# TSAPLAY: A framework to aid reproducibility in and exploration of the field of Targeted Sentiment Analysis

**Sean Bugeja**

Supervisor: Mr. Mike Rosner


**Faculty of ICT**

**University of Malta**

enter a date

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science*

# Faculty of ICT

## Declaration

I, the undersigned, declare that the dissertation entitled:

> TSAPLAY: A framework to aid reproducibility in and exploration of the field of Targeted Sentiment Analysis

submitted is my work, except where acknowledged and referenced.

Sean Bugeja

enter a date

# Acknowledgements

your acknowledgments

# Abstract

Provides a short (typically 1 page) overview of the dissertation's contents including the tackled problem and high-level results/conclusions.

# Contents

# List of Figures

# List of Tables

# Acronyms

**NLP** Natural Language Processing

**LSTM** Long Short Term Memory

**OOV** Out of Vocabulary

**IV** In Vocabulary

**SVM** Support Vector Machine

**ASR** Automatic Speech Recognition

**RC** Reading Comprehension

**LDA** Latent Dirichlet Allocation

**LSA** Latent Semantic Analysis

**CLI** Command Line Interface

# 1. Introduction

## 1.1 Targeted Sentiment analysis

Targeted sentiment analysis (TSA) is a fine-grained text-classification task that stems from the broader, more general, document, or sentence, level sentiment analysis. The former extends on the latter by taking into consideration a particular target or aspect within the context of the document, and aims to identify the sentiment with respect to this target or aspect [51], [39], [54].

Target and aspect are closely related. A *target* is a particular noun or subject within the phrase while an *aspect* can be a more general area or concept that the phrase touches on, without referencing in the literal sense. Consider a sentence such as, "The waiting times were long however the ravioli were simply to die for", a plausible *target* could be "waiting times" for which the statement conveys a negative sentiment. Alternatively, the phrase could be assessed with respect to an *aspect* such as "food quality", for which a positive sentiment is conveyed even though the precise term "food quality" is only implicit.

It is evident that, separating itself from sentence-oriented sentiment analysis, target or aspect-based sentiment analysis requires the careful consideration of the target or aspect in question along with its context. This was initially demonstrated by [29], whose work revealed that a staggering 40% of errors within the field of targeted-sentiment analysis could be attributed to the lack of consideration of the

target or aspect.

Due to the proliferation of social media networks and online shopping, opinions voiced from users on specific topics, products, services and events have never been as readily available for data mining as they are now. The value in having the means to accurately gauge public interest and opinion of very specific topics of interest on such a phenomenal scale cannot be understated. From those in the public sector, such as electoral campaigns who seek to obtain a clearer picture of their constituents' strongest held opinions and expectations, to private businesses who wish to employ the most effective advertising campaign for their products and services, all of these objectives rely heavily on being as cognizant on public sentiment as possible. [64]

Over time the content of these online text sources has become more sophisticated and richer in information. Changes in social media platforms such as *twitter*'s decision to raise the character limit of tweets results in the same unit of data conveying up to twice the amount of information. As this availability increases, so must the resolution at which this information is processed, so as to keep pace with the needs of both producers and consumers alike. This phenomenon further pushes the need to focus on opinion mining at a finer-grained level, perfecting the ability to discern varying sentiments towards separate targets within the same phrase.

## 1.2 Out-Of-Vocabulary Words

When dealing with language-related tasks such as TSA, the word embedding matrix from which different words' are represented numerically can be regarded as the working vocabulary that the model has prior to tackling the task at hand.

It is impossible to account for the entire vocabulary of a particular language when constructing word embeddings. Regardless of the amount of text that is initially used to construct the word embeddings there shall be words that are not encountered within that text and therefore a continuous vector representation of

that word could not be produced. These previously unseen words are consequently referred to as out-of-vocabulary (OOV) words.

The magnitude of the challenge that OOV words present, and the potential repercussions thereof, become evident in the landscape of this analogy. These represent words which the models essentially have no understanding of and can be of little help to it when attempting to extract any information it may convey to the task at hand. While the model can be expected to learn more about these words through repeated encounters in different contexts, the fact that these words, by their own nature, tend to occur infrequently in language substantially diminishes the efficacy of this learning process.

OOV words are of particular concern when dealing with tasks that are generative in nature, such as ASR. The toll of OOV words on the performance of approaches to these tasks is two-fold. Firstly, OOV word may be substituted with an incorrect IV word. Secondly, the OOV word has a direct effect on the neighbouring IV words [48].

A common approach to this problem is clustering OOV words into groups that would are sufficiently expressive of their constituents. Various techniques have been employed towards this goal such syntactic and morphological features, part-of-speech tag information, online resources, and subword-level models to name a few, [48] does a good job of outlining these approaches.

Since sentiment analysis is a classification task, where words are provided as input and subsequently used as keys when looking up the relevant embedding vector, substituting an OOV word for an IV word is of no concern. The effect of an OOV word on its neighboring words, however, is prone to undermine a model's ability to generate an accurate representation of the content as a whole.

This sort of phenomenon is not particularly difficult to imagine since it is often times the case in languages that a single word can have drastic effects on the meaning of a phrase, particularly in situations expressing negation. Consider a phrase such as "It avoids all the predictability found in Hollywood movies.", where

"predictability" conveys a negative sentiment, which is subsequently negated by the verb "avoids".

Moreover, OOV words obviously make the process of comprehending a phrase more difficult by introducing elements that the model has no knowledge of. If the word embedding model that is being used is analogous to the model's understanding of a language, an OOV word is effectively a word the model does not understand, and therefore has limited means by which to gauge the effect of that word on the overall sentiment of the phrase, if any.

A typical approach to this OOV challenge within the field of sentiment analysis is the use of a particular singular token that is meant to represent low-frequency words during the training phase, and subsequently model all OOV words encountered in the test phase. The vector for this token is often times initialized using some bounded random uniform distribution.

As far back as [21], before the popularity of pre-trained word embeddings such as *GloVe* and *Word2Vec*, it was pointed out that using a single token is somewhat crude. It could not possibly encompass the wealth of linguistic information expressed by every OOV word that is encountered; consider that an OOV word can be anything from a spelling mistake to proper noun, such as the name of an entity, and anything in between.

When training an n-gram model, the use of a single $<UNK>$ label for all OOV words will lead to a substantial inconsistency in the frequency of OOV words between training and test datasets [21]. This inconsistency is comparable to the possibly counter productive training that is carried out on the singular $<UNK>$ vector across different samples within the scope of sentiment analysis and word embeddings.

In their work, dealing particularly with OOV tokens within the field of Reading Comprehension (RC), [9] note a considerable drop in performance when taking this approach in some cases and suggest that a unique OOV token would lack the desired level of detail to correctly generate a correct answer.

It is not necessarily useful to approach the OOV word challenge at the word-level. It is assumed that OOV words are scarce, which substantially limits the occasions for a prospective model to learn any discerning information about that word. Attempting to model clusters of OOV words instead, would benefit each member of the cluster by the accumulated frequency of all members [48].

Moreover, within the scope of sentiment analysis, the intuition for clustering words under classes characterized by some particular sentimental value, or a lack thereof; as in the case of registered trademarks, would be evidently beneficial.

Too few classes may not possess a sufficiently fine level of detail in their discerning characteristics and cluster together words which are unrelated and subsequently erroneously trained together. This can be seen from the extreme of this case, where only a single token is used, and the issues that have been reported for this approach.

Conversely, if an excessive number of classes are used, this would naturally decrease the amount of OOV words within each class, and consequently the frequency of words appearing in a particular sample. This hinders a model's ability to learn any distinguishing characteristics of a class. Taken to the extreme, if each class were to contain only a single word, this would effectively render each class as a randomly initialized vector for this word which is rarely encountered, and trained. This undermines the purpose of a word vector, which is to convey as much information about the word as possible.

Within the scope of a RC task, [9], carry out a study to accurately measure the effects that different embeddings and OOV approaches can have on the final result of two benchmark models.

They outline the typical approach to RC problems as initially generating a representation of the source document, possibly through the use of pre-trained word embeddings such as *GloVe* in conjunction with statistical models such as the LSTM [26] which may employ an attention mechanism (eg. [5]). The result of this process is a contextual representation of the document from which a valid answer can be extracted.

It is worth noting that this process is not dissimilar from the majority of approaches that have been adopted recently within the field of sentiment analysis. Both employ similar techniques and maintain the same characteristic order of events in generating a substantive representation of the source, differing only in the objective and hence the final product to be extracted from that representation. While this is by no means an insignificant difference, on a macro level this can be seen merely as adjusting the variables and parameters that are input to the system, as opposed to the system as a whole.

In their work, [9] suggest that there are notable effects on the downstream results of models when comparing the use of different word embeddings, pre-trained or otherwise. Specifically, as an out-of-the-box solution, they recommend the use of GloVe [53] 200-dimension pre-trained embeddings. Moreover, for their benchmark RC models, they recommend assigning random unique vectors for OOV tokens at test time, possibly due to the fact that subjects in generated responses are likely to be OOV token and proper nouns.

Based on these findings, the aforementioned similarity in the process of tackling RC tasks and SA tasks, along with the challenges that OOV words pose in the field of SA as previously outlined, the study of word embedding choice and OOV approaches and their effects therein is something that we believe merits further investigation.

## 1.3 Reproducibility

The ability to reproduce experiments is the integral basis upon which all disciplines of science are founded. Within many fields, NLP among them, this typically entails adherence to three integral guidelines, namely, (a) the provision of sufficiently detailed methodologies, (b) the release of operational code-bases and (c) access to the dataset(s) with clear details pertaining to any processing and/or stratification strategies used. These guidelines ensure that results can be easily reproduced,

evaluated for generalizability and compared to other methods in the field, thereby fostering growth.

In [3], the authors underscore the significance of reproducibility of approaches as well as the generalizability of the results that are reported, and proceed to argue that adherence to the aforementioned tenets has been lacking in recent years, paying particular attention in their work to the field of targeted sentiment analysis.

The authors also draw attention to the fact that a multitude of studies report results on different datasets which often stem from diverse sources that could be composed of language that is centered around a particular topic. Notable still, these datasets also exhibit consequential statistical differences such as the average length of, and/or amount of targets in, a sentence. Occasionally, studies also carry out particular alterations to existing datasets or adopt a specific strategy for merging one or more datasets (eg. [74]). These factors substantially limit the possibility of effectively comparing the novel approaches as they emerge in the field.

Replication studies such as [3] can remedy this issue by attempting to reproduce the studies in a comparative setting, however they outline challenges in this regard as well. The authors note that a number of approaches in the field fail to outline the precise pre-processing steps they adopted, which may have substantial effects on the downstream performance of a model, and thus make reproducing the results difficult. In some situations they also note model settings mentioned that are not necessarily self-evident, such as a "softmax clipping threshold" [63], which, in attempting to reproduce the study, [3] were forced to ignore as they were unfamiliar with the term.

One key observation that [3] also make with regards to deep learning and neural network based approaches such as [63] is the influence that an initial random seed has on the final performance results, particularly when using smaller word embeddings ([50] as cited in [3]). They mention this as the probable reason for other studies [11],[66] (including their own), not being able to recreate the original results reported in [63]. The authors remark that in situations such as this, when

dealing with models of this nature, it would be well-advised to gauge a model's performance over a number of experiment runs as opposed to one.

## 1.4 Objectives

Following the principles outlined in [3], and motivated by the observations on these principles made by [9] in the field of reading comprehension and the effect of out-of-vocabulary(OOV) embedding strategies thereof, the objectives of this study are two-fold.

1. We propose to extend the work carried out by [3] in the field of TSA to cover a wider range of studies, including those that employ techniques that have since emerged focusing on attention mechanisms and memory networks. This entails the attempt to reproduce these models based on the detail provided in the original studies and subsequently carrying out a comparative evaluation of these approaches across a wide range of datasets from varying domains using a number of different pre-trained word embeddings.

2. Inspired by the work and findings of [9], we shall endeavor to investigate the effect that different OOV embedding strategies and pre-trained word embeddings have on the downstream performance of models with respect to TSA. To our knowledge, at the time of writing, this study will be the first to investigate this issue in detail.

To achieve these goals, we make three principal contributions through this work:

- A publicly accessible framework that provides access to a range of frequently cited datasets that have been used in the field of TSA. This framework shall be used to obtain robust performance metrics, such as macro-f1 scores for all models, which have been hitherto unreported for a subset of the models, as well as other informative measures that shed light into the inner workings of the models implemented, such as attention heat-maps (where applicable).

- A comparative evaluation of models across different domains and datasets, using different pre-trained word embeddings to ascertain the degree to which results obtained are reproducible and generalizable.
- Detailed reports on a series of experiments using different OOV embedding strategies across all implemented models, the results of which will allow us to deduce the degree to which these affect downstream performance and whether an optimal approach can be found that proves to be generally beneficial.

Finally, the proposed framework shall also serve as the groundwork for future experimentation into alternative, more sophisticated, OOV embedding approaches while also providing a means of rapidly carrying out comparative evaluations of TSA models across different datasets and pre-trained word embeddings.

## 1.5 Document Structure

Prior to presenting the design and implementation details of our framework, we provide a more detailed description of the field of TSA in chapter 2. First, we provide a wider view of the field in its current state, the challenges it must overcome as well as the metrics commonly used to measure its progress. We follow this with an account of the approaches employed to this task and the emergent concepts thereof which propelled the field forward, from manual feature engineering to sophisticated attention mechanisms.

Chapter 3 illustrates the principles we adopt for the design of each constituent unit of our framework from a high-level point of view, how these units interact and, our motivations for these design choices. A more in-depth review follows in chapter 4 which details specifics of the implementation both of the framework as a whole and its more complex components. We wrap-up the framework specification in chapter 5 by listing the different analysis and evaluation outputs of the framework.

We use our framework to carry out a series of experiments, covering both reproducibility and OOV handling, the results of which we include in chapter 6,

followed by an analysis of these results and our observations. Finally, we present our concluding remarks in chapter 7 accompanied by our views on possible avenues to pursue in the future development of this framework.

# 2. Background and Literature Review

## 2.1 Current State of TSA

### 2.1.1 Challenges

As with any task that requires a deeper understanding of the intricacies of language, there are many challenges that face TSA. Many of these challenges are inherent to parsing the structure of a language such as sarcasm, where sentences such as "Nice perfume. You must shower in it." [68] are composed entirely of positive-sentiment bearing words while expressing a negative sentiment. These notwithstanding, the informal nature of the majority of text data found on social media platforms (upon which this task frequently focuses), supplements these challenges with its own.

Colloquialisms and social short-hands are commonplace within social media networks where many users intend to convey as much information in as few characters as possible, particularly in situations where this number is capped. This phenomenon also leads to intentional, as well as unintentional, spelling errors which further obscures that data for any prospective machine learning model that does not account for these circumstances.

Along with these challenges, the literature also presents a number of obstacles

and particularly problematic instances that need to be taken into account when approaching the task of targeted sentiment analysis. Comparative opinions are one such circumstance where the sentiment being conveyed is obscured by another subject. [64] report challenges of this sort, with phrases such as "I've had better Japanese food at a mall food court".

Other common challenges that are pointed out in [64], are negation and conditional situations, citing the example "but dinner here is never disappointing, even if the prices are a bit over the top", where the sentiment towards the target cannot be easily deduced from the various syntactic structures present.

Moreover, the particular case of expressions which consist of multiple words needs to be given special care. Various approaches that employ word embeddings operate on the word as the atomic unit of operation, and would therefore struggle to correctly model an expression such as "die for" in "the ice cream to die for" [64] from its constituents. [79] also also stress the significance of this issue, and argue that it has not been given sufficient attention, particularly when modelling *targets* that also consist of multiple words.

When considering the opinion of a sentence towards a specific target, it may be the case that the sentence will have opposing sentiments for different targets, this is another degree of complexity that targeted-sentiment analysis models need to account for as opposed to sentiment analysis of the sentence as a whole [65]. Phrases such as "great food but the service was dreadful!" convey different and opposite sentiments towards "food" and "service" [64]. Previous sentence oriented sentiment analysis approaches such as [58], [4] would be incapable of correctly distinguishing this level of granularity [11].

[18], [72] also call attention to the fact that there are several instances where the sentiment conveyed by a particular word is contingent upon the target or aspect that is being considered. An adjective such as "short" can have positive connotations with respect to "waiting times" for a restaurant, on the other hand the same adjective is assumed negative when describing something such as the "battery life"

of a product.

### 2.1.2 Common Evaluation Metrics

| | $y = A$ | $y \neq A$ |
|---|---|---|
| $\hat{y} = A$ | true positive ($tp$) | false positive ($fp$) |
| $\hat{y} \neq A$ | false negative ($fn$) | true negative ($tn$) |

Table 2.1: Confusion Matrix for the binary case of some class label, $A$. $y$ represents the true label while $\hat{y}$ represents the predicted label.

Two commonly extrapolated metrics from which other measures are typically derived are precision and recall. Given some class $A$, the former is a ratio of correctly labelled instances to all instances labelled $A$ whereas the latter compares the amount of correctly labelled instances to all instances of class $A$ present in the data, which is analogous to the accuracy for class $A$. Formally, based on the definitions in table 2.1, the two measures are given by:

$$precision_A = \frac{tp_A}{tp_A + fp_A} \tag{2.1}$$

$$recall_A = \frac{tp_A}{tp_A + fn_A} \tag{2.2}$$

For the case with $C$ classes, the total number of instances for a class $c$, $N_c$ is equal to $tp_c + fn_c$. The prevalent metric for accuracy that is reported in the literature, equivalent to the micro-averaged recall, is thus computed by:

$$accuracy = \frac{\sum_c^C tp_c}{N} = recall^{micro} \tag{2.3}$$

Where $N$ is the total number of instances in the data. Using this micro-averaged metric, however, is not necessarily the most accurate indicator of a model's performance in a classification task, particularly when the dataset that is being utilized is heavily biased to one specific class. Care must be given in the training phase of any machine learning model to ensure that the model is exposed to all classes in

question in a balanced way. This is because in computing the micro-average, the weighting scheme is distributed across all instances in the dataset, as opposed to the classes. A more sophisticated metric that is robust to this issue is the macro-averaged F1-score which equally distributes the weight across all classes as opposed to instances [42]. The macro-averaged F-measure is given by the harmonic mean of the, macro-averaged, precision and recall for a specific class. For some class $A$, this is given by:

$$F1_A^{macro} = \frac{2P_A^{macro}R_A^{macro}}{P_A^{macro} + R_A^{macro}} \tag{2.4}$$

Training a model heavily on one specific class, or not enough on another could lead the model to classify the majority of test samples to the biased class or being unable to correctly classify the class that has been under-represented in training, since the model would not have gathered enough information to discern this class. In the case where the testing dataset would be imbalanced towards the same class, the overall accuracy would lack the sufficient information expected as a metric to illustrate the effectiveness of the model to classify samples into the correct class since the model would have been trained in a biased way towards the class that is prevalent.

As an example, a frequently cited benchmark dataset is presented in [19], this dataset consists of 6248 training and 692 test phrases collected from twitter, each annotated with a particular sentiment (negative, neutral or positive) towards a specific target that appears in the tweet. Within both the training and testing subsets of this dataset, there are twice as many neutral instances as there are positive and negative instances. Works such as [11], [19] correctly point out the shortcoming of accuracy as a valid performance metric in situations such as this, and cite macro-averaged F1 scores in their results.

### 2.1.3 Manual Feature Engineering

Initially, the conventional approach involved manually extracting the most expressive and information rich features from sentences that would subsequently be processed through some statistical model such as a Support Vector Machine (SVM) for classification.

This entailed the formulation of processes by which to obtain these features, and was normally preceded by some form of normalization of the original data before these features could be extracted. Typically many types of these features were used in conjunction, each intended to extrapolate differing particularities about a specific aspect of the text, such as whether a specific token represented a noun or an adjective, or details about the words surrounding it.

The capacity of the SVM had been demonstrated on the general task of sentiment analysis in works such as [52], as well as other tools such as, bag-of-words, part-of-speech tags and other morphological and syntactical features and external resources such as linguistic parsers and sentiment lexicon, employed in works such as [19], [69], [49].

However, as [63] point out, these methods would implicitly impose an external dependency on the system. Moreover, within the context of social media, where conventional rules of language are often times regarded rather as guidelines, various studies question the applicability of dependency parsing techniques that rely on a particular degree of formality, or structure, within the content itself [63], [11]. Nevertheless these features have proven their worth when used in conjunction with powerful models such as the aforementioned SVM [36] [70], as well as neural networks [19], [69], in predicting sentiment polarity.

Even in the work that followed, focusing on increasingly autonomous feature extraction methods and more sophisticated deep learning architectures such as the Long Short Term Memory (LSTM) model, [63] make note of the competitive results obtained by the SVM approach in [36] when compared to their implementations.

### Drawbacks

Although works such as [36], [70]) obtained encouraging results, much of the subsequent literature recognizes that these results were exceedingly contingent on the choice of features [63].

Although the manual feature-based approach fared well in their work, [63] suggest that features of this kind lack the required resolution of detail that would accurately capture the interplay between target and context. The features that had been used had sound rationales behind them, however devising these rationales was in itself becoming increasingly time-consuming. One reason for this is scalability; with the increase of data available, more considerations and specifics must be accounted for when manually devising these features.

As alluded to by [79], with the aforementioned increase in labor involved, these approaches could be regarded as a bottleneck in terms of performance of these models and the wealth of data available. A more autonomous solution that would accurately capture the intricacies of language from an expansive wealth of text at a deeper level, not contingent on a proportionally large amount of labor-intensive manual feature-engineering, was desired to further advance TSA.

## 2.2 Word Embeddings

Word embeddings seek to tackle the non-trivial task of accurately capturing as much of the intricate details that are inherent in language, as possible. To model the intricacies of a word within the context of a particular language corpus, sophisticated models are employed to construct continuous and real-valued vectors for each word. The resulting vectors are commonly referred to as word embeddings, and are meant to be numerical representations of contexts in which each word is used [7] [44] [53] [65].

By first learning a continuous word vector embedding from data [7] [44] [53], most approaches can take advantage of the principle of *compositionality* [20] to

obtain a sentence or indeed a document level representation for a myriad of downstream NLP tasks, including sentiment analysis.

Two of the most prominent word embedding models that are currently employed in many NLP tasks are *word2vec* [44] and *GloVe* [53]. More recently, an extension on the former, termed *fasttext* [10] is also garnering considerable attention within the field, distinguishing itself from the other models mentioned through the use of sub-word information.

The two principal methods for learning such distributional word representations are count-based and prediction-based, each with their own strengths and shortcomings, and neither being clearly superior to the other in every aspect. The former typically involves performing dimensionality reduction on a co-occurrence count matrix, whereas the latter is derives word representations from learning to correctly predict words or contexts within a bounded, moving, window. [53]

Both methods provide a powerful means of autonomously extracting expressive and meaningful features from a wealth of text to the degree that would be unfeasible through manual means alone due to the staggering complexity inherent in language itself as well as the sheer volume of data that is being constantly made available through various online platforms.

One of the benefits that is had from constructing a vector space with distributional representations of words is the ability to model features such as similarity between the constituent words and subsequently group words with similar meanings which expands a downstream model's comprehension of a language.

[43] Matrix factorization methods, such as Latent Semantic Analysis (LSA) [17] as cited in [53], attempt to extrapolate significant statistical information pertaining to a specific corpus, from decomposed matrices that are typically obtained through low-rank approximations. Although techniques such as LSA do this effectively, [53] note a lackluster performance in word-analogy tasks, which ultimately depend, in large part, on accurately capturing the aforementioned similarity between words in the vector space.

Models trained using matrix factorization methods are efficient at exploiting statistical information, and are typically easier to parallelize albeit at a higher initial memory investment involved in storing the co-occurrence matrix [53]. [44] work on the assumption that more related words will tend to appear closer to each other than not, making window-based approaches ideal for word analogy tasks that benefit from accurate word-similarity modelling.

Since predictive methods rely on a bounded context window of some specified width when making predictions a trivial disadvantage that presents itself immediately with this approach is the limited capacity to efficiently exploit redundancy in the data on at a macro-scale. This is due to the fact that these methods operate on the level of their current context window as opposed to the document as a whole [53].

### 2.2.1 Word2Vec

[44] note that the tendency in the field of NLP was to regard words in an isolated fashion as opposed to considering each word within the scope of a distributed space where more in-depth information could be encoded regarding the relationship between words. Models, such as the popular n-gram model where prevalent in the literature due to their simplicity and effectiveness. [43] note that these methods had largely peaked as unlike distributed representations, they were limited in their capacity to model relationships, such as similarity, between words.

There was a need for more sophisticated, and scalable, techniques to address the bottleneck that was presenting itself in the lack of accurately labelled data that was being made available for training models in fields such as Automatic Speech Recognition (ASR) and machine translation. As these models emerged to tackle increasingly large data sets, simpler models, such as the aforementioned n-gram model, were consistently outperformed by neural networks using distributed representations of words in the form of word embeddings. Moreover, neural networks were shown to capture the linear relationships between words more accurately than

previous methods that used LSA, while maintaining a higher level of scalability when compared to LDA [44].

Some of the initial work making use of neural networks to learn word embeddings include [14], later refined in [15]. Their approach consisted of a feedforward neural network tasked with predicting the correct word within a context window of five words including the target word itself.

[44] introduced the Continuous Skip-gram and the Continuous Bag-Of-Words (CBOW) models in an effort to extract word embeddings from large corpora. As illustrated in figure 2.1, the former aims to learn an adequate representation of a word by training to predict the words that are most likely to surround it while, inversely, the latter was trained to predict a specific word given its context, both using a feed-forward neural network with the non-linear hidden layer removed.

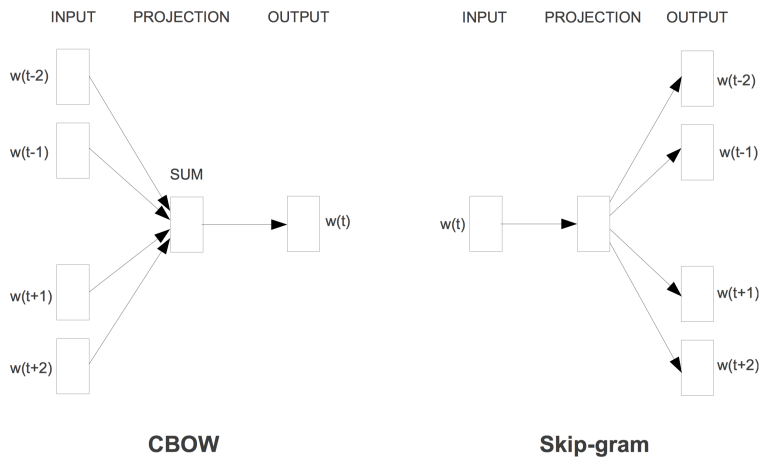


Figure 2.1: The differences between the model architectures proposed by Mikolov in [44]. The Continuous Bag-Of-Words (CBOW) approach predicts a word from its context and, conversely, the skip-gram model predicts the context from a word. [43]

Their work brought about a novel evaluation strategy which measures the expressive capacity of a word vector space through complex multi-dimensional comparisons that went over and above previous scalar metrics that were typically limited to the distance or angle between word vectors. As evidence of the level of sophistication obtained in the representations that were generated using the

*skip-gram* model, the authors note that the composition of vectors for words such as "Germany" and "capital" results in a vector that closely resembles the word "Berlin". Additionally, more sophisticated linear translations could be modeled such as "Madrid" - "Spain" + "France" is resulting in a word vector closest to that of the word "Paris" [45].

While the skip-gram model required no dense matrix multiplications which made it substantially more efficient than most of the neural network implementations that had preceded it, in their experiments they note that the quality of the resultant word vectors could be improved by increasing the window size, however this would carry with it a corresponding increase in computational complexity. Nevertheless, [44] demonstrated that sufficiently expressive vector representations of words could be obtained from large corpora of data without the need for computationally expensive models.

Following their initial work, [43] later expanded on their *word2vec* models, introducing the negative sampling algorithm to improve the efficiency by which the model learned word vectors while proposing a method for accounting for phrases through a simple data-driven approach whereby particular phrases composed of multiple words were treated as a singular token, and trained-for as such.

As noted by [44], the performance of their models were contingent on a set of design choices, most critical of which include the training algorithm, vector size, sub-sampling rate and the size of the training window. They conclude that the optimal configuration for these parameters varies based on the task being tackled.

### 2.2.2 Global Vectors for Word Representation (GloVe)

Using co-occurrence statistics to extract continuous representations of words within a large corpus of data has been explored in NLP in works as early as [56], as cited by [10]

[53] argue that while the significance of word occurrence information within a text when learning word representations in an unsupervised manner is uncontested,

further research is needed into the mechanisms by which these statistical data generate meaningful vector representations. In pursuit of this, they propose the *GloVe* model, characterized by its use of the global, that is to say at the level of the corpus in its entirety, co-occurrence information to produce aptly-called *"global vectors"*.

[53] point out that while count and prediction based methods are not fundamentally dissimilar, since both exploit co-occurrence statistics within a corpus to obtain accurate representations, they argue for the efficiency of the former approach over the latter.

A word-word co-occurrence matrix $X$ is constructed from a vocabulary such that $X_{ij}$ is representative of the number of times word $j$ is found in the context of word $i$. This invariably makes $X$ sparse in nature, since the substantial portion of words within a language cannot be expected to occur within an equally substantial number of words.

[53] develop *GloVe* by only considering non-zero elements of the co-occurrence matrix of the corpus as a whole and as opposed to the sparse matrix in its entirety. This provides a substantial increase in speed and moreover, as their work suggests, generates more expressive representations of each word when compared to a limited window-based approach. *GloVe* was evaluated on three separate tasks, specifically word analogy, word similarity and entity recognition tasks, achieving superior results over the previous literature in all. [53]

Furthering the case for the capacity of the *GloVe* model, the comparative study carried out on the task of reading comprehension by [9] demonstrated that pre-trained *GloVe* embeddings outclassed other embeddings, including *word2vec* [44]. From their experiments, [9] continue to suggest that these embeddings, in their off-the-shelf format, also surpassed embeddings trained on the target text itself as well as, to their surprise, an expanded corpus of data extracted from a domain commensurate with that of the target text.

### 2.2.3 Fasttext

Both *word2vec* and *GloVe* are considered as models operating on a word-level by regarding the word as the atomic operand, however [10] argue that this approach is possibly sub-optimal when considering languages, such as Turkish and Finnish, where single words can have multiple morphologies, or comprise of exceedingly large vocabularies, or both. In contrast, they argue that the use of sub-word information lends itself well to these languages where the multiple morphologies of a word follow some form structure, such as specific verb conjugations.

To tackle this issue they extend the *word2vec* skip-gram model to operate at a sub-word level, adopting a bag-of-characters n-grams approach for word representation. As opposed to having each word represented by a vector, a word is represented as the aggregate sum of its constituent character n-grams.

The authors also demonstrate how a vector for an OOV word can be constructed from its character n-grams with remarkable similarity to a comparable IV word. Some of their results can be seen in 2.2, where an OOV word "microcircuit" shows positive cosine similarity (in red) to an IV word "chip" between its constituent character n-grams "micro" and "circuit".

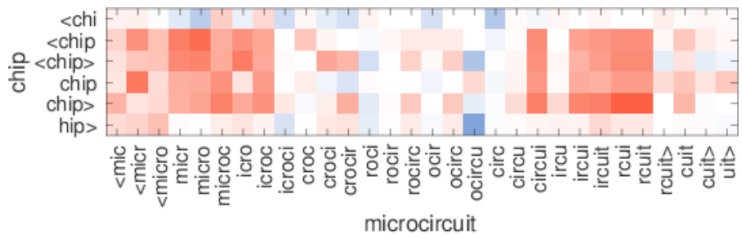


Figure 2.2: Character n-gram similarity between the OOV word "microcircuit" and IV word "chip". Positive and negative cosine similarity are denoted in red and blue respectively. Figure adapted from [10]

## 2.3 Deep Learning

### 2.3.1 Neural Networks

Modelled on the human brain, neural networks typically involve a series of layers composed of neurons. Figure 6.7 illustrates one such simplified neural network architecture with a single hidden layer, $L_2$, preceded by the input layer $L_1$, and followed by the output layer $L_3$. $(x_1, x_2, x_3)$ represents a three dimensional input vector, whereas the neurons, or hidden units, of the of the network are depicted as $(h_1, h_2, h_3)$. The firing action of each neuron is expressed using a non-linear activation function. This action propagates from one neuron in a layer to all other connected neurons in the subsequent layer and is modulated by a particular weight that characterizes each intra-neural connection. At minimum, these networks will incorporate an input and an output layer that will encapsulate one, or more, hidden layers.

The network is able to learn, or model, a function by adjusting the weights to minimize a some measure of error towards a particular objective function. [24]

The hidden layers of a neural network architecture are able to extrapolate features at a level that cannot be carried out manually, thereby capturing more subtle details and generating richer representations of the data being learned. Foregoing this need for extensive manual feature engineering is one of the primary drivers of neural network models, even though these are approaches are typically black-box solutions, with obfuscated inner-workings.

#### Activation Functions

Three of the most commonly used activation functions are sigmoid (eq. 2.5), hyperbolic tangent (eq. 2.6) and Rectified Linear Unit (ReLU; eq. 2.7). Although the choice of activation function is typically heuristic, [23] as cited in [77] notes that the ReLU is easier to compute when compared to the other two and also tends to converge faster, all while maintaining similar or even better performance.

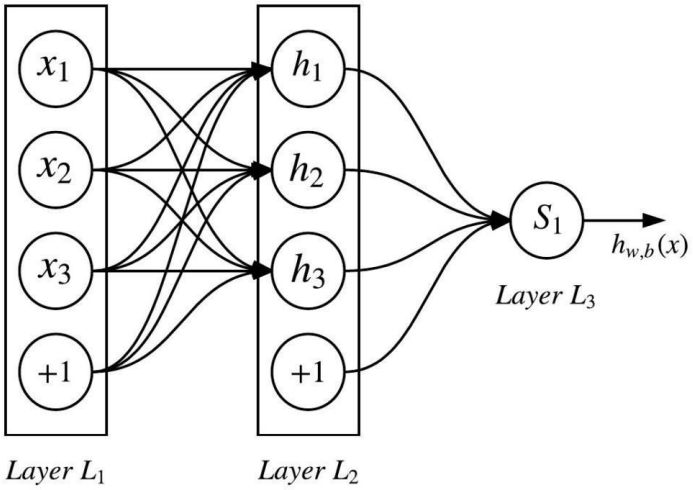


Figure 2.3: Standard feedforward neural network (FFNN) architecture [77]

$$f(W^t x) = sigmoid(W^t x) = \frac{1}{1 + e^{-W^t x}} \tag{2.5}$$

$$f(W^t x) = \tanh(W^t x) = \frac{e^{W^t x} - e^{-W^t x}}{e^{W^t x} + e^{-W^t x}} \tag{2.6}$$

$$f(W^t x) = ReLU(W^t x) = \max(0, W^t x) \tag{2.7}$$

### Training

Neural networks are trained by following the direction of the gradient that lessens the measured error rate of some objective function which is calculated through the repeated application of the chain-rule. The process can be carried out on the training set in its entirety, commonly referred to as batch learning, or in an on-line manner, carrying out updates after each training sample. The latter is known as stochastic gradient descent and is known to be more efficient and robust to local minima when dealing with large datasets [38] as cited in [24].

There will always be noise in the sampled training data that does not translate to the real world test data scenarios and that the model must therefore avoid learning. Training models to the point of becoming overly-sensitive to this noise

is referred to as over-fitting the data, and there are a number of regularization techniques often employed in the literature while training to counteract this.

One common approach is introducing some form of weight penalties, for instance, $L1$ and $L2$ regularization. Other techniques frequently used include *early-stopping*, whereby a fraction of the training data is used as a validation set that the model is tested against at regular intervals during training to test for a sustained improvement [25], and *dropout* [61], in which random neurons in a NN are ignored ("dropped") with some probability $p$, effectively training a diverse range of "thinned" versions of the original NN. An approximated average of those networks is subsequently used as final trained model by scaling its weights by that same dropout probability $p$.

### 2.3.2 Deep Neural Networks

"Deep" neural networks are constructed by stacking a series of layers in such a way that salient features extracted from one layer are passed on as input data to the next layer, and so on. This stacking process, in theory, improves the capacity of the network to extract more abstract and expressive features that reside at a deeper level within the input data. [77]

In recent years, the astounding progress in computing resources such as GPUs and high performance distributed computing have made deep learning accessible on an unparalleled level. Consequently, this has driven interest in the applicability of deep learning architectures such as the CNN and RNN in a range of fields from computer vision to natural language processing [76], [15].

The use of neural network models such as [37] [69] [49] became increasingly widespread within the field of NLP, including targeted and aspect based sentiment classification tasks [19]; [73]; [64] [63].

The most prevalent architectures are the Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNN) and the descendants thereof. The Recursive Neural Network (Rec-NN), which has been employed in works such as [58] and

[59] for syntactic analysis and sentence sentiment analysis respectively. [77]

### Convolutional Neural Networks (CNNs)

The layers typically comprise of filters, also referred to as kernels, of differing widths that slide over the input data to extrapolate various features. Each of these features would be representative of a diverse set of aspects of the data that would aid in defining it with respect to the task being tackled.

These convolution layers are coupled with a max-pooling layer with the intention of extracting the most salient values. These values can subsequently be forwarded to another convolution layer that presumably extrapolates deeper, more abstract features. This process can be repeated a number of times across multiple convolution layers to build a deep CNN that eventually produces a single feature vector representation of the original input data. Figure 2.4 illustrates this process using six filters with three difference widths and two diverse operations for each, the results of which are down-scaled using max-pooling, and subsequently passed through a softmax function for binary classification.

Max-pooling is ideal for producing a fixed-length representation of the data, which is a common prerequisite for classification tasks, while at the same time preserving the most prominent features from the original data. [67]

### CNNs in TSA

Some of the first CNN-based approaches that paved the way for the growth of CNN architectures in the literature that followed were [15], [35], and [47]. [35] used CNN architecture for sentence classification tasks ranging from subjectivity and question type with promising results albeit in the face of a number of challenges that became evident to the authors. Not least of these was the limited capacity for the CNN architecture to capture syntactical dependencies in sentences that occurred over long distances. This challenge was the one of the foremost drivers for the work that followed by [47] who developed the DynamicCNN (DCNN) which consisted of

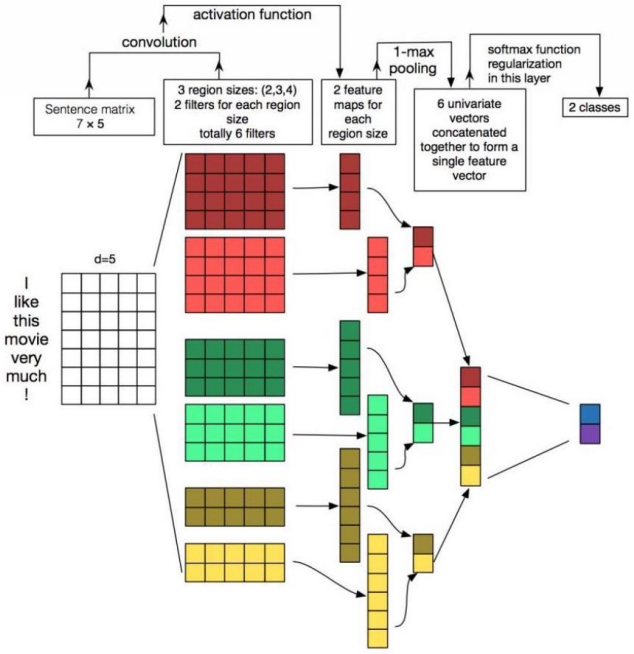


Figure 2.4: An example of a CNN architecture used for sentence modelling and subsequent binary classification [75] as cited in [67].

a series of hierarchical convolution and k-max pooling layers.

Other works cited in [67] include [60] that dealt with sarcasm detection on twitter data, noting a need for additional context information when dealing with short texts of this nature. This observation is also echoed in [31], who noted superior performance of CNN networks when dealing with longer text that would provide more contextual information as opposed to shorter text. Due to the vast amount of parameters that CNNs typically need to learn, scarcity of data is an often cited challenge [67].

Work carried out by [12] made use of a CNN to deduce the sentiment of the target based on the sentiment of the clause surrounding it. As noted by [11] however, this method still operated on the assumption that the most salient features of a word can be extracted from other words in close proximity.

A phrase such as "I bought a mobile phone, its camera is wonderful but the battery life is short, not particularly satisfied overall", challenges this assump-

tion with respect to the "mobile phone", as the intended target since the most sentimentally-laden words appear at the opposite end of the sentence.

### Recurrent Neural Networks (RNNs)

An RNN can be thought of as a chain of recurring modules which typically represent elements within a variable-length sequence, with an internal hidden state that represents the network's "memory". This state is passed forward from one module in the chain to the next.

Unlike a CNN, where each layer has its own set of trainable parameters that must be learned, a RNN uses a single set of parameters across all of the modules in the chain which significantly diminishes the total number of parameters that it must learn.

Through forwarding the hidden state from one time step in the chain to the next, the network is able to "remember" information from previous elements of the sequence and use that information when generating a representation for the current element [63].

The hidden state that characterizes RNNs acts as its "memory" element and makes these networks particularly effective in dealing with data that are sequential in nature. One of the most prominent examples of these data is language, where the significance of a word at one time step may be substantially altered by those that preceded it. Consider, for example, the word "dog", for which the meaning would shift entirely, from an animal to a popular American snack, should it be preceded by the word "hot" [67].

Moreover, the capacity of RNNs to model variable length sequences to a fixed length representation also makes them particularly practical when dealing with different units of resolution in languages including documents, sentences, and even words, all of which are naturally arbitrary in length. [65]

### Vanishing & Exploding Gradients

In the setting of a traditional RNN, the tendency for a gradient to exhibit exponential change to the degree that prevents substantive learning increases with the length of a sequence [8] [26]. This phenomenon is what is implied by the terms "vanishing" or "exploding" gradients, where the changes observed over time steps often vanish with time or, although less frequently, but with equally devastating results, grow exponentially; hindering the network's capacity for learning.

One class of solutions to the vanishing or exploding gradient problem is to carry out particular modifications on top of the standard stochastic gradient descent algorithm, such as gradient clipping, whereby the norm of the gradient vector is *clipped*, or using second order derivatives, which may or may not be influences to a lesser degree. [13] The second, and more popular, class of solutions look instead to introduce further sophisticated, additive, non-linearities to the traditional RNN unit. These would selectively carry forward salient features and filter out irrelevant information from previous time steps, as opposed to overwriting the memory content at each time step.

Variants on the standard RNN network were proposed to address the vanishing and exploding gradient issue. The most popular of these being the Long Short Term Memory (LSTM) and, more recently, Gated Recurrent Unit (GRU). Other approaches include, but are not limited to, Residual Networks (Res-Net).

### Long-Short-Term-Memory (LSTM)

LSTM [26] is an extension on the RNN model that addresses the issue of a vanishing or exploding gradients through the use of gates that control the flow of information from the past states to present states.

To do this, the LSTM introduces three adaptive gates that can be considered additional neural layers on top of the single neural layer that characterizes the typical RNN. The layers introduce an increased level of sophistication in the "remembering" process from one sequence point to the next.

These gates are commonly referred to as the input, forget and output gate. Moreover the LSTM also maintains a two inner states as opposed to one, namely the cell state and the hidden state.

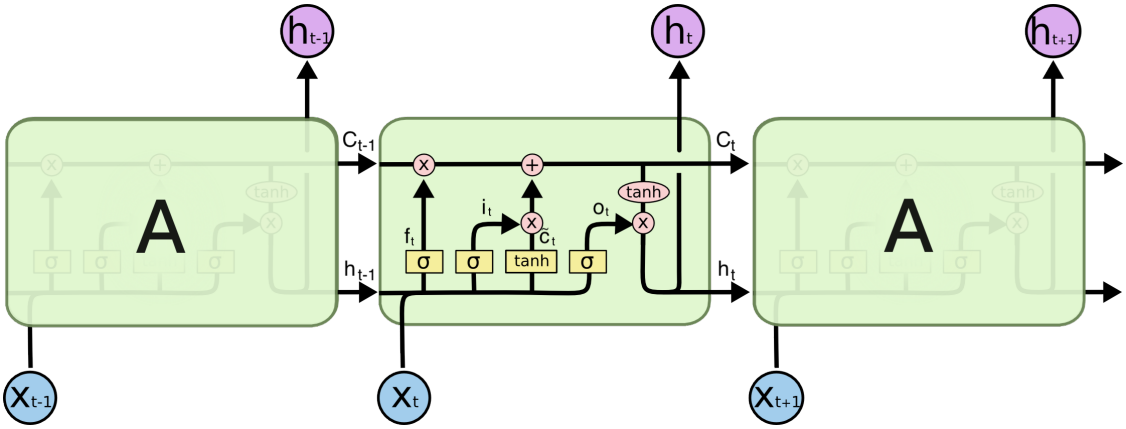


Figure 2.5: LSTM repeating module illustrating the four neural layers (Yellow Boxes) that comprise it. Point-wise vector operations are depicted in red boxes. Image adapted from [1]

The process undergone in each repeating module of the standard LSTM architecture depicted in figure 2.5 starts at the forget gate. Here, the information to be removed from the cell state $C_{t-1}$ is selected through the sigmoid layer (eq. 2.8).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.8}$$

Next, an update vector is produced through a point-wise multiplication operation between the input gate (2.9), which represents the values selected to be updated, and a vector of candidate values, $\tilde{C}$ (2.10). This update vector is added to the current cell state, from which the forget gate had previously removed information deemed redundant.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.9}$$

$$\tilde{C} = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.10}$$

Finally, an output gate (2.11) regulates the amount of cell state information that

is output to the rest of the network through the unit's hidden state $h_t$ (eq. 2.12).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * \tanh(C_t) \tag{2.12}$$

The gating mechanism that is present in the LSTM network allows for the persistence of salient features that are encountered early in the sequence and which would otherwise be overwritten in a typical RNN architecture.

The input and output gates of the LSTM control the amount of memory content that is to be added to the memory cell and the memory content that is exposed to the rest of the network respectively.

Forget gates were later added to the architecture [22], which enabled the memory cells to reset themselves. It is important to note that the LSTM updates the memory cell independently from the forget gate, which is to say that, for the LSTM to "remember" a prior input in a sequence two conditions must be satisfied; the input gate must be closed, preventing the addition of new information and, the forget gate must be maintained open, as otherwise this would cause the existing memory content to be reset.

### LSTMs in TSA

[73] achieved results competitive with those obtained by [47] by using an LSTM, as opposed to CNN, to generate representations of tweets. [67] notes that their model was able to extract features over long distances at a fraction of the complexity of the DynamicCNN [47].

Another variant on the LSTM architecture involved the use of dependency parsing to build tree-structured LSTMs which operate on some specific tree-pattern that is extracted from the data, typically through the use of external parsing tools [59]. Approaches of this variety [30], [32], [80] have obtained promising results, however as [11] note, these are contingent on the data being well-formed structurally

and grammatically, which is not guaranteed when dealing with micro-texts on social media platforms.

Two of the foremost extensions on the classical LSTM model when tackling target-based sentiment analysis, were proposed in [63] with the intent of accounting for the target information, these were termed the Target Dependent LSTM (TD-LSTM) and Target Connection LSTM (TC-LSTM). Their work showed that the changes proposed would improve the performance over a standard LSTM. [63] generate representations of a target's left and right contexts, where the target representation is concatenated to each. These two are then finally chained together to form the final, target-specific, representation of the sentence which is fed to an LSTM for sentiment classification [18].

TC-LSTM [63] was reported as one of the first neural network based methods to obtain state-of-the-art performance without the use of laborious feature engineering and external data sources. It is worth noting however that recent work [3] failed to reproduce the original results that were reported.

### Challenges still facing LSTMs

While the issue of long-distance dependencies and CNN based approaches is often brought up, [11] remarks that LSTMs will still struggle with features that are located too far apart within a sequence, given the phrase "Except Patrick, all other actors don't play well", an LSTM would struggle to identify the positive sentiment on the opinion target "Patrick" due to the distance between the terms "Except" and "don't play well".

From the observation made by [5] when employing LSTM-based models in the field of machine translation, [11] make the case that the TD-LSTM would suffer from in instances where the most sentimentally salient word is located further away from the target being considered.

### Bidirectional LSTMs (BLSTMs)

When dealing with bounded sequences, a natural conclusion one might draw is that valuable information can be extrapolated from future contexts as well as past contexts. Bidirectional RNNs are employed with this specific intention in mind.

Building on top of the LSTM architecture, a BLSTM is so called as it is the result of two stacked LSTMs, each responsible for processing and extracting features from a sequence in opposite directions. Features extracted from these two directions are then typically concatenated into a single, theoretically richer, and more expressive, representation of the sequential data.

It has been argued that the bidirectional nature of these models violates causality, which is justified to an extent, since a future context cannot be exploited if it is the same element that is being predicted. An example of this is predicting future stock market fluctuations. However, for tasks where this information is available, bidirectional variants that make use of this information have consistently been shown to outperform their uni-directional counterparts. [24]

Sentence and document level sentiment classification tasks lend themselves well to the use of bi-directional RNN (BRNN) variants, since the boundaries of the sequence being classified are well-defined a priori. In their work, [11] suggest that using a BLSTM improves the ability of their approach to extrapolate phrase-like features when compared to a uni-directional sequential approach.

An improvement in results stemming, in part, from taking a bi-directional approach is also observed in [78], in both accuracy and macro F1 scores when compared to the work of [69], from which the former was inspired. More recent works (eg. [41], [79]) that have reported notable results in the field also take advantage of bi-directionality in their approach.

### Gated Recurrent Units (GRUs)

The GRU does away with the cell state that is found in the LSTM, maintaining only a single hidden state as its "memory". The second way in which the GRU

differs from the LSTM is in its number of gating units, by foregoing the output gate and combining the input and forget gates into a single update gate, the GRU benefits from having less parameters to learn.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{2.13}$$

As the name implies, when the reset gate (eq. 2.14) nears 0, it allows the GRU unit to drop the previous information, which may be deemed inconsequential at a later time, and reset itself with the current input only. This information is then used by the GRU to produce a vector of candidate activation values (eq. 2.15).

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{2.14}$$

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t] + b) \tag{2.15}$$

Instead of maintaining an internal memory cell, the GRU uses the update gate and carries out a linear interpolation function (eq. 2.16) to control the amount of candidate activation, $\tilde{h}_t$, that is added to the previous activation. In this way the update gate serves as the primary mechanism preventing the GRU from overwriting a previously encountered salient feature.

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{2.16}$$

Whereas the LSTM utilizes the output gate to control the exposure of its internal memory to the rest of the network, the GRU does not have any means by which to control the exposure of its inner state, and therefore exposes its hidden state in its entirety to the rest of the network.

Each unit in a GRU model will have separate update and reset gates that will independently learn to capture long and short term dependencies in a sequence. Units that learn to capture longer-term dependencies will have active update gates whereas, for short-term dependencies, the reset gates will tend to be more active.

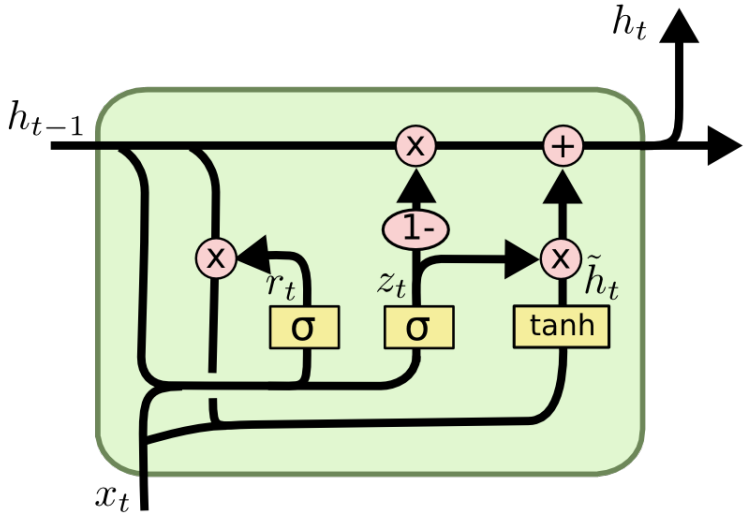


Figure 2.6: Internal gating structure of a GRU unit. [1]

**GRUs in TSA**

[78] model the interplay between targets and their contexts through gated mechanism operating on the a representation of the original sentence split into three components using a gated-RNN.

[40] use a GRU as part of their approach, serving as a gating mechanism to determine whether to update a specific part of memory based on past activations. They claim state of the art results in aspect detection and sentiment classification, outperforming SenticNet [41], and foregoing the need of external knowledge.

[74] use gated tanh-ReLU units to control flow of features from convolutional neural networks to max pooling layer. They deal more with the advantages that a CNN offers, since it is not time-dependent and is therefore easier to parallelize.

[27] generate a sentence representation by stacking two GRU based networks, similar to the BLSTM configuration, opting to go with the GRU architecture instead since it has less parameters to learn, noting similar results. A continuous vector representation of the target is sandwiched between the two GRU direction outputs, and fed to softmax classifier. Authors report accuracy and macro-f1 scores better than state of the art obtained on a twitter dataset [19].

### LSTM vs GRU

Since the architecture of the GRU is less complex than that of the LSTM, there may be circumstances where the former may be more efficient than the latter since it requires less parameters to learn [11], [27].

That being said, there is no clear winner between the two variants, as demonstrated by work conducted in [13]. The results obtained, shown in figure 2.7, concluded that the only demonstrable advantage is that of both variants over a standard RNN architecture. The authors go on to say that the performance of the two variants themselves is contingent on the particular nature of the tasks being addressed. Although it is worth noting that the work carried out by [13] was not specifically in the field of NLP, the deciding factor between these variants in NLP literature still tends to be heuristic [67].

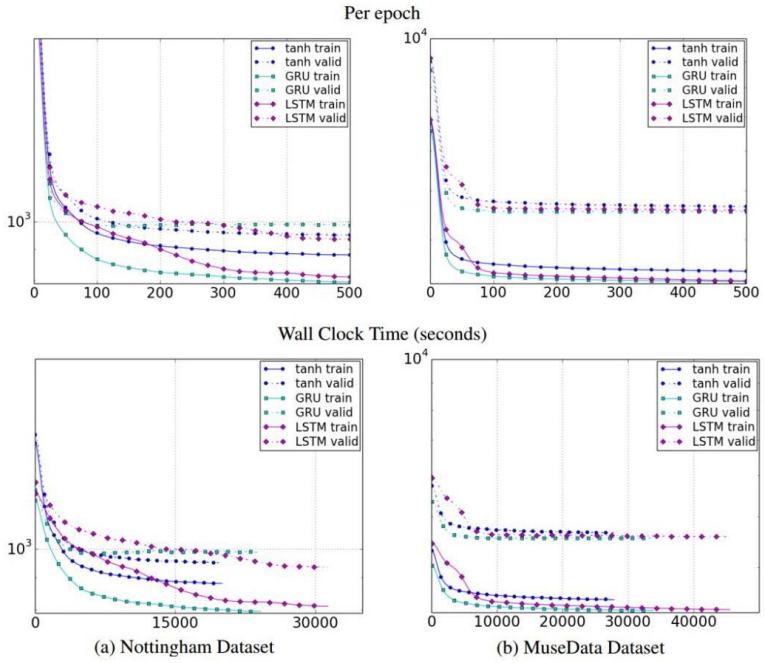


Figure 2.7: Results obtained by [13] during training and validation of different RNN variants illustrating the superiority of LSTM and GRU units over the traditional RNN.

#### CNN vs RNN

In spite of various studies illustrating the fact that CNN networks are data heavy in nature and often require auxiliary data when dealing with micro-texts such as those obtained from social media networks such as twitter, [67] note that assuming the superiority of RNNs for sequential data is inaccurate.

They proceed to cite studies where CNNs performed competitively, and even surpassed RNN-based architectures in tasks for which the latter would, in theory, be better suited, such as language-modelling [16] as cited in [67].

It is worth keeping in mind that when dealing with language modelling, RNNs and CNNs approach the task from separate avenues; while RNNs, and their descendents, have no context boundaries when processing sequential data to generate a representation, CNN-based approaches seek to extrapolate the most meaningful n-grams from bounded-contexts from that sequence, to produce their final representation. [67]

### 2.3.3 Attention

The intuition behind an attention mechanism is that, when dealing with sequential data, different parts of the sequence contribute to varying degrees towards a specific task or goal. This is abundantly apparent in the field of machine translation, where the attention mechanism made its debut [5]

When translating a particularly long sequence, for example, it is natural to focus principally on specific regions related to the current element being translated, as opposed to the source sentence as a whole, as this would result in the most relevant parts possibly being shadowed by unrelated material.

The remarkable power of attention mechanisms for machine translation [5] sparked an interest in investigating their applicability in a range of other tasks, including target, and aspect, based sentiment analysis [73] [11] [18] [79], image captioning [34] and question answering [33], where the notion of the most salient

features being dispersed unevenly across the source data also holds true.

In its simplest form, an attention mechanism involves a layer which produces a weight vector that represents a distribution of salience across an original feature vector, which would otherwise be considered evenly in its entirety, with respect to some target. The nature of the target varies depending on the downstream task, in targeted or aspect based sentiment analysis this is typically the target or aspect in question, whereas in fields such as machine translation this could be the last hidden state that was output.

This process typically takes the form of some scoring function, such as a simple FFNN, which can be trained alongside the rest of the model, followed by a softmax layer. Given a series of representations $[h_1, h_2, ..., h_n]$ and some target $t$, the scoring function $a$ calculates the salience of each $h_i$ with respect to $t$. Subsequently, the softmax layer squashes the attentions scores into a valid distribution vector $\alpha$ with values in the range $(0, 1)$.

$$\alpha_i = \frac{\exp(a(h^i, t))}{\sum_{j=1}^{n} \exp(a(h^j, t))} \tag{2.17}$$

This weight vector is then used to produce a context vector $c$, as the weighted sum of the original feature vector. This process will amplify features with high attention weight values (approaching 1) while attenuating features with low attention weight values (approaching 0).

$$c = \sum_{i=1}^{n} \alpha_i h_i \tag{2.18}$$

### Attention in TSA

ATAE-LSTM [73] was one of the first to incorporate attention into an LSTM model for the purposes of aspect-based sentiment analysis. A vector representation of the target aspect is used as the subject of attention, allowing the model to attend to different parts of a sentence with respect to the aspect.

[11] make use of multiple attention layers to extrapolate the most salient, and sentiment-bearing words with respect to a target, suggesting that through more than a single attention layer, the model would be more effective at extrapolating features over long distances. They subsequently aggregate these attention results non-linearly using a GRU. The authors attribute their preference of a GRU over an LSTM for this stage in the process due to the former requiring less parameters than the latter.

[18] argue that prior works had focused primarily of the representation of contexts and not on targets themselves. When considering targets that consist of multiple words, the idea that words should not necessarily contribute equally to the final representation of that target is a valid assumption to make. They cite the example of "picture quality" as a target and argue that in such a case the word "picture" would play a more important role that "quality". To address this, their Interactive Attention Network (IAN) incorporates two attention networks to model both the target and the context interactively, to obtain a representation of the effect each had on the other.

More recent work, by [79], build on the intuition of producing better target representation as well as context representation. Their LCR-Rot model is characterized by a novel "rotary attention mechanism" that attempts to better capture the interplay between a target and its context as well as the contexts on the target. They argue that left and right contexts affect the target representation to a degree that merits a separate representation of the target for each, one which is "left-aware" and another that is "right-aware". [79] demonstrate the effectiveness of their rotary engine approach citing state-of-the-art results in accuracy on three distinct datasets, suggesting that properly modelling the effect of the target on the context may be as important as that of the context on the target.

### Attending Both Ways

Recently, work carried out by [72] showed an interesting performance boundary on approaches to targeted sentiment analysis that make use of attention mechanisms. In particular, [72] note that when context words have diametrically opposing sentimental bearings based on the target being considered, this cannot be modelled by improving by attending to the context alone.

To illustrate this, [72] consider two different targets, *price* and *resolution*, in four different phrases; "high price", "low resolution", "high resolution" and, "high price". Some sentiment score, $s$, for these phrases, where $s < 0$ implies a negative sentiment and $s > 0$ implies a positive sentiment accordingly (eq. 2.19). The attention weight $\alpha$ will have a value of 1 since the context consists of a single word which is represented by $h$. $v$ represents the target and $W$ is the weight matrix the model must learn.

$$s = W(\sum_{i=1}^{n} \alpha_i h_i + v) = W(h + v) \tag{2.19}$$

From equation 2.19, the following inequalities can be obtained,

$$W(h_{high} + v_{price}) < 0$$
$$W(h_{low} + v_{price}) > 0$$
$$W(h_{high} + v_{resolution}) > 0$$
$$W(h_{low} + v_{resolution}) < 0$$

Expanding these inequalities results in $Wh_{high} < Wh_{low} < Wh_{high}$, which is a contradiction, and can therefore not be learned by the model. [72] point out that this condition cannot be rectified through further attending to the context but rather ameliorating the representation of the targets $v$ to better capture the complex relationship they have with the context in a way that possibly reverses the polarity of the final sentiment score $s$.

Towards this end, [72] experiment using a series of techniques, and note an

improvement over previously cited results as well as a direct improvement on the RAM model [11] when these are coupled with their optimally performing target-sensitive context modelling strategy.

### 2.3.4 Memory

It can be argued that one of the most prominent contributions of the attention mechanism covered in the previous section is the ability to selectively read information, typically from the internal state of a model, in a differentiable manner by reading from all of the data, to varying degrees. This, however, has more profound implications: the same process can be applied to selectively write in a differentiable way. Memory networks are so-called as they exploit this fact, providing an external memory store that a model can learn to refer to and update over time.

This concept was first actualized in two distinct, yet coincident, studies: [2], who proposed their Neural Turing Machine (NTM) and [28] who put forward their Memory Neural Network (MemNN) framework.

In their seminal work, [28] describe their memory network framework as comprising of some tangible memory component, which is represented as an encoded continuous matrix. This external memory is updated through the use of neural network operations which selectively read and write to and from it. They proceed to conceptualize these operations in the form of four fundamental components.

The first component, the input component, $I$, is tasked with mapping incoming data to an internal representation. Pre-processing and embedding look-ups are two examples of operations that may be entailed at this stage. A generalization component, $G$, follows, which uses the data from $I$ to update the memory. $G$ is so-called as it allows for the potential of generalization of existing memory towards some future goal. In its simplest form however, this component simply stores incoming data in the next available memory slot, leaving existing memory unchanged. Using the input data from $I$ and the current memory state an output component, $O$, produces an output feature vector which typically involves inferring the most

relevant memories. This output is finally interpreted by a response component, $R$, to the desired format.

Each component that makes up this process may represent any trainable model such as an RNN or SVM, and trained accordingly. In their original approach, [28] use "hard" attention when probing for the most relevant memory evidences, where the number of highest scoring evidences is a tunable parameter. [62] build upon this idea by opting instead to use a softmax operation, which is conceptually comparable to using a "soft" attention mechanism over the external memory store. Moreover, unlike its "hard" counterpart, this makes the process differentiable, making the model trainable in an end-to-end fashion, requiring less supervision when compared to [28].

This framework put forward in [28] and subsequently extended by [62] served as the foundation from which most memory-based targeted sentiment analysis approaches emerged.

Furthermore, [62] show that the performance of their model can be enhanced by having the $O$ component repeatedly attending to memory for a number of consecutively stacked "hops". Indeed, this observation is echoed in subsequent studies inspired by their work in the field of targeted sentiment analysis (eg. [64], shown in figure 2.8), the intuition being that through these consecutive hops the model is able to attend to richer, more abstractive, features than those existing solely on the surface level of the data.

Memory networks were first put forward towards the goal of targeted sentiment analysis by [64] (figure 2.8). Their approach was inspired by [62], with some differences in the attention function that was used, opting instead to use the method put forward in [5]. The authors reported results outperforming the state-of-the-art SVM-based model by [36], which required extensive manual feature engineering, absent from their proposed memory network, as well as the far more complex LSTM based models [64] which required substantially more time to train. Similar to [62], the authors noted a marginal increase in performance as they increased the number

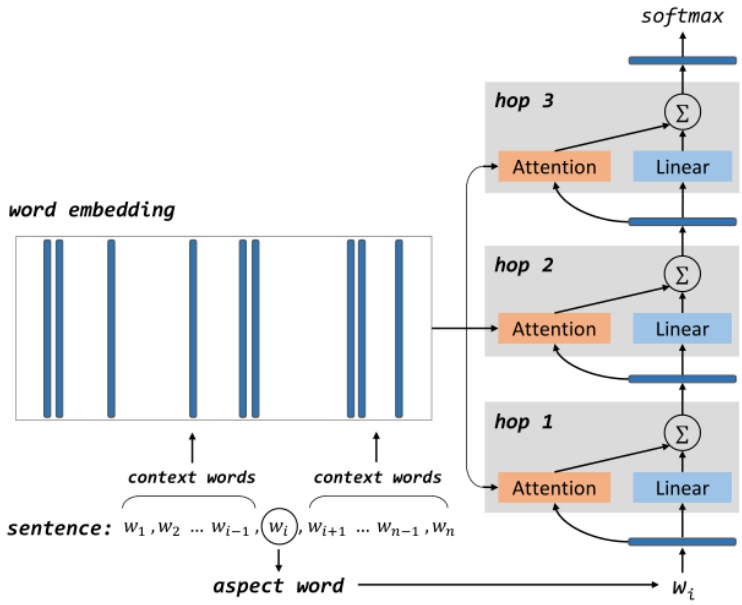


Figure 2.8: The deep memory network approach for targeted sentiment analysis, with 3 hops. The model stored the context of a target in memory and repeatedly attends to that memory with respect to the target word $w_i$. [64]

of computational hops, which capped at around 8 hops.

[11] construct a location-weighted memory module from the hidden states of a BLSTM placed between the input and the attention modules so as to better capture information from phrases comprising of multiple words, such as "not wonderful enough". Moreover, unlike the [64], they combine the results of their recurrent attention layers non-linearly. These improvements led to a performance boost over [64]. The proposed model architecture is illustrated in figure 2.9. It is also worth pointing out that the authors failed to reproduce the results originally cited in [64].

[40] maintain a memory chain of entities that are encountered while processing a phrase and develop a gating mechanism that determines how those memory chains should be updated. The gating mechanism accounts for the content and the location of the memory chains compared to the current input, as well as the past activations, through the use of a GRU unit, when choosing to update a particular memory element.

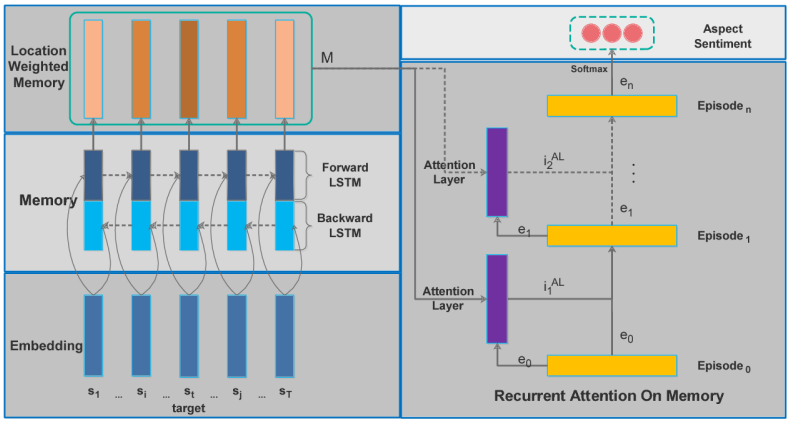


Figure 2.9: The recurrent attention model architecture showing the placement of the location-weighted memory module with respect to the input and attention layers. As opposed to [64], the attention values are combined from one episode to the next non-linearly, using a GRU. [11]

# 3. Design

In pursuit of the objectives laid out in chapter 1 we develop a framework that is centered around TSA. Limiting the scope of the framework to the field of TSA allows us to extrapolate the time-consuming processing that typically comprise an experiment, and provide an API which exposes this functionality to any TSA model built using the framework, reducing the time required to implement and evaluate each model. Moreover, this approach isolates the evaluation environment from the model, making it consistent across experiments using different model implementations.

To account for numerous forms that TSA model approached can adopt, user-extensibility is an essential component of the API design, providing a means of adding functionality at both the data preparation and model evaluation stages.

Finally, it is imperative that a clear delineation is made between aspects of the framework with which the user may interface, and other internal elements which should be abstracted from the end-user, which facilitates experimentation by making the framework more intuitive to use and reduces the risk of costly run-time errors.

This chapter details the two primary modules of the framework, following each seminal component through the experiment pipeline and the respective design choices made in accordance with the aforementioned principles.

## 3.1 Data Module

The primary task of the data module is to produce the features data for the experiment module. We sought to adopt a modular design paradigm since this allows different responsibilities to be delegated to the appropriate sub-component.

Each sub-component of the data module is responsible for writing and indexing data to an external file-system using a specific hashing policy when this data is first generated. In subsequent experiments that require this same data, the sub-component is tasked with querying the file-system for its availability and importing it, instead of having to reproduce it.

Maintaining this indexed data store on the external file-system has the benefits of speeding up computation for subsequent experiments that use the same data with parametric alterations, facilitating investigation of the data produced at intermediary steps, aiding in debugging and allowing the framework to isolate the minimum requirement of data that to upload for experiments carried out through online services.

### 3.1.1 Dataset Import Script

It is imperative to evaluate the performance of TSA models on datasets that have different characteristics, as a measure of the applicability of the model under different scenarios, or for the purposes of fine-tuning a model to a specific domain. Datasets from different platforms and domains vary in their vocabulary and other vernacular features, such as colloquialisms. Different platforms also provide samples of varying lengths and which may express different sentiments towards multiple targets in the same sample.

#### Raw Data Parsing Function

The import script simplifies the process of importing new datasets to only requiring a, typically short, parsing function which reads the raw data files and mutates the

data structure to records consisting of four fields: sentence, target, sentiment label and, offset[1]. The parsing function is an inevitable requirement since it is unfeasible to account for all the possible data formats of prospective datasets, the specifics of this function are explained in §4.1.

Once imported, the dataset is stored and indexed on the file-system for future use under a user-defined name. The imported data is stored in two dictionaries, for training and testing, and exported as binary pickle[2] files, which provides the fastest access times at minimal storage cost.

### 3.1.2 Dataset Component

The dataset component is primarily tasked with loading dataset files which have been imported as well as generating the accompanying information files used at other points in the process. Moreover, the component is also responsible for creating and exporting re-distributed variants of datasets, serving as a wrapper around these operations with which other sub-components in the data module can interact.

The corpus file that is generated by the dataset component provides a record of the occurrence count of each unique term in the dataset and is subsequently used to filter the vocabulary of an embedding to the minimal set of required terms. The *.json* file that is generated provides information on the distribution of the dataset, later used by the experiment component in producing a visual representation of this information. Two variants of each file are created, for the training and testing subsets of the dataset respectively.

#### Dataset Re-Distribution

It is often the case in datasets typically employed for TSA that one or more sentiment labels are over-represented in the data, indeed, this phenomenon is common

[1] required to identify the correct target in situations where it occurs more than once in the same sentence

[2] python serialized object file

across all the datasets gathered for this work. This is accounted for at the output stage by using appropriately weighted performance metrics, as detailed in §5 .

This notwithstanding, such an imbalance in classes causes the model to be over-exposed to a particular sentiment at the cost of others, which may be detrimental to the learning process. To address this issue, while also providing a means of investigating the effects of these imbalances, the framework provides a means of re-sampling the data, either to balance the distribution across all classes, or to any custom distribution as specified by the user.

The implementation details of the algorithm developed for this purpose are provided in §4.2.

### 3.1.3 Embedding Component

As with the dataset component, the goal of the embedding component is to provide a wrapper around reading and writing to and from an indexed embeddings directory by specifying a singular embedding type that exposes all the required data to allow the other sub-components to interact with it, facilitating the integration of additional embedding sources later in the development stage.

The commonly used embeddings are identified using a pre-defined shorthand, which the embedding component takes as input, along with a comma-delimited list of filters to apply on the embedding vocabulary. Function filters are specified using a custom name that is assigned when creating the filter. The "corpus" keyword is reserved within this scope and applies a set filter on the embedding vocabulary using the corpus of the datasets being used for the experiment. The embedding variable is initially obtained using the *GenSim* tool, which provides a means to download some commonly used embeddings. Once the embeddings are downloaded, the data files are stored and indexed within the embeddings directory of the framework, enabling offline access.

### Filtering Embeddings

If one or more filters are specified, the embedding component extracts the vocabulary of the source embedding, unifies the requested filters into a single pipeline, runs the source vocabulary through this pipeline, and finally re-constructs the embedding on this filtered vocabulary. The resulting data is exported and indexed, along with a JSON file with the details of the filtration process and, in the case of function filters, a CSV report detailing which terms were filtered and what condition was met that resulted in this term being filtered.

The first type of filters that the embedding component supports are set filters, which simply define the set of terms to include from the entire embedding vocabulary. These filters are primarily used to limit the size of the embedding to contain only terms that occur in a dataset corpus, substantially reducing the file-size overhead of the embedding. The other type of filters that are supported are function filters, which refer to a function that, for each term, returns true or false as to whether to keep or discard it. These filters can be used to remove vocabulary terms based on particular POS tags and were implemented in an effort to investigate additional methods of reducing embedding sizes with a minimal cost to downstream performance.

Processing an entire embedding vocabulary to obtain NLP tags can be significantly time-consuming due to the size of these vocabularies. In an effort to optimize this process, a base function-filter is implemented from which the end-user extends, to construct custom function-filters. This base class automatically exposes the NLP attributes requested by the user while also allowing the framework to gather all filters into a singular pipeline, used to determine the minimum amount of processing needed to obtain the attributes that this pipeline requires. The implementation details of this procedure are detailed in §4.3.

In order to run multiple experiments using cloud computing services, all of the required data for these experiments need to be uploaded for each job. Apart from maximizing the ability to share data between experiments, reducing redundancy,

the final size footprint of this data needs to be minimized to obtain feasible upload times. Since embedding variables were the largest bottleneck in this regard, set filters were used to reduce the size of embeddings to the bare necessity. Although the full embedding vocabulary would still be used at a production level, this reduction in embedding size also provided a marked speedup in look-up operations when indexing datasets, since the size of the look-up table was drastically reduced, resulting in faster experimentation times during development. Function filters were developed as an exploration of other techniques for further size reduction and their effect on downstream performance. The intuition behind this is that the contribution of subsets of vocabulary, such as adjectives and nouns, would suffice within the scope of TSA.

It is essential that the filtering mechanism be intuitive so as to enable extensibility by the end user. The framework grants access to NLP attributes of each particular term within the scope of a function filter. This design allows for a simple means of constructing custom filters that can operate on POS tags and other language characteristics without the need of manually extracting this information from the vocabulary.

Filtered versions of an embedding are stored under a single directory based on the source embedding from which the filter originated. Indexing these directories using a unique hash that is generated from the filter specification enables the embedding component to load a filtered embedding using its hash identifier if it already exists.

### 3.1.4 Feature Provider Component

The feature provider is detached from the other components of the data module as it is responsible for generating data that is mostly conditional on the specific experiment being carried out, which limits its re-usability, unlike the dataset and embedding components. Aside from managing an indexed directory of this data, the feature provider is primarily responsible for producing both the feature data

and number of auxiliary files. The purpose of the latter includes saving the state of data at different stages of the process, providing valuable information used when creating some of the output visualizations, as well as logging diagnostics on the process, for debugging and profiling purposes.

The steps that the feature provider takes to generate the required features are as follows: first, in the case of multiple datasets, these are merged and encompassing corpora are constructed for training and testing, second, text pre-processing and tokenization are carried out producing string tokens to be mapped to integer indices. A look-up-table is finally constructed from the vocabulary of the embedding and a TensorFlow graph of look-up operations is built to map these tokens to their respective indices within the embedding. Upon completion, the results of the graph are converted and exported into a proprietary TensorFlow Record (*TFRecord*) format to be consumed by the experiment module.

As inputs, this component requires an embedding object, one or more dataset objects, and the specific OOV policy that is to be adopted when constructing mapping tokens into features for the experiment module. The indices mapped to each token are contingent on whether OOV tokens are considered at all, solely during training, or whether a particular bucketing strategy is used. If OOV tokens are ignored, this needs to be addressed at the tokenization stage by excluding these tokens. In the case where OOV tokens are only considered at the training stage, the process is similar, excluding only OOV tokens found in the test data, whereas when using a bucketing strategy, no tokens are excluded, and the mapping process is handled accordingly by the look-up table operation. Moreover, the mapped values for OOV tokens vary based on the OOV policy that is adopted, including the number of OOV buckets that are used, since this directly alters the hashing function that assigns the bucket to each token. All of these factors are taken into consideration and handled internally by the framework's own hashing mechanism, creating new data when required and re-using any computationally expensive data where possible.

Although trivially simple, the mapping process can become increasingly time-consuming for larger look-up tables, as the look-up ops would be required to traverse more data as a consequence. The initial approach to the problem was to optimize the python code responsible, through the use of generators and parallel processing libraries. These benefits from this approach were quickly exhausted however, and the approach proved to be unreliable, requiring fine-tuning based on the nature of the data being processed. To address this, the process was instead implemented using TensorFlow itself which is particularly adept at optimizing parallelizable operations by first expressing them in graph form, and subsequently executing handling resources and parallelizing operations wherever possible during graph execution, providing more consistent performance. Furthermore, the framework also generates a *filtered* vocabulary file containing only the tokens that occur in the datasets being used, and their corresponding index in the larger embedding vocabulary which is used for building a look-up table at a fraction of the size of the original embedding vocabulary, for the exclusive purpose of mapping the tokenized datasets being considered.

### Merging Datasets

As previously alluded to, different datasets are characterized by the different attributes and vocabulary that they provide based on the limitations of the platform from which they are sourced, such as the character limit on Twitter, or a specific domain around which they center, such as reviews of a specific type of product. A mechanism by which these datasets could be merged allows us to capture a wider range of these characteristics in both training and testing data and evaluate the performance of different TSA approaches across these variances.

### Text Pre-Processing

Spacy[3] was used for all text-processing and tokenization tasks. One of the primary reasons for using this tool was its native and intuitive support for parallelization. Secondly, its proprietary language model format enables selectively loading sub-modules based on run-time requirements, improving performance by avoiding superfluous processing.

Text normalization is kept to a minimum since the exploration of these techniques is beyond the scope of this work. First, for embeddings with case-insensitive vocabularies, all tokens are lower-cased accordingly. Furthermore, a default filter function is implemented which omits URLs and email addresses, and which is detached from the feature provider, facilitating extensibility by the end user.

### Output Data

The most salient outputs of the feature provider are the *TFRecord* files containing the feature data for the experiment module. These features are exported to separate directories for training and testing data and further partitioned, following TensorFlow guidelines for ensuring integrity when batching and randomizing this data through their data pipeline API. Other important outputs of the feature provider are the vocabulary files that can be used to visualize the embedding using TensorBoard and a JSON reference file detailing the datasets and embeddings as well as OOV policy employed. The latter includes details on the OOV initialization function and a list of the OOV buckets and their constituent tokens, providing insights into the behavior of the bucketing function[4].

Finally, similar to other components, data at multiple stages are exported to pickle files for two reasons. Firstly, this foregoes the need for carrying out repetitive processing. Secondly, by loading these files at different stages of the feature generation process in an online setting, where computationally expensive tasks are kept

[3]Public NLP library for python, accessible at https://spacy.io/

[4]The default is a basic string hashing function defined internally by TensorFlow

to a minimum, the feature provider can maintain its internal state, preserving the value of all of its class members. Indeed, this is desired for all components; maintaining an identical internal state in both offline and online environments reduces the risk of costly online experiment failures resulting from run-time errors.

## 3.2 Experiment Module

At a high level, all operations that take place in the experiment module are governed by the experiment class, which simultaneously serves the purpose of maintaining an indexed directory of experiments while also preparing and dispatching commands to run those experiments. In this context, preparing an experiment implies setting up the directory for a new experiment, bridging the feature provider and model being trained and, loading any environment configuration settings for TensorFlow and scaffolding the experiment with the necessary code for tools that are included in the framework, such as comet-ml.[5]

The experiment class automatically organizes the experiment directory first by model name, and then either by the specific name for an experiment which is provided by the user, or using a name automatically generated from the experiment parameters provided. Enforcing this approach has two benefits, first, experiment directories are self-documenting since they convey the details of the experiment, secondly, the framework will use the directory and continue training for an existing experiment with the same parameters by default.

Finally, since each experiment also serves as a self-contained model, the experiment class is also responsible for exporting trained models from an experiment. Once exported, trained models can be evaluated using real-world data. Using this data as a means of ensuring the validity of the process as a whole was the motivation behind implementing this functionality.

[5]For more information on this tool refer to §5.3

### 3.2.1 TSA Model Base Class

The Targeted-Sentiment-Analysis (TSA) Model base class serves as the abstraction layer between the end-user and the internal infrastructure of the framework. The implementation adheres to the principle of least privilege, exposing only the core constructs of a TSA model while handling the integration of that model with the rest of the frameworks' features internally. The core constructs for TSA model development are the default parameters of the model and the tensor operations it performs. A third, optional construct, covers any feature pre-processing where feature mutations can be specified that take place before invoking the model definition function. Exposing only these core constructs greatly simplifies development, enabling the user to focus on the model being developed. Moreover, this ensures that the rest of the framework components, particularly those concerning evaluation remain pristine and consistent across different model implementations.

The parameter definition function is the simplest of the three, only required to return a dictionary of default parameter values which may be overridden at runtime before being made available within the scope of the model definition function.

The default feature set is a dictionary of literal string tokens and their respective embedding IDs for target and their contexts (left and right separately). An optional pre-processing function can be specified to mutate the samples to prior to consumption by the model function so as to obtain the features desired by the end-user. This design choice was made to decouple the pre-processing logic from the model definition, making it possible to inspect and test the process in isolation. Typical pre-processing operations may include concatenating the target to the left and right contexts.

The feature pre-processing function must return a dictionary of features, which the framework internally re-couples with the label data. For this reason, the pre-processing function may only mutate the structure of each sample, maintaining the length and order of the features as a whole, otherwise this could result in features being assigned incorrect labels. Token string data is required for certain

downstream visualizations, such as attention heat-maps, where token IDs are not sufficiently informative. It is worth noting that the string literals are not automatically mutated, and maintaining integrity between IDs and string literals must be handled by the end-user.

Finally, before exposing these features to the model definition function, the framework automatically appends `*_emd` and `*_len` entries for each `*_ids` key on the dictionary, representing the embedded sequences and their length, respectively.

The model definition function defines the tensor operations to carry out on the feature data, producing logits for each sample. It exposes four parameters within its scope: a dictionary of features, their corresponding labels, a flag denoting whether the model is being trained, evaluated or running as a prediction service, and finally, a dictionary containing the resolved set of model parameters. Using the produced logits in conjunction with a loss function and optimization technique, which are also defined in this scope, the function must return a call to an in-built class method that generates the "estimator spec" object which encapsulates the model's behavior.

Adhering to the design specifications described herein, the framework is able to internally plug into the TSA model and handle the necessary code infrastructure adding functionality such as online collaboration, various downstream evaluation visualizations and introspection tools, automatic embedding of sequence data, support for Google cloud services and finally, a streamlined process for exporting the trained model as a prediction service.

Certain features, specifically those centering around evaluation methods, such as attention heat-maps, are only applicable to a subset of model types. To address these situations efficiently an add-on mechanism that could isolate particular functionalities and automatically make them accessible to all TSA models was developed.

### 3.2.2 Model Add-On API

The principal motivation behind the add-on API is to enforce a design pattern where a TSA model implementation consists of code concerned solely with the definition of the model. Moreover, due to the importance of visualizations for evaluation purposes, it became apparent that this process would need to be streamlined to meet our objective of facilitating rapid experimentation. The add-on mechanism achieves this by allowing code concerned with specific functionality to be isolated, preventing repetitive code, reducing development time and facilitating testing. Furthermore, while the TSA model class provides out-of-the-box functionality, in some aspects it could be considered a black-box approach, limiting the end user. The add-on API aims to address this by providing an entry point for customization at a deeper level while still maintaining a layer of abstraction between the user and the framework's internals.

Each add-on functions as a pass-through for the estimator spec object generated by the model definition function, enabling the user to inject code before running the model. These commonly take the form of wrappers around hooks, a native Tensorflow construct. Hooks enable the execution of custom code at different points in the model life-cycle such as specific steps during training or evaluation. The applications for these hooks include producing custom visualizations, logging additional metrics and enabling techniques such as early stopping, to name a few. Within the scope of an add-on function, the user has access to all of the same variables available in the model definition function, including the model instance itself, as well as the incoming estimator spec object, which must be returned, so as to either be passed on to the next add-on or execute the model. There are no requirements on the internals of the add-on or any custom hooks that the add-on wraps around, however, it must conform to a function signature to integrate with the framework. A number of examples of this signature are provided in the `addons.py` file along with custom hooks implemented specifically for the purpose of our work in the `hooks.py` file.

An addon decorator is provided which takes as input an array of one or more add-ons to attach to the model definition function it *decorates*. Add-ons run in the order they are attached, after the model definition function is called and an initial estimator spec object has been produced. Additionally, each add-on can be instructed to run only in a subset of scenarios, specifically: training, evaluation, and prediction. Finally, each add-on can be deactivated at runtime through when invoking a new task using the CLI. This is useful when submitting batch jobs to cloud service where there is no access to the codebase in between tasks and add-ons can only be deactivated programmatically between tasks.

## 3.3 Summary

We have discussed the design principles that were considered at each sub-component on a conceptual level, to address the objectives described in §1.4.

Furthermore, we have presented how each of these sub-components assemble to establish a framework which simplifies the process of implementing new TSA models and provides a static evaluation environment with robust metrics and informative visualizations, illustrated in chapter 5. First, however, chapter 4 shall expound on how the design principles outlined herein were actualized into code at particular stages of the framework.

# 4. Implementation

This chapter presents a closer look at the implementation details of the tool-set the framework provides, as well as the interface used to interact with these tools and carry out experiments. Crucially, while the development of these tools is motivated by our own requirements in carrying out experiments with the framework, it is pursued with the overarching goal of providing an encompassing framework architecture that streamlines their use over a series of experiments, using a simplified user interface.

## 4.1 Importing Datasets

The framework provides a CLI command as an entry point for importing new datasets. The command signature requires a path to a directory containing three files at a minimum: a "train" and "test" file and an accompanying parser Python[1] script. This API additionally exposes arguments for specifying a custom identified under which to index the imported data, as well as a specific parser in situations where multiple parsers are provided.

The parsing script will typically consist of two stages: reading and mutating the raw data. The first is contingent on the format of the raw data being imported and can vary in complexity from simply reading from a text file to more complex op-

[1]Python3 syntax is expected

erations on proprietary data formats. The second stage generally involves the use of generators and looping structures to iterate through raw samples and produce the arrays the framework expects. This stage also presents an opportunity for the user to incorporate any conditional filtering or transformations, such as excluding "conflict" labels from raw datasets. The function must produce three arrays consisting of the sentences, targets, and labels. A fourth array, of character offsets of each target, may also be returned. This is optional for datasets where each target appears only once in each sentence, but required otherwise, as the framework would have no means of differentiating between multiple target occurrences in a sentence.

The signature that is enforced by the framework on the parsing function stipulates that it expects solely one parameter without a default value, which is reserved for the path of the file to be parsed. For most cases a single function will suffice, however in some instances, multiple functions may be required for improved separation of concerns. In these situations, both the parsing script and the function that serves as the primary entry point within it must be named according to the provided `parser-name` CLI argument.

Initially, the framework scans the directory for the relevant source files which contain the words "test" and "train" in their names. The parsing script is loaded based on the provided parser name argument or the default parser file-name. Once all required files are located, the framework inspects the parsing script and ensures that the parsing function has the expected signature. Provided all these tests are passed, the framework proceeds to run the parsing script on the raw data then stores the result in its own hashed index of datasets under the user-provided identifier for future use.

This mechanism for importing datasets provides the widest range for support of various data types while maximally extrapolating the code responsible for this operation, thereby simplifying the process for the end-user. Furthermore, the framework is abstracted from the dataset importing process thereby limiting the scope of errors that may occur during the process to the parsing script itself, which is

easier to debug by the end-user as opposed to the internal code of the framework.

Finally, the structure of this approach allows for the possibility of defining multiple parsing functions, each of which may carry out processing or filtering operations prior to importing the dataset, while also delegating the responsibility of storing and accessing these different versions of a dataset to the framework.

## 4.2 Dataset Re-Distribution

The primary consideration when re-distributing a dataset is minimizing the number of original samples that are discarded to maintain as much of the size of the original dataset as possible without selecting duplicate samples from the source.

**Algorithm 1:** Dataset Re-Distribution

**Input:** Source Class Label Counts $\rightarrow$ **counts**

**Input:** Target Class Label Distribution $\rightarrow$ **targets**

**Result:** Redistributed Class Label Counts

`// ...argument validation`

1 **counts**$'$ $\leftarrow$ **counts** $\odot$ **targets**

2 **counts** $\leftarrow$ (counts$'$ $= 0$) ? $\infty$ : counts

3 **while** $\min(counts) \neq \infty$ **do**

4 **smallest** $\leftarrow$ (counts $= \min(counts)$) ? counts : 0

5 **totals** $\leftarrow$ (smallest $\neq 0$) ? $\lfloor \frac{smallest}{target} \rfloor$ : $\infty$

6 total $\leftarrow \min(totals)$

7 **candidates** $\leftarrow \lfloor$ total $\odot$ **targets** $\rfloor$

8 **counts** $\leftarrow$ (candidates $>$ counts) ? counts : $\infty$

9 **end**

10 **return candidates**

Ternary operations are applied element-wise

The first stage of the process validates the arguments provided, ensuring that

a valid distribution is provided.[2] Since the limiting factors in this process are the smallest (most scarce) count numbers, an infinitely large marker must be used to remove elements from consideration. The algorithm(1) starts from the smallest count number as a reference point, from which candidate counts are calculated, and subsequently compared to the original number of samples for each class. If all candidates are less than the number of samples available, the redistribution is considered valid and the algorithm terminates. Otherwise, the process is repeated using the class for which there are insufficient samples as the new reference until all candidate counts are valid.

One drawback of this approach is the substantial reduction in the magnitude of the original dataset that can occur in heavily skewed datasets that contain a small number of samples for some particular class. The amount of samples that will be discarded is directly proportional to the class that is the most scarce in the dataset and its corresponding target distribution in the new dataset.

## 4.3 Filtering Embeddings

The Spacy NLP library defines three different pipes that each are responsible for obtaining POS, NER and Dependency-Parsing tags respectively. A POS pipe, for example, assigns a *pos_* attribute to each token with its corresponding POS tag which can subsequently be used by the end-user as a filtering criterion. These piping operations can carry a heavy computational cost when running on exceedingly large embedding vocabularies. This motivated the development of a filtering process that is intuitively extendible by the end-user while adhering to the core tenant of the framework of maintaining as small a computational overhead as possible to facilitate rapid experimentation.

First, set filters are applied to reduce the size of the embedding, thereby reducing the amount of data each pipe would need to process. All the requested

[2]Omitted from the algorithm for brevity

function-filters are loaded using a lookup registry which allows the user to specify a custom string identifier for a filter. The framework then encapsulates all function-filters into a single function and injects code to enable logging throughout the filtering process. Throughout the process, the framework deduces which pipes are required based on the collective set of filtering criteria specified by all filter-functions, and only runs these pipe operations to obtain the requisite NLP tags and filter tokens accordingly.

Partial functions are provided that conform to the signature that it expects from a function-filter to integrate these functions into a single function and deduce the corresponding pipes required. This makes the process of defining custom function-filters as simple as extending these partial functions, providing the attributes or tags the user intends to filter on. The framework expects an array of tags or attributes that are to be included by default, unless prefixed with a "!" character which conversely implies exclusion based on the same character or tag.

The sample code below illustrates the implementation of two filters, each requiring the POS pipe and therefore extending the `_pos_pipe_filter` partial. The first excludes all proper noun tokens whereas the second stipulates that only adjectives, adverbs, nouns and, verbs should be allowed.[3]

```
no_proper_nouns = partial(_pos_pipe_filter, tags=["!PROPN"])
pos_set_one = partial(_pos_pipe_filter, tags=["ADJ", "ADV", "NOUN", "VERB"])
```

During the filtering process, the framework tracks all tokens that are being filtered along with the corresponding user-provided filter-function. This information is tabulated and exported as a CSV file that is stored in the framework's hashed index of embeddings alongside the filtered embedding, for future reference. The primary motivation behind the filter report file is to enable the end-user to inspect the filtering process at a granular level, providing insight into the tokens being filtered along with the function-filter responsible.

[3]The complete annotation specification can be found at https://spacy.io/api/annotation

### 4.3.1 OOV Policies

The OOV policy of an experiment describes the mechanism used to deal with OOV tokens during feature generation. This policy is controlled by two parameters, an `oov` flag, and a `num_oov_buckets` parameter, which is an integer that instructs the framework to use a bucketing strategy and how many buckets to use.

There are three valid configurations for these parameters described as follows,

1. `oov` = `false`

   In this instance, all OOV tokens in both training and testing datasets are discarded.

2. `oov` = `true`, `num_oov_buckets` = 0

   When using no bucketing strategy, distinct vectors are initialized using a customizable initialization function for all OOV tokens in the training dataset. OOV tokens from the testing dataset are discarded to simulate real-world environments.

3. `oov` = `true`, `num_oov_buckets` $\geqslant$ 0

   All tokens, in both training and testing datasets are considered, with OOV tokens being assigned a bucket using a native Tensorflow hashing function. Similar to (2), vectors for each bucket are initialized using a customizable initialization function.

A list is maintained of which tokens to consider during tokenization, which controls which OOV tokens are discarded. Lookup operations during experimentation requires a vocabulary file which is generated with the tokens and their respective indices in the embedding, including any bucket entries. To speed up experimentation, a second, filtered, vocabulary file is generated containing only indices of tokens relevant to the experiment, thereby reducing the look-up operations' computation time.

This approach implies that any changes made to the `num_oov_buckets` or the `oov` flag will alter the content of the aforementioned files and require the process to be repeated. Changes to the OOV initialization function, however, do not, since vector initialization for OOV tokens and buckets takes place within the scope of the experiment module.

## 4.4 Datasets

The following section presents a brief overview of the datasets that were chosen to be integrated into the framework. We focus on the principal factors that were considered when choosing these datasets with the intention of maximizing the diversity in these attributes. These include the mechanism by which the datasets where originally obtained and annotated, the respective platform from where they were sourced, any specific domain around which they center and finally we present any salient statistical information that characterizes the datasets.

### 4.4.1 Dong Twitter Dataset

[19] is one of the most frequently cited datasets for TSA, used in works such as [63], [11] and [79]. This dataset comprises tweets focusing on a diverse range of topics that were obtained through the twitter streaming API by querying for targets such as "bill gates" and "xbox". This implies that, in each instance, the target string occurs explicitly in the tweet. The respective sentiment of each tweet towards its target is manually annotated as "positive", "neutral" or "negative", and the authors report a fair inter-annotator agreement score of 0.82. The final dataset consists of 6248 training samples and 692 testing samples, each having a distribution consisting of twice as many neutral samples as positive and negative combined. As some studies [71] [57] that followed point out, a notable property of this dataset is that each sample limited to a single target. It is also worth noting that due to the skewed nature of the class distribution, it is imperative that a

metric that accounts for this is reported, such as the Macro-F1, when evaluating performance on this dataset.

### 4.4.2 Saeidi Yahoo Answers Dataset

In their work, [57] combine the tasks of detecting the existence of a particular aspect as well as the sentiment with respect to that aspect within pieces of text obtained from "Yahoo! Answers" concerning specific neighborhoods in or around the city of London. The authors suggest that this text tends to be less constrained than that originating from social media platforms, and that this enables them to obtain reviews that mention multiple neighborhoods per instance with differing sentiments. The final dataset is limited to samples that contain up to two targets and these are manually annotated using binary labels ("positive" and "negative") across eleven separate aspects such as "dining" and "shopping". These aspects do not necessarily occur within the text itself and are therefore inconsequential within the scope of TSA. This notwithstanding, a "general" aspect category is also considered, used to denote a *general* sentiment towards a particular entity or target, and is the most frequently annotated category. We consider only this subset of samples from this dataset and only use the train and test portions, discarding the development portion. It is important to note that all original target mentions within this dataset are obfuscated through the use of "location markers" which eliminates any potential context information conveyed by the original target. The final datasets consists of 1770 sentences, 78% of which are labelled positive. To the best of our knowledge, the baselines implemented in this work have not previously been evaluated on this data.

### 4.4.3 Wang Political Twitter Dataset

A point that [57] stresses on in developing their dataset is the importance of considering an approach's ability to discern different and possibly opposing sentences

towards multiple targets in the same phrase. With this same goal in mind, [71] build a dataset consisting of tweets centered around UK politics, labelled according to the sentiment expressed towards a specific target entity that occurs in the tweet, using a 3-point scale. Across 4077 tweets, a total of 12587 targets are identified with just under half (6015) of those expressing a negative sentiment and an average of 3.09 targets per sentence. As pointed out in [3], there are a number of samples in this dataset where the target locations are not specified, which makes tweets where the target occurs more than once unusable since there is no way of knowing which instance is being considered. We exclude these cases for our experiments, resulting in a reduced dataset which contains 11899 target-sentiment pairs in total; with 2190 unique targets and an average of 2.94 targets per tweet.

### 4.4.4 SemEval Laptop & Restaurant Reviews Dataset

Another dataset commonly cited in the field of targeted sentiment analysis and aspect-based sentiment analysis is the Restaurant and Laptop Reviews dataset [54]. This dataset comprises 7686 sentences from two distinct domains, technology and dining, from which specific aspect terms and the polarity towards them were manually annotated. Sentiment polarity was determined on a 3 point scale with the addition of a "conflict" label. Similar to [18], [11] and [63], we discard "conflict" samples for the purpose of our experiments, which amount to 61(2.07%) and 105(2.22%) samples for the laptops and restaurants datasets, respectively. Compared to data sourced from micro-text platforms the samples in this dataset are typically lengthier and more syntactically sound. The authors note some interesting differences in the nature of the reviews obtained from the two domains such as the restaurant subset containing substantially more aspect terms and a stronger bias towards the positive sentiment compared to the laptop reviews. The majority of targets in both domains consist of a single word. Recently, [74] used this dataset for their evaluation purposed and included results for three of the five baselines implemented in our work, namely, IAN [18], RAM [11] and TD-LSTM [63].

### 4.4.5 SemEval 2015 Twitter Dataset

This dataset was specifically proposed for the task of sentiment analysis on twitter as part of the SemEval challenge [55]. A subset of the dataset which we are interested in within the confines of this study consists of a series of tweets which were collected and manually annotated with a 3-scale sentiment polarity score directed towards a particular entity term occurring in the tweet. The annotation process was carried out using Amazon's Mechanical Turk system with 5 turks, employing a majority vote in cases where an agreement could not be obtained. For the purposes of our study we make use of the training and testing subsets of the dataset and interchange these two subsets so as to have a larger training dataset as opposed to testing set, as in the previous datasets outlined. Moreover, in its original form this dataset consisted only of tweet IDs from this the original tweet can be obtained however only within a specific window. In out experiments we were able to obtain a downloaded version of this data, which included the original tweet text, from a later SemEval submission [6] from their github repository. The final dataset consists of 2872 samples, the majority of which are labelled "positive".

### 4.4.6 SemEval 2016 Twitter Dataset

Similar to the previously mentioned dataset, this was also constructed for the purposes of the SemEval sentiment analysis on twitter challenge [46], with the difference of being graded on a 5-point scale as opposed to a 3-point and also comprising substantially more samples and topics than the former. For the purposes of our work, we reduce the resolution of this dataset from a 5-point scale to a 3-point scale by grouping "very negative" and "weakly negative" samples under a single "negative" label, and similarly for "very positive" and "weakly positive", to obtain "positive" samples. The annotation process that was employed is similar to that outlined in the [55] with minor differences in the majority voting system for conflict resolution around label disagreements due to the different point-scale (the process

| Dataset | Domain | Type | Split | Vocabulary Size | Unique Targets |
|---|---|---|---|---|---|
| Dong | General | Social Media | Test | 3682 | 82 |
| | | | Train | 15037 | 113 |
| | | | **Total** | **16047** | **118** |
| Saeidi | General | Review | Test | 1231 | 2 |
| | | | Train | 1804 | 2 |
| | | | **Total** | **2257** | **2** |
| Wang | Politics | Social Media | Test | 4385 | 755 |
| | | | Train | 9706 | 1855 |
| | | | **Total** | **11211** | **2190** |
| Pontiki - R | Restaurants | Review | Test | 2199 | 553 |
| | | | Train | 4309 | 1268 |
| | | | **Total** | **5122** | **1630** |
| Pontiki - L | Technology | Review | Test | 1573 | 415 |
| | | | Train | 3638 | 1031 |
| | | | **Total** | **4090** | **1295** |
| Rosenthal | General | Social Media | Test | 3243 | 44 |
| | | | Train | 9800 | 137 |
| | | | **Total** | **11575** | **180** |
| Nakov | General | Social Media | Test | 19374 | 60 |
| | | | Train | 43859 | 100 |
| | | | **Total** | **54034** | **160** |

Table 4.1: Dataset domains, vocabulary sizes, and the number of unique targets.

is detailed in [46]). The actual tweet data for this dataset was also obtained from the Github repository of [6] due to the same limitation around this data expiring after a particular time-window. The test and train splits of the dataset were similarly interchanged to obtain a larger training dataset. The final dataset includes over 20000 training samples covering 100 topics, with the overwhelming majority being either "neutral" or "positive". Whereas the testing dataset consists of 6000 samples with entirely different topics and is skewed towards the "positive" label.

| Dataset | Target Length (tokens) | | |
|---|---|---|---|
| | **1** | **2** | **3+** |
| Dong | 2080 | 4851 | 9 |
| Saeidi | 1770 | 0 | 0 |
| Wang | 9702 | 1937 | 260 |
| Pontiki - R | 3504 | 791 | 427 |
| Pontiki - L | 1804 | 824 | 323 |
| Rosenthal | 1554 | 1145 | 170 |
| Nakov | 11450 | 13684 | 1366 |

Table 4.2: Number of targets with different lengths, in tokens.

| Dataset | Split | Samples | Negative | Neutral | Positive |
|---|---|---|---|---|---|
| Dong | Test | 692 | 173 | 346 | 173 |
| | Train | 6248 | 1560 | 3127 | 1561 |
| | **Total** | **6940** | **1733** | **3473** | **1734** |
| Saeidi | Test | 588 | 139 | - | 449 |
| | Train | 1182 | 246 | - | 936 |
| | **Total** | **1770** | **385** | **-** | **1385** |
| Wang | Test | 2541 | 1206 | 957 | 378 |
| | Train | 9358 | 4377 | 3615 | 1366 |
| | **Total** | **11899** | **5583** | **4572** | **1744** |
| Pontiki - R | Test | 1120 | 196 | 196 | 728 |
| | Train | 3602 | 805 | 633 | 2163 |
| | **Total** | **4722** | **1001** | **829** | **2891** |
| Pontiki - L | Test | 638 | 128 | 169 | 341 |
| | Train | 2313 | 866 | 460 | 987 |
| | **Total** | **2951** | **994** | **629** | **1328** |
| Rosenthal | Test | 486 | 56 | 288 | 142 |
| | Train | 2383 | 260 | 1256 | 867 |
| | **Total** | **2869** | **316** | **1544** | **1009** |
| Nakov | Test | 5868 | 749 | 1623 | 3496 |
| | Train | 20632 | 2339 | 10081 | 8212 |
| | **Total** | **26500** | **3088** | **11704** | **11708** |

Table 4.3: Dataset sizes and class distributions.

| Dataset | Average Targets /Sentence | Sentence Length (tokens) | | Distinct Sentiments /Sentence | | |
|---|---|---|---|---|---|---|
| | | Average | /Target | 1 | 2 | 3+ |
| Dong | 1 | 21.1 | 21.1 | 6934 | 0 | 0 |
| Saeidi | 1.24 | 15.2 | 12.4 | 1416 | 23 | 0 |
| Wang | 2.94 | 26.3 | 9 | 2163 | 1640 | 242 |
| Pontiki - R | 1.83 | 20 | 9.4 | 2177 | 379 | 20 |
| Pontiki - L | 1.58 | 21.4 | 12 | 1665 | 193 | 9 |
| Rosenthal | 1.01 | 23 | 22.8 | 2842 | 5 | 0 |
| Nakov | 1.01 | 23.2 | 23.1 | 26355 | 40 | 0 |

Table 4.4: Dataset sentence information

## 4.5 Executing Tasks

A task represents the start of a new experiment or a continuation thereof and is invoked using the `tsaplay.task` submodule:

```
python -m tsaplay.task single \
--batch-size=25 \
--steps=1000 \
--model="lstm" \
--embedding="wiki-50[corpus,only_adjectives]" \
--datasets 'dong[33/34/33,15/70/15]' wang \
--contd-tag="example-experiment-task" \
--model-params hidden_units=50 oov=true num_oov_buckets=100 \
--aux-config logging=false debug=cli \
--run-config save_checkpoints_steps=1000 \
--comet-api="XXXXXXXXXXXXXXXX" \
--verbosity="INFO"
```

This section covers the CLI arguments available on this command when executing a single task.

- `--steps` The number of steps to train for.
- `--batch-size` The number of samples to use in each batch.
- `--model` The name of the model being run.

- `--embedding` The name of the embedding to use for the experiment. A comma-delimited list of filters may be included, as well as the `corpus` keyword to filter on tokens occurring in the datasets.

- `--datasets` One or more datasets specified by the name used to import them. Each dataset argument may be postfixed with a comma delimited list describing of redistribution percentages for training and testing. Each redistribution must have the same number of elements as the amount of unique classes in the dataset, separated using a `/` character. If only one set of values are provided this is applied to both training and testing datasets.

- `--model-parameters` Model hyperparameters which may override the default specified in its parameters function. The OOV policy parameters are also supplied through this parameter set.

- `--run-config` Values which are passed on to the Tensorflow run configuration. These include options such as setting a random seed for an experiment and how often to save model summaries during training.[4]

- `--aux-params` Auxiliary model parameters, separate from its hyperparameters, include programmatic switches for add-ons and a flag for enabling debug mode.

- `--contd-tag` An optional name to index the experiment under. If the name already exists, that experiment will continue training up to the provided `--steps` argument.

- `--comet-api` Comet.ml API key which enables real-time remote logging of the experiment (requires a `contd-tag` argument to be set).

- `--verbosity` Controls the logging level on `stdout` during execution.

[4]https://www.tensorflow.org/versions/r1.12/api_docs/python/tf/estimator/RunConfig

A batch option was also developed to allow multiple tasks to be run in sequence on the google cloud platform without the need to re-build and re-submit a job for each task. This command is also invoked on the same `tsaplay.task` submodule:

```
python -m tsaplay.task batch "batch_tasks.txt"
```

The batch-file syntax allows the user to specify multiple tasks using the single task CLI format where each task is delimited by a new-line character. Additionally, comments may be included in the file using either the '`#`' or '`;`' characters at the start of a new line.[5]

### 4.5.1 Google Cloud Support

Making use of a cloud computing service provides additional computational power through hardware such as GPUs which reduce experimentation times through increased parallel computing capabilities. Additionally, by running experiments in a remote environment, local machines can be used to continue development on future experiments. The Google cloud platform was chosen as it integrates well with the Tensorflow library, itself being a product of the same company. Moreover, this platform also provided us complimentary credits with which we could carry out our work.

While the framework facilitates integration with the Google cloud AI platform, some pre-requisites must be met on the host machine. These include setting up Google cloud CLI tools, as well as a Google cloud project and cloud storage bucket. The specifics of this process are beyond the scope of this work and can be reviewed in the relevant Google cloud API documentation.[6]

Since the framework must be packaged as a module before submitting a job, an external script must be used to programmatically invoke this process:

```
python submit_job.py \
--job-id="example_job" \
```

[5] An example batch file is included in the source code for reference.
[6] https://cloud.google.com/ml-engine/docs/

```
--job-dir="experiments_data" \
--job-labels type=example \
--machine-types masterType=standard_gpu workerType=large_model workerCount=1 \
--stream-logs \
--show-sdist \
--task-args batch "batch_tasks.txt"
```

- `--job-id` A unique string which identifies the job being submitted. This is also used as a job-dir if one is not provided.
- `--job-dir` The name of the parent directory which will house experiment data using the same structure described in §3.2
- `--job-labels` Optional labels which may be attached to a specific job for organizational purposes.
- `--machine-types` Used to specify the specific machine configuration for the job, as offered by the google cloud platform.[7]
- `--stream-logs` A flag which, when set, continues to stream log data to the stdout as the job runs.
- `--show-sdist` A flag which, when set, logs the build process that takes place before submitting a job to the stdout.
- `--task-args` The arguments that are forwarded to the `tsaplay.task` submodule which follows the same syntax described in §4.5.

The `submit_job.py` script is responsible for packaging the framework with the requisite assets then submitting the job request to the Google cloud platform. This process must be repeated for every job that is submitted. Tasks that fall within the scope of the data module are carried out on the host machine, before packaging the framework and requisite assets. In this way, the framework can minimize the size of

[7] https://cloud.google.com/ml-engine/docs/machine-types

the assets that need to be uploaded to the cloud while storing this data for future reuse. The internal assets directory is used to store the requisite embeddings, datasets, and features for a particular job. It is packaged with the rest of the `tsaplay` directory as a distributable python module and uploaded to the Google cloud platform. This module is subsequently installed on the platform to invoke the `tsaplay.task` submodule.

At the time of writing, task execution across multiple machines in a distributed environment is considered experimental as some features, such as comet.ml logging, are known to malfunction in this setting. This is likely a consequence of machine instances submitting data to the comet.ml service concurrently. Attempts to rectify this were made, however the issue was ultimately deferred as it was beyond the scope of this work.

## 4.6 Summary

The tools described herein provide functionality which may be programmatically controlled through the same interface used to carry out experiments or submit jobs. This implementation approach adheres to our objective of providing a simplified interface, minimizing the amount of work required for successive experimentation, while still producing substantive downstream outputs.

In chapter 5 we provide an overview these outputs, from metrics and visualizations to integrations with useful analysis tools, which is followed by the results and observations from our OOV and reproducibility experiments in chapter 6.

# 5. Evaluation Methods

In this chapter we describe the different metrics and visualizations that are produced by the framework from each experiment as well as the tools that are used to consume the data that are output.

We first introduce Tensorboard, a companion to Tensorflow which a richer level of interaction between the user and the output data of an experiment produced by the framework. This is followed by an overview of the elements that comprise these data, their significance, and the ideal outcome for each within the scope of TSA.

Finally, we address the notion of collaboration when carrying out experiments using the framework and how this is addressed efficiently by internally integrating with the Comet-ML platform.

## 5.1 Tensorboard

Tensorboard is a development tool, included with the Tensorflow library, used primarily for visualizing the transient performance and behavior summaries of a model. A summary is a representation of a model at a particular checkpoint, encapsulating all the information about that model at that step in training. At each checkpoint, the model is evaluated on the test dataset consisting exclusively of unseen samples adding the respective metrics and visualization to the summary

object of that checkpoint. The add-on mechanism wraps around hooks which allow custom metrics and visualizations to be added to the summary object.

The checkpoint frequency of an experiment is an adjustable run-configuration parameter. While more frequent checkpoints increase the granularity of the transient data for a model, this also incurs a cost on experimentation times, particularly when dealing with visualizations which may need to be generated multiple times. In certain situations however, such as employing early stopping, more frequent checkpoints may help minimize the risk of overfitting.

The Tensorboard service is started using a CLI command pointing to an experiment directory, stored locally or on a google cloud storage bucket, and is subsequently accessible on a local webserver.

### 5.1.1 Scalar Metrics

Model loss is plotted during training and evaluation to identify possible cases of overfitting. Although not an ideal metric, accuracy is recorded to compare results with papers that report it exclusively. Mean per-class accuracy is also monitored as it is more robust to class imbalances, however, unlike the macro-F1, it does take into consideration false positives.

## 5.2 Visualizations

### 5.2.1 Model Graph

The model graph provides an intuitive way of visualizing the architecture and data flow of a model. Nodes are used to represent single operations or groups thereof, and connections represent the tensor data that flows from one operation to the next.[1]

[1] https://www.tensorflow.org/guide/graph_viz

### 5.2.2 Histograms

Histograms illustrate the evolution of tensor distributions with training. Internally, the framework attempts to produce histograms for all tensors that are trained.[2]

### 5.2.3 Embedding Projections

When preparing feature data for an experiment, the framework also produces a tab-separated-value (tsv) file listing all the tokens, including any OOV buckets, in the order they appear in the embedding. This file can be used in conjunction with a Tensorboard projector tool to attach labels to points of the embedding matrix projected in $\mathbb{R}^3$ space using principal component analysis (PCA). This may provide additional insights into how the representation of OOV buckets evolves with respect to other tokens by visualizing neighboring points.

### 5.2.4 Confusion Matrix

Matplotlib[3] was used to render most of the visualizations offered by the framework for its extensive tool-set. The confusion matrix is generated using a custom-built hook and provides a high-level overview of the model performance. It is rendered as a heat-map spread across all samples of the evaluation dataset. Although this information may be condensed into informative scalar metrics, such as the macro-F1, the confusion matrix may convey more insight into the per-class transient performance of the model. Ideally, with training, the heat-map should consolidate across the diagonal of the matrix.

### 5.2.5 Dataset Distribution Figure

Two pie-charts are generated for the training and test datasets which depict the class distribution of the each dataset. Moreover, when merged datasets are used,

[2]https://www.tensorflow.org/guide/tensorboard_histograms
[3]https://matplotlib.org/

the contribution of each constituent dataset and their class distributions are also included in this figure. This type of visualization intuitively highlights class imbalances as well as their sources while also efficiently conveying the contribution of each dataset relative to the others in a merged dataset.

### 5.2.6 Attention Heat-maps

The attention heat-maps use a color-map to contrast the different weights being placed on each token. Over time, as the model learns to distinguish which tokens maximally contribute to the sentiment of a sentence, they are expected to be highlighted in contrast to their surroundings which signifies a higher attention weight. Two attention heat-maps are generated for each sample, the first illustrating the color-coded attention weight distribution corresponding to each string token and a second, supplementing the first, with the precise real-valued weights.

Attention weight vectors are obtained using custom implementations of attention units which expose these data and their corresponding string tokens. These are used by the corresponding add-on to generate attention heat-maps for a subset of samples from each batch during evaluation. The size of this subset can be adjusted using the auxiliary-configuration parameters to avoid excessive heat-maps from being generated for increasingly large datasets.

For models that employ multiple "hops", such as memory networks, the process is repeated for each hop. These models are expected to concentrate their attention on the most salient tokens with each hop, as well as improving the process on the whole with each evaluation checkpoint.

## 5.3 Comet-ML Integration

The primary motivating factor behind integrating the comet-ml service into the framework is to facilitate collaborative work by making experiment details and results remotely accessible while speeding up development by automatically setting

up the requisite code infrastructure for this service.

The top-level scope, encompassing all experiment data, organized by the name of the models used, is the comet-ml workspace. At a lower level, a dashboard for each model lists all the respective experiments and allows the user to define custom visualizations comparing metrics across multiple experiments. Finally, each experiment may be individually accessed to view all of the data that has been, or is being, uploaded.

The default behavior of the framework is to upload all images and scalar metrics, all CLI parameters used to start the experiment and, their values as they are resolved during runtime so as to ensure the experiment was run with the desired configuration. These parameters also serve as a useful point of reference when referring back to the experiment at a later date.

In addition to these, the framework also uploads the following resources for each experiment:

- Dataset distribution figure
- Model graph definition file
- A log of the `stdout` stream during execution
- The model source code
- All parameters used in the experiment
- Any embedding filter details and corresponding report
- Description of the host environment used to run the experiment
- A list of installed Python packages used for the experiment

### 5.3.1 Embedding Filter Report

The embedding filter report tabulates embedding tokens filtered as a result of a function-filter. The first three columns describe the function-filter responsible,

while the latter two contain the token and its corresponding NLP properties.

The insights provided by the embedding filter report may serve to inform further refinement of a filtering scheme, for example, if a less stringent filtering approach might be beneficial, or if a stricter rule-set is feasible without sacrificing performance.

## 5.4 Summary

We have explored the details of each stage of the framework, from data preparation to model implementation to the output data and tools used to evaluate them. At each of these stages the core principles of the framework have been to enable rapid experimentation through an intuitive interface and extending this ease-of-use into an API which allows users to expand the functionality of the framework at various points of the process such as with novel datasets, embedding filtering techniques and, custom visualizations.

In the next chapter we present the results obtained from two separate use-cases of the framework, a reproducibility study for a number of models, implemented using the framework API and, an investigation into a number of OOV approaches using the same models.

# 6. Results and Observations

In this chapter, we present the details of two separate studies carried out using the framework. First, we detail our attempts at reproducing five recent approaches in the field of TSA, and any challenges faced therein. Following this, we consider the top-performing model from the reproduction experiments and use the framework to investigate the potential benefits of different strategies for handling OOV tokens..

## 6.1 Reproduction Studies

We carry out all experiments using GloVe word embeddings [53] since all of the studies reproduced use some version of these and because these embeddings performed optimally in both [3] and [9] when dealing with NN based approaches. Since these word embeddings are case insensitive, we lowercase all tokens and carry out tokenization using the Spacy library with the default token filter (§4.3) to remove all emails and URLs from the text.

When determining parameters for each model, the following order is adopted; first, parameters from the original paper are set, while also referring to any accompanying source code which could include parameters missing from the papers. Where unavailable, we use configurations adopted by third-party implementations based on their reported results. Finally, we set any other parameters intuitively, based on the respective setting from models with similar architectures.

As we shall outline, a non-trivial portion of the studies covered omitted essential parameter settings, such as the learning-rate of a model or the number of hidden units used, which drastically hinder their reproducibility. These omissions would be understandable provided the work is supplemented with source-code, however, this was also unobtainable or excluded in most cases.

Parameters such as the batch-size and duration of training are assumed based on online implementations, which report results similar to the original work. To account for this,following [3], we adopt early-stopping with a patience of 10 epochs, a maximum allowance of 300 epochs and a minimum of 30 epochs, monitoring the loss metric of each model at every epoch.

We first provide a brief overview of each approach, which is followed by a description of the challenges faced in reproducing each approach in particular, and the counter-measures taken. Finally, we contrast the results obtained from our experiments with those reported in the original work. Since he effect of the random seed on parameter initialization is statistically significant [50] as cited in [3], we report mean values from 3 separate runs using different random seed values, for each experiment.

### 6.1.1 TD-LSTM

The work carried out by Tang et al 2016 was one of the first to employ LSTM based methods for the task of TSA. Three variants were developed for their experiments, first, a baseline LSTM model which did not take any target information into account. A Target Dependent(TD) LSTM integrated target information by concatenating the target vector to both left and right contexts, and using two separate LSTMs for each. Finally, a Target-Connected(TC) LSTM also adopted two LSTMs, however the target vector information was concatenated to each word in the context, rather than the context as a whole as in the TD-LSTM. This modification incurs a cost on computation time, as it effectively doubles the embedding dimension of each word. Each variant is followed by a softmax layer which is used

to obtain the final classification of the sentence.

The authors use the Dong et al. 2014 dataset, and experiment with both 100-dimension and 200-dimension twitter based GloVe embeddings. Furthermore, LSTM weights are initialized using U(-0.003,0.003) and training is carried out using stochastic gradient descent with a lerning rate of 0.01.

A notable omission in Tang et al. 2016 as well as the reproduction carried out by Moore et al. 2018 is the hidden unit count for each LSTM cell.

To choose a value for this parameter, experiments were carried out for each LSTM variant with 200 and 300 hidden units and both were found to converge on similar results. Since the latter requires more computational resources in training we assume a value of 200 for the results presented herein.

The insignificant downstream differences between these two variants notwithstanding, the omission of this parameter makes reproduction attemps more challenging, and leaves room for uncertainty when comparing results to the original work.

Finally, similar to Moore et al 2018, we ignore the reported 'softmax clipping threshold' of 200 as the implications of this remained misunderstood, and no further elaboration on the parameter is provided by Tang et al. 2016.

As there was no mention of the batch-size or the amount of hidden units used in the kernel of the LSTMs these were set to be 200 and 64 respectively. These values were taken from online public implementations which quoted results similar to those reported in [Tang2016a]

Following on the approach by Moore, we vary the embeddings being used in termed of dimensionality so as to test the observed improvement in results with its increase.

We also increase the upper bound to 300d by including the two variants of the glove embeddings which are sourced from a wider vocabulary, not exclusive to twitter, which could make them better suited for the restaurants and laptops datasets.

Finally, the larger of these two variants also takes into account another aspect by differentiating between uppercase and lowercase letters in tokens.

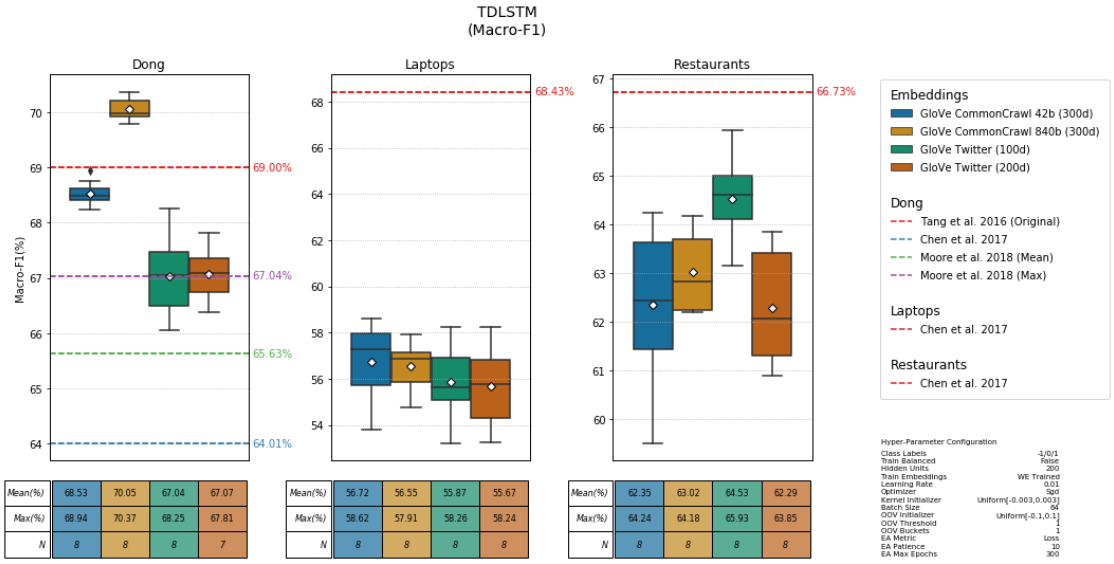


Figure 6.1: TD-LSTM Macro-F1 Results, $N$ refers to the number of runs carried out.

We consider the performance of the TD-LSTM as it is the most commonly quoted variant compared in the literature.

Comparing macro-f1 scores to account for the class imbalances in each dataset we note that the 300d GloVe variants outperform the 100d and 200d embeddings, even though the latter are not sourced from twitter as is the dataset. The 840b variant obtains the best score, suggesting that there is valuable information contained in the case-sensitivity of tokens.

When considering the laptops and restaurants dataset, which differ to dong in terms of class balance and sample length [REF Chapter on datasets], performance drops significantly.

Although the non-twitter embedding variants slightly outperform their counterparts in the laptop dataset, the best mean result is 56.55% indicating the model's poor capacity for learing from the dataset.

The same can be said for the restaurants dataset, considering the wide variance in the results observed using any embedding with the best results being counter-

intuitively obtained by the 100d twitter-sourced glove variant.

When considering the results obtained for the Dong dataset, we note that both 100d and 200d glove embeddings perform similarly, with a slight edge in the mean of the 200d variant. This notwithstanding, both variants' performance is markedly less than that reported in the original paper [Tang], which as stated in [Moore], can be attributed reporting only results from singular runs, thus not taking into account the downstream effects of differences in random initializations of weights.

Finally, the performance of the 300d embedding variants in comparison, is in accordance with the upward trend with increased dimensionality noted in [Moore]

When comparing the results obtained in the laptops and restaurants dataset to those reported in [Chen2017] there is a significant difference irrespective of the embedding used. Although this can also be caused by randomness in the wegiht initializations (no mention of multiple runs is made), it is interesting to note that the opposite phenomena is observed when considering the dong dataset. In this instance the result reported by [Chen2017] is substantially lower than the rest.

This discrepency may be the result of the different approaches that were adopted to account for unknown parameters in the original work [Tang2016a] which were similarly left unreported in [Chen2017].

### 6.1.2 IAN

The Interactive Attention Network (IAN) model [Dehong Ma et. al 2017] separates the input sentence into target and context and feeds the vector representation of these elements to two separate LSTM units. The authors subsequently use max-pooling on the hidden states of these LSTM units and employ an attention mechanism to attend on the output of each LSTM unit with respect to the other's max-pooled hidden state vector. This produces a vector representation of the context weighted by the attention scores with respect to the target and vice-versa. These representations are then concatenated and fed to a softmax classifier which produces a probability distribution over the class labels, the highest of which is

taken to be the final predicted class.

A defining characteristic of this approach is the fact that it is not solely the context's represenation which is dependent on the target, but the target representation itself is affected by its context.

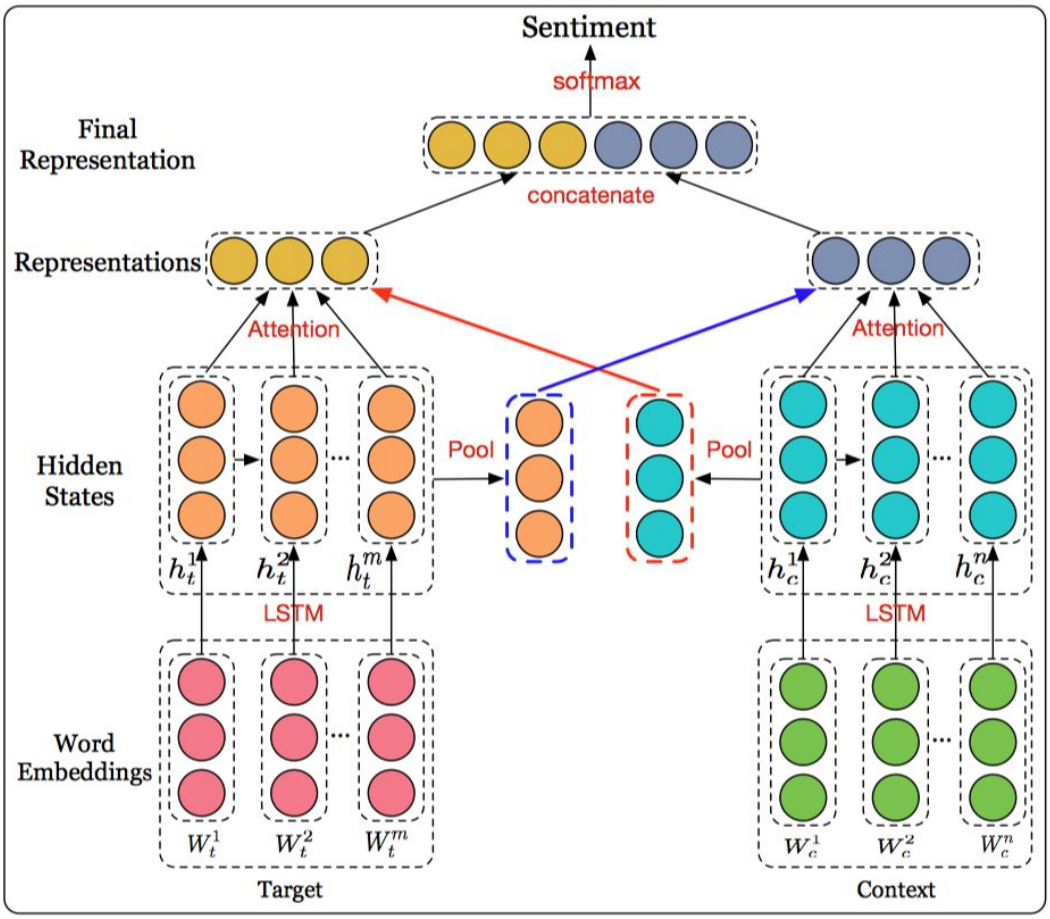


Figure 6.2: Architecture of the IAN Model [Dehong Ma et al. 2017]

The authors make use of a momentum optimizer [Qian et al. 1999] when training their model, however they do not mention the specific momentum parameter that is used when obtaining the reported results. The authors also omit the learning rate hyper-parameter and the batch-size used in their training, both of which are correlated to the momentum hyper-parameter previously mentioned which further exacerbates the problem.

Finally, the authors remark that GloVe [Pennington et al. 2014] embeddings are used with a dimentionality matching that of the hidden unit count in each of the LSTM units, namely 300. It must be noted however that there are two variants of these embeddings with that dimentionality, with substantially different source-vocabulary sizes and only one of which is case-sensitive.

We use values reported in a more recent paper [Zhang et al. 2018] for the momentum, learning-rate and batch size parameters for the default configuration of this model since these parameters are intertwined and in their work [Zhang et al. 2018] report promising results.

We further consider this 'default configuration' as a baseline for the model and evaluate its performance on the datasets reported in the original work as well as the Dong twitter dataset. Furhtermore, we also include four different variants of the GloVe embeddings, 100 and 200 dimentionality twitter variants and both of the aforementioend 300 dimentionality variants so as to observe the difference in downstream performance these make with respect to the reported results.

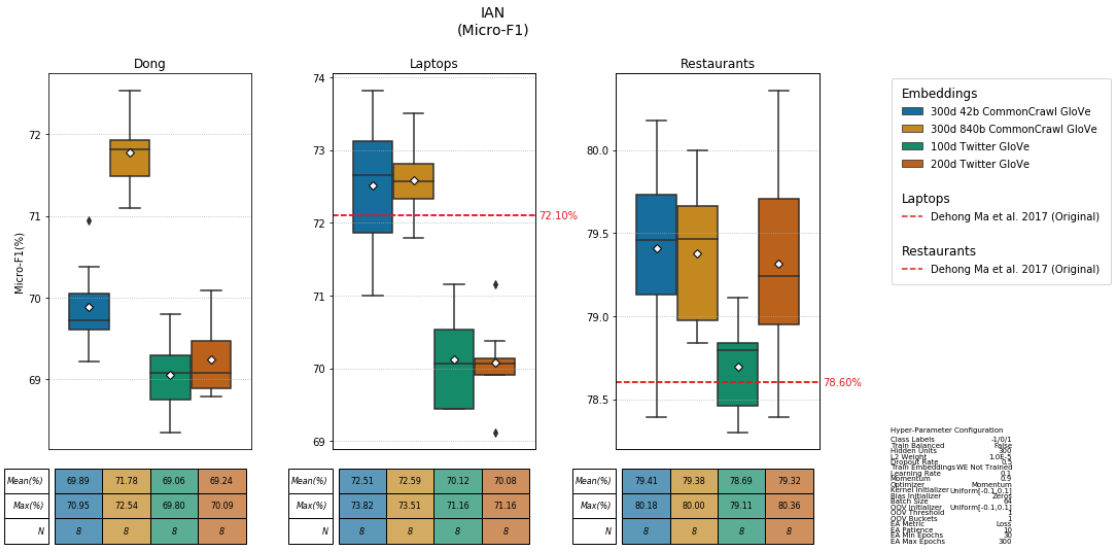


Figure 6.3: IAN Micro-F1 results (compared with originally reported results)

We first compare the micro-F1 (accuracy) results obtained in the our experiments with those reported in the original work [Ma et al. 2017] for the laptops and

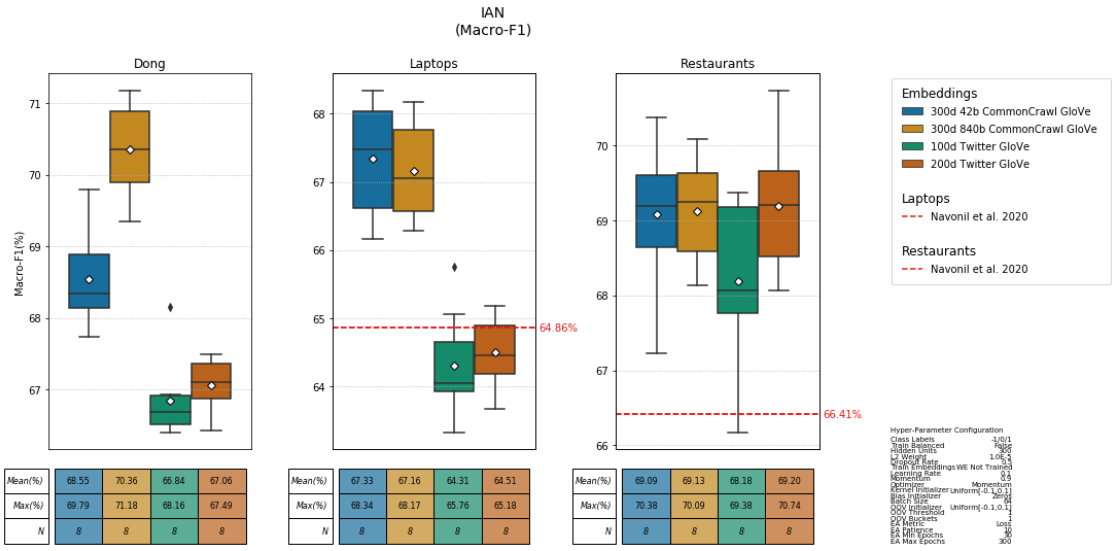


Figure 6.4: IAN Macro-F1 results (compared with results reported in [Navonil et al. 2020])

the restaurants dataset as these are the only metrics which were cited. It is worth pointing out once more that both of these datasets are heavily skewed in their class distribution [REF: datasets chapter for more info] which confines the validity of this metric.

This notwithstanding, in both datasets when employing both variants of the 300d GloVe embeddings we find that the reported micro-f1 results are considered conservative when compared to the mean value of the same metric in our experiments.

The authors do not mention the number of runs that were performed or whether multiple runs were performed, which may explain this discrepancy. We can observe that both reported metrics fall within the range of results obtained with the 42b 300d GloVe embedding, which may suggest that this was the variant the authors were referreing to in their work.

Recently, a novel approach was presented in [Novanil et al. 2020] which included the IAN architecture as a baseline and report macro-f1 scores for the skewed datasets. In their work, the authors report a learning rate of 0.002 as opposed to 0.1 which was employed in our work.

We observe similar, if not more pronounced, differences when considering the 300d GloVe variants in both laptop and restaurants datasets, with the reported metrics in both cases falling squarely below the range of values we obtain in our experiments.

Although this discrepenacy in both cases can be considered 'favorable', it nonetheless continues to underscore the importance of providing all parameter settings for these models to facilitate reproduction by mitigating the overhead involved in attempting to recreate the original results.

Moreover, the importance of reporting metrics which take into account dataset characteristics as outlined in [REF: chapter on metrics] and which take into account the downstream effects of randomness in the initialization process [Gurevych, Moore] continues to emerge from these results.

### 6.1.3 LCR-ROT

From the approaches described thus far, we observe that utilizing attention mechanisms to affect the representation of a context concerning its target is effective when tackling TSA tasks. This may be considered fairly intuitive, however, as we have seen [18] present a compelling argument with their IAN model that the inverse, in favor of using attention mechanisms to affect the target representation concerning its context as well. [79] expand upon this idea in their work presenting a Left-Center-Right Rotary Attention network (LCR-ROT) in several ways, albeit at the cost of a heavier computational load.

First, similar to other works such as [64] and [11], the authors consider the left and right contexts of a target separately as opposed to a single context vector which makes the model more robust when considering the same sentence for different targets.

Secondly, the authors employ Bi-directional LSTMs, which enable the model to extract sequential feature data in both directions of a sequence. This comes at the cost of double the parameters that the model must learn since it is achieved

by stacking the hidden states of two separate LSTMs processing the sequence in opposite directions. Three BiLSTMs are used for the left and right contexts and the target respectively.

An initial, *Target2Context*, attention model is used to weight both left and right context representations against a target representation obtained through average-pooling, producing target-aware context representations. The rotary attention mechanism characterizes the inverse of this process, using both target-aware context representations in combination with a second *Context2Target* attention model to construct two separate left-aware and right-aware target representations.

Finally, all four components are used in conjunction as the sentence feature vector which is fed to a softmax classifier to obtain the sentiment classification.

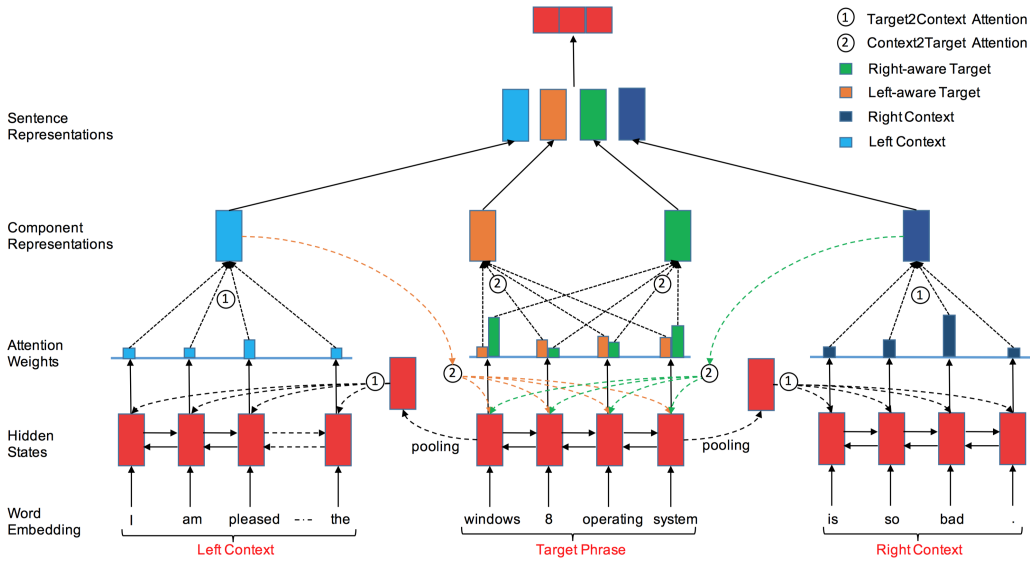


Figure 6.5: Left-Center-Right Separated Neural Network Architecture [79] illustrating the construction of each constituent component of the final sentence representation. Bi-LSTMs are employed for the initial feature extraction followed by two attention models, *Target2Context* and *Context2Target*, which produce target-aware context representations as well as left and right context-aware target representations.

When attempting to reproduce the approach as described in [79], we note that this work provides the best coverage of important parameters thus far. The authors

further indicate their intention to make the source code of their model publically accessible. Although the paper provides no direct link, the repository was sufficiently easy to locate manually. The repository contains an example, which specifies values for the remaining, otherwise unspecified parameters. In this case, these are the batch-size and training duration. Although unclear whether meant merely as a suggestion, we use this configuration for our reproduction experiments on the assumption that this configuration would be the most well-educated even in this eventuality.

However, a more pressing concern is the lack of specificity when describing the word embeddings used for their experiments. We have previously outlined the differences in the two variants of 300-dimension GloVe embeddings, and have repeatedly produced varying results in each of our tests. [79] cite [72] and [64] as motivation for their choice of embeddings; however, these approaches report using different variants to one another.

Furthermore, the authors do not specify whether embeddings were trained further in their experiments. When referring to the codebase, we note both approaches, with the code for further embedding training being disabled. We assume this to imply that excluding embeddings from the training process provided superior results and adopt the same approach in our experiments.

Finally, we draw attention to the use of accuracy (macro-f1) as the definitive metric for performance, particularly when considering the laptops and restaurants datasets. As previously mentioned, the composition of these datasets' constituent classes is heavily skewed. We have described how easily accuracy falls victim to overly-optimistic and biased results by not accounting for this imbalance.

Similar to the previously mentioned studies, we assume the reported metrics are optimal, as the authors did not mention multiple runs. When considering the Dong dataset, our results suggest that the authors employed the larger 840b variant of the GloVe embeddings. The reported micro-f1, in this case, falls within the mean (-0.06), as opposed to the 42b variant, which performs markedly worse.

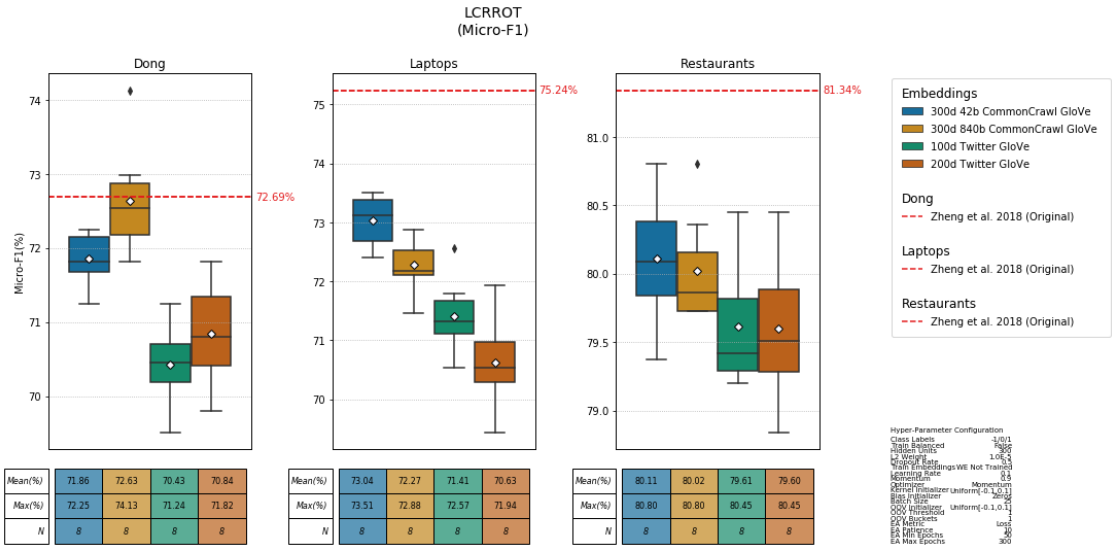


Figure 6.6: LCR-ROT Micro-F1 results (compared with originally reported results)

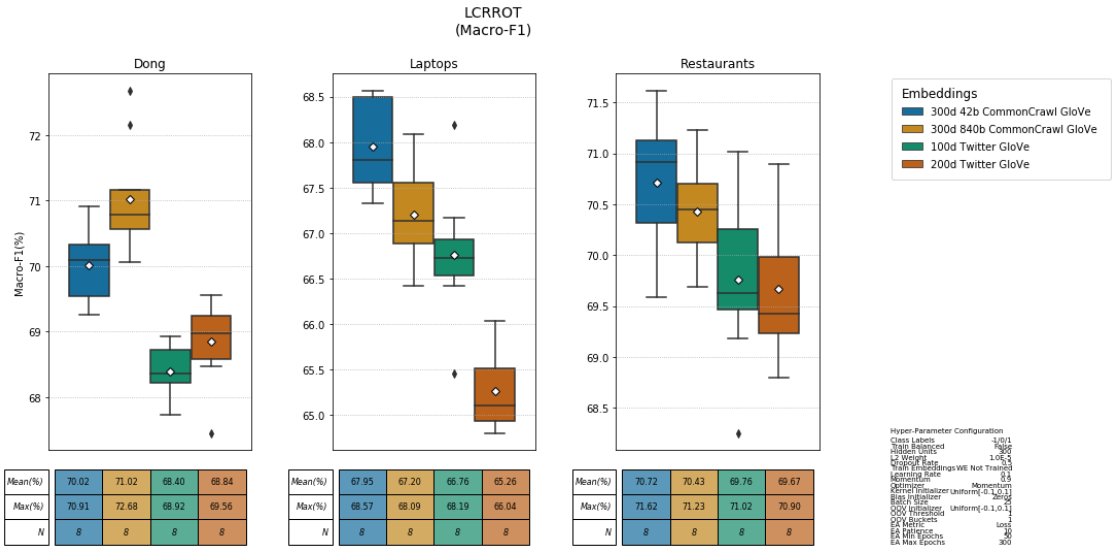


Figure 6.7: LCR-ROT Macro-F1 results

We note that the principles adopted in the LCR-ROT approach's more sophisticated modeling of context and target relations are substantiated. The model outperforms the previous works across all experiment configurations we investigate. Despite this, we were unable to reproduce the reported micro-f1 metrics for the Laptops and Restaurants datasets. The more substantial discrepancy between results for these two datasets, when compared to the Dong dataset, is likely

a consequence of the latter's less pronounced class imbalance. [REF to datasets chapter]

We observe another evident by-product of this distinction in the magnitude of deviation in macro-f1 scores and micro-f1 scores. Considering the best-performing mean scores for each dataset, the macro-f1 scores for the Dong dataset deviates from its respective micro-f1 counterpart by 1.61 points. Comparatively, this figure increases to 5.09 and 9.38 for the Laptops and Restaurants datasets' best-performing configurations. This difference is replicated across all embedding configurations and is an indication of the degree to which the micro-f1 metric falls short in these circumstances.

Finally, we note that in all previously presented experiments, the 840b GloVe embedding consistently outperforms its 42b counterpart in the Dong dataset. The nature of the Dong dataset, being sourced from a social media platform that encourages a relaxed demeanor and enforces brevity, leading to samples that typically ignore correct grammar and punctuation. Initially, it may seem counter-intuitive that the case-sensitive embeddings would perform better in this domain. However, we surmise that by often ignoring the case, in the rare instances which adhere to it, such as proper nouns, provide more information than in other, more formal and verbose, datasets.

### 6.1.4 Concluding Remarks

We have documented herein the process of reproducing three critical studies employing increasingly complex structures to further the field of TSA. In doing so, we have elucidated the following challenges and suggestions to mitigate them accordingly:

- We have outlined the detrimental effects that lack of specificity has on this process, consuming more time and resources, thereby setting back further development. At a minimum, the accompanying literature must endeavor to

include all pertinent parameters. Moreover, making the code of an approach publically available and appropriately referenced in the literature upon publication is optimal. Our framework facilitates this process by minimizing the work required to produce a working implementation with access to datasets and robust performance metrics.

- We have illustrated in each case how reporting the single optimal score for an approach is insufficient. The authors must consider the variance observed in each of our experiments when carrying out multiple runs with different random seeds. Furthermore, we maintain that this approach carries increasing insight into a model with an even higher number of runs.

- Finally, we have shown that careful attention is imperative when considering the performance metric to report, for the datasets used. Notably, it is evident that the macro-f1 scores are more robust in this field and give a more precise, albeit less optimistic, measure of a model's performance compared to the micro-f1 (accuracy).

## 6.2 OOV Studies

Consider a sub-space composed of all Out-Of-Vocabulary(OOV) tokens in the training and testing datasets. We assume the existence of an arbitrary amount of knowledge distributed across this sub-space which may or may not have a direct influence on the sentiment of the context of an OOV token where it appears. Ideally, a model learns an accurate representation of the weight of this influence during training, with each time it encounters the token. OOV tokens hinder this process in two ways; first, OOV words are, by definition, scarce within a given vocabulary and therefore provide limited opportunities to construct and optimize for an accurate representation. Second, when these tokens are encountered, the effect on the sentiment of their context is attenuated when compared to other, more frequent, tokens

for which the model has numerous opportunities to fine-tune its understanding of token's consequential effect on the task at hand. In our OOV sub-space, this is analogous to the information contained therein being spread too thin across all the tokens comprising it.

The motivation underlying the following work is to investigate whether this OOV sub-space can be effectively encoded into a smaller number of clusters, each composed of tokens with shared properties, ideally relating to the sentimental information each carries. Since the information contained within the OOV sub-space is finite, the degree to which such an approach would be effective is modulated by the cluster-cardinality, which determines the range of information we intend to encapsulate in each cluster.

We use our framework to construct a series of experiments consisting of a baseline, followed by variations on the aforementioned range of information. This is achieved through the OOV policy parameter, specifically the OOV-train-threshold and OOV-bucket parameters. Combined, these parameters provide a mechanism by which to cluster single OOV tokens, encountered both at the training and testing stage, into unique vector representations which are subsequently updated in the model's training process.

This section provides a detailed description of the experiment setup and motivations behind the specific choices made, followed by an analysis of the corresponding results.

### 6.2.1 Experiment Setup

When preparing these experiments, our principal consideration is the resource overhead involved in each configuration as a function of both time and cost. Addressing the latter, we select a model from our previous reproduction studies with the lease complexity, namely [Tang et al.] TD-LSTM approach. This model consumes the least amount of resources in training while maintaining a sufficient degree of aptitude for TSA, particularly in the social-media network domain in conjunction with

the common-crawl 840b GloVe embeddings.

Additionally, to obtain more in-depth insight into the effects of these OOV strategies, we seek to maximize the vocabulary size of a dataset, thereby increasing the absolute number of OOV tokens present. Towards this end, we use the [Nakov] dataset, which has a considerably higher vocabulary compared to the [Dong] dataset while retaining the same social-media domain.

The above limitation of time is the tokenization and mapping processes being contingent on the OOV parameters we are investigating. As described in section 3.1, these tasks fall under the responsibility of the data module. They must be carried out locally before being delegated to the cloud. Furthermore, the time required for these tasks, in particular, mapping each token to its respective index in the embedding matrix, increases drastically with the size of both embeddings and the vocabulary of the dataset.

Where we to tackle this problem using a different embedding, we expect a notable decrease in the model's performance as observed in the reproduction experiment. The fact that this embedding variant is the only one that is case-sensitive is also intentional, as this adds a degree of precision when matching tokens, which increases the probability of a token being OOV.

Therefore, we must address the alternative bottleneck, the dataset, by limiting its vocabulary size. This is in direct opposition to our motivation in choosing this dataset, however, and an overly-aggressive approach here negates any insights we may have otherwise obtained by using this dataset as opposed to Dong.

To account for these circumstances, we employ the framework's re-sampling function to downsample the original training dataset into a smaller, evenly distributed version. Notably, we exclude the testing dataset from this process, reducing the amount of OOV tokens that the model may have otherwise encountered in training while maintaining the same amount of OOV tokens in testing. Moreover, this approach ensures that the model is exposed to an equal number of positive, neutral, and negative samples in training. The resultant dataset reduces the time

required by an average of 75% (2hrs vs. 30mins)

We identified several irregularities in the Nakov dataset upon further inspection. Notably, this dataset lacks any specific offset value, which precisely indicates the location of a target in a sentence. We design the framework to address this issue by automatically determining the offset using a standard string search when these are not provided. Despite this, multiple occurrences of the target string in a sentence obscure this process, since there is no means of discerning the intended target.

One approach to this problem is to include the sample multiple times for each target occurrence. However, this imposes the need for a memory element when parsing data to recall if any previous target mentions in a sentence have been accounted for. We determine the added complexity in this approach to be excessive for the portion of the samples affected.

We instead implement a stricter matching function using regular expressions at a minimal additional overhead cost to the parsing. This prioritizes instances of the target surrounded in word boundaries and falls back to the more straightforward string matching approach when this returns no results. We discard the samples for which this approach is still unable to determine the target location in the sentence with certainty.

A minority of multi-word targets in the dataset are separated by characters in the sentence inconsistent with the target entity. The confounding variables surrounding the small subset of samples exhibiting this issue did not merit a generalized approach. Therefore these samples were also discarded from the dataset. Finally, we map the original 5-degree sentiment classes of the dataset to 3-degrees as outlined in [SECTION 4.4]

### 6.2.2 Model Parameters

For each OOV policy, we assess the performance of the model with slight variations to the default parameter set used in the reproduction studies, since this is a novel dataset, to identify the ideal parameter set for it. These variations include

a learning rate of 0.1 as opposed to 0.01 and increasing the hidden unit count of the LSTM to 300 as opposed to 200, as well as a third configuration comprising both. We report results on the best performing parameter set across all OOV policy configurations.

### OOV Train Threshold

The first parameter is the OOV training threshold level, which stipulates a minimum number of times an OOV token must be encountered during training to be assigned its vector in the embedding matrix. We refer to the number of tokens that meet this threshold as a fraction of the total number of OOV token in the training dataset as the train-time OOV coverage. Consequently, a threshold value of 1 results in 100% train-time OOV coverage.

Any value greater than 1 drastically reduces this coverage as in most datasets, the vast majority of OOV tokens appear only once. The framework reserves the 0 value to denote the condition where all OOV tokens in training are ignored and subsequently bucketed, effectively setting train-time OOV coverage to 0%.

With every increment beyond a threshold of 1, however, we observe a rapid decrease in the train-time OOV coverage. For these reasons, we limit our experiments to a threshold value of 2 (6.12% cov.) and 3(2.28% cov.). Values beyond this are nearly indistinguishable from the 0-valued case (0% cov.).

### OOV Test Bucket Number

The second OOV policy parameter determines the number of buckets in which test-time OOV tokens are distributed. These buckets are each assigned a vector in the embedding space. They are analogous to the clusters, as mentioned earlier in our OOV sub-space. Consequentially, this parameter is inversely proportional to the amount of information each bucket is expected to encapsulate on average. Therefore, a common approach of assigning all OOV words to a unique randomly initialized vector is represented by a value of 1, which is also the default value used

in our reproduction studies.

When assigning each token to a bucket, we employ the native hashing function, which is included in the Tensorflow library. We adopt a principled approach to this idea, investigating a broader range of values spanning 3 different orders of magnitude. Doing so identifies the potential degree to which this strategy can manifest positive downstream effects on performance.

We leave further research into more granular fine-tuning of this value as well as more sophisticated hashing algorithms that assign tokens to buckets for future work.

### 6.2.3 Results and Observations

We adopt a similar approach to our reproduction studies, to a lesser extent to account for the increase in the number of experiments. Each experiment configuration is run 4 times and we gauge the performance of each using the macro-F1 score to account for the class imbalance in the testing dataset. We find the best performing parameter set for this modified Nakov dataset is the variant with 300 hidden units and a learning rate of 0.1. We speculate that with the increased vocabulary in the Nakov dataset, the model can effectively leverage the additional parameters to produce a more accurate representation of the dataset. Moreover, the faster learning rate may be a means by which the model is able to extract more information from the reduced sample size of the training dataset by amplifying the effect of each.

We illustrate the results of different threshold level, while mainining the bucket parameter at 1, in 6.9. We observe the most pronounced improvement when $T = 0$, which disregards all OOV tokens in training. We believe the larger deviation of this configuration results from the miniscule effect on train-time OOV coverage that the other values have compared to $T = 1$. More surprising however is the polarity of this difference, which suggests that it is more beneficial to disregard OOV tokens at training altogether.

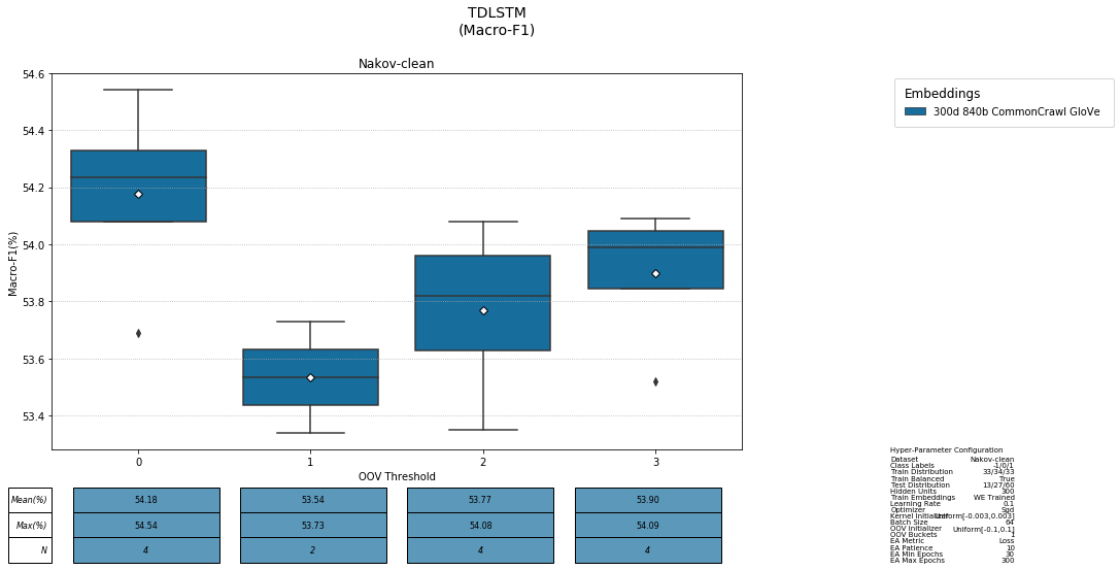


Figure 6.8: Results obtained with varying OOV training threshold parameters. (OOV Buckets = 1)

This may indicate that due to the low occurrence rate of OOV tokens, and the natural distribution of words in languages, there lacks sufficient contextual information to embed the significant of the OOV token in isolation. This effectively reduces the added vectors to noise, which would be reduced by increasing threshold values above 1, which can be observed in the slight gradual improvement of macro-f1 scores in 6.9 Recall that in this configuration, we set the OOV buckets to 1, therefore when $T = 0$ all training OOV tokens are subsequently grouped under a single OOV bucket. Increasing the granularity of this strategy with additional buckets may further improve the performance of this approach.

We carried out our initial experiments investigating increasing OOV buckets compared to the default reference case of $T = 1$ and $B = 1$. Preliminary results from these experiments are illustrated in 6.9 and hint at slight improvements over the mean macro-f1 scores as we increase the number of buckets up to $B = 100$. Similar to the train threshold scenario, given that the nakov dataset has over 2000 OOV tokens, at $B = 1000$ each bucket on average shall group 2 OOV tokens. As we have previously discussed, since the majoriti of OOV tokens only occur once, this

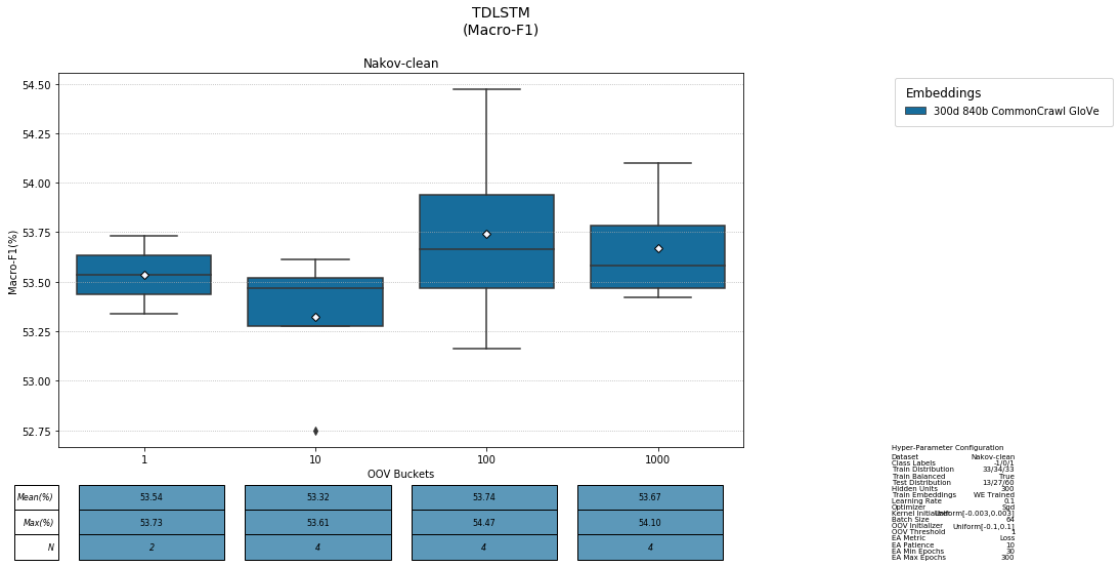


Figure 6.9: Results obtained with different numbers of test time OOV buckets. (OOV Train Threshold = 1)

approach results in buckets bereft of enough contextual information for accurate representation.

We believe that these results merit further investigation into the possibility of accumulating the benefits of each optimal parameter value. Future avenues of research could involve further experiments covering combinations of these parameters, and larger vocabularies obtained by merging individual datasets using our framework. Finally, we present our results as evidence that these parameters, particularly the bucketing strategy, do indeed manifest in downstream performance differences. We believe this merits further investigation into the significance of these parameter configurations and to what degree their influence can be fine-tuned and optimized both in the generalized setting as well as for specific domains and their corresponsing vocabularies.

# 7. Conclusion

We draw a number of conclusions from the development of this framework and its evolution, however, one over-arching observation stands out, and affects all others. The degree of difficulty and complexity that arises from decomposing a machine learning pipeline, even when limited to a singular task, cannot be understated. Although a subset of these challenges stem from having to deal with particular irregularities or inconsistencies within datasets or embeddings, this is not the source of these issues. The primary source of these obstacles lies in the overly-ambitious approach we adopted in development by attempting to account for all edge-cases, focus was placed on the having the widest of the coverage of these features at the cost of depth in other areas which required it.

As a result, our framework succesfully meets the objectives we set out to accomplish *functionally*, in that it provides a simple and accessible means of rapidly evaluating TSA models, with access to a wealth of different datasets and embeddings. We also provide an expansive means of interacting with the framework on a deeper level, through specific entry points which allow users to build on top of the framework and extend its functionality with minimal effort. The majority of the time was spent in developing these features, some of which would not be used in the experiments that followed, others still which we were unable to complete, such as GPU integration and the ability to export a trained model for predictions.

This focus placed time-constraints on our experimentation objectives, a problem

which, as we documented, was further intensified due to missing parameters and implementation details. The extent to which this is a factor, both in time and cost, cannot be understated. From our work in reproducibility we identify this as one of three issues poised to hold back further development of the field. We conclude that more emphasis must be made on the importance of making all code publically accessible when developing new techniques and approaches. The second is the importance of reporting performance in terms of multiple runs when using random initialization, and finally the significance of appropriate performance metric when dealing with unbalanced datasets.

When considering the OOV clustering technique adopted in our latter experiments, we conclude that this approach requires much deeper investigation for it to be substantiated. We show from our experiments that the configuration alterations we propose do have noticable downstream effects, however we note that this approach itself consists of a number of tunable parameters, and the extent to which the effectiveness of the approach is contingent on these parameters is still unknown.

The approach may show more pronounced effects when dealing with larger amounts of OOV tokens. We could not carry out these experiments due to time and moneraty cost, however the framework provides a means of exploring this issue further through it's dataset merging and re-distributing features. Moreover, we also identify the clustering function itself as another point of future development in this approach.

The shortcomings notwithstanding, the principles adopted in the design and implementation of the framework, specifically the modular architecture were invaluable at later stages of the work when we were required to run large amounts of experiments with, at times miniscule, alterations. The framework greatly simplified the process, requiring only command line parameter changes for most, abstracting the more complex and error-prone tasks from the user. While achieving this level of abstraction and ease of use was the most arduous component of the work as a whole, it demanded a deep understanding of all the moving parts within a

TSA-oriented machine learning pipeline, which also made it the most rewarding.

# References


[1] Understanding lstm networks – colah's blog, 12/12/2018.

[2] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[3] Andrew Moore and Paul Rayson. Bringing replication and reproduction together with generalisability in nlp: Three reproduction studies for target dependent sentiment analysis. *CoRR*, abs/1806.05219, 2018.

[4] O. Appel, F. Chiclana, J. Carter, and H. Fujita. A hybrid approach to the sentiment analysis problem at the sentence level. *Knowledge-Based Systems*, 108:110–124, 2016.

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

[6] C. Baziotis, N. Pelekis, and C. Doulkeridis. Datastories at semeval-2017 task 4: Deep lstm with attention for message-level and topic-based sentiment analysis. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 747–754, Vancouver, Canada, 2017. Association for Computational Linguistics.

[7] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[8] Y. Bengio, P. Simard, P. Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[9] Bhuwan Dhingra, Hanxiao Liu, Ruslan Salakhutdinov, and William W. Cohen. A comparative study of word embeddings for reading comprehension. *CoRR*, abs/1703.00993, 2017.

[10] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[11] P. Chen, Z. Sun, L. Bing, and W. Yang. Recurrent attention network on memory for aspect sentiment analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 452–461, 2017.

[12] P. Chen, B. Xu, M. Yang, and S. Li. Clause sentiment identification based on convolutional neural network with context embedding. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 1532–1538, 2016.

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

[14] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[15] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[16] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 933–941. JMLR.org, 2017.

[17] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[18] Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Interactive attention networks for aspect-level sentiment classification. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4068–4074, 2017.

[19] L. Dong, F. Wei, C. Tan, D. Tang, M. Zhou, and K. Xu. Adaptive recursive neural network for target-dependent twitter sentiment classification. In K. Toutanova and H. Wu, editors, *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 49–54, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[20] G. Frege. On sense and reference, 1892. *Readings in the Philosophy of Language*, pages 563–583, 1892.

[21] F. Gallwitz, E. Noth, and H. Niemann. A category based approach for recognition of out-of-vocabulary words. In H. T. E. Bunnell and W. Idsardi, editors, *ICSLP 96*, pages 228–231, New York, 1996. Institute of Electrical and Electronics Engineers.

[22] F. A. Gers. Learning to forget: continual prediction with lstm. In *ICANN 99, Ninth International Conference on Artificial Neural Networks*, Conference publication / Institution of Electrical Engineers, pages 850–855, London, 19XX. IEE.

[23] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[24] A. Graves. Supervised sequence labelling. In A. Graves, editor, *Supervised sequence labelling with recurrent neural networks*, volume 385 of *Studies in Computational Intelligence*, pages 5–13. Springer, Heidelberg and London, 2012.

[25] A. Graves, editor. *Supervised sequence labelling with recurrent neural networks*, volume v. 385 of *Studies in Computational Intelligence*. Springer, Heidelberg and London, 2012.

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

[27] M. Jabreel and A. Moreno. Target-dependent sentiment analysis of tweets using a bi-directional gated recurrent unit. In *WEBIST*, pages 80–87, 2017.

[28] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2014.

[29] L. Jiang, M. Yu, M. Zhou, X. Liu, and T. Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 151–160, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[30] Jiwei Li, Dan Jurafsky, and Eduard H. Hovy. When are tree structures necessary for deep learning of representations? *CoRR*, abs/1503.00185, 2015.

[31] R. Johnson and T. Zhang. Semi-supervised convolutional neural networks for text categorization via region embedding. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 919–927. Curran Associates, Inc, 2015.

[32] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.

[33] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015.

[34] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[35] Y. Kim. Convolutional neural networks for sentence classification. In Q. C. R. I. Alessandro Moschitti, G. Bo Pang, and U. o. A. Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[36] S. Kiritchenko, X. Zhu, C. Cherry, and S. Mohammad. Nrc-canada-2014: Detecting aspects and sentiment in customer reviews. In P. Nakov and T. Zesch, editors, *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 437–442, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[37] H. Lakkaraju, R. Socher, and C. Manning. Aspect specific sentiment analysis using hierarchical deep learning. In *NIPS Workshop on deep learning and representation learning*, 2014.

[38] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In G. Montavon, G. Orr, and K.-R. Müller, editors, *Neural networks*, volume 7700 of *LNCS sublibrary. SL 1, Theoretical computer science and general issues*, pages 9–48. Springer, Heidelberg, 2012.

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

[40] F. Liu, T. Cohn, and T. Baldwin. Recurrent entity networks with delayed memory update for targeted aspect-based sentiment analysis. *arXiv preprint arXiv:1804.11019*, 2018.

[41] Y. Ma, H. Peng, and E. Cambria. Targeted aspect-based sentiment analysis via embedding commonsense knowledge into an attentive lstm. In *Proceedings of AAAI*, pages 5876–5883, 2018.

[42] C. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16(1):100–103, 2010.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

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

[45] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[46] P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoyanov. Semeval-2016 task 4: Sentiment analysis in twitter. In *Proceedings of the 10th international workshop on semantic evaluation (semeval-2016)*, pages 1–18, 2016.

[47] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.

[48] W. NAPTALI, M. TSUCHIYA, and S. NAKAGAWA. Class-based n-gram language model for new words using out-of-vocabulary to in-vocabulary similarity. *IEICE Transactions on Information and Systems*, E95.D(9):2308–2317, 2012.

[49] T. H. Nguyen and K. Shirai. Phrasernn: Phrase recursive neural network for aspect-based sentiment analysis. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2509–2514, 2015.

[50] Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. *CoRR*, abs/1707.09861, 2017.

[51] B. Pang, L. Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends\textregistered in Information Retrieval*, 2(1–2):1–135, 2008.

[52] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: Sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 79–86, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[53] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In Q. C. R. I. Alessandro Moschitti, G. Bo Pang, and U. o. A. Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[54] M. Pontiki, D. Galanis, J. Pavlopoulos, H. Papageorgiou, I. Androutsopoulos, and S. Manandhar. Semeval-2014 task 4: Aspect based sentiment analysis. In

P. Nakov and T. Zesch, editors, *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 27–35, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[55] S. Rosenthal, P. Nakov, S. Kiritchenko, S. Mohammad, A. Ritter, and V. Stoyanov. Semeval-2015 task 10: Sentiment analysis in twitter. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, pages 451–463, 2015.

[56] D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[57] M. Saeidi, G. Bouchard, M. Liakata, and S. Riedel. Sentihood: Targeted aspect based sentiment analysis dataset for urban neighbourhoods. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1546–1556. The COLING 2016 Organizing Committee, 2016.

[58] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng, and C. D. Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 151–161, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[59] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics, 2013.

[60] Soujanya Poria, Erik Cambria, Devamanyu Hazarika, and Prateek Vij. A deeper look into sarcastic tweets using deep convolutional neural networks. *CoRR*, abs/1610.08815, 2016.

[61] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.

[62] S. Sukhbaatar, a. szlam, J. Weston, and R. Fergus. End-to-end memory networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2440–2448. Curran Associates, Inc, 2015.

[63] D. Tang, B. Qin, X. Feng, and T. Liu. Effective lstms for target-dependent sentiment classification. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3298–3307. The COLING 2016 Organizing Committee, 2016.

[64] D. Tang, B. Qin, and T. Liu. Aspect level sentiment classification with deep memory network. In J. Su, K. Duh, and X. Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 214–224, Stroudsburg, PA, USA, 2016. Association for Computational Linguistics.

[65] D. Tang, F. Wei, N. Yang, M. Zhou, T. Liu, and B. Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In K. Toutanova and H. Wu, editors, *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565, Stroudsburg, PA, USA, 2014. Association for Computational Linguistics.

[66] Y. Tay, L. A. Tuan, and S. C. Hui. Dyadic memory networks for aspect-based sentiment analysis. In E.-P. Lim, editor, *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 107–116, New York, NY, 2017. ACM.

[67] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709:1–24, 2017.

[68] Vishal. A. Kharde and Sheetal. Sonawane. Sentiment analysis of twitter data : A survey of techniques. *CoRR*, abs/1601.06971, 2016.

[69] D.-T. Vo and Y. Zhang. Target-dependent twitter sentiment classification with rich automatic features. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1347–1353. AAAI Press, 2015.

[70] J. Wagner, P. Arora, S. Cortes, U. Barman, D. Bogdanova, J. Foster, and L. Tounsi. Dcu: Aspect-based polarity classification for semeval task 4. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 223–229, 2014.

[71] B. Wang, M. Liakata, A. Zubiaga, and R. Procter. Tdparse: Multi-target-specific sentiment recognition on twitter. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 483–493. Association for Computational Linguistics, 2017.

[72] S. Wang, S. Mazumder, B. Liu, M. Zhou, and Y. Chang. Target-sensitive memory networks for aspect sentiment classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1:957–967, 2018.

[73] Y. Wang, M. Huang, x. zhu, and L. Zhao. Attention-based lstm for aspect-level sentiment classification. In J. Su, K. Duh, and X. Carreras, editors, *Proceedings*

*of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Stroudsburg, PA, USA, 2016. Association for Computational Linguistics.

[74] W. Xue and T. Li. Aspect based sentiment analysis with gated convolutional networks. *arXiv preprint arXiv:1805.07043*, 2018.

[75] Ye Zhang and Byron C. Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015.

[76] Yoav Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[77] L. Zhang, S. Wang, and B. Liu. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1253, 2018.

[78] M. Zhang, Y. Zhang, and D.-T. Vo. Gated neural networks for targeted sentiment analysis. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 3087–3093. AAAI Press, 2016.

[79] S. Zheng and R. Xia. Left-center-right separated neural network for aspect-based sentiment analysis with rotatory attention. *arXiv preprint arXiv:1802.00892*, 2018.

[80] X. Zhu, P. Sobihani, and H. Guo. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pages 1604–1612, 2015.