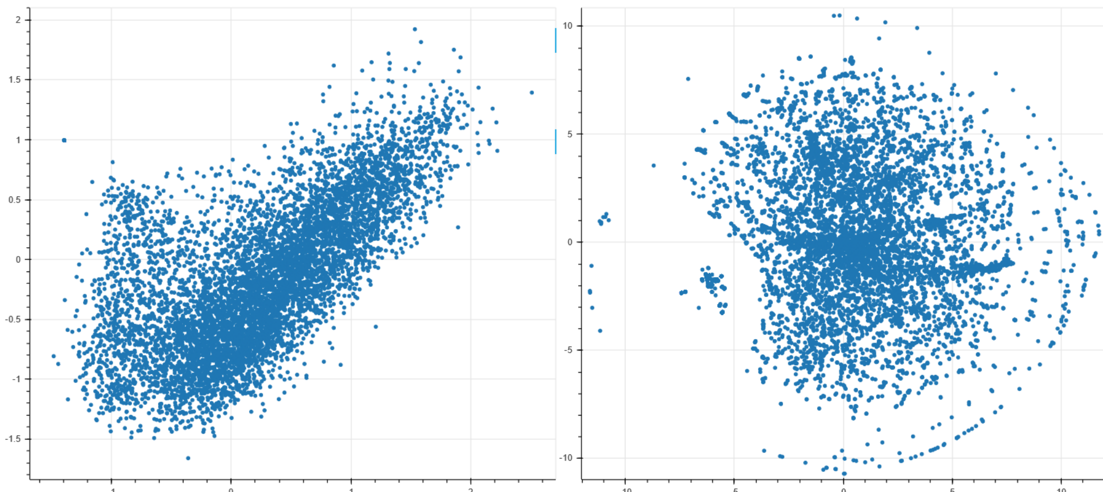

Figure 6.5: Visualization of the Embeddings using (a) PCA (b) t-SNE.

6.3.1.3 Experimental Setup

To optimize our model, we use the categorical-cross entropy loss function and the Adam algorithm. Note that the default parameters of Adam are: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e - 08$. All the hyper-parameters used are shown in Table 6.7.

Hyperparameter	**Value**
Number of epochs	10
Learning rate	0.001
Sequence length	128
Early stopping patience	2
Embedding layer size	400
First BLSTM layer size	400
Second BLSTM layer size	400
BLSTM dropout rate	0.2
First dense layer size	128
First dropout rate	0.5
Second dense layer size	64
Second dropout rate	0.4

Table 6.7: Hyper-parameters of the 2-BLSTM+Att Model.

In addition, in order to tackle the imbalance problem of the Twitter sentiment analysis self-driving cars data set, we perform cost-sensitive learning. In this method, the misclassifications of the minority class are penalized more heavily than the ones of the majority class, meaning the loss is different for each class. Such a penalty system may induce the model to pay more attention to the minority class. The weight of each class is presented in Table 6.8

	Twitter Data Set	SST-5 Data Set
sentiment	**weight**	**weight**
very negative	12.555	1.565
negative	2.028	0.770
neutral	0.327	1.052
positive	0.962	0.736
very positive	3.026	1.327

Table 6.8: Cost-Sensitive Learning: Class Weights.

To implement the models, we used the TensorFlow [98] the Scikit-learn [99] libraries. We trained the neural network on a GTX 2060 (8GB) GPU.

6.3.1.4 Results

The model is evaluated on the test (unseen) data sets. To evaluate the classification performance, we use three metrics: accuracy, macro average and weighted average F1 score. The results are depicted in Table 6.9. Also, we illustrate the confusion matrix (Figure 6.6).

Based on the classification report of the Twitter data set, we see that the model has a problem classifying the three classes that have the lowest proportions in the training

data as compared to the other classes, namely "very negative", "negative" and "very positive". If we focus on the classification matrices, we see that the classifier, even though it has problems distinguishing between some classes, does not confuse the overall positive sentiments with the negative sentiments. The accuracy in this data set is 62% and the macro F1 metric is 42%. A similar behavior can be seen in the SST-5 data set, where the accuracy and the macro F1 score are almost identical at 42%.

	Twitter Data Set				SST-5 Data Set			
sentiment	**precision**	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
very negative	0.18	0.38	0.24	8	0.42	0.36	0.39	279
negative	0.32	0.38	0.35	52	0.51	0.39	0.44	633
neutral	0.81	0.74	0.77	318	0.27	0.35	0.30	389
positive	0.40	0.46	0.43	109	0.39	0.45	0.42	510
very positive	0.40	0.29	0.34	34	0.54	0.51	0.52	399
accuracy			0.61	521			0.42	2210
macro avg	0.42	0.45	0.42	521	0.42	0.41	0.41	2210
weighted avg	0.64	0.61	0.62	521	0.43	0.42	0.42	2210

Table 6.9: Classification Report of 2-BLSTM+Att Model.

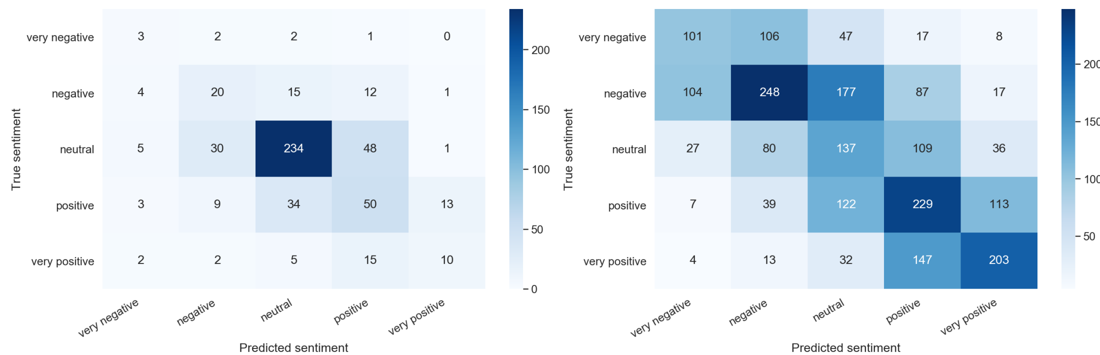

Figure 6.6: Confusion Matrix of 2-BLSTM+Att Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set.

6.3.2 BERT-Based Networks

We train a BERT-base for sequence classification model, a bare BERT-base model [87], a DistilBERT model [100], a RoBERTa model [101] and a BERTweet model [102]. Regarding the data pre-processing, it is a little different in this case. The transformer models are available pre-trained on the text corpora in two different versions: cased and uncased. Each model is trained with its own tokenization method using its cased version.

6.3.2.1 Model Architecture

First, let's briefly describe the different BERT language models we used:

- BERT-Base: BERT [87], as seen in Section 5.6.2, is a Transformers model pre-trained on a large corpus of English data in a self-supervised fashion. BERT-base, in particular, is a version of BERT that consists of 12 encoder layers, 768 hidden

layers, 12 attention heads, and 110 million parameters. The BERT-base model we are going to use in this case is trained on cased English text.
- DistilBERT [100]: It is a small general-purpose language representation model based on the BERT architecture that can be fine-tuned with good performances on a wide range of NLP tasks. It is a fast, cheap and light Transformer model that performs knowledge distillation during the pre-training phase to reduce the size of a BERT model by 40%. DistilBERT runs 60% faster while preserving over 97% of BERT's performances as measured on the GLUE language understanding benchmark. The authors introduce a triple loss combining language modeling, cosine-distance losses and distillation in order to leverage the inductive biases learned by larger models during pre-training.
- RoBERTa [101]: It is a model that builds on BERT pre-training [87] and modifies key hyper-parameters and training data size, More specifically, it removes the next-sentence pre-training objective and trains with much larger mini-batches and learning rates, achieving SOTA results on GLUE, RACE and SQuAD.
- BERTweet [102]: It is the first public large-scale pre-trained language model for English Twitter messages. It has the same architecture as BERT-base [87] and it is trained using the RoBERTa pre-training procedure [101]. Experiments show that BERTweet outperforms strong baselines RoBERTa-base and XLM-R-base [103], producing better performance results than the previous SOTA models on three Tweet NLP tasks: part-of-speech tagging, named-entity recognition, and text classification.

Fine-tuning a BERT model is pretty straightforward and relatively inexpensive compared to the pre-training step. Basically, we plug in the task-specific inputs and outputs into a pre-trained BERT language model, and fine-tune all the parameters end-to-end for a few epochs. For the given task of sentence classification, the inputs stay unchanged. At the output, the [CLS] representation is fed into a dropout layer for some regularization, which is followed by a simple fully-connected layer with softmax activation function for classification. The final output represents the predicted probability distribution of the classes.

6.3.2.2 Experimental Setup

To optimize our models, we use the cross entropy loss function and a special case of the Adam algorithm with weight decay fix [104] . We also create a schedule with the learning rate that decreases linearly from its initial value to 0, after a warm-up period during which it increases linearly from 0 to its initial value set. Note that we set the number of warm-up steps to 0. In addition, in order to tackle the imbalance problem of the Twitter data set, we perform cost-sensitive learning (described in Section 6.3.1.3). Devlin et al. [87] found that the range of possible BERT hyper-parameters' values depicted in Table 6.10 work well across most tasks.

Hyperparameter	**Values**
Batch size	$\{16, 32\}$
Number of epochs	$\{2, 3, 4\}$
Learning rate	$\{5e-5, 3e-5, 2e-5\}$

Table 6.10: Default hyper-parameters' values of BERT.

In order to find the optimal hyper-parameters for our task, a search over these parameters (the number of epochs was extended to 8 and a learning rate of $1e-5$ was added) was run for all the models. Based on this search, we choose the values of the hyper-parameters as shown in Table 6.10.

	BERT (base)	DistilBERT (base)	RoBERTa (base)	BERTweet (base)
Hyperparameter	**Value**	**Value**	**Value**	**Value**
Maximum sequence length	128	128	128	128
Batch size	16	16	32	16
Number of epochs	4	4	8	5
Learning rate	$2e-5$	$3e-5$	$1e-5$	$1e-5$
Dropout rate	0.5	0.5	0.5	0.5
Manual seed	42	42	42	42

Table 6.11: Hyper-parameters of our BERT based models.

To implement the models, we used the PyTorch [105] framework and the Transformers [106] library. We trained the neural networks on a GTX 2060 (8GB) GPU.

6.3.2.3 Results

The models are evaluated on the test (unseen) data sets. To evaluate the classification performance, we use three metrics: accuracy, macro average and weighted average F1 score. The following tables present the classification reports of the aforementioned fine-tuned models. Also, we illustrate the confusion matrices (Figures 6.7, 6.8, 6.9, and 6.10).

	Twitter Data Set				SST-5 Data Set			
sentiment	**precision**	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
very negative	0.20	0.12	0.15	8	0.43	0.45	0.44	279
negative	0.33	0.42	0.37	52	0.48	0.48	0.48	633
neutral	0.83	0.79	0.81	318	0.32	0.41	0.36	389
positive	0.48	0.55	0.52	109	0.49	0.49	0.49	510
very positive	0.50	0.35	0.41	34	0.69	0.49	0.57	399
accuracy			0.66	521			0.47	2210
macro avg	0.47	0.45	0.45	521	0.48	0.46	0.47	2210
weighted avg	0.68	0.66	0.67	521	0.49	0.47	0.47	2210

Table 6.12: Classification Report of BERT (base) Model.

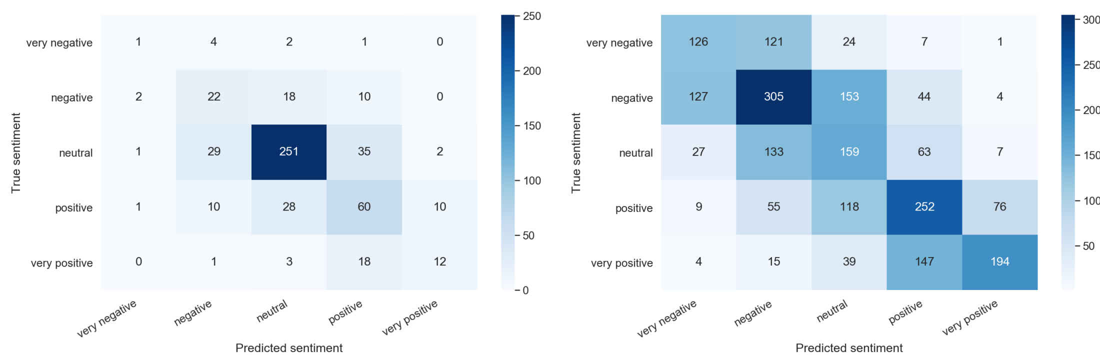

Figure 6.7: Confusion Matrix of BERT (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set.

	Twitter Data Set				SST-5 Data Set			
sentiment	**precision**	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
very negative	0.14	0.12	0.13	8	0.42	0.44	0.43	279
negative	0.27	0.42	0.33	52	0.46	0.52	0.49	633
neutral	0.83	0.72	0.77	318	0.31	0.31	0.31	389
positive	0.43	0.50	0.46	109	0.42	0.43	0.43	510
very positive	0.60	0.53	0.56	34	0.63	0.49	0.55	399
accuracy			0.62	521			0.45	2210
macro avg	0.45	0.46	0.45	521	0.45	0.44	0.44	2210
weighted avg	0.66	0.62	0.64	521	0.45	0.45	0.45	2210

Table 6.13: Classification Report of DistilBERT (base) Model.

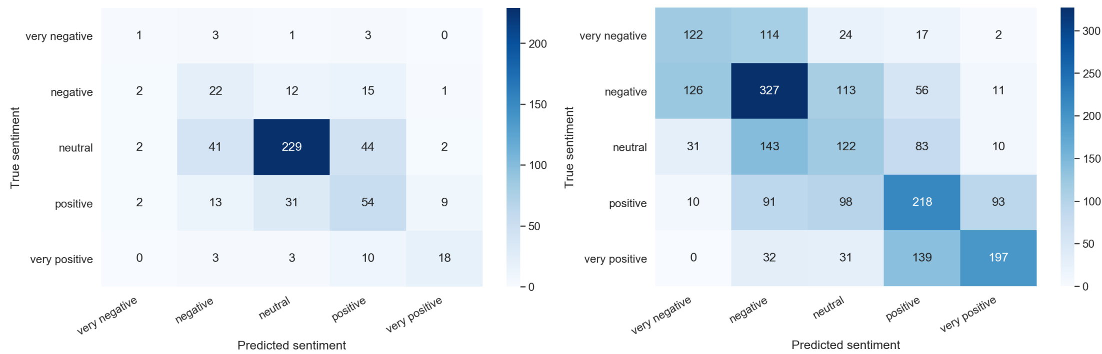

Figure 6.8: Confusion Matrix of DistilBERT (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set.

	Twitter Data Set				SST-5 Data Set			
sentiment	**precision**	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
very negative	0.12	0.12	0.12	8	0.46	0.64	0.54	279
negative	0.31	0.62	0.41	52	0.58	0.53	0.55	633
neutral	0.93	0.68	0.78	318	0.37	0.41	0.39	389
positive	0.41	0.43	0.42	109	0.55	0.50	0.52	510
very positive	0.41	0.76	0.54	34	0.66	0.59	0.63	399
accuracy			0.62	521			0.53	2210
macro avg	0.44	0.52	0.46	521	0.52	0.53	0.53	2210
weighted avg	0.71	0.62	0.64	521	0.54	0.53	0.53	2210

Table 6.14: Classification Report of RoBERTa (base) Model.

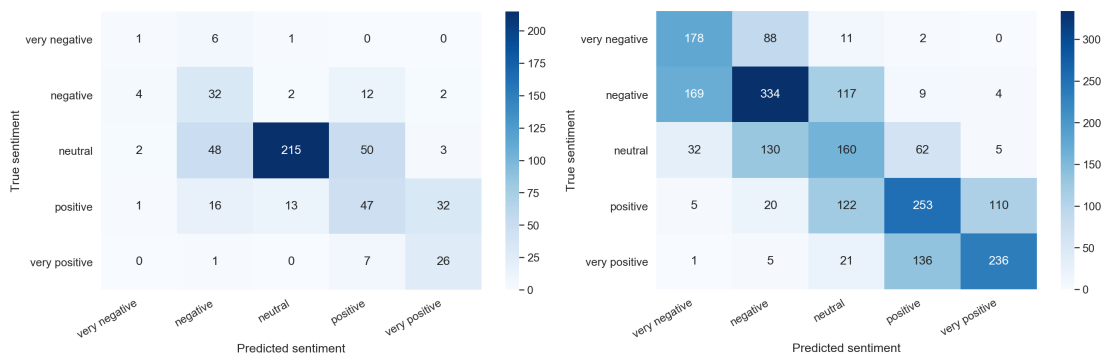

Figure 6.9: Confusion Matrix of RoBERTa (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set.

	Twitter Data Set				SST-5 Data Set			
sentiment	**precision**	**recall**	**f1-score**	**support**	**precision**	**recall**	**f1-score**	**support**
very negative	0.33	0.38	0.35	8	0.49	0.65	0.56	279
negative	0.34	0.56	0.42	52	0.59	0.47	0.52	633
neutral	0.89	0.71	0.79	318	0.39	0.52	0.44	389
positive	0.45	0.50	0.47	109	0.57	0.51	0.54	510
very positive	0.51	0.71	0.59	34	0.67	0.60	0.63	399
accuracy			0.65	521			0.53	2210
macro avg	0.50	0.57	0.53	521	0.54	0.55	0.54	2210
weighted avg	0.71	0.66	0.67	521	0.55	0.53	0.54	2210

Table 6.15: Classification Report of BERTweet (base) Model.

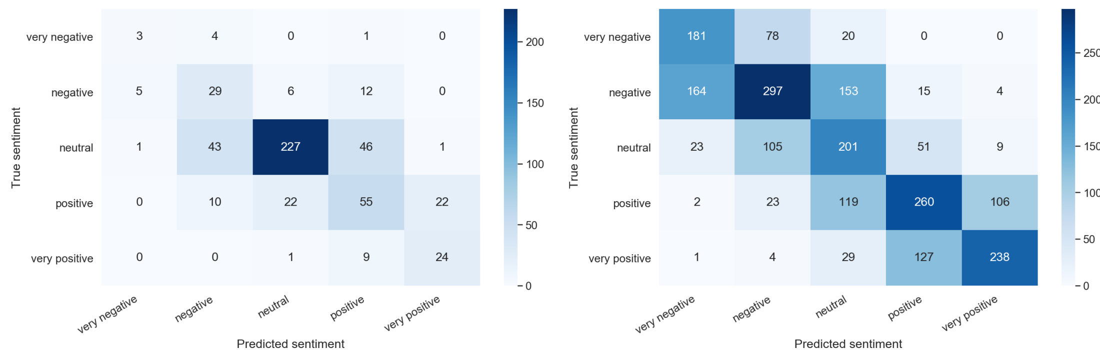

Figure 6.10: Confusion Matrix of BERTweet (base) Model (a) Twitter Sentiment Analysis Self-Driving Cars Data Set (b) SST-5 Data Set.

Based on the results, we see that the BERTweet model gives the best macro F1 score in both data sets. RoBERTa also gives satisfactory results, especially for the SST-5 data set. DistilBERT falls a little behind BERT-base, in both accuracy and F1 score. Note that most models have a problem identifying the class with the lowest proportion in the training Twitter data set, namely the "very negative" class. In the case of the SST-5 data set, "neutral" seems to be the most misclassified sentiment.

6.4 Summarized Results

	Twitter Data Set		SST-5 Data Set	
Model	**Accuracy**	**macro F1**	**Accuracy**	**macro F1**
2-BLSTM+Att	61%	42%	42%	41%
BERT (base)	66%	45%	47%	47%
DistilBERT (base)	62%	45%	45%	44%
RoBERTa (base)	62%	46%	53%	53%
BERTweet (base)	65%	53%	53%	54%
SOTA			59% [107]	

Table 6.16: Comparison of the models across the data sets.

Table 6.16 compares the performance of the LSTM based architecture augmented with a self-attention mechanism we proposed in Section 6.3.1, with the BERT-based models we described in Section 6.3.2. We observe that the best model in regards to

both accuracy and F1 score is the BERTweet model for all data sets. However, we still have not managed to match the SOTA performance on the SST-5 data set[2].

6.5 Classifying Unlabelled Tweets about Autonomous Vehicles

Twitter is a fantastic source of data, with a very large number of tweets sent per second. Those tweets can be downloaded and used to investigate mass opinion on particular issues, such as self-driving cars. As mentioned before, autonomous cars are controversial topics and the perception of the public plays a vital role in the acceptance of this technology.

We use the Twitter API[3] to collect relevant unlabelled data about AVs. Several keywords have been used during the data collection process, such as "autonomouscar", "autonomousvehicle", "automatedcar", "automatedvehicle", "driverlesscar", "driverlessvehicle", "driverless", "selfdrivingcar" and "selfdrivingvehicle". A total of $4,052$ unique tweets were collected. Table 6.17 depicts a sample of the unlabelled data set.

id	text
1	Why does anyone even want a driverless car?
2	impressive ~electric driverless flying taxi at your service
3	@RyanfDuffy Why I will never get into a driverless car.
4	Remember all the hype about driverless cars a few years ago? Yeah that was all transparent horseshit
5	This futuristic #bus is 100% Electric Vehicle and Self-Driving! https://t.co/3liZsBoX9b
6	Police share photos of smiling man riding in backseat of driverless Tesla. https://t.co/B00qExzjAq

Table 6.17: A sample of the Unlabelled Tweets about Autonomous Vehicles Data Set.

We classify the collected tweets using the BERTweet model we trained in Section 6.3.2 on the Twitter sentiment analysis self-driving cars data set. In order to evaluate the results, we look at the sample tweets presented in Table 6.17 and illustrate the confidence of each sentiment of our model.

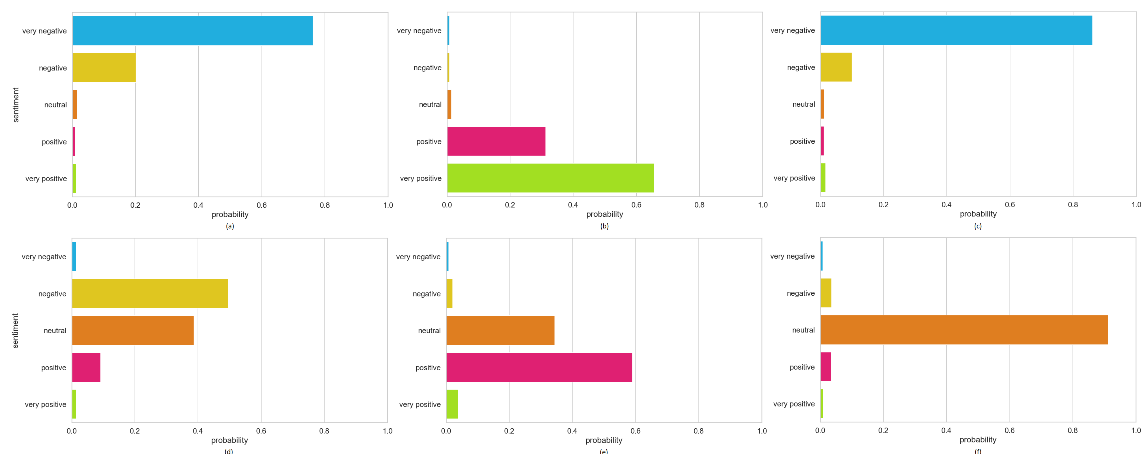

Figure 6.11: Confidence of each sentiment of sample tweet with (a) id=1, (b) id=2, (c) id=3, (d) id=4, (e) id=5, (f) id=6.

[2]https://paperswithcode.com/sota/sentiment-analysis-on-sst-5-fine-grained
[3]https://developer.twitter.com/en/docs

The confidence of the sentiments shown in Figure 6.11, based on our understanding, are really satisfactory, so we continue our analysis by illustrating the classification results of all 4,052 tweets, as shown in Figure 6.12. We observe that the vast majority of tweets seem to be neutral towards AVs. However, it seems that the negative opinions prevail over the positive ones. More specifically, 1.7% of the tweets were very negative, 11.8% were negative, 8.2% were positive and 2% were very positive.

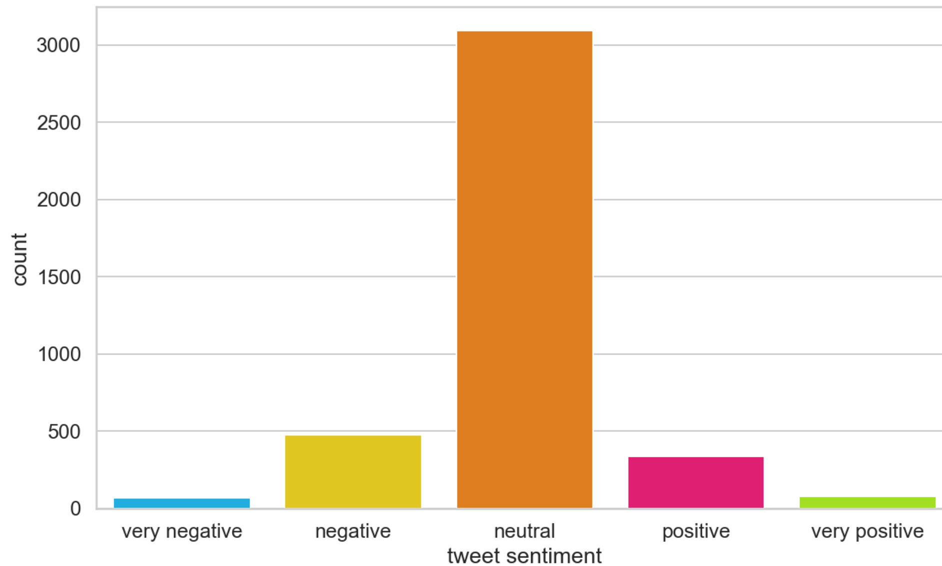

Figure 6.12: Opinions about Autonomous Vehicles.

Finally, we perform the pre-processing steps described in Section 6.2.2, and we calculate the unigrams of each sentiment using the tf–idf vectorizer. The results are presented in Table 6.18.

Very Negative		Negative		Neutral		Positive		Very Positive	
Unigram	**Count**	**Unigram**	**Count**	**Unigram**	**Count**	**Unigram**	**Count**	**Unigram**	**Count**
driverless	7.20	driverless	35.29	driverless	237.33	driverless	21.37	driverless	6.72
car	5.46	cars	25.86	waymo	172.30	cars	14.62	exciting	3.49
cars	3.96	industry	17.30	service	130.14	future	8.01	waymo	3.41
want	2.41	car	14.64	public	116.21	car	7.67	cars	3.07
think	2.13	tesla	13.44	phoenix	112.55	driving	6.28	fully	2.80
going	1.77	problem	10.98	ride	83.45	autonomous	6.27	car	2.48
hell	1.59	cause	10.28	fully	79.80	fully	6.17	ride	2.30
people	1.54	time	9.83	cars	78.36	waymo	5.69	public	2.03
pizza	1.42	autopilot	9.30	autonomous	77.64	self	4.76	milestone	2.02
wants	1.39	crash	9.19	taxi	73.05	phoenix	4.70	wait	1.99

Table 6.18: Top Unigrams of Unlabelled Tweets about Autonomous Vehicles.

Chapter 7

Conclusions

In this thesis, we discussed and compared methods to tackle the problem of the multi-class Twitter sentiment analysis. We implemented in TensorFlow a simple 2-layer bi-directional LSTM model with self-attention (2-BLSTM+Att), the embedding layer of which used a pre-trained Word2Vec model. In addition, we leveraged the PyTorch framework and the Transformers library for conducting experiments using SOTA models including BERT (base), DistilBERT (base), RoBERTa (base), and BERTweet (base). We also added a weighted loss to all models in order to tackle the class imbalance problem in the data sets we used. More specifically, our experiments conducted using the Twitter Sentiment Analysis Self-Driving Cars data set and the SST-5 data set. The Twitter data set consists of tweets about self-driving cars, and it is highly imbalanced. The SST-5 data set consists of movie reviews, and it is used for benchmarking SOTA models. We managed to obtain a F1 score of 0.53 in the Twitter test data set and 0.54 in the SST-5 test data set.

Using the model that gave the best F1 score, we classified $4,052$ unlabelled tweets about AVs that we collected from Twitter by using the Twitter API. The classifier was trained using the Twitter sentiment analysis self-driving cars training data set. We observed that the vast majority of the tweets (76.3%) were neutral towards autonomous cars. Furthermore, 1.7% were very negative, 11.8% negative, 8.2% positive and 2% very positive. So, our findings show that people are still cautious over this controversial technology.

Due to lack of resources, we could not deal with larger versions of BERT and bigger deep learning models in general. Research shows that models with far more parameters than the ones we used, augmented with knowledge-based methods, can achieve sufficient semantic context for accuracy of 70–80% in fine-grained sentiment analysis. So, an idea for future work is to experiment with expensive SOTA models. In addition, we can experiment with more data sets and perform a hyper-parameter optimization, since we did not extensively tune the hyper-parameters of our models. Therefore, further improvements may be possible with hyper-parameter tuning.

Acronyms

ACC Accuracy

Adam Adaptive Moment Estimation

AI Artificial Intelligence

ANN Artificial Neural Network

AV Autonomous Vehicle

bACC Balanced Accuracy

backprop Back-propagation

BCE Binary Cross-Entropy

BERT Bi-directional Encoder Representations from Transformers

BGD Batch Gradient Descent

biLM Bi-directional Language Model

BLSTM Bi-directional LSTM

BOW Bag-of-Words

BPTT Back-propagation Through Time

BRNN Bi-directional Recurrent Neural Network

CBOW Continuous Bag-of-Words

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

ELMo Embeddings from Language Models

FFNN Feed-Forward Neural Network

FN False Negative

FP False Positive

FPR False Positive Rate

GD Gradient Descent

GloVe Global Vectors

HMM Hidden Markov Models

idf Inverse Document Frequency

LDA Latent Dirichlet Allocation

LM Language Modeling

LReLU Leaky ReLU

LSA Latent Semantic Analysis

LSTM Long Short-Term Memory

MDP Markov Decision Process

ML Machine Learning

MLM Masked Language Model

NLP Natural Language Processing

NN Neural Network

NSP Next Sentence Prediction

PCA Principal Component Analysis

PP Perplexity

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

SGD Stochastic Gradient Descent

SOTA State-Of-The-Art

SST Stanford Sentiment Treebank

t-SNE t-Distributed Stochastic Neighbor Embedding

tf Term Frequency

tf–idf Term Frequency-Inverse Document Frequency

TL Transfer Learning

TN True Negative

TNR True Negative Rate

TP True Positive

TPR True Positive Rate

TSA Twitter Sentiment Analysis

Bibliography

[1] C. Hardy. (). “finding similarities between friends by way of social media using shallow and deep natural language processing”, [Online]. Available: `https://xtian.ai/finding-similarities-between-friends`.

[2] H. Rui, Y. Liu, and A. Whinston, “Whose and What Chatter Matters? The Impact of Tweets on Movie Sales”, *Decision Support Systems*, vol. 55, Oct. 2011. DOI: `10.2139/ssrn.1958068`.

[3] J. Bollen, H. Mao, and X.-J. Zeng, “Twitter Mood Predicts the Stock Market”, *Journal of Computational Science*, vol. 2, Oct. 2010. DOI: `10.1016/j.jocs.2010.12.007`.

[4] A. Tumasjan, T. Sprenger, P. Sandner, and I. Welpe, “Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment”, vol. 10, Jan. 2010.

[5] B. O’Connor, R. Balasubramanyan, B. Routledge, and N. Smith, “From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series”, vol. 11, Jan. 2010.

[6] A. Bermingham and A. Smeaton, “Classifying sentiment in microblogs: Is brevity an advantage?”, *Bermingham, Adam and Smeaton, Alan F. (2010) Classifying sentiment in microblogs: is brevity an advantage? In: CIKM 2010 - 19th International Conference on Information and Knowledge Management, 26-30 October 2010, Toronto, Canada. ISBN 978-1-4503-0099-5*, Jan. 2010. DOI: `10.1145/1871437.1871741`.

[7] J. Chung and E. Mustafaraj, “Can Collective Sentiment Expressed on Twitter Predict Political Elections?”, vol. 11, Jan. 2011.

[8] D. Gayo-Avello, “A Meta-Analysis of State-of-the-Art Electoral Prediction From Twitter Data”, *Social Science Computer Review*, vol. 31, Jun. 2012. DOI: `10.1177/0894439313493979`.

[9] J. Jansen, M. Zhang, K. Sobel, and A. Chowdury, “Twitter Power: Tweets as Electronic Word of Mouth”, *JASIST*, vol. 60, pp. 2169–2188, Nov. 2009. DOI: `10.1002/asi.21149`.

[10] K. Dong Sung and J. Kim, “Public Opinion Mining on Social Media: A Case Study of Twitter Opinion on Nuclear Power”, Jul. 2014, pp. 224–228. DOI: `10.14257/astl.2014.51.51`.

[11] N. Diakopoulos and D. Shamma, “Characterizing debate performance via aggregated twitter sentiment”, vol. 2, Jan. 2010, pp. 1195–1198. DOI: `10.1145/1753326.1753504`.

[12] M. Ghiassi, J. Skinner, and D. Zimbra, "Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network", *Expert Systems with Applications*, vol. 40, pp. 6266–6282, Nov. 2013. DOI: 10.1016/j.eswa.2013.05.057.

[13] M. Hagen, M. Potthast, M. Büchner, and B. Stein, "Webis: An Ensemble for Twitter Sentiment Detection", Jan. 2015, pp. 582–589. DOI: 10.18653/v1/S15-2097.

[14] B. Krawczyk, B. Mcinnes, and A. Cano, "Sentiment Classification from Multi-class Imbalanced Twitter Data Using Binarization", Jun. 2017, ISBN: 978-3-319-59649-5. DOI: 10.1007/978-3-319-59650-1_3.

[15] A. Vanzo, D. Croce, and R. Basili, "A context-based model for Sentiment Analysis in Twitter", Aug. 2014.

[16] G. Amati, M. Bianchi, and G. Marcone, "Sentiment Estimation on Twitter", vol. 1127, Jan. 2014.

[17] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas, "Short text classification in twitter to improve information filtering", Jan. 2010, pp. 841–842. DOI: 10.1145/1835449.1835643.

[18] M. Boia, B. Faltings, C. Musat, and P. Pu, "A :) Is Worth a Thousand Words: How People Attach Sentiment to Emoticons and Words in Tweets", Sep. 2013, pp. 345–350. DOI: 10.1109/SocialCom.2013.54.

[19] Y. Priyadarshana, K. Gunathunga, K. N. N. Perera, L. Ranathunga, P. Karunaratne, and T. Thanthriwatta, "Sentiment analysis: Measuring sentiment strength of call centre conversations", in *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2015, pp. 1–9. DOI: 10.1109/ICECCT.2015.7226053.

[20] R. Srivastava and M. : S. Bhatia, "Quantifying modified opinion strength: A fuzzy inference system for Sentiment Analysis", Aug. 2013, pp. 1512–1519, ISBN: 978-1-4799-2432-5. DOI: 10.1109/ICACCI.2013.6637404.

[21] M. Ghiassi, D. Zimbra, and S. Lee, "Targeted Twitter Sentiment Analysis for Brands Using Supervised Feature Engineering and the Dynamic Architecture for Artificial Neural Networks", *Journal of Management Information Systems*, vol. 33, pp. 1034–1058, Oct. 2016. DOI: 10.1080/07421222.2016.1267526.

[22] P. Turney, "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews", *Computing Research Repository - CORR*, pp. 417–424, Dec. 2002. DOI: 10.3115/1073083.1073153.

[23] S.-M. Kim and E. Hovy, "Determining the sentiment of opinions", Jan. 2004. DOI: 10.3115/1220355.1220555.

[24] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment Classification Using Machine Learning Techniques", *EMNLP*, vol. 10, Jun. 2002. DOI: 10.3115/1118693.1118704.

[25] M. Gamon, "Sentiment classification on customer feedback data: Noisy data, large feature vectors, and the role of linguistic analysis", Jan. 2004. DOI: 10.3115/1220355.1220476.

[26] W. Gao and F. Sebastiani, "From Classification to Quantification in Tweet Sentiment Analysis", *Social Network Analysis and Mining*, vol. 6, 19:1–19:22, Apr. 2016. DOI: 10.1007/s13278-016-0327-z.

[27] O. Araque, I. Corcuera-Platas, J. F. Sánchez-Rada, and C. Iglesias, "Enhancing Deep Learning Sentiment Analysis with Ensemble Techniques in Social Applications", *Expert Systems with Applications*, vol. 77, Feb. 2017. DOI: 10.1016/j.eswa.2017.02.002.

[28] J. Zmud, I. Sener, and J. Wagner, "Self-Driving Vehicles Determinants of Adoption and Conditions of Usage", *Transportation Research Record Journal of the Transportation Research Board*, vol. 2565, pp. 57–64, Jan. 2016. DOI: 10.3141/2565-07.

[29] N. J. Greig, *Driver less cars - the view of the consumer?*, 2017.

[30] N. Merat, R. Madigan, and S. Nordhoff, "Human factors, user requirements, and user acceptance of ride-sharing in automated vehicles", in. Feb. 2017.

[31] S. Greaves, B. Smith, T. Arnold, D. Olaru, and A. Collins, "Autonomous vehicles down under: An empirical investigation of consumer sentiment", English, in *Australasian Transport Research Forum 2018 Proceedings*, 40th Australasian Transport Research Forum, ATRF 2018 ; Conference date: 30-10-2018 Through 01-11-2018, Australasian Transport Research Forum, 2018.

[32] T. Liljamo, H. Liimatainen, and M. Pöllänen, "Attitudes and concerns on automated vehicles", *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 59, pp. 24–44, Nov. 2018. DOI: 10.1016/j.trf.2018.08.010.

[33] S. Nordhoff, J. de Winter, M. Kyriakidis, B. Arem, and R. Happee, "Acceptance of Driverless Vehicles: Results from a Large Cross-National Questionnaire Study", *Journal of advanced transportation*, vol. 2018, Apr. 2018. DOI: 10.1155/2018/5382192.

[34] I. Sener, J. Zmud, and T. Williams, "Measures of baseline intent to use automated vehicles: A case study of Texas cities", *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 62, pp. 66–77, Apr. 2019. DOI: 10.1016/j.trf.2018.12.014.

[35] J. Moody, N. Bailey, and J. Zhao, "Public perceptions of autonomous vehicle safety: An international comparison", *Safety Science*, vol. 121, Jul. 2019. DOI: 10.1016/j.ssci.2019.07.022.

[36] A. Rezaei and B. Caulfield, "Examining public acceptance of autonomous mobility", *Travel Behaviour and Society*, vol. 21, pp. 235–246, Oct. 2020. DOI: 10.1016/j.tbs.2020.07.002.

[37] T. Mitchell, *Machine Learning*. McGraw-Hill, 1997, p. 2, ISBN: 978-0-07-042807-2.

[38] (2019). "Supervised Learning vs Unsupervised Learning. Which is better?", [Online]. Available: https://lawtomated.com/supervised-vs-unsupervised-learning-which-is-better/.

[39] G. Koelpin. (2020). "Stop One-Hot Encoding Your Categorical Variables.", [Online]. Available: https://morioh.com/p/811a5d22bbca.

[40] K. P. F.R.S., "LIII. On lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.

[41] L. van der Maaten and G. Hinton, "Viualizing data using t-SNE", *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, Nov. 2008.

[42] S. Gupta. (2020). "Most Common Loss Functions in Machine Learning", [Online]. Available: https://towardsdatascience.com/most-common-loss-functions-in-machine-learning-c7212a99dae0.

[43] S. Lynn-Evans. (2018). "Ten techniques learned from fast.ai", [Online]. Available: https://blog.floydhub.com/ten-techniques-from-fast-ai/.

[44] S. Ruder, "An overview of gradient descent optimization algorithms", Sep. 2016.

[45] A. Bhattacharya, C. Alp, H. Memisoglu, and M. Spehlmann. (). "Variance Reduction Methods", [Online]. Available: http://pages.cs.wisc.edu/~spehlmann/cs760/_site//project/2017/05/04/intro.html.

[46] J. Jordan. (2018). "Setting the learning rate of your neural network.", [Online]. Available: https://www.jeremyjordan.me/nn-learning-rate/.

[47] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, Jul. 2011.

[48] T. Tieleman and G. Hinton, "Lecture 6.5 – RMSProp: Divide the gradient by a running average of its recent magnitude", in *Neural Networks for Machine Learning*, Coursera, 2012.

[49] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *International Conference on Learning Representations*, Dec. 2014.

[50] N. Qian, "On the momentum term in gradient descent learning algorithms", *Neural networks: the official journal of the International Neural Network Society*, vol. 12, no. 1, pp. 145–151, Jan. 1999, ISSN: 0893-6080. DOI: 10.1016/s0893-6080(98)00116-6. [Online]. Available: https://doi.org/10.1016/s0893-6080(98)00116-6.

[51] J. Mohajon. (2020). "Confusion Matrix for Your Multi-Class Machine Learning Model", [Online]. Available: https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

[52] (2008), [Online]. Available: https://commons.wikimedia.org/wiki/File:Neuron_ro.svg.

[53] R. Kotikalapudi. (2020). "Exploring alternative neural computational models", [Online]. Available: https://raghakot.github.io/2017/01/03/Exploring-alternative-neural-computational-models.html.

[54] A. Castrounis. (). "AI, Deep Learning, and Neural Networks Explained", [Online]. Available: https://www.innoarchitech.com/blog/artificial-intelligence-deep-learning-neural-networks-explained.

[55] (2017). "Creating & Visualizing Neural Network in R", [Online]. Available: https://www.analyticsvidhya.com/blog/2017/09/creating-visualizing-neural-network-in-r/.

[56] D. Sheehan. (2017). "Visualising Activation Functions in Neural Networks", [Online]. Available: https://dashee87.github.io/deep%5C%20learning/visualising-activation-functions-in-neural-networks/.

[57] D. Nautiyal. (2020). "Underfitting and Overfitting in Machine Learning", [Online]. Available: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/.

[58] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, Jun. 2014.

[59] J. Dean, "Large-Scale Deep Learning for Intelligent Computer Systems", Google TechTalk, 2016, [Online]. Available: https://www.youtube.com/watch?v=QSaZGT4-6EY.

[60] (2019). "Recurrent Neural Network (RNN)", [Online]. Available: https://docs.paperspace.com/machine-learning/wiki/recurrent-neural-network-rnn.

[61] A. Karpathy. (2015). "The Unreasonable Effectiveness of Recurrent Neural Networks", [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[62] J. Kvita. (2016). "Visualizations of RNN units", [Online]. Available: https://kvitajakub.github.io/2016/04/14/rnn-diagrams/.

[63] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks", *Signal Processing, IEEE Transactions on*, vol. 45, pp. 2673–2681, Dec. 1997. DOI: 10.1109/78.650093.

[64] C. Olah. (2015). "Neural Networks, Types, and Functional Programming", [Online]. Available: http://colah.github.io/posts/2015-09-NN-Types-FP/.

[65] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory", *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[66] A. Mittal. (2019). "Understanding RNN and LSTM", [Online]. Available: https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[67] D. Bahdanau, K. Cho, and Y. Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, 2016. arXiv: 1409.0473 [cs.CL].

[68] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to Sequence Learning with Neural Networks*, 2014. arXiv: 1409.3215 [cs.CL].

[69] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014. arXiv: 1406.1078 [cs.CL].

[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention Is All You Need*, 2017. arXiv: 1706.03762 [cs.CL].

[71] M.-T. Luong, H. Pham, and C. D. Manning, *Effective Approaches to Attention-based Neural Machine Translation*, 2015. arXiv: 1508.04025 [cs.CL].

[72] A. Graves, G. Wayne, and I. Danihelka, *Neural Turing Machines*, 2014. arXiv: 1410.5401 [cs.NE].

[73] J. Cheng, L. Dong, and M. Lapata, *Long Short-Term Memory-Networks for Machine Reading*, 2016. arXiv: 1601.06733 [cs.CL].

[74] Z. Yang, Y. Shen, R. Zhou, F. Yang, Z. Wan, and Z. Zhou, "A transfer learning fault diagnosis model of distribution transformer considering multi-factor situation evolution", *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 15, Nov. 2019. DOI: 10.1002/tee.23024.

[75] K. Weiss, T. Khoshgoftaar, and D. Wang, "A survey of transfer learning", *Journal of Big Data*, vol. 3, May 2016. DOI: 10.1186/s40537-016-0043-6.

[76] S. J. Pan and Q. Yang, "A Survey on Transfer Learning", *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. DOI: 10.1109/TKDE.2009.191.

[77] S. Ruder. (2019). "Neural Transfer Learning for Natural Language Processing (PhD thesis)", [Online]. Available: https://ruder.io/thesis/.

[78] F. Chollet. (2017). "Deep Learning for Text", [Online]. Available: https://freecontent.manning.com/deep-learning-for-text/.

[79] P. Sawhney. (2020). "Introduction to Stemming vs Lemmatization (NLP)", [Online]. Available: https://laptrinhx.com/introduction-to-stemming-vs-lemmatization-nlp-1503911125/.

[80] A. Sharma. (2020). "Understanding Latent Dirichlet Allocation (LDA)", [Online]. Available: https://www.mygreatlearning.com/blog/understanding-latent-dirichlet-allocation/.

[81] A. Maiolo. (2015). "Comparing n-gram models", [Online]. Available: https://web.archive.org/web/20180427050745/http://recognize-speech.com/language-model/n-gram-model/comparison.

[82] Y. Bengio, R. Ducharme, and P. Vincent, "A Neural Probabilistic Language Model", vol. 3, Jan. 2000, pp. 932–938. DOI: 10.1162/153244303322533223.

[83] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781.

[84] J. Pennington, R. Socher, and C. Manning, "Glove: Global Vectors for Word Representation", vol. 14, Jan. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162.

[85] M. Riva. (2021). "Word Embeddings: CBOW vs Skip-Gram", [Online]. Available: https://www.baeldung.com/cs/word-embeddings-cbow-vs-skip-gram.

[86] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations", Feb. 2018.

[87] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. arXiv: 1810.04805 [cs.CL].

[88] S. Ruder. (2019). "The State of Transfer Learning in NLP", [Online]. Available: https://ruder.io/state-of-transfer-learning-in-nlp/.

[89] P. Joshi. (2019). "A Step-by-Step NLP Guide to Learn ELMo for Extracting Features from Text", [Online]. Available: https://www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/.

[90] J. Alammar. (2018). "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)", [Online]. Available: http://jalammar.github.io/illustrated-bert/.

[91] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank", in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://www.aclweb.org/anthology/D13-1170.

[92] D. Chen and C. Manning, "A Fast and Accurate Dependency Parser using Neural Networks", Jan. 2014, pp. 740–750. DOI: 10.3115/v1/D14-1082.

[93] C. Sievert and K. Shirley, "LDAvis: A method for visualizing and interpreting topics", in *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, Baltimore, Maryland, USA: Association for Computational Linguistics, Jun. 2014, pp. 63–70. DOI: 10.3115/v1/W14-3110. [Online]. Available: https://www.aclweb.org/anthology/W14-3110.

[94] F. Godin, B. Vandersmissen, W. De Neve, and R. Van de Walle, "Multimedia Lab @ ACL W-NUT NER Shared Task: Named Entity Recognition for Twitter Microposts using Distributed Word Representations", Jul. 2015. DOI: 10.18653/v1/W15-4322.

[95] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures", *Neural networks : the official journal of the International Neural Network Society*, vol. 18, pp. 602–10, Jul. 2005. DOI: 10.1016/j.neunet.2005.06.042.

[96] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, and S. Lehmann, "Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm", Jan. 2017, pp. 1615–1625. DOI: 10.18653/v1/D17-1169.

[97] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical Attention Networks for Document Classification", Jan. 2016, pp. 1480–1489. DOI: 10.18653/v1/N16-1174.

[98] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu,

and Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: `http://tensorflow.org/`.

[99] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, “Scikit-learn: Machine learning in Python”, *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[100] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”, *CoRR*, vol. abs/1910.01108, 2019. arXiv: `1910.01108`. [Online]. Available: `http://arxiv.org/abs/1910.01108`.

[101] 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”, *CoRR*, vol. abs/1907.11692, 2019. arXiv: `1907.11692`. [Online]. Available: `http://arxiv.org/abs/1907.11692`.

[102] D. Q. Nguyen, T. Vu, and A. T. Nguyen, “BERTweet: A pre-trained language model for English Tweets”, *CoRR*, vol. abs/2005.10200, 2020. arXiv: `2005.10200`. [Online]. Available: `https://arxiv.org/abs/2005.10200`.

[103] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, “Unsupervised Cross-lingual Representation Learning at Scale”, *CoRR*, vol. abs/1911.02116, 2019. arXiv: `1911.02116`. [Online]. Available: `http://arxiv.org/abs/1911.02116`.

[104] I. Loshchilov and F. Hutter, “Fixing Weight Decay Regularization in Adam”, *CoRR*, vol. abs/1711.05101, 2017. arXiv: `1711.05101`. [Online]. Available: `http://arxiv.org/abs/1711.05101`.

[105] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “PyTorch: An Imperative Style, High-Performance Deep Learning Library”, in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[106] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, *HuggingFace's Transformers: State-of-the-art Natural Language Processing*, 2020. arXiv: `1910.03771 [cs.CL]`.

[107] Z. Sun, C. Fan, Q. Han, X. Sun, Y. Meng, F. Wu, and J. Li, *Self-Explaining Structures Improve NLP Models*, Dec. 2020.