DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Aspect-Based Sentiment Analysis Using Deep Neural Networks and Transfer Learning

Sumit Dugar

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Aspect-Based Sentiment Analysis Using Deep Neural Networks and Transfer Learning

Aspektbasierte Sentimentanalyse mittels tiefen neuronalen Netzen und Transferlernen

Author:	Sumit Dugar
Supervisor:	PD Dr. Georg Groh
Advisor:	Gerhard Hagerer
Submission Date:	15.03.2019

I confirm that this master's thesis in informatics is my own work and I have documented all sources and materials used.

München, 15.03.2019 Sumit Dugar

Acknowledgments

This work on aspect-based sentiment analysis (ABSA) was carried out in the year 2018/2019 at Technical University of Munich.

First and foremost I would like to thank PD Dr. Georg Groh and Gerhard Hagerer from the Research Group of Social Computing for giving me this amazing opportunity to work and collaborate with their group. This has been a great learning experience for me and I am very grateful to them for their guidance and support.

I would especially like to thank my advisor Gerhard Hagerer for his supervision and for his interesting ideas which helped in shaping the project. He was very patient and has always encouraged me with full support and enthusiasm to pursue my ideas. His support and guidance has been very crucial in the completion of this thesis and also for my learning during the thesis.

I am also very grateful to each and every member of the SocialROM team for asking the right questions, driving the discussions to a conclusion, and especially for giving me the opportunity to share my ideas and problems during our weekly meetings. These discussions were very fruitful and have definitely helped me remain in check throughout the course of this project. Special thanks to PD Dr. Georg Groh and Gerhard Hagerer for organising these weekly meetings and sharing their knowledge and experience during the discussions. Their remarks gave new insights on various occasions.

Furthermore, I would also like to acknowledge the support and encouragement of my family and family like friends - Neha and Rangoli. I would especially like to thank Neha for sharing her knowledge and remarks patiently during our numerous discussions.

Abstract

In this thesis, we propose a new neural network architecture for aspect-based sentiment analysis (ABSA). Instead of a pipeline approach where an aspect is first detected and then the corresponding sentiment polarities are classified, we use a joint formulation which is similar to the approach used by Schmitt et al. (2018). Furthermore, we also propose an input data transformation technique that provides some interesting advantages to our model. It helps in addressing problems that arise due to the huge number of output classes and it also provides additional capabilities that help our model to apply transfer learning approaches such as Progressive Neural Networks (PNN).

We conducted a thorough study of different ways in which a task such as sentiment analysis or ABSA can be formulated using various sequential modeling techniques such as HMM, CNN, RNN. We performed experiments with different variants of our model on a variety of datasets such as - SemEval-2016 Restaurant, SemEval-2016 Laptops, Organic food, GermEval-2017 Deutsche Bahn. We also experimented with different word embeddings such as GloVe, fastText, ELMo.

We were able to show that transfer learning approaches consistently gives a boost of about 1 or 2 F1 points across all the four datasets. With our experiments especially with GermEval-2017 data, we were able to show that our model performed better than the LSTM variant of the current state of the art approach from Schmitt et al. (2018). However, their CNN variant performed better than our best LSTM based model. To be the best of our knowledge, we are the first ones to present practical results on SemEval-2016 restaurant and laptop dataset for aspect-based sentiment analysis using a joint approach, therefore setting a baseline for future research.

Keywords: Aspect-based sentiment analysis (ABSA), GloVe, fastText, ELMo, Progressive Neural Network (PNN), Single Unified Network (SUN), BiLSTM, deep learning, aspect detection, sentiment analysis, data augmentation, GermEval-2017, SemEval-2016

Contents

1. Introduction

Currently, computers and machines, in general, only understands binary code. High-level programming languages such as Python and Java just provide means to express logical instructions to machines in a language that has limited similarity to our natural language. Since we are striving for general purpose AI we would unquestionably need much more than this. We would need machines that are not just capable of interacting with humans in natural languages but should also be able to understand the subtle nuances of a language (such as sarcasm, symbolism, metaphors, emotions etc).

Natural Language Processing (NLP), Natural Language Understanding (NLU) and Natural Language Generation (NLG) are scientific fields sitting at the intersection of Artificial Intelligence, Machine Learning, Computational Linguistics, Information Engineering, Data Mining and a lot of other fields. Each of which is trying to solve different parts of the puzzle called human-machine interaction. NLP mainly focuses on techniques on how to process a large amount of data in an effective manner such that it can be easily used by various underlying subtasks. NLU, on the other hand, focuses on techniques which can comprehend and discover meaning from textual data. Lastly, the goal of NLG is to effectively generate coherent text in a natural language. In fact, it won't be incorrect to say that NLU and NLG are subfields of NLP focusing on narrower and more specific goals.

The history of Natural Language Processing dates back to 1950s when Alan Turing in his famous article "Computing Machinery and Intelligence" - (Turing, 1950) talked about the Turing test as a measure of intelligence. The test evaluates how well a machine or a computer program is able to impersonate a real human in terms of written conversational content. Although great advances have been made in this direction since 1950, we are still far away from the ideal situation where a human will not be able to distinguish between a machine and a human generated content. Recently, a lot of personal AI assistants such as Alexa, Siri, Cortana and Ok Google have started showing promising results. Specifically, they have started displaying some human-like traits in speech but characteristics related to comprehension of sarcasm (Poria et al., 2016), humor (Ren and Yang, 2017, Petrović and Matthews, 2013), emotions or sentiments (Rosenthal et al., 2017) are still far from perfection. These highly complex AI systems consist of many much smaller systems that handle different subtasks such as machine translation, sentiment analysis, question answering, text summarization, speech recognition, topic modeling, named entity recognition (NER), coreference resolution, parts of speech tagging (POS) and many more (Collobert et al., 2011). These subtasks have shown tremendous improvements in the last decade or so and this, in turn, has improved the ways in which humans and machines interact.

With the rise of machine learning, increase in computing power and availability of abundant data, various fields such as computer vision, robotics, and pattern recognition have shown unimaginable advances. This has sparked a lot of interest in NLP community as well and recent trends show that NLP research is increasingly moving away from the classical rule-based or statistical methods and focusing more on the use of new deep learning based approaches. In figure 1.1 from Young et al. (2017) we can see how the percentage of the number of deep learning papers in ACL, EMNLP and some other prominent NLP conferences has increased over the last few years.

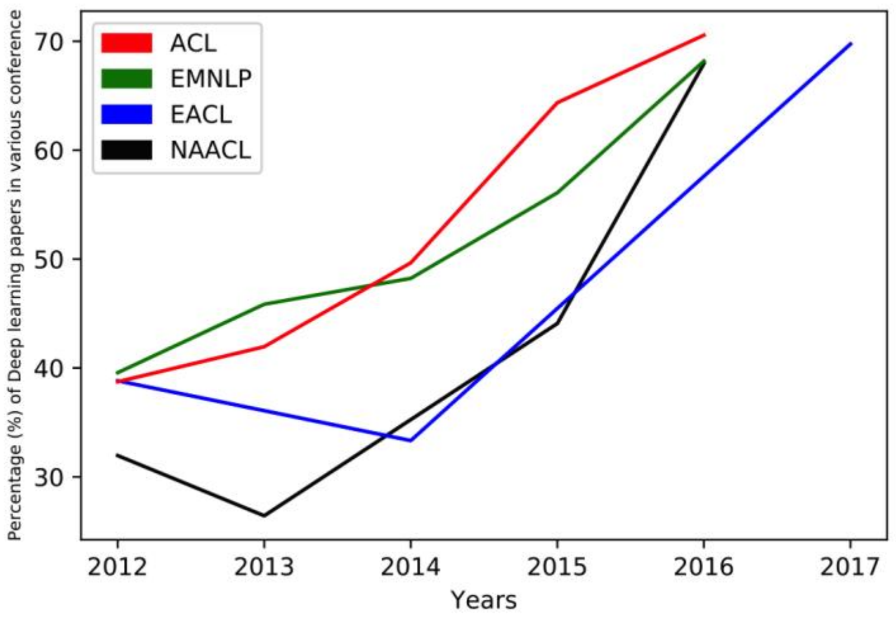


Figure 1.1.: Percentage of deep learning papers in ACL, EMNLP, EACL, NAACL over the last 6 years (long papers). Source: Young et al. (2017)

According to Szathmáry and Maynard Smith (1995) language was one of the most important evolutionary developments in the history of mankind. I believe if we are able to realize this ability for machines, it would be a significant milestone towards the evolution of machines. Just as Industrialization and the Internet changed the ways in which we interact with each other and with our surroundings, this too has a huge potential of impacting our day to day lives.

1.1. Aspect Based Sentiment Analysis

Humans are inherently curious about opinions of other people, be it about products they are interested in buying, leaders they want to elect, job positions they want to apply for or even general gossip about peers. For a long time, it has been an effective way of sharing experiences with one another. Large organizations and companies regard opinion mining as a means to maximize their profit. Understanding what people think about their products, services or their brand helps them to make crucial business decisions (Orestes Appel, 2015). A recent example showing the importance of how public opinions can be used/misused is the Cambridge Analytica-Facebook debacle (Bump, 2018). Cambridge Analytica was apparently successful in shaping people's opinion towards Donald Trump by presenting them with personalized content. This was accomplished with the help of behavioral psychology based on a person's existing opinions about different things. Before the arrival of the Internet, methods such as opinion polls and surveys were generally employed by the companies for obtaining public opinions (Liu, 2010). Nowadays, there is an abundance of raw opinions about any imaginable subject on social media platforms such as Facebook and Twitter. Aspect-based sentiment analysis provides the capability to extract public opinions from this raw unstructured data essentially without any human interference.

Aspect-based Sentiment Analysis (ABSA) is the task of detecting aspects and sentiments expressed towards those aspects from a text given in a natural language. Some basic examples are shown in table 1.1. Traditionally, sentiment has been defined using a discrete polarity scale of positive, neutral, or negative opinion towards something (Mäntylä et al., 2016). However, depending on the requirements it can also be defined on a continuous scale ranging from highly negative to highly positive. Input text can either be in the form of a document representing a review, a comment, or a tweet, or it can be a single sentence. The situation with the definition of an aspect is not much different. Some define it as a single category label e.g - *price, cleanliness*. Others define it as a tuple of an entity and an attribute, for instance, *<food, quality>*, *<graphics, general>*.

In this work, we will adhere to the latter definition.

Input Text	Aspect-Sentiment
The food was lousy - too sweet or too salty and the portions tiny.	FOOD#QUALITY, Negative FOOD#STYLE_OPTIONS, Negative
super fast processor and really nice graphics card..	CPU#OPERATION_PERFORMANCE, Positive GRAPHICS#GENERAL, Positive
Organic is the safest for us and the planet.	ORGANIC_GENERAL#ENVIRONMENT, Positive

Table 1.1.: Some basic examples of aspect based sentiment analysis. First two sentences are taken from SemEval-2016 dataset of restaurant and laptop domain respectively Pontiki et al. (2016a). Last sentence is from our self annotated organic food dataset.

As per Pontiki et al. (2016b), Lakkaraju et al. (2014), Schouten and Frasincar (2016) and many others, extraction of aspects is not considered a primary task for ABSA. According to them, aspect extraction is a separate task which is completed prior to ABSA and hence the primary goal of ABSA is to find the sentiment polarity given the extracted aspect and the text. Wojatzki et al. (2017b), Schmitt et al. (2018) have defined ABSA slightly differently by taking it one step ahead. In their setting, both detection of aspects (out of some finite pre-decided aspects) and sentiments (expressed towards those aspects) are considered primary tasks of ABSA. In this study of work, we will be working with the second formalism.

1.2. Problem Statement

Sentiment Analysis or ABSA is not an easy task to solve given the subtle ways in which humans use natural language. For instance, it is not always the case that we express our opinions using words and phrases in the literal sense. Many times we use complex language constructs such as metaphors, sarcasm for expressing our sentiments. To better understand this point consider the following example sentence taken from Pang and Lee (2008).

> *"If you are reading this because it is your darling fragrance, please wear it at home exclusively, and tape the windows shut." - (a review by Luca Turin and Tania Sanchez of the Givenchy perfume Amarige, in Perfumes: The Guide, Viking 2008.)*

In this sentence, the author has expressed a highly negative sentiment apparently without using any negative words or phrases. So, situations like these make sentiment analysis and ABSA a difficult problem to solve. Additionally, ABSA should not only identify the sentiment but it should also detect the aspect (towards which sentiment was expressed). Generally, expressions and words used to convey sentiments are common across different domains. Words such as - *dazzling, brilliant, phenomenal* convey positive sentiments and words such as - *suck, terrible, awful* convey negative sentiments no matter in which domain they are used. But there could be exceptional situations where the same phrase may express different sentiments for different domains. For instance, consider this example from Pang and Lee (2008).

> *"Go read the book."*

This is likely a positive sentiment for books but probably a negative one for movies. So if the algorithm knows in advance that the current review is talking about the movie domain then, it could make use of this additional information to predict sentiment more accurately. So, how important is the contextual knowledge for correctly identifying sentiments expressed in a sentence? From the last example, it is clear that having some form of contextual knowledge about the subject being reviewed can definitely help in predicting sentiments more accurately. **Hence, how to effectively incorporate contextual information is one of the problems that this work is trying to tackle.**

As already discussed, a lot of companies and organizations see clear advantages of opinion mining for their profitability. This makes ABSA as one of the most widely researched NLP tasks in recent times with widespread applications across multiple domains. We know, it is not very scalable to train and maintain ABSA models separately for individual domains. Also, not every domain has sufficient labeled data. **Therefore, discovering techniques to effectively train an ABSA model with a limited amount of data Or somehow transfer the knowledge of an already trained model (on a source domain) to another model (on a target domain) are some other directions that this study of work in looking into.**

The availability of raw data is much more than it was ever before, yet it is difficult to find labeled data. This is an important blockade not just for NLP but also for other fields that are exploring deep learning methods. Fields like computer vision that mostly rely on image data have developed many augmentation techniques for artificially generating additional data. For textual data, augmentations are not that intuitive and hence only a handful of techniques exists. **This work also tries to study the effects of one of these augmentation techniques on ABSA.**

The number of output classes that the ABSA model has to classify into are generally huge. They also vary a lot from domain to domain. For instance, in SemEval 2016 dataset restaurants domain has around 12 valid aspect classes whereas laptops domain has nearly 116 valid aspect classes (Pontiki et al., 2016a). In ABSA we need to jointly classify both for aspects and sentiments, so this number further increases when we also take into account the possible classes of sentiment polarities. The exact distribution of aspects in the various datasets is presented in appendix A.1. Now, with the limited labeled data, the number of data points per class turns out to be very small. This scarcity of data makes it difficult for neural networks of even decent complexities to learn enough discriminative features for differentiating between hundreds of output classes. **This work is also trying to address this issue that arises out of the huge number of output classes.**

1.3. Proposed Solution

This work is a step towards finding answers to the aforementioned problems. The main contribution of this work is a novel deep learning architecture that is capable of jointly detecting aspects and their corresponding sentiments. The architecture is inspired by Schmitt et al. (2018), Wang et al. (2016), Ruder et al. (2016a), Yang et al. (2016a). Another important contribution of this work is a technique for the transformation/pre-processing of input data. This technique provides some interesting benefits to the proposed model. It helps in addressing some of the problems mentioned above, such as - a huge number of output classes, limitation of labeled data, transfer of knowledge from one domain to another. The technique is shown in figure 4.3.

In order to be context-aware, the network takes the entire comment/review as input. Now, while making predictions for each sentence the network uses contextual knowledge derived from the entire review. The network is implemented using RNN and attention layer which helps in learning the contextual information. For more details about the architecture read chapter 4.

To address the second concern about transferring knowledge from one domain to another a technique called progressive neural networks by Rusu et al. (2016) was explored. To our knowledge, this is the first time this technique has been explored in the context of ABSA. Additionally, because of the input transformation, the output classes of the network were reduced to a fixed number of 4 classes i.e *positive, negative, neutral, not applicable*. So, the number of output classes now neither depends on the domain nor on the number of aspects. This made the network architecture domain independent.

To address the concern about the limitation of training data and its imbalance, a data augmentation technique was explored. In this technique, similar sentences were generated by making use of synonyms or similar words. Additionally, the input transformation also provided a kind of data augmentation. For more details read chapter 4.

The rest of this thesis is organized as follows. In chapters 2 and 3 we will discuss related approaches, followed by a detailed yet brief explanation of background concepts necessary for understanding the rest of the thesis. In chapter 4 we present our methodology, explain our proposed architecture and input transformation. In chapter 5 we briefly discuss various datasets that we have worked with. In chapters, 6 and 7 we discuss experiments and their results, talk about limitations and future works. Finally, in chapter 8 we present our concluding remarks followed by an appendix section.

2. Related Work

In this chapter, we will discuss some of the recent and past approaches that have been applied to the task of aspect-based sentiment analysis or/and sentiment analysis. During our discussion, we will mainly focus on the deep learning based approaches while briefly touching upon other methods as well.

In literature, aspect-based sentiment analysis is generally modeled as a classification problem. Traditional approaches first extract features from the text using labor-intensive hand-crafted feature engineering methods (such as bag-of-words, n-grams, parts-of-speech, negation words etc) (Pontiki et al., 2014, 2016b) and then they simply use a classifier like SVM on top of these extracted features. Soni and Sharaff (2015) is one of the rare papers that uses HMMs for sentiment analysis. For understanding how HMMs can be used to model sentiment analysis read section 3.1.1. A limitation of this approach is that the relationship between reviews and sentiments are hand-crafted and are not learned by the model on its own. Lately, social media users use a lot of abbreviations and emoticons on social media platforms such as Twitter and Facebook. These emoticons, acronyms, ill-formed words are fundamental to microblogging platforms and pose newer challenges for sentiment analysis and related tasks. To overcome these challenges external information in the form of emoticon-to-sentiment-polarity-map, word-sentiment-polarity-map, acronym-dictionary etc are generally employed by a lot of traditional approaches (Agarwal et al., 2011).

After the popularization of word embeddings, the need to use hand-crafted methods for feature extraction has declined. Current approaches mostly rely on the use of these abstract word representation along with various neural network architectures. Since text data is a sequential form of data, so the natural choice of neural network architecture by most approaches is RNN and its variants. RNNs by design are able to handle variable-length sequences and they also have the capability to model long-distance relationships (Tang et al., 2015b). A detailed discussion on how RNNs and its variants are used for modeling sentiment analysis is presented in section 3.1.3. Tang et al. (2015a) uses an LSTM for target-dependent sentiment classification. In this, instead of using a single LSTM for modeling the entire sentence, they use two LSTMs, one for modeling the context prior to the target phrase and another for modeling the context following the target phrase. The output from both of these LSTMs is concatenated before passing into the softmax layer. Ruder et al. (2016a) uses a BiLSTM based hierarchical approach over the entire review for sentence level ABSA. First, a BiLSTM over word vectors is used to model the sentence and then another BiLSTM over sentence vectors is used for modeling the entire review. Additionally, aspect embeddings are concatenated with the sentence representations before they are fed into the sentence level BiLSTM. The intuition behind using a hierarchical model is that the neighboring sentences contain important contextual information which can assist in the sentiment prediction of the current sentence. A limitation of RNNs is that they have a fixed sized vector for storing the context of the entire sequence. In order to overcome this, many recent approaches have started using the attention mechanism. Wang et al. (2016) uses an attention layer over the sequence of words. This was one of the first papers that proposed the use of aspect embeddings by concatenating them with word and/or sentence embeddings. Tay et al. (2017) argues that naive concatenation of aspect vectors is not always desirable. They empirically confirm that this unnecessarily increases the burden of attention layers making the model difficult to train. They propose an additional association layer for modeling the relationship between aspect words and context words.

He et al. (2018), Liu et al. (2016) are some other approaches that use self/inner attention mechanism (see section 3.2) for different NLP tasks. Yang et al. (2016b) goes one step further and uses a hierarchical attention approach for document classification. Attention weights are usually computed by applying softmax over attention scores. It might happen that the attention score of a particular word is very high in comparison to other words and this would cause the model to focus on that particular word more. This might negatively impact the overall model performance. To reduce this effect, Shen and Lee (2016) smooths or normalizes attention scores before computing the attention weights.

Another approach that has shown promising results for ABSA is the use of Memory networks (Weston et al., 2014, Kumar et al., 2015, Sukhbaatar et al., 2015). The central idea of a memory network is to learn a memory component during the training time and then use it for inference. Tang et al. (2016) uses memory network for aspect level sentiment classification. It's a simple neural network (without any LSTMs) with multiple attention layers.

Most of the method consider ABSA as a two-step pipeline process. The first step tries to detect the aspect and the second step does sentiment classification for the detected aspect. Schmitt et al. (2018) models ABSA as a joint end to end task of aspect and sentiment classification. They use a multiheaded architecture where each head corresponds to an aspect and each head tries to detect sentiments corresponding to that aspect. They experimented with two different models using this approach, one with LSTM and another with CNN. According to their experimental results, the CNN version outperformed the LSTM version. Yin et al. (2017) have performed extensive experiments for comparing variants of RNNs and CNNs on different NLP tasks. Ruder et al. (2016b), Mukherjee and Liu (2012) are some of the other papers that uses CNNs for various NLP tasks. In order to understand how a CNN can be used to model the task of sentiment analysis read section 3.1.2.

Just as computer vision, NLP too can be revolutionized by the use of transfer learning and/or domain adaptation techniques. Using word embeddings trained on a huge dataset consisting of text from multiple domains or extracting domain independent features from the text are some of the traditional techniques of applying transfer learning in NLP. For instance, Glorot et al. (2011) uses an unsupervised deep learning approach to extract high-level features from reviews. These features are domain independent and hence are helpful in improving the performance of the sentiment classifier across multiple domains. Balikas et al. (2017) uses a multitask learning kind of approach where they train a model with two classification heads for two different yet similar tasks. The intuition behind this is that the common weights of the model (prior to the classification layer) will learn to represent information from both the tasks and thus will improve the generalization ability of the entire model. Learning without Forgetting (Li and Hoiem, 2016) and Progressive Neural Networks (Rusu et al., 2016) have shown promising results in other fields by overcoming catastrophic forgetting of fine-tuning and multitask learning. It would be interesting to see how well they perform for aspect-based sentiment analysis. Another approach that is increasingly becoming popular nowadays is the use of generalized and context-aware word embeddings such as ELMo (Peters et al., 2018) or ULMFiT (Howard and Ruder, 2018). The idea here is to learn these generic word embeddings by using an ImageNet like data on a general task (that can effectively learn to extract language features which are generic and independent of the domain or/and task) such as language modeling or machine translation. The authors of these papers advocate for language modeling as the source task.

3. Theoretical Background

This chapter gives some theoretical background that is important for understanding the work in this thesis. Concepts such as different ways of modeling sequential data, attention mechanism, word representations, and evaluations metrics are discussed in detail in this chapter.

3.1. Sequence Modelling

A general assumption in most of the machine learning algorithms (especially supervised, including both classification and regression) is that the training samples are drawn independently and identically (iid) from an unknown data distribution (Dundar et al., 2007). In other words, it simply means that the training samples are uncorrelated. For instance, in an image classification task such as the ImageNet challenge, it is assumed that each sample (image, label) is iid. It is typically formalized as follows.

NLP tasks such as language modeling, sentiment analysis, part-of-speech (POS) tagging work with sequential data. Models trained for these tasks often do not consider the iid assumption. Instead, they rely on techniques which can model the dependencies present in sequential data.

As stated by Dietterich (2002), a sequential supervised learning task can be formulated in one of the following ways.

1. **Sequential supervised learning** - In this formulation, each training sample $(\vec{x}_i, \vec{y}_i)$ is a tuple of two sequences, where $\vec{x}_i = \{x_{i,1}...x_{i,t}\}$ and $\vec{y}_i = \{y_{i,1}...y_{i,t}\}$. For example, in POS tagging $\vec{x}_i$ = {*do you want fries with that*} is a sequence of words and the corresponding sequence of tags is given by $\vec{y}_i$ = {*verb pronoun verb noun prep pronoun*} (Dietterich, 2002). So, the aim of this type of model is to correctly predict the label sequence $\vec{y}_i = f(\vec{x}_i)$ given an input sequence $\vec{x}_i$.

2. **Time series** - In this, the goal is to predict the next element x_{t+1} of the sequence, given its previous elements $\{x_1, ..., x_t\}$. A key difference between time series and the sequential supervised formulation is that a time series model makes the prediction x_{t+1} based on the past sequence $\{x_1, .., x_t\}$, whereas a sequential supervised learning model uses the entire sequence $\{x_1, .., x_t\}$ for predicting all the $\{y_1, ..., y_t\}$ values. Another difference is that, in time series formulation, while predicting for the time step $t + 1$ all the true values $\{x_1, .., x_t\}$ up to time t are available, whereas, in sequential supervised learning, no true values $\{y_1, ..., y_t\}$ are given as input.

3. **Sequential classification** - In this, the model has to predict a class label y_i given the input sequence $\vec{x}_i$. For instance, sentiment analysis is generally formulated as a sequential classification problem where given a sequence of words the model has to predict a sentiment expressed in it.

Some commonly used sequential modeling techniques are discussed in the following sections.

3.1.1. Hidden Markov Model (HMM)

According to Ghahramani (2002), Rabiner (1990) Hidden Markov Models (HMM) (Baum and Petrie, 1966) is a statistical tool used to model a dynamic system assuming that the system is a Markovian process with unobserved hidden states. A process is called a Markovian process if its future state is dependent only on the current state and not on the previous sequence of states. An HMM tries to learn a relationship between a sequence of observations $\vec{o}$ and a sequence of hidden states $\vec{q}$. The model is defined using the following three parameters.

1. **Transition matrix** A : Each element of this matrix defines a conditional probability of transitioning to a future hidden state conditioned on the current state.

$$a_{ij} = p(\textit{state } q_j \textit{ at t+1} \mid \textit{state } q_i \textit{ at } t) \tag{3.1}$$

2. **Emission matrix** B : Each element of this matrix defines a conditional probability of an observation conditioned on current state.

$$b_{jk} = p(\textit{observation } o_k \textit{ at } t \mid \textit{state } q_j \textit{ at } t) \tag{3.2}$$

3. **Start matrix** π : It is the initial distribution of hidden states.

Figure 3.1 shows a simple HMM. Here we want to model a relationship between a chosen activity and a weather condition. $\vec{q} = \{Rainy, Sunny\}$ are the possible hidden states. $\vec{o} = \{Walk, Shop, Clean\}$ are possible output observations. Transition matrix (A) is defined by the black arrows between the hidden states. Red and blue arrows between hidden and output states define the emission matrix (B). Arrows from the start node define the start matrix (π). Now, given a sequence of activities such as *Walk, Shop, Walk, Clean, Clean* we can find the most optimum sequence of hidden states using HMM.

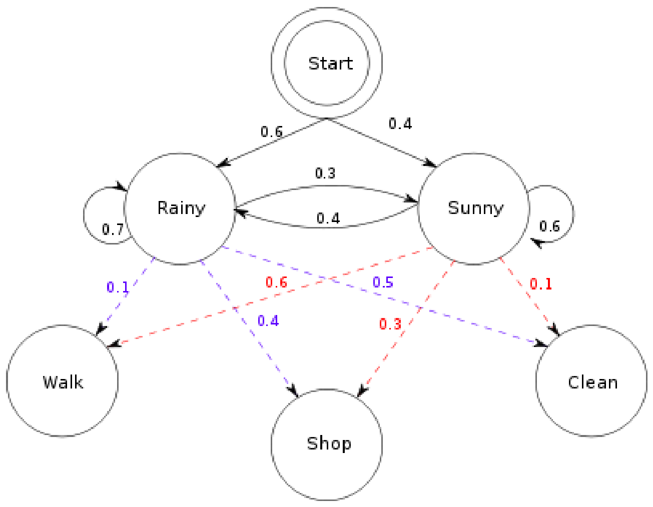


Figure 3.1.: This is a simple example of HMM taken from wikipedia.

The model parameters are learned by deriving maximum likelihood estimation of model parameters given a sequence of output observations. Baum–Welch algorithm, Baldi–Chauvin algorithm, MCMC or algorithms for variational approximations are used for the maximum likelihood estimation. Given the model parameters $\lambda = (A, B, \pi)$ and a sequence of observations $\vec{o}$, we can find $p(\vec{q}|\vec{o}, \lambda)$. After this, we can find the argmax of these posterior probabilities in order to select the best hidden sequence.

Stamp (2004), Moore, Kang (2017) are some tutorial papers and blogs that gives a very good explaination of HMMs.

A framework for sentiment analysis with HMM

Soni and Sharaff (2015) in their paper formulates sentiment analysis in textual data as a Hidden Markov Model. In this approach, sentiment categories are modeled as the hidden states and reviews are modeled as observations.

But there's a small problem with this formalization, the number of possible observation states is huge (because a review can be constructed by any combination of words). As a result, it becomes very difficult to infer an HMM with such a huge number of possible observations. One solution is to somehow restrict the number of possible observations. An interesting approach to do this is to create a fixed number of clusters out of all the review texts and use each cluster centers as the observation states. Soni and Sharaff (2015) specifically use K-means clustering for this. Additionally, they also transform a review text into a review vector (by making use of word embeddings) before applying the clustering algorithm.

3.1.2. Convolutional Neural Network (CNN)

Since their inception in the 1990s by Fukushima (1980), LeCun et al. (1989), Lecun et al. (1998) convolutional neural networks (CNN) have become the de facto neural networks for processing image data. Essentially, they are used for any data having grid-like topology i.e time-series data (1D grid), language modeling using NLP (1D and 2D grid) (Goodfellow et al., 2016). Apart from the fully connected and activation layers used in a regular neural network, CNNs also have a couple of other specialized layers such as convolutional layers and pooling layers. The main motivation of using these additional layers is their ability to provide local or sparse interactions, share parameters and work with variable sized input.

Unlike regular neural networks, CNNs arrange their neurons in a three dimensional (height, width, depth) structure. As seen in figure 3.2 every neuron in the convolution layer operates over a local volume of the 3D input and transforms it into a 3D output. This locality of the interaction (also called the receptive field) is decided by the filter size and the size of the stride taken while moving along the input space (height, width). Although a filter is spatially (i.e along height and width) local, it always operates across the full depth.

The output spatial size (O) is mathematically related to the input spatial dimension (W), the receptive field size of convolution filter (F), stride with which filters are applied (S) and the amount of padding used (P) and can be easily calculated using equation 3.3. Additionally, the depth of the output layer is decided by the number of filters applied.

$$O = \frac{W - F + 2P}{S + 1} \tag{3.3}$$

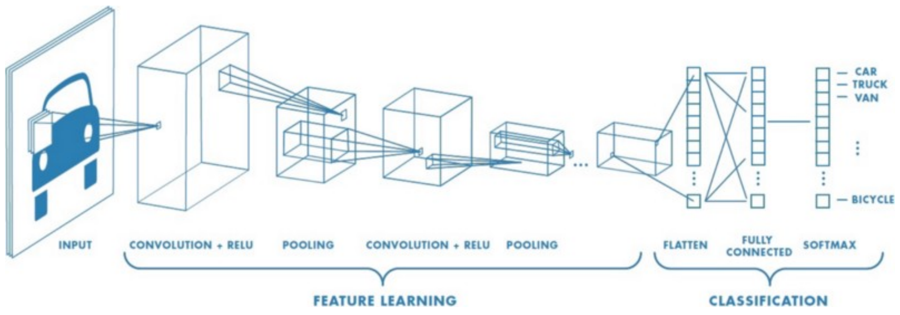


Figure 3.2.: By Prabhu (2018). This is a block diagram of a CNN. In this, a convolution layer consists of filters which are convolved with an input's local volume and each convolution of a filter generates one activation for the output layer. On the other hand, pooling layer is just used for dimensionality reduction. Finally, after a continuous stacking of convolution and pooling layers, a fully connected layer is applied for generating class label probabilities.

Karpathy (2016), Prabhu (2018), Pokharana (2016) are some blogs that explain CNNs very intuitively.

A framework for sentiment analysis with CNN

With an already proven track record in vision-related tasks, acceptance of CNNs in NLP has been pretty smooth (Young et al., 2017). Their ability to provide local interactions is made use of for modeling short term relations in a sequence. Besides this, stacking multiple convolution layers can also model long term dependencies to a limited extent.

Zhang and Wallace (2015) does an in-depth analysis of CNNs for text classification problems. In particular, they provide a framework for sentiment detection using a CNN. For this, fixed length $(d = 5)$ word vectors (of every word in a given text) are stacked together to represent a sentence in a matrix form (of dimension 7x5) as shown in figure 3.3. These word vector can either be learned or a pre-trained version can be used. Now, a convolution layer with various filter sizes (2,3,4) is applied to the input sentence matrix. Since rows of the sentence matrix represent word vectors, so it is reasonable to use filters with widths equal to the dimensionality of the word vectors (i.e., d) and hence only the height of the filters are varied. Then, a pooling layer is applied on the activations from filters and the pooled vectors (6 univariate values in this case) are concatenated together to form a single fixed length feature vector (of size 6 in this case). Finally, a softmax layer is applied to the feature vector for classification. Additionally, we can stack many of such convolution and pooling layers to create a deeper architecture and hence learn more distant dependencies within a given text.

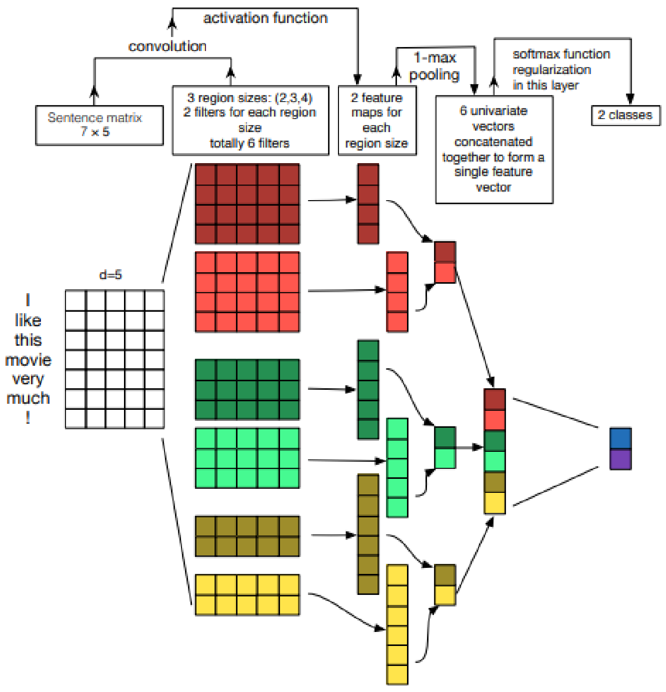

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 3.3.: A framework for text classification with CNN by Zhang and Wallace (2015).

3.1.3. Recurrent Neural Networks (RNN)

Much as CNNs are specialized for processing image data, Recurrent Neural Networks or RNNs (Rumelhart et al., 1988) are specialized for processing sequential data (Goodfellow et al., 2016). Natural Language is inherently sequential, from alphabets and words to sentences and paragraphs all are arranged in a sequence of arbitrary lengths. RNNs ability to model sequences of arbitrary lengths and to remember long distance interactions (at least in theory) makes them ideal for various NLP tasks. In fact, the semantic meaning of words becomes clear by considering previous words in a sentence and similarly, the context of a sentence becomes apparent based on the neighboring sentences.

Figure 3.4(a) illustrates the unfolding of a simple RNN for an input sequence $\{x_0 ... x_t\}$. Each fold in time t can be considered as a hidden layer of a regular neural network.

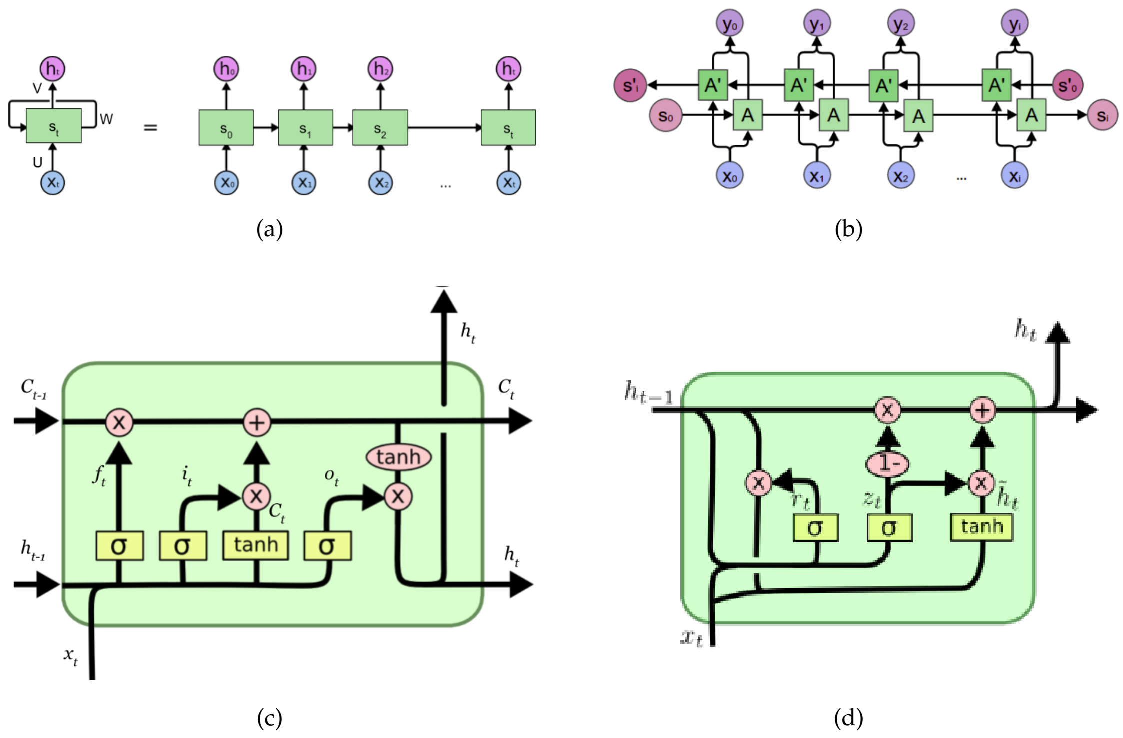


Figure 3.4.: Source : Colah (2015), Gandhi (2018) (a) Unrolling of an RNN (b) Bidirectional RNN (c) LSTM cell (d) GRU cell

However, unlike regular neural networks where each hidden layer has its own weights, RNNs share the same weights across all the folds. It is because of this parameter sharing that RNNs have the ability to handle inputs of arbitrary lengths. As shown in the figure 3.4(a), x_t is the input to the network at time t and s_t represents the cell state at time t. Intuitively, a cell can be regarded as the memory unit which implicitly stores information about previous elements processed in the sequence. s_t depends on the current input x_t and the previous state s_{t-1} as per equation 3.4. h_t is the output at step t and it depends on the current state as per equation 3.5. Functions $fandg$ are usually nonlinearities such as tanh or ReLU. If multiple RNN layers are stacked on top of each other than h_t acts as the input for the next RNN layer. U, V, W are the network parameters that are shared across the time steps. The mathematical formulation of an RNN is as follows.

$$s_t = f(U.x_t + W.s_{t-1}) \tag{3.4}$$

$$h_t = g(V.s_t) \tag{3.5}$$

In practice, however, vanilla RNN suffers from the infamous vanishing and/or exploding gradient problem and this makes it really hard to effectively learn the parameters. This becomes a major problem especially for the initial layers of the network and hinders model's ability to learn long-term dependencies. Some of the variants of RNN that overcome this limitation are LSTM (Hochreiter and Schmidhuber, 1997, Gers et al., 2000, 2002) , GRU (Cho et al., 2014).

Long Short-Term Memory (LSTM)

The mathematical formulation of an LSTM is given by the following equations.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{3.6}$$
$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{3.7}$$
$$\widetilde{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C) \tag{3.8}$$
$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}_t \tag{3.9}$$
$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{3.10}$$
$$h_t = o_t * tanh(C_t) \tag{3.11}$$

Like vanilla RNNs, LSTMs also unfold across the sequence, the only difference lies in the unfolding unit called the LSTM cell. The thing that makes an LSTM cell better equipped for handling the problem of vanishing gradients is the way it updates its cell state. As shown in figure 3.4(c) the update of the cell state is controlled by three gates, namely the forget gate f_t, input gate i_t and the output gate o_t. The goal of the forget gate f_t is to selectively forget information from the previous cell state. This is achieved by first creating a forget mask from the current input x_t and the previous output h_{t-1} as per equation 3.6 and then applying this forget mask on the previous cell state C_{t-1} as per equation 3.9. In the next step, the input gate i_t decides what new information needs to be added to the previous cell state C_{t-1}. For this, initially, a keep mask is created using the current input x_t and the previous output h_{t-1} as per equation 3.7. Then, an intermediate new cell state $\widetilde{C}_t$ is computed using the current input x_t and the previous output h_{t-1} as per equation 3.8 and the keep mask is applied to this intermediate cell state $\widetilde{C}_t$ in order to select the elements that are worth keeping. Finally, this new intermediate cell state $\widetilde{C}_t$ is added to the previous cell state C_{t-1}. In summary, previous cell state C_{t-1} is transformed into the new cell state with some minor linear interactions with the outputs of forget and input gates. These interactions allow the gradients to flow deeper in time. Lastly, the output gate is used for calculating the current output as per equations 3.10 and 3.11.

Gated recurrent units (GRU)

The mathematical formulation of GRUs is as follows.

$$z_t = \sigma(W_z.[h_{t-1}, x_t]) \tag{3.12}$$
$$r_t = \sigma(W_r.[h_{t-1}, x_t]) \tag{3.13}$$
$$\widetilde{h_t} = tanh(W.[r_t * h_{t-1}, x_t] + b_C) \tag{3.14}$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \widetilde{h_t} \tag{3.15}$$

Another popular variant of vanilla RNN is Gated Recurrent Units (GRUs). Unlike RNNs and LSTMs they do not maintain an explicit memory or cell state instead they combine cell state and hidden state into a single unified state h_t as shown in figure 3.4(d). This and some other simplifications in it's structure makes GRUs less complex than LSTMs. Due to this decreased complexity, they sometimes perform better than LSTMs in situations with limited data (Young et al., 2017).

This brings us to the difficult question of choosing between LSTMs and GRUs. Empirical results from a lot of comparative studies by Greff et al. (2015), Józefowicz et al. (2015), Chung et al. (2014) have shown that neither of them is the absolute superior. Depending on the task and the amount of data one variant may perform better than the other.

Bidirectional Recurrent Neural Networks (BiRNN)

RNNs and its variants introduced in the previous paragraphs, model dependencies in input sentences only in a single direction. In other words, the cell state s_t depends only on the previous cell state s_{t-1}, but not on the next state s_{t+1}. As a result, the memory units cannot derive any context from the next token, even though the current token significantly derives its meaning from the next one. To illustrate, consider these two sentences from Gandhi (2018).

> "He said, Teddy bears are on sale" and "He said, Teddy Roosevelt was a great President."

By just looking at the word *Teddy* and its previous words it is not possible to know whether the context is about president or bears. A dependency on the future tokens might help to resolve this ambiguity. Bidirectional RNNs (Schuster and Paliwal, 1997) resolve this issue. As shown in Figure 3.4(b), bidirectional RNNs contain an additional layer of hidden units with reversed connections so that they can model dependencies in both the directions. Different RNN variants such as LSTMs and GRUs can be used in the bidirectional sense.

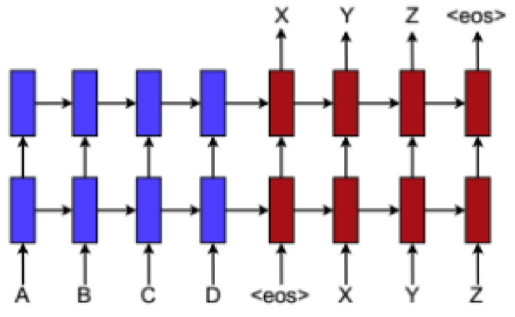
Colah (2015), Karpathy (2015a), Britz (2015) are some blogs that explains RNNs very intuitively.

A framework for sentiment analysis with RNN

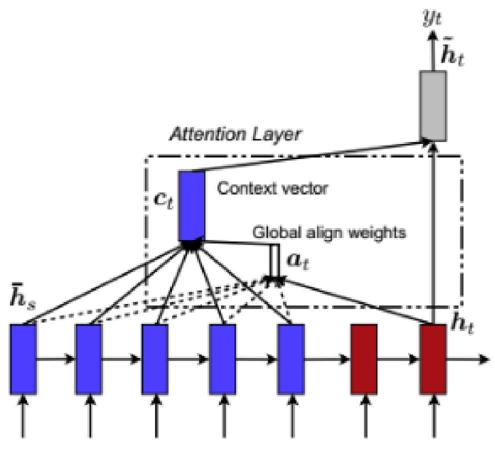
RNNs and its variants are specifically designed for modeling sequential tasks. A lot of NLP tasks such as language modeling, sentiment analysis, machine translations are sequential in nature. In the case of sentiment analysis, an input sequence generally consists of either words or sentences. The input to the RNN is generally a vectorized form of tokens (words or sentences) of a sequence. Pre-trained word embeddings are used for computing word or/and sentence vectors. An RNN is unrolled sequentially on these tokens and results from RNN are transformed into a fixed size vector representing the entire sequence. Finally, a classification layer is applied on this vectorized sequence representation for sentiment detection. A more detailed framework for aspect-based sentiment analysis is discussed in chapter 4.

3.2. Attention Mechanism

The term attention is inspired by the human way of focusing on a smaller subset of things that are most important for the end goal. The need for attention mechanism becomes more clear by looking at some of its recent applications by Bahdanau et al. (2014), Luong et al. (2015) in the domain of Neural Machine Translation (NMT), which translates an input sequence from one language to another. RNNs and its variants are generally employed for modeling the task of NMT. To illustrate, figure 3.5(a) shows a two stack RNN architecture for translating a source sequence A, B, C, D into a target sequence X, Y, Z. It's an encoder-decoder architecture where the blue part encodes the information from the input sequence into a fixed sized hidden vector. Intuitively, hidden vectors are like memory states where the context of the sequence processed so far is stored. By this intuition, the context of the entire input sequence is stored in the last hidden vector. The decoder(red part) then uses this last hidden vector to generate the output translations. A limitation of this approach is the fixed size of the context vector (as the decoder is solely dependent only on the last hidden state). It limits the model's ability to encode the entire context of a really long sequence into a fixed dimensional vector. Attention mechanism overcomes this limitation by creating a context vector c_t as a weighted mean over all or some of the previous hidden states, instead of just using the last hidden state as the context vector. These weights are called attention weights and they represent a probability distribution over the hidden vectors (and thus implicitly a distribution over input tokens too).



(a)



(b)

Figure 3.5.: Source : Luong et al. (2015) (a) A 2 stack RNN architecture for translating a source sequence A, B, C, D into a target sequence X, Y, Z. Here, $< eos >$ marks the end of a sequence. It is an encoder(blue part)-decoder(red part) architecture. (b) The attention layer computes a variable length attention/align weight vector $\vec{a_t}$ based on the current target hidden state h_t and all source hidden states $\vec{h_s}$. Then, a global context vector c_t is computed as the weighted mean over all source states using the attention weights.

Now, by comparing the current target hidden state h_t with all the source hidden states $\vec{h_s}$ a score is computed as per equation 3.16 and then attention weights are computed by normalizing these scores using the softmax function as shown in equation 3.17. Essentially the learned attention weights $\vec{a_t}$ decides how much attention to pay to the source hidden states. Figure 3.5(b) shows what computations are performed in an attention layer. $\widetilde{h_t}$ is specific to Luong et al. (2015) and can be ignored in this situation.

$$\vec{s} = score(h_t, \vec{h_s}) \tag{3.16}$$

$$a_t = \frac{\exp(s_t)}{\sum_t \exp(s_t)} \tag{3.17}$$

$$score(h_t, \vec{h_s}) = h_t^T . \vec{h_s} \tag{3.18}$$

$$c_t = \sum_t a_t . h_{s,t} \tag{3.19}$$

In tasks such as sentiment analysis or document classification where there are no target hidden states for attending source hidden states, a slight variation of attention mechanism called Inner Attention (Liu et al., 2016) or Self Attention (Lin et al., 2017, Shen and Lee, 2016) is used. In this, a learnable semantic vector w is compared with all the source hidden states for computing the score as per equation 3.20. These scores are then normalized using softmax. Intuitively, this semantic vector w usually learns to focus on specific parts of a sentence.

$$\vec{a} = softmax(score(w, \vec{h_s})) \tag{3.20}$$

Yang et al. (2016b) applies self-attention hierarchically first on sentences to attend important words and then on paragraphs to attend important sentences for a document classification task.

3.3. Word Representations

Majority of the neural networks are only capable of taking numerical data as input. With NLP where the goal is to analyze strings, it becomes important to find an effective way of representing words with real-valued vectors. One straightforward way is to make a vocabulary of unique words (say of size N) from the documents that you want to analyze and then represent each word using a one-hot vector (of dimension N). This method has certain drawbacks, firstly, it is not very scalable if the size of the vocabulary N becomes very large. Secondly, every word is equidistant from every other word in the one-hot encoding space (because each word just occupies one dimension) and hence there is no grouping of semantically similar words such as "good", "great". A more useful representation would be where the meaning of the words along with their semantic and contextual relationships are also taken into account when computing the word vector/embedding. Some other methods that try to overcome these limitations are discussed below.

C&W Model

C&W model uses the idea that a language model trained on a large dataset are able to capture syntactic and semantic relationships of words within the corpus. Generally, language models optimize cross entropy loss by maximizing the probability of a word given the previous words in the sequence. But, to calculate this probability one also needs to compute a normalization factor using all the words in the vocabulary and this is computationally expensive. So, as a workaround, C&W model (Collobert et al., 2011, Collobert and Weston, 2008) used a pairwise ranking criteria (as shown by equation 3.21) as the training objective. In this approach, given a window x ($\in X$ set of all possible windows in corpus) of a sequence they generate a corrupted window x^w by replacing the center word with a random word from the vocabulary V. Upon using this objective the model will try to assign a higher score for the correct window $f_\theta(x)$ than the incorrect window $f_\theta(x^w)$. In summary, C&W is a language model that can be trained efficiently.

$$J_\theta = \sum_{x \in X} \sum_{w \in V} max\{0, 1 - f_\theta(x) + f_\theta(x^w)\} \tag{3.21}$$

Word2Vec

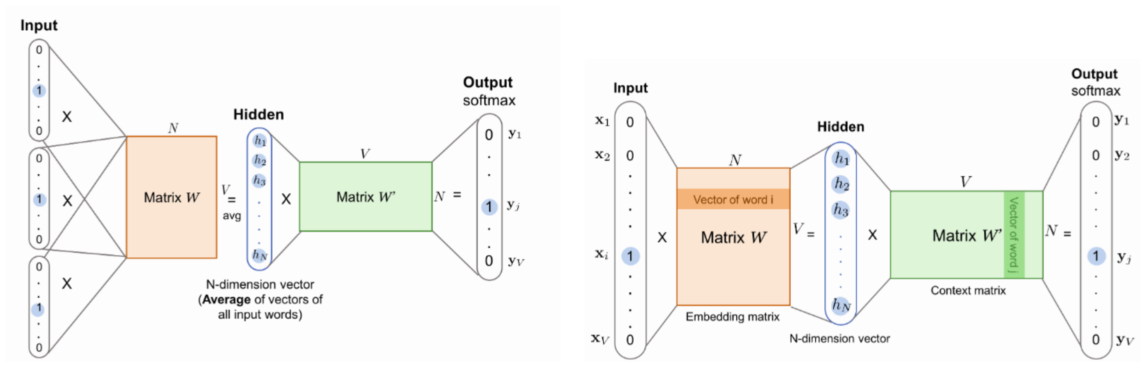
In contrast to C&W, Word2Vec does not use a deep learning approach. Instead, it uses a simple neural network that too without any non-linearities. This simplification in the model and some other training strategies drastically increased the training speed and the quality of the generated word embeddings. Mikolov et al. (2013a,b) proposed two approaches to train Word2Vec models - CBOW and Skip-gram.

Continuous bag-of-words (CBOW)

In this approach, a target word w_t is predicted based on a continuous fixed sized (k) window of words around it. Unlike the language model where only the previous words define the context, in CBOW both previous $(w_{t-k}...w_{t-1})$ and future $(w_{t+1}...w_{t+k})$ words are used to define the context of the target word. Particularly, the model maximizes the probability $p(w_t|w_{t-k}...w_{t-1}, w_{t+1}...w_{t+k})$ of occurrence of the target word given the context words. Figure 3.6(a) shows a simple CBOW model. Let us assume that we have a vocabulary of size V and we want word embeddings of size N. The input to the model is one-hot encoded vectors (each of size V) of context words and the output is also a vector of size V depicting the predicted target word. The input is mapped to a hidden layer of size N using an embedding matrix W of dimension (V, N). There are no non-linearities after the hidden layer. Context matrix W' of dimension (N, V) maps the hidden layer to the final output layer. Also, the projections of various context vectors are averaged to form a fixed length hidden vector.

Skip-gram

This approach is slightly different from CBOW. Here, instead of predicting the target word, the model learns to predict the context words. Particularly, the model maximizes the probability $p(w_{t-k}|w_t)$ of occurrence of a context word given the target word. Figure 3.6(b) shows a simple skip-gram model. All the assumptions and dimensions



(a) (b)

Figure 3.6.: Source : Weng (2017) (a) CBOW model (b) Skip-gram model

are same as CBOW model. Each combination of a target-context tuple within every sliding window forms a training data point. Whereas, in the case of CBOW one slide of window forms only one training point.

Global Vectors (GloVe)

Inherently, GloVe (Pennington et al., 2014) is not very different from Word2Vec. One of the main contributions of this paper is to give a theoretical explanation about the fundamentals of the two Word2Vec methods. According to the authors, prediction-based methods such as Word2Vec implicitly makes use of the underlying word co-occurrence statistics and a count-based method such as GloVe does that explicitly. The first step of the GloVe algorithm is to compute a word co-occurrence matrix X (of a fixed vocabulary size V) by observing word pairs in the corpus that appear together. For every occurrence of a word pair (i, j) an entry X_{ij} in the co-occurrence matrix gets updated. A decreasing weighting factor based on how far (in terms of the number of words between them) the two words are is also computed. This ensures that the distant word pairs that contain less relevant information contribute a little less towards the co-occurrence matrix. Using the global statistics of X GloVe optimizes a least square objective function defined by equation 3.22. In this equation, w_i and w_j are columns of embedding matrix W and context matrix W' respectively and b_i and b_j are scalar biases. f is a weighting function that ensures that frequent and rare co-occurrences get appropriate importance.

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - log(X_{ij}))^2 \tag{3.22}$$

The model is trained by randomly sampling non zero elements from X. After the training, sum of W and W' are used as word embeddings. In order to compute word embeddings of some different dimensionality Word2Vec has to be trained from scratch after changing its embedding dimensionality where GloVe can simply reuse the already computed co-occurrence matrix.

Selivanov (2018), ŘEHŮŘEK (2014) are some blogs that explains GloVe very intuitively.

FastText

FastText (Bojanowski et al., 2016, Joulin et al., 2016) is an extension of the Word2Vec skip-gram model. In this approach, each word is represented by a bag of its character n-grams and the word itself. Instead of a single target word w_t, now, this bag of n-grams is used to train the skip-gram model. These n-grams allow the fastText model to make use of the internal word structure which was ignored my many previous approaches. Additionally, by using morphology of words fastText can easily compute words representations for out of vocabulary (OOV) words. This is possible because it is highly likely that some of the n-grams of OOV words also appear in other words. In particular, word embedding for a word is computed by summing the word representations of all its n-grams.

Embeddings from Language Models (ELMo)

ELMo (Peters et al., 2018) emphasize the importance of contextualized word embeddings over normal word embeddings such as Word2Vec and fastText. In order to encode contextual information, a deep bidirectional language model (biLM) is trained on a large corpus. Internal states of this model are used for computing the word embeddings. In other words, the vector representation of a word is a function of the entire input sequence (sentence or paragraph). Since the language model takes input character-wise, so, similar to fastText, ELMo is also able to make use of sub-word information and effectively compute vector representations even for OOV words. ULMFit (Howard and Ruder, 2018) also uses a similar approach. One major difference between the two approaches is that ULMFit fine-tunes the model on target task whereas ELMo concatenates embeddings generated from its language model to the target task network. Another difference with other contextualized word representations such as CoVe (McCann et al., 2017) is that they just use the final hidden representation as the embeddings whereas ELMo linearly combines hidden representations from all the layers to generate the final embeddings. The authors of ELMo also hypothesize that lower layers of their language model learn more syntactic features whereas higher layers learn semantic features.

So far, the above paragraphs have discussed ways of computing word embeddings. A lot of document/sentence classification tasks require sentences to be represented as vectors. A simple and strong baseline for this is to take the average of word embeddings (Arora et al., 2017). Until recently, unsupervised methods such as Skip-Thought (Kiros et al., 2015) have been the state of the art for generating sentence embeddings. This approach is very similar to the skip-gram method of Word2Vec. It predicts the neighboring sentences instead of the neighboring words. Nowadays, supervised methods have also started showing promising results. For instance, methods such as InferSent (Conneau et al., 2017), Universal Sentence Encoder (Cer et al., 2018) have achieved state of the art results. The general idea of these methods is to train a deep neural network for a supervised task that is generic enough to learn the underlying features and characteristics of a natural language. To illustrate, InferSent uses natural language inference and Universal Sentence Encoder uses machine translation as supervised tasks for learning generic sentence embeddings. Wolf (2018) intuitively explains the current best word and sentence embeddings.

3.4. Evaluation Metrics

Evaluation metrics help in understanding how well a particular algorithm is performing on a given dataset for a given task. It provides a common ground for the comparison of different algorithms. It is really important to choose the correct evaluation metric depending on the type (classification or regression) of the algorithm, the distribution of data (balanced or unbalanced) and the property you want to evaluate for. Some of the commonly used evaluation metrics in case of classification are discussed below.

Accuracy

It is the ratio of the number of correct predictions to the total number of predictions made. It is one of the most common and generally misused evaluation metrics. It only makes sense to use this metric when the data is balanced with respect to the number of data points per class. In the case of an unbalanced dataset, the results from this metric can be misleading about the true performance of the algorithm. For instance, in a credit card fraud detection dataset which has 99% data points within the negative class (without fraud) and only 1% in the positive class, if an algorithm just predicts negative class for each data point then it will have a very good classification accuracy of .99. However, the results will be highly misleading about the true learning capabilities of the algorithm. Moreover, the metric considers the different prediction errors such as

- false positive, false negative with equal consideration. Some other metrics described below might be more suitable for an in-depth analysis of these different prediction errors.

Confusion Matrix

It represents the count of different prediction errors such as true positive (TP), false positive (FP), false negative (FN) and true negative (TN) in a more visual and easy to understand matrix form. Figure 3.7 shows a two-class confusion matrix. Precision is defined as the ratio of correctly predicted true classes (TP) and all of the predicted true classes (TP+FP). The ratio of correctly predicted true classes (TP) and all the actual true classes (TP + FN) is called recall. Ideally, a good classification algorithm should have high recall and high precision score.

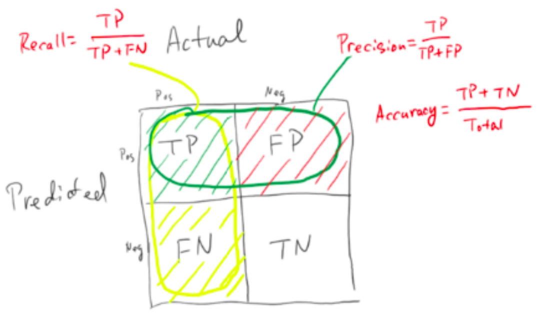


Figure 3.7.: Source: Narkhede (2018). A two-class confusion matrix. Rows of the matrix represent the predictions and columns represent the actual labels.

F1 Score

F1 score is the harmonic mean of recall and precision as per equation 3.23. It is also called F score and F measure. Intuitively, it is a single unified metric that represents a balance between recall and precision. This becomes handy when you are unsure whether the model should have a high recall or high precision or when both are equally important.

$$FMeasure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.23}$$

Macro F1 Score

In the case of multiclass classification (i.e more than two classes) it is not possible to define recall and precision collectively for all the classes. Instead, an average measure is defined. In case of macro average, recall and precision are defined separately for all the classes, then an average of all these individual recalls and precisions is taken. Finally, an F measure is defined using these macro averaged recall and precision. Intuitively, in this case, each class is given equal statistical importance irrespective of the number of data points per class. Because of this, an important thing to keep in mind is that the overall macro average score might get impacted by the statistics of the best performing class. In my opinion, it is safe to assume that most of the time the best performing class is the one that is in majority in the dataset. Therefore, it is not the best metric to use in case of a highly imbalanced dataset.

Micro F1 Score

Contrary to the macro averaging, in this case, TP, TN, FP, FN from all the classes are summed together first and then recall, precision and F1 measure are calculated with these summed values. Due to the way recall and precision are calculated in a multi-class setting, their values are always the same in case of micro averaging (Hessner, 2018). In this case each sample data point is given equal statistical importance irrespective of the class they belong to. Because of this, an important thing to keep in mind is that the overall micro average score will get impacted by the statistics of the majority class. Therefore, it also might not be the best metric to use in case of a highly imbalanced dataset (Narasimhan et al., 2016). We did not investigate a lot in this regard and instead went ahead with the popular belief of using micro average measure in case of a multiclass classification problem on an unbalanced dataset. Although, a weighted average based on the class imbalance could have been another good metric.

Multi-Label Micro F1 Score

In the case of a multi-class multi-label setting for evaluating micro F1 measure, each sample is first split into as many observations as there are labels in it (Zhang and Zhou, 2014). To illustrate, consider an example where a sample looks like this (x, y_1, y_2). Here x can be classified as both y_1 and y_2. Now, this single sample is split into two observations (x, y_1) and (x, y_2). After computing the observation set from all the samples, the multi-label micro F1 score can be calculated by simply applying multi-class micro F1 statistic (explained in the previous paragraph) on this observation set. In other words, in this case, each observation is treated as a separate sample data point.

4. Method

This chapter is divided into three parts. The first part of the chapter describes the proposed architecture and the input transformation used. Next, we will talk about the different methods used for combating the problem of data imbalance. Finally, the third section talks about transfer learning approaches.

4.1. Proposed Architecture

Aspect-based sentiment analysis (ABSA) is a challenging task. Some of its challenges are common to other NLP tasks as well. As already stated in chapter 1, our proposed architecture tries to address the following problems.

1. How to incorporate contextual information into the model?
2. How to effectively train a multi-class, multi-label classification model especially when the number of possible output classes is huge and when the amount of annotated data is scarce?
3. How to transfer the knowledge from a model trained on a source domain to another model to be trained on a different target domain?

Apart from this, another direction that this study of work looks into is a data augmentation technique for textual data.

The main contribution of this work is a novel deep learning architecture that is capable of jointly detecting aspects and the corresponding sentiments. The proposed architecture is shown in figure 4.4 and is inspired by these papers Schmitt et al. (2018), Wang et al. (2016), Ruder et al. (2016a), Yang et al. (2016a). Most of the current ABSA approaches use one of the following ways for classifying aspects and sentiments.

1. Some approaches use a multi-head architecture where the model has a dedicated

head for every possible aspect. The goal of each head is to predict sentiment polarities with respect to a specific aspect. Scalability of this approach becomes a problem with the increasing number of possible aspects. Another drawback of this approach is that different domains usually have a different set of possible aspects and with this architecture, it is impossible to use the same model for multiple domains without first modifying the heads.

2. Some other approaches use a huge softmax layer with all possible classes (which is a product of all possible aspects with all possible sentiment polarities) as the output layer. With this approach, the number of possible output classes can explode and become a huge problem for training, especially when the data is limited. Additionally, with this approach too, it is impossible to use the same model with multiple domains without first modifying the last softmax layer. Another minor issue with this approach is that it can not handle multi-label scenarios out of the box. A workaround generally employed is to threshold the output of softmax layer to generate multiple labels for a given input.

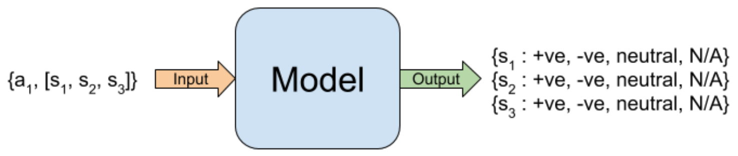

{a1, [s1, s2, s3]}
Input
Model
Output
{s1 : +ve, -ve, neutral, N/A}
{s2 : +ve, -ve, neutral, N/A}
{s3 : +ve, -ve, neutral, N/A}

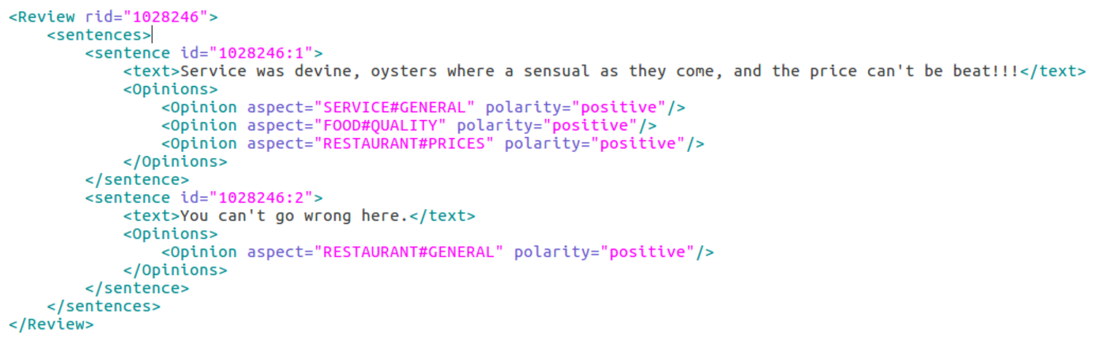
Figure 4.1.: This figure shows the input-output structure of our model. The input is a tuple of an aspect a_1 and a comment which is a sequence of sentences s_1, s_2, s_3. This tuple represents one data point. More data points are generated by checking every comment with every possible aspect of the domain. This is achieved with the help of input transformation. The output classes include $+ve$ sentiment, $-ve$ sentiment, *neutral* sentiment or N/A if the aspect a_1 is not discussed in that particular sentence. Output labels are generated for each sentence in the input comment.

Our proposed architecture tries to overcome the limitations of the above-mentioned approaches. The purpose of our model is to jointly detect aspects and sentiment expressed towards the detected aspects. This is done for every sentence in a comment or review. To this end, the model takes the tuple {*aspect, comment*} as input and tries to detect whether the sentences contain the given aspect and if yes, then, what sentiment is expressed towards the aspect. A comment is checked against all the possible aspects (of a particular domain) and the aggregated results inform us which aspects are present with which sentiment. Figure 4.1 shows the structure of input and output for our model. The raw training data is first transformed in a format that is acceptable by our model. This transformation additionally also provides some interesting benefits which will be discussed in the next section.

4.1.1. Input Transformation

An important contribution of this work is a technique for the transformation/pre-processing of input data. This technique provides some interesting benefits to our proposed model. It helps in addressing some of the problems that arise because of a huge number of output classes or limitation of labeled data. It also provides transfer learning capabilities to our model.

The datasets for ABSA generally have the format as shown in figure 4.2. In this format, a review consists of multiple sentences where each sentence may or may not have multiple aspect-sentiment mappings. Most approaches in the literature use this data as it is with some minor adjustments. For instance, if a sentence has multiple aspect mappings, they simply duplicate the sentence such that each copy of the sentence now has only one unique aspect mapped to it. This strategy works well if each sentence is analyzed independently. But if one decides to make use of contextual information from neighboring sentences then these duplicate sentences might create some issues. Our proposed input transformation doesn't face this dilemma.



```
<Review rid="1028246">
    <sentences>
        <sentence id="1028246:1">
            <text>Service was devine, oysters where a sensual as they come, and the price can't be beat!!!</text>
            <Opinions>
                <Opinion aspect="SERVICE#GENERAL" polarity="positive"/>
                <Opinion aspect="FOOD#QUALITY" polarity="positive"/>
                <Opinion aspect="RESTAURANT#PRICES" polarity="positive"/>
            </Opinions>
        </sentence>
        <sentence id="1028246:2">
            <text>You can't go wrong here.</text>
            <Opinions>
                <Opinion aspect="RESTAURANT#GENERAL" polarity="positive"/>
            </Opinions>
        </sentence>
    </sentences>
</Review>
```

Figure 4.2.: A sample data point from SemEval-2016 Restaurant dataset

Figure 4.3 explains the input transformation technique with some dummy data. Let us assume that this data belongs to some domain D which has $A = 3$ possible aspects

a_1, a_2, a_3. Let the original input data consists of N reviews and each review can have any number of sentences. Each sentence can contain sentiments about multiple aspects. Now, consider the first review which has three sentences $\{s_1, s_2, s_3\}$ with the following aspect-sentiment mappings.

- $s_1 : (a_1, +ve)$
- $s_2 : (a_2, -ve)$
- $s_3 : [(a_1, +ve), (a_2, -ve)]$

Now, after applying the transformation, a review will be transformed into as many reviews as there are possible aspects, i.e 3 in this case. In other words, we started with N reviews or data points and after applying the transformation we will end up with $A * N$ data points in total. After transformation, a data point is described as a collection of an aspect, a review, and a list of sentiments for each sentence against the given aspect. For instance, the first review of our dummy dataset will have these three transformations.

- $Transformation1 : (a_1, [s_1, s_2, s_3], [+ve, n/a, -ve])$. It means that sentence s_1 expresses a positive $+ve$ sentiment and sentence s_3 expresses a negative $-ve$ sentiment towards the aspect a_1. n/a for sentence s_2 means that it doesn't talk about aspect a_1 at all.
- $Transformation2 : (a_2, [s_1, s_2, s_3], [n/a, -ve, -ve])$. It means that sentence s_2 and s_3 express negative $-ve$ sentiment towards the aspect a_2 and sentence s_1 doesn't talk about a_2 at all.
- $Transformation3 : (a_3, [s_1, s_2, s_3], [n/a, n/a, n/a])$. Here it means that none of the three sentences s_1, s_2, s_3 contains any information about the aspect a_3.

After input transformation is performed on all the reviews, a tuple consisting of an aspect and a list of sentences serves as the input to the model. The list of sentiment polarities serves as the output labels. To illustrate, if our raw data had only consisted of a single review, then, the transformed dataset would look something like below.

- Input Data : $\{(a_1, [s_1, s_2, s_3]), (a_2, [s_1, s_2, s_3]), (a_3, [s_1, s_2, s_3])\}$
- Output Labels : $\{([+ve, n/a, -ve]), ([n/a, -ve, -ve]), ([n/a, n/a, n/a])\}$.

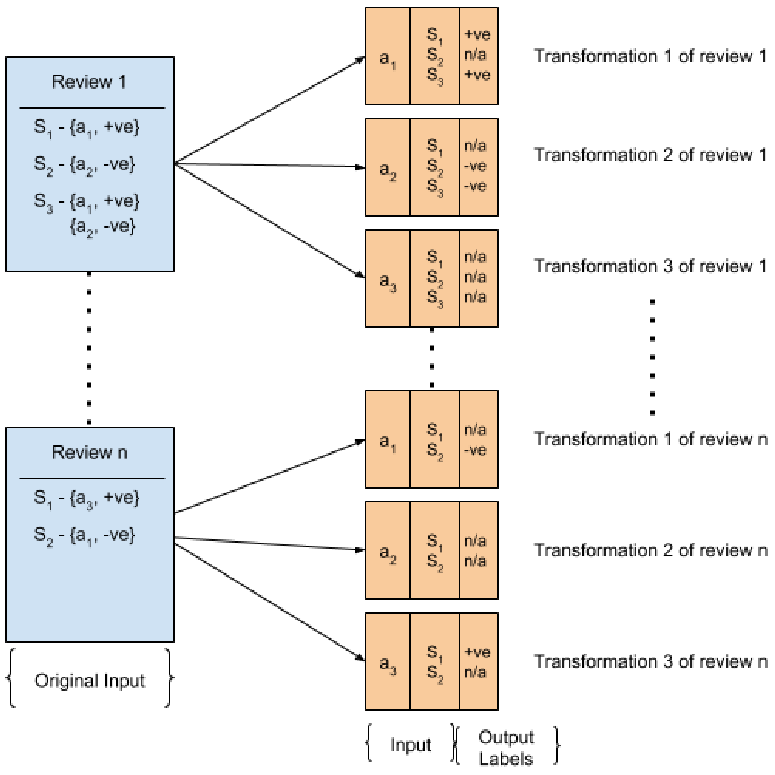

Review 1
S1 - {a1, +ve}
S2 - {a2, -ve}
S3 - {a1, +ve} {a2, -ve}
Review n
S1 - {a3, +ve}
S2 - {a1, -ve}
Original Input
a1 | S1 S2 S3 | +ve n/a +ve
a2 | S1 S2 S3 | n/a -ve -ve
a3 | S1 S2 S3 | n/a n/a n/a
a1 | S1 S2 | n/a -ve
a2 | S1 S2 | n/a n/a
a3 | S1 S2 | +ve n/a
Transformation 1 of review 1
Transformation 2 of review 1
Transformation 3 of review 1
Transformation 1 of review n
Transformation 2 of review n
Transformation 3 of review n
Input
Output Labels

Figure 4.3.: An example explaining the transformation of the input data. Let us assume that this data belongs to some domain D which has three possible aspects a_1, a_2, a_3. Let the original input data consists of N reviews and each review can have any number of sentences. Each sentence can express sentiments about multiple aspects. Now, consider the first review which has three sentences $\{s_1, s_2, s_3\}$ with these $\{(a_1, +ve), (a_2, -ve), [(a_1, +ve), (a_2, -ve)]\}$ aspect-sentiment tuples respectively. Now, after transformation one review will be transformed into as many reviews as there are possible aspects, 3 in this case. The transformation is done such that each transformed data point will be a collection of an aspect, a list of sentences and a list of sentiment polarities for these sentences against the aspect given in this collection. For instance, review 1 will have three transformations like this - $\{(a_1, [s_1, s_2, s_3], [+ve, n/a, -ve]), (a_2, [s_1, s_2, s_3], [n/a, -ve, -ve]), (a_3, [s_1, s_2, s_3], [n/a, n/a, n/a])\}$. The n/a stands for not applicable. It means that, that particular sentence did not express any sentiment with respect to the given aspect. From every collection, aspect and list of sentences serve as the input to the network. List of sentiment polarities serves as the output labels.

Some advantages of processing the data in this format:

- With this approach, the number of output classes has been fixed to these four classes $(+ve, -ve, neutral, n/a)$. Their count no longer depends on the number of possible aspects. Additionally, with the number of output classes being fixed we can train the same model on different domains without performing any modification. This enabled us to build a Single Unified Network (SUN) which we trained on a dataset consisting of data points from different domains such as restaurant, laptops and organic food. We will see later that this unified network performed better than other networks which were trained separately on each domain.

- With this approach, handling of multi-label scenarios is very straight forward. The network learns to predict n/a for a sentence if it doesn't contain a given aspect and it predicts one of the sentiment polarities if the aspect is present in the sentence. In other words, it's not a multi-label classification anymore and we don't even need to perform any kind of thresholding on the output layer.

- A common technique of handling a sentence having multiple aspects is to create duplicate copies of this sentence for each aspect that is present in it. This technique has one drawback that it can limit the ability of a model to learn good contextual information. Our model, on the other hand, can handle this situation more gracefully. In our model a review is already duplicated for each aspect while performing the input transformation so, our network by design does not require any sentence duplication. As a result, the context of the review remains intact because there is no sentence duplication within a review.

- This input transformation strategy also provides a kind of data augmentation. By duplicating the entire review for each possible aspect we are essentially oversampling the reviews.

- Since we are explicitly including aspect words with each review, our model can make use of this extra information to learn similarity or association between an aspect and the words of a sentence. This helps in aspect detection.

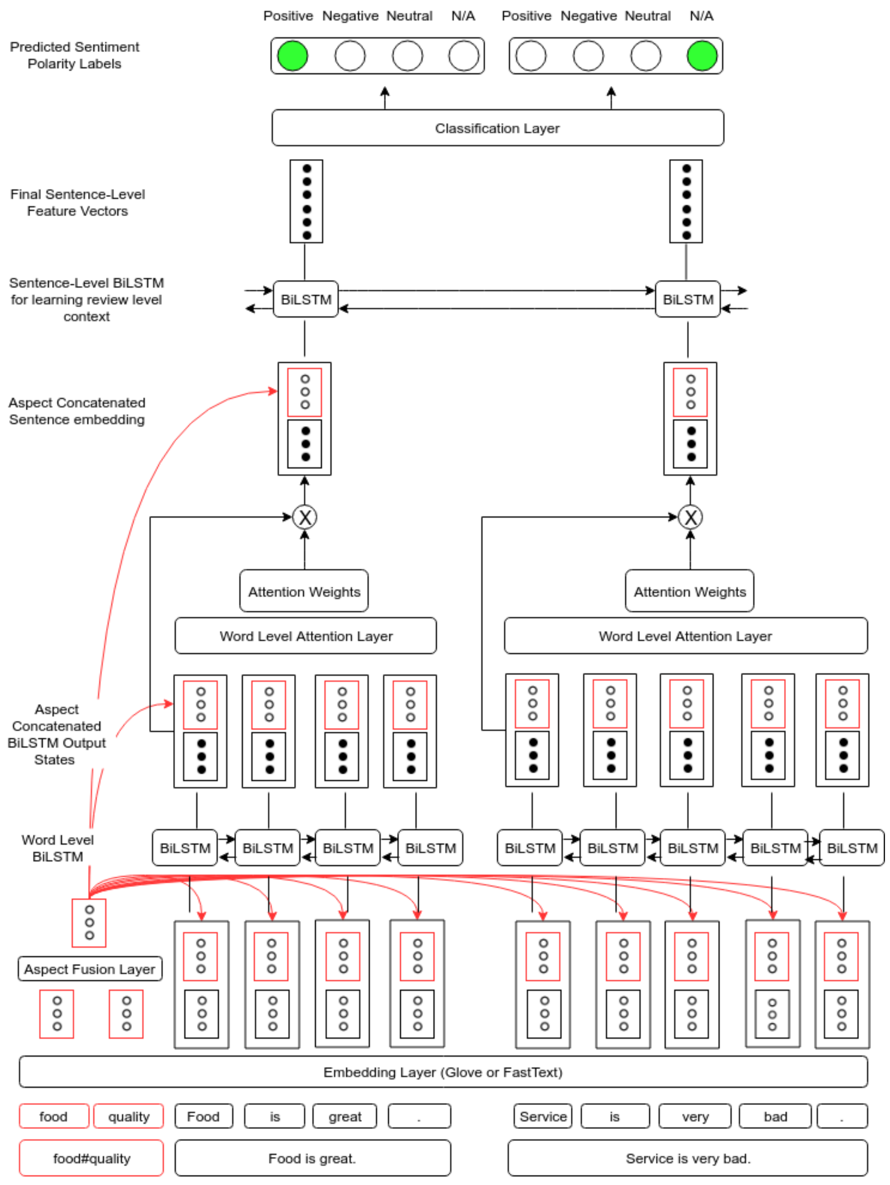

Positive Negative Neutral N/A
Positive Negative Neutral N/A
Predicted Sentiment Polarity Labels
Classification Layer
Final Sentence-Level Feature Vectors
Sentence-Level BiLSTM for learning review level context
BiLSTM
BiLSTM
Aspect Concatenated Sentence embedding
Attention Weights
Attention Weights
Word Level Attention Layer
Word Level Attention Layer
Aspect Concatenated BiLSTM Output States
Word Level BiLSTM
BiLSTM
Aspect Fusion Layer
Embedding Layer (Glove or FastText)
food
quality
Food
is
great
.
Service
is
very
bad
.
food#quality
Food is great.
Service is very bad.

Figure 4.4.: Proposed neural network architecture for aspect-based sentiment analysis. Red colored boxes represent aspect vectors. In this figure a review has two sentences = ['Food is great.', 'Service is very bad.'] and we are trying to analyze both the sentences against the aspect 'food-quality'. First sentence has positive sentiment for this aspect whereas this aspect is not present in the second sentence.

4.1.2. Architecture

Our proposed architecture as shown in figure 4.4 uses BiLSTM (some reasons why we preferred BiLSTMs over other sequence modeling approaches such as HMMs and CNNs will be presented towards the end of this section) for modeling sequential data. The model uses a two-level hierarchical approach which is similar to the approach used by Ruder et al. (2016a). Section 1.2 talks about the importance of contextual information for accurate sentiment detection. The hierarchical nature of the model along with the BiLSTMs helps in better understanding of the context. In other words, the intuition behind using a hierarchical model is that they help to capture the context by combining information from neighboring words and sentences.

The input to the network is a tuple of an aspect (in the form of a list of word ids) and a review (in the form of a matrix of word ids, where each row represents a sentence). The input first goes through an embedding layer and with the help of an embedding matrix E, a 2D array of aspect word vectors A_e and a 3D array R_e of words in the review are computed. The embedding matrix E is generated using pre-trained word embeddings such as GloVe or fastText. E can either be fine-tuned during the training process or it can be kept fixed. Now, the aspect word embeddings are fused together to generate a fixed-sized aspect vector $\vec{a}$. This fusion of aspect word embeddings to generate an aspect vector can be done in many different ways. For instance, we can perform a mean pooling or max pooling of all the aspect words or we can use a fully connected layer for transforming aspect word vectors to a fixed-sized aspect vector $\vec{a}$ using equation 4.1. We have performed experiments with all the above three methods.

$$\vec{a} = \begin{cases} mean(A_e) & \text{mean pooling} \\ max(A_e) & \text{max pooling} \\ tanh[W_a.A_e] & \text{projection via fc layer} \end{cases} \tag{4.1}$$

Now, the aspect vector $\vec{a}$ is concatenated to every word vector $\vec{r_{ij}}$ of the review matrix R_e. This generates an aspect concatenated review matrix R_{ae} as per equation 4.4. This idea was first used by Wang et al. (2016). The intuition here is to somehow inject aspect information into the network so that the model can learn to find associations between the provided aspect and the input sentences. Although there's no mathematical backing for why this approach works but the positive experimental results from a lot of papers in the literature provides sound support towards this approach.

$$R_e = [\vec{r_{11}}, .., \vec{r_{ij}}] \text{ , where } r_{ij} \text{ is word vector of } j^{th} \text{ word in the } i^{th} sentence \tag{4.2}$$
$$\vec{r_{ij}}^{ae} = [\vec{r_{ij}} : \vec{a}] \tag{4.3}$$
$$R_{ae} = [\vec{r_{11}}^{ae}, ..., \vec{r_{ij}}^{ae}] \tag{4.4}$$

Next, we model the sequence of words (i.e a sentence) using a word-level BiLSTM as per equations 4.5, 4.6, 4.7, 4.8. The input to this word-level BiLSTM is a sequence of aspect concatenated word embeddings of a sentence i from the matrix R_{ae}. The output H_i of word-level BiLSTM are again concatenated with the aspect vector $\vec{a}$ as per equation 4.9, 4.10. This concatenation is a bit redundant and therefore we tried to train the model without it too. But the performance was a bit better with this concatenation so we decided to keep it.

$$H^{word} = [H_1^{word}, ..., H_i^{word}] \tag{4.5}$$

where H_i^{word} is output of i^{th} sentence from word-level BiLSTM.

$$R_{ae}^i = [\vec{r_{i1}}^{ae}, ..., \vec{r_{ij}}^{ae}] \tag{4.6}$$

where R_{ae}^i is aspect concatenated word embeddings for sentence i.

$$H_i^{word} = BiLSTM_{word}(R_{ae}^i) \tag{4.7}$$
$$H_i^{word} = [\vec{h_{i1}}^{word}, ..., \vec{h_{ij}}^{word}] \tag{4.8}$$
$$\vec{h_{ij}}^{word:aspect} = [\vec{h_{ij}}^{word} : \vec{a}] \tag{4.9}$$
$$H_i^{word:aspect} = [\vec{h_{i1}}^{word:aspect}, ..., \vec{h_{ij}}^{word:aspect}] \tag{4.10}$$

The output states of an RNN acts as memory units. They tend to remember the context of the sequence they have seen in the past. In case of a sentence, the output state of every word represents the context that the RNN has learned so far by observing the previous words. The size of the output vector is fixed and the sentence length can vary a lot. As a result, it is not always possible to effectively represent the context in a fixed sized vector, especially for longer sentences. The reason for this is that longer sentences may have a lot of unnecessary words which may add noise to the context. In order to overcome this, we have used attention mechanism over the words of a sentence. The purpose here is to create a distribution over words based on their relevance or importance with respect to the current context. This distribution $\vec{\alpha}_i$ is called the attention weights. During the computation of attention weights $\vec{\alpha}_i$, first, a learnable semantic vector $\vec{w}$ is compared with all the source states $H_i^{word:aspect}$ for computing the score as per equation 4.11.

$$\vec{Score_i} = score(H_i^{word:aspect}, \vec{w}) \tag{4.11}$$

$$score(H_i^{word:aspect}, \vec{w}) = \begin{cases} H_i^{word:aspect}.\vec{w} & \text{dot} \\ tanh(W_\alpha.H_i^{word:aspect}).\vec{w} & \text{projection via fc layer} \end{cases} \tag{4.12}$$

$$Score = [\vec{Score_1}, ..., \vec{Score_i}] \tag{4.13}$$

These scores are then normalized using softmax to get the attention weights as per equation 4.14. Intuitively, this semantic vector $\vec{w}$ usually learns to focus on specific parts of a sentence.

$$\alpha_i = softmax(\vec{Score_i}) \tag{4.14}$$

$$\alpha = [\vec{\alpha}_1, ..., \vec{\alpha}_i] \tag{4.15}$$

Finally, in order to generate a sentence representation $\vec{s_i}$ for the i^{th} sentence, we take a weighted mean of it's source states $H_i^{word:aspect}$ with $\vec{\alpha}_i$ being the weights.

$$\vec{s_i} = H_i^{word:aspect}.\alpha_i \tag{4.16}$$

$$S = [\vec{s_1}, ..., \vec{s_i}] \tag{4.17}$$

Now, in order to extract more contextual information we treat these sentence vectors as a sequence and pass them through a sentence-level BiLSTM. But before doing this we again inject aspect information by concatenating aspect vector $\vec{a}$ with the sentence vectors S that we got from the attention layer. Finally, the output $F = [f_1, ..., f_i]$ from the sentence-level BiLSTM acts as the feature vector for a sentence. A classification layer is applied on top of this feature vector to compute a classification probability over these four output classes $\{+ve, -ve, neutral, n/a\}$.

$$\vec{s_i}^{setence:aspect} = [\vec{s_i} : \vec{a}] \tag{4.18}$$

$$S^{setence:aspect} = [\vec{s_1}^{setence:aspect}, ..., \vec{s_i}^{setence:aspect}] \tag{4.19}$$

$$F = BiLSTM_{sentence}(S^{setence:aspect}) \tag{4.20}$$

Now, we will briefly discuss some reasons why we preferred BiLSTMs over other sequence modeling approaches such as HMMs and CNNs.

Why not HMM?

HMM are generally employed in speech based sentiment analysis tasks. The amount of research utilizing HMM for textual sentiment analysis task is very limited (Soni and Sharaff, 2015). Comparative studies such as Panzner and Cimiano (2016) claims that HMM perform better than RNN and its variants under conditions where only few data points or limited computing power is available. In spite of this, we couldn't find any paper that uses HMM for ABSA. Following are some of the possible reasons why RNNs are preferred over HMMs for most NLP tasks.

- State space in HMMs is discrete and has fixed number of states whereas in RNNs hidden variables can take continuous values. So by design the state space for RNNs is a lot bigger and better at describing the current context (YeGoblyn-Queenne, 2016, Hinton, 2013). This also allows RNNs to keep track of long term dependencies in a better way than HMMs. For instance, check experiments from these blogs Goldberg, Karpathy (2015b). Here they are trying to predict linux-kernel code and it shows that HMMs are not effective in learning long-term relations between opening and closing brackets but vanilla RNNs perform remarkably well at this task, thus proving the point. Although theoretically it is possible to introduce any number of states in HMM to make the state space more rich and expressive but practically it becomes unscalable.

- Secondly, HMMs rely on linear transformations whereas RNNs use non linearities that allows them to update their hidden states in complicated ways (Hinton, 2013) which enables them to learn more information.

- Thirdly, RNNs are more flexible in terms of input-output sequences they can process. For instance, RNN can be to used to model problems with input-output relations in any of the following forms - one-to-one, one-to-many, many-to-many, many-to-one.

Why not CNN?

Due to CNN's ability to model local interactions, they have some shortcomings. Apparently, they are not that good in modelling long-distance dependencies for contextual understanding of a text (Young et al., 2017). Another issue generally faced while using CNNs is their ineffectiveness in preserving the sequential order of the data (Kalchbrenner et al., 2014). Also, Yin et al. (2017) performed extensive experiments on comparing variants of RNNs and CNNs on different NLP tasks and they concluded that RNNs generally perform better and are robust in many of the NLP tasks. In spite of these limitations there are many papers in the literature that have shown better results with CNNs than with RNNs and its variants.

4.2. Combating Data Imbalance

Data imbalance is a common problem generally faced by classification algorithms. Data imbalance refers to the uneven distribution of data points within the classes in a given dataset. In other words, a dataset is considered imbalanced when its classes are represented unevenly. For example, let us assume we have a binary classification problem with a dataset of 100 data points. An imbalanced dataset could mean that out of these 100 data points 80 or 90 belong to one class and the rest belong to the other class. The same logic also applies to a multi-class classification problem. The problem with the unbalanced dataset is that it can make the neural network to learn to trivially classify every data point in the majority class. On top of this, if an evaluation measure (like accuracy) which is not robust against data imbalance is used to gauge the performance of the model then it might even give a false impression about the model's learning capabilities.

Let's discuss some of the commonly used tricks and methods that we have used for tackling the class imbalance problem.

Collect More Data

If possible collect more data points for minority classes. This will help in making our data more evenly distributed. We lacked resources required for annotating new data points so, we decided against this approach.

Data Resampling

We can either oversample the data points from minority classes by duplicating them or we can do a negative sampling of the majority class by randomly removing some of the data points belonging to the majority class. Due to the complicated structure of our data, we couldn't apply this method. In our data, a review is considered as input whereas the output labels are predicted for every sentence within that review. So effectively every sentence is a data point. Since our model tries to make use of the contextual information from neighboring sentences so it didn't make sense to randomly duplicate sentences belonging to minority class or remove sentences from the majority class because this would have changed the context of the review.

Data Augmentation

There are many augmentation techniques that can be applied to image data. For instance, we can use rotation, cropping, coloration, flipping or mirroring effects for generating synthetic data. However, very few data augmentation techniques exist for textual data. Machine translation is one trick that is sometimes used for generating synthetic text data. In this approach, a source text is first translated to a target language and then, the translation is retranslated back to the source language. Assuming that the translation algorithm is good enough and is not perfect, we can generate semantically and contextually similar back-translations of the source text. Ideally, these back-translations are like paraphrases of the source text. Another approach that was also used by Zhang and LeCun (2015) in their paper was to generate synthetic text by replacing words with synonyms or similar words. We experimented with both synonyms and similar words. In order to replace words, we needed to find answers to two questions. Firstly, we needed to find words that we want to replace. For this, we decided to replace the adjectives and nouns of a sentence. This is because it is relatively easy to replace these words without interfering a lot with the syntax, grammar or meaning of the original sentence. To illustrate, consider this example review - *"Food is great. Service is very bad."*. Here *Food*, *Service* are nouns and *great*, *bad* are adjectives. If we replace these words with similar or synonym words we get an augmented review that is almost similar in meaning to the original review - *"Meal is good. Facility is very poor."*. We have used spaCy an NLP library, for filtering out adjectives and nouns from a sentence by using spaCy's state of the art parts of speech (POS) tagger. Now, the second question that we needed to answer is how to replace these selected words. For this, we have experimented with the following two techniques.

1. **Synonyms** - We used a dictionary for finding the synonyms. Out of all the possible synonyms for a given word, we only selected those synonym words which had the same POS tag as the word that needs to be replaced. For example, a noun word was replaced by its noun synonym and an adjective word was replaced by its adjective synonym. For finding the synonyms, we have used another NLP library called NTLK which provides an interface for the wordnet dictionary. This interface, not only provides synonym words, but it also gives their POS tags. Some example augmentations generated with this approach are shown in table 4.1. Unfortunately, the augmentations generated using this approach look rather poor. The meaning and grammar of the augmented sentences seem to have distorted a lot from the original sentence. One possible reason for this could be that since the dictionary is not aware of the context and as we know that a word can be used differently in different contexts so, sometimes it might happen that the synonyms we get from the dictionary are not fit for the current context. Another reason could be that the implementation of the dictionary in the NLTK library is not up to the mark.

Original Text	Augmentations using synonyms
The food was lousy - too sweet or too salty and the portions tiny.	The nutrient was lousy - too sweet or too salty and the portions tiny. The food was lousy - too sweet or too salty and the fortune tiny. The food was lousy - too sweet or too salty and the parcel tiny. The intellectual nourishment was lousy - too sweet or too salty and the portions tiny.
Ambiance- relaxed and stylish.	Ambiance- relaxed and fashionable.
I'm very happy with this machine!	I'm very happy with this auto! I'm very happy with this automobile!
Great price and computer!	Great price and calculator! Great Mary Leontyne Price and computer! Great price and information processing system! Great monetary value and computer!

Table 4.1.: Some examples of the augmentations that were generated by replacing words with synonyms.

2. **Similar Words** - For generating similar words, we have used another NLP library called Gensim which has a build in wrapper that gives a list of similar words along with their similarity scores. The wrapper uses a pre-trained word embedding such as GloVe or fastText in its backend and it uses cosine similarity to search for similar words within this embedding space. Some example augmentations generated with this approach are shown in table 4.2. These augmentations are comparatively better than the augmentations we got from the synonym approach. One possible reason for this improvement could be that the context of similar words generally matched with the context used in the original sentence. The reason for this could be that the dataset used for training word embeddings might contain words which are used in a similar context as used in our dataset. One downside of this approach is that sometimes similar words returned from this wrapper can also have opposite meanings. The reason for this is that opposite words such as good and bad generally occupy semantically similar positions in the word embedding space of GloVe or fastText. When we used this method to augment our minority classes $\{+ve, -ve, neutral\}$, we observed a decline in the overall performance of the model. This probably happened because many of

the positive looking sentences got transformed into negative looking sentences due to the augmentation. Due to this reason, we decided to use this technique only for augmenting neutral sentences. This resulted in some performance gains. This decision was based on the assumption that neutral sentences express both positive and negative sentiments and so even if we replace a word with another word of opposite sentiment it would have little effect on the overall sentiment of this sentence.

Original Text	**Augmentations using similar words**
The food was lousy - too sweet or too salty and the portions tiny.	The food was crappy - too sweet or too salty and the portions tiny. The food was lousy - too sweet or too salty and the portions small. The food was lousy - too sweet or too salty and the portions large. The snacks was lousy - too sweet or too salty and the portions tiny.
Ambiance- relaxed and stylish.	Ambiance- relaxed and fashionable. Ambiance- quiet and stylish. Ambiance- relaxed and sleek.
I'm very happy with this machine!	I'm very always with this machine! I'm very happy with this equipment! I'm very glad with this machine!
Great price and computer!	Great discount and computer! Amazing price and computer! Great price and laptop!

Table 4.2.: Some examples of the augmentations that were generated by replacing words with similar words.

Based on this augmentation technique we have also created a small standalone python library called *simsent* which can be used as an augmentation toolkit for generating similar looking sentences or paraphrases.

Class Weighted Loss

This is another common way of tackling the problem of data imbalance. In this technique, the loss penalization is more if a model makes a wrong classification for a minority class. This can inspire the model to learn to be extra cautious when classifying data points belonging to minority classes. In order to penalize the loss, we multiply it with weights. These weights are computed using the inverse statistics of the class distribution. As a result minority class gets a higher weight than the majority class and consequently a misclassification for a minority class will cost more. We experimented with the following two methods for computing these weights.

- **Static Weights** - These weights are computed using the inverse class statistics of the entire dataset and hence we call them static weights.
- **Dynamic Weights** - These weights are dynamically computed during every iteration using the inverse class statistics of the current batch and hence we call them dynamic weights.

4.3. Transfer Learning Approaches

Transfer learning can either imply knowledge transfer from one task to another task or it can mean knowledge transfer from one domain to another domain. Universal or contextual word embeddings such as ELMo, ULMFiT are good examples of the first form of transfer learning where knowledge learned from a source task such as - language modeling is used to enhance the performance of the target tasks such as machine translation, document classification or aspect-based sentiment analysis. In this work, we will concentrate on the second approach where the knowledge acquired for the task of aspect-based sentiment analysis from a source domain (say restaurants) will be used to enhance the performance of the same model for the same task but for another target domain (say laptops). Some of the common transfer learning approaches employed in NLP are discussed in chapter 2. Out of these approaches, we essentially focus on Progressive Neural Network (PNN) because of its immunity against catastrophic forgetting. In fact, to our best knowledge, ours is the first model that has used progressive neural networks for aspect-based sentiment analysis. Besides this, as a side advantage of our proposed architecture, we are also able to train a single instance of our model on a multi-domain dataset. What this means is that a Single Unified Network (SUN) simultaneously trained on multiple domains can be used for inference on individual domains. In the following sections, we will discuss the details of these two approaches.

4.3.1. Progressive Neural Network (PNN)

Fine tuning is the most commonly used technique of transfer learning. In this technique, a model is initially pre-trained on a source domain (or task) and then the output (generally the final classification layer) layers are fine-tuned on the target domain (or task). This is done by backpropagating through the output layers. During this process, the weights of the initial layers are generally frozen. Intuitively, what fine tuning does is that it provides better and more meaningful initialization of the weights. This, in turn, helps the model to converge quickly on the target domain, with lesser data, and with supposedly improved performance. But it is a destructive process for the previously learned connections. In other words, fine tuning on a target domain causes the model to forget prior knowledge learned from the source domain.

Progressive Neural Network (PNN) (Rusu et al., 2016) is a neural network architecture that improves upon some of the limitations of transfer learning. With the help of lateral connections from a stack of networks which were trained on some source tasks, Progressive Neural Networks (PNNs) tries to extract useful features for the target task. The weights of these lateral connections are learned by the new network. Because of this, the network gets some extra freedom to decide how much prior knowledge it wants to take from the previous tasks for the current task. By accessing previous knowledge in this manner, PNNs are able to create a reliable balance between the current and previous information. These lateral connections are can be applied to different layers of the network. As a result of this design, PNNs are immune to catastrophic forgetting and have the ability to solve all the learned tasks with optimum performance at the inference time.

Figure 4.5 shows a simple block diagram of a progressive neural network. In this, the first two columns represent the pre-trained networks which will be used to train the third network. Progressive neural networks can also be used in a continuous learning paradigm. That is, for instance, the second network, in this case, is trained using the first network and then the third network is trained using the first two networks.

Formally, a PNN starts with a neural network having L layers with output activations h_i^1, where $i \leq L$. The parameters θ^1 of this network are trained till convergence. Now, when training the second network (second column) parameters θ^1 of the first network are kept frozen and the parameters θ^2 of the second network are randomly initialized. Every layer h_i^2 of the second network receives input from the previous layer of the same network h_{i-1}^2 and from the previous layer of the earlier network h_{i-1}^1 via lateral connections. A generalized equation for K columns is shown below.

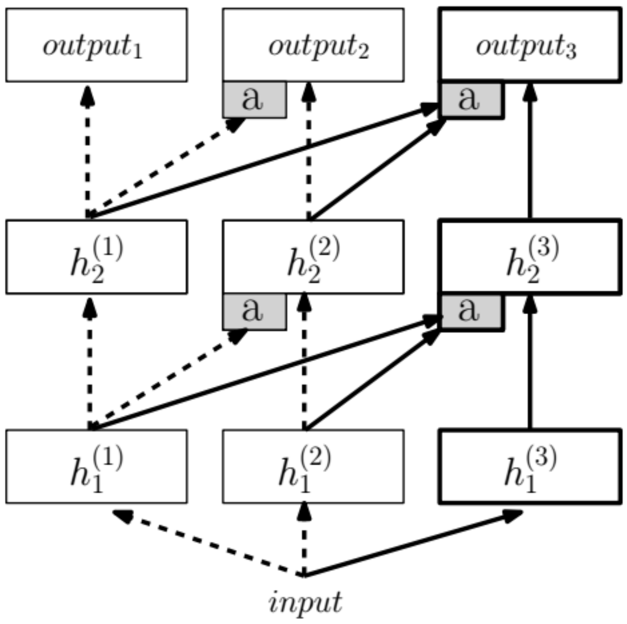


Figure 4.5.: Source : Rusu et al. (2016) A block diagram of three columned progressive neural network. The grey boxes labeled as *a* represents the adapter layers. Network from the second column is trained using the first network and then the network from the third column is trained using the first two columns. All three networks are optimized for different tasks.

$$h_i^k = \sigma(W_i^k h_{i-1}^k + U_i^{k:j} \sigma(V_i^{k:j} \alpha_i^{<k} h_{i-1}^{<k})) \tag{4.21}$$

Here W_i^k is the weight matrix of the i^{th} layer of the k^{th} column and $U_i^{j:k}$ is the weight matrix of lateral connections from j_{th} column to k_{th} column. In practice, instead of linear lateral connections, non-linear lateral connections called adapters are used. Adapters help in improving the lateral connectivity and they also help in reducing the number of parameters as K grows. In equation 4.21 matrix $V_i^{k:j}$ controls the dimensionality of lateral activations. For a network with dense layers, we can simply pass the lateral activations $h_{i-1}^{<k}$ (from a previous column) through a fully connected layer (with non-linearity) and then make a lateral connection with the layers of the next column. Sometimes, in practice, lateral activations are multiplied with a learnable scalar α_i before feeding them into the fully connected layer. This helps in adjusting the scales of different inputs.

Progressive Neural Networks for Aspect-based Sentiment Analysis

To the best of our knowledge, ours is the first approach that has used progressive neural networks for aspect-based sentiment analysis. We have used a PNN architecture with non-linear lateral connections along with learnable scalars α_i for the appropriate scaling of lateral activations. We experimented with only two column ($K = 2$) PNNs and hence didn't perform any dimensionality reduction of lateral activations. As a result, we also didn't use the matrix V_i. Defining the adapter layer for our model was not that straight forward since our network uses BiLSTMs which are sequential models. The adapter layer needs to be applied to every step of the sequence independently. That is, each element of the sequence is input to the adapter layer. However, the elements of the sequence need not be processed sequentially. In other words, we group all the elements of a sequence to create a new batch and then apply our adapter layer on this batch. We have used the TimeDistributed layer of tensorflow for implementing the adapter layer.

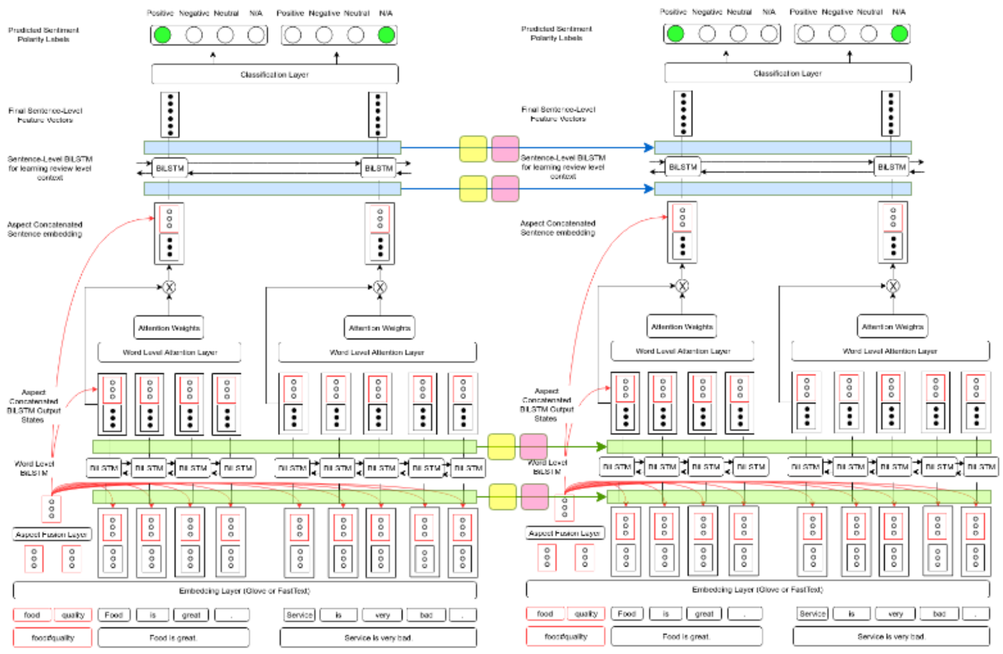

Positive Negative Neutral N/A Positive Negative Neutral N/A
Predicted Sentiment Polarity Labels
Classification Layer
Final Sentence-Level Feature Vectors
Sentence-Level BiLSTM for learning review level context
BiLSTM
Aspect Concatenated Sentence embedding
Attention Weights
Word Level Attention Layer
Aspect Concatenated BiLSTM Output States
Word Level BiLSTM
BiLSTM
Aspect Fusion Layer
Embedding Layer (Glove or FastText)
food quality Food is great .
Service is very bad .
food#quality
Food is great.
Service is very bad.

Figure 4.6.: This is an image of our proposed architecture showing lateral connections for progressive neural network. Long green rectangles represent the lateral connections from input and output layers of word-level BiLSTM and long blue rectangles represent lateral connections from input and output layers of sentence-level BiLSTM. Small yellow square represents learnable scalar and the small pink square represents the adapter layer.

We have four lateral connections in our progressive neural network architecture. These are shown in figure 4.6. The first two connections (shown in long green rectangles) are from the input and output layers of word-level BiLSTM and the next two connections (shown in long blue rectangles) are from the input and output layers of sentence-level BiLSTM. Small yellow square represents learnable scalar and the small pink square represents the adapter layer.

4.3.2. Single Unified Network (SUN)

Our proposed architecture can also be trained on a multi-domain dataset. Unlike multi-headed architectures and some other approaches where the last layer is domain dependent, our model has the advantage of being domain independent when it comes to the final classification layer. For this independence, we first transform the raw data from multiple domains using the input transformation explained in section 4.3. After this, we simply merge the transformed data points from each domain to create a single unified dataset. Since our model can be trained on this single unified dataset so, we refer it as the Single Unified Network (SUN).

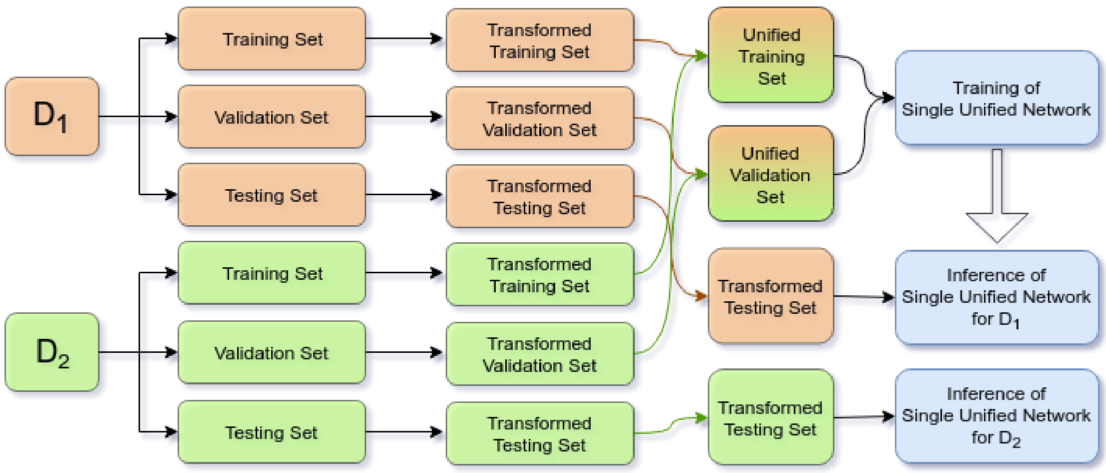


Figure 4.7.: This image shows the steps performed for combining data from two domains D_1 and D_2. The combined training and validations sets were used for training the Single Unified Network. Inference was performed separately for each domain on the uncombined test sets.

Figure 4.7 explains the steps we took for mixing data from multiple domains. To illustrate, let's assume we have training data D_1 and D_2 from two domains. Each dataset initially has N_{d1} and N_{d2} number of data points (reviews) respectively. Number and type of aspects in each domain vary a lot so, lets assume D_1 has A_{d1} aspects and D_2 has A_{d2} aspects. After applying the input transformation (from section 4.3) on both the domains independently the transformed dataset $\widetilde{D_1}$ (of domain D_1) will have N_{d1}xA_{d1} data points and $\widetilde{D_2}$ (of domain D_2) will have N_{d2}xA_{d2} data points. We perform these steps for training, validation, and test sets of both the domains separately. Now, in order to create a unified training set $\widetilde{U}^{train}$, we simply combine the transformed training datasets $\widetilde{D_1}^{train}$ and $\widetilde{D_2}^{train}$ of both the domains. Similarly, we also combine the transformed validation datasets $\widetilde{D_1}^{val}$ and $\widetilde{D_2}^{val}$ for generating a unified validation set $\widetilde{U}^{val}$. We do not combine the test sets $\widetilde{D_1}^{test}$ and $\widetilde{D_2}^{test}$ because the inference is performed separately for each domain.

Inference results of Single Unified Network showed some improvements in comparison to the baseline results of the models trained separately for each domain. We did not study the reasons for these improvements but our hypothesis is that this could be due to the increased (because of the unification of multiple domains) amount of training data. Another possible reason could be that the mixture of data acted as noisy data for every domain which in turn served as a regulariser and hence simply helped the model to generalize well across all the domains.

5. Data

In this chapter, we briefly discuss various datasets that we have worked with. We will present their statistics and will also discuss a bit about how we annotated Organic food data.

	SemEval 2016 Restaurant	**SemEval 2016 Laptop**	**GermEval 2017 Deutsche Bahn**	**Organic Fine grained**	**Organic Coarse grained**
Review count	350	450	19432	962	962
Review count with IT	4200	52200	388640	106782	17316
Review count with IT & DA	5654	55136	129299*	-	-
Sentence count	2000	2500	-	6973	6973
Entities count	6	22	-	9	3
Attributes count	5	9	-	14	6
All Aspects count	30	198	20	126	18
Valid Aspects count	12	116	20	111	18

IT=Input Transformation, DA=Data Augmentation with similar words, *= Here, we performed negative sampling of the majority class

Table 5.1.: This table reports the statics of training set of various datasets.

Table 5.1 reports the statistics of training set and the table 5.2 shows the output classes distribution for various datasets that we have used. Data augmentation was performed by generating similar sentences (using similar words technique) for SemEval-2016 Restaurant and Laptops dataset and for GermEval-2017 we simply performed negative sampling of majority class. Data augmentation was only performed for data points belonging to the neutral class. The first part of the table 5.2 is for Restaurant data, the second part is for Laptop data, the third part for GermEval data, fourth part for Organic fine-grained data and finally the last part is for organic coarse-grained data. We did not perform any data augmentation for both fine-grained and coarse-grained organic data. This is because with respect to the three output classes (*Positive*, *Negative*, *Neutral*) the organic dataset seemed balanced and applying data augmentation on *Neutral* class would have disrupted this balance.

	Positive	**Negative**	**Neutral**	**N/A**
Before Data Augmentation (Restaurant)	6.83%	3.08%	0.41%	89.66%
After Data Augmentation (Restaurant)	6.44%	2.91%	6.04%	84.58%
Before Data Augmentation (Laptops)	0.56%	0.37%	0.06%	98.99%
After Data Augmentation (Laptops)	0.55%	0.37%	1.06%	98.00%
Before Data Augmentation (DB)	0.54%	2.30%	2.64%	94.49%
After Data Augmentation* (DB)	1.61%	6.81%	7.82%	83.74%
Before Data Augmentation (Organic fine)	0.18%	0.16%	0.21%	99.43%
Before Data Augmentation (Organic coarse)	1.10%	1.01%	1.34%	96.53%

* = Here, we performed negative sampling of the majority class

Table 5.2.: This table shows the output class distribution of datasets for both before and after applying the data augmentation. Data augmentation was performed by generating similar sentences for SemEval-2016 Restaurant and Laptops dataset and for GermEval-2017 we did a negative sampling of majority class. Data augmentation was only performed for the neutral class. The first part of the table is for Restaurant data, the second part is for Laptop data, the third part for GermEval data, fourth part for Organic fine-grained data and finally the last part is for organic coarse-grained data.

5.1. SemEval-2016 Restaurant and Laptops

In this thesis, we are mainly focusing on sentence-level aspect-based sentiment analysis (ABSA). This is subtask 1 of task 5 of SemEval-2016 dataset (Pontiki et al., 2016b). According to SemEval the subtask 1 has two parts, the first part focuses on aspect (tuple of aspect and attribute) categorization and the goal of the second part is to detect sentiment polarity towards a given aspect. We have slightly deviated from this formulation and have formulated ABSA as a joint task of detecting both aspect and its corresponding sentiment simultaneously. We have used the restaurant and laptops datasets of SemEval-2016 for our experiments. The annotation details about these datasets are presented in a paper by Pontiki et al. (2016a). The statistics of these datasets both prior to applying data augmentation and input transformation techniques and after applying them are discussed in tables 5.1 and 5.2. For a list of valid aspects see appendix A.1.

5.2. GermEval-2017 Deutsche Bahn

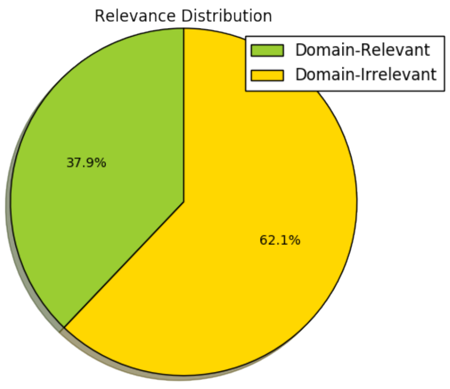
This dataset consists of customer feedback on social media about Deutsche Bahn in the German language. For this dataset, we focus on subtask 3 which is about Aspect-level Polarity detection. The aim of this subtask is to identify all contained aspects and their associated polarity (Wojatzki et al., 2017a). In this dataset, aspects and associated sentiment are annotated at the comment/review level instead of the sentence level. The statistics of this dataset both prior to applying data augmentation and input transformation techniques and after applying them are discussed in tables 5.1 and 5.2. For a list of valid aspects see appendix A.1.

5.3. Organic

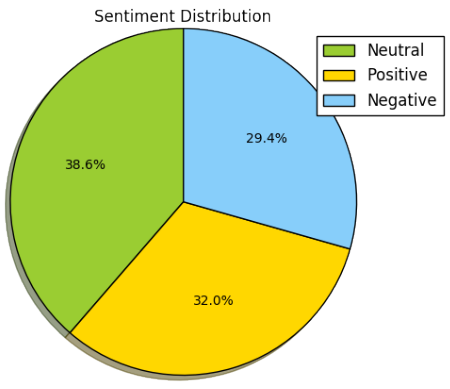
The data for organic food was collected by crawling social media platforms such as Facebook, Twitter, Quora, and other discussion forums. First of all, the raw data was pre-processed and comments which were not in the domain were filtered out. A comment is considered in the domain if at least one of its sentences is relevant to the domain.

5.3.1. Annotation and Split Strategy

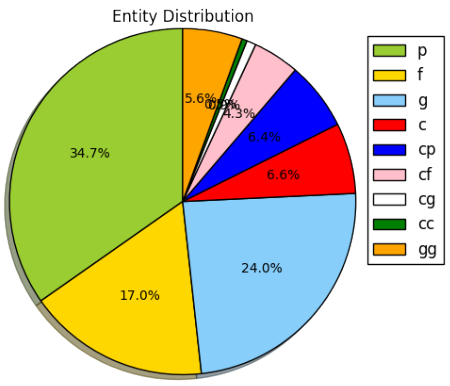
The entire data was divided into multiple groups of thousand sentences each and none of these groups shared any sentences between them. Each group was annotated by a different annotator and annotations by every annotator were considered the gold label. In other words, no voting algorithm was used to decide the final gold label for a sentence because none of the sentences were shared among the annotators.



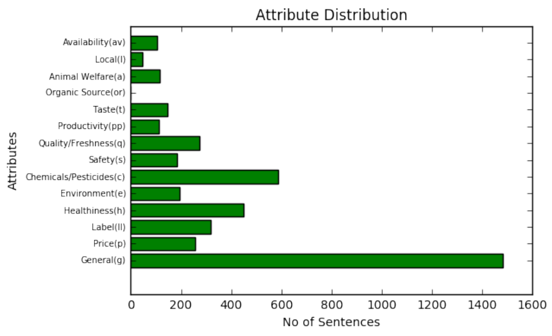
(a)



(b)



(c)



(d)

Figure 5.1.: This figure shows the distribution of organic training data over relevance, sentiment, entity and attribute. (a) Relevance distribution (b) Sentiment distribution (c) Entity distribution (d) Attribute distribution

After annotation, we split the data into three different sets - training set (with 80% comments), validation or dev set (with 10% comments), and test set (with 10% comments). The goal here was to divide the data in such a way that the data distribution over fields such as relevance, sentiment, entity, and attributes remains almost similar within the different sets. We followed the following steps to achieve this.

1. Firstly, we verified all the annotated files for trivial annotation mistakes and then combined all of them to create a single annotation file.

2. We then created a map of aspect (which is a tuple of entity and attribute) versus comments. All those comments which had at least one sentence that was labeled with a particular aspect a got mapped to this aspect a.

3. Next, for every aspect in the map, we distributed the comments mapped to that aspect in these three sets (training, validation, test) according to 80-10-10 ratio. If an aspect had fewer than 3 comments mapped to it then we simply added all its comments to the training set. This was done to ensure that the test or validation set don't get stuck with aspects which are not present in the training set. If an aspect had more than 3 comments and it was still not possible to perfectly distribute its comments among the three sets, then, with a probability of 0.5 we decided to either include one comment in the validation set and one in the test and the remaining in the training set. This was done to make sure that validation and test set don't remain underrepresented. Another important thing to note was that a comment could be mapped to multiple aspects and so we needed to make sure that a comment didn't get assigned to multiple sets.

Figure 5.1(a),5.1(b),5.1(c),5.1(d) shows the distribution of organic data over relevance, sentiment, entity and attribute. These distributions are for the training set. However, validation and test set and the overall set also follow a similar distribution (see appendix A.3).

6. Experiments and Results

This chapter describes the implementation details and discusses experimental results. For each experiment, we will first describe the experimental setup and then present the results followed by a short discussion about the obtained results. For some experiments, we will also present some interesting ideas that we couldn't explore to due time constraints.

6.1. Training and Evaluation

Majority of the implementation was done by us while some small modules were borrowed from a repository on Hierarchical Attention Models by Ezhov (2017). The architecture was implemented using TensorFlow 1.4.0. Details of the training environment are presented in table 6.1.

Specifications	**Machine 1**	**Machine 2**
GPU	4GB GeForce GTX 970	12GB GeForce GTX TITAN X
Operating System	Ubuntu 16.04.5 LTS	Ubuntu 18.04.1 LTS
Python Version	Python 3.5.2	Python 3.5.6
RAM	128GB	64GB
CPU Count	24	12
Nvidia Driver	Driver Version: 375.66	Driver Version: 390.87
CUDA Version	release 8.0, V8.0.61	release 8.0, V8.0.44
CPU	Intel(R) Xeon(R) CPU E5-2643 v3 @ 3.40GHz	Intel(R) Core(TM) i7-3930K CPU @ 3.20GHz

Table 6.1.: Specifications of the training environments.

We have used Cross Entropy loss along with Adam optimizer (Kingma and Ba, 2014) for training our networks. In order to compare the performance of our networks, we have used F1 scores (both micro and macro averages) as the evaluation metric. Due to our problem statement and due to the design of our proposed architecture, we had to use the multi-label version of micro F1 score. For this multi-label implementation, we borrowed logic for multi-label implementation from the GermEval-2017 evaluation script (originally implemented in Java). During our implementation, we found some bugs in this script. We have also opened an issue regarding this on their GitHub page but, didn't receive any acknowledgment till the time of writing this report. So, in order to be consistent in our comparison with other results, we decided to use the existing logic. Details about why this evaluation metric was chosen and what exactly it measures are described in section 3.4.

To being with, we first performed a hyperparameter search over these three model parameters - learning rate, batch size, and dropout keep probability. For this, we conducted a grid search using a library called tune. Values of some other hyperparameters such as the maximum value of gradient-norm, word embedding size, number of LSTM units were either fixed by trial and error or were directly borrowed from the literature. Hyperparameter tuning was only performed over SemEval-2016 Restaurant dataset. All experiments with the rest of the datasets also used these tuned values of the hyperparameters. No pre-trained word embeddings were used during hyperparameter tuning. Instead, the embedding matrix was randomly initialized and was optimized during the training process. A list of hyperparameters along with their tuned values is presented in table 6.2.

Hyperparameter	**Value**
Learning Rate	0.001
Batch Size	64
Word Embedding Size	300
Aspect Embedding Size	300
Number of LSTM units for $BiLSTM_{word}$	64
Number of $BiLSTM_{word}$ layers/stacks	1
Number of LSTM units for $BiLSTM_{sentence}$	64
Number of $BiLSTM_{sentence}$ layers/stacks	1
Attention output size	300
Dropout keep probability	0.8

While using ELMo embedding we have used word and aspect embedding of size 1000 and due to the lack of GPU resources we had to reduce the batch size to 32.

Table 6.2.: This table shows a list of hyperparameters values, these were fixed either by using a grid search or trail & error or were directly picked from the literature.

The first baseline model on which hyperparameter tuning was performed had about 2 million trainable parameters (with a restricted vocabulary size of about only 5000 words). A model with such a high complexity will probably overfit when it is trained with very little data. In fact, this exact thing happened when we trained our model with SemEval-2016 Restaurant dataset. Figure 6.1 shows the training vs validation accuracy graph of this model. It clearly indicates that the network is overfitting. One of the ways of tackling this overfitting problem is to obtain more training data. Since this was not possible so, we had to rely on other methods. We tried using an L2 regularizer for trainable weights but apparently, that didn't work. In fact, with L2 regularization the performance became worse. A possible reason for this performance drop could be that it is difficult to use L1/L2 regularization with RNNs because it hinders an LSTM cell's ability to learn and retain information over the time steps (Pascanu et al., 2012). Next, we tried using dropouts as a regularizer. It helped a bit but the model was still overfitting. Finally, we had to resort to an early stopping technique and dropouts for tackling the overfitting problem. For inference, we essentially used a checkpoint that performed best on the validation test during training time.

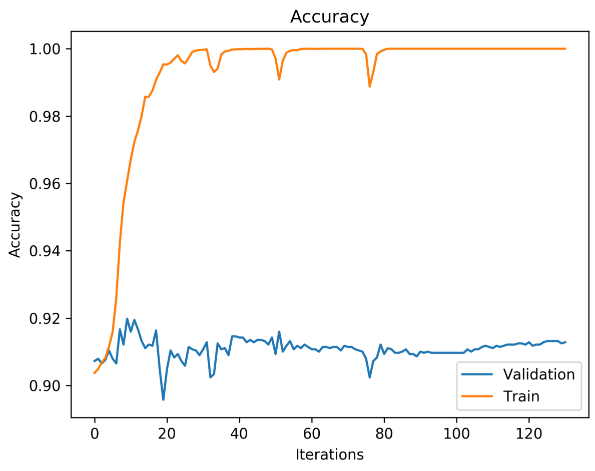


Figure 6.1.: Training vs validation accuracy graph of the baseline model before employing the dropouts and early stopping.

Formulation of aspect-based sentiment analysis as a joint task of simultaneous detection of both aspect and sentiment is less common in the literature. Instead, most methods use a pipeline based approach where they first detect an aspect and then detect the associated sentiment assuming that the given aspect already exists. We couldn't find any baseline results on SemEval-2016 dataset from the models that use joint formulation. As a result, we decided to use our current network (with no pre-trained word embeddings, no data augmentations) as our baseline for future experiments. Additionally, the best results from our experiments on SemEval-2016 dataset can also be considered as the state of the art on this particular dataset for a joint end-to-end approach. For GermEval-2017 we did find an approach that uses joint formulation so, for this dataset, we considered the results from this approach as the baseline and compared our results against it.

6.2. Word Embeddings

In this work, we majorly experimented with two word-embedding techniques, namely GloVe and FastText. For theoretical details about these and some other embeddings read section 3.3.

GloVe

In the case of GloVe embedding, we have used the Common Crawl pre-trained version with a vocabulary of 840 billion tokens having an embedding dimension of 300. Since it was difficult for us to work with such a huge embedding matrix and also, our datasets were really small having a vocabulary of only a few thousand words so, we decided to use only a subset of the entire embedding matrix. We created our own smaller embedding matrix using only the vocabulary from the training set. All the words from the validation and test set that were not present in the aforementioned vocabulary were treated as unknown or out of vocabulary (OOV) words. For managing these unknown words in the embedding matrix we just mapped them to a fixed random 300-dimensional vector. We know that a neural network can only take real-valued inputs and hence it's not possible for the neural network to consume words in their original form. So words in the form of words-ids are injected into the model as input and the network then maps these word-ids to their corresponding word vectors using the embedding matrix.

Results and Discussions

Table 6.3 presents the results of our model for the task of aspect-based sentiment analysis on SemEval-2016 Restaurant data. We see a considerable improvement in the overall performance in comparison to the baseline model which tried to learn its own word embeddings from scratch. These results were expected even though the word embeddings were pre-trained on a completely unrelated dataset. The reason for this is

Variant	**Macro F1 Score**					**Micro F1 Score**
	average	*positive*	*negative*	*neutral*	*n/a*	
Baseline	0.47	0.551	0.378	0.000	0.962	0.337
PT-G	0.57	0.688	0.418	0.205	0.969	0.454
PT-F	0.52	0.580	0.323	0.222	0.951	0.354
PT-E	0.54	0.686	0.478	0.000	0.970	0.455
FT-G	0.53	0.632	0.307	0.188	0.967	0.379
FT-F	0.48	0.566	0.329	0.047	0.957	0.331

PT-G=Pre-Trained-GloVe, PT-F=Pre-Trained-fastText, PT-E=Pre-Trained-ELMo, FT-G=Fine-Tuned-GloVe, FT-F=Fine-Tuned-fastText

Table 6.3.: This table shows the performance of different variants of our model on SemEval-2016 Restaurant dataset for ABSA. In the Baseline model word embedding matrix was randomly inilialized. Pre-trained version used GloVe/-fastText embeddings and Fine-tuned version inilialized word embedding matrix with pre-trained GloVe/fastText embeddings and then fine tuned them during the training process. All the three variants had the same hyperparameters as mentioned in table 6.2. Here we also show the individual macro average scores for the four output classes $(positive, negative, neutral, n/a)$.

that word embeddings to some extent are able to transfer the semantic and syntactic knowledge learned from a source dataset to another target dataset. So essentially, using word embeddings can also be considered as a form of transfer learning. For the fine-tuning experiment, we initialized our word embedding matrix with the pre-trained vectors and then fine-tuned them during the training process. The results of this experiment were not that encouraging. The performance became worse than the pre-trained version but it was still better than the randomly initialized baseline version. This is probably because fine-tuning disrupted the existing learned structure of the pre-trained embeddings and in the absence of enough training data, we were not able to acquire any extra knowledge from the new dataset. To summarize, in the absence of a lot of training data simply using the pre-trained embeddings gave better results than fine-tuning or learning the embeddings from scratch.

fastText

We also experimented with the fastText embedding. Going by the empirical results of a lot of different papers, fastText has improved the performance of various NLP tasks. Another benefit of fastText is that it can manage out of vocabulary (OOV) words out of the box. Unlike GloVe where we explicitly had to assign some random vector to unknown words, fastText always returns a vector for any given known or unknown word. This is because in fastText an embedding for a word is generated by combining the embeddings of its n-grams. Now, it happens so that an n-gram of one word is also the n-gram of many other words, so given an unknown word it is highly possible that some of its n-grams are present in many known words. Also, there's a small implementation detail regarding OOV words that should be taken care of while using fastText. Always use the pre-trained fastText model instead of using the pre-trained word-embedding matrix file. This is because the model will always return a word vector for a word by making use of its n-grams. On the contrary, pre-trained word-embedding matrix file which is just a map between a word and its embedding will return nothing in case of OOV words.

For our experiments with SemEval-2016 restaurant and laptops dataset, we have used fastText model trained with subword information on Common Crawl that generates 300-dimensional word embeddings. Since fastText can generate embeddings for OOV words so it wouldn't hurt to create a word-embedding matrix file of a combined vocabulary from training, validation, and test set in advance. Using this embedding matrix file and the same set of hyperparameters we trained our model for aspect-based sentiment analysis task on the restaurant dataset.

Results and Discussions

The results of this experiment are shared in table 6.3. Contrary to our expectations the performance of the model with fastText embeddings is a bit worse than with the GloVe embeddings. To make the matter more confusing, our experiments with the laptops dataset showed better results with fastText than with GloVe. It seems there is no definitive answer to the question that which embedding among GloVe and fastText is better. An important take away from this experiment is that try different embeddings on your dataset and choose the one that works best for your dataset.

ELMo

ELMo is currently considered the state of the art word embedding technique. It is a contextualized word embedding. For more details about ELMo embeddings read section 3.3. ELMo embeddings were computed using allennlp library. It takes a list of sequence in the form of a file (where each line represents a sequence and each token within a sequence should be space separated) as input and outputs a map of sequence-id and a list of word vectors for each word in that sequence. For instance, if the input sequence has N tokens and the model computes an embedding of size D

then the output shape for this sequence would be (N, D). The final word vectors are 1024-dimensional and are an average of four 1024-dimensional word vectors that come out of four different layers of the ELMo model. These word vectors are computed based on the context of the current sequence and therefore a particular word when used in multiple contexts in different sequences will have multiple word representations. As a result, it is not that straight forward to generate a simple word-embedding file that maps each word to its unique word vector (because a word might not even have a unique word representation). Due to this reason, we had to slightly change the input pipeline of our model. Now, instead of inputting the word-ids and then mapping them to word vectors during the training process, we directly input the 1024-dimensional word vectors. Due to the large embedding size of 1024, we didn't have enough GPU memory for storing a batch of size 64 so, for using ELMo we had to reduce our batch size from 64 to 32.

Results and Discussions

The results are shown in table 6.3. They look good and are very comparable with the results of GloVe embeddings. Due to the limitation of resources (while training our model with ELMo embeddings) and also due to its similar performance in comparison to GloVe, we decide to experiment with ELMo for just one dataset i.e semEval-2016 restaurant.

Figure 6.2 shows the performance of our model with different word embeddings on aspect detection task. The relative size of bubbles denotes the amount of data we had for that particular aspect. From the figure, we can observe that GloVe performed better for most of the aspects. In fact, it performed relatively better than other embeddings especially for the aspects which had few data points. Please note that for computing these results we did not train our model explicitly for aspect detection task. Instead, it was trained for the joint task of aspect-based sentiment analysis. Aspect detection scores were computed using the predictions of the joint model in the following manner. As already stated our model tries to predict one of these four classes $(positive, negative, neutral, n/a)$ as output given an aspect and a sentence as input. So, if the given aspect is present in the given sentence then it tries to predict the correct sentiment for it, otherwise, it tries to predict n/a. Now out of these four possible output classes, if the model predicts a sentiment polarity, we assume that it has detected the given aspect and if it predicts n/a we assume that it didn't detect the given aspect.

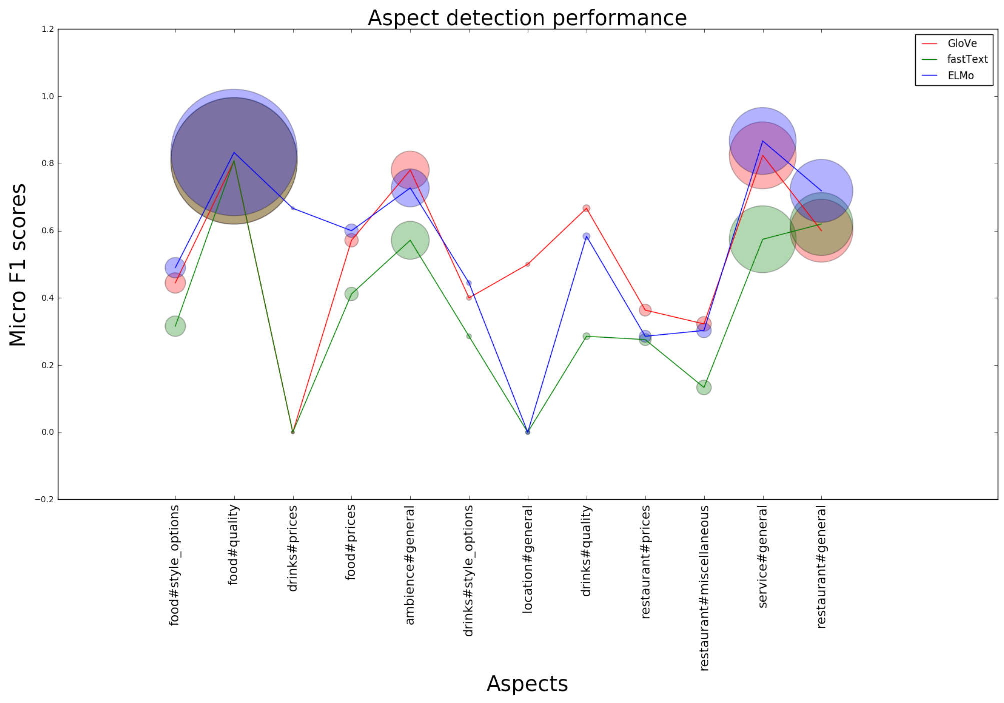


Figure 6.2.: This figure shows the performance of our model for aspect detection. The relative size of the bubble denote the number of data points we had for a particular aspect. Red is for GloVe, green for fastText and blue for ELMo.

6.3. Combating Data Imbalance

In this section, we will discuss the two techniques that we have used for combating the data imbalance problem and how much did they help in improving the overall performance of the model. We saw in the last section that pre-trained word embeddings gave better results than fine-tuned or randomly initialized word embeddings. In this section, we will build upon the previous knowledge and will try to improve the model performance by using data augmentation and class weighted loss along with the GloVe and fastText pre-trained embeddings.

6.3.1. With the help of class weighted loss

In this experiment, we use a weighted loss. With this loss, the penalization is more if the minority class is incorrectly classified. This encourages the model to learn to be extra cautious while classifying data points belonging to minority classes. The weights are computed using the inverse statistics of the class distribution. As a result minority class gets a higher weight than the majority class and consequently a misclassification for a minority class will cost more. We have used two different techniques for computing the weights. One is the static approach where we use the entire dataset and other is the dynamic approach where weights are dynamically calculated using the statistics of every batch.

Results and Discussions

The results of both these approaches were almost similar. We chose to use the static approach for the rest of our experiments because it is computationally more friendly. Table 6.4 shows the performance scores of our model after applying the static class weighted loss (CWL). We can see that there is some definite improvement in the overall performance. By looking at the micro F1 scores we can see that the overall performance gain for the model using fastText embeddings was more compared to the model using GloVe embeddings. Moreover, with respect to both the embeddings, model's performance for minority class showed some improvement and that too without affecting the majority class performance a lot.

6.3.2. With the help of data augmentation

For augmenting the data we use an approach where we replace a word either with its synonym or with a similar word. Details about how we decide which words to replace and how to generate synonym or similar words are described in section 4.2. Some sample augmentations generated by both the synonym and similar word approach can be seen in the appendix section A.2. As the augmentations generated using the synonym approach looked rather poor so, we decided to adhere to the similar word approach. Some possible explanations about why synonym approach didn't produce good augmentations are discussed in section 4.2. An important thing to note is that for generating similar words we have used GloVe embeddings because with fastText embeddings a lot of the times generated similar words were just the n-grams of the original word.

Variant	**Macro F1 Score**					**Micro F1 Score**
	average	*positive*	*negative*	*neutral*	*n/a*	
PT-G	0.57	0.688	0.418	0.205	0.969	0.454
PT-G + CWL	0.62	0.667	0.428	0.363	0.967	0.456
PT-G + CWL + DA	0.64	0.648	0.488	0.315	0.966	0.460
PT-F	0.52	0.580	0.323	0.222	0.951	0.354
PT-F + CWL	0.53	0.577	0.464	0.095	0.949	0.389
PT-F + CWL + DA	0.53	0.603	0.425	0.105	0.960	0.394

PT-G=Pre-Trained-GloVe, CWL=Class Weighted Loss, DA=Data augmentation, PT-F=Pre-Trained-fastText

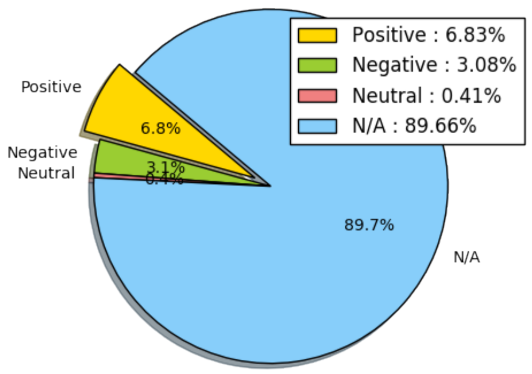
Table 6.4.: This table shows the performance of different variants of our model on SemEval-2016 Restaurant dataset for ABSA. Pre-trained version used GloVe/-fastText embeddings. We have used static class weighted loss. For data augmentation we generated similar sentences using similar words approach. All the variants had the same hyperparameters as mentioned in table 6.2. Here we also show the individual macro average scores for the possible four classes $(positive, negative, neutral, n/a)$.

Similar words don't necessarily imply words with similar meaning. Two words are said to be similar if they are very close in the embedding space. Sometimes it might happen that words which are similar according to the embedding space can be opposite in meaning. Because of this, we have used this approach to only augment *neutral* sentences. More about this is discussed in section 4.2.

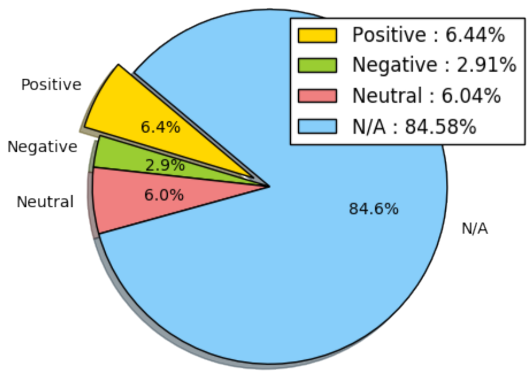
Results and Discussions

Figure 6.3(a) shows the distribution of SemEval-2016 restaurant data before applying the data augmentation. In this, we can see that *neutral* class is highly imbalanced consisting of only 0.41% of the total data. After applying the augmentation it got improved a bit to about 6.04% as shown in figure 6.3(b). We got these results by limiting the number of similar words for a given word to three. That is, for every replaceable word in a sentence we generated a maximum of three similar words and used these for generating the paraphrases. Using more words led to the generation of a lot more paraphrases which in turn made the data imbalanced with respect to *positive* and *negative* classes so, we just used the top three similar words for generating augmented sentences.

Table 6.4 shows the performance score of our model (PT-F + CWL + DA) which



(a)



(b)

Figure 6.3.: This figure shows data distribution of SemEval-2016 Restaurant dataset over the four possible output classes. Please note that both these distributions are computed after applying the input transformation of section 4.3. (a) Data distribution before applying similar sentence augmentation. (b) Data distribution after applying similar sentence augmentation.

was trained on the augmented dataset. It also used the class weighted loss. We can observe some improvement in the overall performance after training the model with the augmented dataset. The performance gain is significant in case of fastText embedding when compared to a model that was trained without class-weighted-loss and without augmentation (PT-F) than to a model that was trained with class-weighted-loss (PT-F + CWL).

Table 6.5 shows the performance of different variants of our model on the aspect detection task. The variants were not explicitly trained for aspect detection rather they were trained for ABSA and aspect detection results were computed using the process mentioned in section 6.2. We can observe from the table that GloVe embedding generally performed better on this dataset. Also, the variant which was trained on augmented data along with class weighted loss performed the best across both the embeddings. Another thing to note here is that all the model variants generally performed poorly for aspects which had very few data points such as *drink-prices, location-general*. But the variants with class weighted loss or/and data augmentation performed relatively better than the baseline variants (PT-G, PT-F) for other aspects. A thing that can be investigated into more detail is that why certain aspects such as *drink-prices, location-general* had zero scores for one embedding technique and non zero scores for the other embedding technique.

Aspects (Data Count)	PT-G	PT-G & CWL	PT-G & CWL & DA	PT-F	PT-F & CWL	PT-F & CWL & DA
food-style-options (137)	0.586	0.481	0.487	0.315	0.313	0.359
food-quality (849)	0.769	0.804	0.823	0.808	0.786	0.802
drinks-prices (20)	0.000	0.000	0.000	0.000	0.181	0.571
food-prices (90)	0.482	0.499	0.631	0.411	0.466	0.266
ambience-general (255)	0.799	0.819	0.738	0.571	0.531	0.645
drinks-style-options (32)	0.266	0.470	0.666	0.285	0.545	0.499
location-general (28)	0.285	0.444	0.444	0.000	0.000	0.000
drinks-quality (47)	0.521	0.444	0.399	0.285	0.545	0.235
restaurant-prices (80)	0.380	0.399	0.615	0.275	0.307	0.249
restaurant-miscellaneous (98)	0.344	0.230	0.153	0.133	0.226	0.099
service-general (449)	0.797	0.799	0.796	0.574	0.774	0.698
restaurant-general (442)	0.492	0.607	0.630	0.619	0.629	0.688
Micro F1 Average	0.495	0.516	0.535	0.407	0.449	0.458

PT-G=Pre-Trained-GloVe, CWL=Class Weighted Loss, DA=Data augmentation, PT-F=Pre-Trained-fastText

Table 6.5.: This table shows the performance of different variants of our model on SemEval-2016 Restaurant dataset for the aspect detection task. All the variants had the same hyperparameters as mentioned in table 6.2. All the scores are micro F1 scores.

6.4. Various Aspect Fusion Techniques

An aspect is represented by a set of two or more aspect words. For example, *service-general* have two words and *food-style-options* consists of three words. Now, to create a fixed sized aspect vector (required by our model) we fuse the embeddings of these individual aspect words together. We experimented with three approaches to fuse aspect words. In one of the approaches, we simply took the mean of the aspect word embeddings. In another approach, we selected the maximum of all the aspect words vectors across all the dimensions, i.e max pooling of aspect word embeddings. For the third approach, we concatenated all the aspect word embeddings together and then projected the concatenated vector to a smaller fixed-sized vector using a fully connected layer. For this last approach, we assumed an upper limit to the number of aspect words in an aspect (three for our experiments) and we zero padded those aspects which had lesser number of aspect words.

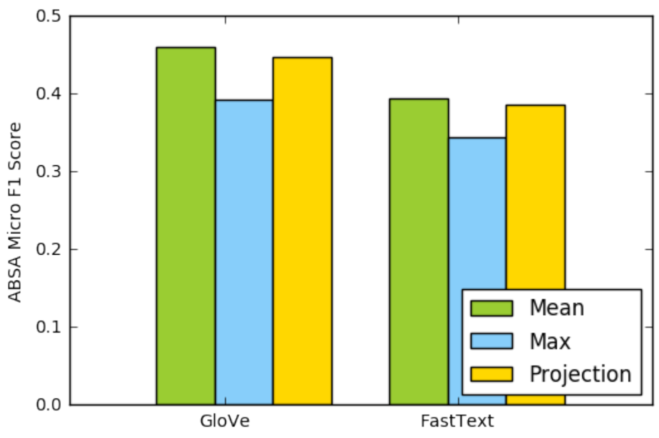


Figure 6.4.: The figure shows the results of our model trained for ABSA. The variants used different aspect fusion approaches with different word embeddings. Variants were trained with the SemEval-2016 Restaurant augmented dataset using weighted class loss.

Results and Discussions

Figure 6.4 shows the results of our model trained for ABSA. The variants used different aspect fusion approaches with different word embeddings. For these experiments, we have trained the variants with the augmented dataset and we have also used class weighted loss. We can see from the figure that the mean and the projection fusion approaches performed better than max fusion for both the GloVe and fastText embeddings. Using the projection approach increases the model complexity and with the limited data that we have it also raises the chance of overfitting, so we decided to stick with the mean fusion approach for the rest of our experiments.

An interesting extension for future work could be to use a list of seed words for defining an aspect. One way of generating these seed words could be, to make use of synonyms of the original aspect words. For instance, an original aspect consisting of two aspect words *[food, quality]* can have seed words such as *[meal, lunch, dinner]* for the aspect word *food* and similarly we can have seed words such as *[standard, condition, taste]* for the aspect word *quality*. An alternative approach could be that, with the help of domain experts, we can manually define these seed words. Additionally, we should also make sure that seed words for one aspect do not overlap with the seed words of another aspect. Now, a fusion of all these seed words could be used to generate a fixed sized aspect vector.

6.5. Experiments with SemEval-2016 Laptops data

This is another dataset from SemEval-2016 which contains reviews about laptops. The size of the dataset in terms of the number of reviews is almost similar to that of the semEval-2016 restaurant dataset but, the number of possible valid aspects is ten times more than we had for restaurant dataset. The list of possible valid aspects for the laptop dataset is listed in the appendix A.1. Out of the 198 total possible aspects we decided to work with 116 and excluded the rest (such as *display-price, cpu-portability*) which didn't make any sense.

Results and Discussions

We did not perform any hyperparameter search for this dataset but instead adopted the existing hyperparameters from table 6.2. The results from two of the best variants, one with GloVe embedding and the other with fastText embedding are shown in table 6.6. Contrary to the results with restaurant dataset where the best performance was achieved with GloVe embedding, in this case, the best performance is obtained by a variant using fastText embedding. From this observation, we can conclude that it is difficult to assess which embedding will be best for all the datasets. One reasonable way is to empirically investigate the performance of various embeddings on your dataset and then choose the best one.

Variant	Macro F1 Score					Micro F1 Score
	average	*positive*	*negative*	*neutral*	*n/a*	
PT-G + CWL + DA	0.42	0.393	0.208	0.054	0.993	0.220
PT-F + CWL + DA	0.44	0.426	0.255	0.068	0.993	0.246

PT-G=Pre-Trained-GloVe, CWL=Class Weighted Loss, DA=Data augmentation, PT-F=Pre-Trained-fastText

Table 6.6.: This table shows the performance of our model on SemEval-2016 Laptops dataset for ABSA. We have used pre-trained GloVe/fastText embeddings. For class weighted loss we used the static version. For data augmentation we generated similar sentences using similar words. All the variants had the same hyperparameters as mentioned in table 6.2. Here we also show the individual macro average scores for the possible four classes $(positive, negative, neutral, n/a)$.

During this experiment, we also tried to investigate our model's performance on aspect detection task on semEval-2016 laptops dataset. Like before, the variants were not explicitly trained for aspect detection rather they were trained for ABSA and the aspect detection results were computed using the process mentioned in section 6.2. The results of this experiment are presented in table 6.7. We can see that out of the 116 possible valid aspects 35 had no data points in the training set. In fact, more than half (79 to be precise) of the valid aspects only had 10 or fewer sentences in the training set. With such a limited amount of training data, it was definitely going to be really difficult for the model to detect these aspects and the same is also indicated by the aspect detection results of these aspects. Only 13 out of 116 aspects had 50 or more training data points and our model performed decently on these aspects. 9 out of these 13 aspects were about the entity *laptop* and its various attributes. This means that most of the other 21 entities such as *keyboards, mouse, support* had less than 50 data points. All this brought us to the perspective that how much imbalanced the laptop dataset is with respect to the aspects.

Count of data points (C)	Number of aspects with C data points	PT-G & CWL & DA (Avg of per aspect micro F1 Score)	PT-F & CWL & DA (Avg of per aspect micro F1 Score)
$C == 0$	35	0.000	0.000
$0 < C \leq 10$	44	0.023	0.054
$10 < C \leq 20$	14	0.171	0.144
$20 < C \leq 50$	10	0.288	0.334
$50 < C \leq 100$	5	0.459	0.382
$C > 100$	8	0.405	0.459
Overall Micro F1 Score	116	0.220	0.246

PT-G=Pre-Trained-GloVe, CWL=Class Weighted Loss, DA=Data augmentation, PT-F=Pre-Trained-fastText

Table 6.7.: This table shows the performance of our model on SemEval-2016 Laptops dataset for aspect detection task. In this we report average of micro f1 scores of aspects belonging to a group. The aspects were grouped based on the number of training data points C they had. All the variants had the same hyperparameters as mentioned in table 6.2.

A side effect of our proposed input transformation became more evident while experimenting with the SemEval-2016 laptops dataset. As explained in section 4.1.1, according to this transformation each comment is duplicated for every possible valid aspect. For instance, in this dataset, a comment will be duplicated 116 times. The labels for every sentence in the comments are *(positive, negative, neutral)* if a sentence is talking about the concerned aspect, otherwise, it's *n/a*. Most of the times a comment only discusses 5 or 6 aspects and hence all the other (110 or 111 in this case) duplicated comments of this original comment will have all its sentences labeled as *n/a*. This leads to a highly imbalanced dataset with respect to the output classes *(positive, negative, neutral, n/a)* which is skewed towards the *n/a* class. Even with our data augmentation technique of generating similar sentences we were not able to balance it to a satisfactory level. For this dataset, about 98% sentences were labeled as *n/a*, 1% were labeled as *neutral*, 0.6% as *positive* and about 0.4% as *negative*. For more details about the distribution of the datasets read chapter 5.

6.6. Experiments with GermEval-2017 Deutsche Bahn data

Schmitt et al. (2018) uses a joint end-to-end approach for detecting aspects and its associated sentiments. The experiments in this paper are performed on GermEval-2017 Deutsche Bahn data. For the sake of completeness and in order to compare the performance of our model with an established baseline and the current state of the art results, we have also trained our proposed model on the GermEval-2017 Deutsche Bahn dataset.

For our experiments with GermEval-2017 Deutsche Bahn dataset, we kept the proposed data split and performed no pre-processing and did not even perform the proposed input transformation (see section 4.1.1). The only form of pre-processing we did was that we randomly selected one of the sentiment polarity for some training instances where an aspect was assigned two different sentiment polarities. For combating with the data imbalance we trained our model with a class weighted loss. Hyperparameter tuning was performed on the validation/dev set and the selected hyperparameter values were the same as presented in table 6.2 except for the attention output size, for which a value of 500 was selected. Going by the views of Schmitt et al. (2018) that fastText is more effective for morphologically rich languages such as German because of its ability to make use of n-grams, we only experimented with fastText. We have used two variants of fastText embeddings, one off the shelf version which was pre-trained on Wikipedia and the other which we trained ourselves using the GermEval-2017 training set data and a corpus of approximately 113K tweets mentioning at least one of @DB info and @DB Bahn. This is the same corpus that was used by Schmitt et al. (2018) and we obtained it from them.

Results and Discussions

Table 6.8 reports the comparison of our results against the current state of the art

Variant	**Macro F1**					**Micro F1**
	average	*positive*	*negative*	*neutral*	*n/a*	
Pipeline LSTM-F	-	-	-	-	-	0.297
End-to-end LSTM-F	-	-	-	-	-	0.315
Pipeline CNN-F	-	-	-	-	-	0.295
End-to-end CNN-F	-	-	-	-	-	0.423
E2E LSTM-F	0.55	0.164	0.432	0.596	0.975	0.383
E2E LSTM-F + NS	0.53	0.113	0.290	0.583	0.945	0.266
E2E LSTM-F + STF	0.56	0.209	0.436	0.598	0.976	0.384
MC baseline	-	-	-	-	-	0.315
GermEval baseline	-	-	-	-	-	0.322
GermEval BS	-	-	-	-	-	0.354

E2E=End to End, NS=Negative Sampling, STF=Self Trained fastText Embeddings, LSTM-F=LSTM with fastText, MC=Majority Class, BS=Best Submission

Table 6.8.: This table reports the comparison of our results against the current state of the art and also against the best submission of the GermEval-2017 competition for ABSA. The results are reported using micro F1 scores. We also report individual macro average scores for the four classes output $(positive, negative, neutral, n/a)$. The results in the middle are from our model.

and also against the best submission of the GermEval-2017 competition. It shows micro F1 scores for ABSA (i.e both aspect category and aspect polarity classification) as computed by the GermEval evaluation script. The results from the top part of the table were borrowed from Schmitt et al. (2018) and those from the bottom part were borrowed from Wojatzki et al. (2017b). We can see from the table that the performance of our best model is better than the baseline and the best submission of the competition. Our results are close to the current state of the art but unfortunately, we couldn't improve upon it. Schmitt et al. (2018) experimented with both LSTMs and CNNs and they got the state of the art results using CNNs. The performance of our model which uses LSTMs is significantly better in comparison to their LSTM variant. As an extension of the current work, in the future, we can also experiment with CNNs instead of LSTMs. For one of our variants, we tried to reduce the class imbalance by the down/negative sampling of the majority class samples, unfortunately as the results indicate this decreased the overall performance of the model. In another variant, we used the concatenation of pre-trained (300-dimensional) fastText embedding with

Variant	Micro F1
End-to-end LSTM-F	0.442
End-to-end CNN-F	0.523
E2E LSTM-F	0.492
E2E LSTM-F + NS	0.347
E2E LSTM-F + STF	0.508
MC baseline	0.442
GermEval baseline	0.481
GermEval BS	0.482

E2E=End to End, NS=Negative Sampling, STF=Self Trained fastText Embeddings, LSTM-F=LSTM with fastText, MC=Majority Class, BS=Best Submission

Table 6.9.: This table reports the comparison of our results against the current state of the art and also against the best submission of the GermEval-2017 competition for aspect detection task. The results are reported using micro F1 scores. The results in the middle are from our model. As before, the results from the top part of the table were borrowed from Schmitt et al. (2018) and those from the bottom part were borrowed from Wojatzki et al. (2017b).

our self-trained (100-dimensional) fastText embedding. This slightly improved the performance of the model. We concluded that the principal advantage of fastText was its ability to model subword information and the small domain-specific corpus which was used to self-train the fastText embedding had a very limited positive impact.

Table 6.9 reports the micro F1 scores for aspect detection task. These results were also computed by the GermEval evaluation script. From the results, we can see that the end to end LSTM variant of Schmitt et al. (2018) had worse performance than the baselines and the best submission, whereas, our end to end LSTM variants were able to surpass the scores of baselines and best submission of GermEval-2017. Also, the best result of our model is close to the state of the art result of the end to end CNN model.

Figure 6.5 shows the performance of our model for aspect detection on GermEval dataset. The relative size of the bubbles denotes the number of data points we had for that particular aspect. In the figure, we can see that except for a few aspects such as *Design*, *Reisen_mit_Kindern*, *Image*, *QRCode*, rest all the aspects had fairly consistent and decent f1 scores, irrespective of the amount of training data we had for those aspects.

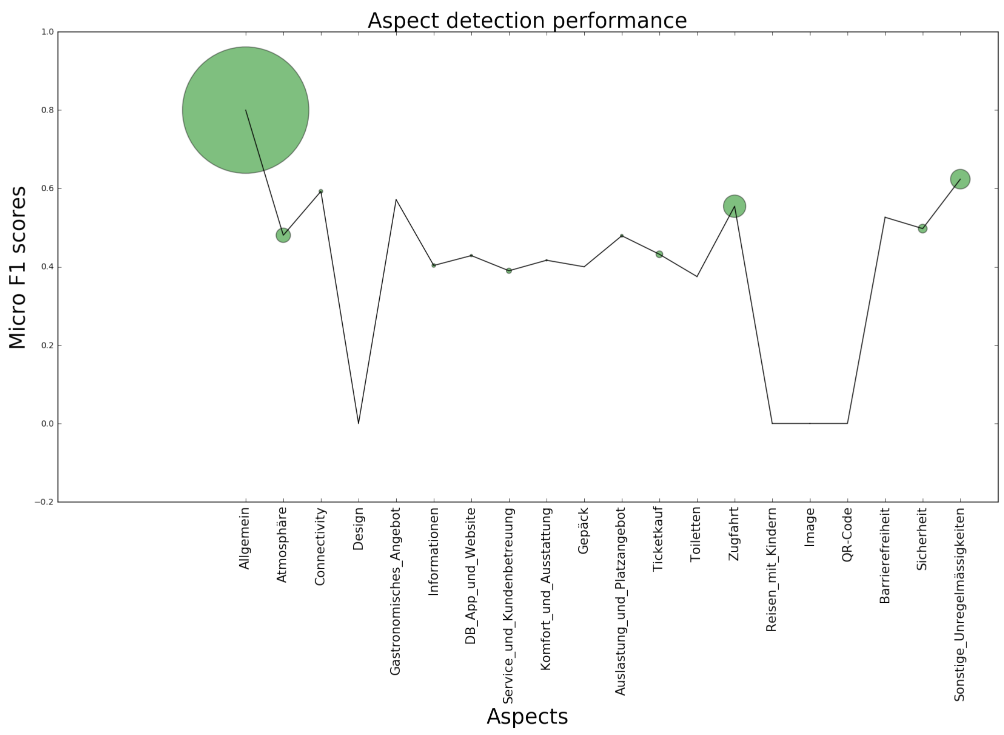


Figure 6.5.: This figure shows the performance of our model for aspect detection on GermEval dataset. The relative size of the bubbles denote the number of data points we had for that particular aspect.

6.7. Experiments with Organic food data

A set of experiments were also conducted on organic data. This data was annotated by us and was split into three sets namely training set, validation set, test set. For more details about the data annotation process and the logic behind the data split read section 5.3. Similar to SemEval-2016 laptops dataset organic data too had a lot of aspects. There were 111 valid possible aspects that were present in the training set (see appendix A.1 for a list of valid aspects). In order to compare the influence of the number of aspects on the model performance, we generated two versions of the organic dataset, one was the fine-grained version with 111 aspects and other was the coarse-grained version with 18 aspects. The coarse-grained version was produced by substituting many of the closely correlated fine-grained aspects with a more general aspect.

We did not perform any hyperparameter tuning of our model for this data, instead, we simply used the hyperparameters from table 6.2 that were tuned on the semEval-2016 restaurant dataset. Additionally, we only experimented with the fastText embeddings (because of the time limitations for the project). Some of the comments in the dataset had more than 50 sentences and due to the limited resources we couldn't fit them on GPU memory so, as a workaround, we trimmed such comments and only considered the first 50 sentences.

Variant	**Macro F1**					**Micro F1**
	average	*positive*	*negative*	*neutral*	*n/a*	
PT-F + CWL + Fine	0.36	0.082	0.063	0.097	0.981	0.059
PT-F + CWL + Coarse	0.39	0.197	0.122	0.231	0.966	0.129

PT-F=Pre-trained fastText embedding, CWL=Class Weighted Loss

Table 6.10.: This table reports the results of our model against the fine-grained and coarse-grained variant of organic data for the task of ABSA. The results are reported using micro F1 scores. The model was trained using the hyperparameters mentioned in the table 6.2

Results and Discussions

Table 6.10 reports the micro f1 scores that we obtained for the two dataset variants. Our model performed very poorly, particularly with the fine-grained dataset. Interestingly, it gave significantly better numbers when it was trained with the same hyperparameters and the same word embedding on semEval-2016 laptops dataset, even though the laptop dataset was half in size compared to the organic data. We believe an in-depth analysis is required to better understand why this happened but due to the time constraints, we propose to take this up as a future task. Nevertheless, the following are some of our hypothesis for the poor performance of our model on organic data.

- **Unclean data** - In laptops data, only about 18% sentences were out of domain whereas in the organic data 42% sentences are irrelevant. Some irrelevant sentences from both the domains are shown in table 6.11. Here we can observe that irrelevant sentences for organic data contain a lot of terms such as *soil, tea, protein* which are related to attributes present in the dataset such as *environment, products, nutrition*. As a result, it becomes much harder for the network to learn which sentences are relevant and which are irrelevant when the majority of the terms used in both the type of sentences are similar. On the contrary, in the laptops dataset, most of the irrelevant sentences are just gibberish and don't talk about

SemEval-2016 Laptops	Organic
Well, they don't care a bunch.	But you are free to make your own choices.
I have owned it only two months!	None of these can be achieved by compost tea or soil because you will never know exactly what is in either of those.
Being a PC user my whole life....	We can't currently produce meat (in much quantity, anyway) that contains a good protein source without using actual meats to start with.
Yes, I have it on the highest available setting.	Pests that might eat crops are in turn eaten by other animals who may also be beneficial to the biosphere.

Table 6.11.: The table shows some irrelevant sentences from SemEval-2016 laptops and organic dataset.

any of the possible aspects.

- **Ambiguous attributes** - Some of the possible attributes are too similar and it becomes extremely difficult even for humans to correctly choose between them. For instance, consider this sentence from the organic dataset - *"Organic growing forbids using any refined minerals, so you are eating potentially toxic impurities along with the minerals."*. What would you annotate it with *organic-products-healthiness* or *organic-farming-safety* or *organic-products-safety*?

- **Conflicting annotations** - Generally, for deciding the gold labels for a data point, its annotations from multiple annotators are taken into consideration. But in our case due to resource limitations, we couldn't afford to get a data point annotated by multiple annotators. So gold label for a particular data point is simply decided by a single annotator. Given the fact that the aspect annotation is very subjective and ambiguous, this might lead to problems where sentences having similar terms or meaning are annotated with different or conflicting labels. For instance, consider this sentence - *"I believe they allow organic meats to have been fed GMO feedstock, however."*. The annotator labeled it with *organic-products-healthiness*. For this sentence, in our opinion, the label *gmo-genetic-engineering-general* would have been more appropriate. To check the quality of annotations and the agreement among annotators we can perform an inter-rater reliability test by asking them to annotate a bunch of shared sentences.

Count of data points (C)	Number of aspects with C data points	PT-F & CWL & Fine (Avg of per aspect micro F1 Score)
$0 < C \leq 10$	57	0.005
$10 < C \leq 20$	16	0.064
$20 < C \leq 50$	16	0.068
$50 < C \leq 100$	10	0.188
$C > 100$	12	0.278
Overall Micro F1 Score	111	0.059

CWL=Class Weighted Loss, PT-F=Pre-Trained-fastText

Table 6.12.: This table shows the performance of our model on fine-grained Organic dataset for aspect detection task. In this we present average of micro f1 scores of aspects belonging to a group. The aspects were grouped based on the number of training data points C they had.

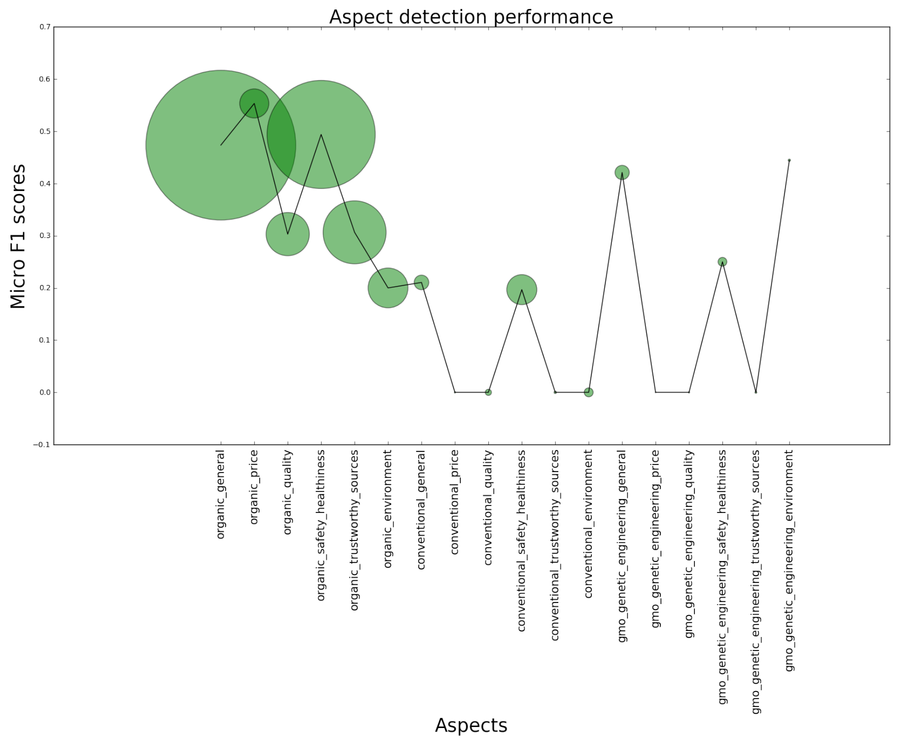


Figure 6.6.: This figure shows the performance of our model for aspect detection on coarse-grained Organic dataset. The relative size of the bubbles denote the number of data points we had for that particular aspect. The overall micro f1 score for all the aspects for coarse-grained dataset was 0.237.

Table 6.12 reports the results of our model for aspect detection task on the fine-grained organic dataset. Like before, the model was not explicitly trained for aspect detection rather it was trained for ABSA and aspect detection result was computed using the process mentioned in section 6.2. As expected majority of the aspects had 10 or fewer data points and hence it was really difficult for the model to learn to classify them.

Figure 6.6 shows the performance of our model for aspect detection on the coarse-grained organic dataset. The relative size of the bubbles denotes the number of data points we had for that particular aspect. In this figure, we can see that for some of the aspects the model performed decently and for some of the aspects, the model performed very poorly with zero micro f1 scores.

6.8. Progressive Neural Network (PNN)

Progressive Neural Network (PNN) is a transfer learning approach that improves upon some of the limitations of fine-tuning. Unlike fine-tuning PNNs are immune to catastrophic forgetting. For more details about how the structure of a progressive neural network looks like and how did we implement it with our model for aspect-based sentiment analysis read section 4.3.1.

Currently, our implementation only supports a two-column progressive neural network. We refer the network in the first column as the previous network from which we want to transfer the knowledge and the network in the second column as the current network whose performance we want to improve with the help of transfer learning. We conducted three experiments with our PNN architecture and compared its results with the results from fine-tuning and the baseline models. First of all, we trained two networks independently on semEval-2016 restaurant and laptops dataset. Results from these models formed our baselines. Now, for the first experiment, we used the model trained on the laptop data as the previous network of our PNN architecture and trained the PNN with restaurant data. The intention here was to transfer the knowledge from a previously trained network (on laptop domain) to a current network that is being trained for a different domain (restaurant). Similarly, for the second experiment, we used the model trained on the restaurant data as the previous network and trained the PNN with the laptop data. For the third experiment, we wanted to transfer the knowledge from both the laptops and the restaurant domain to the organic domain. Since our current PNN implementation only supports one previous network, so for this, we first trained a Single Unified Network using restaurant and laptop data and used this as the previous network to train a PNN on the organic data. Lastly, for the fine-tuned version of a model on restaurant data we just fine-tuned the baseline network trained on the laptop data and for the fine-tuned version on laptop data we fine-tuned the baseline network that was trained on the restaurant data.

Variant	Restaurant PT-F CWL DA	Laptop PT-F CWL DA	Organic (Coarse) PT-F CWL
Baseline	0.394	0.246	0.129
Fine-tuned	0.324	0.229	0.062
PNN	0.471	0.276	0.170

PT-F=Pre-trained fastText embedding, CWL=Class Weighted Loss, DA=Data Augmentation, PNN=Progressive Neural Network

Table 6.13.: This table reports the results of progressive neural network approach for the task of ABSA. The results are reported using micro F1 scores. The model was trained with the hyperparameters mentioned in the table 6.2. All the experiments were performed using fastText.

Results and Discussions

Before we talk about the results lets first talk about how did we create data for these experiments. We first created a combined vocabulary from all the three domains. Using the word-ids of this combined vocabulary we generated three different datasets for the three individual domains. Additionally, a combined dataset (of laptop and restaurant) for the third experiment was also formed.

Table 6.13 reports the results of the above three experiments that we had performed with the progressive neural network architecture. For the first experiment PNNs performed exceptionally good. They outperformed the baseline on restaurant data with an impressive improvement of almost 8 points. The results from the second experiment were also positive and we observed an improvement of 3 points over the baseline (trained on laptop data). Similarly, for the experiment with organic data, we also saw an improvement of a couple of points. Contrary to our expectations the results from fine-tuned networks were worse than the baseline networks for all the three data domains. The progressive neural network is able to solve ABSA task all the domains without incurring any catastrophic forgetting for the previous domains but, as a drawback, we still need to decide which column of the PNN to use for which domain. Another downside of PNN is that as the number of tasks/domains increases the model complexity also increases and along with this training time also increases.

6.9. Single Unified Network (SUN)

As already stated in section 4.3.2 our proposed architecture has an inherent capability to get trained on a common dataset consisting of data points from multiple domains. We refer a network trained on a combined dataset as a Single Unified Network (SUN). For understanding the advantages of this architecture and the details about how a unified dataset is generated read section 4.3.2.

Variant	Baseline	Fine-tuned	PNN	SUN (without organic)	SUN (with organic)
Restaurant PT-F CWL DA	0.394	0.324	0.471	0.399	0.353
Laptop PT-F CWL DA	0.246	0.229	0.276	0.260	0.224
Organic (Coarse) PT-F CWL	0.129	0.062	0.170	-	0.145

PT-F=Pre-trained fastText embedding, CWL=Class Weighted Loss, DA=Data Augmentation, PNN=Progressive Neural Network, SUN=Single Unified Network

Table 6.14.: This table reports the results of single unified network approach for the task of ABSA. The results are reported using micro F1 scores. The models was trained with the hyperparameters mentioned in the table 6.2.

Results and Discussions

For this experiment, we created two unified training datasets one with the organic data and one without the organic data. We independently trained two instances of our model on these two datasets. Table 6.14 reports the inference results of Single Unified Network (trained on a combined dataset) for different domains. Here we can observe that, when the SUN was trained without organic data, it showed some improvements in comparison to the baseline results of the models trained separately for each domain. We did not study the reasons for these improvements in detail but our hypothesis is that this could be due to an increase in the training data because of the unification of multiple domains. Another possible reason could be that the combined data was noisy compared to the individual datasets and this, in turn, served as a regulariser and hence simply helped the model to generalize well across all the domains. Interestingly, the same network when it was trained with a unified dataset consisting of organic data, it didn't perform that well. Its results were worse than the baseline for the restaurant and the laptop domain but were better than the baseline for the organic domain.

7. Limitations and Future Work

In this work, we have presented a new approach to the task of aspect-based sentiment analysis. Due to the scarcity of time, we couldn't examine all the possibilities and hence left some areas unexplored. For the same reason, we were also unable to closely investigate some of the results. In this chapter, we will present some extensions and limitations of our current approach.

In all our variants we have used LSTMs because of our belief that RNNs are better than CNNs at modeling sequential data, especially when it comes to long term relations. There are many approaches in the literature which have empirically claimed to have achieved better results with CNNs than with RNNs or its variants. Going by the same spirit we feel it would be totally worth it to study how well a CNN variant of our approach performs.

Intuitively, our model is trying to concentrate on those words in the text that are correlated or define a given aspect. In order to do this, our model requires a fixed sized aspect vector. An interesting idea for generating a more effective aspect vector is by fusing various seed words that can define a given aspect. This idea has been discussed in more detail in section 6.4.

Another simple extension could be to use a smoothened version of attention as proposed by Shen and Lee (2016). Moreover, as we didn't spend much time exploring various loss functions, we would also suggest having a look in this direction.

Furthermore, an in-depth analysis of various results especially an analysis of why results on organic data were so different than the results on laptops data, is highly encouraged.

One drawback of the proposed input transformation is that it inherently induces class imbalance and this becomes apparent with a dataset which has a lot of possible aspects. We faced this problem when experimenting with the laptops dataset. A small analysis of this drawback is presented in section 6.5.

8. Conclusion

The objective of the thesis was to propose a neural network architecture for aspect-based sentiment analysis. Additionally, the goal was also to make use of as much contextual information as possible. Also, since the available datasets were very small so it was desired to have a network which would have the capability to learn from a limited amount of training data.

Our proposed novel architecture is capable of jointly detecting aspects and corresponding sentiments. With the use of BiLSTMs and Attention mechanism, our network extracts contextual information from the entire review/comment while predicting aspect and sentiment polarities for each sentence of the review. Another important contribution of this work was the input transformation technique. This technique provided some interesting benefits to the proposed model. It helped in addressing three important problems. Firstly, with the help of this input transformation, we were able to limit the number of output classes to 4. In other words, the number of output classes no longer depended on the number of aspects. Secondly, by fixing the number of output classes, our model became capable of getting trained on datasets of different domains without requiring any architectural modifications. With this added capability we were able to transfer the knowledge from a source domain to a target domain using the approach of Progressive Neural Network (PNN) and Single Unified Network (SUN). To address the problem of limited and imbalanced data we implemented a data augmentation library (called simsent) that generates paraphrases or similar sentences by replacing some of its words with synonyms or similar words.

We performed experiments with four datasets (SemEval-2016 Restaurant, SemEval-2016 Laptops, Organic food, GermEval-2017 Deutsche Bahn) belonging to four different domains. Out of these four datasets, baseline and state of the art results were only reported for GermEval-2017 in the literature for a joint end-to-end setting. The results of our experiments with the GermEval-2017 dataset are very close to the current state of the art (Schmitt et al., 2018). In fact, when compared to its LSTM variant our model performed better for both ABSA and aspect detection tasks. We observed that augmenting data by replacing words helped in reducing the class imbalance and thus also helped in improving the model performance. Furthermore, our experiments with transfer learning approach such as Progressive Neural Networks (PNN) and Single Unified Network (SUN) improved the overall model performance by 1 to 2 F1 points on an average. We also observed that fastText is not always necessarily better than GloVe. Our experiments with restaurant data showed better results with GloVe whereas our experiments with laptops data showed better results with fastText.

To our best knowledge, we are the first ones to provide practical results on SemEval-2016 restaurant and laptop dataset for aspect-based sentiment analysis using a joint approach. These results can now act as a new baseline for future research in this area.

List of Figures

List of Tables

A. Appendix

A.1. Valid aspects for different domains

A.1.1. SemEval-2016 Restaurants

restaurant-general	food-style_options	service-general
restaurant-prices	drinks-prices	location-general
restaurant-miscellaneous	drinks-quality	
food-prices	drinks-style_options	
food-quality	ambience-general	

A.1.2. SemEval-2016 Laptops

laptop-general	cpu-price
laptop-price	cpu-quality
laptop-quality	cpu-operation_performance
laptop-operation_performance	cpu-design_features
laptop-usability	cpu-miscellaneous
laptop-design_features	motherboard-general
laptop-portability	motherboard-price
laptop-connectivity	motherboard-quality
laptop-miscellaneous	motherboard-design_features
display-general	motherboard-miscellaneous
display-quality	hard_disc-general
display-operation_performance	hard_disc-price
display-usability	hard_disc-quality
display-design_features	hard_disc-operation_performance
display-portability	hard_disc-design_features
display-miscellaneous	hard_disc-miscellaneous
cpu-general	memory-general

memory-price
memory-design_features
memory-miscellaneous
battery-general
battery-quality
battery-operation_performance
battery-design_features
battery-miscellaneous
power_supply-general
power_supply-price
power_supply-quality
power_supply-operation_performance
power_supply-design_features
power_supply-miscellaneous
keyboard-general
keyboard-quality
keyboard-operation_performance
keyboard-usability
keyboard-design_features
keyboard-miscellaneous
mouse-general
mouse-quality
mouse-operation_performance
mouse-usability
mouse-design_features
mouse-miscellaneous
fans_cooling-general
fans_cooling-quality
fans_cooling-operation_performance
fans_cooling-design_features
fans_cooling-miscellaneous
optical_drives-general
optical_drives-quality
optical_drives-operation_performance
optical_drives-design_features
optical_drives-miscellaneous
ports-general
ports-quality
ports-operation_performance
ports-design_features
ports-miscellaneous
graphics-general
graphics-quality
graphics-design_features
graphics-miscellaneous
multimedia_devices-general
multimedia_devices-quality
multimedia_devices-operation_performance
multimedia_devices-usability
multimedia_devices-design_features
multimedia_devices-miscellaneous
hardware-general
hardware-quality
hardware-operation_performance
hardware-usability
hardware-design_features
hardware-miscellaneous
os-general
os-quality
os-operation_performance
os-usability
os-design_features
os-miscellaneous
software-general
software-price
software-quality
software-operation_performance
software-usability
software-design_features
software-miscellaneous
warranty-general
warranty-price
warranty-miscellaneous
shipping-general
shipping-price
shipping-quality
shipping-miscellaneous
support-general

support-price
support-quality
support-miscellaneous
company-general

A.1.3. GermEval-2017

Allgemein
Atmosphäre
Connectivity
Design
Gastronomisches_Angebot
Informationen
DB_App_und_Website
Service_und_Kundenbetreuung
Komfort_und_Ausstattung
Gepäck
Auslastung_und_Platzangebot
Ticketkauf
Toiletten
Zugfahrt
Reisen_mit_Kindern
Image
QR-Code
Barrierefreiheit
Sicherheit
Sonstige_Unregelmässigkeiten

A.1.4. Organic fine-grained

organic-general-general
organic-general-price
organic-general-taste
organic-general-nutritional-quality-freshness-appearance
organic-general-safety
organic-general-healthiness
organic-general-chemicals-pesticides
organic-general-label
organic-general-origin-source
organic-general-local
organic-general-availability
organic-general-environment
organic-general-animal-welfare
organic-general-productivity
organic-products-general
organic-products-price
organic-products-taste
organic-products-nutritional-quality-freshness-appearance
organic-products-safety
organic-products-healthiness
organic-products-chemicals-pesticides
organic-products-label
organic-products-origin-source
organic-products-local
organic-products-availability
organic-products-environment
organic-products-animal-welfare
organic-products-productivity
organic-farming-general
organic-farming-price
organic-farming-taste
organic-farming-nutritional-quality-freshness-appearance
organic-farming-safety
organic-farming-healthiness
organic-farming-chemicals-pesticides

organic-farming-label
organic-farming-origin-source
organic-farming-local
organic-farming-availability
organic-farming-environment
organic-farming-animal-welfare
organic-farming-productivity
organic-companies-general
organic-companies-price
organic-companies-taste
organic-companies-nutritional-quality-freshness-appearance
organic-companies-safety
organic-companies-healthiness
organic-companies-chemicals-pesticides
organic-companies-label
organic-companies-origin-source
organic-companies-local
organic-companies-availability
organic-companies-environment
organic-companies-animal-welfare
organic-companies-productivity
conventional-general-general
conventional-general-price
conventional-general-nutritional-quality-freshness-appearance
conventional-general-safety
conventional-general-healthiness
conventional-general-chemicals-pesticides
conventional-general-label
conventional-general-origin-source
conventional-general-productivity
conventional-products-general
conventional-products-price
conventional-products-taste
conventional-products-nutritional-quality-freshness-appearance
conventional-products-safety
conventional-products-healthiness
conventional-products-chemicals-pesticides
conventional-products-label
conventional-products-origin-source
conventional-products-local
conventional-products-availability
conventional-products-environment
conventional-products-animal-welfare
conventional-products-productivity
conventional-farming-general
conventional-farming-price
conventional-farming-taste
conventional-farming-nutritional-quality-freshness-appearance
conventional-farming-safety
conventional-farming-healthiness
conventional-farming-chemicals-pesticides
conventional-farming-label
conventional-farming-origin-source
conventional-farming-environment
conventional-farming-animal-welfare
conventional-farming-productivity
conventional-companies-general
conventional-companies-taste
conventional-companies-safety
conventional-companies-chemicals-pesticides
conventional-companies-label
conventional-companies-availability
conventional-companies-environment
conventional-companies-animal-welfare
conventional-companies-productivity
gmo-genetic-engineering-general
gmo-genetic-engineering-price
gmo-genetic-engineering-taste
gmo-genetic-engineering-nutritional-quality-freshness-appearance
gmo-genetic-engineering-safety
gmo-genetic-engineering-healthiness
gmo-genetic-engineering-chemicals-

pesticides
gmo-genetic-engineering-label
gmo-genetic-engineering-origin-source
gmo-genetic-engineering-environment
gmo-genetic-engineering-productivity

A.1.5. Organic coarse-grained

organic-general
organic-price
organic-quality
organic-safety-healthiness
organic-trustworthy-sources
organic-environment
conventional-general
conventional-price
conventional-quality
conventional-safety-healthiness
conventional-trustworthy-sources
conventional-environment
gmo-genetic-engineering-general
gmo-genetic-engineering-price
gmo-genetic-engineering-quality
gmo-genetic-engineering-safety-healthiness
gmo-genetic-engineering-trustworthy-sources
gmo-genetic-engineering-environment

A.2. Data Augmentation

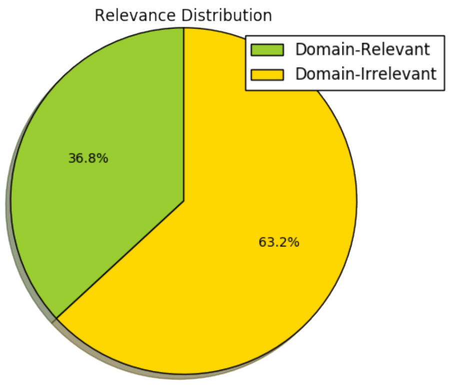
Original Text	Augmentations using synonyms
For the price, you cannot eat this well in Manhattan.	For the monetary value, you cannot eat this well in Manhattan. For the Mary Leontyne Price, you cannot eat this well in Manhattan.
The food was lousy - too sweet or too salty and the portions tiny.	The nutrient was lousy - too sweet or too salty and the portions tiny. The food was lousy - too sweet or too salty and the fortune tiny. The food was lousy - too fresh or too salty and the portions tiny. The intellectual nourishment was lousy - too sweet or too salty and the portions tiny. The food was lousy - too sweet or too salty and the parcel tiny.
Ambiance- relaxed and stylish.	Ambiance- relaxed and stylish. Ambiance- relaxed and fashionable.
I'm very happy with this machine!	I'm very happy with this auto! I'm very happy with this automobile!
the features are great, the only thing it needs is better speakers.	the features are great, the only affair it needs is better speakers. the features are great, the only thing it needs is better verbaliser. the characteristic are great, the only thing it needs is better speakers. the features are great, the only thing it needs is well speakers. the features are great, the only matter it needs is better speakers. the feature of speech are great, the only thing it needs is better speakers. the features are great, the only thing it needs is better talker. the features are great, the only thing it needs is good speakers.
Great price and computer!	Great price and calculator! Great Mary Leontyne Price and computer! Great price and information processing system! Great monetary value and computer!

Table A.1.: Some examples of the augmentations that were generated by replacing words with synonyms.

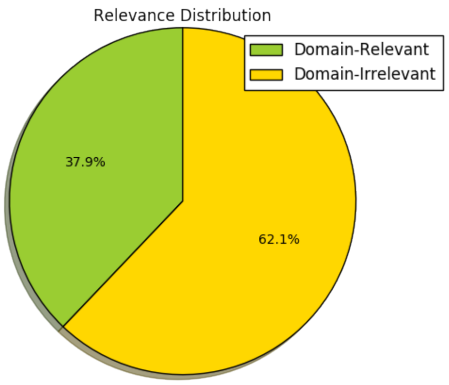
Original Text	Augmentations using similar words
For the price, you cannot eat this well in Manhattan.	For the discount, you cannot eat this well in Manhattan. For the buy, you cannot eat this well in Manhattan.
The food was lousy - too sweet or too salty and the portions tiny.	The food was crappy - too sweet or too salty and the portions tiny. The food was shitty - too sweet or too salty and the portions tiny. The food was lousy - too sweet or too salty and the portions small. The food was lousy - too sweet or too salty and the portions large. The snacks was lousy - too sweet or too salty and the portions tiny. The cooking was lousy - too sweet or too salty and the portions tiny.
Ambiance- relaxed and stylish.	Ambiance- relaxed and fashionable. Ambiance- quiet and stylish. Ambiance- relaxed and sleek. Ambiance- easy-going and stylish. Ambiance- relaxed and trendy.
I'm very happy with this machine!	I'm very always with this machine! I'm very happy with this equipment! I'm very thankful with this machine! I'm very glad with this machine! I'm very happy with this Machine!
the features are great, the only thing it needs is better speakers.	the features are great, the only thing it needs is well speakers. the features are excellent, the only thing it needs is better speakers. the features are great, the only thing it needs is better subwoofers. the features are terrific, the only thing it needs is better speakers. the features are great, the only thought it needs is better speakers.
Great price and computer!	Great discount and computer! Amazing price and computer! Great pricing and computer! Wonderful price and computer! Great price and laptop!

Table A.2.: Some examples of the augmentations that were generated by replacing words with similar words.

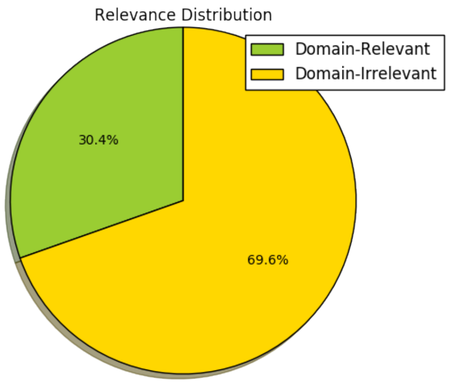
A.3. Organic data distribution



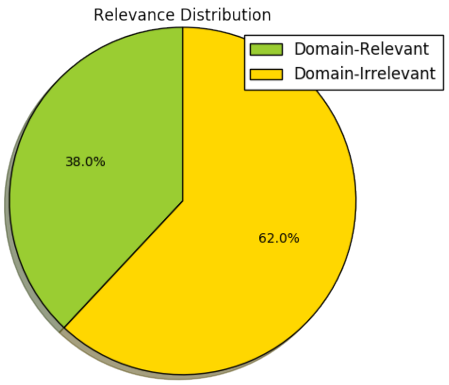
(a)



(b)

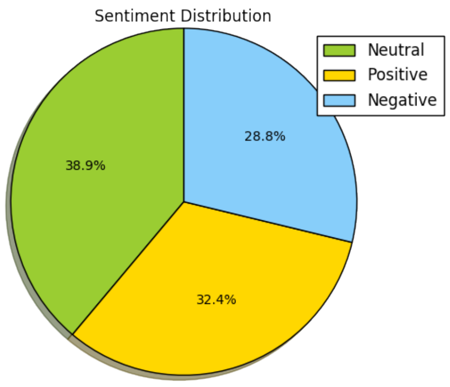


(c)

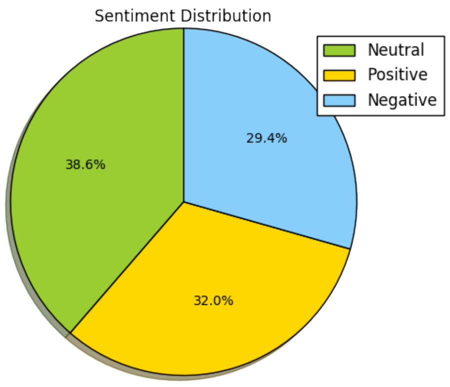


(d)

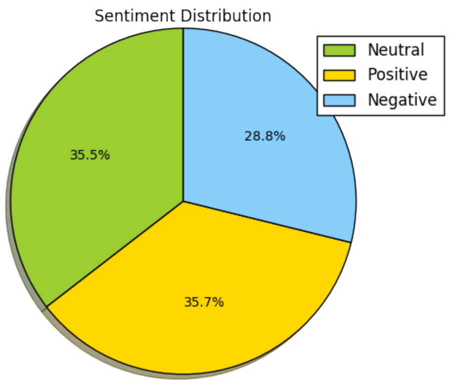
Figure A.1.: (a) Entire data (b) Training set (c) Validation set (d) Test set



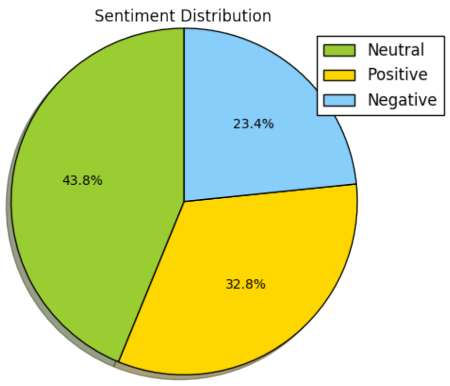
(a)



(b)

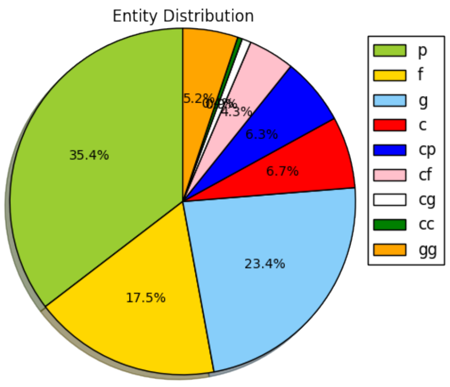


(c)

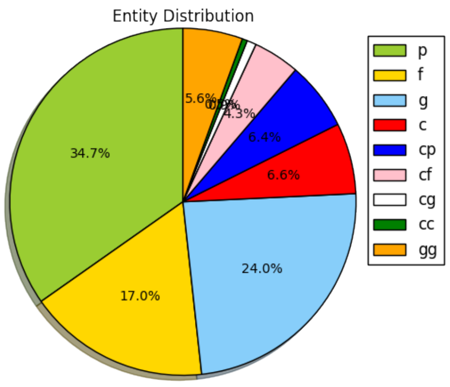


(d)

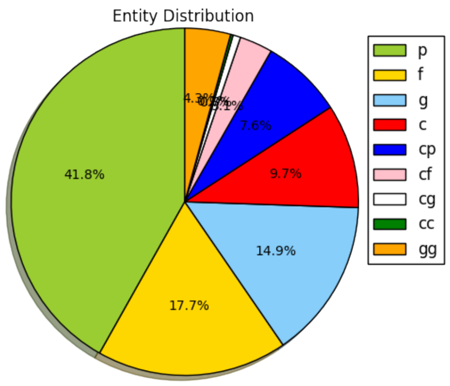
Figure A.2.: (a) Entire data (b) Training set (c) Validation set (d) Test set



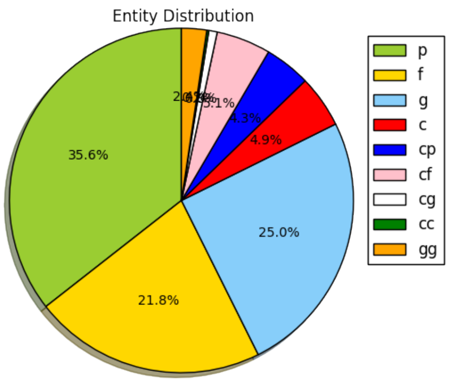
(a)



(b)



(c)



(d)

Figure A.3.: (a) Entire data (b) Training set (c) Validation set (d) Test set

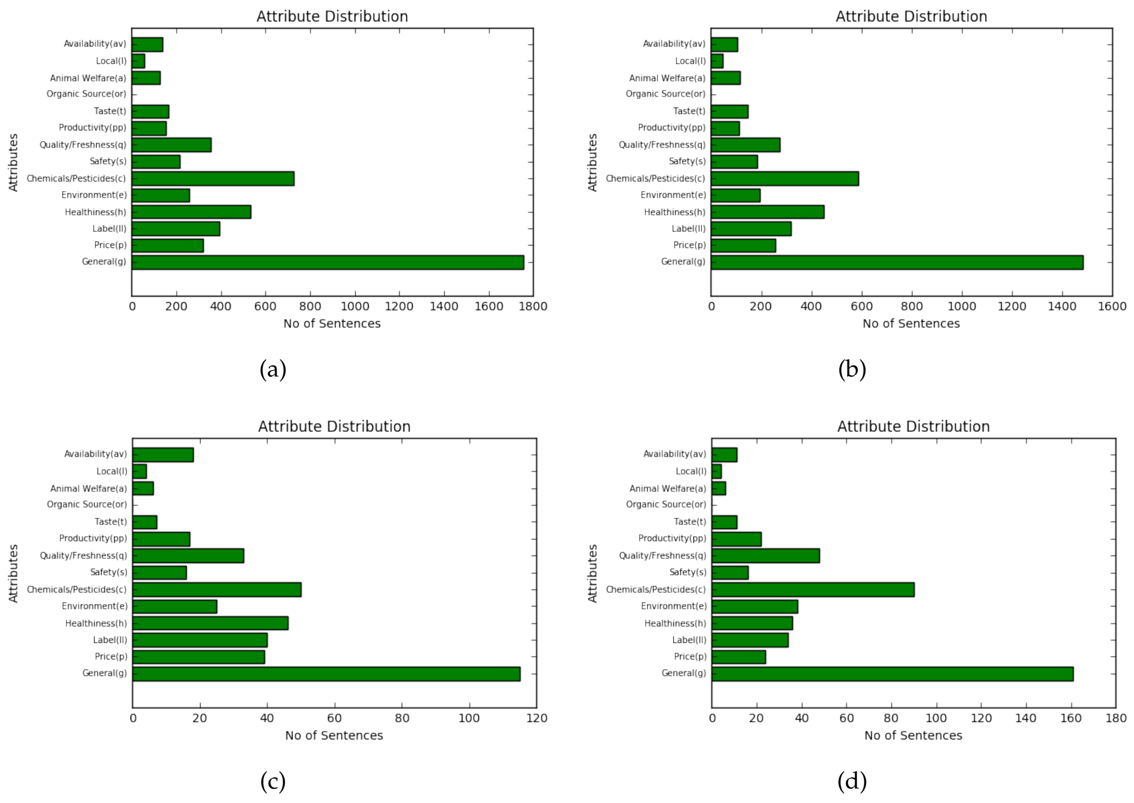


Figure A.4.: (a) Entire data (b) Training set (c) Validation set (d) Test set

Bibliography

Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. 2011. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Language in Social Media (LSM 2011)*, pages 30–38. Association for Computational Linguistics.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Georgios Balikas, Simon Moura, and Massih-Reza Amini. 2017. Multitask learning for fine-grained twitter sentiment analysis. *CoRR*, abs/1707.03569.

Leonard E. Baum and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state markov chains. *Ann. Math. Statist.*, 37(6):1554–1563.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *CoRR*, abs/1607.04606.

Danny Britz. 2015. Recurrent neural networks tutorial, part 1 – introduction to rnns.

Philip Bump. 2018. Everything you need to know about the cambridge analytica-facebook debacle.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. 2018. Universal sentence encoder. *CoRR*, abs/1803.11175.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Colah. 2015. Understanding lstm networks.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364.

Thomas G. Dietterich. 2002. Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30, Berlin, Heidelberg. Springer Berlin Heidelberg.

Murat Dundar, Balaji Krishnapuram, Jinbo Bi, and R. Bharat Rao. 2007. Learning classifiers when the training data is not iid. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 756–761, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Matvey Ezhov. 2017. Document classification with hierarchical attention networks in tensorflow.

Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202.

Rohith Gandhi. 2018. Introduction to sequence models-rnn, bidirectional rnn, lstm, gru.

Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. 2000. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. 2002. Learning Precise Timing with LSTM Recurrent Networks. 3:115–143.

Zoubin Ghahramani. 2002. Hidden markov models. chapter An Introduction to Hidden Markov Models and Bayesian Networks, pages 9–42. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 513–520, USA. Omnipress.

Yoav Goldberg. The unreasonable effectiveness of character-level language models.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. 2015. LSTM: A search space odyssey. *CoRR*, abs/1503.04069.

Ruidan He, Wee Sun Lee, Hwee Tou Ng, and Daniel Dahlmeier. 2018. Effective attention modeling for aspect-level sentiment classification. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1121–1131. Association for Computational Linguistics.

Simon Hessner. 2018. Why are precision, recall and f1 score equal when using micro averaging in a multi-class problem?

Geoffrey Hinton. 2013. Recurrent neural networks.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759.

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *ICML*.

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

Eugine Kang. 2017. Blog post about a revealing introduction to hidden markov models.

Andrej Karpathy. 2015a. The unreasonable effectiveness of recurrent neural networks.

Andrej Karpathy. 2015b. The unreasonable effectiveness of recurrent neural networks.

Andrej Karpathy. 2016. Cs231n: Convolutional neural networks for visual recognition.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-thought vectors. *CoRR*, abs/1506.06726.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, abs/1506.07285.

Himabindu Lakkaraju, Richard Socher, and C. Manning. 2014. Aspect specific sentiment analysis using hierarchical deep learning.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

Zhizhong Li and Derek Hoiem. 2016. Learning without forgetting. *CoRR*, abs/1606.09282.

Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130.

Bing Liu. 2010. Sentiment analysis and subjectivity. In *Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca*.

Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. 2016. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR*, abs/1605.09090.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025.

Mika Viking Mäntylä, Daniel Graziotin, and Miikka Kuutila. 2016. The evolution of sentiment analysis - A review of research topics, venues, and top cited papers. *CoRR*, abs/1612.01556.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. *CoRR*, abs/1708.00107.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

Andrew W. Moore. Slides on hidden markov models.

Arjun Mukherjee and Bing Liu. 2012. Aspect extraction through semi-supervised modeling. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 339–348, Stroudsburg, PA, USA. Association for Computational Linguistics.

Harikrishna Narasimhan, Weiwei Pan, Purushottam Kar, Pavlos Protopapas, and Harish G. Ramaswamy. 2016. Optimizing the multiclass f-measure via biconcave programming. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1101–1106.

Sarang Narkhede. 2018. Understanding confusion matrix.

Jenny Carter Orestes Appel, Francisco Chiclana. 2015. Main concepts, state of the art and future research questions in sentiment analysis. In *Acta Polytechnica Hungarica - Volume 12, Issue Number 3*, pages 87–108. Acta Polytechnica Hungarica.

Bo Pang and Lillian Lee. 2008. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2(1-2):1–135.

Maximilian Panzner and Philipp Cimiano. 2016. Comparing hidden markov models and long short term memory neural networks for learning action representations. In *Machine Learning, Optimization, and Big Data*, pages 94–105, Cham. Springer International Publishing.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR*, abs/1802.05365.

S Petrović and D Matthews. 2013. Unsupervised joke generation from big data. 2:228–232.

Harsh Pokharana. 2016. The best explanation of convolutional neural networks on the internet!

Maria Pontiki, Dimitris Galanis, Haris Papageorgiou, Ion Androutsopoulos, Suresh Manandhar, Mohammad AL-Smadi, Mahmoud Al-Ayyoub, Yanyan Zhao, Bing Qin, Orphee De Clercq, Veronique Hoste, Marianna Apidianaki, Xavier Tannier, Natalia Loukachevitch, Evgeniy Kotelnikov, Núria Bel, Salud María Jiménez-Zafra, and Gülşen Eryiğit. 2016a. Semeval 2016 task 5 aspect based sentiment analysis (absa-16) annotation guidelines. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, San Diego, California. Association for Computational Linguistics.

Maria Pontiki, Dimitris Galanis, Haris Papageorgiou, Ion Androutsopoulos, Suresh Manandhar, Mohammad AL-Smadi, Mahmoud Al-Ayyoub, Yanyan Zhao, Bing Qin, Orphee De Clercq, Veronique Hoste, Marianna Apidianaki, Xavier Tannier, Natalia Loukachevitch, Evgeniy Kotelnikov, Núria Bel, Salud María Jiménez-Zafra, and Gülşen Eryiğit. 2016b. Semeval-2016 task 5: Aspect based sentiment analysis. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 19–30, San Diego, California. Association for Computational Linguistics.

Maria Pontiki, Dimitris Galanis, John Pavlopoulos, Harris Papageorgiou, Ion Androutsopoulos, and Suresh Manandhar. 2014. Semeval-2014 task 4: Aspect based sentiment analysis. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 27–35, Dublin, Ireland. Association for Computational Linguistics and Dublin City University.

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

Prabhu. 2018. Understanding of convolutional neural network (cnn)—deep learning.

Lawrence R. Rabiner. 1990. Readings in speech recognition. chapter A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

He Ren and Quan Sheng Yang. 2017. Neural joke generation.

Sara Rosenthal, Noura Farra, and Preslav Nakov. 2017. Semeval-2017 task 4: Sentiment analysis in twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 502–518. Association for Computational Linguistics.

Sebastian Ruder, Parsa Ghaffari, and John G. Breslin. 2016a. A hierarchical model of reviews for aspect-based sentiment analysis. *CoRR*, abs/1609.02745.

Sebastian Ruder, Parsa Ghaffari, and John G. Breslin. 2016b. INSIGHT-1 at semeval-2016 task 5: Deep learning for multilingual aspect-based sentiment analysis. *CoRR*, abs/1609.02748.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1988. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *CoRR*, abs/1606.04671.

Martin Schmitt, Simon Steinheber, Konrad Schreiber, and Benjamin Roth. 2018. Joint aspect and polarity classification for aspect-based sentiment analysis with end-to-end neural networks. *CoRR*, abs/1808.09238.

Kim Schouten and Flavius Frasincar. 2016. Survey on aspect-level sentiment analysis. *IEEE Transactions on Knowledge and Data Engineering*, 28:813–830.

M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Dmitriy Selivanov. 2018. Glove word embeddings.

Sheng-syun Shen and Hung-yi Lee. 2016. Neural attention models for sequence classification: Analysis and application to key term extraction and dialogue act detection. *CoRR*, abs/1604.00077.

Swati Soni and Aakanksha Sharaff. 2015. Sentiment analysis of customer reviews based on hidden markov model. In *Proceedings of the 2015 International Conference on Advanced Research in Computer Science Engineering &#38; Technology (ICARCSET 2015)*, ICARCSET '15, pages 12:1–12:5, New York, NY, USA. ACM.

Mark Stamp. 2004. A revealing introduction to hidden markov models.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. Weakly supervised memory networks. *CoRR*, abs/1503.08895.

Eörs Szathmáry and John Maynard Smith. 1995. The major evolutionary transitions. *Nature*, 374(1):227–32.

Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. 2015a. Target-dependent sentiment classification with long short term memory. *CoRR*, abs/1512.01100.

Duyu Tang, Bing Qin, and Ting Liu. 2015b. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1422–1432. Association for Computational Linguistics.

Duyu Tang, Bing Qin, and Ting Liu. 2016. Aspect level sentiment classification with deep memory network. *CoRR*, abs/1605.08900.

Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. 2017. Learning to attend via word-aspect associative fusion for aspect-based sentiment analysis. *CoRR*, abs/1712.05403.

A. M. Turing. 1950. I.—computing machinery and intelligence. *Mind*, LIX(236):433–460.

Yequan Wang, Minlie Huang, xiaoyan zhu, and Li Zhao. 2016. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615. Association for Computational Linguistics.

Lilian Weng. 2017. Learning word embedding.

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

Michael Wojatzki, Eugen Ruppert, Sarah Holschneider, Torsten Zesch, and Chris Biemann. 2017a. GermEval 2017: Shared Task on Aspect-based Sentiment in Social Media Customer Feedback. In *Proceedings of the GermEval 2017 – Shared Task on Aspect-based Sentiment in Social Media Customer Feedback*, pages 1–12, Berlin, Germany.

Michael Wojatzki, Eugen Ruppert, Torsten Zesch, and Chris Biemann. 2017b. *Proceedings of the GermEval 2017-Shared Task on Aspect-based Sentiment in Social Media Customer Feedback*.

Thomas Wolf. 2018. The current best of universal word embeddings and sentence embeddings.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016a. Hierarchical attention networks for document classification. In *Proceedings of the 2016*

Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1480–1489. Association for Computational Linguistics.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016b. Hierarchical attention networks for document classification. In *HLT-NAACL*.

YeGoblynQueenne. 2016. Comparing a recurrent neural network with a markov chain.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of CNN and RNN for natural language processing. *CoRR*, abs/1702.01923.

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2017. Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709.

Min-Ling Zhang and Zhi-Hua Zhou. 2014. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26:1819–1837.

Xiang Zhang and Yann LeCun. 2015. Text understanding from scratch. *CoRR*, abs/1502.01710.

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

RADIM ŘEHŮŘEK. 2014. Making sense of word2vec.