Department of Computer Science

Submitted in part fulfilment for the degree of BSc.

Aspect Based Sentiment Analysis

Carlo Segat

Version 1.00, 2019-April-30

Supervisor: Dr Suresh Manandhar

Contents

List of Figures

List of Tables

Executive Summary

Aim of the project The aim of this project is to evaluate systems for aspect based sentiment analysis (ABSA). I followed the approach set by SemEval 2016 and considered ABSA as comprised of 3 sub-tasks: aspect category detection (ACD), opinion target expression (OTE) and polarity assignment. Given an opinionated piece of text such as a restaurant review, we seek to determine towards which categories the opinions are expressed (ACD), what words in the text corresponds to the identified categories (OTE) and finally, whether the opinions are positive, negative or neutral. Evaluation is carried out separately for each sub-task.

My evaluation of existing methods consists in comparing simple multilayer perceptron models to more complex convolutional neural networks and long-short term memory networks.

I also focused on understanding whether explicitly including syntactical information does improve performance. I do this by using the information of dependency parse-trees and by leveraging pre-trained word embeddings trained with syntactical information. The assumption is that providing a system with explicit syntactic information can improve ABSA; as a simple example consider an opinion and its target, often the former is an adjective and the latter a noun ("good food"). Syntactical information should make a system aware of the functional properties of words and therefore should result in finer-grain distinction and better classification performance. The hope is that by adding syntactic information one could obtain the performance boost that would normally require some level of feature engineering.

Applications of Sentiment Analysis Sentiment analysis, also known as opinion mining, is often used to analyse large amount of online information and automatically get an understanding of people's attitude towards various topics. It finds applications in politics and marketing, where it often helps politicians and marketeers to tailor their decisions based on the information that can be gathered through social media.

Social Issues Nowadays the internet is the modern forum, where all kinds of issues are discussed and where people's opinions can circulate. If sentiment analysis becomes very precise at determining someone's opinion

and nuances of attitude towards a topic, it would not be hard to imagine its deployment as tool for political oppression, censorship or other kinds of hindering to freedom of speech.

Methods I experimented with various classes of artificial neural networks (ANN). I used simple multi-layer perceptrons (MLP), convolutional neural networks (CNN) and long-short term memory networks (LSTM).

The SemEval 2016, task 5, subtask 1, restaurant domain data-set was used to evaluate all methods. I spit it into the standard train, evaluation and test sets.

One fundamental step to leverage the capabilities of ANNs is to transform the input sentences into word embedding representations. Word embeddings are dense representation of words encoding many types of linguistic information. I experimented with three word embeddings, obtained with different methods and training data and therefore having different properties. Each neural network architecture is evaluated against a number of experimental setting where the following parameters vary: inclusion of syntactical information, word embedding, inclusion of punctuation and inclusion of stop-words. For one considered model the total number of experiments is therefore 24. A model is assessed on the test data using micro-averaged F1 as the evaluation metric. F1 scores are averaged over many runs to average the effect of randomness and provide robust metrics.

Results I did not managed to consistently show that including syntactical information does improve performance. Although I could confirm the claim in some of my experiments, many others blatantly contradicted it.

The expectation was to see better performance with CNNs and LSTMs over MLP as this latter simpler architecture is inferior in its capabilities of modelling language. CNN architectures seem to do better than LSTM on the task of polarity extraction.

Overall I found that the task specific Yelp embeddings perform better than the general purpose komn and google embeddings. This shows what I think is a limitation of standard word embeddings: to obtain competitive performances they need to be trained or at least fine tuned on the task at hand.

Since I varied my experimental settings by considering the sentences with the punctuation and the stop-words kept and removed, I found that stop-words and punctuation can be beneficial for some tasks.

1 Literature Review

1.1 History

"Sentiment analysis is a series of methods, techniques, and tools about detecting and extracting subjective information, such as opinion and attitudes, from language"[1].

People have concerned themselves with others' opinions since the dawn of civilisation, but the modern field of sentiment analysis, or often interchangeably opinion mining, is much more recent.

Seminal papers were published since the 1990's. For example Wiebe describes an algorithm to determine which character point of view a sentence is taking [2].

What really kick-started the field was the rise of the Web 2.0 and it's emphasis on user-generated content [1]. The increasing corpora of online reviews fuelled the advance of sentiment analysis during the 2000's. In 2002 Turney presents in [3] an algorithm to classify online reviews as positive or negative; he is able to achieve an accuracy of 74% only by using mutual information between the words in A review and two reference words, "excellent" and "poor".

The 2010's have seen a shift from online reviews to social media content. In 2010 Tumasjan proposed to use sentiment analysis to analyse tweets on the micro-blogging service Twitter in order to predict election's results [4]. Although not as accurate as traditional opinion polls, the results were promising.

In recent years the field has seen dramatic progress due to the rising of deep neural network models, which are the current state of the art in sentiment analysis and many natural language processing (NLP) tasks.

1.2 Applications

On the commercial side, we can think of systems that automatically scans the web for reviews about a company or its products. Such systems could identify problems as soon as they arise. For example ABSA, could help a computer company detecting negative reviews about a recently released laptop and specifically what the object of disappointment is.

Sentiment analysis could be employed to asses the literary reputation of authors by analysing how they are being cited by others. One metric could be the degree of agreement that a citing author grants to the source [5].

ABSA can also be integrated into other NLP systems such as question-answering systems and in general to improve human-computer interaction [5] . Differentiating between a user input that is factual and objective and one that is attitudinal and opinionated would be useful in order to provided a congruous and more human-like reply.

In E-commerce a seller's reputation is often determined by users feedback. Sometimes users can mistakenly leave a negative review (1 star) although their review clearly expresses satisfaction about their purchase. Sentiment analysis could detect such mistakes and prompt the user to correct it. [5].

Politics is another area where ABSA is employed. The applications are vast and can be callous. For example a candidate could use it asses or confirm which issues see the electorate strongly divided. The candidate could then avoid expressing a strong opinion towards such issues and instead concentrate the debate on topics that enjoy a broader agreement, in this way preventing a possible loss of consent.

1.3 SemEval

SemEval, short for semantic evaluation, is a series of evaluation of computational semantic analysis [6]. Computational semantic is a broad field of study whose ultimate goal is to endow computers with the ability to perform task that require human-understanding of natural languages. Every year a competition is hosted, teams taking part choose from a number of NLP tasks. The 2016 edition included tasks such as textual similarity, question-answering, parsing and ABSA.

My project approach to ABSA follows how the task is outlined in SemEval 2016, task 5, Sub-task 1: Sentence-level ABSA [7]. The task offered to the participants two domains they could choose from: restaurant and laptops. Like the majority of the participants, I chose the restaurant domain because it is simpler. As we can see from Figure 1.1 the laptop domain has many more aspect and categories to identify compared to the restaurant domain. Another reason why the laptop domain poses more difficulties is that the language used to talk about its different categories can be quite similar. Consider for example SOFTWARE and OS, in a sentence such as "I wiped nearly everything off of it, installed OpenOffice and Firefox, and I am operating an incredibly efficient and useful machine for a great price" it is ambiguous whether the customer is complaining about the OS or the installed software.

Entity Labels
RESTAURANT, FOOD, DRINKS, AMBIENCE, SERVICE, LOCATION
Attribute Labels
GENERAL, PRICES, QUALITY, STYLE_OPTIONS, MISCELLANEOUS

Entity Labels
LAPTOP, DISPLAY, KEYBOARD, MOUSE, MOTHERBOARD, CPU, FANS_COOLING, PORTS, MEMORY, POWER_SUPPLY OPTICAL_DRIVES, BATTERY, GRAPHICS, HARD_DISK, MULTIMEDIA_DEVICES, HARDWARE, SOFTWARE, OS, WARRANTY, SHIPPING, SUPPORT, COMPANY
Attribute Labels
GENERAL, PRICE, QUALITY, DESIGN_FEATURES, OPERATION_PERFORMANCE, USABILITY, PORTABILITY, CONNECTIVITY, MISCELLANEOUS

Figure 1.1: SemEval 2016 aspects and categories. The restaurant domain on the left and the laptop domain on the right. Note that out of all the possible pairs, not all apply.

SemEval 2016, task 5, Sub-task 1 is composed of 3 slots: aspect category detection (ACD), opinion target expression (OTE) and polarity assignment.

ACD can be considered as a multi-class, multi-label classification problem as we seek to attach aspect-category pairs from a predefined inventory to an input sentence. Figure 1.2 shows an example. In the restaurant domain the possible aspect-category pairs (output classes) are 12 in total.

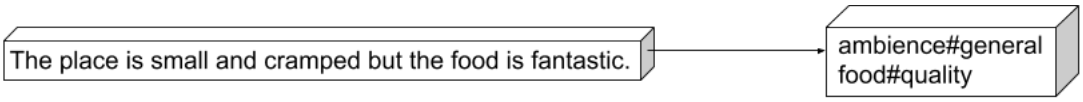


Figure 1.2: An input sentence and the correct aspect-category pairs it refers to.

OTE is a sequence-labelling task that given a sentence and a number of pre-identified aspect-category pairs, seeks to find

the words in the sentence that make up the opinion target. The tagging scheme I adopted distinguishes between: words that are not part of the opinion target, words that are the first word of an opinion target and words that are continuations of an opinion target. Figure 1.3 provides an example.

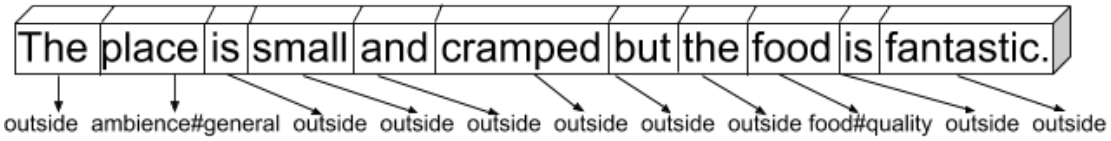


Figure 1.3: Example of tagging for OTE. If the sentence included a compound opinion target such as "Scotch eggs" the word "eggs" would be tagged as a continuation of the opinion target.

Polarity assignment is about assigning the label "Positive", "Neutral" or "Negative" to a sentence and a given aspect-category, based on the type of sentiment expressed towards the aspect-category pair.

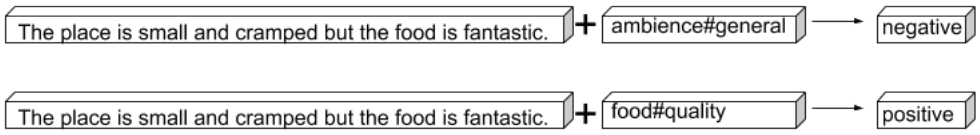


Figure 1.4: Polarity assignment. Note how the same sentence contains different polarities associated with different aspect-categories.

1.4 Methods

Traditionally there has been a distinction between two classes of methods for sentiment analysis: machine learning and lexicon based techniques [8].

Machine learning treats the problem as supervised classification. The input is a piece of text and one category from a predefined set. The output can be binary in the simple scenario or multi-class in models that differentiate between sentiment intensities. Examples of machine learning techniques are: Naive Bayes which uses Bayes theorem and assumes independence of features. Decision Tree, that makes use of decision

rules. Support Vector Machine (SVM) that fits a decision hyperplane in a high-dimensional space. Another common method is Maximum Entropy classifier [9]. Traditional machine learning methods achieved good results but required the labour-intense and error-prone process of feature engineering. They often rely on simple language models such as the bag-of-words model which disregard structure and takes into account only words presence or absence.

Lexicon based techniques do not require supervision. A lexicon is a list of words that are likely to belong to a certain class. For example a polarity lexicon for two classes consists of two list of words such as *[good, excellent, proper, ...], [bad, awful, cringe, ...]*. Similarly, a lexicon can be constructed for aspect categories. A lexicon based technique simply classify a sentence by comparing the frequencies of words of different classes.

More recently, deep neural networks methods have enjoyed broad adoption as they provide better performance for less feature-engineering. An important role in this shift is played by word embeddings that are replacing simpler language models.

1.5 Deep Neural Networks in NLP

Deep neural networks represent the current state of the art of many NLP tasks [10]. This current trend is in opposition with the shallow methods adopted previously such as SVM and logistic regression. DNN do not rely on sparse, hand-crafted features but allow for automatic and dense feature representation learning.

1.5.1 Convolutional Neural Network

Convolutional neural networks were initially employed in the field of computer vision as they proved effective at the task of image recognition [11]. Their effectiveness is also applicable in the field of NLP, as Yoon Kim showed in an influential paper on sentiment polarity[12].

Figure 1.5, taken from Zhang and Wallace [13] shows a simplified diagram of a CNN applied to binary sentence classification. This diagram is not dissimilar to the network described by Yoon Kim. Starting from the left we have the input sentence "I like

this movie very much". The raw sentence is converted into a dense vector representation and therefore we have a transformed input consisting of a length 5 vector for each word. Next comes the convolution layer where we can see 6 rectangles corresponding to convolution filters of three different sizes (4, 3 and 2). The filters independently apply a convolution operation (which is learnt during training) to the dense sentence representation and independently produce a feature map (FM). FMs, which we can see on the right of the convolution filters, encode whether the pattern of the convolution filter is present or not and how accentuated it is in different positions of the sentence. Similarly, since the convolutions are applied over n-grams of size equal to the filter's size, the feature maps can be interpreted as the extracted most important n-gram. In order to reduce dimensionality the max pooling operation is applied which simply takes the element of the FM with the highest value. Those elements are concatenated, as the multi-colour column array in the image shows. What follows is usually one or two fully connected dense layers and a softmax layer that produces the final prediction.

One strength point of CNNs is the ability of the convolution filters to capture semantic and syntactic patterns from the word embeddings. We can think of it in a similar fashion as CNNs would learn to detect features, such as edges, in images. Since the word embeddings are the source of information for the CNN, performance is strongly correlated to their quality.

CNNs' shortcoming are their inability to preserve sequential information as the results of the max pooling layer are binary indication of presence or absence of features. Another limitation is that long-distance relationships between words are lost.

1.5.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a class of artificial neural networks where a sequential input is processed element-by-element and at every time-stamp context information is retained 1.6. This characteristic makes RNN, differently from CNNs, able to model the sequential nature of natural languages [10] as they can remember information of past elements as new elements are processed. Another advantage over CNN is that RNN are flexible in terms of the length of the input sequence. The maximum length of an input needs to be fixed in CNN, whereas RNN can accept variable-length inputs.

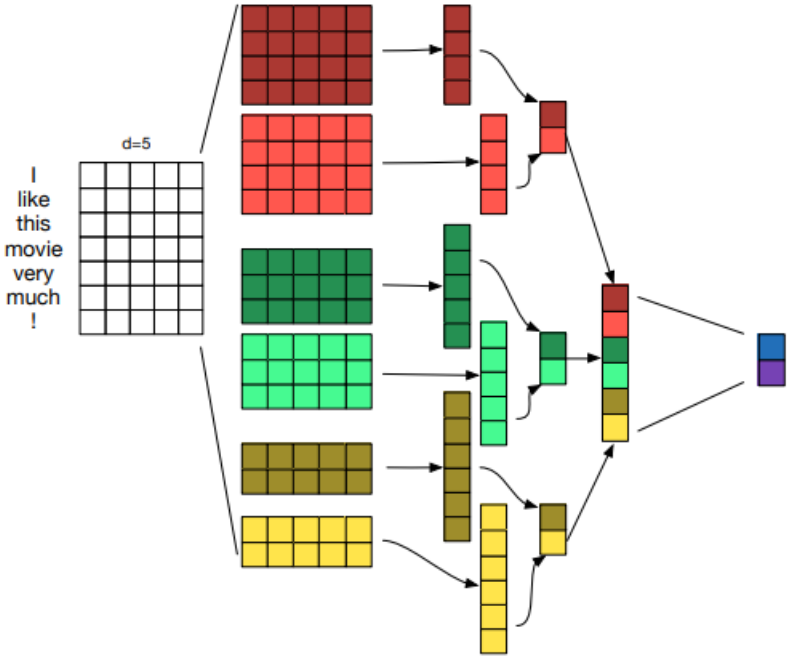


Figure 1.5: Diagram of a CNN architecture.

Despite their theoretical advantages, RNNs do not always outperform CNNs. As pointed out by Young et al. [10], the semantic requirements of the task at hand determines which architecture will perform better.

A well-known problem that occurs when training RNN, as early pointed out by Bengio et al. [14], is the exploding and vanishing gradient. Those mathematical phenomena occur when the gradient calculation of the loss function, in the backpropagation process, results in a very large or small number. The gradient exploding problem can be tackled with gradient normalisation or by using the relu activation function. On the other hand the vanishing gradient is an harder problem as it stops the weights of the network from changing .

RNNs do not model well long-distance relationships between elements of a sequence. The fundamental reason behind this problem is that the weights for the hidden state (encoding the context information) are not dependent on the network's input but only depend on the previous hidden state.

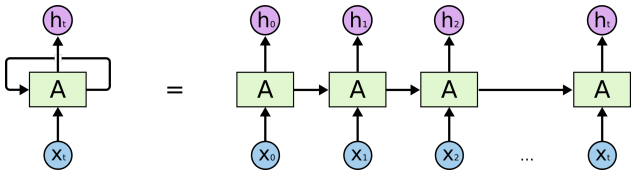


Figure 1.6: Schematics of a RNN. The unrolled version is on the right.

1.5.3 Long-short Term Memory Networks

Long-short term memory networks (LSTM) are another class of RNN. They solve the problem of standard RNN where long-distance relationship between elements of a sequence are not preserved. Their proposal, by Hochreiter and Schmidhuber [15], was strongly motivated by the need to ameliorate the issues of gradient vanishing, especially undermining with long input sequences.

The diagram in Figure 1.7 shows three cells of a simple LSTM. We can see how there are two states passed on between each cell: the hidden state which represents the current state of the output and the cell state which stores the context information. The equations in Figure 1.8 show how the outputs are calculated given an input and the previous cell's outputs. The σ is the sigmoid function, the square brackets represent the concatenation operation. The first sigmoid on the left is part of the forget gate, it scales the cell state thereby determining how much of it is kept. The input gate is composed of the second sigmoid and the tanh, it controls how much input is written to the network memory (the cell state). Finally the output gate disciplines how much cell state is output as the next hidden state. The hidden state of the last LSTM cell is normally considered to be the network output.

LSTM-based encoder-decoder systems have found applications in machine translation: a sequence from language A is passed through an encoding LSTM and the resulting hidden representation is then fed to a decoding LSTM that outputs a sequence from language B. Another example of a NLP application of LSTMs is to train an encoder-decoder system to reproduce the input sentence itself and use the output of the encoder as a sentence representation.

A common modification of the standard architecture is the bi-directional LSTM (biLSTM). Proposed in the 1990's by Mike Schuster and Kuldip K. Paliwal, [16] who applied it to RNN, it was found to improve the RNN performance on classification

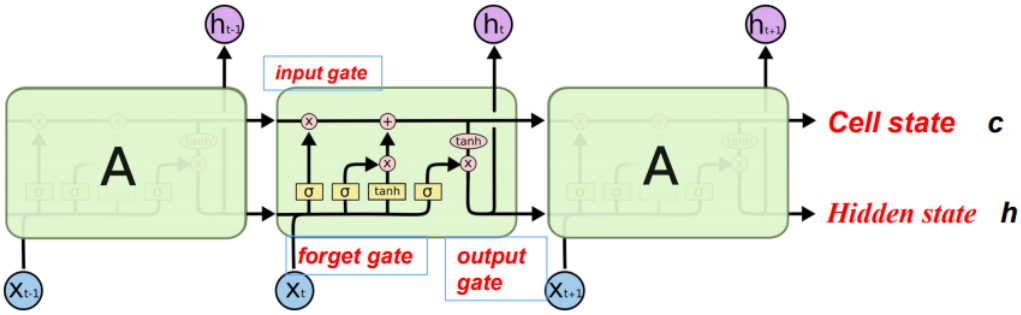


Figure 1.7: LSTM diagram taken from [17]. The Xs are inputs to the network, the Hs are the hidden states.

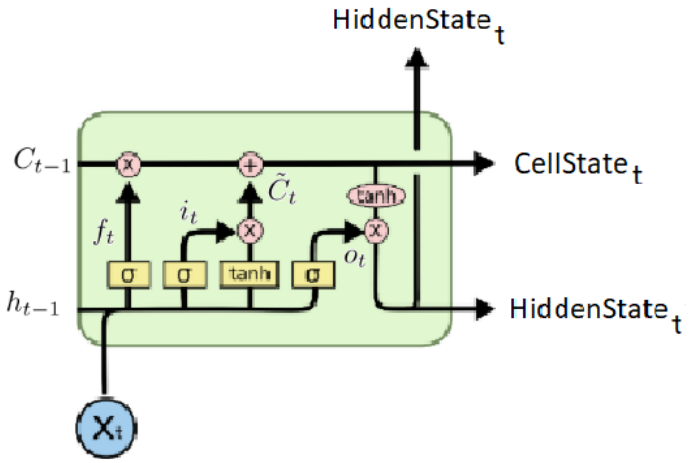
problems. The idea behind biLSTM is to train two LSTMs (with independent weights), one reading the input forward and the other backward. The two networks outputs are then concatenated. The backward read of the sentence captures dependencies between words that the forward read does not and is therefore an useful addition.

1.5.4 Attention Mechanisms

One shortcoming of the LSTM model so far described is that, especially for long sequences, the final hidden state alone is not appropriate to capture well all the information present in the sentence.

Attention is a mechanism that improves this by considering all the intermediate hidden states of the LSTM and not just the final one. The goal is to attach more importance to the hidden states coming from more important elements of the sequence. Usually a weighted sum of all the intermediate representations is considered to be the output of the LSTM with attention. A way of computing the weights consists in taking the dot product, considered to be a measure of similarity, between the hidden states and an application specific query vector.

There is flexibility regarding how the query vector is provided. It can be learnt during training like it is described in this method for ACD [19] where the query vector represents an aspect-category of the SemEval 2016 restaurant domain. By comparing each hidden states with such query vector it is possible to determine which words is more related to the aspect-category as Image 1.9 makes possible to visualise.



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

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

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

$$\text{CellState}_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

$$\text{HiddenState}_t = o_t * \tanh\left(C_t\right)$$

Figure 1.8: LSTM equations, images taken from [18]

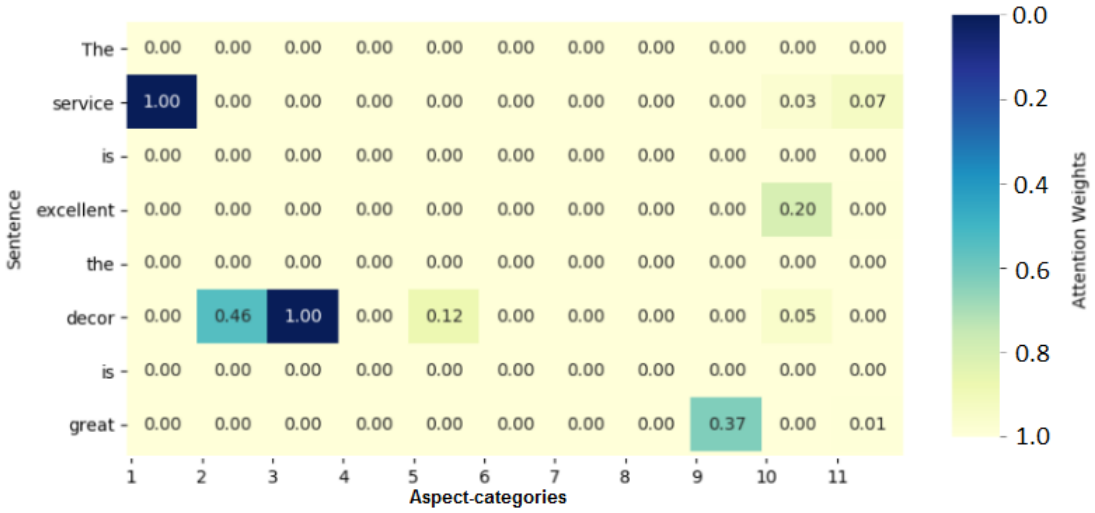


Figure 1.9: Attention visualisation from [19]. We can see that the weight for the word "service" with respect to the aspect-category 1 is 100%. Is reasonable to assume that the aspect-category query vector 1 is encoding the SERVICE#GENERAL aspect-category.

1.6 Word Embeddings

Together with DNN, distributed representation of words, known as word embeddings, are a crucial factor of many state-of-the-art results in NLP. The theoretical foundation behind word embeddings is the distributional hypothesis as put forward by Harris in 1954 [20]. The hypothesis is based on the statistical observation that words occurring is similar contexts tend to have similar meanings.

Practically, word embeddings are vectors of real numbers obtained from a self-supervised learning process. There are many ways of formulating the learning process but the key idea is to force the model to learn hidden representations of words that allow it to solve a problem that requires semantic understanding. Two common learning procedures are the skip-gram and the cbow models. Skip-gram formulates the problem as predicting the context words around a target word. Cbow does the reverse and seek to predict the target word given the context words.

It is possible to learn new word embeddings for each new task we are presented with, or use pre-trained ones. Pre-trained embeddings are an appealing option not only because is easier to adopt, but also because it allows for transfer learning. Embedding trained on very large text corpora such as Google News embeddings [21] allow to leverage vast, general, linguistic information into solving domain-specific tasks for which not much training data may be available.

Sparse representation of words such as vector space model, regardless of the measure used for the terms weights, make the system suffer from the curse of dimensionality. They also generalise poorly, as they do not consider together the contexts of different words with similar meanings (such as "house" and "apartment").

One useful property of word embeddings is the possibility of defining non-trivial operations on them. For example we could solve proportions such as woman:queen = man:king as Figure 1.10 shows.

Limitations of word embeddings include the inability to model compound words (such as "personal computer") as taking a linear combination of the individual words would not work. One way of of approaching this problem is to apply collocation extraction as a prepossessing step and treat compound words as one entity. Another limitation, that comes directly from taking

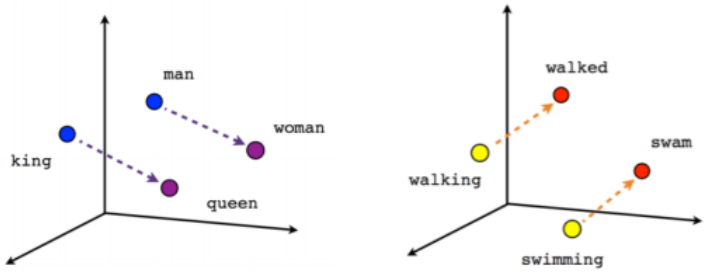


Figure 1.10: On the left symmetry between king-man and queen-woman; symmetry between verbs in different tenses on the right. Image taken from [22]

the distributional hypothesis, is that words expressing opposite sentiments (such as "good" and "bad") end up having a similar representation and are therefore treated equally. Since a word is mapped to a single word embedding, polysemy is not modelled effectively. For example, if the word "bank" appeared with the meaning of "financial institution" the same number of times it did with the meaning of "land around a river", the learnt word embedding would be an average encoding both meanings.

1.6.1 Kominos and Manandhar Embeddings

The Kominos and Manandhar embeddings (Komn) [23] are word embeddings trained with dependency contexts taken into account. Figure 2.1 shows an example of dependency parse tree and in particular node words and corresponding dependency contexts. Training is performed with a modified verison of the skip-gram model and there are three training objectives: given a node word, words connected in the dependency parse tree are predicted; given a target word, dependency contexts are predicted; given each dependency context words of a node, predict the node itself and the other dependency context words. In this way the embeddings encode their syntactical context.

Another advantage is that homonyms with different functional properties such as "can" when used as a noun and when used as a verb, are represented differently. It is important to realise that there are two kinds of entries in the komn word embeddings: plain words and words tagged with their dependency role. Therefore if we consider the word "can" there will be various entries depending on the dependency types: consider the following as a non exaustive list: the plain "can", "auxilary_can" for when it is used as an auxiliary of another verb and "determ-

inant_inverse_can" when the determinant "the" refers to the noun "can".

It has been shown that using the plain entries of those embeddings is better than other mainstream embeddings on many NLP tasks [24]. Although this superiority has been recognised, not many researchers have tried to incorporate the entries tagged with the dependency roles into their systems.

1.7 Hyperparameters and Evaluation

Hyper parameter selection is crucial for a NN performance, often marking the difference between the mediocre and the state of the art. Deep neural networks have many, real-valued hyperparameters, therefore the hyper parameter space is very large and their optimal values hard to find. Methods have been developed to search the hyperparameter space. Grid search is one approach that tries all possible hyperparameters combinations, it does eventually lead to an optimal solution but it is computationally impracticable. Another approach, suggested by Bergstra and Bengio [25], is randomized search; it avoids sweeping the entirety of the hyperparameter space by randomly sampling it a fixed number of times. Bergstra and Bengio showed that this approach is as good as grid search; the fundamental reason for this counter-intuitive finding is that most hyperparameters do not make a difference and only a few make a significant difference.

Gurevych and Reimers conducted extensive studies on hyperparameters selection and the most robust way of reporting performance [24], [26]. Their findings confirm that not all hyper parameters play the same role. Although their experiments were performed on NLP tasks such as Part of Speech Tagging (POS), Chunking, Named Entity Recognition (NER) and Event Detection, their results are relevant to most NLP tasks including ABSA. The hyperparameter that they recommed are proven to perform robustly independently on the other hyperparameters and random factors. For example, in order to determine which optimiser performs the best, *n* random network configurations are sampled from the hyperparameter space and each of them is trained with a different optimiser. Performance for each optimiser is then averaged over the *n* configurations.

In order to provide a robust estimation of a network performance, Gurevych and Reimers suggests that a score distribution

should be reported, rather than the maximum observed score [26]. Averaging the score over many runs is important to balance the effects of the non-determinism associated with neural networks training: for example, consider the random number generator seed, the weights initialisation values and the random dropout masks, those are factors that if changed can lead to convergence to different minima of the loss function. Another advantage of reporting score distributions is that the standard deviation gives an indication of the strength of the effects of random factors.

I summarise Gurevych and Reimers hyperparameters recommendations in Table 1.1

Table 1.1: Gurevych and Reimers hyperparameters recommendations

Hyperparameter	**Best Choice**
Word Embedding	Komn
Optimizer	Adam with Nesterov Momentum
Gradient Adjustment	Gradient Normalisation
Tagging scheme	BIO
Classifier for tagging tasks	CRF
Dropout for Recurrent Layers	Variational Dropout
LSTM Recurrent Units	100
Batch Size for small data sets	between 1 and 16

The Adam optimizer is a modification of the stochastic gradient descent algorithm. The fundamental difference is that a separate, dynamic learning rate is maintained for each of the network weights. The learning rate depends on past gradient updates (first and second moments), stopping the optimization process from oscillating too much and focusing its direction. Nesterov Momentum is a technique that essentially involves computing ahead of time the next gradient. Including this value in the calculation of the current weights update is beneficial especially in order to catch sharp minima .

Gradient normalization [27] consists in rescaling the gradient so that the L2 norm is below a certain threshold, this is done to avoid weights updates that are too large and would lead to the exploding gradient problem.

The BIO tagging scheme consist of three tags per category: Begin, Inside and Outside. Tagging a word with *Begin* means

that it represents the first word of a given category, *Inside* means that the word is part of the category but it is not the first world and finally *Outside* means that a world is not part of the specified category.

Single-chain conditional random field classifier (CRF) outperformed softmax on labelling tasks were there is interdependence between tokens. This is due to CRF maximising the global probability of a tag assignment whereas softamx makes token-by-token, indipendent predictions.

Variational dropout is a technique that instead of applying droput exclusively to the input and ouput vectors of a RNN, it also applies the same dropout mask to all the recurrent hidden representations. This is shown to constrast the overfitting tendency of RNN [28]

2 My Experiments

2.0.1 Training and optimiser

In all experiments the optimiser is nadam with the default parameters [29]. All networks are trained using back-propagation.

2.0.2 Loss function

If not stated otherwise, the loss function is categorical cross-entropy $-\sum_i (y'_i \log(y_i) + (1 - y'_i) \log(1 - y_i))$ or binary cross-entropy $-(y' \log(y) + (1 - y') \log(1 - y))$ depending whether a multiclass or a binary classifier is trained.

2.0.3 Word Embeddings

Each experiment is performed with three word embeddings: Komn embedings, Google News embeddings and word embeddings generated using word2vec and the Yelp restaurant reviews data-set [30] (Yelp embeddings). I chose the Komn embeddings because they are trained by considering syntactical relationships between words. It also contains different embedding for the same word depending on its syntactical function. It was trained on a corpora of 2 billion words. I chose the Google News embeddings because they were trained on a large 33 billion words corpora and it is a very popular choice for NLP tasks [21]. The Yelp embeddings, although trained on a significantly smaller corpora of about 500.000 sentences, is an interesting choice as it has been trained on restaurant review and is therefore a domain-specific embedding.

2.0.4 Syntactic Information

Syntactic information is either present or absent for each experiment. The way I encoded this information follows the con-

catenation method described by komninos and Manandhar in [23]. The syntactic embedding of a word X is the averaged sum of the syntactical embeddings of words that are directly connected in the dependency graph to word X. For example, in Figure 2.1 the syntactical embedding of 'price' would be the weighted sum of the embeddings for 'case_for', 'det_the' and 'nmod:for_in_eat'.

The dependency graphs of the restaurant reviews are obtained by using the Stanford CoreNLP Tool [31] which is used by Kominos and Manhandar as well and therefore gives matching keys into the Komn embeddings.

2.0.5 Stop Words and Punctuation

Experiments are repeated with the stop-words and the punctuation removed and kept. The number of experimental settings per architecture is therefore 24 ($3embeddings * 2syntax * 2punctuation * 2stopwords$). Although stop-words are commonly removed for sentence classification and sentiment analysis tasks, there are studies [32] that question the validity of this pre-processing step. Therefore I decided to experiment without committing to one choice.

2.0.6 Performance

For performance I used micro-average F1 score. Micro-average F1 is more suited for data-sets with class imbalance as it takes into account the number of elements per class.

F1 score formula: $2 * (Precision * Recall) / (Precision + Recall)$.

Macro-averaged F1 would calculate the precision term as follows: $Precision = \frac{\sum_c Precision_c}{C}$ where C is the number of classes.

Micro-averaged calculation of precision on the other hand: $Precision = \frac{\sum_c TruePositives_c}{\sum_c TruePositives_c + FalsePositive_c}$.

2.0.7 Tools

I conducted my experiments with Python using the Keras library for deep learning [33]. Keras is a high-level library which

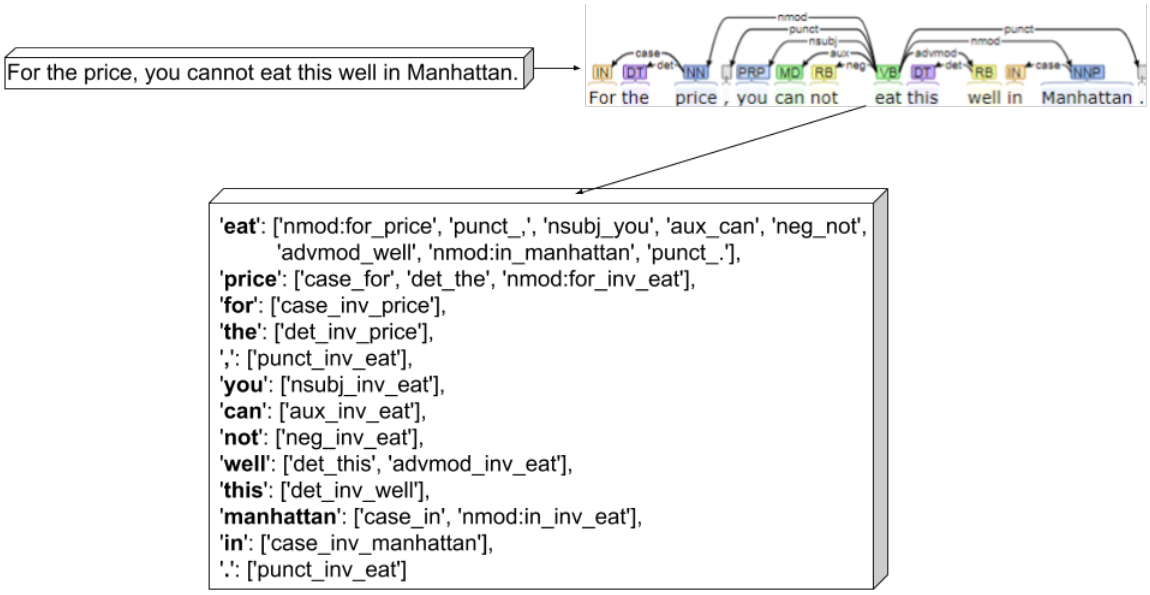


Figure 2.1: The dependency parse tree of a sentence and the mapping of each original word (in bold) with its dependency context.

```
<sentence id="1154550:1">
    <text>The place is small and cramped but the food is fantastic.</text>
    <Opinions>
        <Opinion target="place" category="AMBIENCE#GENERAL" polarity="negative" from="4" to="9"/>
        <Opinion target="food" category="FOOD#QUALITY" polarity="positive" from="39" to="43"/>
    </Opinions>
</sentence>
```

Figure 2.2: An example of an annotated sentence from the SemEval 2016 dataSet

uses a lower level backend: Google's TensorFlow [34] was my backend of choice. Another common backend is Theano, but as Gurevych and reimers reported in [24] there does not seem to be a performance difference between the two.

I trained my models using a GeForce GTX 1060 3GB and Intel i5-8500 CPU.

2.0.8 Data Set

I have used the manually annotated data-set provided as part of SemEval 2016. This includes a number of training and test sentences and corresponding opinion tuples as indicated by Figure 2.3. Figure 2.2 provides one example of such annotated sentences. The authors have incorporated SemEval 2015 data-set into the 2016 edition.

Lang.	Domain	Subtask	Train			Test		
			#Texts	#Sent.	#Tuples	#Texts	#Sent.	#Tuples
EN	REST	SB1	350	2000	2507	90	676	859

Figure 2.3: Data statistics for SemEval 2016.

2.1 ACD

Given a sentence we seek to determine to which predefined aspect-categories the sentence refers to.

2.1.1 MLP

Following the example of [30] I constructed a two layers fully-connected network with relu activation and softmax output. The input to the network are sum of words (SOW) like described here [30]. This approach consists in summing element wise and the averaging the word embeddings of the words composing a sentence. This method allows to deal elegantly with variable length sequences as every sentence is mapped to a vector with the same length.

When the syntactical information is used, the inputs I described so far are concatenated with the syntactical SOW of the sentence. As an example the syntactical SOW for the sentence in figure 2.1 would be the sum of the syntactical embeddings for 'eat', 'price', ..., 'in', '.'.

The output is an array of length 12 where each element represents the score for the corresponding class. Consult Figure B.1 for a graphical visualisation of the model.

Results The graphs of Figures A.1, A.3, A.2 show the results of 8 experiment settings each. Each of the 8 curves is obtained from 10 runs with a limit of 50 epochs, if early stop does not preempt the training sooner. Although I was not able to replicate the results of [30] it is interesting to note how the settings with the syntactical information always obtain higher average F1 scores and have less variance, symptom of robustness.

Keeping or removing punctuation does not seem to have a drastic effect with average F1 scores of 58,57 and 58,61 for the punctuation kept and removed. Stop-words have a slightly more pronounced but still minor effect, with average F1 scores

of 58,68 for the stop-words kept and 58,50 for the stop-words removed.

2.1.2 CNN

The inputs to the network are word indexes that are internally mapped to non-trainable word embeddings. When syntax is considered, each word embedding is concatenated with the corresponding syntactical embedding. The convolution layer consists of 300 filters of three sizes (2, 3, 5). A dense layer follows, taking as input the maxpooled and concatenated output of the convolution layer. The dense layer output dimension is one and sigmoid is the activation function. I adopted a one-vs-all strategy therefore I trained 12 binary models, one for each class. Consult Figure B.2 for a graphical visualisation of the model.

Results The results for the CNN, Figures A.4, A.6, A.5, seem to contradict the results for the Feed Forward network. Only with the google embeddings the syntax information improves the F1 score by a small margin.

Not surprisingly the more sophisticated CNN architecture achieves better average F1 scores than the simple MLP. Across all experimental settings the average F1 scores for the MLP and the CNN are respectively 58.59 and 65.49.

Keeping stop-words seems to be beneficial, with an average F1 score 65.82 compared to 65.15. Also keeping punctuation seems beneficial, with an average F1 score 65.76 compared to 65.22.

Removing punctuation

2.1.3 biLSTM

This architecture is similar to what is described in 2.1.2 with the important difference that a bidirectional LSTM layer substitutes the convolution layer. Intermediate states are discarded and the output of the LSTM is a concatenation of the final hidden states resulting from the forward and backward passing. The one-vs-all strategy is adopted here as well. See Figure B.3 for a schematic of the architecture.

Unfortunately I did not manage to produce the results for this experiment.

2.2 OTE

Given a sentence, we seek to tag each word with a label. The label indicates if the word belongs to an opinion target or not, it also indicates the type of opinion target if the sentence contains more than one.

The architecture I adopted for opinion target extraction is the one described by Hofer et al. in [35]. Their application is intended for named entity recognition on medical text, a domain with little annotated data available and where acronyms are very common. Although restaurant reviews do not abound with acronyms, I believe that it would not be reasonable to expect the Hofener et al. model, effective on a complex domain such as Electronic health records, to perform well in this simpler restaurant review scenario. With minor modifications to accommodate for the new input, I used the implementation as publicly made available by the authors themselves on git hub.

An input to my modified version of the network consists of the following: a sentence's words indices mapped to non-trainable word embeddings, characters indexes mapped to trainable characters embeddings and the syntactical SOW as described previously. In their original architecture Hofener et al. also use casing information as a feature. I decided to ignore casing information as I empirically noticed no improvements and because it was originally put in place to better deal with acronyms, which are often all-caps. See figure B.4

The loss function used for this task is sparse categorical cross-entropy because each input word can belong only to one class (in 2.1 the input sentence could belong to more categories simultaneously). The sparse version of categorical cross-entropy is mathematically equivalent but computationally more efficient as only the prediction value for the true class is considered.

I did not experimented with MLP and CNN as those architectures are not suitable for a labelling task.

Results The results in Figures A.7, A.8 and A.9 seem to contradict the hypothesis. On average including syntactical

information seems to worsen performance. The average F1 score over the three embeddings when syntax is included is 57.15 compared to 57.68 when it is not. The yelp embedding perform the best over komn and google. Removing the stopwords is beneficial with an average F1 score of 59.62 against 55.20 when they are kept. The same finding applies to the punctuation, removing it does improve performance as the average F1 score is 58.13 against 56.69 when is kept.

2.3 Polarity Detection

Given a sentence and one identified aspect-category we seek to determine the polarity (positive, neutral, negative) of the opinion towards the aspect-category. Aspect-categories are considered one at a time.

2.3.1 MLP

The network take as input the SOW of the words making up the sentence and, depending on the experimental scenario, also the syntactical SOW of the sentence as described previously 2.1.1. Also the identified aspect-category needs to be input and this is done by mapping the index corresponding to an aspect category to a dense vector representation that is learnt during training. The dimension of the aspect-category vector is set to match the dimension of the used word-embeddings. The two SOWs and the aspect category vector are finally concatenated and fed to a 2 layers fully-connected network identical to the one used for ACD 2.1.1 with the difference that the output is a vector of three probabilities for each class. See figure B.5 for reference.

Results The results seem to heavily contradict my assumption that including syntax would be beneficial. Figures A.10, A.11 and A.12 show that on average, not including syntax improves the F1 score. Including syntax with the google embeddings makes the worst difference with a drop in F1 score of approximately 10 points. Regardless of the embeddings my results show a wider spread for the settings where syntax was included, an indication that random effects are stronger for those settings.

Likewise for the ACD task, both keeping stop-words and punctuation is beneficial. Average F1 score of 74,56 against 73,93 for punctuation and 74,40 against 74,09 for stop-words.

2.3.2 CNN

This architecture is a modification of the CNN I described earlier 2.1.2. An important difference is that the output of the convolution is concatenated with the aspect-embedding, which is learnt during training. A relu fully-connected layer follows and a softmax layer gives the probabilities. See figure B.6 for reference.

Results The results for this architecture are more promising. Google and Yelp embeddings A.13, A.14 seem to prove that syntax is beneficial with an average increase in F1 score of 0.8411 and 0.3202 respectively. For the Komn embeddings A.15 it seems that including syntax does not have an impact, the F1 score difference is only 0.0058.

In this case both keeping the stop-words and the punctuation does not lead to better performance: 81.65 for keeping stop-words against 82.39 for removing them. Similarly the scores are 81.81 and 82.23 for the punctuation.

2.3.3 LSTM - ATAE

I experimented with the Attention-based LSTM for Aspect-level Sentiment Classification presented by Wang et al. in [36]. I based my implementation on a publicly available implementation on git hub [37]. The main idea behind the architecture is to learn aspect embeddings, which are concatenated to each word embedding input and to each hidden states that is output by the LSTM. The two concatenations serve different purposes: adding the aspect-category embedding to each input allows the LSTM to model the interdependence between words and input aspects. Concatenating to the hidden states allows the attention mechanism to use the aspect information to focus on relevant parts of the sentence. The architecture is described in Figure B.7. The attention mechanism compares the hidden states with the aspect embeddings to calculate a weight for each word. Those are used to scale the hidden states which are then summed. The output of the attention mechanism is

passed through a dense layer. Wang et al. esperimentally found that additionally considering the final hidden state of the LSTM improves performance. Therefore in parallel, the last LSTM hidden state is passed through an identical dense layer. The outputs of the two dense layers are summed and the hyperbolic tangent function is applied. Two dense layers follow and the final prediction is given by softmax.

Results The findings are not in favour of syntax. Figures A.16, A.17 and A.18 show the F1 scores and for all the three embeddings the no-syntax scenario achieves the best results on average.

The average F1 scores averaged over all architectures and experimental settings for polarity detection is 77.45 when syntax is included and 80.28 when it is not.

3 Future Work

It would be interesting to re-run the experiments with the syntax embeddings as the only input, without concatenating it to the conventional embeddings.

10 runs per experimental setting may not be enough to gain robust statistical evidence, for example Gurevych and Reimers run their experiments at least 100 times.

Concatenating the syntax embeddings may not be a good idea for every architecture. Increasing the input size without increasing the number of training samples deteriorates the ability of the model to learn and introduce dimensionality issues. This is a possible explanation why the feed-forward architecture for polarity detection 2.3.1 down-performs so drastically when syntax is included.

Another improvement would be performing a distant, semi-supervised learning step where the chosen embeddings are fine tuned to encode polarity of words. A method to do this is described in this submission to SemEval 2017 [38] where sentences obtained from scraping twitter and containing happy and sad emoji are considered as a noisy data-set for positive and negative sentiment polarity.

My reported scores for the polarity detection task are quite high, this is not solely because of the effectiveness of the models used but it is also due to the, perhaps inappropriate, use of micro-averaged F1 score to obtain such scores. With micro-average F1 a wrongly predicted "positive" class would still contribute to increase the F1 score. This is because the wrong prediction is given some credit for not having labelled the sentence as "neutral". More realistically one should use the average F1 score obtained from the three polarity classes independently.

4 Conclusion

My experiments did not manage to consistently show that providing syntax directly to a system for ABSA improves performance. In some instances there were improvements, but in general the performance seems to be lower when syntax is included. A simple architecture such as MLP is outperformed by more complex ones such as CNN and LSTM. Although the LSTM architecture is more suitable to model the sequential nature of language and therefore should be superior to CNN, results for the polarity experiments show the opposite.

The addition of syntax does not seem to provide the performance boost that can be obtained by using domain-specific embeddings: the yelp embeddings, trained on restaurant reviews, not surprisingly outperform the general purpose komn and google. Only in the experiment using CNN for polarity detection we can see that the google embeddings with the addition of syntax manage to give the best performance.

Interestingly, keeping punctuation and stop-words can be beneficial for some tasks such as ACD but it has negative impact on others such as OTE.

A Results Appendix

Graphs plotting F1 score distributions. For each task, each architecture is associated with three graphs, one for each word embeddings. Each individual graph plots 8 F1 score distributions obtained with different experimental parameters. For example, "google syntax stop-words punctuation" refers to an experiment with the google embeddings that does include the syntactical information and does not remove neither stop-words nor punctuation.

A.1 ACD

A.1.1 Feed Forward

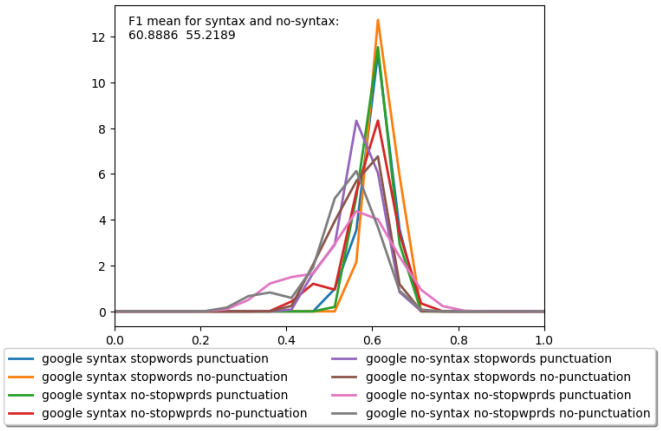


Figure A.1:

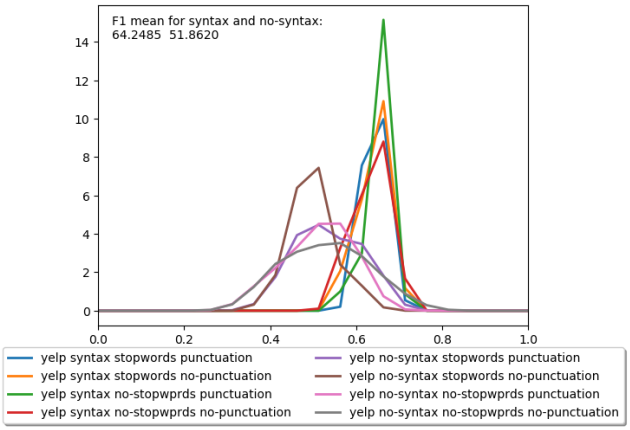


Figure A.2:

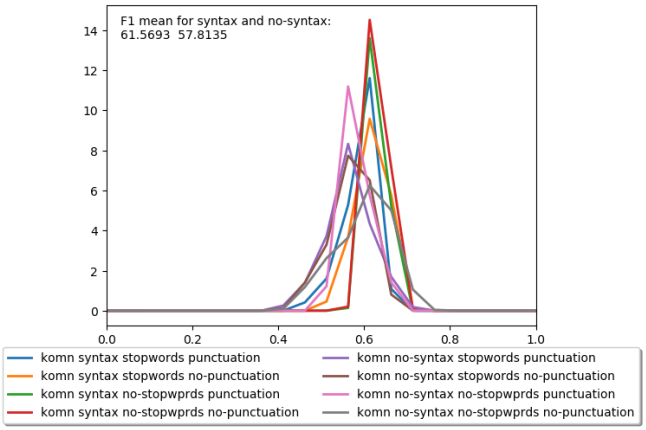


Figure A.3:

A.1.2 CNN

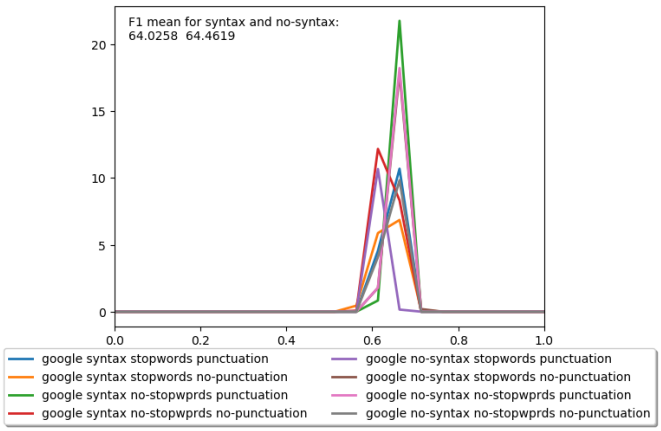


Figure A.4:

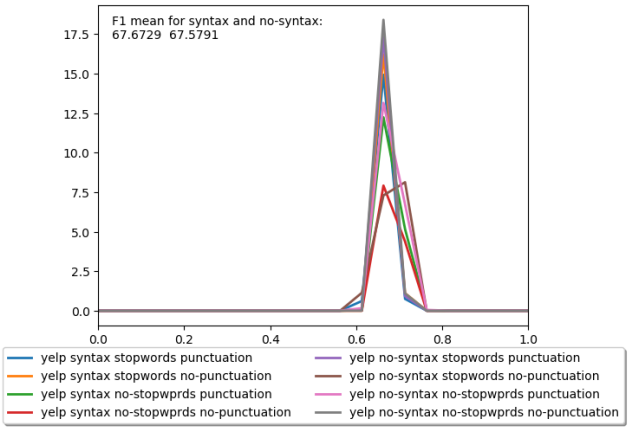


Figure A.5:

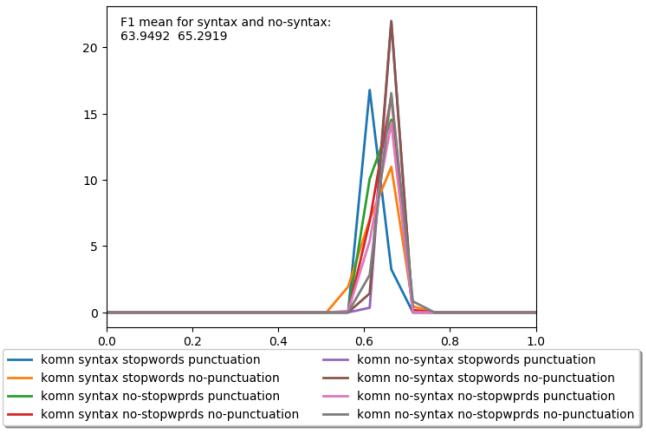


Figure A.6:

A.1.3 LSTM

No results available.

A.2 OTE

A.2.1 LSTM

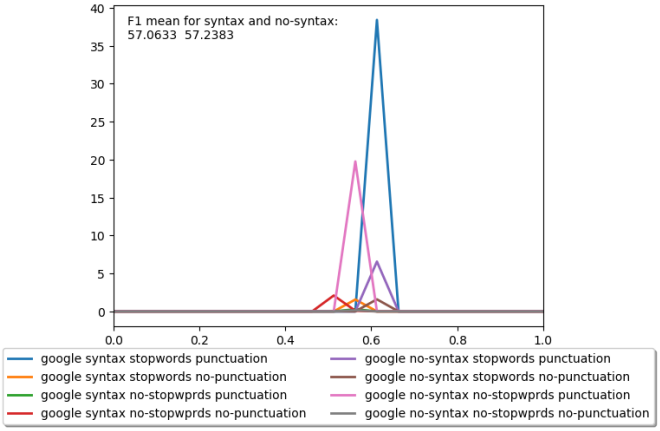


Figure A.7:

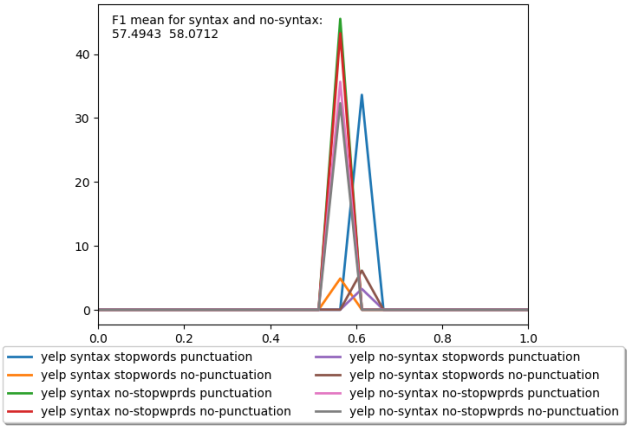


Figure A.8:

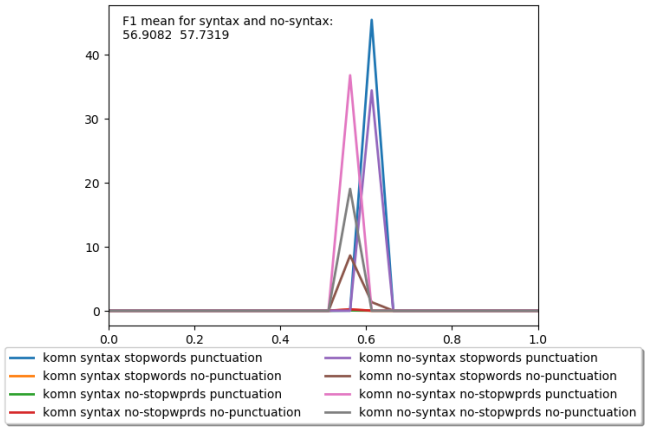


Figure A.9:

A.3 Polarity Detection

A.3.1 Feed-Forward

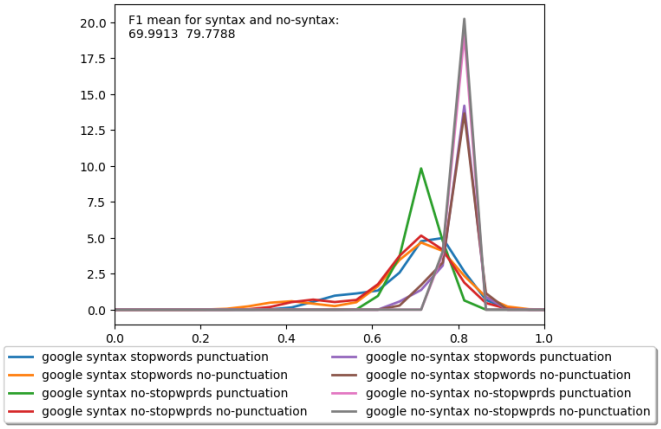


Figure A.10:

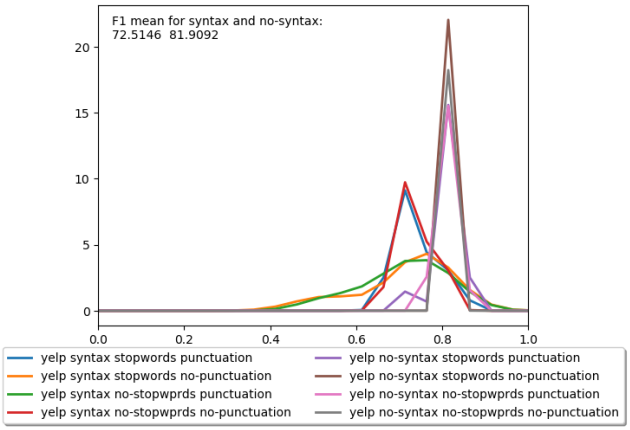


Figure A.11:

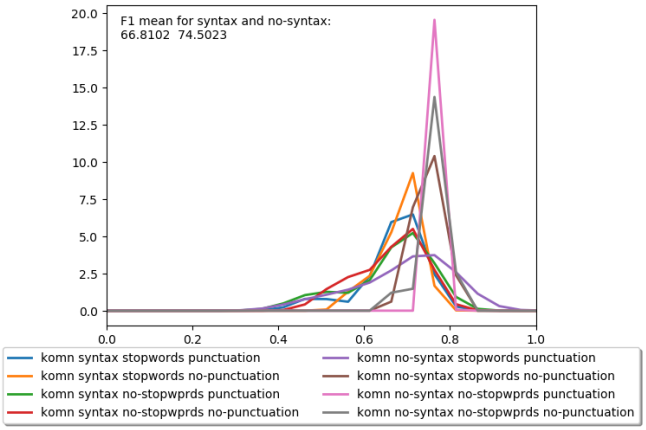


Figure A.12:

A.3.2 CNN

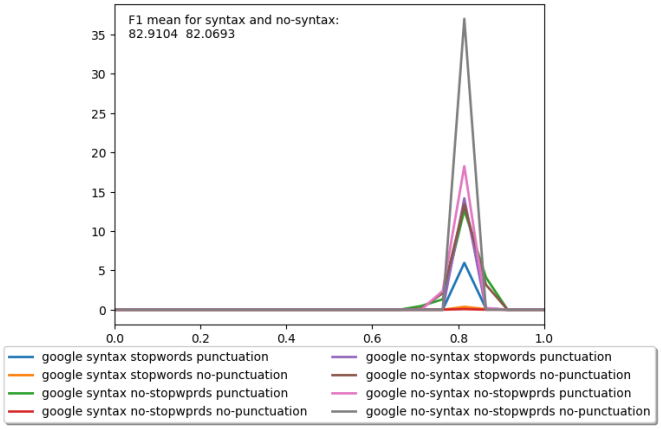


Figure A.13:

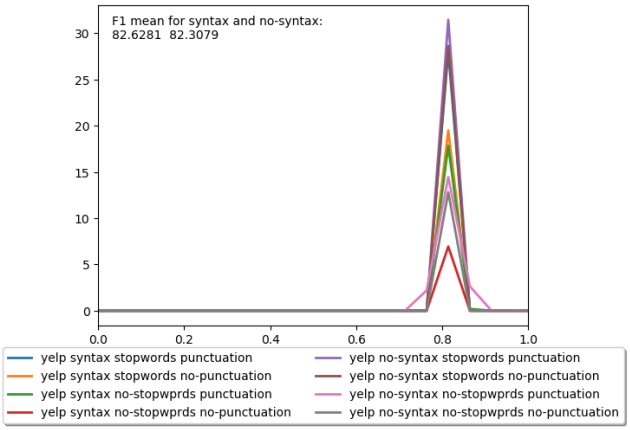


Figure A.14:

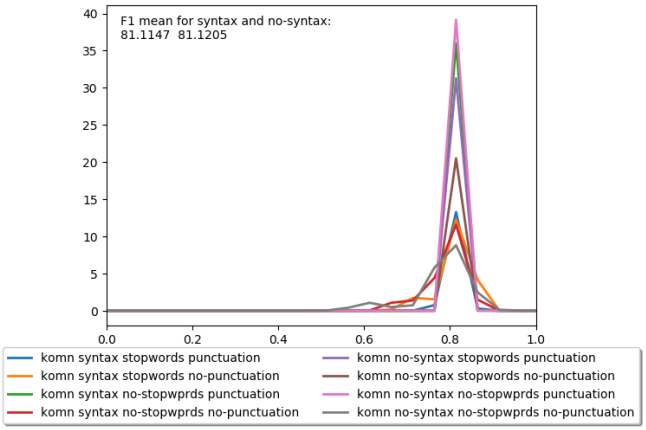


Figure A.15:

A.3.3 ATAE

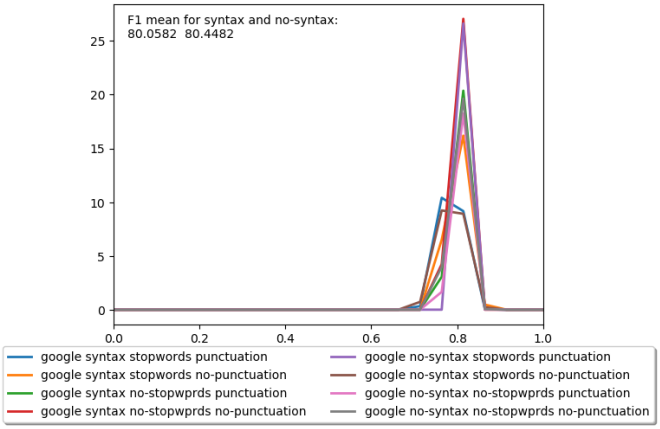


Figure A.16:

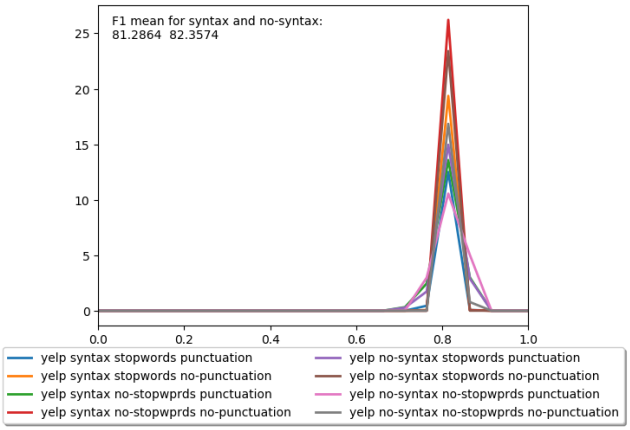


Figure A.17:

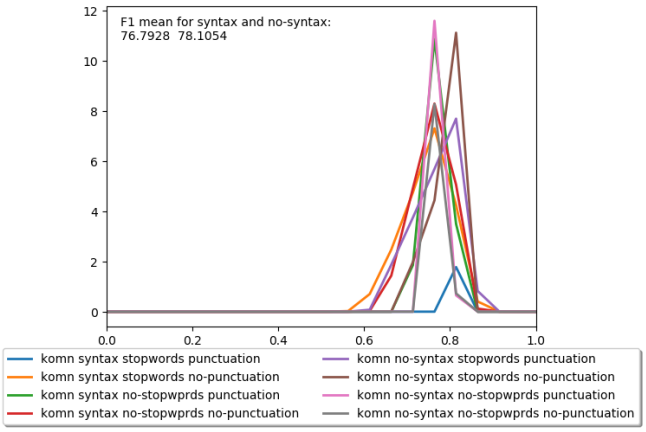


Figure A.18:

B Model Schematics Appendix

The model schematics are included here for reference. They have been automatically obtained using keras plot_model function but some manual editing has been applied. In particular drop-out layers have been excluded to reduce the height of the images and input layers have been coloured in light blue.

B.1 ACD

Schematics for ACD models

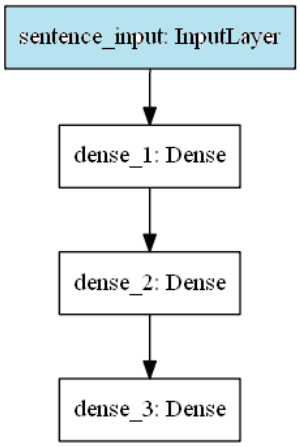


Figure B.1: Schematics of the feed-forward model used for aspect category detection.

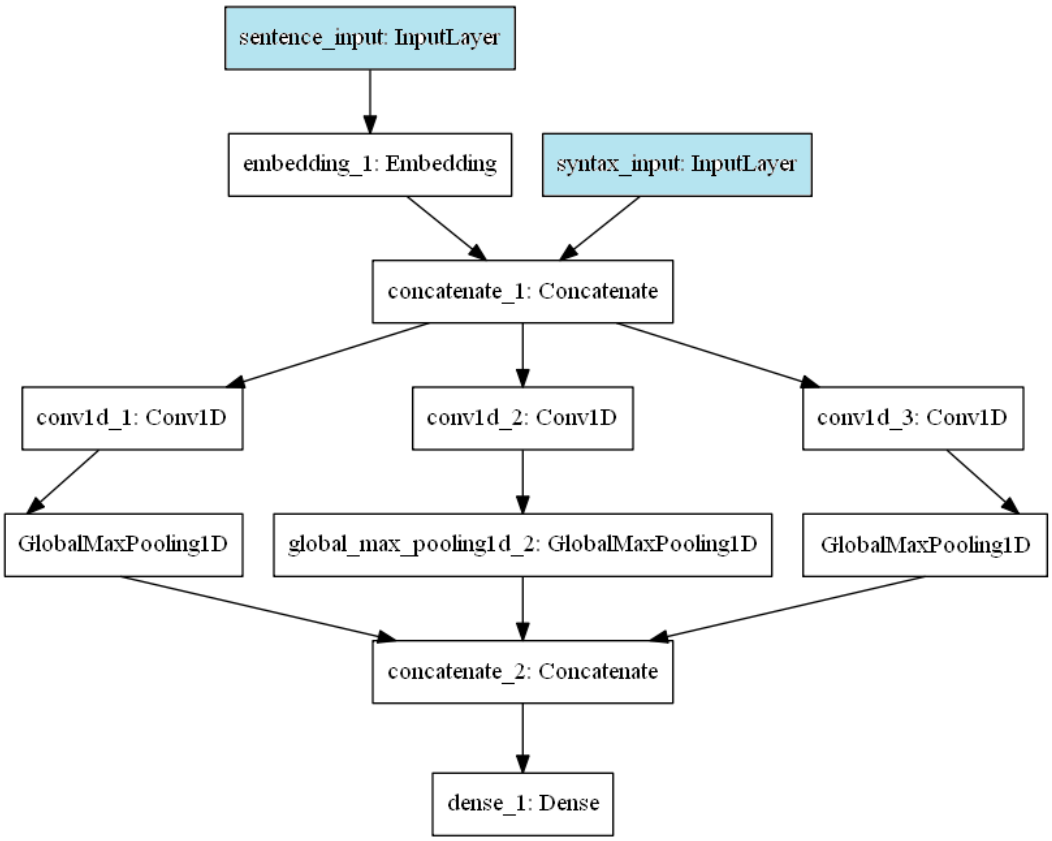


Figure B.2: Schematics of the CNN model used for aspect category detection.

B.2 OTE

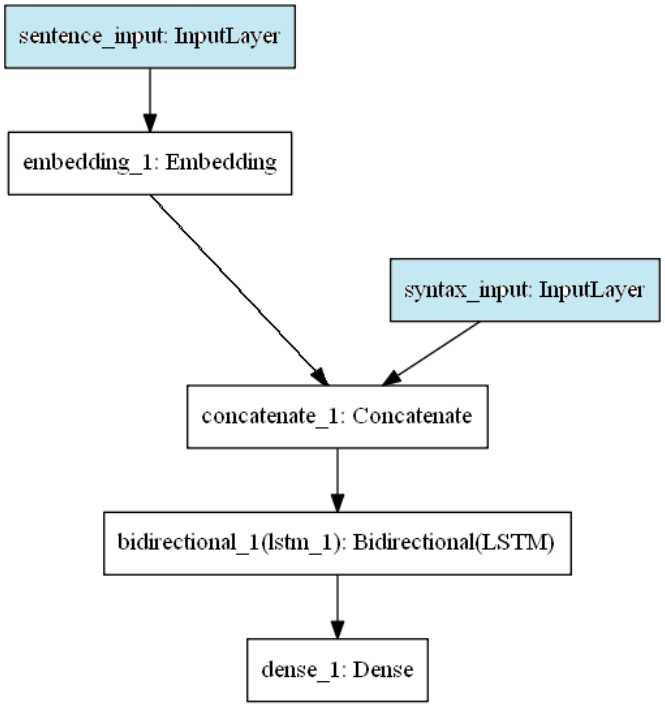


Figure B.3: Schematics of the LSTM model for aspect category detection.

B.3 Polarity Detection

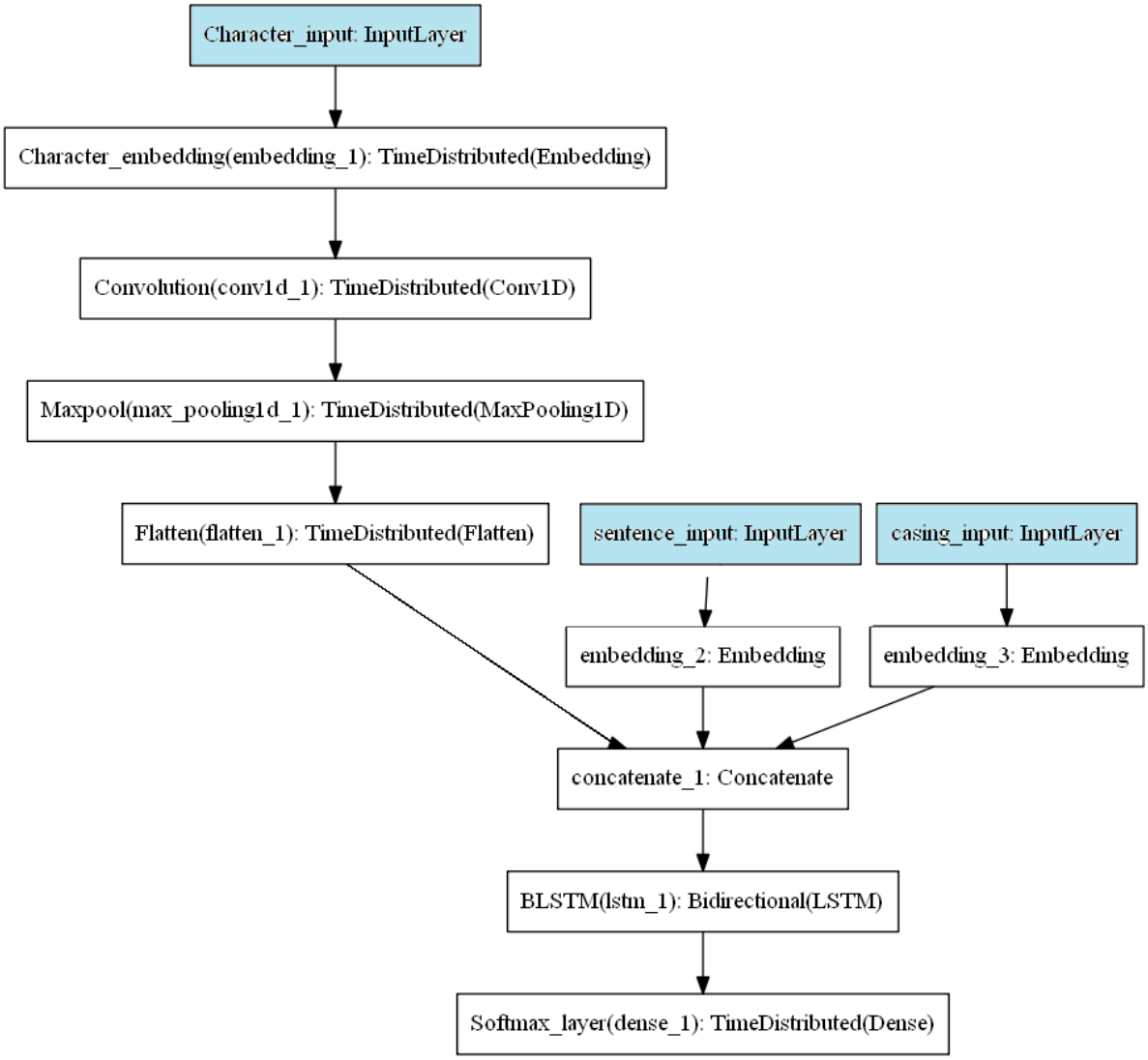


Figure B.4: Schematics of the LSTM Hofer et al. model for opinion target extraction.

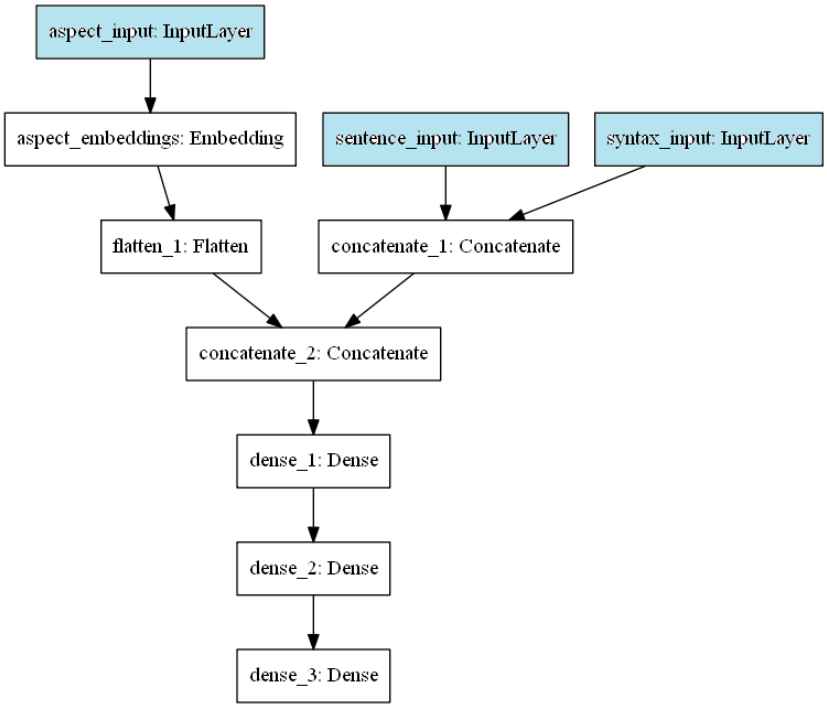


Figure B.5: Schematics of the feed-forward model used for polarity detection.

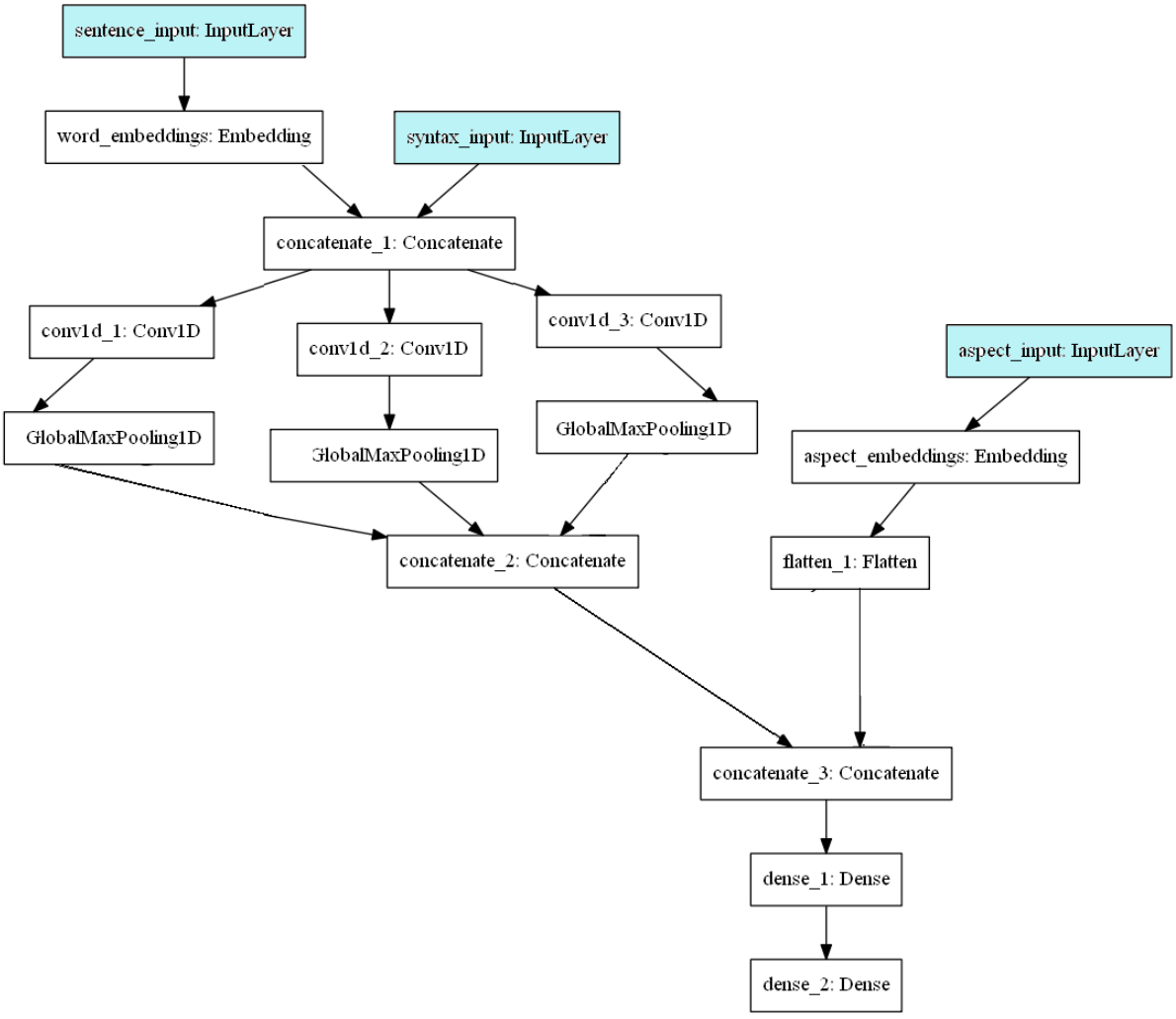


Figure B.6: Schematics of the CNN model used for polarity detection.

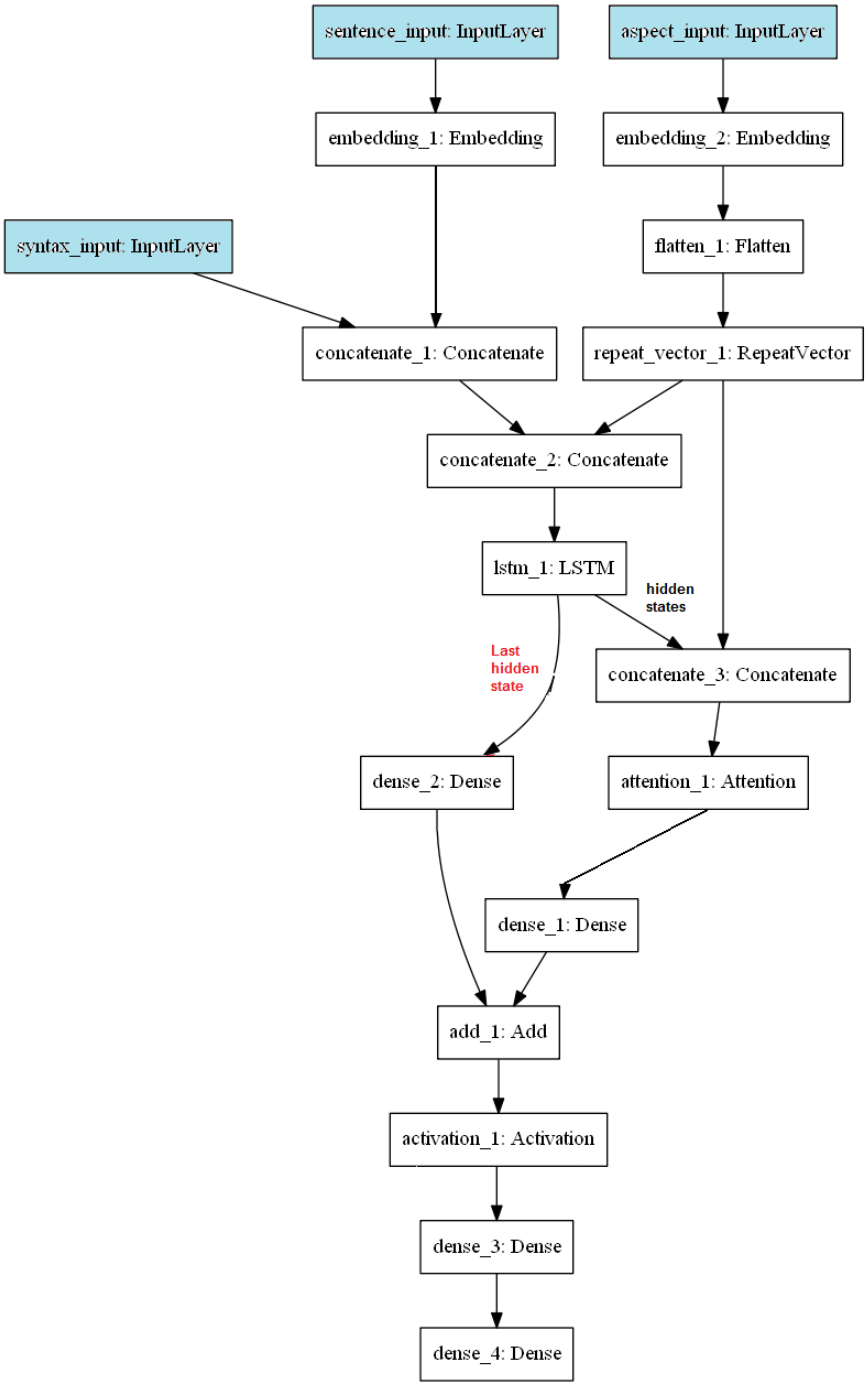


Figure B.7: Schematics of the ATAE model used for polarity detection.

Bibliography

[1] M. K. Mika V. Mantyla Daniel Graziotin, 'The evolution of sentiment analysis - a review of research topics, venues and top cited papers', *Computer Science Review*, vol. 27, pp. 16–32, Feb. 2018.

[2] J. M. Wiebe, 'Identifying subjective characters in narrative', *Department of Computer Science University of Toronto*, 1990.

[3] P. Turney, 'Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews', in *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 417–424. DOI: 10.3115/1073083.1073153. [Online]. Available: https://www.aclweb.org/anthology/P02-1053.

[4] A. Tumasjan, T. Oliver Sprenger, P. Sandner and I. Welpe, 'Predicting elections with twitter: What 140 characters reveal about political sentiment', vol. 10, Jan. 2010.

[5] B. Pang and L. Lee, 'Opinion mining and sentiment analysis', *Found. Trends Inf. Retr.*, vol. 2, no. 1-2, pp. 1–135, Jan. 2008.

[6] (). Semeval portal, [Online]. Available: https://aclweb.org/aclwiki/SemEval_Portal (visited on 19/04/2019).

[7] (2016). Semeval 2016, task 5, [Online]. Available: http://alt.qcri.org/semeval2016/task5/ (visited on 19/04/2019).

[8] P. J. B. T. MR. S. M. VOHRA, 'A comparative study of sentiment analysis techniques', *JOURNAL OF INFORMATION, KNOWLEDGE AND RESEARCH IN COMPUTER ENGINEERING*, vol. 2, pp. 313–317, 2 Jan. 2012.

[9] (2016). Nltk methods for text classification, [Online]. Available: https://www.nltk.org/book/ch06.html (visited on 19/04/2019).

[10] T. Young, D. Hazarika, S. Poria and E. Cambria, 'Recent trends in deep learning based natural language processing [review article]', *IEEE Computational Intelligence Magazine*, vol. 13, pp. 55–75, Aug. 2018. DOI: 10.1109/MCI.2018.2840738.

[11] K. Simonyan and A. Zisserman, 'Very deep convolutional networks for large-scale image recognition', *arXiv preprint arXiv:1409.1556*, 2014.

[12] Y. Kim, 'Convolutional neural networks for sentence classification', in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1746–1751.

[13] Y. Zhang and B. Wallace, 'A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification', in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Taipei, Taiwan: Asian Federation of Natural Language Processing, Nov. 2017, pp. 253–263.

[14] R. Pascanu, T. Mikolov and Y. Bengio, 'Understanding the exploding gradient problem', *CoRR*, vol. abs/1211.5063, 2012.

[15] S. Hochreiter and J. Schmidhuber, 'Long short-term memory', *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] M. Schuster and K. K. Paliwal, 'Bidirectional recurrent neural networks', *IEEE Transactions on Signal Processing*, vol. 45, pp. 2673–2681, 1997.

[17] S. Manandhar, *Sequence models - natural language processing lecture slides*, Mar. 2019.

[18] (). Understanding lstm networks, [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 23/04/2019).

[19] S. M. at al., 'Aspect category detection via topic-attention network', *arXiv:1901.01183*, 2019.

[20] Z. Harris, 'Distributional structure', *Word*, vol. 10, no. 23, pp. 146–162, 1954.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, 'Distributed representations of words and phrases and their compositionality', in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[22] (). Word embeddings, [Online]. Available: https://cbail.github.io/textasdata/word2vec/rmarkdown/word2vec.html (visited on 30/04/2019).

[23] A. Komninos and S. Manandhar, 'Dependency based embeddings for sentence classification tasks', in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 1490–1500.

[24] N. Reimers and I. Gurevych, 'Optimal hyperparameters for deep lstm-networks for sequence labeling tasks', *arXiv preprint arXiv:1707.06799*, Jul. 2017.

[25] B. James and B. Yoshua, 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research*, pp. 281–305, Feb. 2012.

[26] N. Reimers and I. Gurevych, 'Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging', *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 338–348, Nov. 2017.

[27] R. Pascanu, T. Mikolov and Y. Bengio, 'On the difficulty of training recurrent neural networks', in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, 17–19 Jun 2013, pp. 1310–1318.

[28] Y. Gal and Z. Ghahramani, 'A theoretically grounded application of dropout in recurrent neural networks', in *Advances in Neural Information Processing Systems 29*, 2016, pp. 1019–1027.

[29] J. B. Diederik P. Kingma, 'Adam: A method for stochastic optimization', *arXiv:1412.6980*, 2015.

[30] (2017). Sentence vector representation methods for aspect category detection, [Online]. Available: https://github.com/Nomiluks/Aspect-Category-Detection-Model/blob/master/Research%5C%20paper/paper.pdf (visited on 20/04/2019).

[31] (). Stanford corenlp - natural language softwaren, [Online]. Available: https://stanfordnlp.github.io/CoreNLP/ (visited on 22/04/2019).

[32] H. Saif, M. Fernandez, Y. He and H. Alani, 'On stopwords, filtering and data sparsity for sentiment analysis of twitter', *Proceedings of the 9th International Language Resources and Evaluation Conference (LREC'14)*, pp. 810–817, Jan. 2014.

[33] (). Keras, [Online]. Available: https://keras.io/ (visited on 20/04/2019).

[34] (). Tensorflow, [Online]. Available: https://www.tensorflow.org/ (visited on 20/04/2019).

[35] A. Kormilitzin, M. Hofer, P. Goldberg and A. Nevado-Holgado, *Few-shot learning for named entity recognition inuikeyinputleftarrow medical text*, Nov. 2018.

[36] Y. Wang, M. Huang, X. Zhu and L. Zhao, 'Attention-based lstm for aspect-level sentiment classification', pp. 606–615, Nov. 2016.

[37] (). Atae implementation on git hub, [Online]. Available: https://github.com/AlexYangLi/ABSA_Keras (visited on 25/04/2019).

[38] M. Cliche, 'Bb$_t wtratsemeval - 2017 task 4 : Twittersentimentanalysiswithcnnsandls$
pp. 573–580, Jan. 2017.