ARISTOTLE UNIVERSITY OF THESSALONIKI

Sentiment Analysis using Deep Learning methods

by

Dionysios Karamouzas - 8827

A thesis submitted in partial fulfillment for the
Undergraduate degree

in the
Faculty of Engineering
School of Electrical and Computer Engineering

Supervision: Prof. Ioannis Pitas
Co-Supervision: Dr. Ioannis Mademlis

August 1st, 2022

Declaration of Authorship

I, Dionysios Karamouzas, declare that this thesis titled, "Sentiment Analysis using Deep Learning methods" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Acknowledgements

I would like to express my gratitude to my primary supervisor, Ioannis Pitas, who trusted and guided me throughout this project. I would also like to thank my second supervisor, Ioannis Mademlis, for responding to my questions and messages in a regular basis and helping me finalize this project. Last but not least, I want to thank my family and friends who supported me in tough moments.

Contents

Declaration of Authorship i

Acknowledgements ii

List of Figures vii

List of Tables x

1 Natural Language Processing 1
- 1.1 What is NLP ? . . . 1
  - 1.1.1 Short definitions . . . 1
  - 1.1.2 Natural Language . . . 2
  - 1.1.3 Challenge . . . 2
  - 1.1.4 From Linguistics to NLP . . . 2
    - 1.1.4.1 Linguistics . . . 2
    - 1.1.4.2 Computational Linguistics . . . 3
    - 1.1.4.3 CL vs NLP . . . 3
- 1.2 History . . . 4
  - 1.2.1 Roots . . . 4
  - 1.2.2 Three periods . . . 4
    - 1.2.2.1 Symbolic NLP (classical programming) . . . 4
      - Milestones . . . 4
    - 1.2.2.2 Statistical NLP – Machine Learning . . . 5
      - Machine learning . . . 5
        - Supervised learning . . . 6
        - Unsupervised learning . . . 6
      - Why Statistical NLP . . . 7
      - Milestones . . . 7
    - 1.2.2.3 Neural NLP . . . 7
      - Artificial Neural Network (ANN) . . . 7
      - Milestones . . . 8
- 1.3 Methods: Rules, statistics, neural networks . . . 10
  - 1.3.1 Rules . . . 10
  - 1.3.2 Statistics . . . 10
    - 1.3.2.1 Why ML ? . . . 11

1.3.2.2 ML Algorithms for NLP . . . 11
Supervised ML . . . 11
Unsupervised ML . . . 12
1.3.3 Neural Networks . . . 13
1.3.3.1 Why NNs . . . 13
NNs Feature Learning . . . 13
NNs End-to-End . . . 14
1.3.3.2 Types of NNs for NLP . . . 14
Embedding Layer . . . 14
Multilayer Perceptron (MLP) . . . 15
Convolutional Neural Networks (CNNs) . . . 16
Recurrent Neural Networks (RNNs) . . . 17
Long Short Term Memory networks . . . 18
RNN vs CNN . . . 19
Hybrid NNs . . . 20
CNN + LSTM . . . 20
LSTM + CNN . . . 20
CNN // LSTM . . . 21
1.4 Word representations . . . 22
1.4.1 Fixed representations . . . 22
1.4.1.1 One-hot Encoding . . . 23
1.4.1.2 Bag of words (BOW) . . . 23
1.4.1.3 Term Frequency-Inverse Document Frequency (TF-IDF) . . . 24
1.4.1.4 Fixed representations - Pros & Cons . . . 24
1.4.2 Distributed representations (embeddings) . . . 25
1.4.2.1 Classic word embeddings . . . 25
Word2Vec . . . 26
Skip-gram training . . . 26
Global Vectors (GloVe) . . . 28
FastText . . . 29
Classic word embeddings problem . . . 29
1.4.2.2 Contextualized word embeddings . . . 29
Contextualized Vectors (CoVe) . . . 30
ELMo . . . 30
Transformer . . . 31
Transformer vs RNN . . . 31
Transformer visual walkthrough . . . 31
Pretrained language models based on Transformer . . . 38
OpenAI Generative Pre-trained Transformer (GPT) . . . 38
Bidirectional Encoder Representations from Transformers (BERT) . . . 38
1.5 Common NLP tasks . . . 41
1.5.1 Text and speech processing . . . 41
1.5.2 Morphological analysis . . . 42
1.5.3 Syntactic analysis . . . 42
1.5.4 Lexical semantics . . . 42
1.5.5 Relational semantics . . . 42

1.5.6 Discourse . . . 42
1.5.7 Higher-level NLP applications . . . 43

2 Sentiment Analysis 44
2.1 About SA . . . 44
2.1.1 What is SA? . . . 44
2.1.2 Applications of SA . . . 45
2.2 A Text Classification Task . . . 45
2.2.1 Levels of granularity . . . 45
2.2.2 Baseline algorithms . . . 45
2.2.2.1 Rule (lexicon)-based methods . . . 46
TextBlob . . . 46
VADER . . . 46
SentiWordNet . . . 47
2.2.2.2 Supervised ML methods . . . 47
2.2.3 Text Pre-processing . . . 48
2.2.3.1 Common pre-processing steps for SA . . . 48
Stemming/Lemmatization . . . 49
Tokenization . . . 49
Text Vectorization . . . 50
2.2.4 Sentiment Analysis with Neural Networks (RNN, CNN, BERT) . . . 50
2.2.4.1 Main idea . . . 50
2.2.4.2 Recurrent Neural Networks (RNNs) . . . 51
Simple RNN . . . 52
Problem with simple RNNs – short memory . . . 53
Long-Short Term Memory (LSTM) . . . 53
Text classification example with LSTM . . . 54
Stacked LSTM . . . 55
Bidirectional LSTM . . . 56
2.2.4.3 Convolutional Neural Networks (CNNs) . . . 57
Convolutional Kernels . . . 57
Convolution over Word Sequences . . . 58
Recognizing General Patterns . . . 59
1D Convolutions . . . 60
Multiple Kernels . . . 60
Max-pooling . . . 60
Text classification example with CNN . . . 61
2.2.4.4 CNN+LSTM Hybrid . . . 62
2.2.4.5 Text classification with BERT . . . 63

3 Public Opinion Monitoring through Collective Semantic Analysis of Tweets 65
3.1 Abstract . . . 65
3.2 Introduction . . . 65
3.3 Related Work . . . 67
3.3.1 Public opinion description . . . 67
3.3.1.1 Non-semantic methods . . . 68
3.3.1.2 Semantic methods without aggregation . . . 68

3.3.1.3 Semantic methods with aggregation . . . 69
3.3.1.4 Semantic analysis dimensions and algorithms . . . 70
3.3.2 Timeseries forecasting . . . 71
3.4 Proposed Mechanism . . . 72
3.4.1 Step 1: Selecting the desired pool of tweets . . . 72
3.4.2 Step 2: Individual descriptor extraction per tweet . . . 72
3.4.2.1 Training Datasets . . . 73
3.4.2.2 Text Preprocessing . . . 74
3.4.2.3 Neural Models . . . 74
3.4.2.4 Hyperparameters . . . 75
3.4.3 Step 3: Aggregation . . . 76
3.5 Evaluation . . . 76
3.5.1 Datasets . . . 77
3.5.2 Analysis 1: Timeseries Forecasting . . . 77
3.5.2.1 Implementation . . . 78
3.5.2.2 Metrics and Results . . . 79
3.5.3 Analysis 2: Visualizations and Qualitative Evaluation . . . 82
3.6 Discussion . . . 92
3.7 Conclusions . . . 93

4 Neural Knowledge Transfer for Improved Sentiment Analysis in Texts with Figurative Language 94
4.1 Abstract . . . 94
4.2 Introduction . . . 94
4.3 Related Work . . . 95
4.3.1 Sentiment Analysis on Figurative Language . . . 95
4.3.2 Neural Knowledge Transfer . . . 96
4.4 Proposed method . . . 97
4.5 Quantitative Evaluation . . . 99
4.5.1 Implementation Details . . . 99
4.5.2 Evaluation Setup . . . 100
4.6 Conclusions . . . 100

Bibliography 102

List of Figures

1.1 The major levels of linguistic structure. . . . 3
1.2 Flowchart - A diagrammatic representation of an algorithm. . . . 4
1.3 Machine learning approach vs Rule based approach . . . 5
1.4 Neural NLP history. . . . 10
1.5 Example use of embedding layer in a neural network. . . . 15
1.6 Multilayer Perceptron with five hidden layers. . . . 16
1.7 CNN model used for classifying movie reviews. . . . 17
1.8 The unfolded representation of the RNN's repeating module. . . . 17
1.9 The structure of an LSTM unit . . . 18
1.10 LSTM used for classifying movie reviews. . . . 19
1.11 CNN + LSTM example architecture. . . . 20
1.12 LSTM + CNN example architecture. . . . 21
1.13 CNN // LSTM example architecture. . . . 22
1.14 One-hot encoding example. . . . 23
1.15 Bag of words encoding example. . . . 23
1.16 TF-IDF encoding example. . . . 24
1.17 Skip-gram vs CBOW. . . . 26
1.18 Concept of Skip-gram training task. . . . 26
1.19 Negative Sampling example . . . 27
1.20 The training dataset of Skip-gram model. . . . 27
1.21 The embeddings look up matrices of Skip-gram model. . . . 28
1.22 The training process of Skip-gram model. . . . 28
1.23 The encoders and decoders stacks of Transformer. . . . 32
1.24 The inside structure of encoder-decoder blocks. . . . 32
1.25 Encoder data flow. . . . 33
1.26 Query, Key and Value vectors. . . . 34
1.27 The Self-attention calculation steps. . . . 34
1.28 The Self-attention calculation steps in matrix form. . . . 35
1.29 The Multi-headed attention calculation. . . . 35
1.30 Addition of positional encodings to account for the order of words. . . . 36
1.31 End to end machine translation with Transformer. . . . 37
1.32 Linear and Softmax layers convert the Decoder stack output into a word. . . . 38
1.33 Masked Language Model task. . . . 40
1.34 Next sentence prediction task. . . . 40
1.35 The encoder stack of BERT produces contextualized word embeddings. . . . 41

2.1 Algorithms used for Sentiment Analysis. . . . 46
2.2 Stemming vs Lemmatization. . . . 49

2.3 Tokenization example. . . . 50
2.4 Three word embeddings. . . . 50
2.5 General sentiment analysis (text classification) schema. . . . 51
2.6 Sequence processing with a simple RNN model. . . . 52
2.7 Calculations inside the simple RNN cell. . . . 53
2.8 Simple RNN short memory problem. . . . 53
2.9 The LSTM cell. . . . 54
2.10 Sentiment Analysis of movie reviews using LSTM. . . . 55
2.11 Calculations performed by the LSTM cell at each time step. . . . 55
2.12 Sentiment Analysis example with Stacked LSTM. . . . 56
2.13 Sentiment Analysis example with Bi-LSTM. . . . 57
2.14 A convolutional kernel. . . . 58
2.15 Convolution operation applied on text. . . . 59
2.16 Example of convolutional kernel capturing positive features. . . . 59
2.17 1D convolution output. . . . 60
2.18 Max-pooling applied to the 1D convolution output. . . . 61
2.19 Sentiment Analysis (text classification) example using CNN. . . . 62
2.20 Sentiment Analysis example using CNN+LSTM architecture. . . . 63
2.21 Text classification example using the pre-trained BERT. . . . 63

3.1 Quantitative public opinion forecasting using the proposed mechanism/semantic descriptor. . . . 72
3.2 Daily number of tweets for Democrats (Dems) and Republicans (Reps) in 2016 dataset. The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). . . . 81
3.3 Daily number of tweets for Democrats (Dems) and Republicans (Reps) in the 2020 dataset. The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). . . . 81
3.4 Per-channel day-by-day values of the 4D timeseries constructed from the 2016 data using the proposed descriptor and the mean aggregation strategy, separately for Democrats (Dems) and Republicans (Reps). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). . . . 82
3.5 Per-channel day-by-day values of the 4D timeseries constructed from the 2020 data using the proposed descriptor and the mean aggregation strategy, separately for Democrats (Dems) and Republicans (Reps). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). . . . 83
3.6 PCA-based 2D visualization of the constructed 4D timeseries for the Democrats, using a mean aggregation strategy, across the entire 2016 dataset time range (163 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). . . . 85
3.7 PCA-based 2D visualization of the constructed 4D timeseries for the Republicans, using a mean aggregation strategy, across the entire 2016 dataset time range (163 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). . . . 86

3.8 PCA-based 2D visualization of the constructed 4D timeseries for the Democrats, using a mean aggregation strategy, across the entire 2020 dataset time range (25 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). 86
3.9 PCA-based 2D visualization of the constructed 4D timeseries for the Republicans, using a mean aggregation strategy, across the entire 2020 dataset time range (25 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second). 87
3.10 Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Democrats on Nov 9, 2016 (the day after election). 87
3.11 Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Republicans on Nov 9, 2016 (the day after election). 88
3.12 Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Democrats on Nov 4, 2020 (the day after the election). 88
3.13 Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Republicans on Nov 4, 2020 (the day after the election). 89
3.14 Number of tweets concerning Democrats, separately for each class of the four descriptor dimensions, on November 9, 2016. The two colors distinguish between the opposite classes of each semantic dimension. 90
3.15 Number of tweets concerning Republicans, separately for each class of the four descriptor dimensions, on November 9, 2016. The two colors distinguish between the opposite classes of each semantic dimension. 91
3.16 Number of tweets concerning Democrats, separately for each class of the four descriptor dimensions, on November 4, 2020. The two colors distinguish between the opposite classes of each semantic dimension. 91
3.17 Number of tweets concerning Republicans, separately for each class of the four descriptor dimensions, on November 4, 2020. The two colors distinguish between the opposite classes of each semantic dimension. 91

4.1 The proposed teacher-student training architecture. 97

List of Tables

3.1 Achieved accuracy of each of the four opinion classifiers on the test set of the respective training dataset. 75
3.2 Hyperparameters used for training the sentiment classifiers. 75
3.3 Hyperparameters used for training the forecasting model. 79
3.4 Forecasting results on the US 2016 Presidential Election Tweets dataset for the six constructed timeseries. "Dem" denotes the Democrats, "Rep" denotes the Republicans, while "mean", "med" and "trim" imply the three respective aggregation strategies: mean, median and trimmed mean. In each case, the SMAPE/-MASE metrics have been independently averaged across the four descriptor channels using both the mean and the median operator. A lower value is better for both metrics, while SMAPE is a percentage. 79
3.5 Forecasting results on the US 2020 Presidential Election Tweets dataset for the six constructed timeseries. "Dem" denotes the Democrats, "Rep" denotes the Republicans, while "mean", "med" and "trim" imply the three respective aggregation strategies: mean, median and trimmed mean. In each case, the SMAPE/-MASE metrics have been independently averaged across the four descriptor channels using both the mean and the median operator. A lower value is better for both metrics, while SMAPE is a percentage. 80

4.1 Evaluation results on the S15-T11 dataset. Higher/lower is better for the COS/MSE metric, respectively. Best results are in bold. 99

Dedicated to my grandmother, Maria, who hosted and supported me during this project.

Chapter 1

Natural Language Processing

In this first chapter we are going to explore the wide field of Natural Language Processing (NLP). We are going to define the term and track its historic timeline. Furthermore, we discuss about the implementation part, different methods and algorithms used in NLP and how the language is modeled as processable data. Finally, we briefly present some common NLP tasks.

1.1 What is NLP ?

1.1.1 Short definitions

Many definitions are found across the web about NLP. They are obviously all similar as they describe the same thing, yet I chose the following as more simplistic and accurate.

Definition #1 : Natural Language Processing (NLP) can be defined as the automatic manipulation of natural language, like speech and text, by software [1].

or,

Definition #2 : Natural Language Processing (NLP) consists of automatic methods that take natural language as input or produce natural language as output [1].

Some people may think that NLP is a completely new field of study but the reality is that it has been around for more than 50 years. In fact, NLP grew out of the field of linguistics with the rise of computers and has made enormous progress in the last decade due to astonishing discoveries made by researchers.

1.1.2 Natural Language

Natural language can simply be defined as the way we, humans, communicate with each other. The word natural indicates that this language has developed naturally in use as contrasted with an artificial language or computer code. As every language, natural language contains a broad set of words and rules which tell us how to combine them in order to communicate the desired message. These two sets differ from country to country giving us all the different languages in the world (English, French, German etc.). Natural Language can be used in two ways, verbally – speech and written down – text. Both are used by people to express themselves, each one for its best suited case. Consequently, given the importance of this type of data, it becomes an obligation to find methods to understand and reason about natural language.

1.1.3 Challenge

Natural language is often hard and ambiguous because it is messy. There are few rules. It is also ever changing and evolving. People are great at producing language and understanding language, and are capable of expressing, perceiving, and interpreting very elaborate and nuanced meanings. At the same time, while we humans are great users of language, we are also very poor at formally understanding and describing the rules that govern language. So it's hard working with such data.

1.1.4 From Linguistics to NLP

1.1.4.1 Linguistics

Linguistics is the scientific study of language and its structure [2]. The major levels of linguistic structure can be seen in figure 1.1.

- *Pragmatics* is the study of how context contributes to meaning.
- *Semantics* is the study of the meaning of words and sentences.
- *Syntax* is the study of sentence structure, and of how sentence structure interacts with other dimensions of linguistic information.
- *Morphology* is the study of words, how they are formed, and their relationship to other words in the same language.
- *Phonology* is the study of the patterns of sounds in a language and across languages.

- *Phonetics* is the study of speech sounds and their physiological production and acoustic qualities.

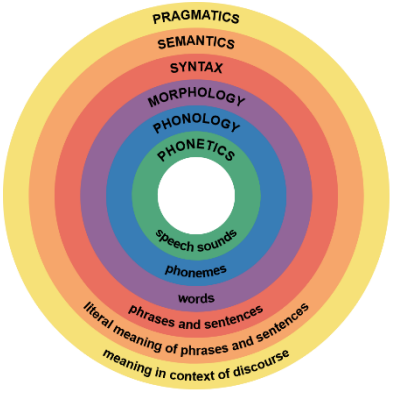

FIGURE 1.1: The major levels of linguistic structure.

1.1.4.2 Computational Linguistics

Computational linguistics (CL) is the modern study of linguistics using the tools of computer science. Computers help us to handle large text data efficiently and therefore open new areas of research that were impossible to explore before.

1.1.4.3 CL vs NLP

Computational linguistics has both a scientific and an engineering side.

- Engineering side - Natural language processing (NLP): Building computational tools that do useful things with language.
- Scientific side - Natural language understanding (NLU): Seeks to study/understand language using computers and corpora.

Both sides employ same means but the goal is different. NLP researchers will build a useful system and show that it works really well. CL researcher would be more interested in which language features are useful indicators of some meaning and why.

1.2 History

1.2.1 Roots

Natural language processing has its roots in the 1950s. Already in 1950, Alan Turing published an article titled ”Computing Machinery and Intelligence” [3] which proposed what is now called the Turing test as a criterion of intelligence. In other words, it would test if a machine exhibits human-like intelligence. The proposed test includes a task that involves the automated interpretation and generation of natural language.

1.2.2 Three periods

- Symbolic NLP (1950s - early 1990s)
- Statistical NLP (1990s - 2010s)
- Neural NLP (2010s - present)

1.2.2.1 Symbolic NLP (classical programming)

Given a collection of hand written rules the computer emulates natural language understanding by applying those rules to the data it confronts [4]. A set of such rules is called algorithm in programming and can be visualized with a flowchart (figure 1.2).

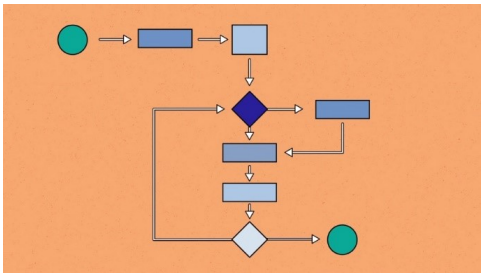

FIGURE 1.2: Flowchart - A diagrammatic representation of an algorithm.

Milestones

- ***1950s*** : The Georgetown experiment in 1954 [5] involved fully automatic translation of more than sixty Russian sentences into English. The authors claimed that within three or five years, machine translation would be a solved problem. However, real progress was much slower.

- ***1960s*** : A notably successful natural language processing system was ELIZA [6], a simulation of a Rogerian psychotherapist, written by Joseph Weizenbaum between 1964 and 1966. Using almost no information about human thought or emotion, ELIZA sometimes provided a startlingly human-like interaction. When the ”patient” exceeded the very small knowledge base, ELIZA might provide a generic response, for example, responding to ”My head hurts” with ”Why do you say your head hurts?”

- ***1970s*** : During this time, the first chatbots were written (e.g., PARRY [7]). A chatbot is a software application used to conduct an on-line chat conversation via text or text-to-speech, in lieu of providing direct contact with a live human agent.

- ***1980s*** : The 1980s and early 1990s mark the hey-day of symbolic methods in NLP. Focus areas of the time included research on rule-based parsing (e.g., the development of HPSG [8] as a computational operationalization of generative grammar) and semantics (e.g., Lesk algorithm [9]).

1.2.2.2 Statistical NLP – Machine Learning

Up to the 1980s, most natural language processing systems were based on complex sets of hand-written rules. Starting in the late 1980s, however, there was a statistical turn in natural language processing with the introduction of machine learning algorithms for language processing [4].

Machine learning

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy [10]. The difference between machine learning and conventional hand-written algorithms is depicted in figure 1.3.

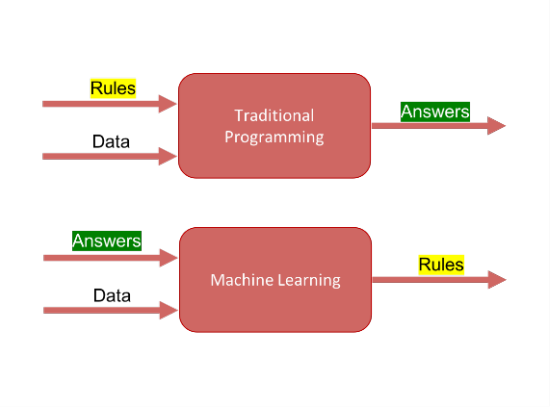

FIGURE 1.3: Machine learning approach vs Rule based approach

Machine learning algorithms are categorized based on the kind of data they use for training. There are two kinds of data, labeled and unlabeled data. Labeled data has both the input and output parameters in a completely machine-readable pattern, but requires a lot of human labor to label the data with the desired output. Unlabeled data only has the input parameters. This negates the need for human labor but requires more complex solutions. The deriving machine learning categories are supervised and unsupervised learning respectively [11].

Supervised learning

In supervised learning, the algorithm finds relationships between the input and output parameters given, essentially establishing a cause and effect relationship between the variables in the dataset. At the end of the training, the algorithm has an idea of how the data works and the relationship between the input and the output [12]. Supervised learning problems can be further grouped into regression and classification problems [13].

- ***Classification*** : A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”.
- ***Regression*** : A regression problem is when the output variable is a real value, such as “dollars” or “weight”.

Unsupervised learning

Unsupervised learning does not have labels to work off of, resulting in the creation of hidden structures. Relationships between data points are perceived by the algorithm in an abstract manner, with no input (supervision) required from human beings. The creation of these hidden structures is what makes unsupervised learning algorithms versatile. Instead of a defined and set problem statement, unsupervised learning algorithms can adapt to the data by dynamically changing hidden structures [14]. Generally, this task is much more difficult than supervised learning, and typically produces less accurate results for a given amount of input data. However, the enormous amount of non-annotated data available can often make up for the inferior results, if the algorithm used has a low enough time complexity to be practical. Unsupervised learning problems can be further grouped into clustering and association problems [13].

- ***Clustering*** : Grouping of similar data together is called as Clustering. This is obtained by calculating the distance between the points.
- ***Association*** : An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Why Statistical NLP

The main reasons that led to this 'revolution' in NLP were the following [4] :

- Increase in computational power (Moore's law).
- Gradual lessening of the dominance of Chomskyan theories of linguistics which discouraged the machine-learning approach.
- Better results, speed, and robustness.
- Increased amount of data (web).
- Disenchantment with symbolic top-down approaches.

Milestones

- ***1990s*** : Many of the notable early successes on statistical methods in NLP occurred in the field of machine translation, due especially to work at IBM Research. These systems were able to take advantage of existing multilingual textual corpora. A great deal of research has gone into methods of more effectively learning from limited amounts of data.
- ***2000s*** : With the growth of the web, increasing amounts of raw (unannotated) language data has become available since the mid-1990s. Research has thus increasingly focused on unsupervised and semi-supervised (combination of annotated and non-annotated data) learning algorithms.

1.2.2.3 Neural NLP

Neural NLP refers to using deep learning (neural networks) for processing natural language. Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

Artificial Neural Network (ANN)

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives signals then processes them and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of

its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times [15].

Deep learning sets its roots back in 1962 when Frank Rosenblatt described all of the basic ingredients of the deep learning systems in his book "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" [16]. The first general, working learning algorithm for supervised, deep, feedforward, multilayer perceptrons was published by Alexey Ivakhnenko and Lapa in 1967 [17]. However, neural networks started being used efficiently for NLP tasks in 2000s and exploded in 2010s with remarkable breakthroughs [18].

Milestones

- ***2001 - Neural language models*** : The first neural language model, a feed-forward neural network was proposed by Bengio et al. [19]. Language modelling is the task of predicting the next word in a text given the previous words. Such models are used to learn good vector representations for words – word embeddings (defined in 1.4.2).
- ***2008 - Multi-task learning*** : Multi-task learning was first applied to neural networks for NLP by Collobert and Weston [20]. In their model, the look-up tables (word embedding matrices) are shared between two models trained on different tasks. Multi-task learning is a general method for sharing parameters between models that are trained on multiple tasks. Sharing the word embeddings enables the models to collaborate and share general low-level information in the word embedding matrix, which typically makes up the largest number of parameters in a model. This paper proved influential beyond its use of multi-task learning. It spearheaded ideas such as pre-training word embeddings and using convolutional neural networks (CNNs) for text that have only been widely adopted for NLP in the last years.
- ***2013 - Word embeddings*** : Word embeddings have been used as early as 2001 as we have seen above. The main innovation that was proposed in 2013 by Mikolov et al. [21] was to make the training of these word embeddings more efficient by removing the hidden layer from Bengio [19] model and approximating the objective. These changes together with the efficient word2vec (see 1.4.2.1) implementation enabled large-scale training of word embeddings. While these embeddings are no different conceptually than the ones learned with a feed-forward neural network, training on a very large corpus enables

them to approximate certain relations between words such as gender, verb tense, and country-capital relations. What cemented word embeddings as a mainstay in current NLP was that using pre-trained embeddings as initialization was shown to improve performance across a wide range of downstream tasks.

- ***2013 - Neural networks for NLP*** : 2013 and 2014 marked the time when neural network models started to get adopted in NLP. Two main types of neural networks became the most widely used: recurrent neural networks (see 1.3.3.2) and convolutional neural networks (see 1.3.3.2). Before 2013, RNNs were still thought to be difficult to train; Ilya Sutskever's PhD thesis [22] was a key example on the way to changing this reputation. With convolutional neural networks being widely used in computer vision, they also started to get applied to language (Kalchbrenner et al., 2014 [23]; Kim et al., 2014 [24]).

- ***2014 - Sequence-to-sequence models*** : Sutskever et al. [25] proposed sequence-to-sequence learning, a general framework for mapping one sequence to another one using a neural network. In the framework, an encoder neural network processes a sentence symbol by symbol and compresses it into a vector representation; a decoder neural network then predicts the output symbol by symbol based on the encoder state, taking as input at every step the previously predicted symbol. Machine translation (MT) turned out to be the killer application of this framework. Encoders for sequences and decoders are typically based on RNNs.

- ***2015 - Attention*** : Attention (Bahdanau et al.[26]) is one of the core innovations in neural MT (NMT) and the key idea that enabled NMT models to outperform classic phrase-based MT systems. The main bottleneck of sequence-to-sequence learning is that it requires to compress the entire content of the source sequence into a fixed-size vector. Attention alleviates this by allowing the decoder to look back at the source sequence hidden states, which are then provided as a weighted average as additional input to the decoder. Multiple layers of self-attention are at the core of the Transformer (see 1.4.2.2) architecture (Vaswani et al., 2017 [27]), the current state-of-the-art model for NMT.

- ***2015 - Memory-based networks*** : Attention can be seen as a form of fuzzy memory where the memory consists of the past hidden states of the model, with the model choosing what to retrieve from memory. Memory Networks proposed by Weston et al., 2015 [28] are models with a more explicit memory. Memory is often accessed based on similarity to the current state similar to attention and can typically be written to and read from. Memory-based models are typically applied to tasks, where retaining information over longer time spans should be useful such as language modelling and reading comprehension.

- ***2018 – Pre-trained Language Models (PLMs)*** : PLMs are large neural networks that are used in a wide variety of NLP tasks. These models are first pre-trained over a large

text corpus, usually in a word prediction task, in order to learn good contextualized word embeddings (see 1.4.2.2). Then, they are loaded and maybe fine-tuned to perform a downstream task. Pre-trained language models were first proposed by Dai & Le, 2015 [29] only recently were they shown to be beneficial across a diverse range of tasks. The most popular pre-trained language models are CoVe (B. McCann et al, 2017 [30]), ELMo (M. Peters et al, 2018 [31]), GPT (OpenAI, 2018 [32]) and BERT (J. Devlin et al, 2018 [33]) all analyzed in 1.4.2.2. Using pre-trained language models embeddings can improve the overall performance as they encapsulate knowledge from huge corpora. Furthermore, we obviously save time and need less data for training on downstream tasks.

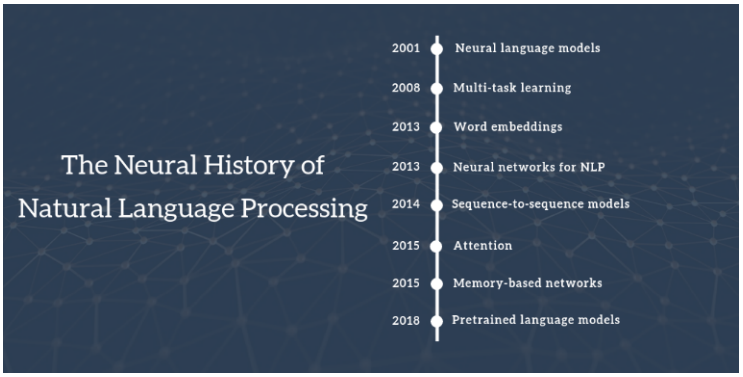

Figure 1.4: Neural NLP history.

1.3 Methods: Rules, statistics, neural networks

1.3.1 Rules

Here lie the Symbolic NLP methods. These methods involve the hand-coding (by the programmer) of a set of rules that handle the input data in a strictly deterministic way. For example, the IF-THEN logic structure is followed. When an IF linguistic condition is met, the THEN output is generated. This makes it easy to establish clear and explainable rules, providing full transparency into how it works. Also, we usually find a dictionary lookup in such algorithms, such as by writing grammars or devising heuristic rules for stemming.

1.3.2 Statistics

The statistical revolution in the 1990s made machine learning algorithms dominate the NLP field. In these algorithms the rules are not hand-written as in Symbolic NLP but they are automatically learned through the analysis of large corpora. A large set of ”features” that are generated from the input data by the programmer (manually), are fed as input to the machine

learning algorithm. The algorithm then uses statistical inference to detect the patterns in the input features in order to produce the desired output.

1.3.2.1 Why ML ?

Cons of hand-crafted rules

- Not at all obvious where the effort should be directed
- Handling unfamiliar and erroneous input is extremely difficult
- Systems can only be made more accurate by increasing the complexity of the rules -> hard process

1.3.2.2 ML Algorithms for NLP

Supervised ML

- ***Support Vector Machines (SVMs)*** : Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. SVM maps training examples to points in space so as to maximize the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall [34].
- ***Naïve Bayes*** : Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features in order to minimize the probability of misclassification [35].
- ***Maximum Entropy*** : The principle of maximum entropy is a model creation rule that requires selecting the most unpredictable (maximum entropy) prior assumption if only a single parameter is known about a probability distribution. The goal is to maximize "uniformitiveness," or uncertainty when making a prior probability assumption so that subjective bias is minimized in the model's results [36].
- ***Decision Trees*** : The goal of using a Decision Tree is to create a training model that can be used to predict the class (classification tree) or value (regression tree) of the target variable by learning simple decision rules inferred from prior data (training data). In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the

basis of comparison, we follow the branch corresponding to that value and jump to the next node. In a decision tree, leaves represent the final output (class or real value) and branches represent conjunctions of features that lead to those output values [37].

- ***Random Forests*** : Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting to their training set. Random forests generally outperform decision trees [38].

- ***K-nearest Neighbors (k-NN)*** : k-NN is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in a data set. The output depends on whether k-NN is used for classification or regression [39] :

  - In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
  - In *k-NN regression*, the output is the property value for the object. This value is the average of the values of k nearest neighbors.

Unsupervised ML

- ***Hierarchical clustering*** : Hierarchical clustering is a method that seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

  - *Agglomerative* : This is a "bottom-up" approach: each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
  - *Divisive* : This is a "top-down" approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

  In general, the merges and splits are determined in a greedy manner. The results of hierarchical clustering are usually presented in a dendrogram [40].

- ***K-means clustering*** : k-means clustering is a method of vector quantization, originally from signal processing, that aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data

space into Voronoi cells. k-means clustering minimizes within-cluster variances (squared Euclidean distances) [41].

- ***Matrix Factorization*** : Matrix Factorization is another technique for unsupervised NLP machine learning. It uses "latent factors" to break a large matrix down into the combination of two smaller matrices. Latent factors are similarities between the items. Think about the sentence, "I threw the ball over the mountain." The word "threw" is more likely to be associated with "ball" than with "mountain" [42].

1.3.3 Neural Networks

Since the neural turn (early 2010s), statistical methods in NLP research have been largely replaced by neural networks (NNs) as they proved to perform better in many tasks. NNs require more data but less linguistic expertise to train and operate. It is therefore obvious why they become so popular since the early 2010s when the continuously expanding world wide web solved the data availability problem.

1.3.3.1 Why NNs

Cons of statistical methods:

- Elaborate feature engineering
- Relying on a pipeline of separate intermediate tasks for learning a higher-level task

NNs Feature Learning

Deep learning methods have the ability to learn feature representations whereas machine learning methods require experts to manually specify and extract features from natural language [43].

Cons of manually designed features

- Overspecified or incomplete
- Long time to design and validate
- Only get you to a certain level of performance

Pros of learned features

- Continually and automatically improve
- Easy to adapt
- Fast to train

NNs End-to-End

With deep learning we have the ability to develop one neural network to learn the whole problem end-to-end rather than pipelines of specialized systems. This is desirable both for the speed and simplicity of development in addition to the improved performance of these models. For example, the large blocks of an automatic speech recognition pipeline are speech processing, acoustic models, pronunciation models, and language models. However the properties and importantly the errors of each sub-system are different. This motivates the need to develop one neural network to learn the whole problem end-to-end [43].

1.3.3.2 Types of NNs for NLP

There are five deep learning methods that deserve the most attention for application in natural language processing.

- Embedding Layers
- Multilayer Perceptrons (MLP)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)
- Hybrid – Combinational Neural Networks

Embedding Layer

- Its job is to map the one-hot encoded words (see 1.4.1.1) to their word embeddings (see 1.4.2).
- Used on the front end of a neural network, like in figure 1.5.
- It can be used alone to learn a word embedding that can be saved and used in another model later.
- It can be used as part of a deep learning model where the embedding is learned along with the model itself trained on a specific NLP task.

- It can be used to load a pre-trained word embedding model, a type of transfer learning.

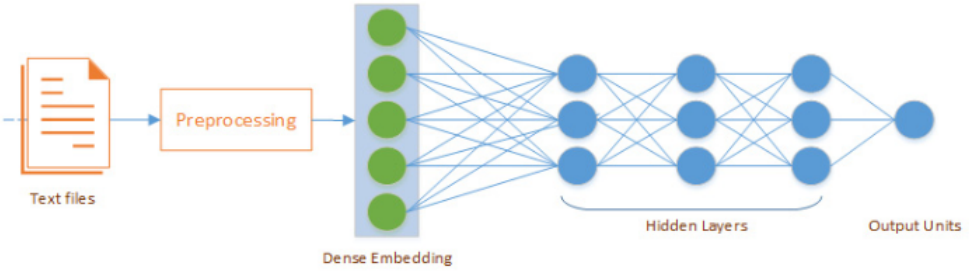

FIGURE 1.5: Example use of embedding layer in a neural network.

The word embeddings are actually the weights of the embedding layer and that is why we say the embeddings are learned. They are optimized during the training of this layer in a specific NLP task. Since we are talking about language modeling the task is usually related with word prediction in a sequence. However, in order to get better results in a text classification task, embeddings can be trained along with the entire model on this task. Learning a word embedding can be very time consuming as it requires lots of data in order to learn good, meaningful representations for all words [44]. When implementing the embedding layer (keras library) three basic arguments are needed:

- input_dim: Size of the input vocabulary – number of unique words in the input corpus
- output_dim: Size of each word embedding vector (e.g. 200d vector)
- input_length: Length of input sequences (e.g. Tweet – 15 words)

Multilayer Perceptron (MLP)

An MLP is a fully connected class of feedforward artificial neural network (ANN). In fully connected networks each neuron in one layer is connected to all neurons in the next layer, passing the input from one layer to another with weight multiplications. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training the networks weights. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. That is why it can distinguish data that is not linearly separable [45]. In figure 1.6 an MLP with one input, one output and five hidden layers is depicted. After we have obtained the word embeddings from a language model or from an embedding layer we can feed them to an MLP to perform a specific NLP task, let's say sentiment analysis (see 2.1.1).

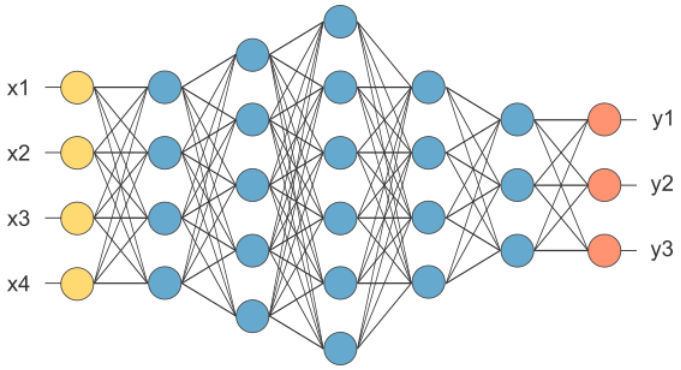

FIGURE 1.6: Multilayer Perceptron with five hidden layers.

Convolutional Neural Networks (CNNs)

CNN is a class of artificial neural network, most commonly applied to analyze visual imagery due to their ability to identify spatially local input patterns. In a CNN, the hidden layers include layers that perform convolutions – convolutional layers. This layer performs a dot product of the convolution kernel/filter with the layer's input matrix. As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. CNNs are also known as shift invariant, based on the shared-weight architecture of the convolution kernels/filters that provide translation-invariant responses. This means that the resulting feature map is invariant under shifts of the locations of input features in the visual field. In other words, we can learn the same feature extractor (the convolution) for every location in an image, which is a massive savings in parameters over a fully connected network (not invariant to shifts of the input), which would have to relearn the same feature extractor for each location [46].

A convolutional layer is usually followed by a max-pooling layer. Pooling operation reduces the sizes of feature maps (convolutional layer output), by selecting a single value from a pool (rectangular area) of values that slides like a convolutional filter across the input. If the value selected is the maximum one then we call it max-pooling. The learned "filters" produce the strongest response to a spatially local input pattern and that is why max-pooling is the most popular choice as it helps isolate these local input patterns. Pooling layers provide translation-invariant responses too. However, convolution or pooling layers within a CNN that do not have a stride greater than one are equivariant, as opposed to invariant, to translations of the input. Recently CNNs were also adopted in NLP. Specifically, they are used in text classification tasks where they take advantage of all their useful properties explained above. The difference is that instead of sliding in the two-dimensional space (image), now the convolutional kernels slide in the one-dimensional time (words in a sequence). Figure 1.7 shows an application example of CNN on movie reviews classification.

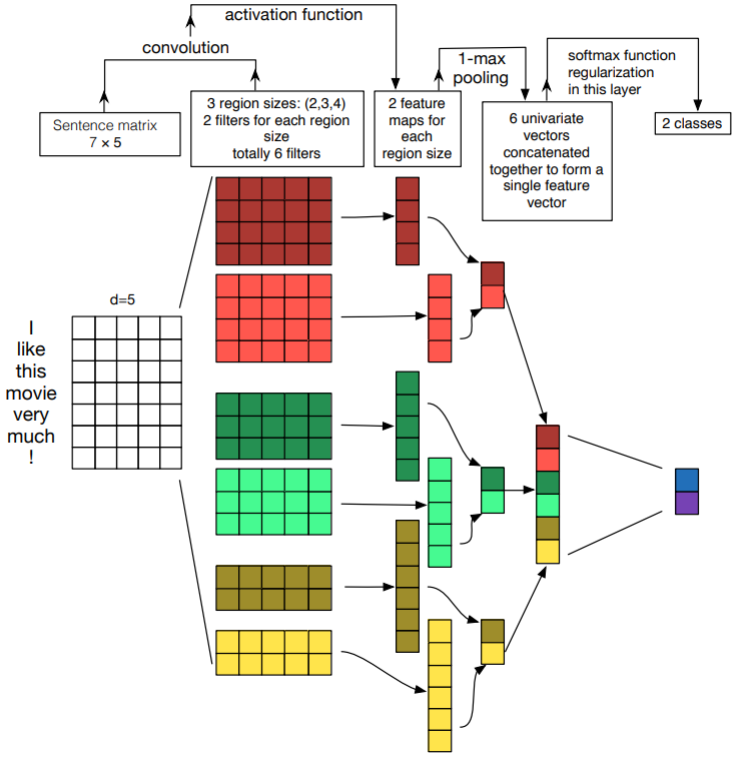

FIGURE 1.7: CNN model used for classifying movie reviews.

Recurrent Neural Networks (RNNs)

RNNs are a type of artificial neural network that were based on David Rumelhart's work in 1986 [47]. They are networks with loops in them, allowing information to persist.

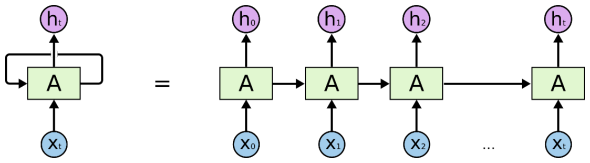

FIGURE 1.8: The unfolded representation of the RNN's repeating module.

Figure 1.8 shows the basic building block/repeating module of an RNN, the RNN unit. A chunk of neural network, A, looks at some input xt and outputs a value ht. A loop allows information to be passed from one step of the network to the next. RNN is actually multiple copies of the same network, each passing a message to a successor. This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data. In the last few years, there have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning. One of the appeals of RNNs is that they are able to connect previous information to the present task. In cases where the gap between the relevant

information and the place that it's needed is small, RNNs can learn to use the past information (short-term memory). Unfortunately, as that gap grows (long-term memory), RNNs become unable to learn to connect the information [48].

Long Short Term Memory networks

LSTMs are a special kind of RNN introduced by Hochreiter & Schmidhuber in 1997 [49]. LSTMs are explicitly designed to avoid the long-term dependency problem. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way shown in figure 1.9.

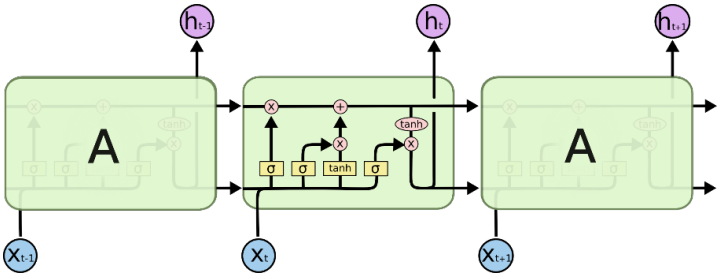

FIGURE 1.9: The structure of an LSTM unit

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. It runs straight down the entire chain, letting information flow along it. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. An LSTM has three of these gates, to protect and control the cell state [50].

In theory, classic RNNs can keep track of arbitrary long-term dependencies in the input sequences. The problem with vanilla RNNs is computational (or practical) in nature: when training a vanilla RNN using back-propagation, the long-term gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the process, which use finite-precision numbers. RNNs using LSTM units partially solve the vanishing gradient problem, because LSTM units allow gradients to also flow unchanged. However, LSTM networks can still suffer from the exploding gradient problem. Figure 1.10 depicts an example application of LSTM on movie review classification/sentiment analysis. We will see more about sentiment analysis in chapter 2.

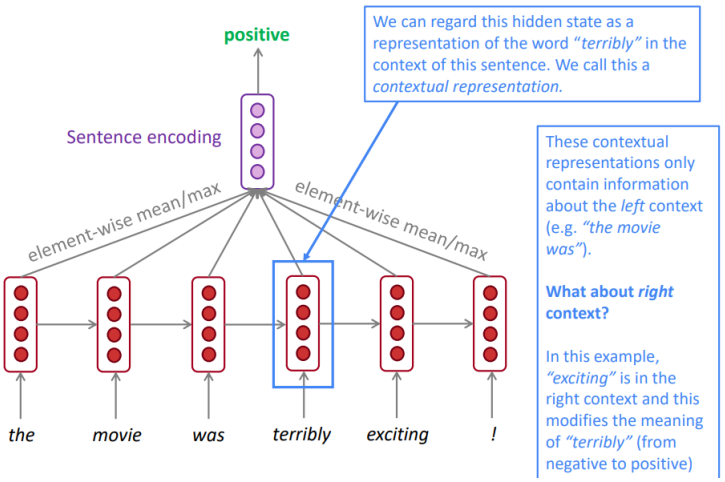

FIGURE 1.10: LSTM used for classifying movie reviews.

RNN vs CNN

RNN

- Recognize patterns across time
- Comprehension of global/long-range semantics
- Account for the order of words in text
- All sequential data tasks: Question-Answering, Machine Translation, POS tagging, Image Captioning, Language Modeling etc.
- Slow training – can't use parallelism

CNN

- Recognize patterns across space
- Local feature detection
- Don't account for the order of words (convolution + pooling)
- Only Classification tasks: Sentiment Analysis, Spam Detection, Topic Categorization
- Fast training (5x) – GPU utilization for convolutions

Hybrid NNs

Hybrid or ensemble neural network is a network with a combination of different artificial neural networks and approaches. It utilizes features of varying neural networks and approaches to achieve optimum results. The most popular combination found in literature is that of a CNN and an LSTM. Such models were originally developed for visual time series prediction problems and the application of generating textual descriptions of single image or sequence of images (videos). However, this architecture has also been used on speech recognition and natural language processing problems where CNNs are used as feature extractors for the LSTMs on audio and textual input data [51]. CNN LSTMs are appropriate for problems that:

- Have spatial structure in their input such as the 2D structure or pixels in an image or the 1D structure of words in a sentence, paragraph, or document.
- Have a temporal structure in their input such as the order of images in a video or words in text, or require the generation of output with temporal structure such as words in a textual description.

They can be combined in three different ways as shown below.

CNN + LSTM

A sequential architecture where the input word embeddings are first fed to the CNN part and then the extracted features pass through the LSTM component. A final output layer is usually employed after the LSTM layer. An example of this architecture is depicted in figure 1.11.

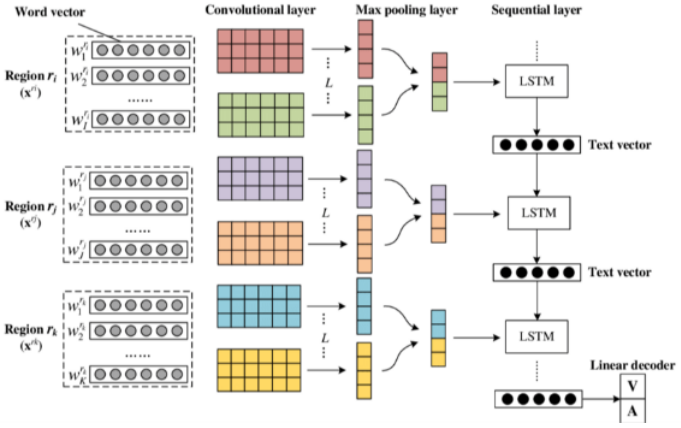

Figure 1.11: CNN + LSTM example architecture.

LSTM + CNN

A sequential architecture where the LSTM layer is placed before the CNN part. In this case, the CNN extracts features from the LSTM outputs and feeds them to the output layer. This type of model is not commonly used as it is less efficient in training compared to the CNN + LSTM architecture. That's because the CNN is used to reduce the dimensionality of the input by producing the feature maps. Therefore, the LSTM will have to process smaller inputs compared to the LSTM + CNN where the LSTM processes the full input representations making it slower to train. An example of this architecture is depicted in figure 1.12.

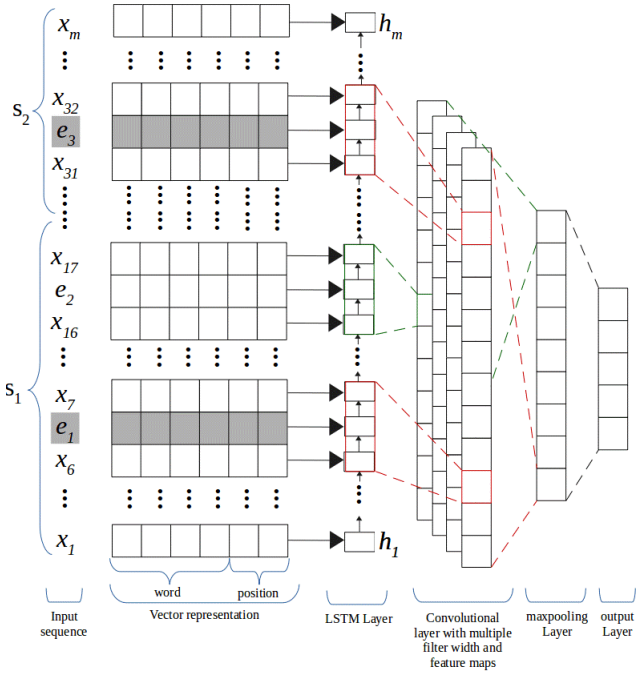

FIGURE 1.12: LSTM + CNN example architecture.

CNN // LSTM

A parallel combination of these two types of neural networks. The input embeddings are fed both to the CNN and LSTM components and their respective outputs are usually concatenated. The concatenated features are then fed to the following (output) layers. An example of this architecture is depicted in figure 1.13.

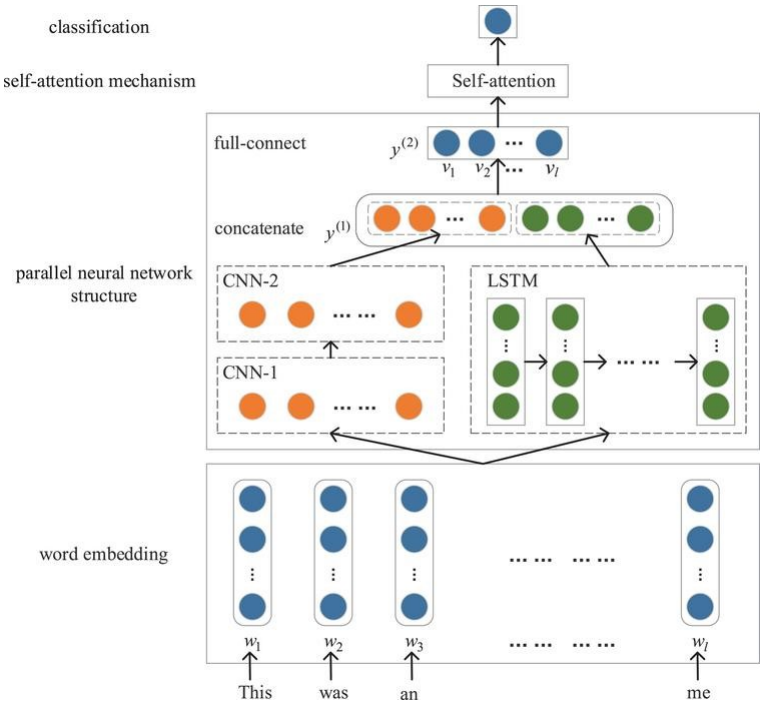

FIGURE 1.13: CNN // LSTM example architecture.

1.4 Word representations

In order to make text data processable by any kind of algorithm we need to convert it into numbers. These numerical representations are in vector form and most commonly represent words. There are two categories of word representations/vectors.

- ***Fixed representations (sparse)*** : Every dimension/value of the feature vector corresponds to a specific word from the input vocabulary.
- ***Distributed representations (dense)*** : Features/dimensions of the vector do not correspond to words from the vocabulary but to some meaning/entity.

1.4.1 Fixed representations

This type of representations are also called sparse because they are usually very large and contain little information about the semantics of words. Three are the main examples of fixed representations, listed below.

- One-Hot Encoding
- Bag of words (count vectors)
- TF-IDF

1.4.1.1 One-hot Encoding

A vocabulary that contains all the unique words found in the input corpus is formed. One-hot encoding produces a vector representation for each word where all values are zero except the one in word's corresponding position in the vocabulary. Therefore, the word vectors have same size as the vocabulary [52]. An example is shown in figure 1.14.

Word	ID
Red	1
Green	2
Blue	3
Yellow	4
Orange	5
Purple	6

→

Green	0	1	0	0	0	0
Yellow	0	0	0	1	0	0
Orange	0	0	0	0	1	0

FIGURE 1.14: One-hot encoding example.

1.4.1.2 Bag of words (BOW)

Bow is a method used to represent phrases, sentences, paragraphs, documents, not single words. The produced vectors have the size of the vocabulary but in contrast with one-hot encoding more than one columns are non-zero since it represents more than one word [53]. When the non-zero values are binary indicating just the existence of words in the represented context, we call it binary BOW. In classical BOW, the word counts (appearance frequency) are used. In figure 1.15 we see the respective examples for each case.

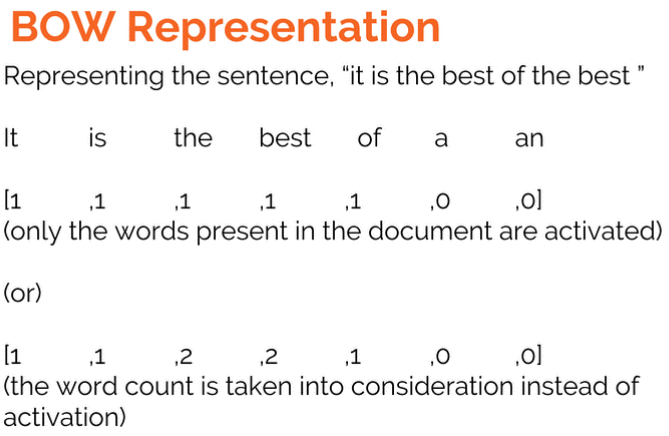

FIGURE 1.15: Bag of words encoding example.

1.4.1.3 Term Frequency-Inverse Document Frequency (TF-IDF)

This method calculates a statistical measure to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Typically, the TF-IDF weight is composed by two terms [54] :

- ***Term Frequency (TF)***, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- ***Inverse Document Frequency (IDF)***, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as ”is”, ”of”, and ”that”, may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

  IDF(t) = log_e(Total number of documents in corpus / Number of documents with term t in it).

Figure 1.16 shows an example where sentences A, B are represented using TF-IDF method. Word column is the vocabulary containing all unique words in corpus.

Word	TF		IDF	TF*IDF	
	A	B		A	B
The	1/7	1/7	log(2/2) = 0	0	0
Car	1/7	0	log(2/1) = 0.3	0.043	0
Truck	0	1/7	log(2/1) = 0.3	0	0.043
Is	1/7	1/7	log(2/2) = 0	0	0
Driven	1/7	1/7	log(2/2) = 0	0	0
On	1/7	1/7	log(2/2) = 0	0	0
The	1/7	1/7	log(2/2) = 0	0	0
Road	1/7	0	log(2/1) = 0.3	0.043	0
Highway	0	1/7	log(2/1) = 0.3	0	0.043

FIGURE 1.16: TF-IDF encoding example.

1.4.1.4 Fixed representations - Pros & Cons

Cons

- Require large memory
- No semantic information

Pros

- Easy to use

1.4.2 Distributed representations (embeddings)

In contrast with fixed representations that are sparse, since they have the size of the vocabulary, distributed representations are dense. This means that they consist of a pre-defined number of entities/dimensions each one capturing a specific semantic property. Therefore, a word is represented as a distribution to these entities. The most important thing is that words with similar meanings have similar representations/vectors as the words' semantics are embedded in the vector (word embedding). Such representations are learned like the weights of a NN in training. The models used to train/learn meaningful word embeddings are called language models. They are trained with large corpora, in a word prediction task. There are two categories of word embeddings listed below.

- *Classic word embeddings*
  - Word2Vec (T. Mikolov et al, 2013 [21])
  - GloVe (J. Pennington et al, 2014 [55])
  - FastText (P. Bojanowski et al, 2016 [56])
- *Contextualized word embeddings*
  - CoVe (B. McCann et al, 2017 [30])
  - ELMo (M. Peters et al, 2018 [31])
  - GPT (OpenAI, 2018 [32])
  - BERT (J. Devlin et al, 2018 [33])

1.4.2.1 Classic word embeddings

With classic word embeddings each word has a standard real-valued vector representation that is learned by a language model.

Word2Vec

Word2Vec is a two-layer neural network proposed by T. Mikolov in 2013 [21]. This paper marked the beginning of a new era in NLP as the proposed embeddings enabled the efficient training of neural networks that were unable to perform well with the previous sparse representations. Word2Vec was trained to reconstruct linguistic contexts of words. Specifically, training involved pairs of context-target words giving us two variations of the method (figure 1.17).

- *Skip-gram* : The embeddings are learned while the model tries to predict the context given the target word as input
- *CBOW* : The context words embeddings are randomly initialized and learned while the model tries to predict the target word given the context as input

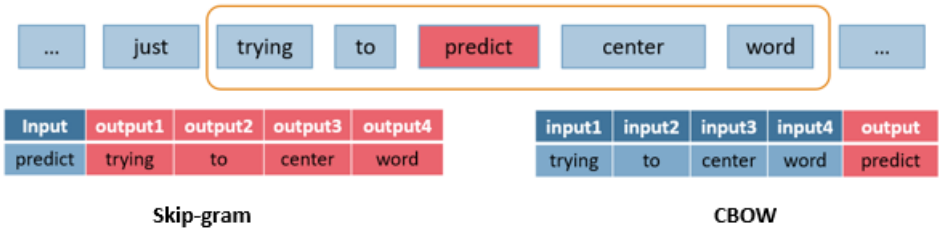

Figure 1.17: Skip-gram vs CBOW.

Skip-gram training

In this section we are going to explain the skip-gram version (most popular) of Word2Vec and see how it is trained [57]. In Skip-gram version, a target word is given as input and the model tries to predict its surrounding words. By nature, this is a pretty hard task and was made a lot easier with the following idea. Instead of asking the model to predict the neighboring word, ask it if a word is neighbor. This makes processing much faster, allows training on huge corpora and consequently better embeddings are learnt. Figure 1.18 depicts the training task.

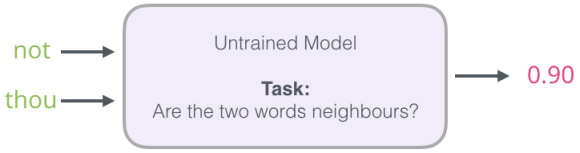

Figure 1.18: Concept of Skip-gram training task.

Negative sampling

To balance the training dataset, negative sampling is employed. For each sample in the initial dataset, random words from the vocabulary – not neighbors – are put as negative examples. In this way we avoid getting a 'lazy' model that gives poor embeddings. An example of negative sampling is shown in figure 1.19.

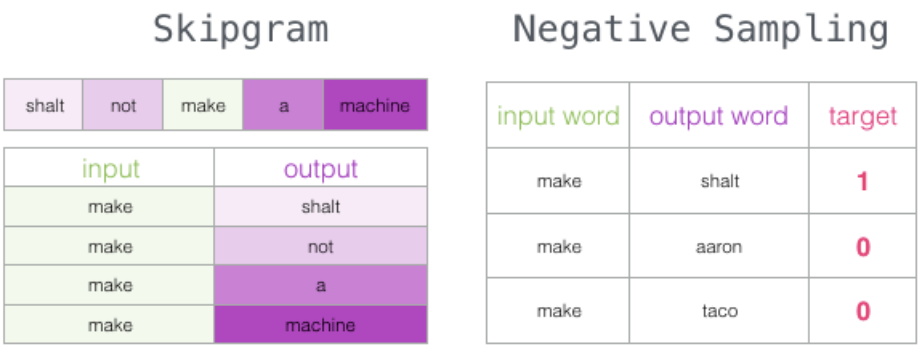

FIGURE 1.19: Negative Sampling example

Training process

First, embedding (green) and context (pink) matrices are created (both vocabulary_size x embedding_size) as shown in figure 1.20. These two have an embedding for each word and are getting updated during training. The goal of training is to get a good embedding matrix with all the word representations. We can then load that pre-trained matrix to deal with any NLP task.

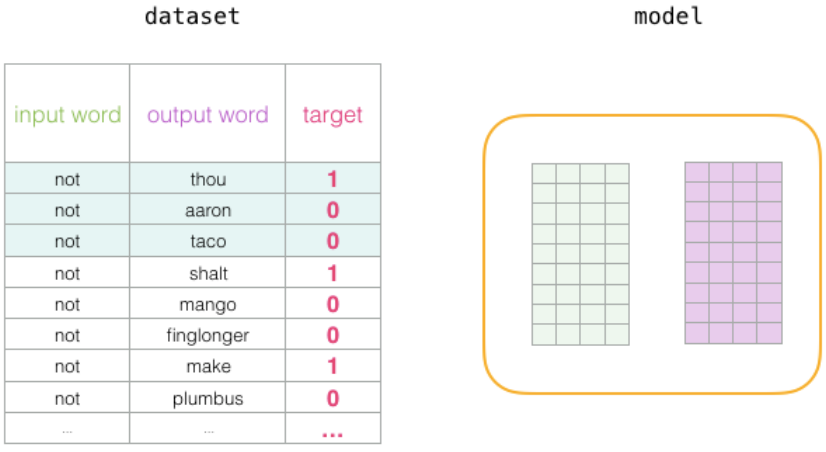

FIGURE 1.20: The training dataset of Skip-gram model.

The training process goes like this:

1. Take one positive example and its associated negative examples (figure 1.20).

2. Look up embeddings: Input words -> Embedding matrix, Output/context words -> Context matrix (figure 1.21).

3. Take the dot product of the corresponding embeddings - that number indicates the similarity of the input and context embeddings (figure 1.22).

4. Use sigmoid function (suitable for binary classification) to get the model's output.

5. Calculate prediction error and use it to update model parameters (the two matrices).

6. Iterate through the whole dataset to get a well-trained embedding matrix.

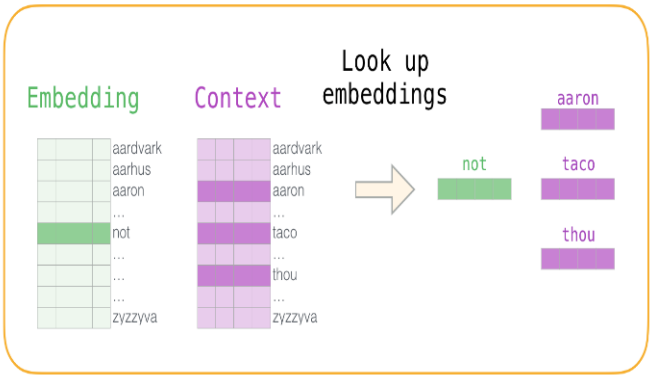

FIGURE 1.21: The embeddings look up matrices of Skip-gram model.

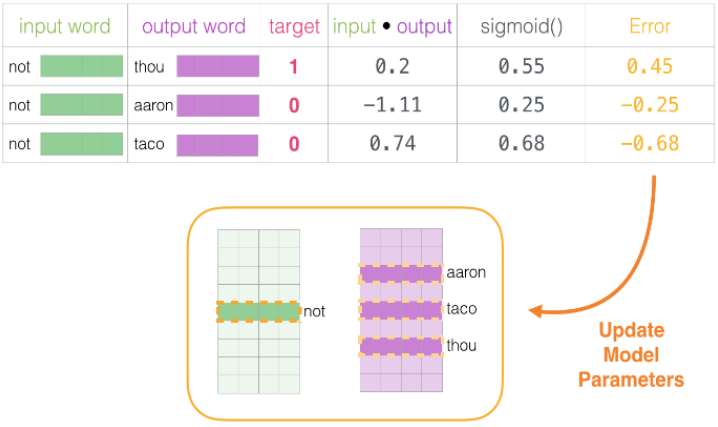

FIGURE 1.22: The training process of Skip-gram model.

Global Vectors (GloVe)

GloVe is a log-bilinear regression model for unsupervised learning of word representations proposed by J. Pennington in 2014 [55]. It combines the features of two model families, namely the global matrix factorization and local context window methods (Word2Vec). GloVe efficiently leverages statistical information by training only on the nonzero elements in a word-word co-occurrence matrix, rather than on the entire sparse matrix or on individual context windows in

a large corpus. A matrix of term co-occurrences is constructed from the whole corpus. Specifically, for each word (e.g. water), computes P(k|water) = probability of k and water to co-occur, where k=word from the vocabulary. This high-dimensional context matrix is then reduced by normalizing counts and log-smoothing.

FastText

Popular models that learn word representations usually ignore the morphology of words, by assigning a distinct vector to each word. This is a limitation, especially for languages with large vocabularies and many rare words. FastText, proposed by P. Bojanowski in 2016 [56] from Facebook's AI Research team, solved this problem. The method is based on Word2Vec's skipgram model. The novelty here is that sub-word units are considered, and words are represented by the sum of the vector representations of its character n-grams plus the word itself. As the name suggests, it is a fast method which allows training models on large corpora quickly and that's why Facebook makes available pre-trained models for 294 languages. FastText also enables us to compute word representations for words that did not appear in the training data.

Below we see the example of representing word 'where' using character 3-grams:
where , n=3 -> <wh, whe, her, ere, re>, <where>

The vector representation for word 'where' will derive from the sum of the respective vector representations from the right side entities. It is highly possible that some n-grams are shared between words with similar morphology and therefore these words will have similar representations. Sharing the representations across words allows to learn reliable representations for rare words while previous models performed poorly from that aspect.

Classic word embeddings problem

A problem that we have to face when using classic word embeddings is polysemy. Polysemy is met when the same word has different meaning based on its context. The solution to this problem is brought by Contextualized embeddings. These embeddings are similar with the classic ones but they additionally take into account the context of word.

1.4.2.2 Contextualized word embeddings

In contrast with classic word embeddings, contextualized are not standard for each word. As the name suggests, they change according to the context that the word is found in. This means that the same word can have more than one representation. For example the word 'match' can be found with words 'football' or 'fire' having a total different meaning in each case. Therefore, contextualized word embeddings are very useful and lead to better results in many NLP

tasks. We will discuss about four popular language models that learn contextualized word embeddings.

Contextualized Vectors (CoVe)

CoVe, proposed by B. McCann in 2017 [30] was the first approach that tried to solve the polysemy problem by considering a word's context when learning vector representations. An encoder-decoder model is trained on a machine translation task (supervised learning) to learn contextualized embeddings. For the encoder, a two-layer bidirectional LSTM was employed while the decoder used attentional unidirectional LSTMs. While training, the encoder must learn how to capture syntactic and semantic meanings of words, and output contextualized embeddings. Then, this pre-trained encoder can be used for a downstream task. It was shown in [30] that adding these context vectors (CoVe) improves performance over using only unsupervised word and character vectors on a wide variety of common NLP tasks.

Limitations

- Supervised training - limited labeled data
- Downstream task architecture still needs to be defined

What we want

- Unsupervised training - unlimited data
- Apply on downstream tasks with small architecture changes

ELMo

ELMo stands for "Embeddings from Language Models" and was proposed by M. Peters in 2018 [31]. ELMo models both complex characteristics of word use (e.g., syntax and semantics), and how these uses vary across linguistic contexts (i.e., to model polysemy). The difference from CoVe is that the embeddings are learned by training ELMo in an unsupervised manner. Specifically, the model learns to predict the next/previous word given the previous/next ones (bidirectional). ELMo's architecture is composed of stacked bidirectional LSTMs. The unsupervised training solved the limitation of insufficient labeled data and allowed training with huge corpora. The problem that remains unsolved though, is that we still need an extra model for downstream tasks (ELMo only gives us the embeddings).

Transformer

In this section we will discuss about the Transformer architecture on which the following two language models, BERT and GPT, are based. Transformer is a deep learning model proposed by Vaswani in 2017 [27]. It combines seq2seq (encoder-decoder) architecture [25] with the attention mechanism [26]. Transformer was originally used in NLP tasks like machine translation and text summarization but is also found in computer vision tasks.

Transformer vs RNN

Similarities

- Both can handle sequential input data

Differences

- Transformers process the whole input sequence at once (due to the attention mechanism), not sequentially (word-by-word) like RNNs
- This feature allows for more parallelization than RNNs and therefore reduces training times
- The additional training parallelization allows training on larger datasets
- This led to the development of pre-trained language models such as BERT and GPT

Transformer visual walkthrough

Below we present the transformer architecture and how it works in a sequence-to-sequence task (Machine Translation) [58].

Basic blocks

The basic building blocks of the Transformer architecture are the encoders and decoders stacks as shown in figure 1.23. In the original architecture, there are six encoders and six decoders in each stack.

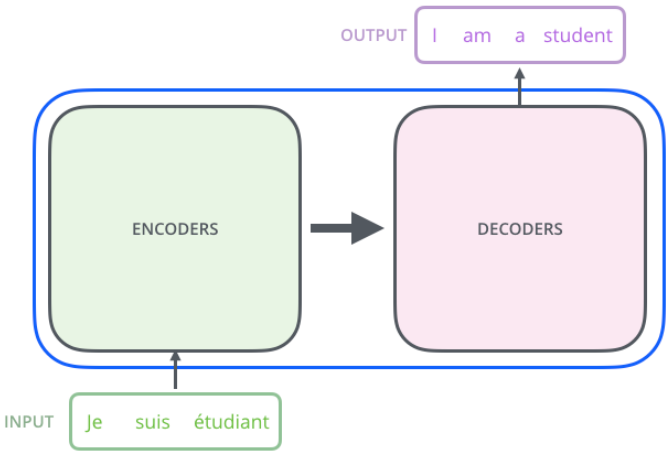

FIGURE 1.23: The encoders and decoders stacks of Transformer.

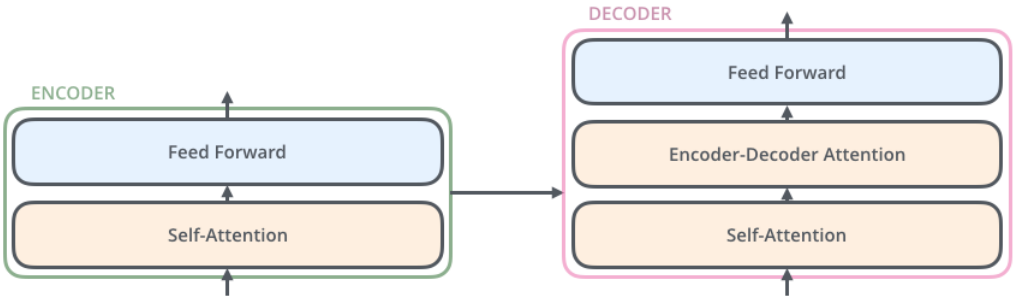

FIGURE 1.24: The inside structure of encoder-decoder blocks.

Each encoder is consisted of a self-attention layer and a feed forward neural network. The self-attention layer helps the encoder look at other words in the input sentence as it encodes a specific word. The decoder has an extra attention layer that helps focus on relevant parts of the input sentence. Figure 1.24 shows the inside structure of encoder-decoder blocks.

Encoder data flow

All the encoders receive a list of vectors each of the size 512. They pass these vectors into a self-attention layer, then into a feed-forward neural network which sends out the output upwards to the next encoder (shown in figure 1.25).

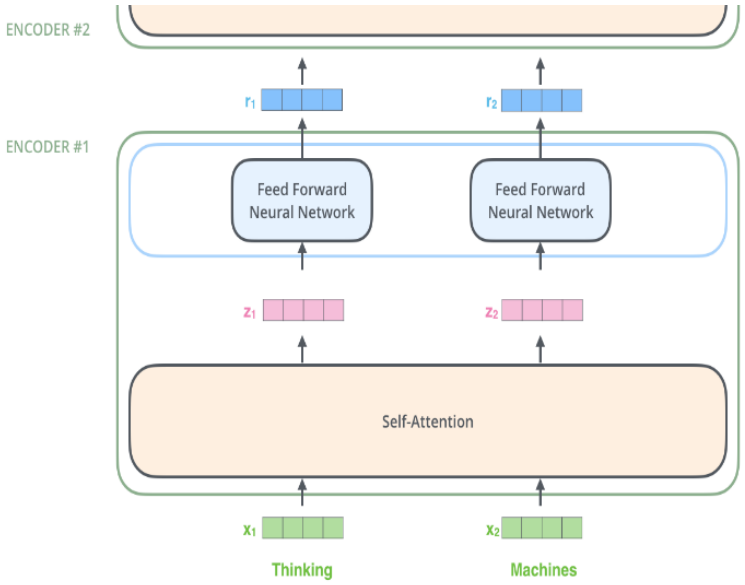

Figure 1.25: Encoder data flow.

Self-attention

The output of self-attention layer is calculated by the following six steps (depicted in figure 1.27):

1. Create Query, Key and Value vectors (size=64) from each of the encoder's input vectors (size=512). These vectors are created by multiplying the input vectors by the weight matrices Wq, Wk, Wv (figure 1.26) that we trained during the training process as the parameters of the model. For the first only encoder the input vectors are the initial word embeddings.

2. Using q,k calculate a score for each word that determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.

3. Divide the scores by the square root of the dimension of the key vectors.

4. Pass the result through a softmax operation to normalize the scores. This softmax score determines how much each word will be expressed at this position.

5. Multiply each value vector by the softmax score.

6. Sum up the weighted value vectors to produce the output of the self-attention layer at this position.

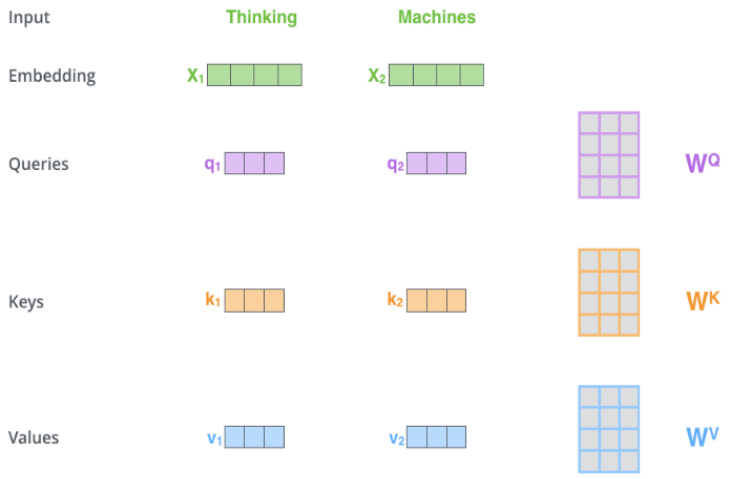

FIGURE 1.26: Query, Key and Value vectors.

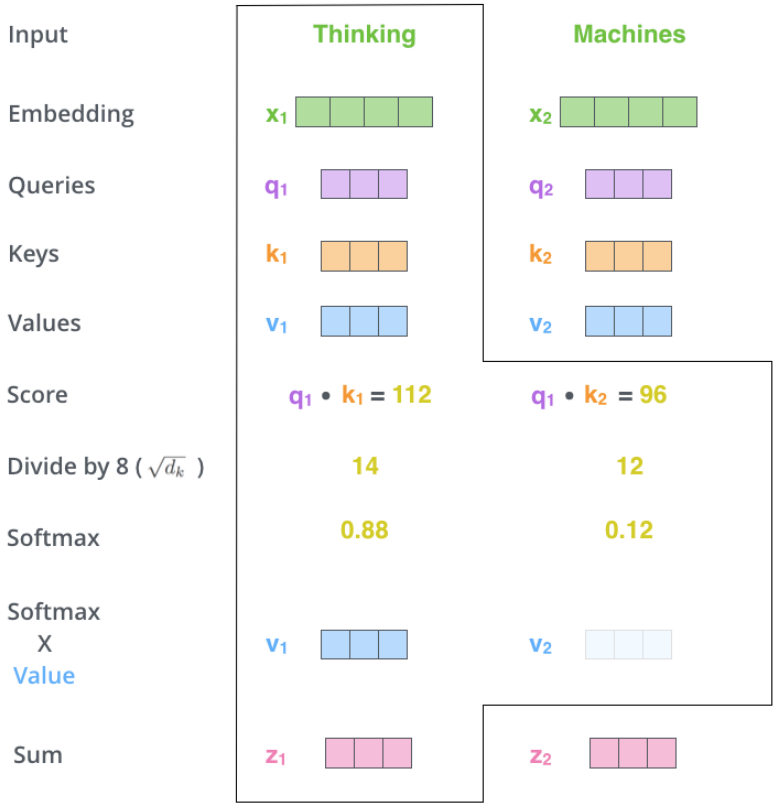

FIGURE 1.27: The Self-attention calculation steps.

That concludes the self-attention calculation. The resulting vector is one we can send along to the feed-forward neural network. In the actual implementation, however, this calculation is done in matrix form for faster processing. Specifically, the input embeddings are packed into a matrix X and then multiplied by weight matrices Wq, Wk, Wv to give us the Query, Key and Value matrices (instead of vectors) respectively. Finally, steps two through six can be

performed with matrix calculations as shown in figure 1.28 to produce the output matrix Z of the self-attention layer.

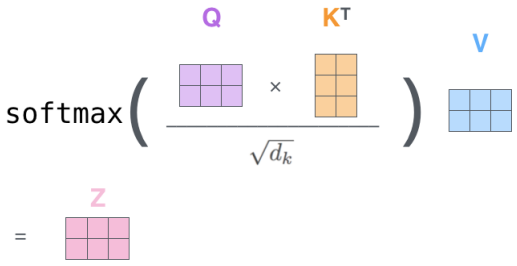

FIGURE 1.28: The Self-attention calculation steps in matrix form.

Multi-headed attention

The paper [27] further refined the self-attention layer by adding a mechanism called "multi-headed" attention. This improves the performance of the attention layer in two ways:

1. It expands the model's ability to focus on different positions. In the example above, z1 contains a little bit of every other encoding, but it could be dominated by the actual word itself.

2. It gives the attention layer multiple "representation subspaces" since there is not only one, but multiple sets of Query/Key/Value weight matrices.

Each of these sets is randomly initialized. Then, after training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.

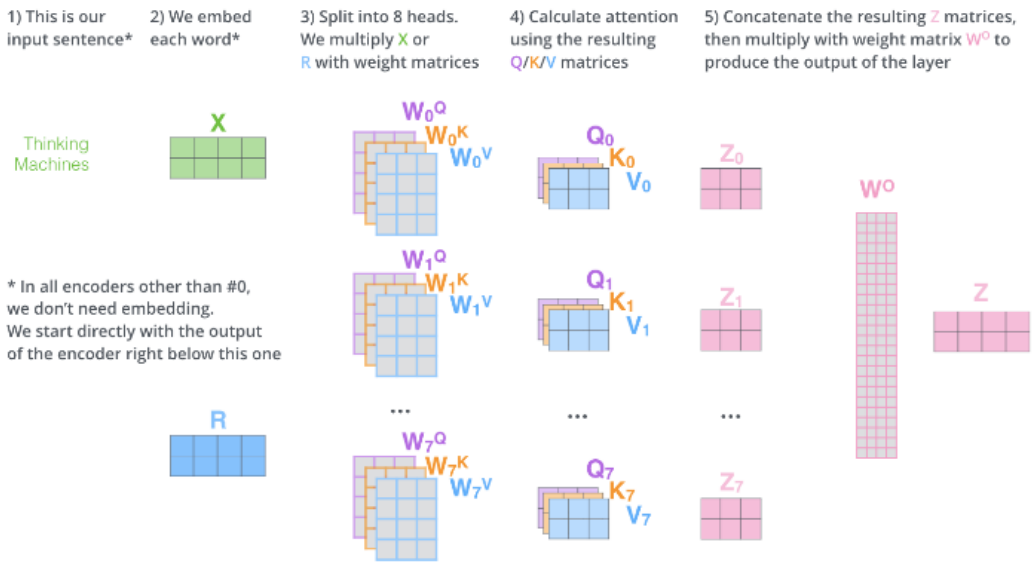

FIGURE 1.29: The Multi-headed attention calculation.

In multi-headed attention we do the same self-attention calculation eight different times with different weight matrices and we end up with eight different Z matrices. The problem is that feed-forward layer expects a single matrix not eight. That's why these eight matrices are concatenated and then multiplied by an additional weights matrix WO that was trained jointly with the model to get the final Z matrix and pass it to the feed-forward layer. The multi-headed attention calculation steps are shown in figure 1.29.

Positional encodings

To account for the order of words in the input sequence, the transformer adds a vector - positional encoding - to each input embedding as shown in figure 1.30.

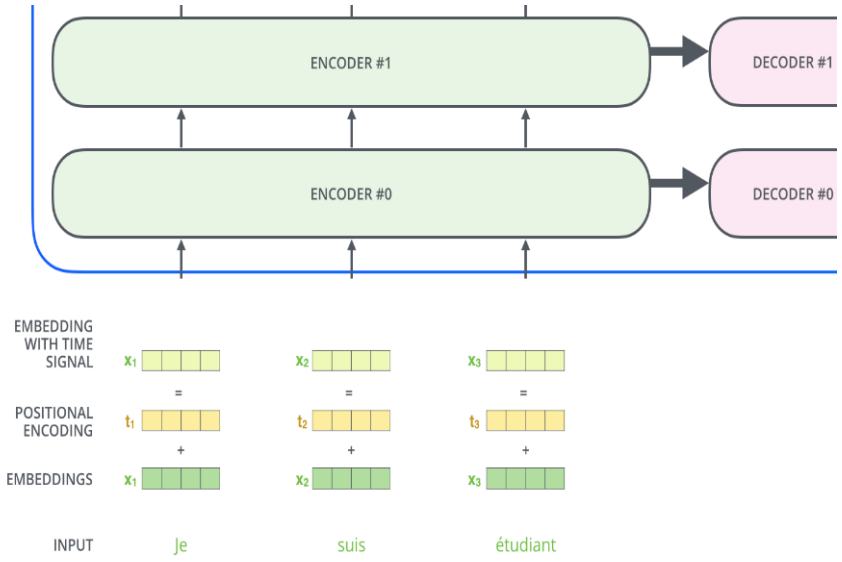

FIGURE 1.30: Addition of positional encodings to account for the order of words.

Decoder

The encoders start by processing the whole input sequence. The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its "encoder-decoder attention" layer which helps the decoder focus on appropriate places in the input sequence. At each time step the decoders stack output one word from the new (translated) sequence. The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did. Like the encoder inputs, positional encodings are added to those decoder inputs to indicate the position of each word. These steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output. The procedure is depicted in figure 1.31.

The self attention layers in the decoder operate in a slightly different way than the one in the encoder. In the decoder, the self-attention layer is only allowed to attend to earlier positions in

the output sequence. This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation. The "Encoder-Decoder Attention" layer works just like multi-headed self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.

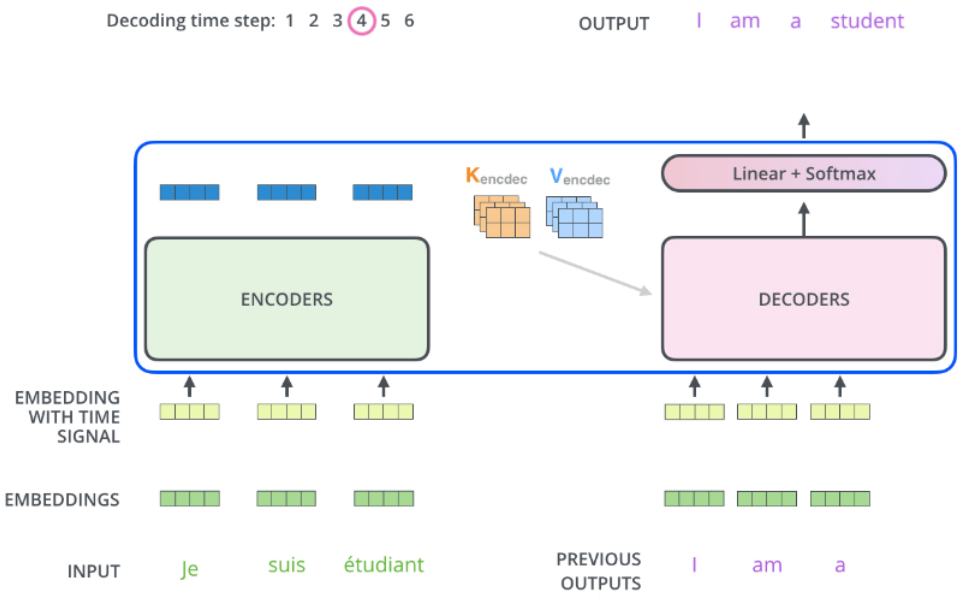

FIGURE 1.31: End to end machine translation with Transformer.

Final output

The decoder stack outputs a vector of floats. A final Linear layer which is followed by a Softmax layer are responsible for converting the float vectors into words (as shown in figure 1.32).

Linear layer (Fully-Connected) : Projects the vector produced by the stack of decoders, into a much larger vector called a logits vector.

Logits vector size = output vocabulary size (each cell corresponds to the score of a unique word)

Softmax layer : Turns those scores into probabilities. The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

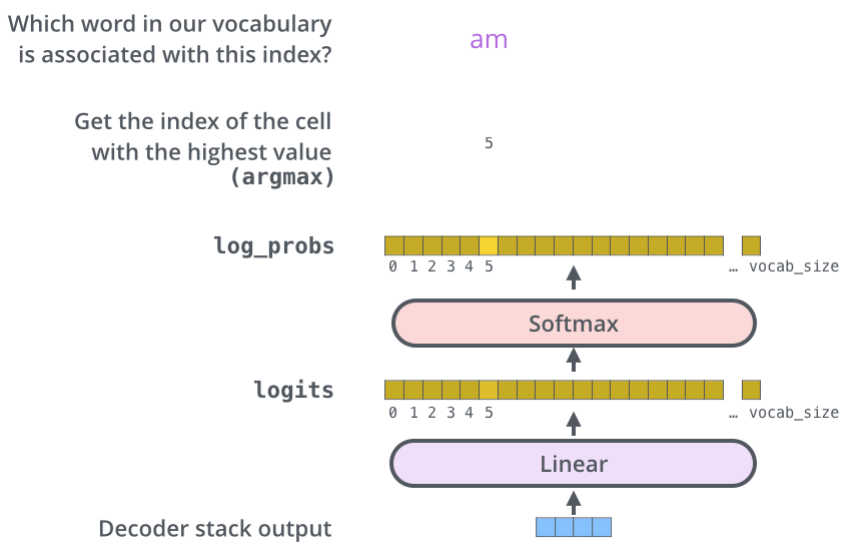

FIGURE 1.32: Linear and Softmax layers convert the Decoder stack output into a word.

Pretrained language models based on Transformer

OpenAI Generative Pre-trained Transformer (GPT)

On June 11, 2018, OpenAI researchers and engineers posted their original paper on generative language models [32]. Although large unlabeled text corpora are abundant, labeled data for learning supervised tasks is scarce, making it challenging for discriminatively trained models to perform adequately. To alleviate this limitation the authors proposed generative pre-training of a language model on a diverse corpus of unlabeled text, followed by discriminative fine-tuning on each specific task. Its architecture is based on Transformer's decoder. Unlike ELMo, GPT is trained only to predict the future. In contrast to previous approaches, GPT can be used directly for all end tasks with only slight modifications. GPT outperforms discriminatively trained models that use architectures specifically crafted for each task as presented in [32].

vs ELMo

- ELMo gives richer embeddings due to its bidirectional nature (understands better the context of the word)
- ELMo can't be used directly for all end tasks (only encoding-front end)

Bidirectional Encoder Representations from Transformers (BERT)

BERT was proposed by J. Devlin (Google research team) in 2018 [33]. It is at its core a transformer language model with a variable number of encoder layers and self-attention heads. The

architecture is ”almost identical” to the original transformer implementation in [27]. The original English-language BERTBASE consists of 12 encoders with 12 bidirectional self-attention heads. BERT is pre-trained from unlabeled data extracted from the BooksCorpus with 800M words and English Wikipedia with 2,500M words. Its pre-training (unsupervised) consists of two tasks. A) Mask Language Model: Find the masked/hidden words by looking at their context. B) Next Sentence Prediction: With two sentences as inputs, A and B, determine if B is following A or not. Unlike recent language representation models like OpenAI GPT, BERT is designed to learn deep bidirectional representations from unlabeled text by being trained to predict the context from both left and right. The pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. BERT obtained new state-of-the-art results on eleven natural language processing tasks as shown in [33].

vs GPT

- GPT is trained only to predict the future (forward language model), giving us poorer embeddings compared to BERT (bidirectional language model).
- Both GPT and BERT can be used directly for all end tasks with minor architecture modifications.

BERT visual walkthrough

Below we present how BERT is pre-trained in two different tasks and how we can use this model to deal with NLP tasks [59].

Pre-training

Masked Language Model

The problem with bidirectional conditioning was that it would allow each word to indirectly see itself in a multi-layered context. That’s why the use of masks was introduced for the word prediction task. Specifically 15% of tokens were masked and BERT was trained to predict them from context. Beyond masking 15% of the input, BERT also mixes things a bit in order to improve how the model later fine-tunes. Sometimes it randomly replaces a word with another word and asks the model to predict the correct word in that position. We can get the intuition of the training process in figure 1.33.

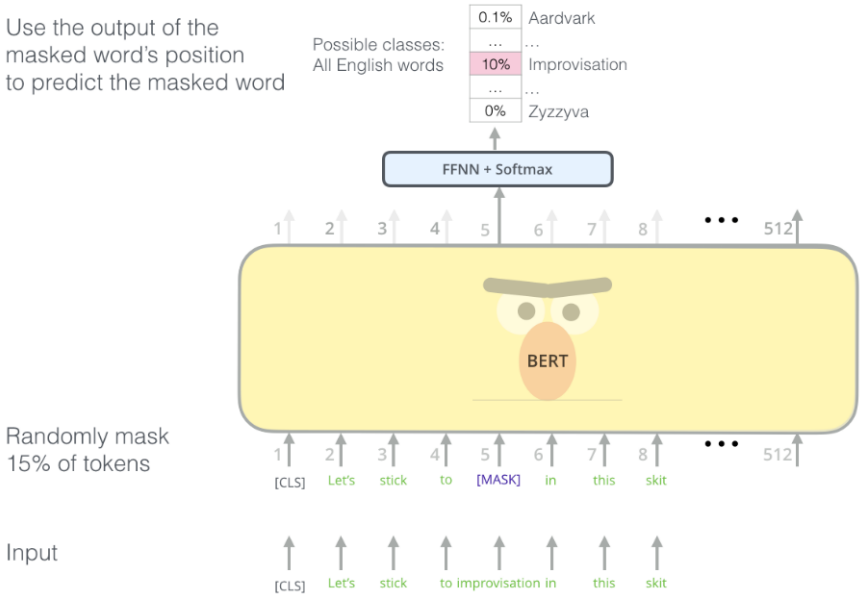

FIGURE 1.33: Masked Language Model task.

Next sentence prediction

To make BERT better at handling relationships between multiple sentences, the pre-training process includes an additional task: Given two sentences A and B, is B likely to be the sentence that follows A, or not? We can get the intuition of the training process in figure 1.34.

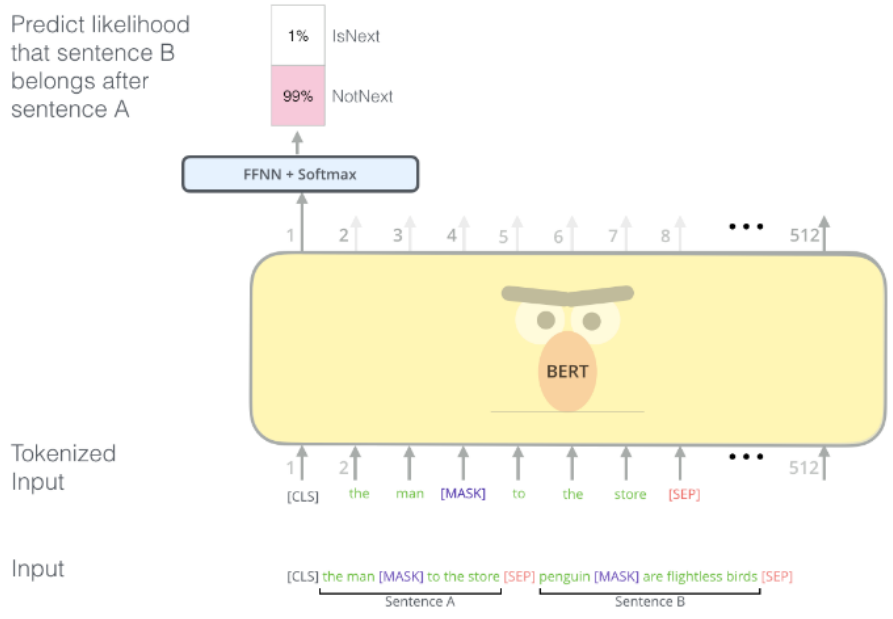

FIGURE 1.34: Next sentence prediction task.

Embeddings

The fine-tuning approach isn't the only way to use BERT. Just like ELMo, you can use the pre-trained BERT to create contextualized word embeddings. Then you can feed these embeddings

to your existing model for a downstream task.

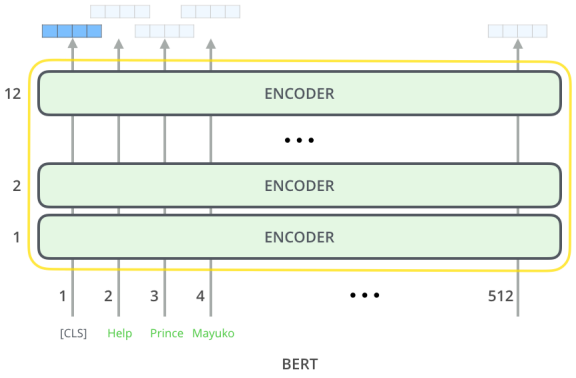

FIGURE 1.35: The encoder stack of BERT produces contextualized word embeddings.

The output of each encoder layer along each token's path can be used as a feature vector representing that token. We can take the output vectors of whichever encoder we want but usually prefer the last one as shown in figure 1.35. The embedding size would be equal to the Hidden layer size of Feed Forward Neural Network inside the encoder. The embeddings produced for each word contain info from context (contextualized word embeddings) due to Self-Attention layer inside each encoder.

1.5 Common NLP tasks

There are numerous NLP tasks that researchers are dealing with. The available tools and methods have improved significantly in the last decade, leading us to better results in many of these tasks. However, our artificial systems are still far from simulating human understanding of natural language. A list of the most popular NLP tasks is presented below [4].

1.5.1 Text and speech processing

- *Speech recognition* : Given a sound clip of a person or people speaking, determine the textual representation of the speech
- *Text-to-speech* : Given a text, transform those units and produce a spoken representation
- *Word segmentation (Tokenization)* : Separate a chunk of continuous text into separate words

1.5.2 Morphological analysis

- *Lemmatization* : Removing inflectional endings only and to return the base dictionary form of a word
- *Stemming* : Same with lemmatization but doesn't always return a valid word
- *Part-of-speech tagging* : Given a sentence, determine the part of speech (POS) for each word

1.5.3 Syntactic analysis

- *Grammar induction* : Generate a formal grammar that describes a language's syntax
- *Sentence breaking* : Given a chunk of text, find the sentence boundaries
- *Parsing* : Determine the parse tree (grammatical analysis) of a given sentence

1.5.4 Lexical semantics

- *Distributional semantics* : How can we learn semantic representations from data
- *Named entity recognition* : Given a stream of text, determine which items in the text map to proper names (e.g. person, location, organization)
- *Sentiment analysis* : Identify and categorize opinions expressed in a piece of text (e.g. positive, negative, or neutral)

1.5.5 Relational semantics

- *Relationship extraction* : Given a chunk of text, identify the relationships among named entities (e.g. who is married to whom)
- *Semantic Role Labelling* : Given a single sentence, identify and disambiguate semantic predicates (e.g. verbal frames), then identify and classify the frame elements (semantic roles)

1.5.6 Discourse

- *Coreference resolution* : Given a sentence or larger chunk of text, determine which words ("mentions") refer to the same objects ("entities")
- *Topic segmentation and recognition* : Given a chunk of text, separate it into segments each of which is devoted to a topic, and identify the topic of the segment

1.5.7 Higher-level NLP applications

- *Automatic summarization* : Produce a readable summary of a chunk of text
- *Book generation* : Creation of full-fledged books
- *Question answering* : Given a human-language question, determine its answer
- *Machine translation* : Automatically translate text from one human language to another

Chapter 2

Sentiment Analysis

In this chapter we are going to look deep into a specific text classification task of NLP, known as Sentiment Analysis (SA). We will define the problem of SA and see how we can solve it. Specifically, we briefly mention the required preprocessing steps along with some baseline algorithms used for SA. Finally, we go into more detail about the deep learning models that are currently the state-of-the-art for SA. These neural architectures were utilized for our experiments presented in the final two chapters.

2.1 About SA

2.1.1 What is SA?

Sentiment analysis or *Opinion mining* is a text classification task that belongs to the wide field of NLP. It involves the automatic interpretation and classification of sentiment (positive, negative or neutral) within text data using text analysis techniques [60].

or,

Sentiment analysis is the task of assigning a class label to a corpus of written text, where each class expresses a possible sentiment of the author concerning the content of the text. Sentiment may simply be *polarity* (ranging from very negative to very positive attitude), or multidimensional (identifying the presence or absence of different emotions). Additional semantic text properties that are correlated with opinion, besides sentiment, can also be identified using almost identical algorithms (for instance, *bias* or *sarcasm*).

2.1.2 Applications of SA

SA has many application areas where we are interested in monitoring the public opinion. Reviews (products, movies) is a classic example. Websites like amazon or IMDB apply SA to provide viewers with information about the product. Another popular application field of SA is social media (Twitter, Facebook). These platforms are widely used by people to express their feelings/opinions. SA helps mine these opinions and get an image of what people think about certain topics (social, political etc.). Monitoring public opinion is also useful to big companies and governments for adapting their products/strategies in order to win customers/voters preference. Finally, SA can be used to make predictions. These predictions could indicate the elections winner or future stock prices. Therefore, it becomes clear that SA is a very powerful tool that provides the user with valuable insights.

2.2 A Text Classification Task

As we explained earlier, SA is actually about classifying a piece of text in usually three classes (positive, negative, neutral). Some variations involve extra classes (very positive, very negative) for fine-grained SA or less (just positive, negative) for simple SA. Emotion classes (happy, sad, angry, nervous) can be used for *Emotion Analysis* and *Sarcasm Detection* can be performed with sarcastic/non-sarcastic classes. The implementation procedure followed for all these tasks is almost identical.

2.2.1 Levels of granularity

- *Sentence level* : The piece of text to be classified is a sentence. Tweets or article headlines are some examples.
- *Paragraph level* : The piece of text to be classified is a paragraph. Movie/product reviews are some examples.
- *Document level* : The piece of text to be classified is a document. Articles or emails are some examples.

2.2.2 Baseline algorithms

The algorithms used for SA are divided in two basic categories: *Machine learning* methods and *Rule (lexicon)-based* methods. Figure 2.1 shows a tree visualization of SA algorithms.

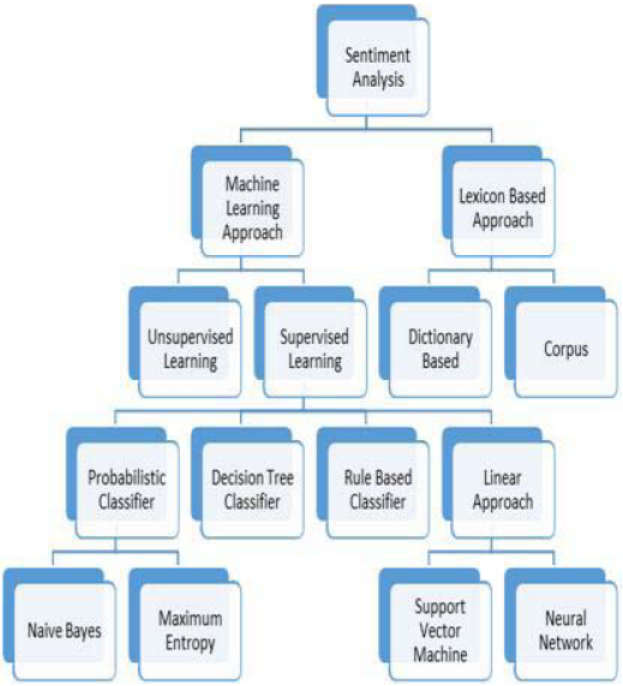

FIGURE 2.1: Algorithms used for Sentiment Analysis.

2.2.2.1 Rule (lexicon)-based methods

This is a practical approach to analyzing text without training or using machine learning models. The result of this approach is a set of hand-crafted rules based on which the text is labeled as positive/negative/neutral. These rules are also known as lexicons. Hence, the Rule-based approach is also called lexicon-based approach [61]. Widely used lexicon-based approaches are TextBlob, VADER and SentiWordNet.

TextBlob

TextBlob [62] scores each word in the lexicon with a numeric value between -1 (negative) and +1 (positive) to indicate the polarity of the word. A score of 0 is interpreted as neutral. When calculating a sentiment for a single word, TextBlob uses the "averaging" technique that is applied on values of polarity to compute a polarity score for a single word and hence similar operation applies to every single word and we get a combined polarity for longer texts.

VADER

Similarly, VADER [63] sentiment analysis relies on a dictionary which maps lexical features to sentiment scores. The sentiment score of a text can be obtained by summing up the scores of each word in the text. Sentiment score is measured on a scale from -4 (most negative)

to +4 (most positive). The midpoint 0 again represents a neutral sentiment. This dictionary was constructed by using human raters from Amazon Mechanical Turk. VADER returns a sentiment score in the range -1 to 1, from most negative to most positive. Although individual words have a sentiment score between -4 to 4, the returned sentiment score of a sentence is between -1 to 1. This is achieved by applying a normalization technique to the total score in order the output to be from -1 to 1.

SentiWordNet

SentiWordNet [64] operates on the database provided by WordNet. WordNet is a lexical database composing English words, grouped as synonyms into what is known as synsets. All the words are linked together by the ISA relationship (more commonly, Generalisation). For example, a car is a type of vehicle, just as a truck. The additional functionality that it provides is the measure of positivity, negativity and neutrality as is required for Sentiment Analysis.

Thus, every synset s is associated with a Pos(s): a positivity score, Neg(s): a negativity score and Obj(s): an objectivity (neutrality) score. The scores are very precise, pertaining to the word itself along with its context. All three scores range within the values [0,1] and add up to 1.

Pos(s) + Neg(s) + Obj(s) = 1

Analysis example of a movie review with SentiWordNet : "I disliked the movie."

The negativity score for the word dislike is 0.5. The remaining tokens, like I and the in the sentence will be filtered out during preprocessing. Meanwhile, the positivity and negativity score of movie is zero, thus making its objectivity score 1.0. Thus, the overall sentiment of the sentence will be negative, since only neutral and negative terms are used to calculate the sentiment.

2.2.2.2 Supervised ML methods

In this approach, machine learning is utilized to train models with labeled data (supervised learning). This means that we need a dataset with text samples that are manually annotated with a sentiment label in order to train the models on how to produce the desired output. Given the data, the only remaining thing is to choose the proper model and specify its hyper-parameters in order to obtain high prediction accuracies while avoiding overfitting (performing well on the training data but failing to generalize on new/unseen data). The most popular ML classifiers are listed below (see 1.3.2.2).

- Naïve Bayes Classifier

- Maximum Entropy
- Support Vector Machine (SVM)
- Decision Tree
- Random Forest
- K-nearest Neighbors
- Neural Networks

We will not get into more detail about the implementation specifics of each one for sentiment analysis. Instead we will focus on neural networks (state-of-the-art) and how they are used to analyze the sentiment found in text.

2.2.3 Text Pre-processing

Raw text data might contain unwanted or unimportant text due to which our results might not give efficient accuracy, and might make it hard to understand and analyze. So, proper pre-processing must be done on raw data before most natural language processing tasks. The ultimate goal of cleaning and preparing text data is to reduce the text to only the words that we need for our task [65].

2.2.3.1 Common pre-processing steps for SA

The most commonly used pre-processing steps for Sentiment Analysis [66] are the following:

- *Text cleaning* : Remove all unnecessary words/characters (URL, hashtags, stopwords) since they are sentimentally neutral.
- *Lowercase all words* : So as not to make the tokenizer count the same word twice (once in lower and once in capitals).
- *Stemming/Lemmatization* : Again we want to avoid having multiple words with same meaning separately handled by our algorithm.
- *Tokenization* : Convert text in sequences of integers in order to be processed easier by our model.
- *Vectorization* : Represent each word with a meaningful, numerical feature vector so the model can learn which features determine the overall sentiment.

Stemming/Lemmatization

The goal of both *stemming* and *lemmatization* [67] is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. Stemming chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization uses a vocabulary and morphological analysis of words, aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. Figure 2.2 depicts an example for each method.

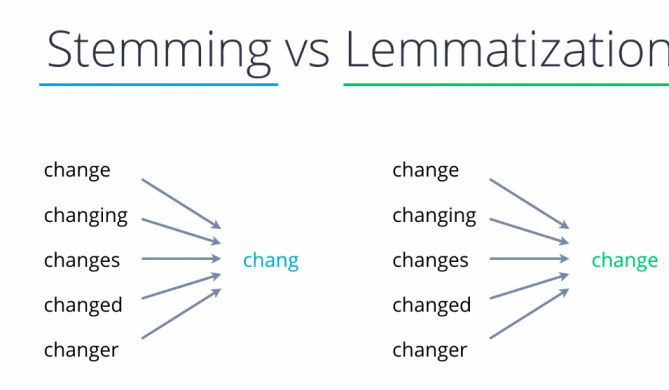

Figure 2.2: Stemming vs Lemmatization.

Tokenization

Tokenization [68] is a way of separating a piece of text into smaller units called tokens. Tokens can refer to either words, characters, or subwords. Hence, tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization. The most popular and practical type is word tokenization since words are the most meaningful entities and are easy to extract by using space as delimiter.

Tokenization is the foremost step while modeling text data. It is performed by assigning an integer value (token) to each word in the vocabulary. The term vocabulary refers to all unique words in the corpus. The obtained tokens are then used to prepare a dictionary that contains all vocabulary words paired with their corresponding token. The dictionary can be constructed by considering every unique token in the corpus or by considering the top K Frequently Occurring Words.

Figure 2.3 shows an example of tokenization. Usually, most frequent words appear higher in the token dictionary (“the”, “of”, “so”). The tokenized sentences are fed as inputs to the deep-learning models where each token (word) is then mapped to its vector representation (word embedding) and processed with the neural layers.

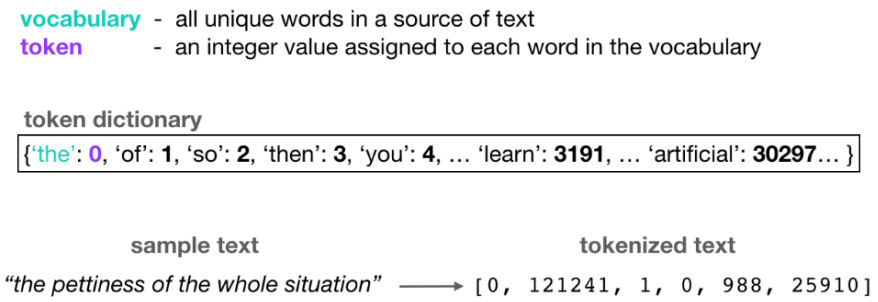

FIGURE 2.3: Tokenization example.

Text Vectorization

As we explained in 1.4, we cannot work directly with text as input so we use word representations. These representations are numerical vectors and there are many ways to get them. There are two main categories based on the produced vectors; Fixed (Sparse) and Distributed (Dense) representations. However Fixed representations cannot account for the similarity between words as they do not contain any information about the meaning of words. On the other hand, Distributed representations embed (word embeddings) the semantics of words in real valued vectors. Hence, words with similar meaning have similar vector representations. Moreover, the density of distributed representations made computationally possible the heavy task of training a neural network with such data. Figure 2.4 shows an example of three word embeddings.

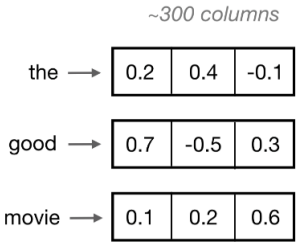

FIGURE 2.4: Three word embeddings.

2.2.4 Sentiment Analysis with Neural Networks (RNN, CNN, BERT)

2.2.4.1 Main idea

- Input: Real valued vectors (word embeddings) that represent a piece of text. For example, a sentence can be represented as a 2D matrix where each row is a word embedding.
- Output: A probability distribution across sentiment classes.

- In a neural architecture, the embeddings are output by the embedding layer or a language model (Word2Vec, GloVe, BERT).
- After getting the embeddings, text and image classification are almost the same. Both take a matrix representation (image or sentence) as input and extract high level features with hidden layers. These features are fed to the output layer/classifier (fully-connected layer) which is called to finally classify the input sample into one of the sentiment classes. The classification schema is shown in figure 2.5.

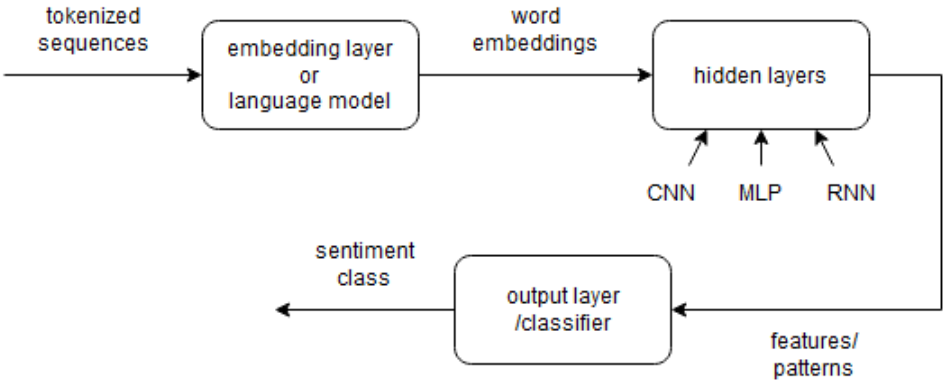

FIGURE 2.5: General sentiment analysis (text classification) schema.

Binary classification (positive/negative) : A sigmoid activation function would be used in the output layer to output a single value between 0 and 1. These two numbers correspond to the two classes and by rounding the sigmoid output we get the final output class that describes the sentiment found in the input sentence.

Multiclass classification (positive/negative/neutral) : A softmax activation function would be used. In that case the classifier would return a probability distribution (3 positive numbers that add up to 1) about the respective classes. To get the final output class we just pick the max value (highest probability) from the softmax outputs.

As depicted in the classification schema 2.5, the hidden layers commonly used are convolutional (CNN), recurrent (RNN) or fully-connected (MLP). In the following sections we will get into the details of how CNNs and RNNs work in a sentiment analysis (text classification) task.

2.2.4.2 Recurrent Neural Networks (RNNs)

RNNs (see 1.3.3.2) are designed to handle sequential data where the order of the sequence's elements is important to be modeled. In sentiment analysis, the order in which words appear in text doesn't play a significant role. We are more interested in detecting special feature words

that show sentiment. However since text is sequential data, RNNs are clearly a natural way to go. Below we examine the simple RNN model and the LSTM variant.

Simple RNN

Figure 2.6 shows how an input sequence is processed with a simple RNN model. The input sequence here is [1, 3, 4, 7]. The recurrent unit/cell (blue box) is unfolded for each time step as we go to the right. On the first time step, number 1 is the input of RNN and 0.4 is the output state that is passed to the next time step as input. On the second time step, number 3 is the input to RNN along with previous output/state 0.4. The output of the second time step is 0.9 and is passed likewise to the following time step unit. The procedure continues until we reach the final number of the sequence (7) when we obtain the final output of RNN (0.8) for the entire sequence. The final output corresponds to the entire sequence because the RNN has processed all the numbers and passed the information from each one through each time step's output as states. All that information about previous data that the final unit gets to produce the final output can be interpreted as memory.

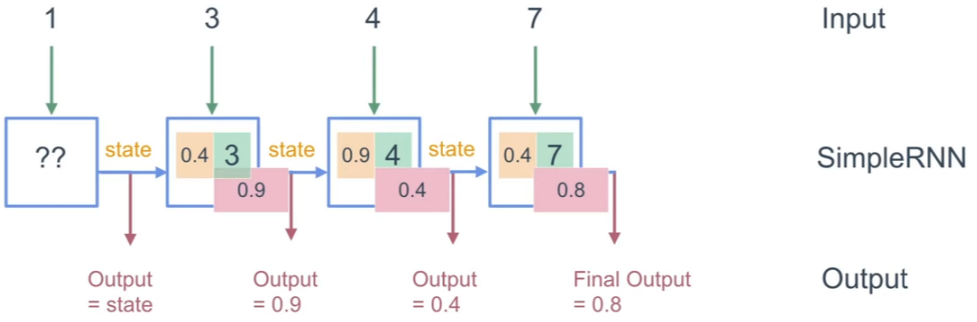

Figure 2.6: Sequence processing with a simple RNN model.

Figure 2.7 shows how the output of each cell is computed. The two inputs (current + state/previous output) are multiplied by some learned weights. The weighted sum is then passed through an activation function (tanh) to give us the output at that time step (0.9).

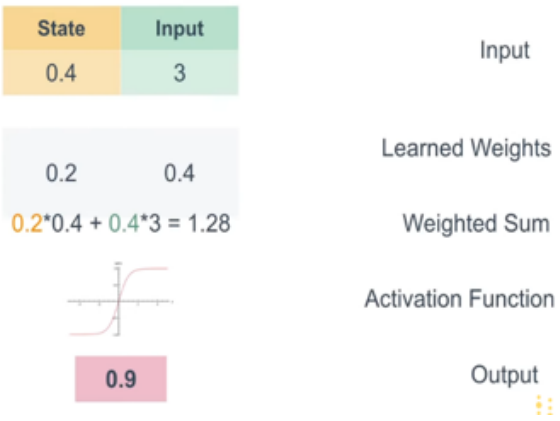

FIGURE 2.7: Calculations inside the simple RNN cell.

Problem with simple RNNs – short memory

The problem with simple RNNs is that it's really hard for them to encode long range dependencies. This has to do with the way the network is actually constructed. The input state at each time step is multiplied by a weight in order to calculate the output that is equal to the next cell's state. However, it's very hard for the network to keep the same state for a long period of time. For example if we move 20 steps forward in time, those 20 consecutive multiplications make the original state a lot smaller. This means that the network at time step 20 forgot the state at time step 1. Figure 2.8 shows the impact of consecutive multiplications in just 4 time steps.

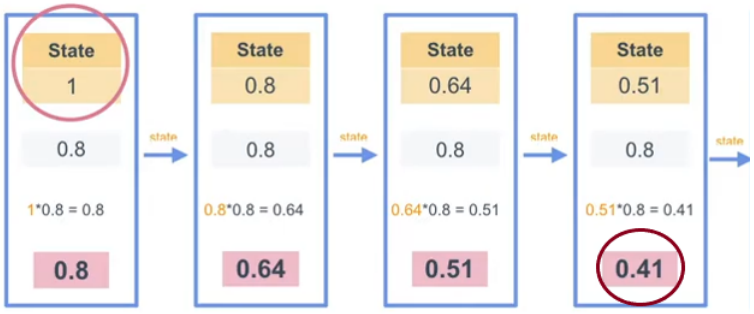

FIGURE 2.8: Simple RNN short memory problem.

Long-Short Term Memory (LSTM)

LSTMs [49] were invented to solve that problem and be able to encode both long and short term state (memory). In standard RNNs, the repeating module consists of a single tanh fully connected layer. However, in LSTMs the repeating module has four fully connected layers (3 sigmoid + 1 tanh). Figure 2.9 gives us a glimpse to the LSTM cell along with its mathematical formulation [50].

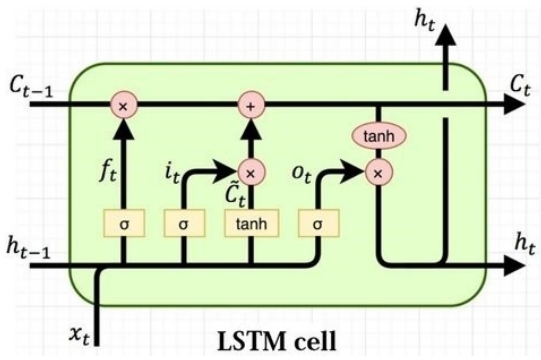

FIGURE 2.9: The LSTM cell.

All four layers (U, W) look at h_{t-1} (previous output) and x_t (current input), and output a vector (i_t, f_t, o_t, $\tilde{C}_t$) with same size as the cell state C_{t-1}. All red colored operations in figure 2.9 are element-wise.

- Forget gate layer decides what information we're going to throw away from the cell state ($f_t * C_{t-1}$).
- The tanh layer creates a vector of new candidate values $\tilde{C}_t$, that could be added to the cell state.
- Input gate layer decides which candidate values will be added to the cell state ($i_t * \tilde{C}_t$).
- Output gate layer decides what parts of the slightly modified new cell state we will output ($tanh(C_t) * o_t$).

So the forget and input gates are responsible for updating the old cell state C_{t-1}, into the new cell state C_t. All three gates (forget, input, output) have sigmoid activations in order to output values between 0 and 1 and control the data flow.

Text classification example with LSTM

For a sentiment analysis application of LSTM, we consider as input vectors x_t the word embeddings of a movie review that we want to classify as positive/negative (figure 2.10). The aforementioned calculations (figure 2.11) will be performed as many times as the number of words in the sentence. The final output of LSTM is produced at the final time step (end of input sequence) and then is fed to a sigmoid output/dense layer which gives us the final classification result.

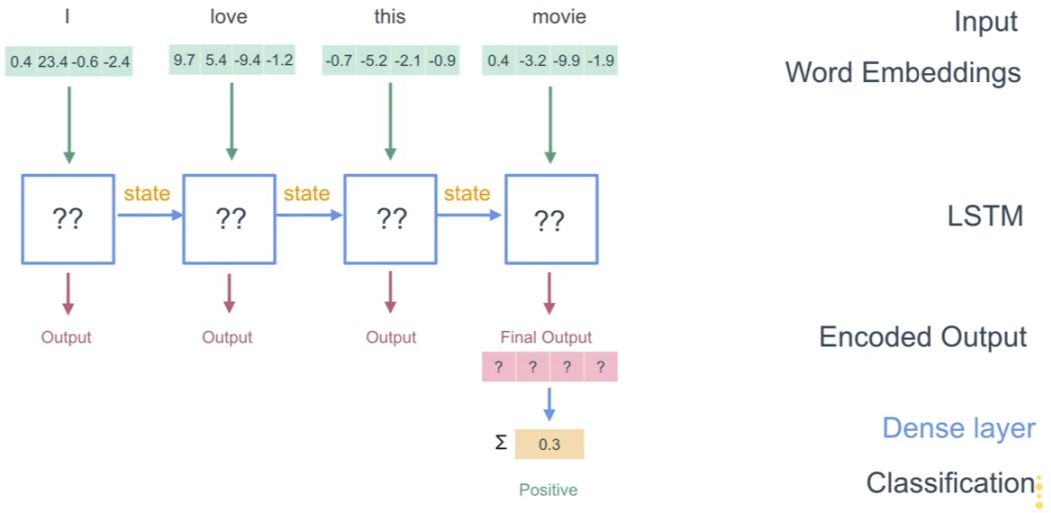

FIGURE 2.10: Sentiment Analysis of movie reviews using LSTM.

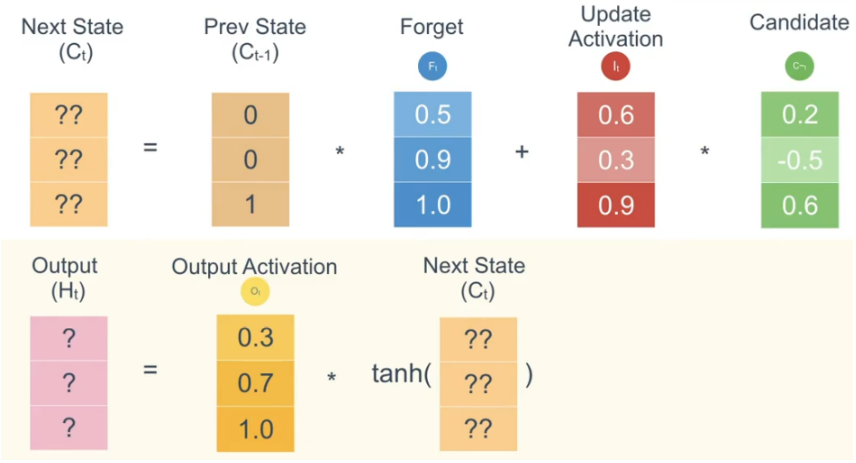

FIGURE 2.11: Calculations performed by the LSTM cell at each time step.

Of course, the first predictions will be inaccurate. This is why we need the labeled data for supervised learning. While training the model, every output is compared with the desired one and the error is used to update the models parameters/weights in order to minimize the error. When training is done, we would probably have a model able to classify the sentences correctly.

Other use cases of LSTM model for sentiment analysis are [69], [70], [71], [72], [73], [74].

Stacked LSTM

LSTMs can also be stacked in a single architecture [75]. In this case (figure 2.12), higher LSTM cells pass their outputs to the lower ones as inputs. We can use as many layers in the stack as we want. The lowest LSTM layer in the architecture produces at the final time step (end

of input sequence) the final output. The rest classification steps are the same as with a single LSTM architecture.

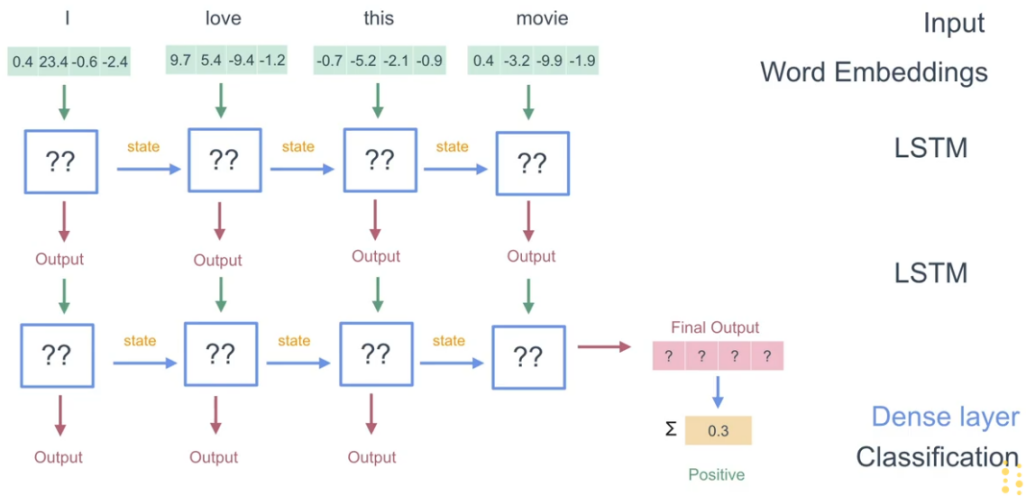

Figure 2.12: Sentiment Analysis example with Stacked LSTM.

Why increase depth?

The success of deep neural networks is commonly attributed to the hierarchy that is introduced due to the several layers [76]. Each layer processes some part of the task we wish to solve, and passes it on to the next. In this sense, the network can be seen as a processing pipeline, in which each layer solves a part of the task (easier than the whole task) before passing it on to the next layer, until finally the last layer provides the output. From another perspective, the additional layers are understood to recombine the learned representation from prior layers and create new representations at high levels of abstraction. For example, from lines to shapes to objects (computer vision).

Bidirectional LSTM

Bidirectional LSTM (Bi-LSTM) [77] uses a forward and a backward LSTM (figure 2.13). The forward one 'sees' the words in their normal reading order. The backward one 'sees' them in reverse. The final outputs from these two LSTMs are concatenated to form the final output of the whole Bi-LSTM. This vector is again fed to the output layer which gives us the sentiment score.

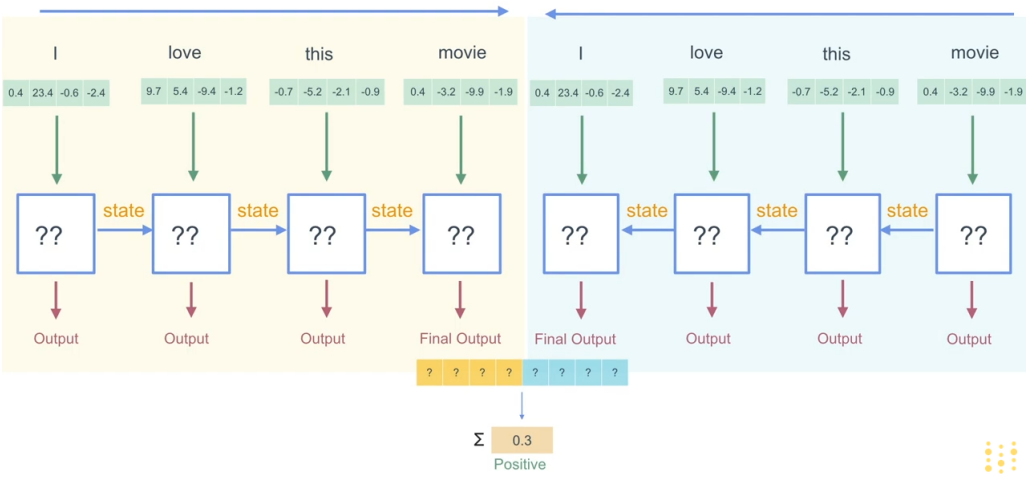

FIGURE 2.13: Sentiment Analysis example with Bi-LSTM.

Why use Bi-LSTM?

Bi-LSTM preserves information from both past and future because it has seen inputs from both the past (forward LSTM) and the future (backward LSTM). However, relying on knowledge of the future seems at first sight to violate causality. How can we base our understanding of what we've heard on something that hasn't been said yet? The use of providing the sequence bi-directionally was initially justified in the domain of speech recognition because there is evidence that the context of the whole utterance is used from humans to interpret what is being said rather than a linear interpretation. In other words, sounds, words, and even whole sentences that at first mean nothing are found to make sense in the light of future context. What we must remember is the distinction between tasks that are truly online – requiring an output after every input – and those where outputs are only needed at the end of some input segment. That is why we use Bi-LSTMs on sequence classification problems. In such tasks (sentiment analysis) they usually perform better than unidirectional LSTMs as they can understand context better [78].

2.2.4.3 Convolutional Neural Networks (CNNs)

CNNs are traditionally used in image processing since convolutional filters are designed to find spatial patterns. However, in the last decade they have been adopted for text classification tasks showing great results. Moreover, CNNs implementation allow for GPU utilization (parallelism) leading to fast training times. We will discuss how CNNs can be used to find general patterns in text and perform text classification. The following illuminating visual guide can be found in Cezanne Camacho's post [79].

Convolutional Kernels

Convolutional layers are designed to find spatial patterns in an image by sliding a small kernel window over an image. These windows are often small, perhaps 3x3 pixels in size, and each kernel cell has an associated weight. As a kernel slides over an image, pixel-by-pixel, the kernel weights are multiplied by the pixel value in the image underneath, then all the multiplied values are summed up to get an output, filtered pixel value.

In the case of text classification, a convolutional kernel will still be a sliding window, only its job is to look at embeddings for multiple words, rather than small areas of pixels in an image. The dimensions of the convolutional kernel will also have to change, according to this task. To look at sequences of word embeddings, we want a window to look at multiple word embeddings in a sequence. The kernels will no longer be square, instead, they will be a wide rectangle with dimensions like 3x300 or 5x300 (assuming an embedding length of 300). Figure 2.14 shows the shape of the convolutional kernel.

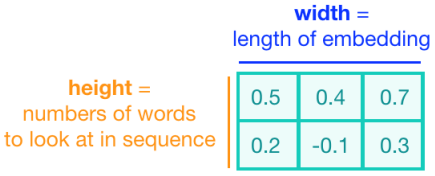

Figure 2.14: A convolutional kernel.

Convolution over Word Sequences

Figure 2.15 shows an example of what a pair (a 2-gram) of filtered word embeddings will look like. For ease of visualization the embedding length is equal to 3. To look at two words in this example sequence, we can use a 2x3 convolutional kernel. The kernel weights are placed on top of two word embeddings; in this case, the downwards-direction represents time, so, the word "movie" comes right after "good" in this short sequence. The kernel weights and embedding values are multiplied in pairs and then summed to get a single output value of 0.54.

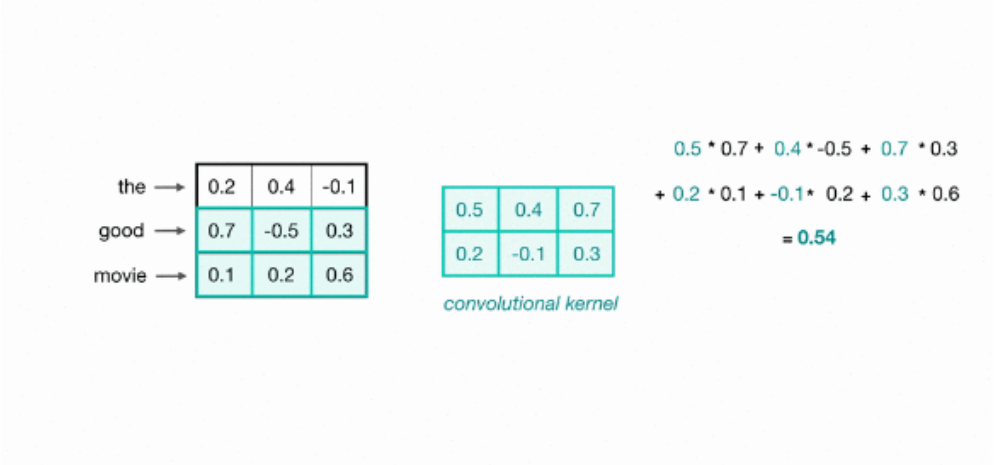

FIGURE 2.15: Convolution operation applied on text.

A convolutional neural network will include many of these kernels, and, as the network trains, these kernel weights are learned. Each kernel is designed to look at a word, and surrounding word(s) in a sequential window, and output a value that captures something about that phrase. In this way, the convolution operation can be viewed as window-based feature extraction, where the features are patterns in sequential word groupings that indicate traits like the sentiment of a text, the grammatical function of different words, and so on.

Recognizing General Patterns

As we explained in 1.4.2, similar words will have similar embeddings. Therefore, since convolution operation is just a linear operation on these vectors, when a convolutional kernel is applied to different sets of similar words, it will produce a similar output value. This is illustrated in figure 2.16.

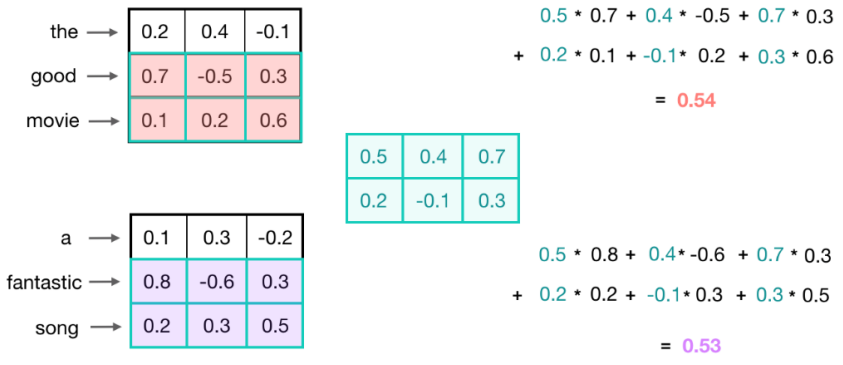

FIGURE 2.16: Example of convolutional kernel capturing positive features.

In this example, the convolutional kernel has learned to capture a more general feature; not just a good movie or song, but a positive thing, generally. Recognizing these kinds of high-level features is the key to dealing with text classification tasks, which often rely on general groupings. For example, in sentiment analysis, a model would benefit from being able to represent negative, neutral, and positive word groupings.

1D Convolutions

To process an entire sequence of words, these kernels will slide down a list of word embeddings, in sequence. This is called a 1D convolution because the kernel is moving in only one dimension: time (whereas in images the kernel moved in two dimensions -2D- as it slid across the image). A single kernel will move one-by-one down a list of input embeddings, looking at the first word embedding (and a small window of next-word embeddings) then the next word embedding, and the next, and so on. The resultant output will be a feature vector that contains about as many values as there were input embeddings. Figure 2.17 shows the output of a 1D convolution applied to a short sequence of word embeddings (movie review).

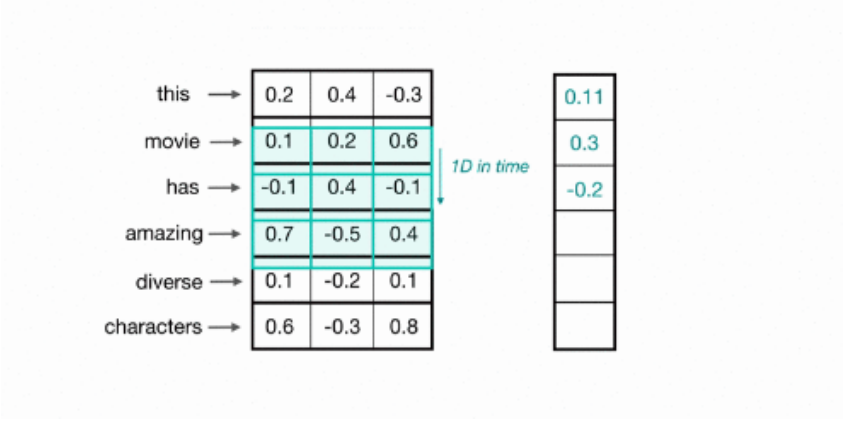

Figure 2.17: 1D convolution output.

Multiple Kernels

Just like in a typical convolutional neural network, one convolutional kernel is not enough to detect all the different kinds of features that will be useful for a classification task. To set up a network so that it is capable of learning a variety of different relationships between words, you'll need many filters of different heights. So, as a short, convolutional kernel slides over word embeddings, one at a time, it is designed to capture local features or features within a nearby window of sequential words. The stacked, output feature vectors that arise from several of these convolutional operations is called a convolutional layer.

Max-pooling

If we are trying to classify movie reviews, and we see the phrase, "great plot," it doesn't matter where this appears in a review; it is a good indicator that this is a positive review, no matter its location in the source text. In order to indicate the presence of these high-level features, regardless of the location within the larger input sequence, we use a max-pooling operation. In max-pooling, a sliding frame (like the convolutional kernel) passes over the whole input and outputs the maximum value found inside the frame at each step. So this operation forces the network to discard less-relevant, locational information by retaining only the maximum value in a feature vector, which should be the most useful. Convolutional kernels produce the strongest response to a local input pattern/feature and that is why the maximum value of the feature vector is the most useful. A pooling example is depicted in figure 2.18.

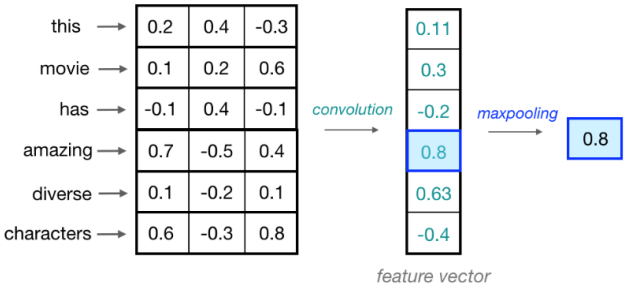

FIGURE 2.18: Max-pooling applied to the 1D convolution output.

The max-values produced by processing each of our convolutional feature vectors will be concatenated and passed to a final, fully-connected layer. This layer will produce the final sentiment score, classifying the movie review as positive or negative.

Text classification example with CNN

Figure 2.19 illustrates a complete CNN architecture used for sentiment analysis of movie reviews [80]. The complete network will see a batch of movie reviews as input. These go through a pre-trained embedding layer, then the sequences of word embeddings go through several convolutional operations. Each single kernel convolution outputs a feature vector. A whole convolutional layer outputs the feature vectors that were produced by every kernel (same height) of the layer. The max-values (max-pooling) from the different convolutional layers outputs (different kernel heights) are concatenated and passed to the output layer which outputs the sentiment class our input text piece belongs to.

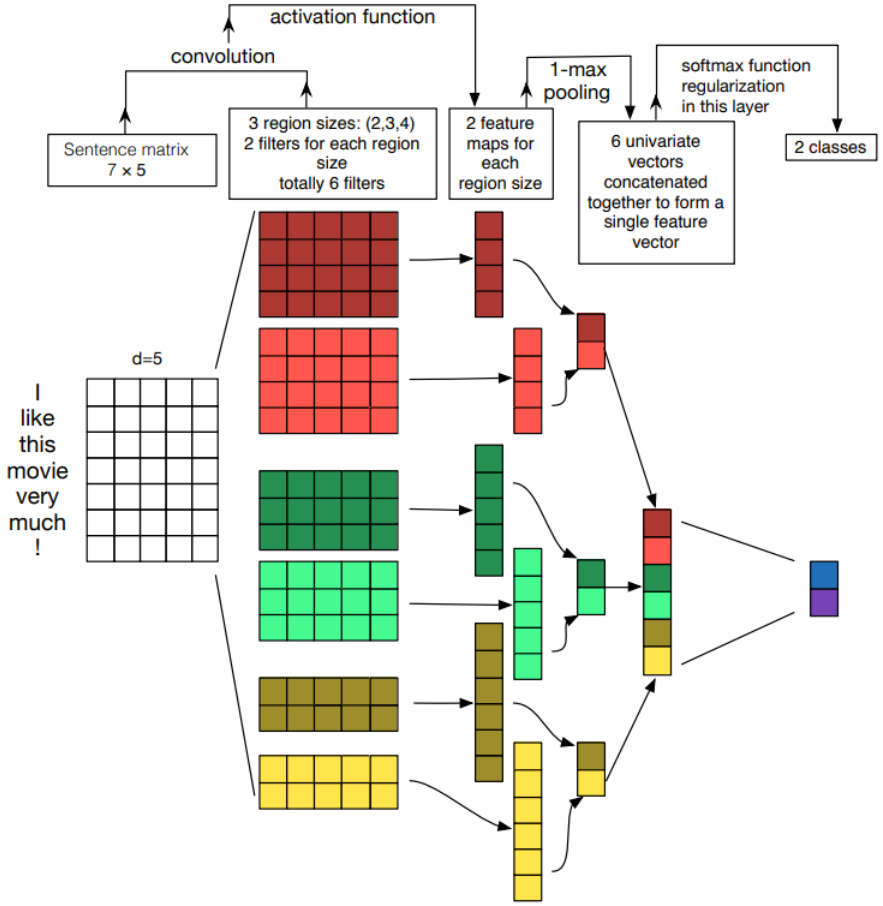

FIGURE 2.19: Sentiment Analysis (text classification) example using CNN.

Other use cases of the CNN architecture for sentiment analysis are [81], [82].

2.2.4.4 CNN+LSTM Hybrid

As we explained in 1.3.3.2, CNN and LSTM can be combined in an ensemble model that takes advantage of their good properties while compensating for the lacks of each one [83]. This means that CNN will detect local features (word groupings) and LSTM will model long-distance temporal relationships in the input sequence of embeddings. Figure 2.20 shows the most common architecture of this type of models where the CNN precedes the LSTM. The CNN is used first to reduce the dimensionality of the input by producing the feature vectors. These vectors are input to the LSTM. Therefore, the LSTM will have to process smaller inputs than the full input representations, making its training a lot faster and efficient. As always, a fully-connected output layer is needed at the end of the model for the classification result.

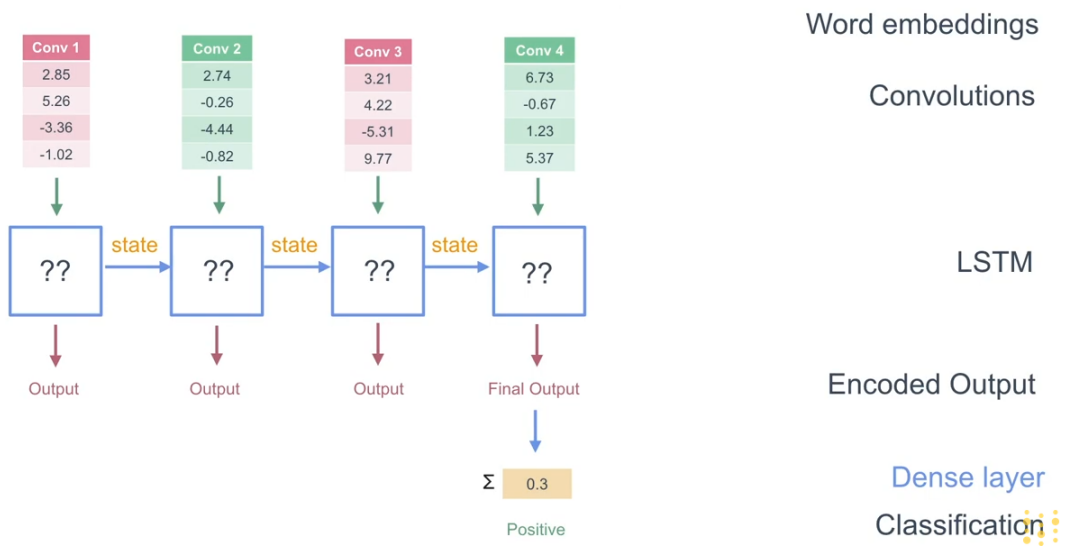

FIGURE 2.20: Sentiment Analysis example using CNN+LSTM architecture.

Other use cases of the hybrid architecture for sentiment analysis are [84], [85], [86], [87], [88].

2.2.4.5 Text classification with BERT

We explained earlier (classification schema 2.5) that a language model can be used instead of the embedding layer to produce the word embeddings from the tokenized sequences. BERT (see 1.4.2.2) is a state-of-the-art language model that leads to top results due to its high quality contextualized embeddings (see 1.4.2.2).

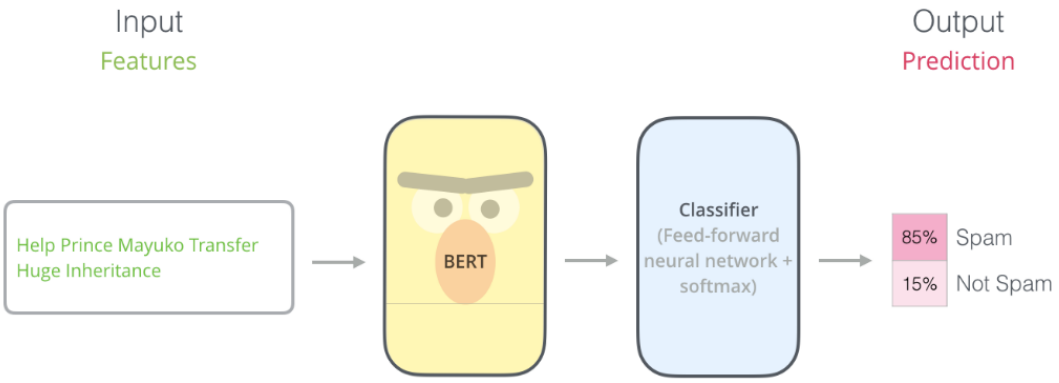

FIGURE 2.21: Text classification example using the pre-trained BERT.

For classification tasks we load the pre-trained BERT and add a classifier on top of it as shown in figure 2.21. This specific example [59] is about Spam Detection and can be used for Sentiment Analysis by just changing the classes to positive/negative. The classifier can be a CNN, LSTM, MLP or just a single fully connected layer. BERT feeds the classifier with the word embeddings

to produce the final output. Since BERT is pre-trained, we just need to train the classifier for the classification task using our labeled dataset. Another option would be fine-tuning BERT along with the classifier on the specific task for improving the overall performance. However this is computationally expensive and is usually avoided since the benefit is often very small or zero.

Chapter 3

Public Opinion Monitoring through Collective Semantic Analysis of Tweets

3.1 Abstract

The high popularity of Twitter renders it an excellent tool for political research, while opinion mining through semantic analysis of individual tweets has proven valuable. However, exploiting relevant scientific advances for collective analysis of Twitter messages in order to quantify general public opinion has not been explored. This study presents such a novel, automated public opinion monitoring mechanism, consisting of a semantic descriptor that relies on Natural Language Processing (NLP) algorithms. A four-dimensional descriptor is first extracted for each tweet independently, quantifying text polarity, offensiveness, bias and figurativeness. Subsequently, it is summarized across multiple tweets, according to a desired aggregation strategy and aggregation target. This can then be exploited in various ways, such as training machine learning models for forecasting day-by-day public opinion predictions. The proposed mechanism is applied to the 2016/2020 US Presidential Elections tweet datasets and the resulting succinct public opinion descriptions are explored as a case study.

3.2 Introduction

Social media have gradually risen to be central elements of modern life in the Western world. The easy access to on-line platforms and the benefits of constant social interaction keep the

number of users steadily growing. They allow people to directly express their opinions and feelings using a variety of media, like text, images, videos, etc.

Consequently, these platforms can be used to monitor public opinion related to a subject of particular interest. Public opinion "represents the views, desires, and wants of the majority of a population concerning a certain issue, whether political, commercial, social, or other"[89]. Twitter seems to be the medium of choice for stating opinions regarding sociopolitical matters like COVID-19, elections, racism, etc. The massive number of people utilizing Twitter for staying up-to-date and expressing their views has provided politicians with the opportunity to put their message across quickly and cheaply, without going through the traditional media briefings and news conferences [90].

The potential of Twitter regarding political events was first highlighted during the US presidential elections of 2008, where Barack Obama used the platform efficiently for his campaign [91]. After that successful Twitter campaign, all major candidates and political parties quickly established a social media presence. Moreover, the popularity of Twitter provides a unique opportunity for e-government initiatives, especially with regard to simplified communication between government institutions and citizens [92]. This may allow for greater transparency and increased citizen confidence in local institutions.

Given this very high potential of Twitter, opinion mining of tweets can provide us with valuable information. Automated semantic text analysis tools, relying on modern Artificial Intelligence (AI)-based Natural Language Processing (NLP) algorithms, can identify the mood of a tweet (e.g., polarity/sentiment [93]) with remarkable precision. Deep Neural Networks (DNNs) have greatly advanced the relevant state-of-the art, especially in *Sentiment Analysis*.

Thus, for instance, modern AI makes it possible to get a feeling about which candidate is more likely to win the next elections by analyzing the sentiment of related tweets [94]. Moreover, opinion mining can be used to determine the public's views regarding a crucial matter, e.g., a referendum [95], and help the government take the right decisions.

However, collective and multidimensional semantic analysis of tweets, based on state-of-the-art DNNs, in order to quantify and monitor general public opinion has not been significantly explored, despite the obvious potential. Most related methods only extract limited amounts of semantic content (typically polarity), instead of multiple semantic dimensions, while the tweets are processed individually; the outcomes are simply summarized for manual human overview. Automated collective analysis of social media-extracted content from multiple semantic aspects, in order to identify tendencies in the overall public opinion as a whole, is rather scarce.

This study attempts to investigate the relevant unexplored possibilities of multidimensional collective tweet analysis through state-of-the-art DNNs. Thus, a novel, automated public opinion monitoring mechanism is proposed, consisting of a composite, quantitative, semantic descriptor that relies on DNN-enabled Natural Language Processing (NLP) algorithms. A four-dimensional vector, i.e., an instance of the proposed descriptor, is first extracted for each tweet independently, quantifying *text polarity, offensiveness, bias and figurativeness*. Subsequently, the computed descriptors are summarized across multiple tweets, according to a desired aggregation strategy (e.g., arithmetic mean) and aggregation target (e.g., a specific time period). This can be exploited in various ways; for example, aggregating the tweets of each days separately allows us to construct a multivariate timeseries which can be used to train a forecasting AI algorithm, for day-by-day public opinion predictions [96]. In order to evaluate the usefulness of the proposed mechanism, it has been applied to the large-scale 2016 US Presidential Elections tweet dataset. The resulting succinct public opinion descriptions are explored as a case study.

The remainder of this study is organized in the following manner. Section 3.3 discusses related previous literature, focusing not on NLP or timeseries forecasting methods (which are exploited by us in a black-box manner) but on various existing mechanisms for extracting and monitoring public opinion by applying NLP and/or timeseries forecasting on social media posts. Section 3.4 presents the proposed semantic descriptor of public opinion. Section 3.5 discusses experimental evaluation on the 2016/2020 US Presidential Elections tweet datasets and the succinct public opinion descriptions that were derived using the proposed mechanism. Subsequently, Section 3.6 discusses key-findings on the employed datasets which were extracted using the proposed mechanism, as well as the latter's main novel contributions. Finally, Section 3.7 draws conclusions from the preceding presentation.

3.3 Related Work

This Section presents a brief overview of existing AI-based (NLP and/or DNN) approaches to semantic analysis of social media text for: a) quantitative description of public opinion, and b) timeseries forecasting.

3.3.1 Public opinion description

There is growing scientific interest on analyzing social media posts since the early 2010s, with Twitter dominating relevant research. For instance, [97] evaluated current public opinion regarding political advertising on Facebook, by automatically extracting topics of discussion from relevant tweets published in October 2019. Manual inspection of these topics was conducted,

leading to the conclusion that user perception of Facebook advertising is gradually decreasing, as a result of privacy concerns related to trust in the platform. Similarly, having the goal of analyzing tweets from the candidates' perspective, [98] and [99] processed the content of Donald Trump's and Hilary Clinton's tweets during the US 2016 presidential elections. The goal was to qualitatively identify which issue each candidate emphasized and what communication strategies they used.

However, this work does not concern topic modeling and manual inspection of identified topics. Instead, it relates to methods that exploit semantic content attributes of tweets to quantify public opinion in an automatic manner. These methods can be broadly categorized into: a) non-semantic ones, which do not perform AI-based semantic analysis on tweets, and b) semantic ones, which typically perform a type of AI-enabled sentiment analysis/opinion mining on tweets.

3.3.1.1 Non-semantic methods

Non-semantic methods only consider keyword frequencies and/or tweet volume, resulting in rather inaccurate and/or purely qualitative insights. For instance, [100] performed a statistical analysis on tweets from the Spanish 2019 presidential campaign, which were selected based on keywords and quantified through their volume over time. The goal was to reveal political discourse of the parties engaged and highlight the main messages conveyed and their resulted impact in the share of candidates' voice. Machine learning classifiers were only used to detect spammers and the conclusions were purely qualitative. Similarly, [101] investigated social media activity during a political event, by analyzing the US 2016 GOP debate and observing the volumes of special keywords in Twitter posts.

Evidently, the mechanism proposed in this work is entirely different in nature from these non-semantic approaches: it assigns each tweet a 4D semantic numerical vector acquired with DNN/NLP-based text classifiers, thus going a step further from conventional statistical approaches, and then aggregates these outputs for all tweets in order to construct an overall, quantitative public opinion descriptor.

3.3.1.2 Semantic methods without aggregation

Semantic methods are more advanced and provide more accurate results. The majority of such methods operate on Twitter, but do not treat the relevant tweets collectively and mostly just consider the volume of tweets per sentiment class. For instance, [89] presented a framework for monitoring the evolution (16 months) of public opinion related to the topic of climate change. The proposed mechanism is able to on-the-fly identify and monitor sentiment in a desired

set of individual tweets, possibly as they are being published, but only rudimentary analysis is performed to these outcomes as a collection. Thus, public opinion as a whole is scarcely considered. Having a different goal in mind, [102] conducted sentiment analysis of tweets for predicting election outcome. A simple CNN model was employed for that task, while the total volume of tweets (non-semantic attribute) was compared against sentiment polarity (volume of positive tweets) to find out which is a better predictor of election results. The semantic descriptor was found to be more accurate. Advancing on this line of research, [103] aimed to predict not only the winner but also the voting share of each candidate in the 2019 Spanish Presidential elections, by considering the volume of positive tweets per candidate. Still, the semantic attributes themselves were not aggregated over the set of all tweets.

In a similar manner, [104] performed opinion mining on Italian tweets about vaccination for a 12-month period and simply counted the number of tweets per polarity class ("against", "in favor" or "neutral") for each month. It was found that vaccine-related events influenced the distribution of polarity classes, but no aggregation of the semantic content was performed. Under an almost identical general idea, [105] aimed to determine the critical time window of public opinion concerning an event, by applying multi-emotional sentiment classification to microblog posts in Sina Weibo (published within a short time period of approximately 10 days after certain events). The volume of tweets per class (among the employed 7 emotion classes) was simply examined to find out that monitoring the negative emotions trend is crucial for predicting the influence of events.

3.3.1.3 Semantic methods with aggregation

In contrast to these approaches, a set of more advanced semantic methods do perform aggregation of the semantic content and thus treat the tweets collectively, by constructing a semantic public opinion descriptor in the form of a low-dimensional timeseries. This is exactly the method family to which the mechanism proposed in this study belongs to. For instance, [106] explored the change of public sentiment in China after "Wenzhou Train Collision", by performing sentiment analysis on posts from the Sina Weibo microblogging platform, by aggregating eight identified emotions per tweet over time in order to produce an 8D daily vector, being monitored as 8 separate timeseries for a 10 day interval. However, sentiment analysis accuracy at the time was not high and the results were not particularly useful. Similarly, in [107], tweets about the 2017 Anambra State gubernatorial election in Nigeria are semantically analyzed and the outcomes are aggregated for every two-hour interval posts. The produced time-series cover an 18-hour time-frame on the election day.

Operating also in this direction, [108] employed tweet semantic analysis and aggregation to construct an average daily sentiment timeseries for each party, covering a 21-day pre-election

period during the US presidential elections. Finally, in [109], similar ideas were applied to the prediction of cryptocurrency price returns through collective semantic analysis of tweets. Relevant timeseries were constructed through day-by-day aggregation of individual tweet semantic outcomes augmented with financial data, covering a period of 2 months, and a learning model was trained for timeseries forecasting. Twitter-derived public sentiment was found to indeed have predictive power, but it was not enough on its own for accurate forecasting. Overall, methods of this type are most similar to ours, but the scale of experimental evaluation (e.g., temporal duration of constructed timeseries) is significantly limited compared to this study. Other important differences are discussed in the following Subsection.

3.3.1.4 Semantic analysis dimensions and algorithms

The vast majority of the semantic methods presented above only utilize tweet sentiment, thus they can be considered as exploiting one-dimensional text semantics. This is most obvious in cases where the semantic analysis outputs a polarity (e.g., binary or ternary classification into positive/negative tweets, or into positive/neutral/negative ones). This limited approach is the most dominant one [102] [103] [108] [104] [109]. However, one-dimensional sentiment analysis can be considered to be the case even when multi-emotional classifiers are being employed instead of simple polarity. E.g., in [106] (expect, joy, love, surprise, anxiety, sorrow, angry and hate), [105] (happiness, like, sadness, disgust, astonishment, anger, and fear) and [89] (joy, inspiration, anger, discrimination, support). Although it is a more nuanced approach, these emotions still fall under the general umbrella of sentiment, thus these methods keep ignoring other semantic text attributes. The only case where multidimensional semantics are considered is [107], where 2 different opinion dimensions (polarity and bias) are both taken into account. In contrast, this study proposes *a 4-dimensional mechanism jointly considering polarity, bias, figurativeness and offensiveness, which are all different text attributes, and experimentally verifies their usefulness.*

Another relevant aspect is how semantic tweet analysis is performed in such published methods. The vast majority among them employ outdated algorithms for text description, relying on lexicons or older representations. E.g., [106] uses the HowNet lexicon [110], [105] uses an emotional dictionary [111] and a negative word dictionary, [108] uses SentiStrength [112], [104] uses a Bag-of-Words approach, etc. Only a few rely on modern DNN-based word representation schemes, such as [105] and [89] which exploit Word2Vec [113]. The situation is even more dire when examining the type of learning models employed for actual semantic analysis. Almost all of the presented methods utilize outdated approaches, such as [105], [104] or [107], which exploit a simple K-Nearest Neighbours (KNN) classifier, a Support Vector Machine (SVM) or Textblob's Naive Bayes Classifier[1], respectively. [103] evaluates a variety of

[1] `https://textblob.readthedocs.io/en/dev/`

traditional (non-neural) machine learning algorithms, while [109] exploits a sentiment analysis rule set [114]. Recent DNN-based learning models were only exploited in [89] (Bidirectional Long Short-Term Memory network) and [102] (Convolutional Neural Network).

Contrary to all of the above approaches, the mechanism proposed in this study relies end-to-end on *state-of-the-art DNN solutions, both for word representation and for semantic analysis.*

3.3.2 Timeseries forecasting

Forecasting of timeseries derived by sentiment analysis of tweets has mainly been previously employed for predicting future financial indices. Thus, [115] explored the effect of different major events occurring during 2012–2016 on stock markets. A similar approach was followed in [116]. [117] examined the use of polarity values, extracted from tweets about the United States foreign policy and oil companies, in order to forecast the direction of weekly WTI crude oil prices. Finally, [118] investigated whether a public polarity indicator extracted from daily tweets on stock market or movie box office can indeed improve the forecasting of a related timeseries.

In all of these cases, the only exploited semantics dimension was polarity. It was shown that forecasting accuracy improves by using polarity, but public opinion extracted through tweets was only employed as an auxiliary information source for a financial-domain task, complementing a main source (e.g., stock exchange data in [115]). Moreover, outdated word representation and opinion mining algorithms were employed in all of these studies: [115] utilized SentiWordNet[2] lexicons, [117] exploited SentiStrength and the Stanford NLP Sentiment Analyzer[3], while [118] used a Naive Bayes classifier.

Similarly, older algorithms were mainly used for the timeseries forecasting task itself. [115] exploited linear regression and Support Vector Regression (SVR), [117] utilized SVM, Naïve Bayes and Multi-Layer Perceptron (MLP) learning models, while [118] relied on linear regression, MLP and SVMs to predict the target timeseries' immediate trajectory, considering the polarity or volume of tweets as input features.

In contrast to the above methods for financial forecasting exploiting tweet-derived polarity estimations, *this study focuses on the proposed semantic public opinion descriptor itself and its potential uses for political analysis (including forecasting of public opinion).* This descriptor compactly captures *multiple opinion/semantic dimensions, instead of simply polarity.* Timeseries forecasting is utilized as an example application, among others, and *state-of-the-art DNN models are exploited during all stages, in contrast to previous methods in the literature.*

[2] https://github.com/aesuli/SentiWordNet
[3] https://nlp.stanford.edu/sentiment/

FIGURE 3.1: Quantitative public opinion forecasting using the proposed mechanism/semantic descriptor.

3.4 Proposed Mechanism

The proposed novel automated public opinion monitoring mechanism consists of a composite, quantitative, semantic descriptor that relies on NLP/DNN-based classifiers. By utilizing them, a four-dimensional vector, i.e., an instance of the proposed descriptor, is first extracted for each tweet independently, thus quantifying *text polarity, offensiveness, bias and figurativeness*. Subsequently, the computed descriptors are summarized across multiple tweets, according to a desired aggregation strategy (e.g., arithmetic mean) and aggregation target (e.g., a specific time period). The summarized descriptors can be exploited in various ways: for instance, by aggregating them on a day-by-day basis allows us to construct a multivariate timeseries which can be used to train a forecasting DNN for predicting future summarized descriptors. As an example, the pipeline of such a public opinion forecasting application/case study employing our proposed mechanism is depicted in Figure 3.1.

Below, the steps of computing the proposed public opinion descriptor are analyzed in detail, along with the algorithmic machinery for implementing the process.

3.4.1 Step 1: Selecting the desired pool of tweets

The Twitter API allows easy and automated extraction of tweets based on manually set criteria about their topic and date. For instance, the presence of specific keywords and/or hashtags, the tweet timestamp, the fact of having been published within a desired range of dates, etc. Monitoring public opinion concerning an issue (e.g., attitude towards the incumbent party during an extended pre-election period) evidently requires smart adjustment of these criteria, so that an actually relevant corpus of user messages can be obtained. However, in general, this is extremely straightforward and simple, therefore we will not elaborate further.

3.4.2 Step 2: Individual descriptor extraction per tweet

The second step of the proposed mechanism is to semantically describe each tweet from the selected message pool as a 4-dimensional (4D) real-valued opinion vector. This description vector is separately extracted for each Twitter message. A set of 4 pretrained DNN models

are employed to this end. Based on the state-of-the-art in NLP, two different neural architectures were employed. The hybrid CNN-LSTM from [119] was separately trained three time ex nihilo, using three different public annotated datasets for recognizing *offensiveness*, *bias* and *figurativeness* (sarcasm, irony and/or metaphor) in tweets. Additionally, a state-of-the-art pretrained, publicly available neural model[4] was employed for *polarity* recognition. In all four cases, the desired task was posed as binary text classification, with corresponding tweet labels (offensive/non-offensive, biased/non-biased, figurative/literal, negative/positive). Importantly, forcing classification to be as uncomplicated as possible (discriminating between two classes is typically easier than discriminating between multiple classes) renders the employed DNNs more robust, accurate and dependable for the actual problem tackled by the proposed mechanism, i.e., public opinion monitoring. All trained classifiers output a real value within the range $[0, 1]$ for each test tweet, with a value of 0/1 implying perfect and uncontested assignment of one of the two opposite labels (e.g., 0/1 means definitely and fully negative/positive sentiment, respectively, in the case of polarity).

Practical details about training the four DNNs that form the backbone of the proposed mechanism follow below.

3.4.2.1 Training Datasets

SemEval-2019 Task 6 sub-task A (S19-T6) [120]: This dataset contains 14,100 tweets annotated as offensive/non-offensive and was used for training the offensiveness recognition DNN.

Political Social Media Posts (PSMP) from Kaggle[5] : This dataset contains 5,000 messages from Twitter and Facebook annotated as neutral/partisan and was used to create a bias recognition DNN. The presence of Facebook messages in the dataset did not pose a problem, as they also lie in the general category of short opinionated texts, similarly to tweets.

Tweets with Sarcasm and Irony (TSI) [121]: This dataset contains approximately 76,000 tweets annotated as ironic/sarcastic/figurative/literal. In the context of this study, the first three classes were grouped in a single class called "figurative", so that a binary figurativeness recognition DNN could be trained.

YouTube Comments (YTC): Moving on to the pretrained polarity recognition DNN, it was originally trained on a dataset with 12,559 YouTube comments. The comments were scrapped from 8 different YouTube videos related to the 2020 US presidential elections. Annotation was performed automatically via TextBlob[6] and so a positive/negative label was assigned to each comment.

[4] https://github.com/DheerajKumar97/US-2020-Election-Campaign-Youtube-Comments-Senti
[5] https://www.kaggle.com/crowdflower/political-social-media-posts
[6] https://textblob.readthedocs.io/en/dev/

3.4.2.2 Text Preprocessing

Identical preprocessing was applied to the three training datasets, i.e., stopwords, hashtags, mentions and URLs were removed. These entities either provide us no semantic information, or only encode information about the discussed topic. However, the proposed mechanism assumes that the topic has been manually selected by the user (in Step 1); therefore, the presented automated individual tweet descriptor relying on the four pretrained DNNs only captures complementary semantic information, such as polarity, bias, etc. Additionally, lemmatization was applied to avoid having multiple words with identical meaning. Finally, all words were converted into lower-case. These are typical text preprocessing options in NLP.

3.4.2.3 Neural Models

The DNN architecture that was separately trained for offensiveness, bias and figurativeness recognition [119] consists in a hybrid, parallel BiLSTM-CNN. The input text representations (computed after preprocessing), fed to this neural architecture during both the training and test stages, are derived by using 200-dimensional embedding vectors from a pretrained GloVe model [122]. The CNN applies convolution of kernel sizes 3, 4 and 5, thereby learning fixed length features of 3-grams, 4-grams, and 5-grams, respectively. The convolution is followed by a ReLU activation function. These convolutional features are then downsampled by using a 1-D max pooling function. The CNN outputs are concatenated, combined with the BiLSTM output and jointly fed into a fully connected output neural layer, activated by a sigmoid function to produce the final semantic score: a real number in the range [0, 1].

The pretrained neural model employed for polarity recognition is based on a BiLSTM architecture. A fully-connected embedding layer is used in the front-end of the network, that has learnt to map each input word to a 200-dimensional vector representation. These embeddings are fed to a BiLSTM layer followed by a max-pooling operation. Then, multiple ReLU-activated fully-connected neural layers with dropout are employed to further reduce the dimensionality of the output. The produced dense feature representation is fed to the final sigmoid-activated fully-connected layer that gives us the final real-valued sentiment score for polarity in the range [0, 1].

Table 3.1 summarizes the achieved recognition accuracy (%) of each of the four opinion classifiers on the test set of the respective training dataset.

During the test stage for all four DNN models, each pretrained DNN is actually employed for computing a different part of the individual 4D real-valued tweet descriptor for each incoming tweet. The output semantic score denotes how offensive/biased/figurative/negative the message is judged to be. An output score of 0 means very high possibility of it being non-offensive,

TABLE 3.1: Achieved accuracy of each of the four opinion classifiers on the test set of the respective training dataset.

Model	***Accuracy***
Bias	75.64%
Figurativeness	84.45%
Polarity	88.00%
Offensiveness	84.00%

non-biased, figurative and negative, respectively. An output score of 1 means very high possibility of it being offensive, biased, literal and positive, respectively. A score near 0.5 would imply that the tweet is judged to be neutral in the corresponding attribute, or it simply cannot be classified.

3.4.2.4 Hyperparameters

The optimal hyperparameters used for training the DNN classifiers were obtained by manual tuning and are presented in Table 3.2. The pretrained polarity classifier used the hyperparameters found in the relevant software repository[7].

N_layers denotes the number of hidden layers in the LSTM and N_hidden is the size (nodes) of these layers. Weight_decay denotes the L2 regularization factor. Lr_decay denotes the learning rate multiplying factor. Wd_multiplier denotes the weight decay multiplying factor. Batch_size denotes the number of tweets processed at each step of the optimization. Dropout, dropout_enc and dropout_op denote the dropouts used after the LSTM, Embedding and Output layer respectively.

TABLE 3.2: Hyperparameters used for training the sentiment classifiers.

Hyperparameter	***Values***
n_hidden	200
n_layers	2
dropout	0.5
weight_decay	1e-7
dropout_enc	0.2
dropout_op	0.5
lr_decay	0.7
wd_multiplier	6
learning_rate	5e-3
batch_size	52

[7] https://github.com/DheerajKumar97/US-2020-Election-Campaign-Youtube-Comments-Senti

3.4.3 Step 3: Aggregation

Having obtained an individual 4D, real-valued semantic descriptor per tweet, the next step is to aggregate the derived vectors into the desired public opinion descriptor. To do this, the user has to select the **aggregation strategy** and the **aggregation target**.

The target refers to the granularity of the aggregation process and directly influences the final number of aggregate public opinion description vectors. Two possible choices are the most straightforward:

- *Complete aggregation.* All individual tweet descriptors are merged into a single aggregate public opinion description vector, representing the entire message pool extracted in Step 1.
- *Temporally segmented aggregation.* The overall range of dates out of which the entire message pool was extracted in Step 1 is partitioned in isochronous, non-overlapping and consecutive time periods. All individual tweet descriptors falling under each period are merged into a single aggregate public opinion description vector. This is separately performed for each period.

With complete aggregation, the outcome is a single 4D vector. With temporally segmented aggregation, the outcome is a 4D timeseries. Examples of temporally segmented aggregation targets would be day-by-day or week-by-week aggregation. Depending on the application, different additional aggregation targets may also be envisioned.

The aggregation strategy refers to how a set of individual 4D tweet descriptors are combined into a single aggregate 4D descriptor. Three possible choices are the most straightforward:

- *Element-wise vector mean.*
- *Element-wise vector median.*
- *Element-wise vector trimmed mean.*

All three of these choices may be implemented simply by performing computations separately along each of the four descriptor dimensions. As before, different additional aggregation strategies may also be envisioned, depending on the application.

3.5 Evaluation

The proposed mechanism was evaluated on the well-known 2016 and 2020 United States Presidential Election Tweets datasets, using a day-by-day temporally segmented aggregation target.

All three aggregation strategies described in Section 3.4 were separately followed and assessed. Details and results follow in the next Subsections.

3.5.1 Datasets

The 2016 US Presidential Election tweet dataset from Kaggle[8] contains 61 million rows. Their overall time range is from 2016-08-30 to 2017-02-28, with 20 days missing, leading to a total of 163 days. From this initial dataset, we retained approximately 32 million tweets after applying a common cleaning process: removal of empty rows, of non-English text, of duplicate tweets and of messages that contained less than 5 words after text preprocessing. This is important for proper semantic analysis, since an adequate number of words per tweet is essential to achieving high opinion mining accuracy. Subsequently, the keywords "Clinton", "Obama" and "Trump" were exploited for partitioning the messages into ones referring to Democrats and ones referring to Republicans.

Overall, this entire manual process was equivalent to performing Step 1 of the proposed mechanism. It was not needed in its entirety for the smaller second dataset that we employed, i.e., the US Election 2020 Tweets from Kaggle[9], since its tweets are preseparated in two partisan groups (Democrats and Republicans). It contains 1.72 million rows in total, with an overall time range from 2020-10-15 to 2020-11-08, meaning 25 days in total. From this initial dataset, approximately 720 thousand tweets were kept after applying the cleaning process described above.

Subsequently, the proposed mechanism was separately applied to the two message pools of each dataset. As a result, two day-by-day 4D descriptor timeseries were derived, covering overall the exact same time range: one for the Republicans and one for the Democrats (separately for each dataset). Indicatively, for the 2016 US Presidential Election tweet dataset, generating the descriptors for all relevant tweets required 24 hours. Day-by-day aggregation required 10 minutes for each aggregation strategy. Experiments were performed on a desktop computer with an AMD Ryzen 5@3.2GHz CPU, 16GB of DDR4 RAM and an nVidia GeForce GTX1060 (6GB RAM) GP-GPU.

3.5.2 Analysis 1: Timeseries Forecasting

The first type of evaluation performed on the derived timeseries was to assess their predictability using AI-enabled forecasting. Since the two constructed timeseries compactly capture public opinion about the two respective parties during a heated pre/post-election period, forecasting has obvious political usefulness: it may allow an interested organization to predict

[8] https://www.kaggle.com/paulrohan2020/2016-usa-presidential-election-tweets61m-rows
[9] https://www.kaggle.com/datasets/manchunhui/us-election-2020-tweets

near-future changes in its public image, using only Twitter data. Of course, in this context, "near-future" implies a forecasting horizon of a few weeks at the most.

3.5.2.1 Implementation

The two timeseries were day-by-day 4D descriptions of evolving public opinion about the Democrats and the Republicans, respectively. However, since three different aggregation strategies were employed (mean, median, trimmed mean), in fact six 4D timeseries were derived overall, with all of them covering the same period. Since forecasting is typically performed on univariate timeseries, each descriptor channel was then handled separately, leading to a total of 24 different timeseries.

A moderate 7-day forecasting horizon was selected, since this allows for rather reliable predictions while still being practically useful. A recent stacked LSTM architecture was adopted [123], with each LSTM cell being followed by a fully connected neural layer. The overall DNN was trained separately for each timeseries, using Back-Propagation Through Time (BPTT) and a Continuous Coin Betting (COCOB) optimizer [124]. This training approach was selected over alternative optimizers (Adagrad and Adam) because it displayed superior performance in [123]. The fact that COCOB attempts to minimize the loss function by self-tuning its learning rate, accelerates convergence in comparison to other gradient descent-based algorithms with a constant or decaying learning rate, where the convergence becomes slower close to the optimum. The use of BPTT for updating model parameters during training is necessary when employing LSTMs.

Optimal hyperparameters for training the forecasting DNN model were obtained by using Sequential Model-based Algorithm Configuration (SMAC) [125] and are presented in Table 3.3. minibatch_size denotes the number of timeseries considered for each full backpropagation in the LSTM. Epoch_size denotes how many times the dataset is traversed within each epoch. L2_regularization and Gaussian noise added to the input are used to reduce overfitting. LSTM unit weights were initialized using a random_normal_initializer.

Out of the two evaluation datasets, only the 2016 one was used for training the forecasting model. A segment was withheld for test purposes from the end of each timeseries, with a length equal to the forecasting horizon; the remaining data constituted the training dataset. Moreover, this pretrained model was also separately tested on the 2020 dataset. The results from testing on both datasets are presented in the sequel.

Deseasonalisation was applied as a common preprocessing step, since DNNs are weak at modelling seasonality [126]. That was achieved by decomposing each timeseries into seasonal,

TABLE 3.3: Hyperparameters used for training the forecasting model.

Hyperparameter	*Value*
cell_dimension	20
gaussian_noise_stdev	1e-4
l2_regularization	1e-4
max_epoch_size	1
max_num_epochs	2
minibatch_size	4
num_hidden_layers	1
random_normal_initializer_stdev	1e-4

trend, and remainder components, in order to subsequently remove the seasonality component, by employing STL decomposition. If a timeseries exhibited no seasonality, this step simply returned zero seasonality.

Sliding window schemes were adopted for feeding inputs to the DNN and deriving the outputs, with the output window size n set to be equal to the size of the forecasting horizon $H = 7$. The input window size m was empirically set to $9 = n1.25$. Each training timeseries was broken down into blocks of size $m + n$, thus forming the input–output pairs for each LSTM cell instance.

TABLE 3.4: Forecasting results on the US 2016 Presidential Election Tweets dataset for the six constructed timeseries. "Dem" denotes the Democrats, "Rep" denotes the Republicans, while "mean", "med" and "trim" imply the three respective aggregation strategies: mean, median and trimmed mean. In each case, the SMAPE/MASE metrics have been independently averaged across the four descriptor channels using both the mean and the median operator. A lower value is better for both metrics, while SMAPE is a percentage.

Timeseries	*Mean SMAPE*	*Median SMAPE*	*Mean MASE*	*Median MASE*
Dem-Mean	0.1096	0.0563	1.2396	1.0799
Dem-Med	0.1380	0.0913	1.0620	1.1711
Dem-Trim	0.1798	0.0676	1.3364	1.0885
Rep-Mean	0.0529	0.0280	0.7689	0.6937
Rep-Med	0.0492	0.0330	0.5158	0.5303
Rep-Trim	0.0737	0.0314	0.6931	0.6549

3.5.2.2 Metrics and Results

A set of common timeseries forecasting evaluation quantitative metrics were employed for assessing the predictability of the computed timeseries.

First, the *Symmetric Mean Absolute Percentage Error* (SMAPE) is defined as follows:

TABLE 3.5: Forecasting results on the US 2020 Presidential Election Tweets dataset for the six constructed timeseries. "Dem" denotes the Democrats, "Rep" denotes the Republicans, while "mean", "med" and "trim" imply the three respective aggregation strategies: mean, median and trimmed mean. In each case, the SMAPE/MASE metrics have been independently averaged across the four descriptor channels using both the mean and the median operator. A lower value is better for both metrics, while SMAPE is a percentage.

Timeseries	***Mean SMAPE***	***Median SMAPE***	***Mean MASE***	***Median MASE***
Dem-Mean	0.1793	0.1490	1.6975	1.6943
Dem-Med	0.3431	0.2088	1.7526	1.6620
Dem-Trim	0.2847	0.1903	1.7535	1.7434
Rep-Mean	0.0961	0.0813	1.5233	1.4572
Rep-Med	0.1867	0.1303	1.7228	1.6949
Rep-Trim	0.1472	0.0999	1.5961	1.5637

$$SMAPE = \frac{100\%}{H} \sum_{k=1}^{H} \frac{|F_k - Y_k|}{(|Y_k| + |F_k|)/2}, \tag{3.1}$$

where H, F_k, and Y_k indicate the size of the horizon, the forecast of the DNN and the ground-truth forecast, respectively.

Due to the low interpretability and high skewness of SMAPE [127], the scale-independent *Mean Absolute Scaled Error* (MASE) metric was also employed. For non-seasonal timeseries, it is defined as follows:

$$MASE = \frac{\frac{1}{H}\sum_{k=1}^{H}|F_k - Y_k|}{\frac{1}{T-1}\sum_{k=2}^{T}|Y_k - Y_{k-1}|}, \tag{3.2}$$

In Eq. (3.2), the numerator is the same as in SMAPE, but normalised by the average in-sample one-step naive forecast error. A MASE value greater than 1 indicates that the performance of the tested model is worse on average than the naive benchmark, while a value less than 1 denotes the opposite. Therefore, this error metric provides a direct indication of forecasting accuracy relatively to the naive benchmark.

Since these metrics are computed for univariate timeseries forecasting, we employed mean and median aggregation across the four descriptor channels for each of the six timeseries. The results obtained for the 2016 and 2020 datasets are shown in Tables 3.4 and 3.5, respectively, where larger SMAPE or MASE values indicate worse forecasting accuracy.

In general, the timeseries constructed using the proposed mechanism seem to be predictable to an acceptable degree by using the employed DNN model. Moreover, forecasting behaves similarly for both datasets, leading us to draw a common set of conclusions. First, the mean

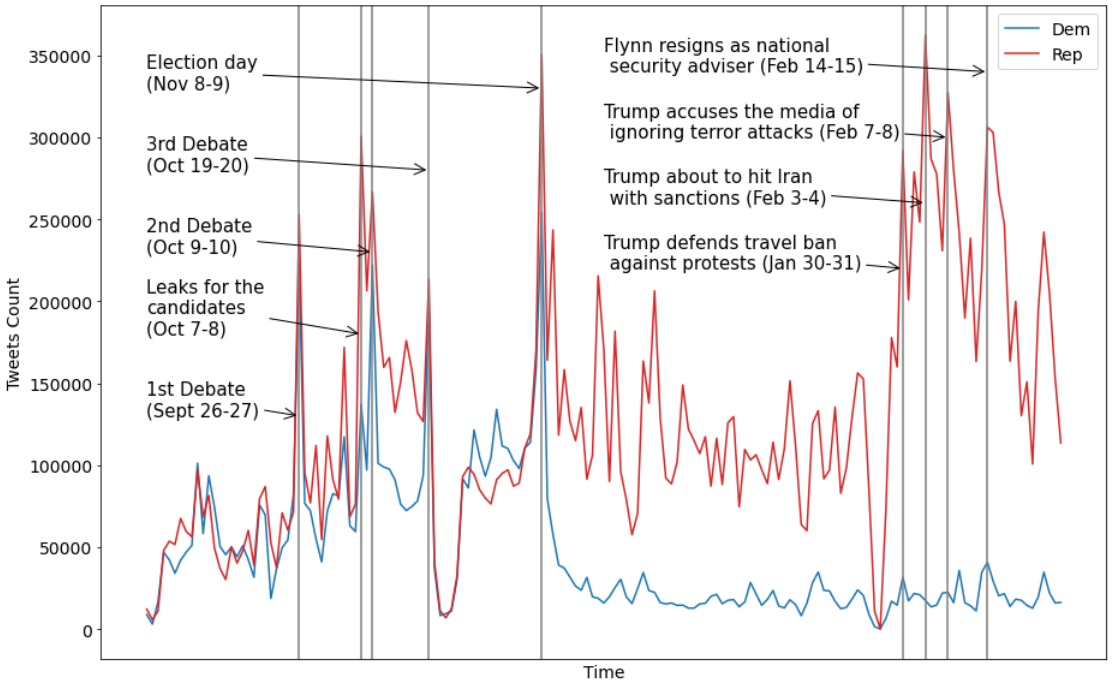

FIGURE 3.2: Daily number of tweets for Democrats (Dems) and Republicans (Reps) in 2016 dataset. The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

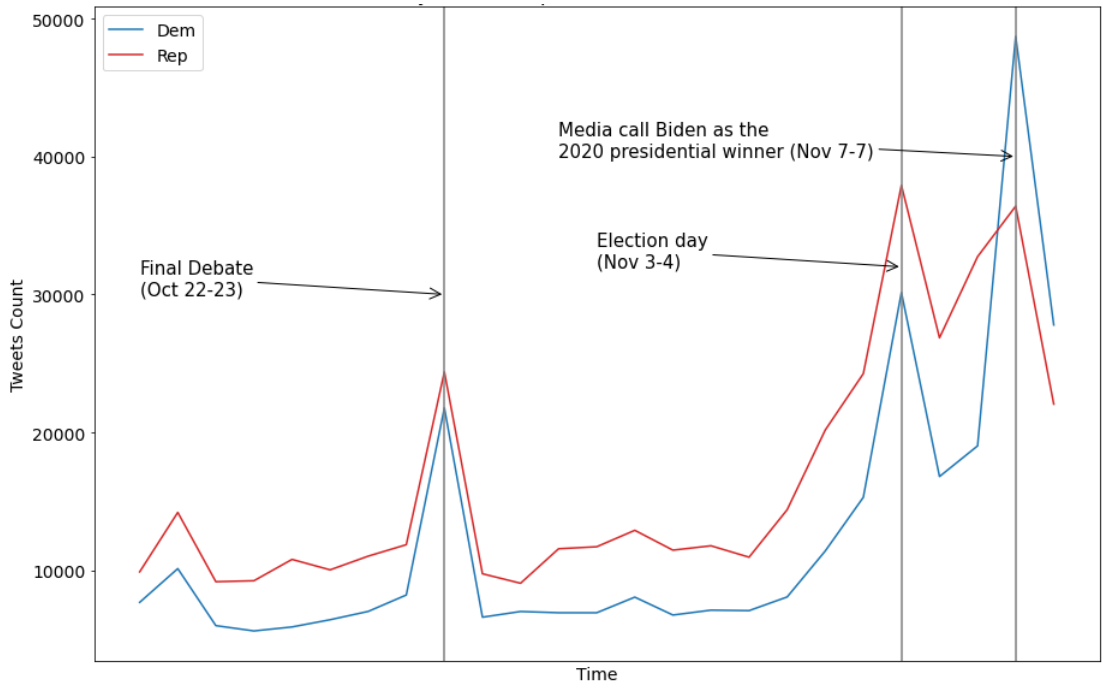

FIGURE 3.3: Daily number of tweets for Democrats (Dems) and Republicans (Reps) in the 2020 dataset. The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

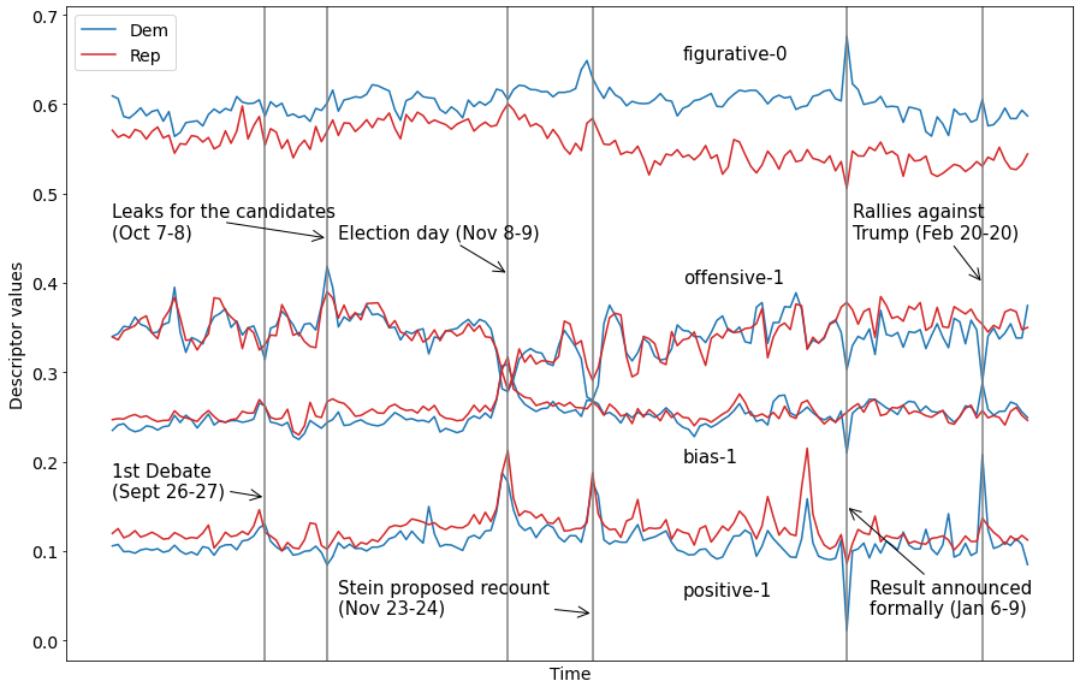

FIGURE 3.4: Per-channel day-by-day values of the 4D timeseries constructed from the 2016 data using the proposed descriptor and the mean aggregation strategy, separately for Democrats (Dems) and Republicans (Reps). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

aggregation strategy resulted in the timeseries with the best overall forecasting behaviour. Second, based on both metrics, it is clear that forecasting performs worse for the Democrats than for the Republicans, implying that public opinion concerning them (as expressed in Twitter) was less stable and predictable during the examined period. Finally, we notice a drop in accuracy when testing on the 2020 dataset (small in absolute terms), compared to the 2016 one. This is to be expected, since the forecasting DNN was pretrained only on the training set of the 2016 dataset.

3.5.3 Analysis 2: Visualizations and Qualitative Evaluation

A set of visualizations were computed from the 4D timeseries constructed using the proposed mechanism, in order to facilitate manual inspection of the outcome. Given the conclusions of Subsection 3.5.2, only the timeseries derived by mean aggregation were exploited here. This Subsection presents this qualitative evaluation process and its results, along with auxiliary information about the original 2016/2020 US Presidential Elections datasets.

First, Figure 3.2 depicts the number of tweets posted every day in the complete dataset's time range (from 2016-08-30 to 2017-02-28), separately for the Democrats and the Republicans. For the most important events like the three presidential debates and the election day, increased Twitter traffic is observed for both parties. Leaks for both Clinton and Trump that took place in

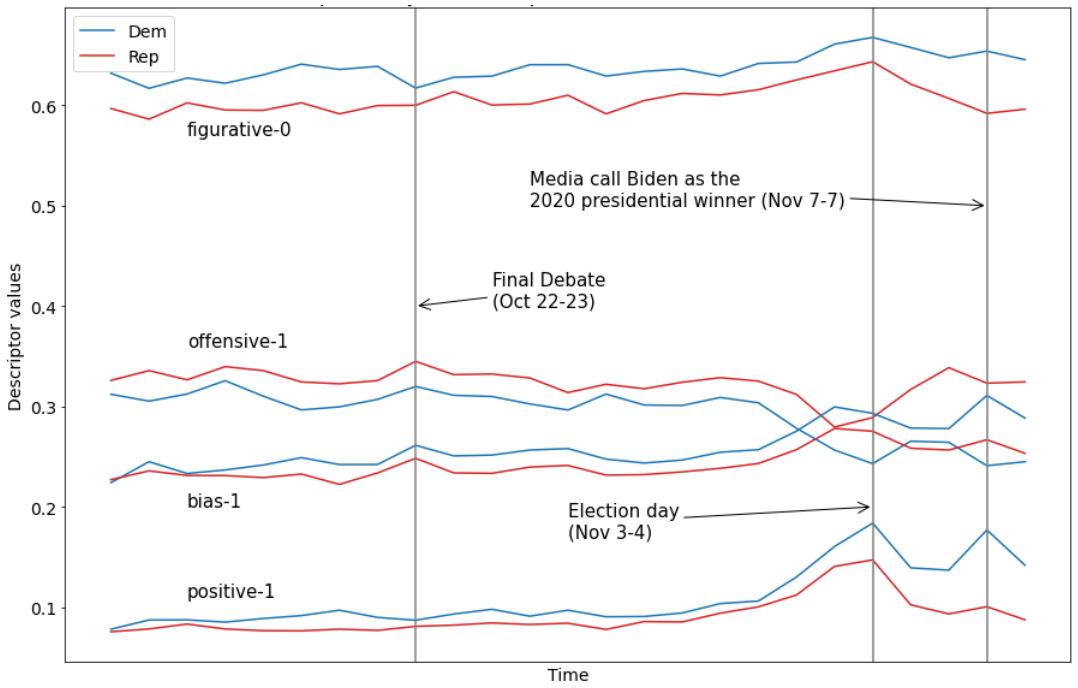

FIGURE 3.5: Per-channel day-by-day values of the 4D timeseries constructed from the 2020 data using the proposed descriptor and the mean aggregation strategy, separately for Democrats (Dems) and Republicans (Reps). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

2016-10-07 seem to have affected more the latter candidate, as the majority of posts expressed an opinion about him. Equal traffic is observed for both parties just before the election day (2016-11-08), since in that stage Twitter plays a significant role in the campaign of both candidates. However, the number of tweets concerning Democrats drops significantly after the election day and their defeat. In contrast, people kept tweeting frequently about the winner and center of attention Donald Trump regarding his actions as a president of the US, like the travel ban, Iran sanctions, etc.

Figure 3.3 depicts the number of tweets posted from 2020-10-15 to 2020-11-08, separately for the Democrats and the Republicans. Again, increased Twitter traffic is observed for both parties during the most important events. However less events are observed in the 2020 plot, which is due to the smaller size of the dataset as a whole, in comparison compared to the 2016 dataset. In general, there were more tweets posted about Trump up until November 7. On that day, the media called Biden as the 2020 presidential winner and Twitter traffic exploded for Biden, surpassing Trump posts by a significant margin. This obviously makes sense as Biden's victory is officially announced for the first time.

Having established original Twitter traffic patterns, concurrent daily values of the 4D timeseries constructed using the proposed descriptor and the mean aggregation strategy are depicted in Figure 3.4, separately for each party of the 2016 US election (party affiliation is color-coded). In this Figure, as in many following ones, we label each timeseries by one of the two

opposite class labels (e.g., "positive" or "figurative") followed by the tweet classifier output value which implies this label (e.g., a tweet fully and undoubtedly classified as figurative/positive, has been assigned a value of 0/1 by the figurativeness/polarity classifier, respectively). It is evident that the timeseries maintain a stable class along all four dimensions and across the entire time range for both parties: their class is negative (in polarity), unbiased, non-offensive and literal. This indicates a general public stance towards the competing politicians, which reflects a judgemental and indignant (negative + literal) but simultaneously educated (unbiased + non-offensive) public.

Figure 3.5 depicts the mean daily 4D descriptor for each party of the 2020 US election. Again, the timeseries maintain the same stable classes with the 2016 data along all four dimensions and across the entire time range for both parties. An interesting conclusion that can be drawn by comparing the plots of the 2016 and the 2020 elections is that the public seems to have a rather fixed stance towards the competing parties, carried over from one election period to the next one, with the winner determined by a small margin/difference.

Moreover, by visually inspecting Figures 3.2, 3.4 and 3.3, 3.5 for the 2016 and 2020 datasets, respectively, a correlation can be identified between the occurrence of crucial events and abrupt changes (spikes) in the number of tweets or public opinion. As expected, this reaction in Twitter takes place the day after the event.

By comparing the timeseries of the two parties in figure 3.4, one can observe that tweets about Republicans are less negative, less unbiased and less literal, while there is no clearly distinguishable difference between the two parties concerning offensiveness. These observations shed new light to the election results of November 8th. A less negative opinion is clearly an advantage in itself for Republicans, but combining it with a more biased opinion reflects the possibility that there were more Trump's partisans active in Twitter. Given that partisans are decided voters that do not easily change their opinion, these conclusions drawn from analysing the constructed timeseries paint the picture of a significant Republican advantage, by only using public Twitter data.

Interestingly, posts about Trump appear to be less literal. Despite common perceptions that figurative language is most often used to express negative opinions, tweets about Republicans are on average less negative. A possible explanation is that figurativeness doesn't reflect on the voters' decision and is mainly being used by Twitter users to attract higher attention. Therefore, our analysis indicates that if a voter is clearly against a candidate, it is more probable for them to be straightforward in their comments.

Corresponding conclusions can be drawn from Figure 3.5 regarding the 2020 US elections. One can observe that the tweets about Democrats are less negative, less unbiased, more literal and less offensive. The less negative and less unbiased attributes can be interpreted as beneficial

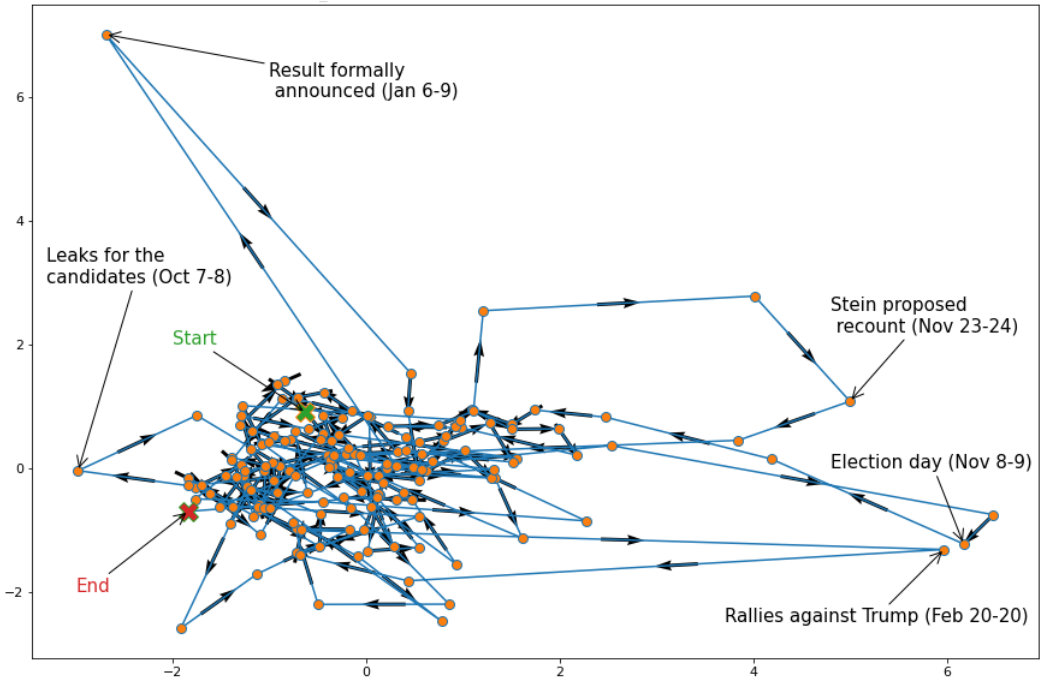

FIGURE 3.6: PCA-based 2D visualization of the constructed 4D timeseries for the Democrats, using a mean aggregation strategy, across the entire 2016 dataset time range (163 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

factors for the Democratic party (like in 2016). The main difference in the 2020 data are the less offensive and more literal tweets that seemingly further contribute to Democratic dominance, as figurative language usually implies negativity [128].

Principal Component Analysis (PCA) was exploited for applying dimensionality reduction to the 4D mean Republican/Democrat timeseries, so that they can be visualized in 2D plots. The 2D descriptor points per party are presented in Figures 3.6, 3.7 for the 2016 dataset and 3.8, 3.9 for the 2020 dataset. Here, outlying data points correspond to the spikes of the original time-domain plots of Figures 3.2, 3.4 and 3.3, 3.5. Thus, outliers in PCA Figures indicate the occurrence of crucial events. This visualization can help us identify incidents that significantly affect public opinion, but does not immediately provide us with corresponding information about the semantic descriptor values.

However, the following observations can be made based on the outliers. Regarding the 2016 data we can tell from Figures 3.6, 3.7 that the events influencing public opinion about the Democrats the most were the elections themselves, the formal announcement of the results and the leaks about the candidates. The respective events for the Republicans are identical, with the addition of the second candidate debate and the compliments made by Donald Trump on President Putin. This possibly reflects the increased relevance of national security concerns in the US public discourse. Regarding the 2020 data, Figures 3.8, 3.9 confirm the three crucial events indicated by the spikes in Figures 3.3, 3.5.

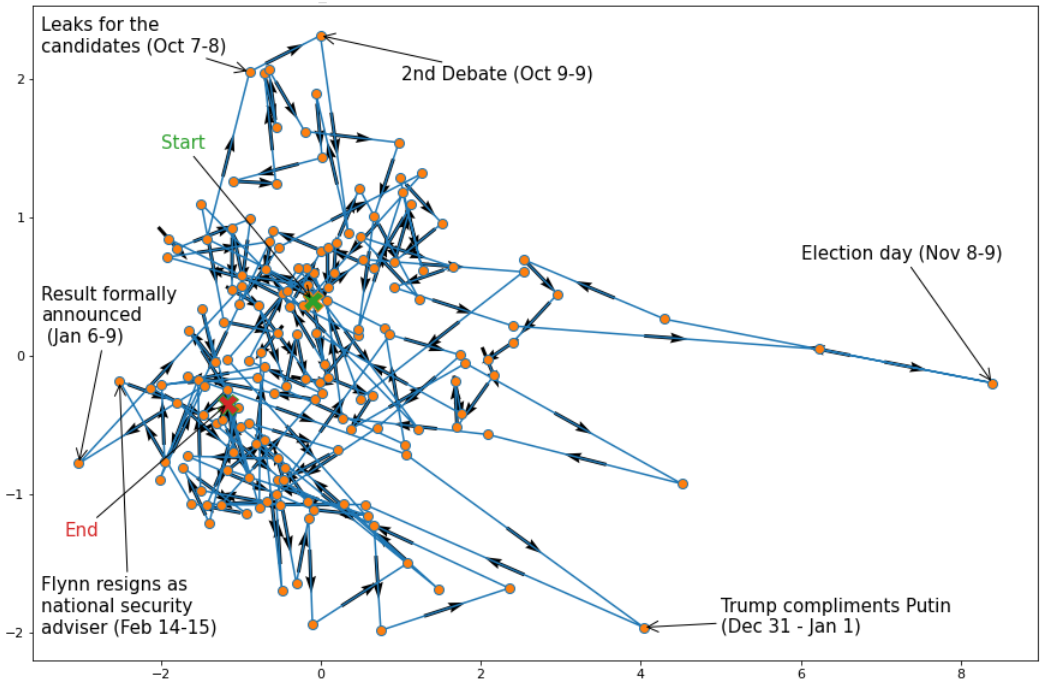

FIGURE 3.7: PCA-based 2D visualization of the constructed 4D timeseries for the Republicans, using a mean aggregation strategy, across the entire 2016 dataset time range (163 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

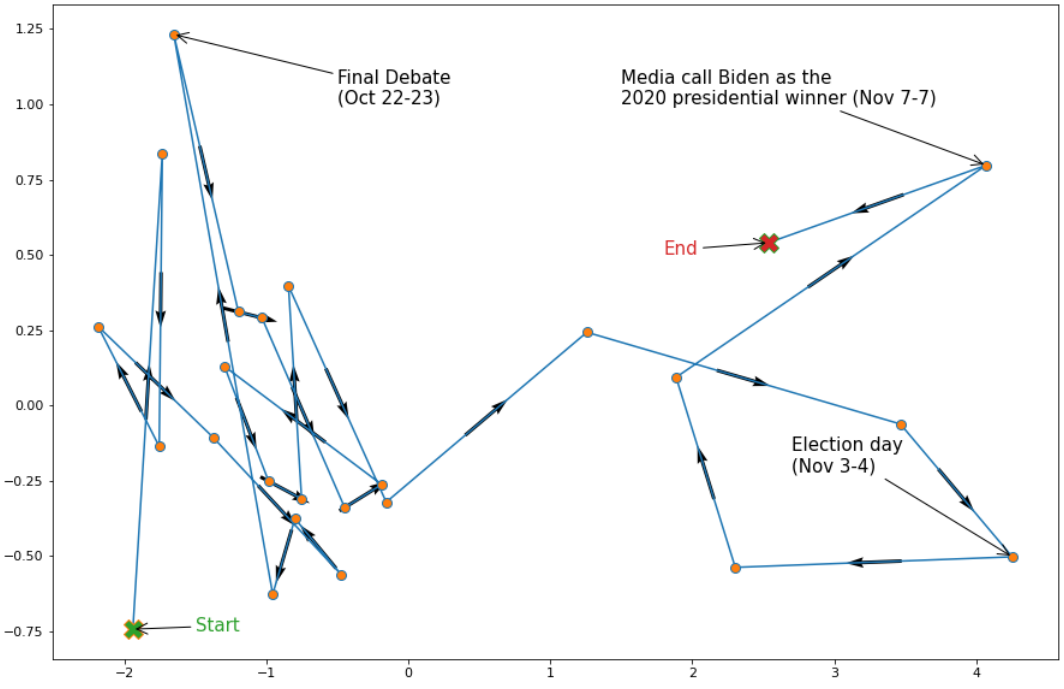

FIGURE 3.8: PCA-based 2D visualization of the constructed 4D timeseries for the Democrats, using a mean aggregation strategy, across the entire 2020 dataset time range (25 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

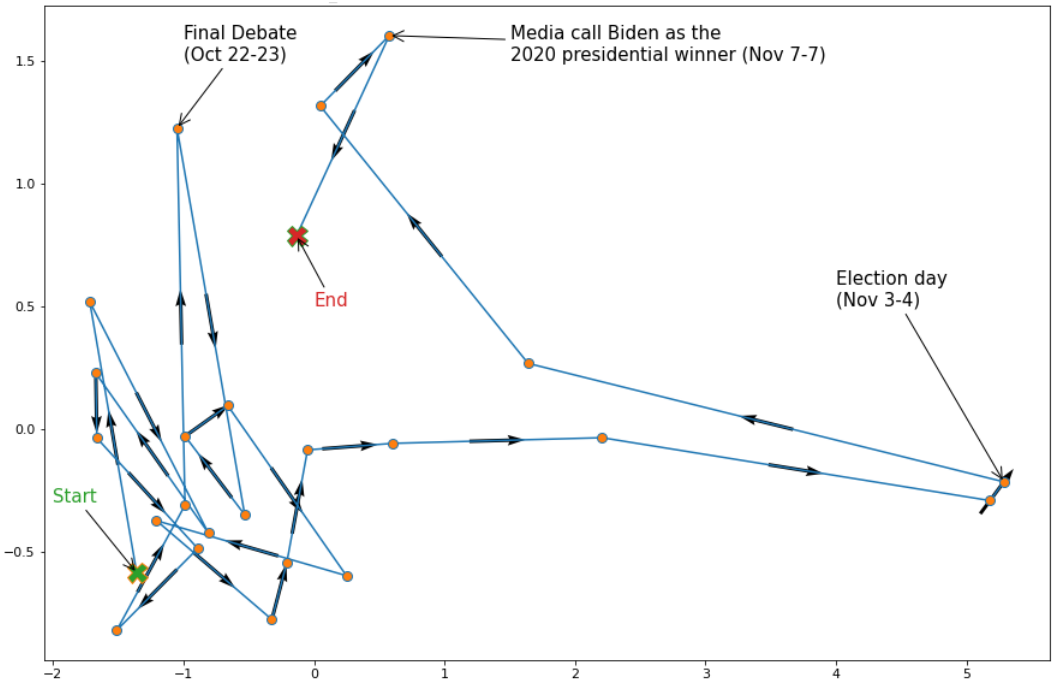

FIGURE 3.9: PCA-based 2D visualization of the constructed 4D timeseries for the Republicans, using a mean aggregation strategy, across the entire 2020 dataset time range (25 days). The two dates given per event are the date of that event (first) and the date of the respective reaction in Twitter (second).

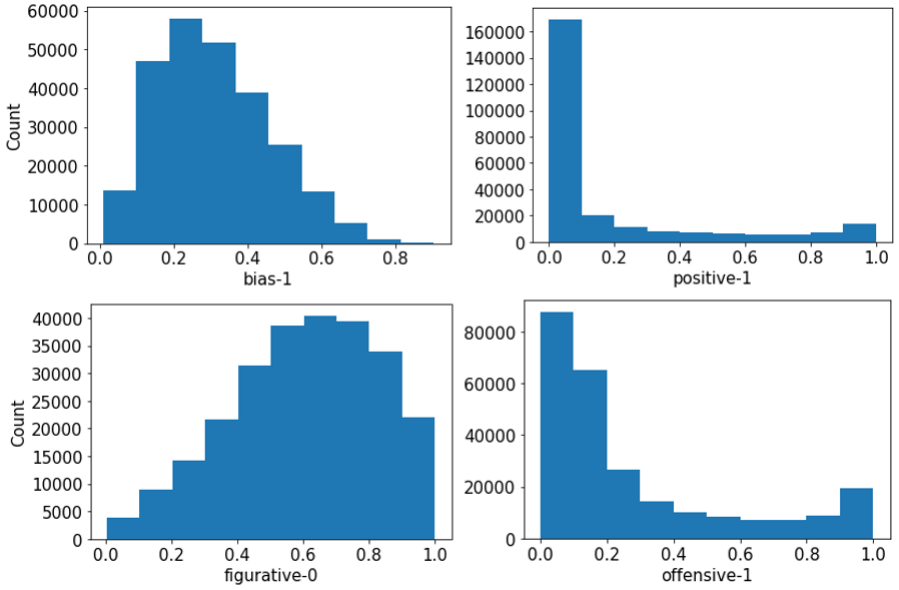

FIGURE 3.10: Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Democrats on Nov 9, 2016 (the day after election).

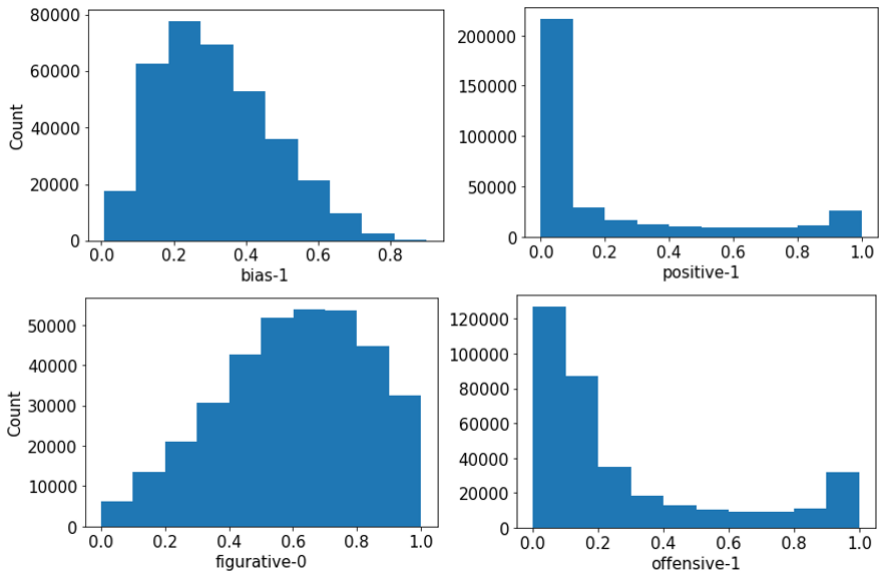

FIGURE 3.11: Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Republicans on Nov 9, 2016 (the day after election).

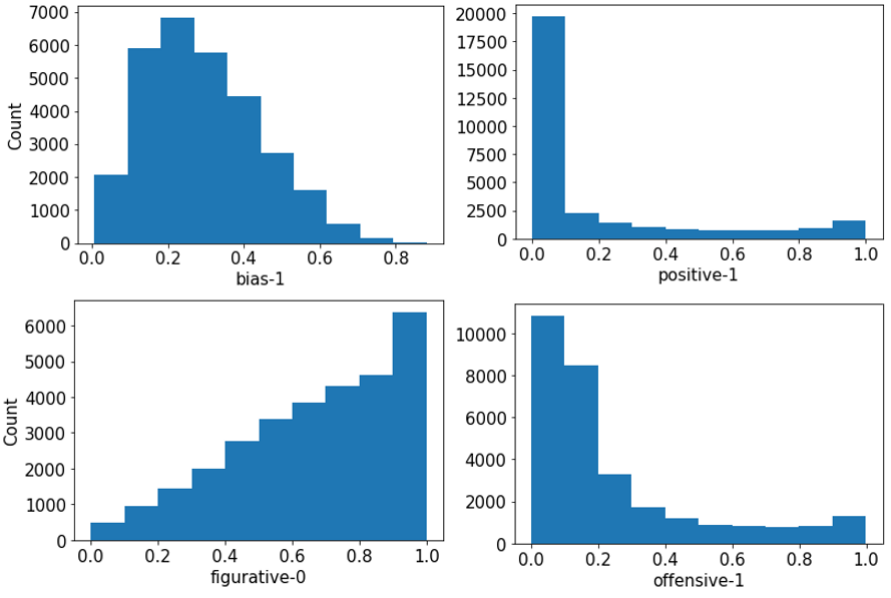

FIGURE 3.12: Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Democrats on Nov 4, 2020 (the day after the election).

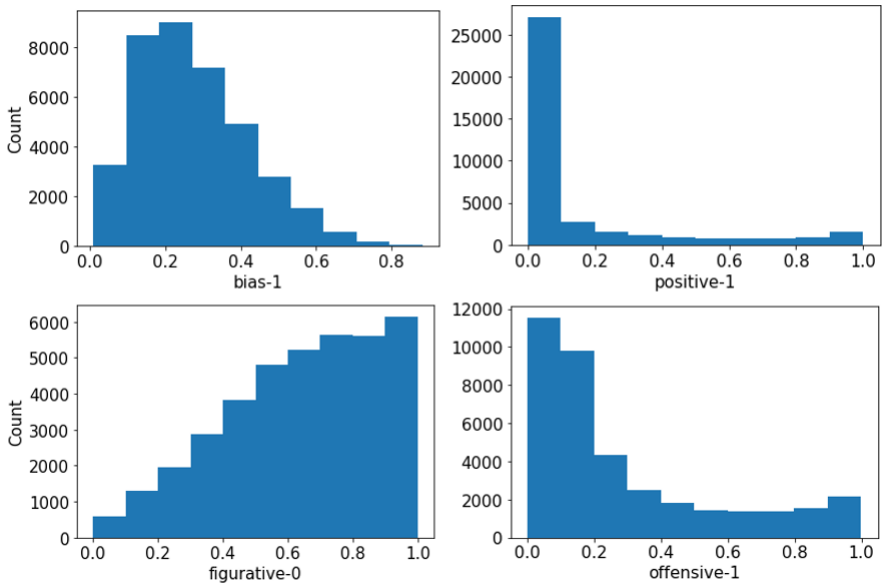

FIGURE 3.13: Histograms of the four descriptor dimensions, depicting how the number of tweets is distributed over the DNN classifier outputs. These histograms concern the Republicans on Nov 4, 2020 (the day after the election).

Given the explanatory power of specific dates shown to be semantic outliers in the constructed timeseries, a different way to exploit the proposed public opinion description mechanism was also investigated: to focus on individual salient dates. In the context of this study and given the previously discussed observations, the day after the elections (November 9, 2016 and November 4, 2020) was selected as the target date.

Figures 3.10 and 3.11 show how the tweets posted on November 9, 2016 were distributed along the four descriptor dimensions, separately for the two parties and before any aggregation strategy was applied. These 10-bin histograms show the distribution of the number of tweets (vertical axis) over the semantic values of each of the four descriptor dimensions (horizontal axis). The employed semantic values (outputted by the 4 pretrained DNN classifiers) were real numbers in the interval [0,1]. Therefore, the horizontal axis has not been normalized in range: each of the 10 bins corresponds to a subrange of length 0.1.

The histograms are almost identical for Democrats and Republicans, an observation compatible with the behaviour captured in Figure 3.4. Moreover, similar histogram shapes can be discerned for the bias-figurativeness and for the polarity-offensiveness features. Bias and figurativeness have approximately shifted normal distributions, implying that the mean aggregation strategy is indeed a good choice during timeseries construction. In contrast, polarity and offensiveness histograms are significantly more polarized in shape, rendering the mean aggregation strategy less reliable in their case for that particular date.

The fact that the majority of tweets fall within the interval [0,0.2] for the polarity and offensiveness dimensions, means that they have been clearly classified as rather negative and

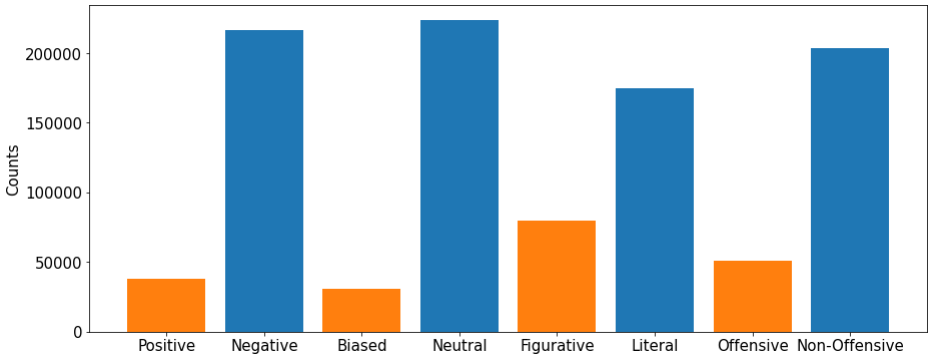

FIGURE 3.14: Number of tweets concerning Democrats, separately for each class of the four descriptor dimensions, on November 9, 2016. The two colors distinguish between the opposite classes of each semantic dimension.

non-offensive: a fully negative/positive and absolutely non-offensive/offensive tweet would be characterized by a value of 0/1 in both dimensions, respectively. These histograms paint the picture of a public that is carping and complaining in the aftermath of the elections, yet avoids the use of offensive language. Concerning the relative absence of intermediate values lying within the range [0.2,0.8], we can say that classification regarding polarity and offensiveness was straightforward and the respective models pretty confident. This implies that indeed most tweets were clearly negative or positive, as well as clearly non-offensive or offensive, without many users being neutral in these respects. In contrast, classification regarding bias and figurative attributes does not lead to such polarized results. This is because the DNN models have trouble classifying these tweets as pure instances of a specific class (e.g., the "figurative" or the "literal" class), leading to intermediate values near 0.5. This implies that most users were rather neutral with regard to these semantic dimensions. Still, we can clearly see that the majority of tweets tend to be non-biased and literal.

Similar histogram shapes are observed for the respective tweet distributions of November 4, 2020 for both Democrats 3.12 and Republicans 3.13. This was no surprise given the similarity of Figures 3.4 and 3.5. However, there is a noticeable difference in the figurative elements of 2016 and 2020, where the distribution is shifted right, indicating more literal language used on that day's tweets. This can be confirmed by comparing Figures 3.4 and 3.5.

Finally, Figures 3.14, 3.15 and 3.16, 3.17 depict the number of tweets per party, separately for each class of the four descriptor dimensions, on the election days November 9, 2016 and November 4, 2020. This visualization provides a glimpse to the non-dominant classes that disappeared when constructing the timeseries using the mean aggregation strategy.

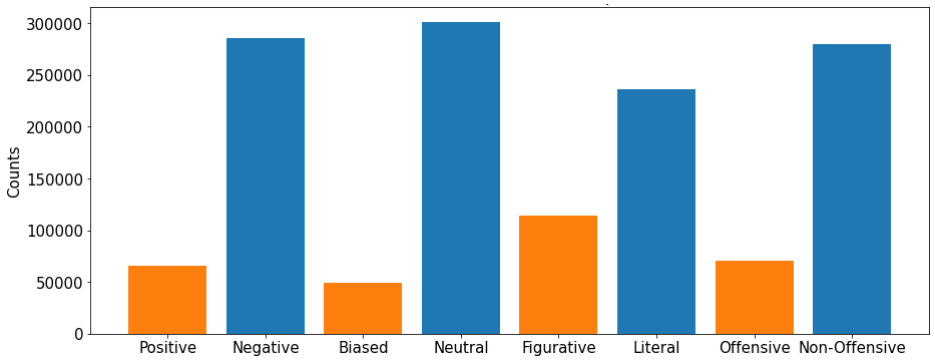

FIGURE 3.15: Number of tweets concerning Republicans, separately for each class of the four descriptor dimensions, on November 9, 2016. The two colors distinguish between the opposite classes of each semantic dimension.

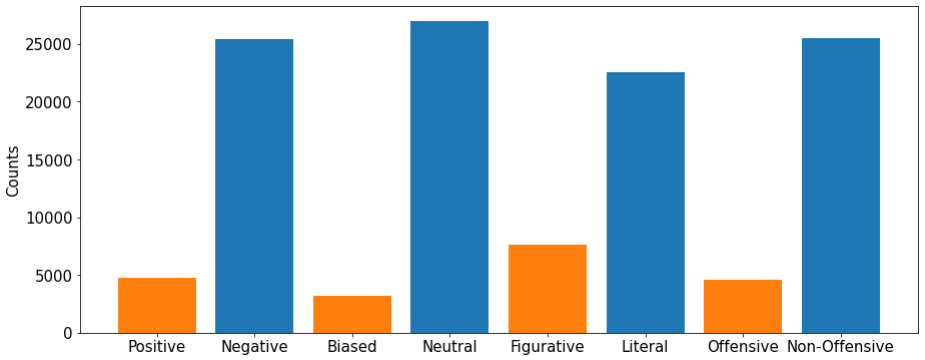

FIGURE 3.16: Number of tweets concerning Democrats, separately for each class of the four descriptor dimensions, on November 4, 2020. The two colors distinguish between the opposite classes of each semantic dimension.

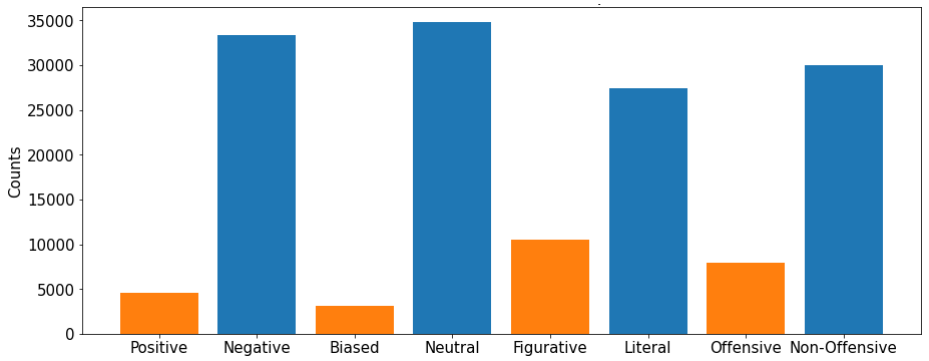

FIGURE 3.17: Number of tweets concerning Republicans, separately for each class of the four descriptor dimensions, on November 4, 2020. The two colors distinguish between the opposite classes of each semantic dimension.

3.6 Discussion

The evaluation presented in Section 3.5 indicates that the proposed mechanism for automated public opinion monitoring through Twitter is a very powerful tool, able to provide valuable information for more efficient decision-making. The **multidimensional nature** of the presented descriptor conveys rich insights (analyzed in Section 3.5) that are not typically captured by existing relevant methods, which only exploit sentiment (and, rarely, also bias). This is shown through the qualitative insights extracted in Subsection 3.5.3. Moreover, unlike the vast majority of previously published methods, the proposed mechanism relies on state-of-the-art **DNN-based NLP tools**, a fact which guarantees enhanced accuracy in comparison to existing comparable approaches.

To succinctly demonstrate the usefulness of the proposed mechanism in political analysis, the most important findings extracted by applying it to the datasets of Section 3.5 (concerning the US presidential elections of 2016 and 2020) are summarized below:

- Public opinion concerning Democrats was less stable and less predictable, in comparison to public opinion about Republicans.
- However, in general, the public has an overall relatively stable stance towards the competing politicians: judgmental and indignant (negative + literal) but simultaneously educated (unbiased + non-offensive).
- The timeseries derived through the proposed mechanism paint a rather accurate picture of the favored candidate. This is shown through visualizing/inspecting the timeseries.
- The winning party is referenced during the pre-election period in tweets that are jointly less negative + less offensive + more biased. Strong partisan presence in Twitter seems to be heavily correlated with high vote percentages.
- Crucial events do directly lead to abrupt changes in the daily number of tweets (which is to be expected), but also in public opinion. This indicates that committed/stable partisan supporters are always a minority in the American Twitter. Moreover, post hoc visualizations of the timeseries automatically derived through the proposed mechanism can actually showcase which events were the most crucial to shifts in public opinion.

A few of these findings verify similar conclusions previously drawn in the existing literature: [108] (the public has an overall relatively stable, negative stance towards the competing politicians, during a specific pre-election period, while the public sentiment timeseries paint a rather accurate picture of the favored candidate) and [101] [106] [108] (crucial events directly lead to abrupt changes in public opinion). However, the majority of our findings for the US presidential elections of 2016 and 2020, as detailed in Section 3.5, are original contributions of this study.

Most importantly though, the proposed mechanism is not tied to these specific elections. *It is a fully generic and almost fully automated method*, that allows interested users to easily extract similar insights *for any time period*.

3.7 Conclusions

Automated public opinion monitoring using social media is a very powerful tool, able to provide interested parties with valuable insights for more fruitful decision-making. Twitter has gained significant attention in this respect, since people use it to express their views and politicians use it to reach their voters.

This study presented a novel, automated public opinion monitoring mechanism, consisting of a composite, quantitative, semantic descriptor that relies on NLP algorithms. A four-dimensional vector, i.e., an instance of the proposed descriptor, is first extracted for each tweet independently, quantifying text polarity, offensiveness, bias and figurativeness. Subsequently, the computed descriptors are summarized across multiple tweets, according to a desired aggregation strategy (e.g., arithmetic mean) and aggregation target (e.g., a specific time period). This can be exploited in various ways; for example, aggregating the tweets of each days separately allows us to construct a multivariate timeseries which can be used to train a forecasting AI algorithm, for day-by-day public opinion predictions.

In order to evaluate the usefulness of the proposed mechanism, it was applied to the large-scale 2016/2020 US Presidential Elections tweet datasets. The resulting succinct public opinion descriptions were successfully employed to train a DNN-based public opinion forecasting model with a 7-day forecasting horizon. Moreover, the constructed timeseries were thoroughly inspected in a qualitative manner in order to deduce insights about public opinion during a heated pre/post-election period.

Data availability

The datasets generated during the current study are available from the corresponding author on reasonable request. The raw 2016 US Presidential Elections tweet dataset is publicly available at `https://www.kaggle.com/paulrohan2020/2016-usa-presidential-election-tw` and the one for the 2020 USA elections at `https://www.kaggle.com/datasets/manchunhui/us-election-2020-tweets`.

Chapter 4

Neural Knowledge Transfer for Improved Sentiment Analysis in Texts with Figurative Language

4.1 Abstract

Sentiment analysis in texts, also known as opinion mining, is a significant Natural Language Processing (NLP) task, with many applications in automated social media monitoring, customer feedback processing, e-mail scanning, etc. Despite recent progress due to advances in Deep Neural Networks (DNNs), texts containing figurative language (e.g., sarcasm, irony, metaphors) still pose a challenge to existing methods due to the semantic ambiguities they entail. In this study, a novel setup of neural knowledge transfer is proposed for DNN-based sentiment analysis of figurative texts. It is employed for distilling knowledge from a pretrained binary recognizer of figurative language into a multiclass sentiment classifier, while the latter is being trained under a multitask setting. Thus, hints about figurativeness implicitly help resolve semantic ambiguities. Evaluation on a relevant public dataset indicates that the proposed method leads to state-of-the-art accuracy, surpassing all competing approaches.

4.2 Introduction

Recent progress due to advances in Deep Neural Networks (DNNs) has augmented tremendously the performance of state-of-the-art sentiment analysis methods. Current approaches rely on DNN architectures, such as Convolutional Neural Networks (CNNs) and/or Long Short-Term Memory (LSTM) variants, that are trained for classification or regression under a supervised learning setting. Under this framework, handcrafted input textual features have largely

been replaced by DNN-derived learnt word embeddings [129], which manage to capture significant semantic properties.

However, texts containing figurative language (e.g., involving sarcasm, irony, metaphors, etc.) still pose a challenge to existing methods [130]. This is due to difficulties in identifying whether a figurative phrase actually implies negative or positive opinion, given its inherent semantic ambiguity. The issue is especially pronounced because of the ubiquitous nature of figurative language (FL) in human communication. For example, sarcasm and irony are known to attract higher attention in on-line communities [131], with many people being adept at their usage and exploiting them to increase the impact of their social media posts (e.g., on Twitter).

In this study, a novel setup of knowledge distillation [132] is proposed for DNN-based sentiment analysis of figurative texts [130]. It is employed for transferring knowledge from a binary pretrained FL recognizer (teacher) into a multiclass sentiment classifier (student), while the latter is being trained under a multitask setting. Up to now, *knowledge distillation has mainly been applied in softmax-based classification settings, where the teacher and the student solve identical tasks.* This study, instead, proposes *distillation from a binary classification teacher with sigmoidal output, in order to aid a multiclass classification student.* To the best of our knowledge, it is the first time this approach is being employed for a real-world problem using regular DNNs. Moreover, it is the first time any form of neural distillation is exploited for increasing sentiment analysis accuracy in texts with FL. Evaluation on a relevant public dataset indicates that the proposed method leads to state-of-the-art performance, surpassing all competing approaches.

4.3 Related Work

This Section presents the existing state-of-the-art concerning: a) sentiment analysis in texts with figurative language (FL), and b) knowledge distillation.

4.3.1 Sentiment Analysis on Figurative Language

Sentiment analysis on FL was originally proposed at the Semantic Evaluation Workshop 2015 - Task 11 [130]. Given a set of tweets that are rich in metaphor, sarcasm and irony, the goal was to determine whether a user has expressed a positive, negative or neutral sentiment in each one of them. The participants were provided with both integer and real-valued labels, thus allowing the use of either classification or regression approaches. All published studies related to sentiment analysis on FL are evaluated on the S15-T11 tweet collection, due to the lack of other public relevant datasets.

Initial methods tackling the task relied on Support Vector Machines (SVMs), decision trees or regression learning models, operating on handcrafted features (e.g., computed on n-grams or

tf–idf statistics) and lexicons (e.g., SentiWordnet, Depeche Mood, American National Corpus, etc.) [133] [134]. Later, [135] approached the task as a regression problem and exploiting a CNN architecture. Unigrams with more than two occurrences were employed as input textual features, while hashtags designating the FL category types were replaced by binary indicators. DESC [136] integrated a Bi-LSTM, an Attention LSTM and an MLP, with the first two being fed pretrained GloVe word embeddings [129] and the MLP being fed handcrafted tweet features, such as unigrams, bigrams and tf-idf statistics. DESC was extended in [137] by utilizing contextual input embeddings from a pretrained RoBERTa model [138], which are fed to a Recurrent Convolutional Neural Network (RCNN). The task was formulated as a classification problem.

4.3.2 Neural Knowledge Transfer

Knowledge distillation is the most prevalent form of neural knowledge transfer. It implies training a student DNN according to the response of a pretrained teacher DNN to input samples (so-called "transfer dataset"), possibly consisting of unknown/unlabeled data. Typically, the student is assumed to be initially untrained (encoding no prior knowledge), while the two networks may have different neural architectures [132]. The student DNN is trained with an objective function that penalizes the deviation of its output from the corresponding output of the fixed teacher DNN on the transfer set [132]. The goal is to transfer so-called *dark knowledge* from the teacher to the student.

In classification tasks, abnormally high softmax temperatures are typically employed for extracting both outputs while training the student, in order to obtain soft distributions over the class labels that implicitly encode similarities among data samples. After training, the student is deployed with a more conventional softmax temperature. If labels are available for supervising training on the transfer set, a regular classification term may also be added to the objective. In this case, the distillation term mainly has a regularizing effect.

Several variants of this base neural distillation scheme have surfaced over the years. For instance, in case the transfer set is known/labeled, the parameters/synaptic weights of the trained teacher may be employed for initializing the student smartly before proceeding with regular supervised training [139]. Hints from hidden layer activations in the teacher when processing the transfer set may also be employed for guiding student training, besides the final output [140]. Dimensionality reduction may be needed when the layer dimensions differ between student and teacher. To bypass this limitation, student training in [141] is guided by the similarities between transfer set data samples representations constructed by the teacher. Intermediate "teacher assistant" networks, in the context of a multi-step process, was also proposed in [142] for scenarios where the two models differ in complexity. Finally, distillation from a deep linear teacher for binary classification was presented in the context of a purely theoretical analysis

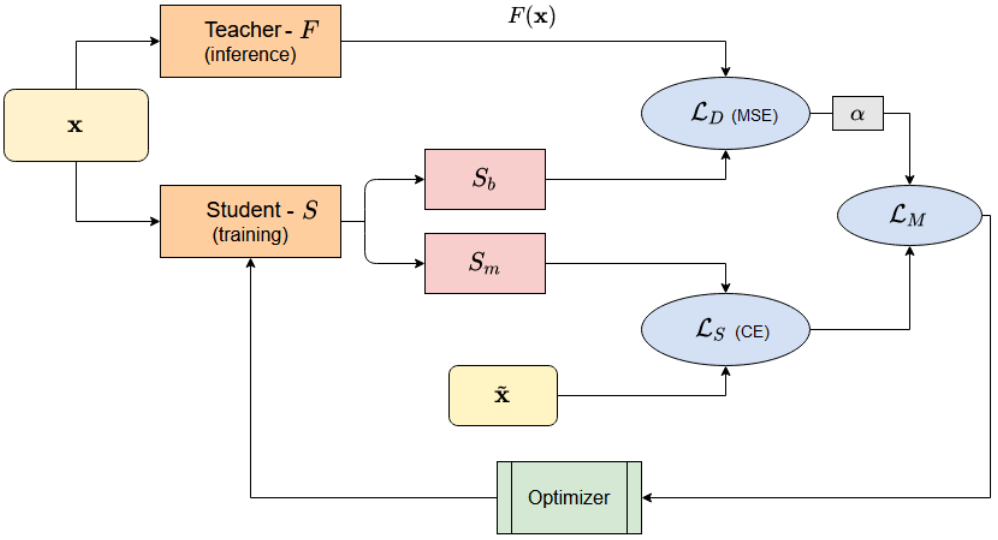

FIGURE 4.1: The proposed teacher-student training architecture.

in [143], while a similar investigation was conducted in [144] for shallow feed-forward neural networks.

4.4 Proposed method

The proposed method exploits knowledge distillation [132] for increasing DNN-based sentiment analysis accuracy on texts with FL, that employ sarcasm, irony and/or metaphor. Thus, during training, a teacher-student architecture is utilized to enrich the student model with the knowledge of a pretrained FL recognizer. The latter one is a binary classifier, while the former one is a multiclass classifier tasked with identifying sentiment in input texts. Thus, due to the nature of the teacher and the different task it solves compared to the student, an atypical kind of distillation is proposed that has not been previously employed in regular DNNs for real-world tasks.

Below, all neural models are assumed to be trained with error back-propagation and a variant of stochastic gradient descent. Let us also assume that a DNN-based binary text classifier F has been pretrained under a regular supervised setting on a database containing two classes: "figurative", "literal/non-figurative". Since it is common for binary neural classifiers to end with a single sigmoidal neuron, we assume this is the case for F. Thus, a real-valued scalar output of $0/1$ corresponds to figurative/literal class prediction, respectively, while a typical output $F(\mathbf{x})$ for a respective input data point $\mathbf{x}$ would actually lie in the interval $[0, 1]$.

The student S is the neural model we actually want to optimize; on a different, sentiment-annotated dataset. Without loss of generality, we assume that it is being trained under a supervised multiclass text classification setting. Typically, $N \geq 3$ classes are employed for the

sentiment analysis/opinion mining task ("positive", "neutral", "negative", etc.) and a final softmax activation layer used for deriving the class prediction. S is trained by a regular, suitable loss function $\mathcal{L}_S$, such as Cross-Entropy (CE).

The proposed method consists in training S with the following multitask loss function:

$$\mathcal{L}_M = \mathcal{L}_S + \alpha \mathcal{L}_D, \tag{4.1}$$

where $\mathcal{L}_D$ is being computed at each iteration by exploiting the pretrained F. In essence, $\mathcal{L}_D$ distills the knowledge of F concerning the current training data point $\mathbf{x}$, i.e., $F(\mathbf{x})$. As noted in [143] for the deep linear scenario, sigmoidal output activation for the binary classification case is equivalent to *soft labels* typically employed for softmax-based multiclass distillation [132]. To compute this loss term, a parallel output layer S_b serving as an auxiliary binary classification head is architecturally plugged onto the penultimate layer of S, while $F(\mathbf{x})$ serves as real-valued/continuous substitute "ground-truth" for $\mathcal{L}_D$. To avoid confusion, the normal softmax-based multiclass classification head of S is denoted below by S_m. Thus, assuming N sentiment classes, S_m/S_b is an output neural layer consisting of $N/1$ neuron(s), respectively.

By employing the Mean Squared Error cost (MSE) for $\mathcal{L}_D$, the proposed complete multitask loss function is:

$$\mathcal{L}_M = \mathcal{L}_S \left(S_m(\mathbf{x}), \tilde{\mathbf{x}}\right) + \alpha \left(S_b(\mathbf{x}) - F(\mathbf{x})\right)^2, \tag{4.2}$$

where $\tilde{\mathbf{x}}$ is the actual, *sentiment* ground-truth class label corresponding to $\mathbf{x}$, in the context of multiclass classification. Notably, no *figurativeness* ground-truth label is exploited or required to exist for $\mathbf{x}$.

An overview of the proposed method is depicted in Figure 4.1. Importantly, no actual/real ground-truth annotation concerning the presence or type of FL is required or exploited while training S for sentiment analysis. Of course, after S has been fully trained, both the entire F model and the auxiliary output layer/binary classification head S_b can be safely discarded.

The underlying intuition behind the proposed multitask loss function is the conjecture that dark knowledge concerning the degree of figurativeness of an input text should aid a sentiment classifier in resolving ambiguities about the expressed sentiment, that arise due to sarcastic, metaphorical or ironical language. The proposed distillation loss term should have the effect of tuning the multiclass sentiment classifier towards identifying and overcoming such ambiguities. The FL recognizer F was selected to be a binary classifier in order to maximize its inference-stage success rate in this auxiliary task, by making the classification problem as easy as possible. Notably, knowledge distillation from binary classifiers with sigmoidal output, in order to aid a multiclass classifier on a different task, has not been previously presented for regular DNNs.

4.5 Quantitative Evaluation

4.5.1 Implementation Details

The neural architecture ROB-RCNN from [137] was recreated and adopted for the base sentiment analysis student model S. This neural architecture utilizes a pretrained RoBERTa language model [138], combined with an RCNN [145], in order to efficiently capture contextual text information when representing each word. The final prediction is the output of a softmax layer. The reason behind choosing ROB-RCNN as a baseline was solely practical; *in principle, the proposed method can be used to augment any other sentiment classifier, as well.*

Table 4.1: Evaluation results on the S15-T11 dataset. Higher/lower is better for the COS/MSE metric, respectively. Best results are in bold.

Method	***COS***	***MSE***
ELMo [146]	0.71	3.61
USE [147]	0.71	3.17
NBSVM [148]	0.69	3.23
FastText [149]	0.72	2.99
XLnet [150]	0.76	1.84
BERT-Cased [151]	0.72	1.97
BERT-Uncased [151]	0.79	1.54
RoBERTa [138]	0.78	1.55
UPF [133]	0.71	2.46
ClaC [152]	0.76	2.12
DESC [136]	0.82	2.48
ROB-RCNN [137]	0.82	1.92
ROB-RCNN + Proposed ($\mathcal{L}_{\mathcal{D}}$)	**0.85**	**1.50**

The CNN/Bi-LSTM neural architecture OSLCfit [153] was pretrained for FL recognition, following the training process prescribed in [153], and then adopted as the binary classification teacher model F. Its input text representations are derived by using 200-dimensional embeddings from a pretrained GloVe model [129].

This teacher model was pretrained on the annotated dataset from [154], which contains 81,4K tweets grouped under 4 different class labels ("sarcasm", "irony", "figurative" and "regular"). The first three classes were combined in a general "figurative" class, in order to train F as a binary figurative text classifier.

The student S was trained using Adam optimization and Cross Entropy (CE) as the main multiclass classification student loss function $\mathcal{L}_S$.

4.5.2 Evaluation Setup

The S15-T11 dataset [130] was used for evaluating the proposed method and comparing it against competing approaches. It contains 8000/4000 tweets for training/test, respectively, including tweets with ironic, sarcastic and metaphorical language. The 12000 data points are grouped under 11 classes annotated with integers in an 11-point scale, ranging from -5 to +5, that denote the polarity of each tweet, from "very negative" to "very positive". Since it is a sentiment analysis dataset, it *does not* contain ground-truth annotations/labels concerning the presence or type of FL.

Two evaluation metrics were employed: cosine similarity (COS, higher is better) and Mean Squared Error (MSE, lower is better). Assuming a test set of T data points, both are computed by comparing two T-dimensional integer vectors, respectively containing the predicted and the ground-truth class labels. The proposed method implementation is in fact ROB-RCNN [137] augmented with $\mathcal{L}_{\mathcal{D}}$ during training, while the baseline that we improve upon is ROB-RCNN trained with simple $\mathcal{L}_S$, instead of the proposed Eq. (4.2). Optimal hyperparameters were adopted from [137], while 5-fold cross-validation resulted in best $\alpha = 0.5$. Test-phase evaluation results are presented in Table 4.1, including the accuracy achieved by several competing methods. All reported figures are lifted from [137], except the ones for ROB-RCNN. The latter method was recreated, trained and evaluated ab initio by us, following strictly all implementation minutiae and hyperparameter values detailed in [137].

Overall, the proposed method implementation ROB-RCNN + $\mathcal{L}_{\mathcal{D}}$ achieves state-of-the-art performance in both metrics, thus confirming the validity of our underlying intuition. Remarkably, compared to DESC, it manages to decrease MSE from 2.48 to 1.50, while simultaneously increasing COS from 0.82 to 0.85. In contrast, baseline ROB-RCNN achieves MSE improvements over DESC, without gains in COS performance.

4.6 Conclusions

Natural Language Processing tasks, such as sentiment analysis in texts, have significantly progressed thanks to advances in Deep Neural Networks. However, any presence of figurative language (sarcasm, metaphor, irony) significantly increases the difficulty of the sentiment analysis task. This study exploits the intuition that estimations about the existence of figurative language in an input text can boost the accuracy of a sentiment classifier, by helping it to internally resolve semantic ambiguities. Thus, the proposed method consists in a novel setup of knowledge distillation from a pretrained binary recognizer of figurative language, employed as an auxilliary task while training a multiclass sentiment analysis neural model under a multitask setting. Notably, no ground-truth annotation about figurativeness is required to exist

while training for the sentiment analysis task. Evaluation on a relevant public dataset indicates that the proposed method leads to state-of-the-art performance, surpassing all competing approaches.

Bibliography

[1] Jason Brownlee. What is natural language processing?, 2017. URL https://machinelearningmastery.com/natural-language-processing/. Accessed June 23, 2022. 1.1.1

[2] Wikipedia contributors. Linguistics — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Linguistics&oldid=1094492783. [Online; accessed 24-June-2022]. 1.1.4.1

[3] Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009. 1.2.1

[4] Wikipedia contributors. Natural language processing — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Naturallanguageprocessing&oldid=1093868703. [Online; accessed 23-June-2022]. 1.2.2.1, 1.2.2.2, 1.2.2.2, 1.5

[5] W John Hutchins. The georgetown-ibm experiment demonstrated in january 1954. In *Conference of the Association for Machine Translation in the Americas*, pages 102–114. Springer, 2004. 1.2.2.1

[6] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966. 1.2.2.1

[7] Wikipedia contributors. Parry — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=PARRY&oldid=1023645514. [Online; accessed 23-June-2022]. 1.2.2.1

[8] Wikipedia contributors. Head-driven phrase structure grammar — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Head-drivenphrasestructuregrammar&oldid=1080201954. [Online; accessed 23-June-2022]. 1.2.2.1

[9] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, pages 24–26, 1986. 1.2.2.1

[10] IBM Cloud Education. What is machine learning?, 2020. URL `https://www.ibm.com/cloud/learn/machine-learning`. Accessed June 23, 2022. 1.2.2.2

[11] Wikipedia contributors. Machine learning — Wikipedia, the free encyclopedia, 2022. URL `https://en.wikipedia.org/w/index.php?title=Machinelearning&oldid=1094363111`. [Online; accessed 23-June-2022]. 1.2.2.2

[12] Wikipedia contributors. Supervised learning — Wikipedia, the free encyclopedia, 2022. URL `https://en.wikipedia.org/w/index.php?title=Supervisedlearning&oldid=1093417731`. [Online; accessed 23-June-2022]. 1.2.2.2

[13] Jason Brownlee. Supervised and unsupervised machine learning algorithms, 2016. URL `https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/`. Accessed June 23, 2022. 1.2.2.2, 1.2.2.2

[14] Wikipedia contributors. Unsupervised learning — Wikipedia, the free encyclopedia, 2022. URL `https://en.wikipedia.org/w/index.php?title=Unsupervisedlearning&oldid=1083933993`. [Online; accessed 23-June-2022]. 1.2.2.2

[15] Wikipedia contributors. Artificial neural network — Wikipedia, the free encyclopedia, 2022. URL `https://en.wikipedia.org/w/index.php?title=Artificialneuralnetwork&oldid=1092513725`. [Online; accessed 23-June-2022]. 1.2.2.3

[16] Charles C Tappert. Who is the father of deep learning? In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 343–348. IEEE, 2019. 1.2.2.3

[17] et al. Ivakhnenko, Aleksei Grigorevich. *Cybernetics and forecasting techniques*, volume 8. American Elsevier Publishing Company, 1967. 1.2.2.3

[18] Sebastian Ruder. A review of the neural history of natural language processing, 2018. URL `https://ruder.io/a-review-of-the-recent-history-of-nlp/index.html#2001neurallanguagemodels`. Accessed June 23, 2022. 1.2.2.3

[19] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000. 1.2.2.3

[20] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008. 1.2.2.3

[21] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 1.2.2.3, 1.4.2, 1.4.2.1

[22] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, ON, Canada, 2013. 1.2.2.3

[23] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014. 1.2.2.3

[24] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. URL `http://arxiv.org/abs/1408.5882`. 1.2.2.3

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014. 1.2.2.3, 1.4.2.2

[26] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1.2.2.3, 1.4.2.2

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1.2.2.3, 1.4.2.2, 1.4.2.2, 1.4.2.2

[28] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014. 1.2.2.3

[29] Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28, 2015. 1.2.2.3

[30] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30, 2017. 1.2.2.3, 1.4.2, 1.4.2.2

[31] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202. 1.2.2.3, 1.4.2, 1.4.2.2

[32] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018. 1.2.2.3, 1.4.2, 1.4.2.2

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1.2.2.3, 1.4.2, 1.4.2.2

[34] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000. 1.3.2.2

[35] Irina Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001. 1.3.2.2

[36] Silviu Guiasu and Abe Shenitzer. The principle of maximum entropy. *The mathematical intelligencer*, 7(1):42–48, 1985. 1.3.2.2

[37] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017. 1.3.2.2

[38] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 1.3.2.2

[39] Naomi S Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992. 1.3.2.2

[40] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1): 86–97, 2012. 1.3.2.2

[41] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003. 1.3.2.2

[42] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. 1.3.2.2

[43] Jason Brownlee. Promise of deep learning for natural language processing, 2017. URL https://machinelearningmastery.com/promise-deep-learning-natural-language-processing/. Accessed June 23, 2022. 1.3.3.1, 1.3.3.1

[44] Sawan Saxena. Understanding embedding layer in keras, 2020. URL https://medium.com/analytics-vidhya/understanding-embedding-layer-in-keras-bbe3ff1327ce. Accessed June 23, 2022. 1.3.3.2

[45] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009. 1.3.3.2

[46] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017. 1.3.3.2

[47] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986. 1.3.3.2

[48] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020. 1.3.3.2

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 1.3.3.2, 2.2.4.2

[50] Christopher Olah. Understanding lstm networks, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed June 23, 2022. 1.3.3.2, 2.2.4.2

[51] Jason Brownlee. Cnn long short-term memory networks, 2017. URL https://machinelearningmastery.com/cnn-long-short-term-memory-networks/. Accessed June 23, 2022. 1.3.3.2

[52] Jeet. One hot encoding of text data in natural language processing., 2020. URL https://medium.com/analytics-vidhya/one-hot-encoding-of-text-data-in-natural-language-processing-2242 Accessed June 23, 2022. 1.4.1.1

[53] Wikipedia contributors. Bag-of-words model — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Bag-of-wordsmodel&oldid=1087243933. [Online; accessed 23-June-2022]. 1.4.1.2

[54] Wikipedia contributors. Tf–idf — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=1092578440. [Online; accessed 24-June-2022]. 1.4.1.3

[55] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 1.4.2, 1.4.2.1

[56] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017. 1.4.2, 1.4.2.1

[57] Jay Alammar. The illustrated word2vec, 2019. URL https://jalammar.github.io/illustrated-word2vec/. Accessed June 23, 2022. 1.4.2.1

[58] Jay Alammar. The illustrated transformer, 2018. URL https://jalammar.github.io/illustrated-transformer/. Accessed June 23, 2022. 1.4.2.2

[59] Jay Alammar. The illustrated bert, elmo, and co. (how nlp cracked transfer learning), 2018. URL https://jalammar.github.io/illustrated-bert/. Accessed June 23, 2022. 1.4.2.2, 2.2.4.5

[60] Wikipedia contributors. Sentiment analysis — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Sentiment analysis&oldid=1091286443. [Online; accessed 23-June-2022]. 2.1.1

[61] Harika Bonthu. Rule-based sentiment analysis in python, 2021. URL https://www.analyticsvidhya.com/blog/2021/06/rule-based-sentiment-analysis-in-python/. Accessed June 23, 2022. 2.2.2.1

[62] Steven Loria. Textblob: Simplified text processing, 2020. URL https://textblob.readthedocs.io/en/dev/. Accessed June 23, 2022. 2.2.2.1

[63] Clayton Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, volume 8, pages 216–225, 2014. 2.2.2.1

[64] Andrea Esuli and Fabrizio Sebastiani. SENTIWORDNET: A publicly available lexical resource for opinion mining. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2006/pdf/384pdf.pdf. 2.2.2.1

[65] Nishtha Arora. Pre-processing of text data in nlp, 2021. URL https://www.analyticsvidhya.com/blog/2021/06/pre-processing-of-text-data-in-nlp/. Accessed June 23, 2022. 2.2.3

[66] Harshith. Text preprocessing in natural language processing, 2019. URL https://towardsdatascience.com/

text-preprocessing-in-natural-language-processing-using-python-61
Accessed June 23, 2022. 2.2.3.1

[67] Stanford University. Stemming and lemmatization, 2008. URL https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. Accessed June 23, 2022. 2.2.3.1

[68] Aravindpai Pai. What is tokenization in nlp? here's all you need to know, 2020. URL https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/. Accessed June 23, 2022. 2.2.3.1

[69] Mohd Zeeshan Ansari, MB Aziz, MO Siddiqui, H Mehra, and KP Singh. Analysis of political sentiment orientations on twitter. *Procedia Computer Science*, 167:1821–1828, 2020. 2.2.4.2

[70] LILIS Kurniasari and ARIEF Setyanto. Sentiment analysis using recurrent neural network-lstm in bahasa indonesia. *J. Eng. Sci. Technol*, 15(5):3242–3256, 2020. 2.2.4.2

[71] Yi Cai, Qingbao Huang, Zejun Lin, Jingyun Xu, Zhenhong Chen, and Qing Li. Recurrent neural network with pooling operation and attention mechanism for sentiment analysis: A multi-task learning approach. *Knowledge-Based Systems*, 203:105856, 2020. 2.2.4.2

[72] Dipak Gaikar, Ganesh Sapare, Akanksha Vishwakarma, and Apurva Parkar. Twitter sentimental analysis for predicting election result using lstm neural network. *IRJET*, 2019. 2.2.4.2

[73] Baidya Nath Saha, Apurbalal Senapati, and Anmol Mahajan. Lstm based deep rnn architecture for election sentiment analysis from bengali newspaper. In *2020 International Conference on Computational Performance Evaluation (ComPE)*, pages 564–569. IEEE, 2020. 2.2.4.2

[74] Mei Wang, Yangyang Zhu, Shulin Liu, Chunfeng Song, Zheng Wang, Pai Wang, and Xuebin Qin. Sentiment analysis based on attention mechanisms and bi-directional lstm fusion model. In *2019 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 865–868. IEEE, 2019. 2.2.4.2

[75] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013. 2.2.4.2

[76] Jason Brownlee. Stacked long short-term memory networks, 2017. URL https://machinelearningmastery.com/

stacked-long-short-term-memory-networks/. Accessed June 23, 2022. 2.2.4.2

[77] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997. 2.2.4.2

[78] Jason Brownlee. How to develop a bidirectional lstm for sequence classification in python with keras, 2017. URL https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/. Accessed June 23, 2022. 2.2.4.2

[79] Cezanne Camacho. Cnns for text classification, 2019. URL https://cezannec.github.io/CNNTextClassification/. Accessed June 23, 2022. 2.2.4.3

[80] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015. 2.2.4.3

[81] Chenquan Gan, Lu Wang, Zufan Zhang, and Zhangyi Wang. Sparse attention based separable dilated convolutional neural network for targeted sentiment analysis. *Knowledge-Based Systems*, 188:104827, 2020. 2.2.4.3

[82] Marco Pota, Massimo Esposito, Marco A Palomino, and Giovanni L Masala. A subword-based deep learning approach for sentiment analysis of political tweets. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 651–656. IEEE, 2018. 2.2.4.3

[83] Anwar Ur Rehman, Ahmad Kamran Malik, Basit Raza, and Waqar Ali. A hybrid cnn-lstm model for improving accuracy of movie reviews sentiment analysis. *Multimedia Tools and Applications*, 78(18):26597–26613, 2019. 2.2.4.4

[84] P Kaladevi and K Thyagarajah. Retracted article: Integrated cnn-and lstm-dnn-based sentiment analysis over big social data for opinion mining. *Behaviour & Information Technology*, 40(9):XI–XIX, 2021. 2.2.4.4

[85] Muhammad Umer, Imran Ashraf, Arif Mehmood, Saru Kumari, Saleem Ullah, and Gyu Sang Choi. Sentiment analysis of tweets using a unified convolutional neural network-long short-term memory network model. *Computational Intelligence*, 37(1):409–434, 2021. 2.2.4.4

[86] Shervin Minaee, Elham Azimi, and AmirAli Abdolrashidi. Deep-sentiment: Sentiment analysis using ensemble of cnn and bi-lstm models. *arXiv preprint arXiv:1904.04206*, 2019. 2.2.4.4

[87] Fazeel Abid, Muhammad Alam, Muhammad Yasir, and Chen Li. Sentiment analysis through recurrent variants latterly on convolutional neural network of twitter. *Future Generation Computer Systems*, 95:292–308, 2019. 2.2.4.4

[88] Mingjie Ling, Qiaohong Chen, Qi Sun, and Yubo Jia. Hybrid neural network for sina weibo sentiment analysis. *IEEE Transactions on Computational Social Systems*, 7(4):983–990, 2020. 2.2.4.4

[89] M. El Barachi, M. AlKhatib, S. Mathew, and F. Oroumchian. A novel sentiment analysis framework for monitoring the evolving public opinion in real-time: Case study on climate change. *Journal of Cleaner Production*, page 127820, 2021. 3.2, 3.3.1.2, 3.3.1.4

[90] D. M. Romero, B. Meeder, and J. Kleinberg. Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on Twitter. In *Proceedings of the International Conference on World Wide Web*, 2011. 3.2

[91] J. C. Baumgartner, J. B. Mackay, J. S. Morris, E. E. Otenyo, L. Powell, M. M. Smith, N. Snow, F. I. Solop, and B. C. Waite. *Communicator-in-chief: How Barack Obama used new media technology to win the White House*. Lexington Books, 2010. 3.2

[92] D. Lorenzi, J. Vaidya, B. Shafiq, S. Chun, N. Vegesna, Z. Alzamil, N. Adam, S. Wainer, and V. Atluri. Utilizing social media to improve local government responsiveness. In *Proceedings of the Annual International Conference on Digital Government Research*, 2014. 3.2

[93] K. Ravi and V. Ravi. A survey on opinion mining and sentiment analysis: tasks, approaches and applications. *Knowledge-based systems*, 89:14–46, 2015. 3.2

[94] J. Ramteke, S. Shah, D. Godhia, and A. Shaikh. Election result prediction using Twitter sentiment analysis. In *Proceedings of the IEEE International Conference on Inventive Computation Technologies (ICICT)*, 2016. 3.2

[95] A. Agarwal, R. Singh, and D. Toshniwal. Geospatial sentiment analysis using Twitter data for UK-EU referendum. *Journal of Information and Optimization Sciences*, 39(1): 303–317, 2018. 3.2

[96] C. Chatfield. *Time-series forecasting*. CRC Press, 2000. 3.2

[97] L. F. Bright, K. L. Sussman, and G. B. Wilcox. Facebook, trust and privacy in an election year: Balancing politics and advertising. *Journal of Digital & Social Media Marketing*, 8 (4):332–346, 2021. 3.3.1

[98] J. Lee and Y.-S. Lim. Gendered campaign tweets: the cases of Hillary Clinton and Donald Trump. *Public Relations Review*, 42(5):849–855, 2016. 3.3.1

[99] L. Buccoliero, E. Bellio, G. Crestini, and A. Arkoudas. Twitter and politics: Evidence from the US presidential elections 2016. *Journal of Marketing Communications*, 26(1): 88–114, 2020. 3.3.1

[100] D. Grimaldi. Can we analyse political discourse using twitter? evidence from Spanish 2019 presidential election. *Social Network Analysis and Mining*, 9(1):1–9, 2019. 3.3.1.1

[101] Michael Cornfield. Empowering the party-crasher: Donald J. Trump, the first 2016 GOP presidential debate, and the Twitter marketplace for political campaigns. *Journal of Political Marketing*, 2017. 3.3.1.1, 3.6

[102] B. Heredia, J. Prusa, and T. Khoshgoftaar. Exploring the effectiveness of Twitter at polling the United States 2016 presidential election. In *Proceedings of the IEEE International Conference on Collaboration and Internet Computing (CIC)*, 2017. 3.3.1.2, 3.3.1.4

[103] D. Grimaldi, J. D. Cely, and H. Arboleda. Inferring the votes in a new political landscape: the case of the 2019 Spanish presidential elections. *Journal of Big Data*, 7(1):1–19, 2020. 3.3.1.2, 3.3.1.4

[104] L. Tavoschi, F. Quattrone, E. D'Andrea, P. Ducange, M. Vabanesi, F. Marcelloni, and P. L. Lopalco. Twitter as a sentinel tool to monitor public opinion on vaccination: an opinion mining analysis from September 2016 to August 2017 in Italy. *Human Vaccines & Immunotherapeutics*, 16(5):1062–1069, 2020. 3.3.1.2, 3.3.1.4

[105] M. Wang, H. Wu, T. Zhang, and S. Zhu. Identifying critical outbreak time window of controversial events based on sentiment analysis. *PLOS One*, 15(10):e0241355, 2020. 3.3.1.2, 3.3.1.4

[106] W. Shi, H. Wang, and S. He. Sentiment analysis of Chinese microblogging based on sentiment ontology: a case study of '7.23 Wenzhou Train Collision'. *Connection Science*, 25(4):161–178, 2013. 3.3.1.3, 3.3.1.4, 3.6

[107] I. Onyenwe, S. Nwagbo, N. Mbeledogu, and E. Onyedinma. The impact of political party/candidate on the election results from a sentiment analysis perspective using# anambradecides2017 tweets. *Social Network Analysis and Mining*, 10(1):1–17, 2020. 3.3.1.3, 3.3.1.4

[108] U. Yaqub, S. A. Chun, V. Atluri, and J. Vaidya. Sentiment based analysis of tweets during the US presidential elections. In *Proceedings of the Annual International Conference on Digital Government Research*, 2017. 3.3.1.3, 3.3.1.4, 3.6

[109] O. Kraaijeveld and J. De Smedt. The predictive power of public Twitter sentiment for forecasting cryptocurrency prices. *Journal of International Financial Markets, Institutions and Money*, 65:101188, 2020. 3.3.1.3, 3.3.1.4

[110] F. Qi, C. Yang, Z. Liu, Q. Dong, M. Sun, and Z. Dong. Openhownet: An open sememe-based lexical knowledge base. *arXiv preprint arXiv:1901.09957*, 2019. 3.3.1.4

[111] S. Zhang, Z. Wei, Y. Wang, and T. Liao. Sentiment analysis of Chinese micro-blog text based on extended sentiment dictionary. *Future Generation Computer Systems*, 81:395–403, 2018. 3.3.1.4

[112] M. Thelwall, K. Buckley, G. Paltoglou, D. Cai, and A. Kappas. Sentiment strength detection in short informal text. *Journal of the American society for information science and technology*, 61(12):2544–2558, 2010. 3.3.1.4

[113] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2013. 3.3.1.4

[114] C. Hutto and E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the International AAAI Conference on Web and Social Media*, 2014. 3.3.1.4

[115] H. Maqsood, I. Mehmood, M. Maqsood, M. Yasir, S. Afzal, F. Aadil, M. M. Selim, and K. Muhammad. A local and global event sentiment based efficient stock exchange forecasting using deep learning. *International Journal of Information Management*, 50:432–451, 2020. 3.3.2

[116] J. Kordonis, S. Symeonidis, and A. Arampatzis. Stock price forecasting via sentiment analysis on Twitter. In *Proceedings of the Pan-Hellenic Conference on Informatics*, 2016. 3.3.2

[117] M. Oussalah and A. Zaidi. Forecasting weekly crude oil using Twitter sentiment of US foreign policy and oil companies data. In *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI)*, 2018. 3.3.2

[118] M. Arias, A. Arratia, and R. Xuriguera. Forecasting with Twitter data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):1–24, 2014. 3.3.2

[119] R. Kiran, P. Kumar, and B. Bhasker. OSLCFit (organic simultaneous LSTM and CNN Fit): a novel deep learning based solution for sentiment polarity classification of reviews. *Expert Systems with Applications*, 157:113488, 2020. 3.4.2, 3.4.2.3

[120] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar. Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). *arXiv preprint arXiv:1903.08983*, 2019. 3.4.2.1

[121] J. Ling and R. Klinger. An empirical, quantitative analysis of the differences between sarcasm and irony. In *European Semantic Web Conference*. Springer, 2016. 3.4.2.1

[122] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 3.4.2.3

[123] H. Hewamalage, C. Bergmeir, and K. Bandara. Recurrent Neural Networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37 (1):388–427, 2021. 3.5.2.1

[124] F. Orabona and T. Tommasi. Training deep networks without learning rates through coin betting. *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 30: 2160–2170, 2017. 3.5.2.1

[125] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*. Springer, 2011. 3.5.2.1

[126] O. Claveria, E. Monte, and S. Torra. Data preprocessing for neural network-based forecasting: does it really matter? *Technological and Economic Development of Economy*, 23 (5):709–725, 2017. 3.5.2.1

[127] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006. 3.5.2.2

[128] D. K. Tayal, S. Yadav, K. Gupta, B. Rajput, and K. Kumari. Polarity detection of sarcastic political tweets. In *Proceedings of the International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2014. 3.5.3

[129] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. 4.2, 4.3.1, 4.5.1

[130] A. Ghosh, G. Li, T. Veale, P. Rosso, E. Shutova, J. Barnden, and A. Reyes. Semeval-2015 Task 11: Sentiment analysis of figurative language in Twitter. In *Proceedings of the International Workshop on Semantic Evaluation (SemEval)*, 2015. 4.2, 4.3.1, 4.5.2

[131] S. Muresan, R. Gonzalez-Ibanez, D. Ghosh, and N. Wacholder. Identification of nonliteral language in social media: A case study on sarcasm. *Journal of the Association for Information Science and Technology*, 67(11):2725–2737, 2016. 4.2

[132] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 4.2, 4.3.2, 4.4, 4.4

[133] F. Barbieri, F. Ronzano, and H. Saggion. UPF-taln: SemEval 2015 Tasks 10 and 11. Sentiment analysis of literal and figurative language in Twitter. In *Proceedings of the International Workshop on Semantic Evaluation (SemEval 2015)*, 2015. 4.3.1, 4.1

[134] C. Van Hee, E Lefever, and V. Hoste. LT3: sentiment analysis of figurative tweets: piece of cake# NotReally. In *Proceedings of the International Workshop on Semantic Evaluations (SemEval 2015)*, 2015. 4.3.1

[135] T. Hercig and L. Lenc. The impact of figurative language on sentiment analysis. In *Proceedings of the Recent Advances in Natural Language Processing (RANLP)*, 2017. 4.3.1

[136] R.-A. Potamias, G. Siolas, and A. Stafylopatis. A robust deep ensemble classifier for figurative language detection. In *Proceedings of the International Conference on Engineering Applications of Neural Networks (EANN)*. Springer, 2019. 4.3.1, 4.1

[137] R. A. Potamias, G. Siolas, and A.-G. Stafylopatis. A Transformer-based approach to irony and sarcasm detection. *Neural Computing and Applications*, 32(23):17309–17320, 2020. 4.3.1, 4.5.1, 4.1, 4.5.2

[138] 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. *arXiv preprint arXiv:1907.11692*, 2019. 4.3.1, 4.5.1, 4.1

[139] T. Chen, I. Goodfellow, and J. Shlens. Net2Net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015. 4.3.2

[140] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 4.3.2

[141] N. Passalis and A. Tefas. Unsupervised knowledge transfer using similarity embeddings. *IEEE Transactions on Neural Networks and Learning Systems*, 30(3):946–950, 2018. 4.3.2

[142] S. I. Mirzadeh, M. Farajtabar, A. Li, N. Levine, A. Matsukawa, and H. Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. 4.3.2

[143] M. Phuong and C. Lampert. Towards understanding knowledge distillation. In *Proceedings of the International Conference on Machine Learning*, 2019. 4.3.2, 4.4

[144] L. Saglietti and L. Zdeborová. Solvable model for inheriting the regularization through knowledge distillation. *arXiv preprint arXiv:2012.00194*, 2020. 4.3.2

[145] S. Lai, L. Xu, K. Liu, and J. Zhao. Recurrent Convolutional Neural Networks for text classification. In *Proceedings of AAAI conference on artificial intelligence*, 2015. 4.5.1

[146] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018. 4.1

[147] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. John, N. Constant, M. Guajardo-Céspedes, S. Yuan, and C. Tar. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018. 4.1

[148] S. I. Wang and C. D. Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2012. 4.1

[149] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016. 4.1

[150] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2019. 4.1

[151] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pretraining of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 4.1

[152] C. Ozdemir and S. Bergler. CLaC-SentiPipe: Semeval2015 Subtasks 10 b, e, and Task 11. In *Proceedings of the International Workshop on Semantic Evaluation (SemEval 2015)*, 2015. 4.1

[153] R. Kiran, P. Kumar, and B. Bhasker. OSLCFit (Organic Simultaneous LSTM and CNN Fit): a novel deep learning-based solution for sentiment polarity classification of reviews. *Expert Systems with Applications*, 157:113488, 2020. 4.5.1

[154] J. Ling and R. Klinger. An empirical, quantitative analysis of the differences between sarcasm and irony. In *Proceedings of the European Semantic Web Conference*. Springer, 2016. 4.5.1