Ludwig-Maximilians-Universität München

Department of Statistics

Aspect-based Sentiment Analysis: A Theoretical and Practical Comparison of Different Approaches

Master's Thesis

Elisabeth Lebmeier

Supervisors: Dr. Christian Heumann
Matthias Assenmacher

Submission Date: 22.07.2021

Abstract

In the last five years, aspect-based sentiment analysis has been a field of great interest within Natural Language Processing. Supported by the Semantic Evaluation Conferences in 2014-2016, a lot of methods have been developed competing in improving performances on benchmark data sets. Exploiting the transformer architecture behind BERT, results were improved and efforts in this direction still continue today. In this thesis, we give a categorization of existing approaches and explain their details in terms of architecture. As our main practical contribution, we evaluate six of them on five benchmark data sets. Our experiments show that reported performance results are not reproducible in general.

Contents

1 Introduction **1**

2 Prerequisites **2**
2.1 Task Description . . . 2
2.1.1 Aspects, Sentiments and Opinion Terms . . . 2
2.1.2 ABSA vs. ATSC vs. ACSC . . . 3
2.1.3 Single-task vs. Multi-task vs. Pipeline Approaches . . . 4
2.2 Modelling Procedure . . . 4
2.3 Linguistic Features . . . 5

3 Theoretical Background: (Deep) Architectures **7**
3.1 CRFs . . . 7
3.2 CNNs . . . 8
3.3 RNNs . . . 9
3.4 Attention-based Models . . . 13
3.4.1 Attention Mechanism . . . 13
3.4.2 Transformer . . . 15
3.4.3 BERT . . . 16
3.5 Graph-based Architectures . . . 17
3.5.1 Graph Convolutional Networks (GCNs) . . . 18
3.5.2 Graph Attention Networks (GATs) . . . 18

4 Meta-Analysis of Approaches **20**
4.1 ATSC Methods . . . 20
4.1.1 CNN-based models . . . 24
4.1.2 RNN-based Models . . . 25
4.1.3 Attention-based Models . . . 33
4.1.4 BERT-based Models . . . 36
4.1.5 Models based on Extra Data . . . 38
4.1.6 Models based on Local Context Focus (LCF) . . . 44
4.1.7 Methods based on Graphs . . . 48
4.1.8 Models based on Capsule Networks . . . 61

4.2 ATE+ATSC Methods . . . 66
4.2.1 Models with a Pipeline Architecture . . . 67
4.2.2 Models with a Joint Labeling Scheme . . . 73
4.2.3 Models with a Collapsed Labeling Scheme . . . 82

5 Experiments **94**
5.1 Selected Approaches . . . 94
5.2 Data Sets . . . 96
5.2.1 SemEval-14 . . . 96
5.2.2 MAMS . . . 98
5.2.3 ARTS . . . 100
5.3 Data Preparation . . . 101
5.4 Evaluation Metrics . . . 102
5.5 Results . . . 104
5.5.1 General Outcome . . . 104
5.5.2 Model-specific Observations . . . 106
5.5.3 Data-specific Results . . . 109

6 Conclusion **111**

7 Outlook **113**

List of Figures **116**

List of Tables **118**

Bibliography **119**

Appendix **129**

Declaration of Authorship **137**

1 Introduction

Whereas for a very long period numerical analysis was in the center of attention, nowadays text is becoming more and more important as an object of research. The field of *Natural Language Processing (NLP)* has profited a lot from technical and algorithmic improvements within the last years. Before the successful times of Machine Learning and Deep Learning, NLP was mainly based on what linguists knew about how languages work, i.e. grammar and syntax. Thus, primarily rule-based approaches were employed in the past. Nowadays, far more generalized models based on neural networks can be set up that can learn the desired language features.

On the other hand, data in written form is available in huge amounts and thus might be an important source for valuable information. For instance, the internet is full of comparison portals, forums, blogs and social media posts where people state their opinions on a broad range of products, companies and other people. Product developers, politicians or other persons in charge could profit from this information and improve their products, decisions and behavior.

Thus, we focus on the specific topic of *Aspect-Based Sentiment Analysis (ABSA)* in this work. It is the task of identifying the polarity of single words, so-called aspects, within a given text based on their context. These methods can be used to establish a more differentiated view on a certain topic. We are going to have a look at both the theoretical and practical behavior of existing approaches.

This thesis is divided into four parts. The first is about the theoretical background in general, including a clarification of the terminology and different types of ABSA tasks. It is followed by a chapter introducing the reader to underlying deep learning architectures like CNNs, RNNs and BERT. The third part gives a categorization of existing methods and we go into the details of every model. For a subset of them, we explore their practical performance in the last chapter. There, we first give a detailed overview about available ABSA data sets, ways to prepare and evaluate the data and finally present the results of our analysis.

2 Prerequisites

2.1 Task Description

Aspect-based sentiment analysis gained a lot of attention through the corresponding tasks at Semantic Evaluation (SemEval) conferences in 2014-2016 [64–66]. However, terminology is not unique when referring to this topic. Thus, we give a short overview about related terms and state which ones we are going to use throughout this thesis. In general, the data used in this work are review texts about certain products, often restaurants and laptops. Usually, they are of arbitrary length and there are no restrictions to their content.

2.1.1 Aspects, Sentiments and Opinion Terms

To start with the basics, we take the following example from [9]: *"The place is small and cramped but the food is delicious."* Here, "place" and "food" are facets of the visited restaurant which we call *aspects* or more precisely *aspect terms*, sometimes also *targets*. Our task is to assign a *sentiment* to each of these aspects. By sentiment we mean a positive, negative or neutral polarity that the sentence conveys. In some cases, neutral polarity is equal to none and some works like [64] also include a conflict sentiment when the context is not clear about the aspect. This happens when contradicting statements are made with respect to a certain aspect. Here, the polarities would be positive for "food" and negative for "place". Sometimes, also *opinion terms* are explicitly extracted from texts. These are words that convey the sentiment. In our example, they are "small", "cramped" and "delicious".

2.1.2 ABSA vs. ATSC vs. ACSC

When we speak of *Aspect-Based Sentiment Analysis (ABSA)*, we take this as a generic term for several unique tasks. This is also caused by the inconsistency of terms in literature, where many different names are widely used. An alphabetical overview over the most common one can be found in Table 1. Since we want to be as precise as possible in this thesis, we are going to use different terms than ABSA to refer to the exact tasks. The main one is stated as subtask 2 of [64]: Assume that in each text aspect terms are already marked and thus given exactly as written in the text. Then, the task is a classification of sentiments for those aspect terms. This is why we find the term *Aspect Term Sentiment Classification (ATSC)* the most accurate and we are going to use it throughout this thesis.

Full Name	Abbreviation	Source
Aspect-Based Sentiment Analysis	ABSA	[25]
Aspect-Based Sentiment Classification	ABSC	[97]
Aspect-Level Sentiment Classification	ALSC	[97]
Aspect Sentiment Classification	ASC	[53]
Aspect Target Sentiment Classification	ATSC	[70]
Aspect Term Sentiment Analysis	ATSA	[91]
Aspect Term Sentiment Classification	ATSC	[35]
Target-Based Sentiment Analysis	TBSA	[46]
Targeted(-Dependent) Sentiment Analysis	TSA	[14, p.240]
Targeted Sentiment Classification	TSC	[4]

Table 1: Overview over common names for the task of classifying sentiments of given aspect terms.

In contrast to the definition of aspects from above, the authors of [14, p.243f] mean *aspect categories* when referring to aspects in their task of aspect-level sentiment analysis. This use of *aspects* is not unusual and thus it is important to be aware of the difference. It is also the reason why we insist on ATSC as the correct term for our task. Aspect categories do not have to be mentioned explicitly in the text and usually there is a small, fixed amount of them, e.g. "food", "service", "price" for restaurants and "performance" or "price" for laptops. Usually, this is called *Aspect Category Sentiment Classification (ACSC)*. Although this task might seem quite similar to ATSC, methods for one of these tasks cannot be easily employed on the other as ATSC approaches often rely on the explicit position of the aspect within the given text. Thus, we are not going any deeper into the topic of ACSC. Nevertheless, it is important to keep this difference in mind when getting to know new approaches in order to distinguish between ATSC and ACSC. Both can be meant when talking about aspect-based sentiment analysis.

2.1.3 Single-task vs. Multi-task vs. Pipeline Approaches

When referring to ATSC methods, we usually think of *single-task* approaches. These methods are designed to carry out only aspect term sentiment classification as the aspect terms are already given. Whether these were identified manually or by an algorithm is not relevant in this setting.

In practice, however, it is usually not the case that aspect terms are already known. Thus, approaches dealing with the step of aspect term extraction (ATE) have been developed. They can either work on their own or be combined to an ATSC method. These combined methods which we call ATE+ATSC are also channelled under the generic term ABSA. They can either be *pipeline*, *joint* or *collapsed* models. In pipeline models, ATE and ATSC are simply stacked one after another, i.e. the output of the first model is used as input to the second model. The latter two are often also referred to as *multi-task* models because both tasks are carried out simultaneously or in an alternating way. These models differ in their labeling mechanisms: There are two label sets for joint models, one to indicate whether a word is part of an aspect term and the other one to state the polarity of the aspect term. For collapsed models, one labeling scheme indicates whether a word is part of a positive, negative or neutral aspect term or not. Examples will be presented in the next section when labeling schemes are introduced.

2.2 Modelling Procedure

One usually does not simply apply a model directly on an input text, but some steps are necessary beforehand. We give a short overview about them here, but refer to literature for more details.

Tokenization At first, however, a sentence has to be divided into several pieces, most intuitively words and sub-words (in the case of long or combined terms). This procedure is called *tokenization*. Two famous approaches are WordPiece (cf. [86]) and BytePairEncoding (cf. [75]). Note that although the latter has "encoding" in its name, the outputs of the tokenization are still sequences of letters.

Embeddings Thus, it is most important to turn these tokens into numerical representations in the next step. As one-hot encoding for each word based on the complete

vocabulary would lead to sparse word vectors, other methods are applied. These dense, numerical vectors called *embeddings* use a mapping function that is often pretrained on a large text corpus. A widely used example are *GloVe embeddings* (cf. [62]). It is also possible to extract more complex, context-based representations from models like BERT (see Section 3.4.3).

Labeling In order to make model training for ATSC or ATE+ATSC possible, not only well-prepared input texts are needed, but also labels indicating the polarity of aspects or the aspects within a sentence. Therefore, different schemes are used. For the ATSC task, usually numerical labels are sufficient, with 1 indicating *positive*, -1 *negative* and 0 *neutral*. However, labeling aspects within a sentence is more difficult. A widely used approach is based on the so-called *BIO(ES)*-scheme, where the single letters stand for *beginning*, *inside*, *end* of and *single-word* aspect and *O* stands for *Other*, i.e. no aspect at all. Depending on the chosen scheme, the last word of an aspect term is either labeled as *I* (BIO) or *E* (BIOES). These positional labels can stand alone, can be more informative together with a suffix "ASP" for "Aspect" or can be combined with a sentiment, e.g. "B-POS". The following example from [53] about a laptop points out the differences:

Text	Nice	operating	system	and	keyboard
Joint: Aspects	O	B	I	O	B
Joint: Sentiments	O	POS	POS	O	POS
Collapsed	O	B-POS	I-POS	O	B-POS

Table 2: Comparison of joint and collapsed labeling scheme according to BIO tags.

2.3 Linguistic Features

Having encountered tokens as a small unit of text in the previous section, we now present some linguistic features that can be extracted on token-level. We focus on those that will appear in the following parts of the thesis.

POS Tags *Part-of-speech (POS)* tags are very common. Each word is assigned its part of speech, e.g. noun, verb or adjective, with the corresponding abbreviation, depending on the chosen tagging scheme. For detailed definitions and explanations on different tagging schemes, we refer the reader to [36, p.149ff].

Chunks Based on the part of speech tags, a sentence can be divided into several phrases, e.g. noun phrase (NP) and verb phrase (VP). Usually, they are disjunct and called *chunks* (cf. [2]).

Constituent Trees Starting with chunks, *constituent trees* give more detailed information about how text is structured. As illustrated in Figure 1, constituent trees are not directed and they rely on POS tags. Following the explanations on abbreviations in [36, p.149ff], for the first noun phrase "Economic news", we get to know that "economic" is an adjective and "news" is a singular or mass noun.

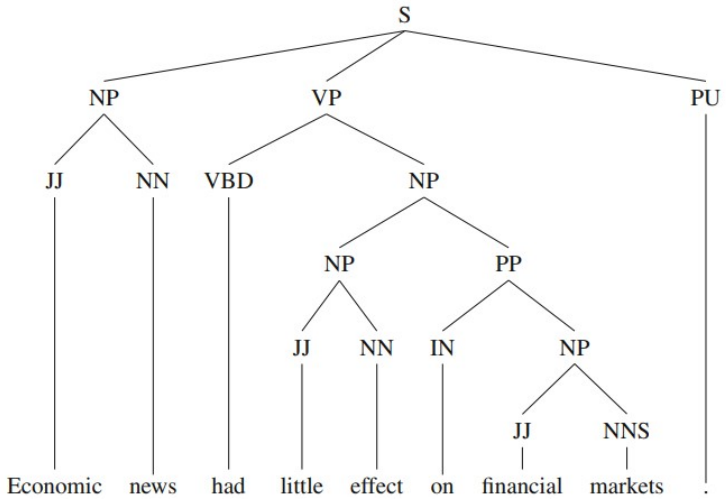


Figure 1: Example for a constituent tree. *Source:* [14, p.82]

Syntactic Dependency How words interact with each other with respect to syntax, is stated in the syntactic dependency relations. They can be represented in so-called *dependency trees* as shown in Figure 2. In contrast to the constituent trees from above, they are directed, with tokens as vertices and the relations as edges or arcs (cf. [14, p.82]). The figure illustrates for the same example sentence that the root of the tree is the predicate "had". To it, the subject "news" belongs which is modified by "economic". The word where an arrow starts is considered the *head* and the one where it ends the *child* (cf. [2]).

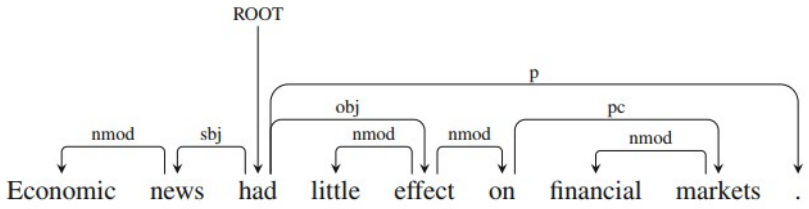


Figure 2: Example for a dependency tree. *Source:* [14, p.82]

3 Theoretical Background: (Deep) Architectures

After a clarification of the task and some basic definitions in the previous part, we now introduce the architectures that work as basis for the models in Section 4. We strive to give an overview about the essential characteristics of each model and refer to the corresponding papers for details. The models appear roughly in the order of creation, starting with simpler ones and ending with more complex architectures.

3.1 CRFs

Conditional Random Fields (CRFs) were proposed in [41] and are widely used on all kinds of data. Their key feature is that instead of tag predictions for single words, the whole sequence of labels is predicted at once. Following the explanation of [36, p.162f], the probability distribution over a sequence of labels $Y = y_1, \ldots, y_n$ is defined as

$$p\ (Y|X) = \frac{\exp\left(\sum_{k=1}^{K} w_k F_k(X, Y)\right)}{\sum_{Y' \in \mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k F_k(X, Y')\right)} \tag{3.1}$$

with inputs $X = x_1, \ldots, x_n$. The set of all possible label sequences is denoted by $\mathcal{Y}$ and weights by w_k. The K functions F_k are called global features and consist of local, position-wise features $f_k(y_{i-1}, y_i, X, i)$, more precisely $F_k(X, Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$. They extract feature vectors from X at position i based on labels y_{i-1} and y_i. Using only these two labels is the characteristic of a so-called *linear-chain CRF*. This type of CRFs is usually applied in NLP. In general, a CRF is trained by minimizing the negative log likelihood of Equation 3.1 (cf. [21, p.224]).

3.2 CNNs

Convolutional Neural Networks (CNNs) are a modification of fully-connected neural networks and they were originally designed for Computer Vision (cf. [44]). We follow the notations of [14, p.29] to present a CNN architecture for sentence classification. The procedure is illustrated in Figure 3.

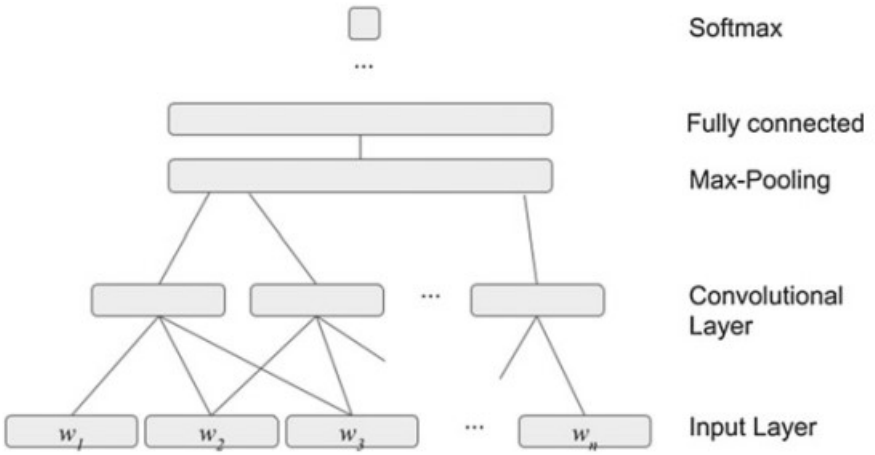


Figure 3: CNN Architecture. *Source:* [14, p.29]

CNNs receive their name from the convolution operator which is its main building block. During the convolution, a so-called *filter matrix* U is applied on the input words W to extract features

$$c_i = \tanh(UW_{i:i+h-1} + b)$$

with bias b. Note that not all the words in W are used, but just a window of size h. This is a characteristic of the CNN which reduces the number of connections between the input layer and the new feature layer in comparison to a fully-connected NN. Applying the same filter U everywhere on the input, makes a CNN obey to the principle of translation invariance. Thinking of the original use-case of image classification, this is sensible as a certain object should be detected anywhere in an image. Additionally, trainable parameters are shared and thus fewer computations are necessary with respect to fully-connected NNs. The next operation on the new features c is pooling. Depending on the chosen variant, the maximum (*max pooling*) or the average (*average pooling*) value is forwarded to the next layer. This step reduces the dimensionality of the representations. The combination of applying a convolution followed by pooling can be repeated several times. Then, a fully-connected layer and a softmax are applied for classification.

3.3 RNNs

Working in the field of Natural Language Processing, our data is text, no matter whether it was directly written or transcribed from audio. Due to the human way of processing it, i.e. reading word by word, it is considered sequential data. A model class especially adapted to this kind of data are *Recurrent Neural Networks (RNNs)*. As explained in [96, p.321-326], RNNs are neural networks with hidden states which are designed to store knowledge from previous inputs. In literature, the order of the inputs is often given as time steps $t = 1, \ldots, T$. By storing all the information encountered until a certain point of time, i.e. a certain word, the human reading process is imitated and the model can rely on the same knowledge as a human could. The basic architecture of an RNN consists of input, hidden and output layers for each time t which is illustrated in Figure 4.

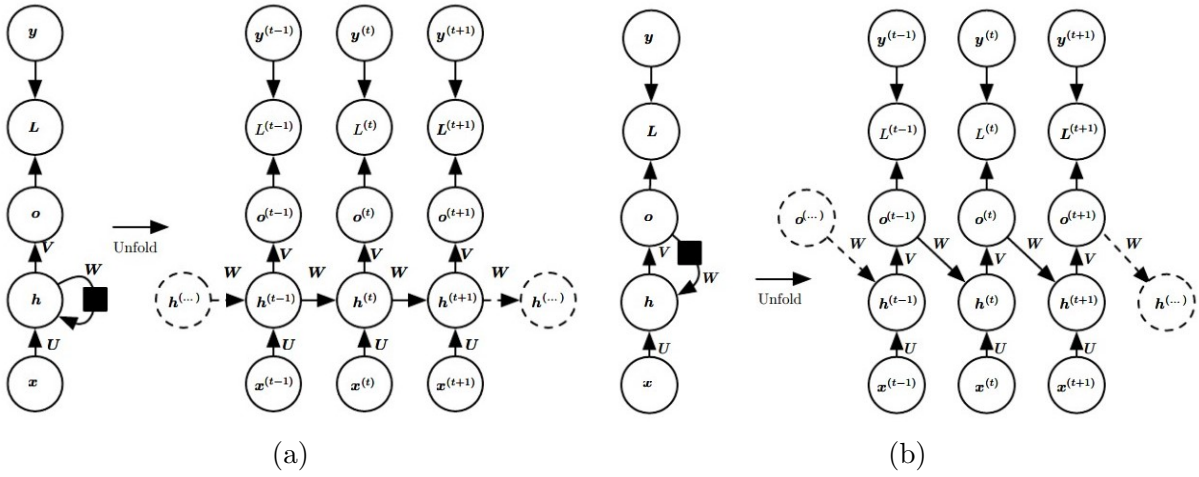


Figure 4: RNN Architecture with recurrence (a) within the hidden layer h (b) between predicted outputs o and hidden layer h. The input is denoted as x, the loss as L and true outputs as y. *Source:* [23, p.378,380]

To highlight the differences to a standard neural network, we shortly give the equations for a standard fully-connected two-layer neural network:

$$H = \phi(XW_{xh} + b_h)$$
$$O = HW_{hq} + b_q.$$

Here, O stands for the output and H for a layer in between. Furthermore, we have weights and biases $W_{xh} \in \mathbb{R}^{d \times h}$, $b_h \in \mathbb{R}^{1 \times h}$, $W_{hq} \in \mathbb{R}^{h \times q}$ and $b_q \in \mathbb{R}^{1 \times q}$. However, for a recurrent neural network with inputs $X_t \in \mathbb{R}^{n \times d}$ at time t, the corresponding hidden state is $H_t \in \mathbb{R}^{n \times h}$. It is calculated using the current input X_t, the previous hidden state

H_{t-1} and the activation function ϕ of the hidden layer:

$$H_t = \phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h),$$

where $W_{hh} \in \mathbb{R}^{h \times h}$ is the new weight parameter for the hidden state. The weight W_{xh} and the bias b_h are the same as for the architecture without a hidden state. The final output then has the form

$$O_t = H_t W_{hq} + b_q,$$

with W_{hq} and b_q having the same sizes as in the architecture from above. This kind of neural network is called *recurrent* as the definition of the current hidden state is based on the same definition of the previous hidden state. Of course, this recurrence cannot only take place in a hidden layer as shown in Figure 4a, but also between output and hidden layer as illustrated in Figure 4b.

In general, RNNs are optimized by gradient-based methods. To do so, the gradient of the loss with respect to weights and biases is calculated by the chain rule. This procedure is called backpropagation. Due to the time component in RNNs the optimization process is then named backpropagation through time.

LSTM

Big problems of classic RNNs are vanishing and exploding gradients as well as the storage of long-term dependencies of information. These are caused by long input sequences: During backpropagation the number of multiplication of weights W_{hh} with themselves increases with the length of the sequence. In order to deal with these issues, gating mechanisms and extra memory states were introduced. They decide which information should be kept or forgotten in each time step and transferred further to the next one. A very widely known example for so-called Gated RNNs is *Long Short-Term Memory (LSTM)* proposed in [29]. Its key feature is the separation of memory into long- and short-term memory. The long-term memory state C_t has the advantage that it is not defined based on matrix multiplications, but only on simple operations like addition and point-wise multiplication. This makes gradient calculations easy as there are no power operations in the formula. Consequently, the risk of exploding or vanishing gradients is much smaller than for standard RNNs.

For explanations, we follow [96, p.352-355]. The LSTM, as shown in Figure 5, consists of an additional *memory cell or state* $C_t \in \mathbb{R}^{n \times h}$ at each time step which is controlled by forget, input and output gates $F_t, I_t, O_t \in \mathbb{R}^{n \times h}$. The state C_t is responsible for long-term memories. Its inputs at time t are the current input $X_t \in \mathbb{R}^{n \times d}$ and the previous hidden

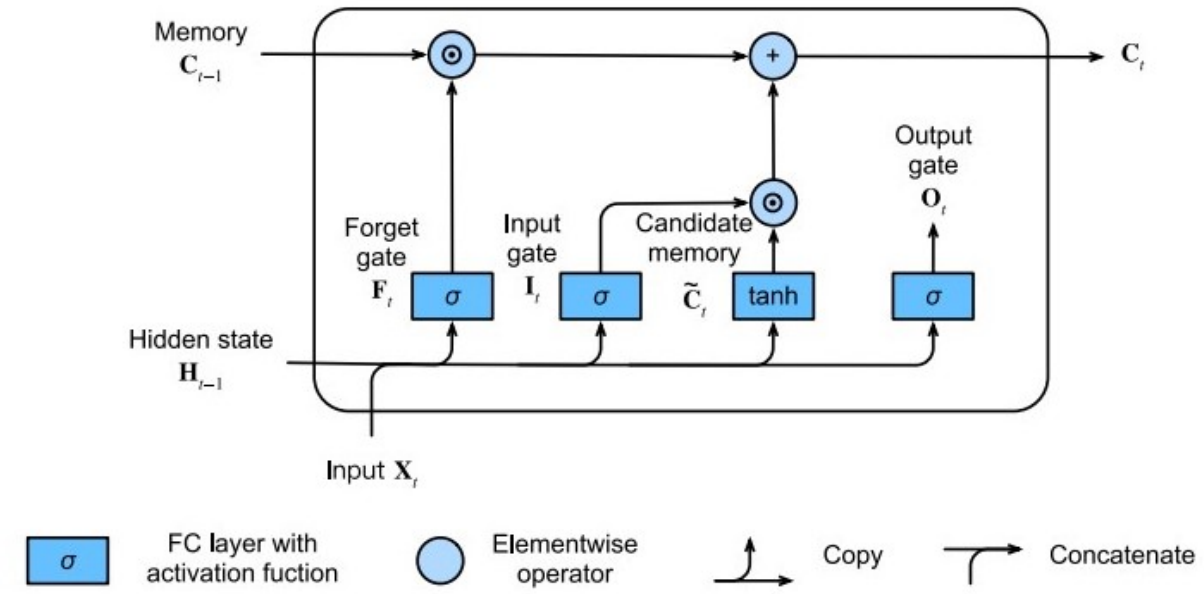


Figure 5: LSTM Architecture. *Source:* [96, p.355]

state $H_{t-1} \in \mathbb{R}^{n \times h}$. In contrast to C_t, the hidden state works as the short-term memory. The gates themselves are fully-connected layers with a sigmoid activation:

$$\begin{aligned} F_t &= \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f), \\ I_t &= \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i), \\ O_t &= \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o), \end{aligned}$$

where $W_{xj} \in \mathbb{R}^{d \times h}$, $W_{hj} \in \mathbb{R}^{h \times h}$ and $b_j \in \mathbb{R}^{1 \times h}$ are weights and biases for $j \in \{f, i, o\}$. The forget gate is used to restrict the information from the previous long-term memory cell C_{t-1} to be taken into account into the current memory cell. In order to set up C_t, a so-called *candidate memory cell* $\tilde{C}_t$ is computed as

$$\tilde{C}_t = \tanh\left(X_t W_{xc} + H_{t-1} W_{hc} + b_c\right),$$

with the same parameter shapes as above. The input gate decides which new information from this candidate memory cell should be stored in C_t:

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t.$$

Based on this cell memory, the hidden state is calculated via

$$H_t = O_T \odot \tanh\left(C_t\right).$$

This architecture allows the model to keep long-term dependencies by setting the forget gate approximately to 1 and the input gate to approximately 0. Whether these dependencies are necessary is determined during the training process. An output gate value of about 1 pushes the current cell memory forward to be processed further in the next steps, whereas a value of about 0 keeps all information in the cell itself.

GRU

Gated Recurrent Units (GRUs), are very similar to LSTMs, yet they are less complex. They were proposed in [10], but we refer to [96, p.345-349] for our explanations. As shown in Figure 6, compared to an LSTM it has only two gates and one hidden state H_t.

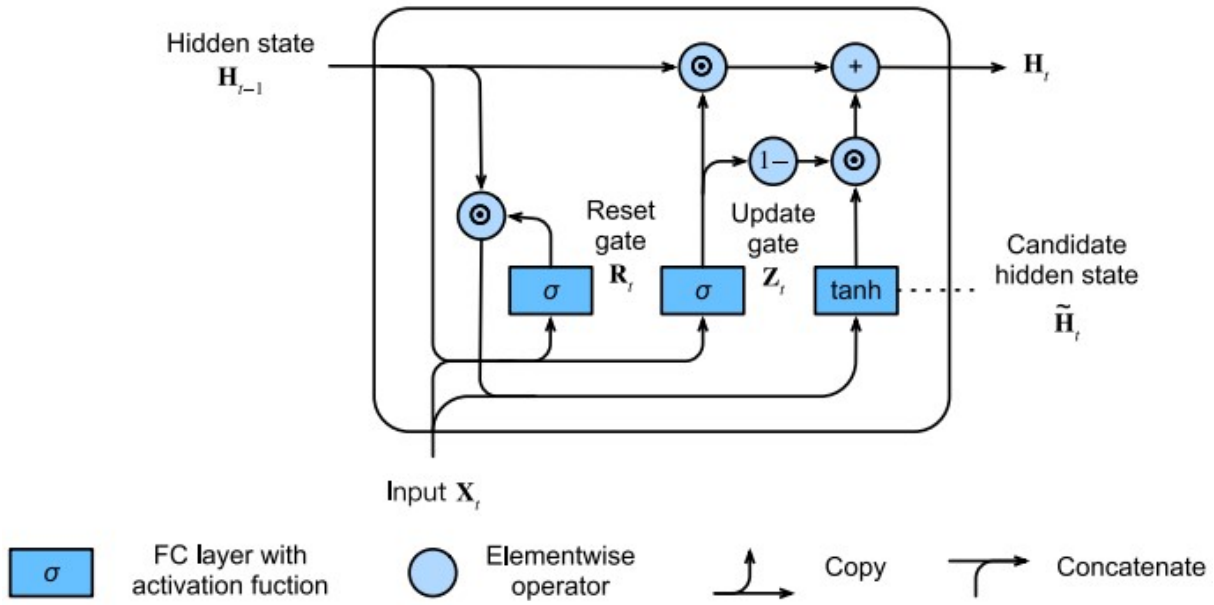


Figure 6: GRU Architecture. *Source:* [96, p.349]

The reset gate R_t can be used to dismiss information in the hidden state; thus, it is responsible for the short-term memory. In contrast to that, the update gate Z_t decides how much old information is kept in the new state, which makes it capture long-term dependencies. The gates are calculated by

$$R_t = \sigma(X_t W_{xr} + H_{t-1} W_{hr} + b_r) \text{ and}$$
$$Z_t = \sigma(X_t W_{xz} + H_{t-1} W_{hz} + b_z)$$

with weights W_{xr}, W_{hr}, W_{xz} and W_{hz} and biases b_r and b_z. Based on the reset gate, a candidate hidden state is given by

$$\tilde{H}_t = \tanh\left(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h\right),$$

where W_{xh} and W_{hh} are weights and b_h is a bias term. The new hidden state then is

$$H_t = Z_t \odot H_{t-1} + (1 - Z_t) \odot \tilde{H}_t.$$

Like an LSTM, a GRU deals with the issues of classical RNNs and lowers the risk of vanishing gradients. This is possible due to performing only simple operations on the hidden state instead of matrix products, which makes backpropagation for H_t much faster. Simultaneously, the ability of storing long-term information is improved.

3.4 Attention-based Models

3.4.1 Attention Mechanism

The longer an input sequence is, the more long-term dependent information has to be stored. This causes problems both in terms of memory capacity and performance. To deal with this issue, so-called *attention mechanisms* were introduced which are able to decide which part of information is relevant at which point of time. Figure 7 shows the structure of a typical attention mechanism.

Following the notation of [96, p.404-407], we have a *query vector* $q \in \mathbb{R}^q$ and *key-value pairs* $(k_i, v_i), i = 1, \ldots, m$ with $k_i \in \mathbb{R}^k$ and $v_i \in \mathbb{R}^v$. With an *(attention) scoring function* $a : \mathbb{R}^q \times \mathbb{R}^k \to \mathbb{R}^m$ for each key a so-called *relevance score* (cf. [23, p.475]) is calculated by

$$e_i = a(q, k_i), \ i = 1, \ldots, m. \tag{3.2}$$

Then, for each score we have the corresponding *attention weight* which is received by applying softmax:

$$\alpha_i = \frac{\exp(e_i)}{\sum_{j=1}^{m} \exp(e_j)}. \tag{3.3}$$

The *attention output* is the following weighted sum:

$$f(q, (k, v)) = \sum_{i=1}^{m} \alpha_i v_i. \tag{3.4}$$

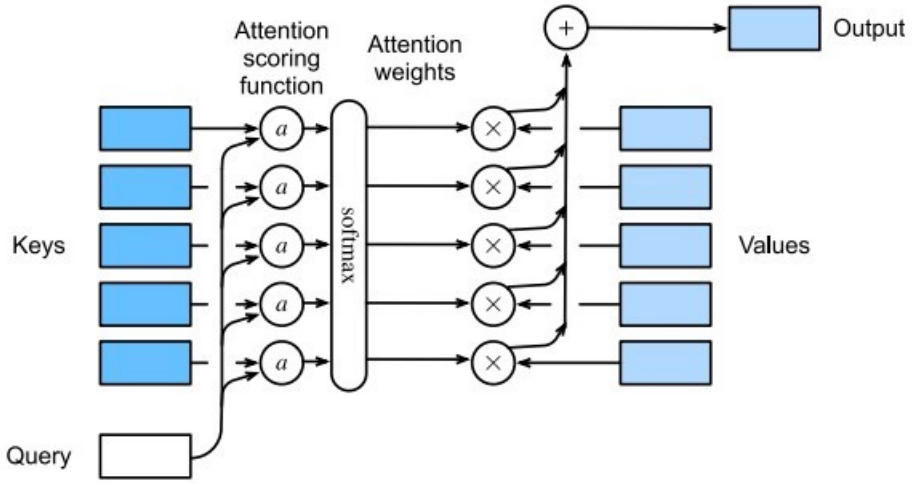


Figure 7: Attention Architecture. *Source:* [96, p.404]

There are several possibilities for the choice of the scoring function, for instance *additive attention* with

$$a(q,k) = w_v^\top \tanh(W_q q + W_k k)$$

is suitable when queries and key vectors have a different length. As usual, W_q, W_k and w_v denote the weights. The *scaled dot-product attention* scoring function

$$a(q,k) = \frac{q^\top k}{\sqrt{d}}.$$

is computationally more efficient for the same vector length d of both q and k.

In the following, we explain some specialized attention architectures. In *multi-head attention* (cf. [96, p.414]), several attention layers or *heads* are applied in parallel as shown in Figure 8. This allows to capture and combine different behaviors of an attention mechanism with the same query, key and value vectors. Before they enter the attention layer, the vectors are fed to different linear projections which are independently learned. After the attention layer, the outputs are concatenated and forwarded to a fully-connected layer.

Self-attention or *intra-attention* (cf. [96, p.418]) is a special case of attention where the keys, values and queries all come from the same set of tokens. This means that each token can attend to itself as well. In contrast to that, *cross-attention* takes its query vectors from another set of tokens. Consequently, *multi-head self-attention* is basically just the combination of multi-head and self-attention.

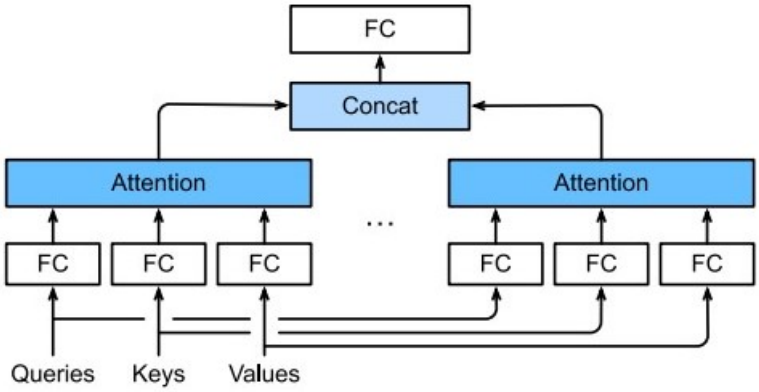


Figure 8: Multi-head attention architecture. *Source:* [96, p.415]

3.4.2 Transformer

The *Transformer* architecture which is solely based on attention was proposed in [80]. As illustrated in Figure 9, the model can be divided in two parts: an encoder on the left and a decoder on the right. The encoder consists of $N = 6$ identical layers with two sub-layers each. These are a multi-head self-attention and a fully-connected feed-forward neural network. Precisely, in the multi-head self-attention layer, keys, queries and values are all outputs of the previous encoder. Both sub-layers have a residual connection and layer normalization employed on top of them. A *residual connection* (cf. [96, p.286f]) gives the model the opportunity to skip a layer if it finds it of no use, i.e. in case a transformation of the input would make the model perform worse. Formally, this means that the layer should learn the identity function $f(x) = x$ for an input x. This could be done with the standard architecture as well, but learning a so-called *residual mapping* $f(x) - x$ is easier. For instance, in case of f being a fully-connected layer, this could be reached by simply setting weights and biases to zero. In Figure 9, the residual connection is depicted by arrows starting before a layer (e.g. "Feed Forward"), going around it and ending in the "Add & Norm" layer. The latter stands for *layer normalization* which is basically normalization of values across the feature dimension (cf. [96, p.426]).

The decoder also contains six layers, but each of them has three sub-layers. The fully-connected feed-forward neural network is basically the same as in the encoder, while the multi-head self-attention layer incorporates masking in order not to allow the model to see outputs that are not yet generated. The additional sub-layer is a multi-head attention layer that takes the output of the encoder as key and value vectors and the output of the previous decoder layer as query vectors. According to the definition from above, this can be considered cross-attention.

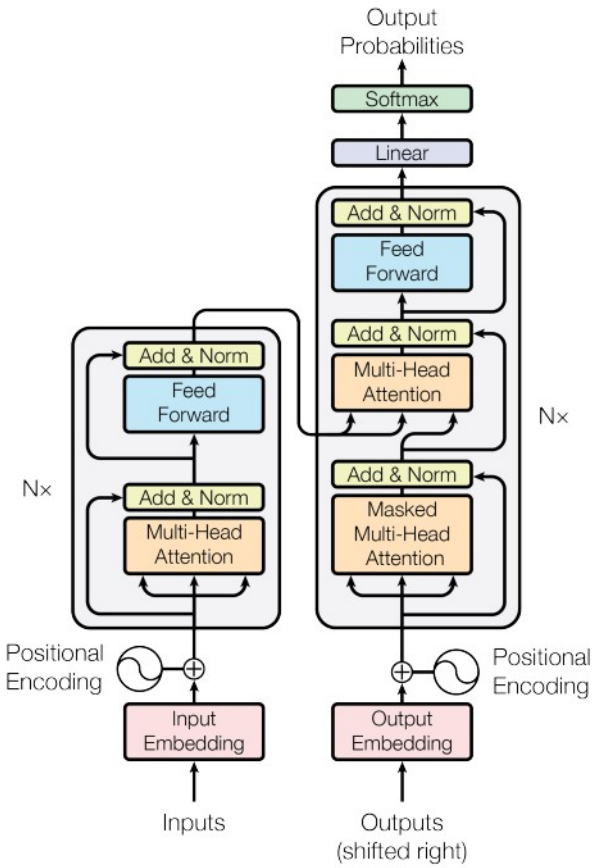


Figure 9: Transformer Architecture. *Source:* [80, p.3]

3.4.3 BERT

Bidirectional Encoder Representations from Transformers (BERT) were introduced in [15]. Their basic architecture is a multi-layer bidirectional Transformer encoder. This means that for the BERT-BASE model, it consists of twelve transformer encoder blocks with hidden size 768 and twelve self-attention heads. There also exists a larger version of BERT, called BERT-LARGE.

In general, BERT is a language model, i.e. a model that is trained to understand how a language is structured and which vocabulary is used in which way. In order to reach this goal, there are two phases: During the pretraining phase, the model learns the structure of the language and the vocabulary in general. This knowledge is used on a certain topic in the fine-tuning phase, where the model learns a specific task and certain terminology.

BERT was pretrained on unlabeled data sets (BooksCorpus, see [99], and English Wikipedia) of 3,300M words with two tasks: Task one was *Masked Language Modelling* which is about predicting randomly masked words within a text. For this task, 15% of the tokens in each sentence were randomly selected to be replaced by a `[MASK]` token. In order to keep pretraining and fine-tuning aligned, it was not exactly 15% masked tokens, but only 80% of the chosen tokens were really masked, while ten percent each were substituted by a random token or not changed at all. BERT's specialty is to work in a bidirectional way, i.e. it is able to condition its prediction for the masked word on the context on both sides. This allows BERT to better deal with different meanings of words as more information is captured. The other task was *Next Sentence Prediction* in which the model is given a pair of sentences that truly appear one after another in half of the cases and that are randomly combined in the other half. Then, the model has to tell whether the order is really given like that or not.

In order to use BERT on texts about certain topics, it is necessary to fine-tune it with labeled data. This means that the model is initialized with the pretrained parameters and then learns task-specific weights for an extra layer. Fine-tuning BERT can be done for several tasks, like question answering, token and sequence classification.

An essential part of the application of BERT during fine-tuning is the preparation of word sequences as inputs. Depending on the task, there is either one sequence on its own or there is a pair of them, e.g. question and answer. First, words are tokenized using the vocabulary determined by the pretrained language model. Then, the tokens are turned into numbers by embeddings, in this case WordPiece embeddings [86]. At the beginning of each sequence, a so-called *classification token* `[CLS]` is added. In case of pairs of sequences, the *separator token* `[SEP]` is added between them and at the end, otherwise it is put only at the end. BERT is used to always receive inputs of the same length which is called the *maximum sequence length*. The maximum that can be chosen is 512; smaller values make sense in case of shorter samples. Inputs longer than the given maximum sequence length are truncated, while shorter ones are filled with so-called padding tokens. It is marked which tokens are real and which are there due to padding. In case of sequence pairs, it is also marked to which sequence a token belongs. When all this is done, BERT can be fine-tuned on the data.

3.5 Graph-based Architectures

Based on the trees introduced in Section 2.3, we now present models that work with the information that can be extracted from them.

3.5.1 Graph Convolutional Networks (GCNs)

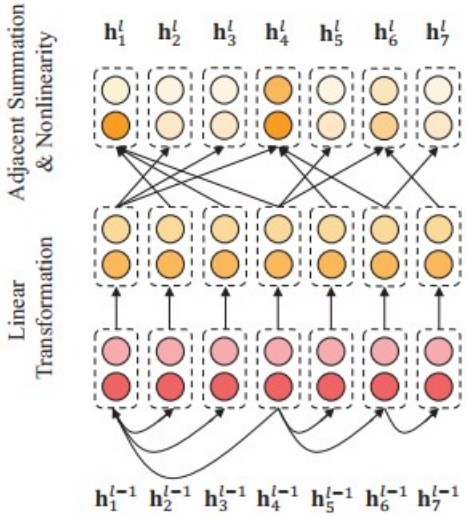


Figure 10: A single GCN layer. *Source:* [97, p.4569]

Graph Convolutional Networks (GCNs) were introduced in [40] and consist of one kind of layer which is repeated several times. Such a layer is illustrated in Figure 10. For an undirected graph $\mathcal{G}$, an adjacency matrix A indicates which nodes are connected to each other. In each layer l, the activation matrix is updated by

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}H^{(l)}W^{(l)}\right) \tag{3.5}$$

with representations of the input nodes X as initialization $H^{(0)}$. Moreover, σ denotes an activation function, $W^{(l)}$ the weights, $\tilde{A} = A + I_N$ and $\tilde{D} = \sum_j \tilde{A}_{jj}$.

3.5.2 Graph Attention Networks (GATs)

The idea of *Graph Attention Networks (GATs)* was developed in [81]. They are constructed by stacking several *graph attentional layers* onto each other. A single layer consists of the following calculation steps: A set $h = \{\overrightarrow{h_1}, \ldots, \overrightarrow{h_N}\}$ of N nodes works as inputs. Self-attention with relevance scores $e_{ij} = a(W\overrightarrow{h_i}, W\overrightarrow{h_j})$ is applied on those nodes $j \in \mathcal{N}_i$ which are the first-order neighbors of node i (including $i = j$). Hereby, the score function $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ is a dense layer, and W are weights. Attention weights α_{ij} are computed according to Equation (3.3) as softmax of e_{ij}. Instead of using the usual attention output formula from Equation (3.4), multi-head attention is applied in order to

stabilize the learning process. This yields the following outputs of the graph attentional layer:

$$\overrightarrow{h}'_i = ||_{k=1}^{K} \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k W^k \overrightarrow{h_j} \right). \tag{3.6}$$

Here, K attention heads are used and $||$ symbolizes concatenation. As the final graph attentional layer usually is responsible for predictions, a special formula based on averaging instead of concatenation is proposed by

$$\overrightarrow{h_i'} = \sigma \left(\frac{1}{K} \sum_{k=1}^{K} \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k W^k \overrightarrow{h_j} \right). \tag{3.7}$$

4 Meta-Analysis of Approaches

The main part of this thesis concerns itself with a detailed comparison of existing approaches for aspect-based sentiment analysis methods, both ATSC and ATE+ATSC. Due to the vast amount of methods and the ongoing developments in this field, we make no claim to completeness. Out of this collection of methods we are going to identify a subgroup of approaches which we are going to analyze practically in Section 5.

In order to present the selected approaches in a structured manner, we created two big groups based on the task, one for ATSC only and another one for the combination of ATE and ATSC. Within these groups, we made up some categories to which the methods are assigned. Some of them may belong to a combination of several subcategories which would make a nice overview quite difficult. Thus, we assigned each approach to that category about which we thought it had the highest impact on. Within the categories, we usually order the approaches according to their publication date, beginning with the oldest.

4.1 ATSC Methods

Here, we explain those methods that are classifying sentiments for given aspects. Figure 11 gives an overview about those for which accuracy on SemEval-14 Restaurant data has been reported. It illustrates that almost all models have been developed in the last six years. A clear change is visible at the end of 2018 when BERT and the transformer architecture were introduced. This increased the performance results of all following models that were build on top of BERT. Since the equivalent graphs for other data sets and other metrics do not provide more information on the general development, they can be found in the appendix (see Figures 62, 63 and 64). All these reported accuracies and F1 Macro Scores for ATSC methods are listed in Tables 3 and 4. We also marked those model in Figure 11 which we will use later in our practical part: RGAT-BERT [4] as it is the latest approach, MGATN [18] as it is the best of the "old" methods, LCF-ATEPC [93] as one of its variants is the best on this data set and CapsNet-BERT [34] for the group of capsule networks.

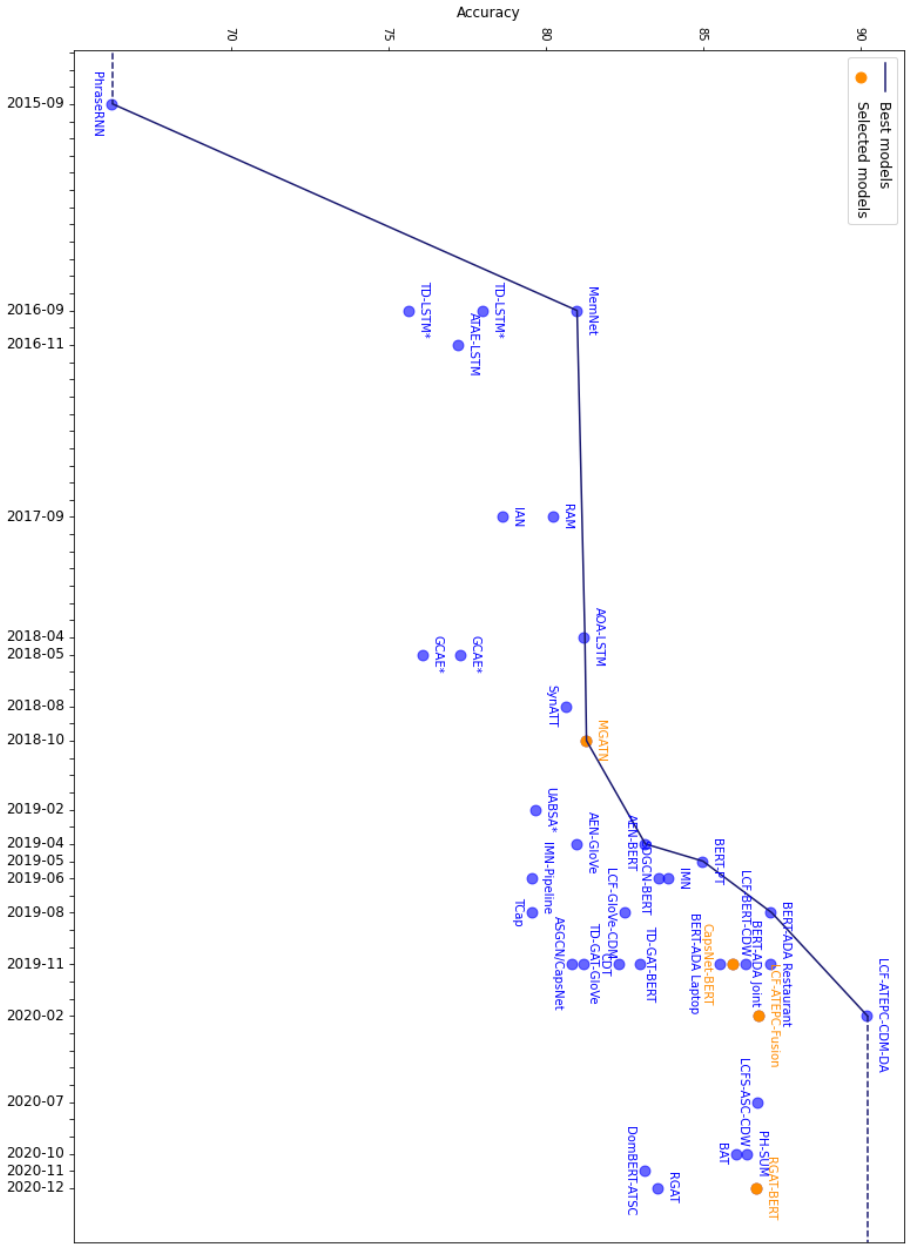


Figure 11: Development of reported ATSC Accuracy on SemEval-14 Restaurants. An asterisk indicates that the corresponding values are not taken from the original papers.

ACCURACY		**Data Set**	SemEval-14		ARTS		MAMS	Dong
		Domain	Restaurant	Laptop	Restaurant	Laptop	Restaurant	Twitter
Model Category	**Model**	**Value Calculation**			ARS	ARS	mean of 5 runs	
RNN-based	TC-LSTM							71.5
	TD-LSTM		75.63 [98] - 78.00 [7]	68.13 [76] - 71.83 [18]	30.18	22.57	74.60	70.8
	ATAE-LSTM		77.2	68.7	14.64	9.87	77.05	69.58 [49]
	IAN		78.6	72.1			76.60	72.50 [97]
	RAM		80.23	74.49				69.36
	AOA-LSTM	best of 10 runs	81.2	74.5			**77.26**	72.30 [97]
		mean of 10 runs	79.7	72.6				
	MGATN		**81.25**	**75.39**				**72.54**
Attention-based	MemNet (4)		80.95	72.37	21.52	16.93	64.57	66.91 [7] - 71.48 [97]
	AEN-GloVe		80.98	73.51			66.72	72.83
	AEN-BERT		**83.12**	**79.93**				**74.71**
BERT-based	P-SUM	mean of 9 runs	86.30	**79.55**				
	H-SUM	mean of 9 runs	**86.37**	79.40				
	BAT (3)	mean of 9 runs	86.03	79.35				
Extra data	BERT-PT	mean of 9 runs	84.95	78.07	**59.29**	**53.29**		
	BERT-ADA Laptop	mean of 9 runs	85.51	**79.19**				
	BERT-ADA Restaurant	mean of 9 runs	**87.14**	78.60				
	BERT-ADA Joint	mean of 9 runs	86.35	78.96				
	DomBERT-ATSC	mean of 10 runs	83.14	76.72				
LCF-based	LCF-GloVe-CDM	best	82.5	76.02				72.25
	LCF-BERT-CDW	best	87.14	82.45				**77.31**
	LCFS-ASC-CDW		86.71	80.52				
	LCF-ATEPC-Fusion		86.77	80.97				76.7
	LCF-ATEPC-CDM-DA		**90.18**	**83.02**				
Graph-based	AdaRNN							66.3
	PhraseRNN (1)		66.20					
	SynATT	mean of 5 runs	80.63	71.94				
	SDGCN-BERT		83.57	**81.35**				
	CDT		82.30	77.19			80.70 [4]	74.66
	ASGCN-DT	mean of 3 runs	80.86	74.14	24.73	19.91		71.53
	ASGCN-DG	mean of 3 runs	80.77	75.55				72.15
	TD-GAT-GloVe (1)		81.2	74.0				
	TD-GAT-BERT (1)		83.0	80.1				
	RGAT		83.55	78.02			81.75 [4]	75.36
	RGAT-BERT		**86.68**	80.94			**84.52** [4]	**76.28**
Capsule Networks	CapsNet (2)	mean of 5 runs	80.79				79.78	
	CapsNet-BERT (2)	mean of 5 runs	**85.93**		55.36	25.86	83.39	
	TCap (5)	mean of 5 runs	79.55	73.87				
	GCAE		76.09 [8] - 77.28 [34]	68.72 [8]				
Pipeline Models	IMN-Pipeline (ATSC)	mean of 5 runs	79.56	72.29				
	SPAN (ATSC)			**81.39**				
Joint Models	IMN (ATSC)	mean of 5 runs	83.89	75.36				
Collapsed Models	UABSA (ATSC)		79.68 [63]	72.30 [25]				

Table 3: Reported Accuracy for 3-class-ATSC. Values for MAMS and ARTS are from those papers; all other sources are marked accordingly. Bold printed values indicate the best model of one category, underlined ones the best performance on the corresponding data set. Remarks: (1) number of layers taken as hyperparameter, (2) classification assumed to be 3-class, (3) restaurant data assumed to be SemEval-14 Restaurant, (4) number of hops taken as hyperparameter, (5) data sets of the auxiliary task taken as hyperparameter.

F1 MACRO		**Data Set**	SemEval-14		MAMS	Dong
		Domain	Restaurant	Laptop	Restaurant	Twitter
Model Category	**Model**	**Value Calculation**			mean of 5 runs	
RNN-based	TC-LSTM					69.5
	TD-LSTM		64.16 [49] - 66.73 [7]	62.28 [49] - 68.43 [7]		69.0
	ATAE-LSTM		64.95 [49] - 66.36 [8]	62.45 [49] - 63.24 [8]		56.72 [49]
	IAN		67.71 [8] - 70.09 [97]	63.72 [8] - 67.38 [97]		**70.81** [97]
	RAM		70.80	71.35		67.30
	AOA-LSTM		70.42 [97]	67.52 [97]		70.20 [97]
	MGATN		**71.94**	**72.47**		**70.81**
Attention-based	MemNet		65.83 [49] - 69.64 [97]	62.60 [8] - 65.17 [97]		66.91 [76]
	AEN-GloVe		72.14	69.04		69.81
	AEN-BERT		**73.76**	**76.31**		**73.13**
BERT-based	P-SUM	mean of 9 runs	**79.68**	**76.81**		
	H-SUM	mean of 9 runs	79.67	76.52		
	BAT (2)	mean of 9 runs	79.24	76.50		
Extra data	BERT-PT	mean of 9 runs	76.96	**75.08**		
	BERT-ADA Laptop	mean of 9 runs	78.09	74.18		
	BERT-ADA Restaurant	mean of 9 runs	**80.05**	74.09		
	BERT-ADA Joint	mean of 9 runs	78.89	74.18		
	DomBERT-ATSC	mean of 10 runs	75.00	73.46		
LCF-based	LCF-GloVe-CDM	best	73.92	70.58		70.92
	LCF-BERT-CDW	best	81.74	79.59		**75.78**
	LCFS-ASC-CDW		80.31	77.13		
	LCF-ATEPC-Fusion		80.54	77.86		74.54
	LCF-ATEPC-CDM-DA		**85.88**	**79.84**		
Graph-based	AdaRNN					65.9
	PhraseRNN (1)		59.32 [4]			
	SynATT	mean of 5 runs	71.32	69.23		
	SDGCN-BERT		76.47	**78.34**		
	CDT (3)		74.02	72.99	79.79 [4]	73.66
	ASGCN-DT	mean of 3 runs	72.19	69.24		69.68
	ASGCN-DG	mean of 3 runs	72.02	71.05		70.40
	RGAT		75.99	74.00	80.87 [4]	74.15
	RGAT-BERT		**80.92**	78.20	**83.74** [4]	**75.25**
Capsule Networks	TCap (4)	mean of 5 runs	71.41	70.10		
	GCAE		63.29 [8]	63.32 [8]		
Pipeline Models	IMN-Pipeline (ATSC)	mean of 5 runs	69.59	68.12		
Joint Models	IMN (ATSC)	mean of 5 runs	75.66	72.02		
Collapsed Models	UABSA		68.38 [63]	68.24 [25]		

Table 4: Reported F1 Macro Scores for 3-class-ATSC. Values for MAMS and ARTS are from those papers; all other sources are marked accordingly. Bold printed values indicate the best model of one category, underlined ones the best performance on the corresponding data set. Remarks: (1) number of layers taken as hyperparameter, (2) restaurant data assumed to be SemEval-14 Restaurant, (3) according to implementation, values are F1 Macro Scores, (4) data sets of the auxiliary task taken as hyperparameter.

4.1.1 CNN-based models

The first model to be presented is the only one in our thesis to be based on CNNs. The so-called *Gated Convolutional Network with Aspect Embeddings (GCAE or sometimes GatedCNN)* was proposed in [91] and is shown in Figure 12.

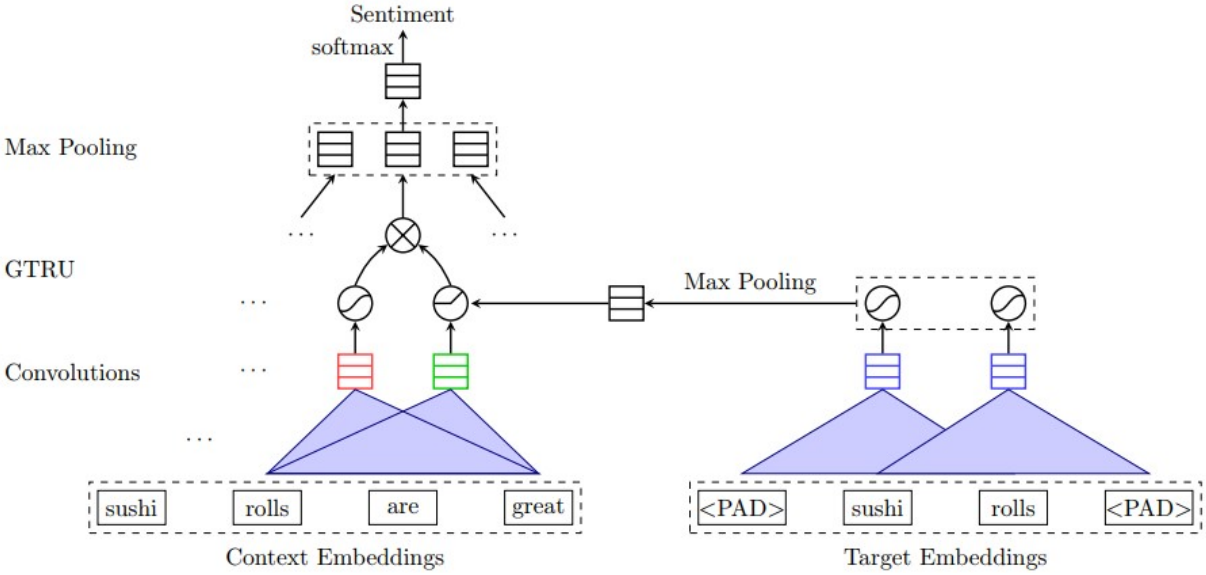


Figure 12: GCAE Architecture for ATSC. *Source:* [91, p.5]

The modeling procedure of a GCAE is the following: First, the embedded words $X = [v_1, \ldots, v_L]$ of a sentence of length L are fed to convolutional layers, where n-gram features $c_i, i = 1, \ldots, L$ are computed position-wise. So-called *Gated Tanh-ReLU Units (GTRU)* are used for their calculation:

$$
\begin{aligned}
c_i &= s_i \times a_i \text{ with} \\
s_i &= \tanh(X_{i:i+K} * W_s + b_s) \\
a_i &= ReLU(X_{i;i+K} * W_a + V_a v_a + b_a).
\end{aligned}
$$

The convolution is denoted by $*$, W_s, W_a are weights and b_s, b_a biases. The representation of the aspect term v_a is retrieved from a CNN over aspect terms and a max pooling layer. The new features are created based on aspect features a_i and sentiment features s_i. They are then forwarded to a max pooling, a fully-connected and a softmax layer. Model training is done by the minimization of the cross-entropy loss.

GCAE is also applicable on ACSC with a slightly different architecture for which we refer the reader to [91] as this would go beyond the scope of our work.

4.1.2 RNN-based Models

Before Transformer and BERT were designed, RNNs and especially LSTMs were very popular to use for a broad variety of tasks, for instance ATSC. Thus, we present these models here, although they do not reach the performance of later approaches, as shown in Figure 11.

TD-LSTM and TC-LSTM

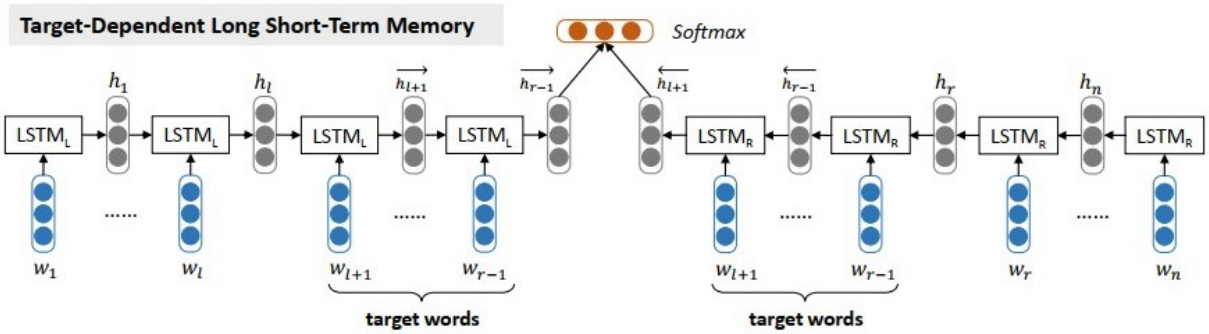


Figure 13: TD-LSTM Architecture. *Source:* [78, p.2]

Two of the first approaches, *target-dependent* and *target-connected LSTMs*, TD-LSTM and TC-LSTM, were introduced in [78]. The TD-LSTM consists of two uni-directional LSTMs which are shown in Figure 13: $LSTM_L$ takes the left part of the sentence including the target as input, while $LSTM_R$ takes the right part plus the target. The former processes the input from left to right, the latter from right to left. Their hidden layers are then concatenated and fed into a softmax layer for classification.

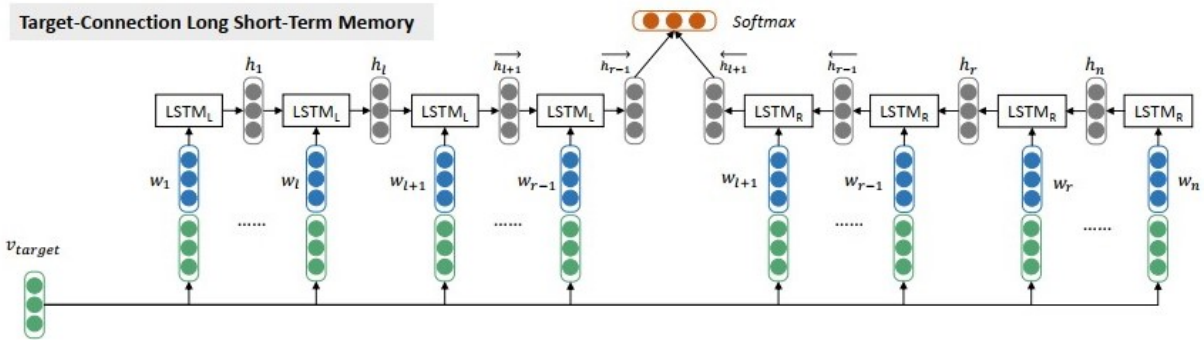


Figure 14: TC-LSTM Architecture. *Source:* [78, p.3]

The TC-LSTM is based on the TD-LSTM as one can see in Figure 14. Here, v_{target} is calculated as the average of words vectors belonging to the target. Concatenated position-wise with the embedding of each word, this is the input of the TD-LSTM. Both models are trained by minimizing cross-entropy loss.

ATAE-LSTM

Also in [83], an approach based on LSTMs was proposed, in this case combined with an attention mechanism. Its name ATAE-LSTM stands for *Attention-based LSTM with Aspect Embedding* and its architecture can be found in Figure 15.

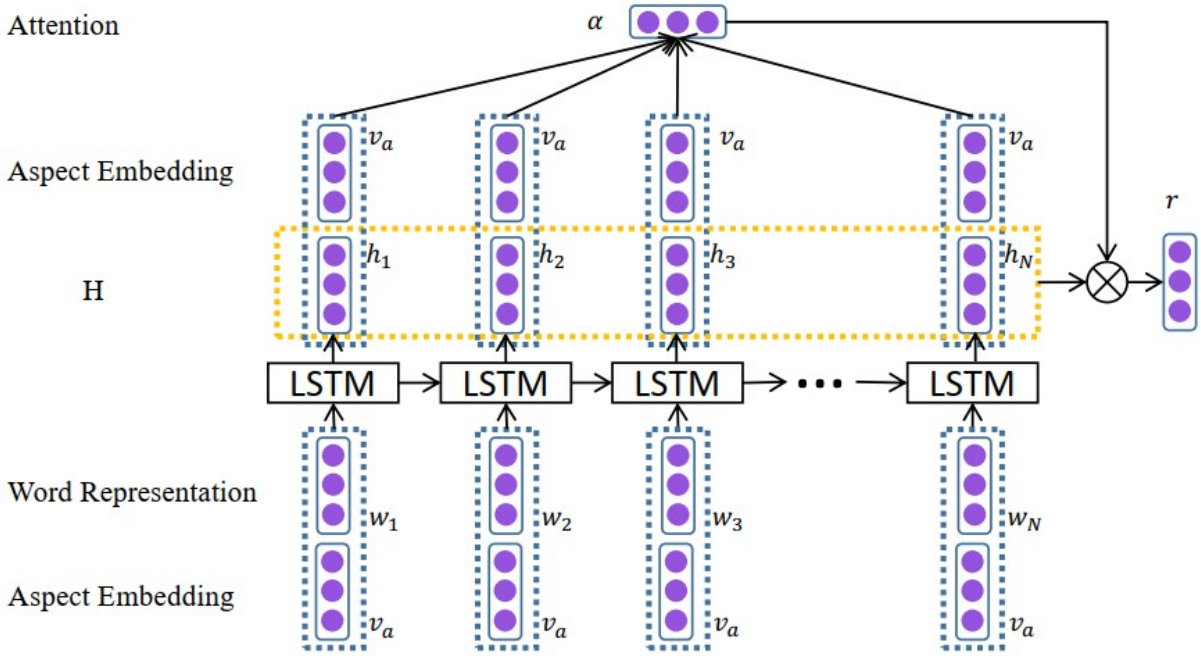


Figure 15: ATAE-LSTM Architecture. *Source:* [83, p.610]

Similar to TC-LSTM, we feed not only word representations $w_1, \ldots, w_N$ with sequence length N to an LSTM with d hidden layers, but also aspect embeddings v_a. They are appended to the word representations. The outputs of the LSTM, i.e. the hidden states $H \in \mathbb{R}^{d \times N}$, are then forwarded to an attention mechanism which calculates relevance scores and attention weights. They made the following modifications of Equations (3.2) and (3.3):

$$M = \tanh\left(\begin{bmatrix} W_h H \\ W_v v_a \bigotimes e_N \end{bmatrix}\right) \text{ and } \alpha = softmax(w^\top M),$$

where W_h, W_v and w are weights and e_N is a vector of length N filled with ones. The $\bigotimes$ operator produces a vector of length N consisting of $W_v v_a$. The attention output is calculated according to Equation (3.4) as $r = H\alpha$. The final sentence representation is $h^* = \tanh(W_p r + W_x h_N)$ with weights W_p and W_x. It is fed to a linear and a softmax layer for classification. The model it trained by minimizing cross-entropy loss with L2 regularization. It can also be used for the ACSC task.

IAN

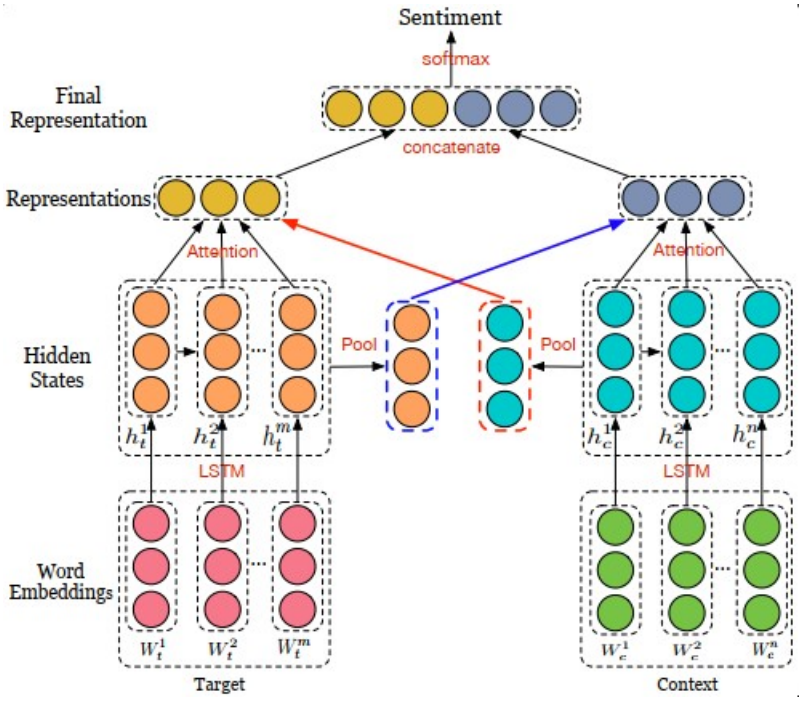


Figure 16: IAN Architecture. *Source:* [55, p.2]

So-called *Interaction Attention Networks* were proposed in [55]. As depicted in Figure 16, context and aspect terms are separately encoded with word embeddings and fed to separate LSTMs. Here, with context we mean the whole sentence including the aspect term. Averaging over the final hidden states of the LSTMs leads to representations c_{avg} and t_{avg}, respectively. On them, an attention mechanism is applied, where the relevance scores from Equation (3.2) for the context are calculated as

$$\gamma(h_c^i, t_{avg}) = \tanh(h_c^i W_a t_{avg}^\top + b_a)$$

with weights W_a and bias b_a. For the aspect term, the relevance score is $\gamma(h_t^i, c_{avg})$. The

corresponding attention weights for context α_i and aspect term β_i are received following Equation (3.3). Equation (3.4) leads to the respective attention outputs c_r and t_r which are then concatenated. For classification, a linear layer with tanh-activation and a softmax are applied. The model is optimized using cross-entropy loss with L2 regularization.

RAM

RAM, a *Recurrent Attention Network on Memory* in [7], uses both LSTMs, GRUs and Attention. As shown in Figure 17, it consists of five modules.

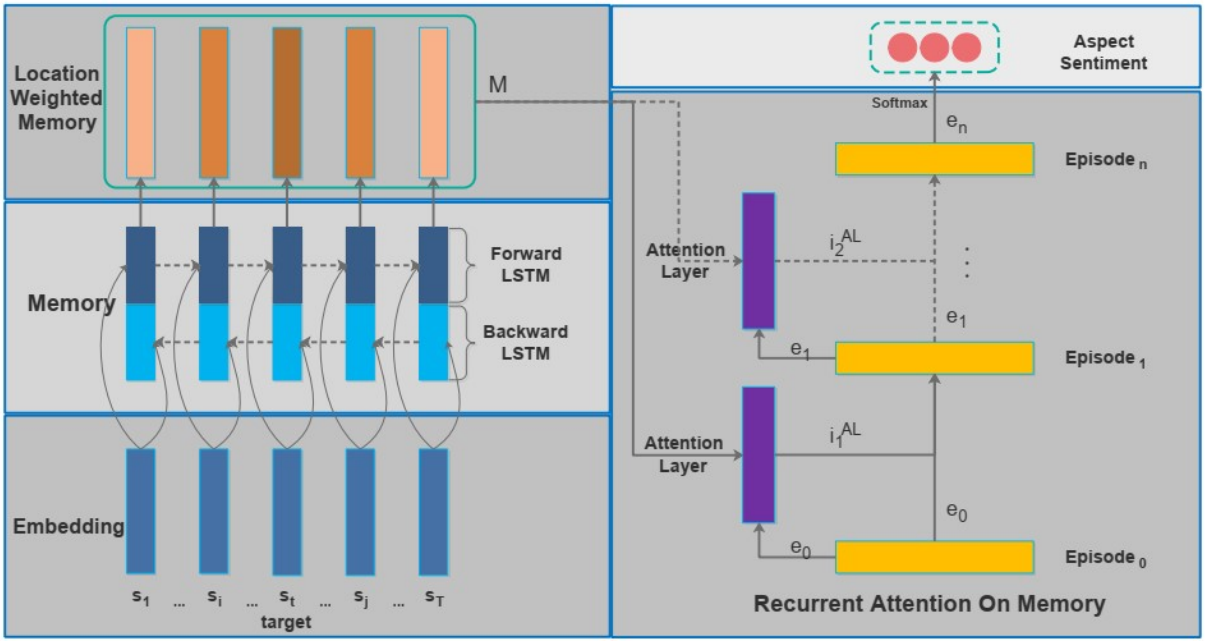


Figure 17: RAM Architecture. *Source:* [7, p.454]

In the first step, words are turned into numerical representations using GloVe embeddings (cf. [62]). These embeddings are then fed into a bidirectional LSTM with two layers. For each word t, the last hidden states $\overrightarrow{h_t^2}$ and $\overleftarrow{h_t^2}$ are concatenated to $m_t^* = (\overrightarrow{h_t^2}, \overleftarrow{h_t^2})$ which builds the memory $M^* = \{m_1^*, \ldots, m_T^*\}$ for the whole sequence with length T. This memory is modified in the third module in order to assign a specific weight to each word with respect to an aspect term at position τ. The idea is to assign a higher weight for words closer to the aspect term. This weight is calculated by $w_t = 1 - \frac{|t-\tau|}{t_{max}}$, where t indicates the position of the word and t_{max} the truncation length of the sequence. Additionally, the relative offset between the words and the aspect term is measured by $u_t = \frac{t-\tau}{t_{max}}$. In case of multi-word aspect terms, τ is the index of the first or last word depending on whether

the word t is on the left or right side of the aspect, respectively. Then the final position-weighted memory for an aspect term is $M = \{m_1, \ldots, m_T\}$ with $m_t = (w_t * m_t^*, u_t)$. This memory is forwarded to n attention layers that work as described in Section 3.4.1. The number of layers n was subject to additional experiments of the authors for which they reported different results.

The attention layers interact with $n + 1$ GRU layers which are referred to as episodes $e_0, \ldots, e_n$ in the following way: The first GRU episode e_0 is initialized with zeros. Based on it, the attention relevance score from Equation (3.2) for the first attention layer $j = 1$ is calculated as

$$g_t^j = W_j^{AL}(m_t, e_{j-1}[, v_\tau]) + b_j^{AL}$$

for word positions t, weights W_j^{AL} and bias b_j^{AL}. Note that we differ from the authors' notation here in order not to be confused with word positions t and layer or episode numbers $j = 1, \ldots, n$. The optional input variable v_τ is used when the aspect term refers to an aspect of a product, but not if it is a person. In case of multi-word aspects, v_τ is received by averaging over the embeddings of all words. Based on the attention relevance score g_t^j, attention weights and outputs i_j^{AL} are calculated according to Equations (3.3) and (3.4), respectively. Each attention output i_j^{AL} is then fed to the next episode GRU layer e_j which works as explained in Section 3.3. In the last module, the outputs of the final GRU e_n are the inputs of a softmax layer for sentiment classification. The model is trained by minimizing cross-entropy loss with L2 regularization.

AOA-LSTM

AOA-LSTM is short for *Attention-over-Attention LSTM* which was proposed in [32]. As Figure 18 shows, the model consists of four parts, namely an embedding layer, a Bi-LSTM, an attention-over-attention module and a prediction layer.

In the embedding layer, the whole sentence and the aspect term are embedded separately. Each set of embeddings is fed into a bidirectional LSTM which outputs the concatenation of hidden states of each direction. The hidden target states h_t and the hidden sentence states h_s are used to calculate a pair-wise interaction matrix $I = h_s h_t^\top$. Each entry of this matrix indicates the correlation between the target and a word of the sentence. Target-to-sentence attention weights α and sentence-to-target attention weights β are received by applying softmax column- and row-wise, respectively. Averaging β column-wise yields target-level attention weights $\bar{\beta}$. They are used in order to calculate sentence-level attention weights $\gamma = \alpha\bar{\beta}^\top$. Based on this, the final sentence representation is $r = h_s^\top \gamma$. A linear layer followed by a softmax turns this representation into a probability

for each sentiment class. This model is trained by minimizing a cross-entropy loss with L2-regularization.

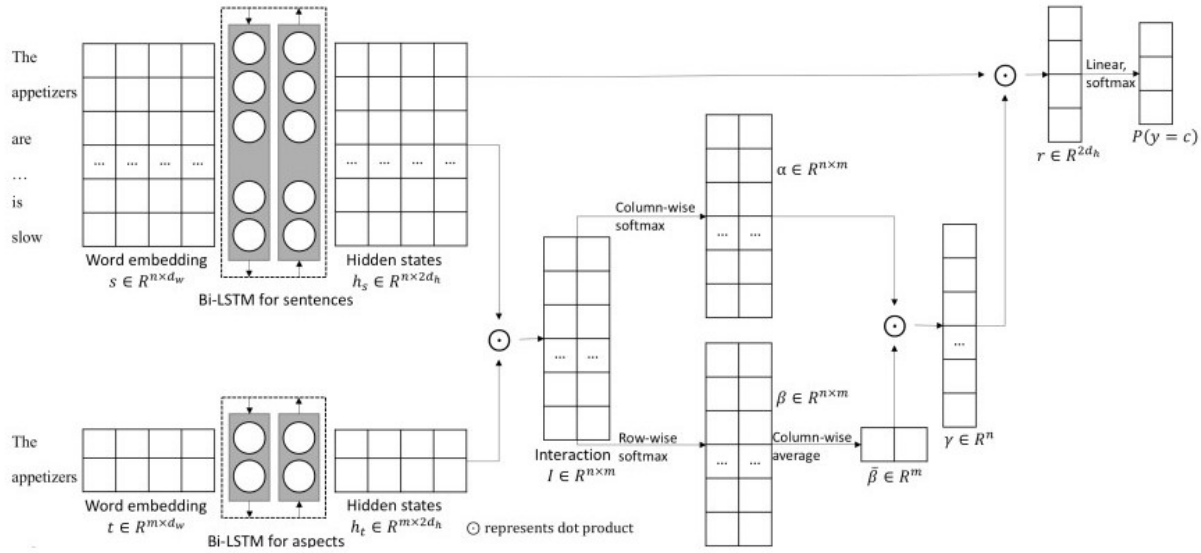


Figure 18: Attention-over-Attention LSTM Architecture. *Source:* [32, p.3]

MGATN

A *multi-grained attention network (MGATN)* was proposed in [18]. As Figure 19 shows, it consists of four layers. In the first one, GloVe embeddings (cf. [62]) are applied on both the whole sentence and the aspect term. These are then separately fed to two BiLSTMs yielding contextual outputs H^* for the sentence and Q for the aspect term. This is the same procedure as in the AOA-LSTM before. Differently to that, weights indicating the distance between a word t and the aspect term are calculated as

$$w_t = 1 - \frac{l}{N - M + 1},$$

where N is the sentence length, M the aspect term length and l the absolute distance between a word t and the aspect ($l = 0$ if the word is part of the aspect). These weights modify the contextual outputs in the following way: $H = [H_1^* w_1, \ldots, H_N^* w_N]$.

The specialty of MGATN is the *multi-grained attention* which also takes the interaction between aspects into account. One part of it, here called *coarse-grained attention*, is what is usually known as attention. It works bidirectionally as described in Section 3.4.1, yielding C-Aspect2Context and C-Context2Aspect attention (where the C stands for "coarse").

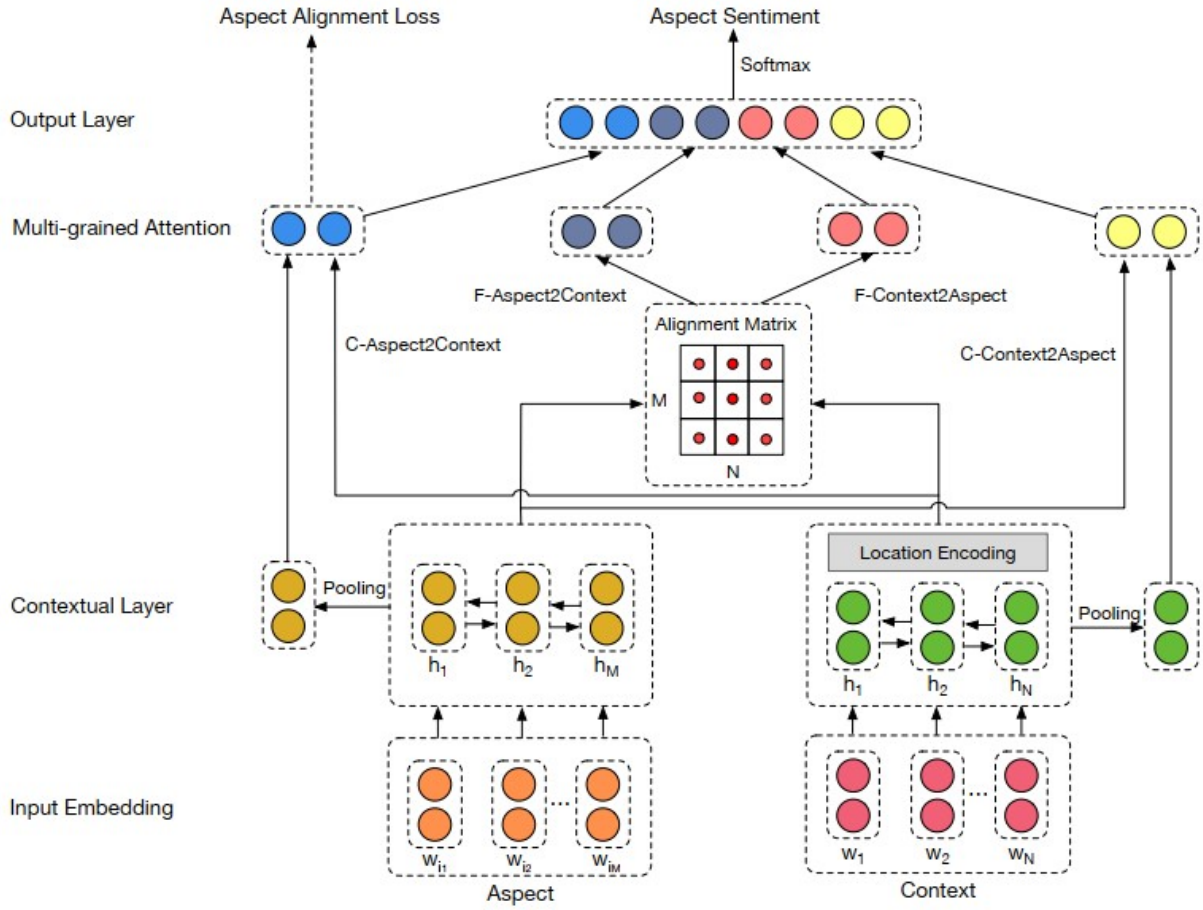


Figure 19: MGATN Architecture. *Source:* [18, p.3436]

With Q_{avg} and H_{avg} obtained by average pooling over Q and H, respectively, the relevance scores are

$$s_{ca}(Q_{avg}, H_i) = Q_{avg} W_{ca} H_i$$
$$s_{cc}(H_{avg}, Q_i) = H_{avg} W_{cc} Q_i$$

with weights W_{ca} and W_{cc}. Here, cc is short for "coarse-context" and ca for "coarse-aspect". The attention weights a_i^{ca} and a_i^{cc} are received by applying softmax to these equations. The attention outputs of context and aspects are

$$m^{ca} = \sum_{i=1}^{N} a_i^{ca} H_i \text{ and } m^{cc} = \sum_{i=1}^{M} a_i^{cc} Q_i.$$

Fine-grained attention deals with word-level interactions between both aspect and context

words. This also includes relationships within the aspect terms. To do so, an alignment matrix measuring the similarity between a context word i and an aspect word j is defined as

$$U_{ij} = W_u[H_i; Q_j; H_i * Q_j]$$

with weights W_u and element-wise multiplication $*$. Like before, attention vectors for both directions are calculated. F-Aspect2Context attention identifies the context word which is most similar to one of the aspect words and thus might be crucial for sentiment classification. The attention weights a_i^{fa} are computed as the softmax of relevance scores $s_i^{fa} = max(U_{i,:})$, where fa stands for "fine-aspect". The attention output then is $m^{fa} = \sum_{i=1}^{N} a_i^{fa} H_i$. F-Context2Aspect attention does the opposite, i.e. identifying aspect words that are most important to a certain context word. The attention calculations are the following:

$$\begin{aligned} a_{ij}^{fc} &= \frac{\exp(U_i j)}{\sum_{k=1}^{M} \exp(U_{ik}} \\ q_i^{fc} &= \sum_{j=1}^{M} a_{ij}^{fc} Q_j \\ m^{fc} &= avgpooling\ ([q_1^{fc}, \ldots, q_N^{fc}]). \end{aligned}$$

The last line denotes the attention output with fc meaning "fine-context". It is concatenated with all the other attention vectors to the final representation $m = [m^{ca}; m^{cc}; m^{fa}; m^{fc}]$. This is fed to a linear layer and a softmax of which the output is denoted as $p \in \mathbb{R}^C$ with C sentiment classes.

In order to further include the relationship between aspects, an aspect alignment loss is defined based on the C-Aspect2Context attention weights. Comparing aspects with each other may reveal the most important words for sentiment classification which then could have assigned a higher attention value with respect to a certain aspect. To do so, the distance between aspects a_i and a_j is estimated by $d_{ij} = \sigma(W_d[Q_i; Q_j; Q_i Q_j])$ with weights W_d and sigmoid function σ. Then, the aspect alignment loss is defined as

$$\mathcal{L}_{align} = -\sum_{i=1}^{M-1} \sum_{j=i+1, y_i \neq y_j}^{M} \sum_{k=1}^{N} d_{ij} (a_{ik}^{ca} - a_{jk}^{ca})^2,$$

where y_i, y_j are the true labels of a_i, a_j and a_{ik}^{ca}, a_{jk}^{ca} are the attention weights of context

word k with respect to a_i, a_j. The overall loss for training is calculated by

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(p_i) + \beta \mathcal{L}_{align} + \lambda ||\Theta||^2$$

with hyperparameters $\lambda, \beta \geq 0$.

4.1.3 Attention-based Models

Although we already mentioned some models that include attention mechanisms in the previous section, we now create a whole section of attention-based models. The reason for this is that the models from above are all based on RNNs, whereas the following models do not rely on them as basis.

MemNet

The original idea of *Memory Networks* came from [84], but was taken up and combined with attention into MemNet in [79]. Its architecture is illustrated in Figure 20.

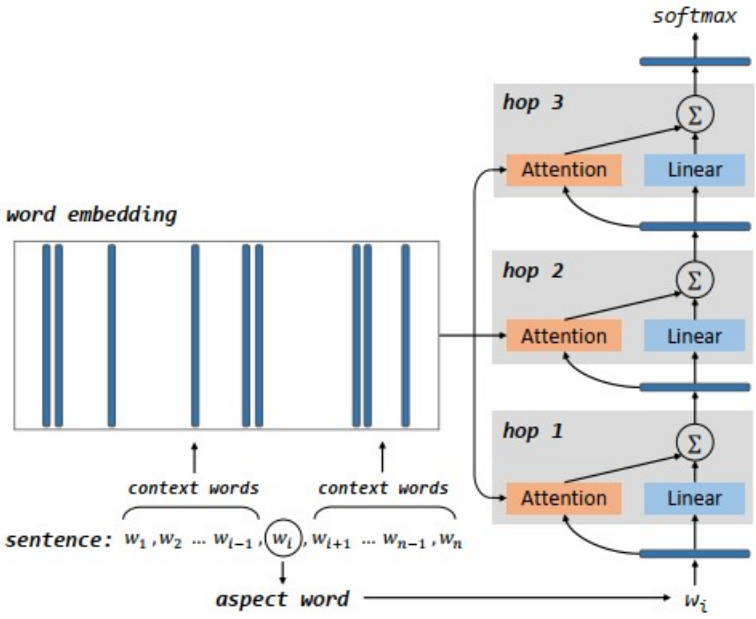


Figure 20: MemNet Architecture. *Source:* [79, p.3]

As a first step, the aspect term and the remaining parts of the text are embedded separately, where the latter is referred to as context or external memory m and the aspect vector is denoted as v_{aspect}. On the embeddings, a so-called *content attention* mechanism is applied with the scoring function

$$g_i = \tanh(W_{att}[m_i; v_{aspect}] + b_{att})$$

with weights W_{att} and bias b_{att}. Attention weights and outputs are obtained following Equations (3.3) and (3.4) with m_i as value vectors.

Additionally, so-called *location attention* is incorporated into the memory vector m which assigns more importance to those words that are closer to the aspect term. To do so, the location l_i of a word is defined as the absolute distance to the aspect term. The authors proposed four variants of location attention, yet they state their preference for one of them. Thus, we focus on their "Model 2" and refer the reader to [79] for the remaining variants. The location vector is calculated as $v_i = 1 - l_i/n$ with sequence length n. It is used to update the memory via $m_i = e_i \odot v_i$, where e_i is the embedding of the corresponding word.

As Figure 20 shows, in each layer or hop the content attention output is added to a linear transformation of the aspect vector, which then serves as input for the next layer. The output of the last hop is forwarded to a softmax layer for classification. The model is trained by minimizing cross-entropy loss.

AEN

The *Attentional Encoder Network (AEN)* was proposed in [76] and, as Figure 21 shows, its building blocks are an embedding layer, an attentional encoder layer, a target-specific attention layer and an output layer.

The embedding layer can either employ GloVe (cf. [62]) or BERT embeddings (see Section 3.4.3), determining the model names AEN-GloVe and AEN-BERT. The attentional encoder layer can be divided into two parts, where the first is multi-head attention (MHA, see Section 3.4.1). For introspective context words, self-attention (here: intra-MHA) is chosen, while multi-head cross-attention (here: inter-MHA) is applied to model context-perceptive words belonging to the aspect term. This basically just means that self-attention is used on context words, while cross-attention is used on the aspect terms.

The attention score function from Equation (3.2) is defined as

$$f_s(k,q_j) = \tanh([k_i; q_j]W_{att})$$

with weights W_{att}, keys k_i and queries q_j. The outputs of these two attention mechanisms are introspective context representations c^{intra} and context-perceptive target representations t^{inter}. They are fed to a *point-wise convolution transformation (PCT)* layer separately which is defined as

$$PCT(X) = \sigma(X * W_1 + b_1) * W_2 + b_2 \tag{4.1}$$

with inputs X, weights W_1, W_2 and biases b_1, b_2. For AEN, σ is the ELU activation function, where ELU is short for *Exponential Linear Unit*. Applying the PCT yields

$$h^c = PCT(c^{intra}) \text{ and } h^t = PCT(t^{inter}).$$

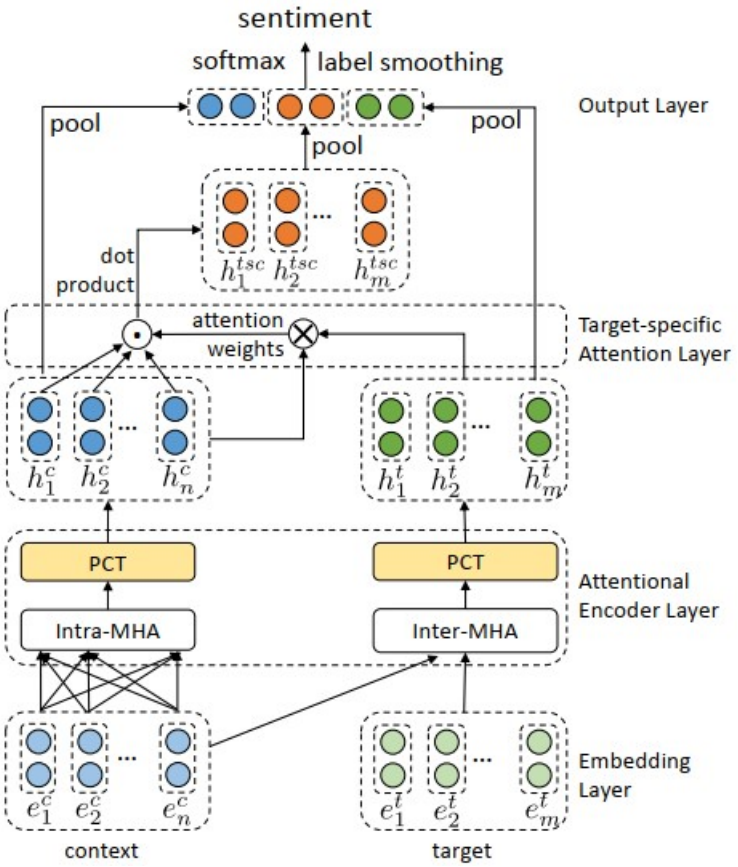


Figure 21: AEN Architecture. *Source:* [76, p.3]

Then, in the target-specific attention layer, another MHA is applied on both of these representations together in order to receive target-specific context representations $h^{tsc} = MHA(h^c, h^t)$. In the output layer, average pooling leads to h^c_{avg}, h^t_{avg} and h^{tsc}_{avg} which are concatenated and forwarded to a linear and a softmax layer for classification.

An issue the authors point out is the label unreliability of the neutral sentiment. This means that the neutral label often cannot be clearly distinguished from positive or negative polarities. Thus, they include *Label Smoothing Regularization* (LRS). This results in using smoothed labels (e.g. 0.1 and 0.9) instead of clear labels 0 and 1 during training. More formally, the true label distribution $q(k|x)$ of labels k is replaced by

$$q'(k|x) = (1 - \epsilon)\ q(k|x) + \epsilon\ u(k),$$

where ϵ is a smoothing parameter and $u(k)$ the prior label distribution. Here, $u(k) = 1/C$ with C sentiment classes is chosen to be uniform. The corresponding loss is equivalent to the Kullbach-Leibler divergence between $u(k)$ and the predicted distribution p_θ:

$$\mathcal{L}_{lrs} = -D_{KL}(u(k)||p_\theta).$$

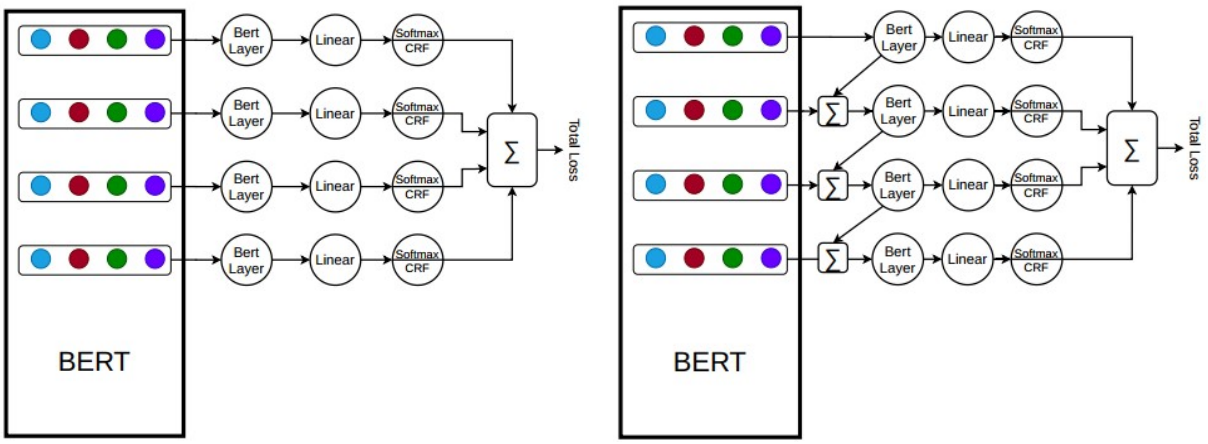
This loss is added to cross-entropy loss with L2-regularization which is to be minimized during training.

4.1.4 BERT-based Models

In this section, we focus on models that are based on a BERT model. In contrast to that, we have already mentioned approaches like AEN-BERT that use only BERT embeddings and do not rely on the whole architecture itself.

PH-SUM

Exploiting the layer-wise representations of BERT is the key behind the methods proposed in [39] which can be employed for both ATE and ATSC tasks. They propose two modules to be stacked on top of BERT. The first one is P-SUM which aggregates information in parallel as shown in Figure 22a. On top of the outputs of each of the last four BERT layers, another BERT layer is added. Each of these four lines of information is utilized for predictions which are then fed to a linear and a softmax or CRF layer. Their losses are summed up in the end. A CRF layer is chosen for the task of ATE in order to take the joint distribution of labels into consideration.



(a) (b)

Figure 22: Variants of PH-SUM: (a) P-SUM (b) H-SUM. *Source:* [39, p.3]

The other module, called H-SUM, merges information in a hierarchical way as depicted in Figure 22b. The idea behind it is based on Feature Pyramid Networks in [50] and it works like this: The output of the last BERT layer (which is on top of the image) is fed to another BERT layer. The corresponding output representations are aggregated with the output of the previous BERT layer, i.e. the one directly below the top layer, and then forwarded to a new BERT layer again. This procedure is applied to the last four BERT layers. The final outputs of each of them are then processed like in P-SUM.

BAT

Adversarial Training is used to improve BERT in BAT in [38] which is short for *BERT Adversarial Training*. Using this technique, the original input is changed a little in order to confuse the model to make wrong predictions. This procedure takes place during training and aims for a more robust model. The entire procedure is shown in Figure 23, where the linear layer accounts for sentiment classification.

Whereas in Computer Vision, these perturbations are possible to be made directly on the input, for text it is done on word embeddings. More precisely, the adversarial examples are created using the gradient of the classification loss. This idea was proposed in [56] who apply so-called *white-box attacks* that know the model parameters. Solving the following equation leads to the worst possible perturbations

$$r_{adv} = \arg\min_{r,||r||\leq\epsilon} \log p(y|x+r;\hat{\theta}), \tag{4.2}$$

where $p(y|x+r;\hat{\theta})$ is the probability of label y conditioned on input x and model parameters θ. Moreover, r stands for the perturbations of the input with a maximum of ϵ and $\hat{\theta}$ is a constant copy of the model parameters. Approximating Equation (4.2) with linear functions yields the perturbations

$$\tilde{r}_{adv} = -\epsilon \frac{g}{||g||_2} \text{ with } g = \nabla_x \log p(y|x;\hat{\theta})$$

which are added to the original input embeddings. These new samples are then again encoded with BERT and fed to a linear classification layer afterwards. The specific adversarial loss is calculated by

$$-\log p(y|x+\tilde{r}_{adv};\theta)$$

and summed up together with the original classification loss to the overall loss that is minimized during training. Note that the `[CLS]` token is not modified by perturbations.

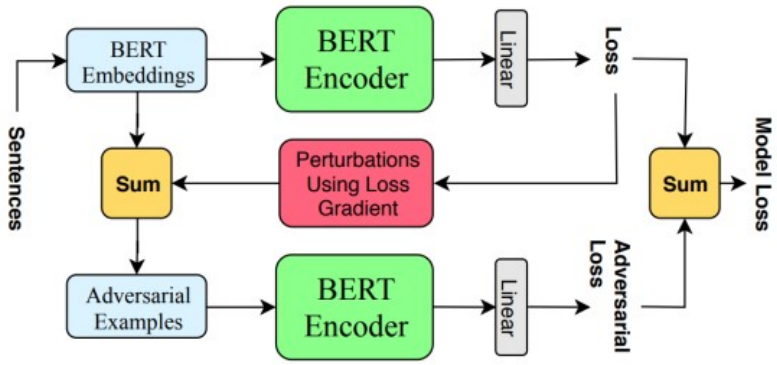


Figure 23: BAT Architecture. *Source:* [38, p.3]

4.1.5 Models based on Extra Data

While BAT works only with variations of the original data sets, here, we collect all those methods that rely on additional sources of data during training. In case they were evaluated in a cross-domain way, i.e. trained on one domain and tested on another, their results are stored in the Appendix (see Table 13).

MGALN

A *Multi-Granularity Alignment Network* was proposed in [49]. We refer to it as MGALN in order to distinguish it from MGATN in [18] which is usually also abbreviated with

MGAN. The novelty of MGALN is that it includes not only transfer learning between domains, but also between the tasks of ACSC and ATSC. This means that additional data X^s from domain D_s with sentiment labels for aspect categories are given. As usual, the task is to predict sentiments for aspect terms of a data set X^t of domain D_t. As aspect terms are more granular than aspect categories, this model is described as multi-granularity network. Its architecture is shown in Figure 24.

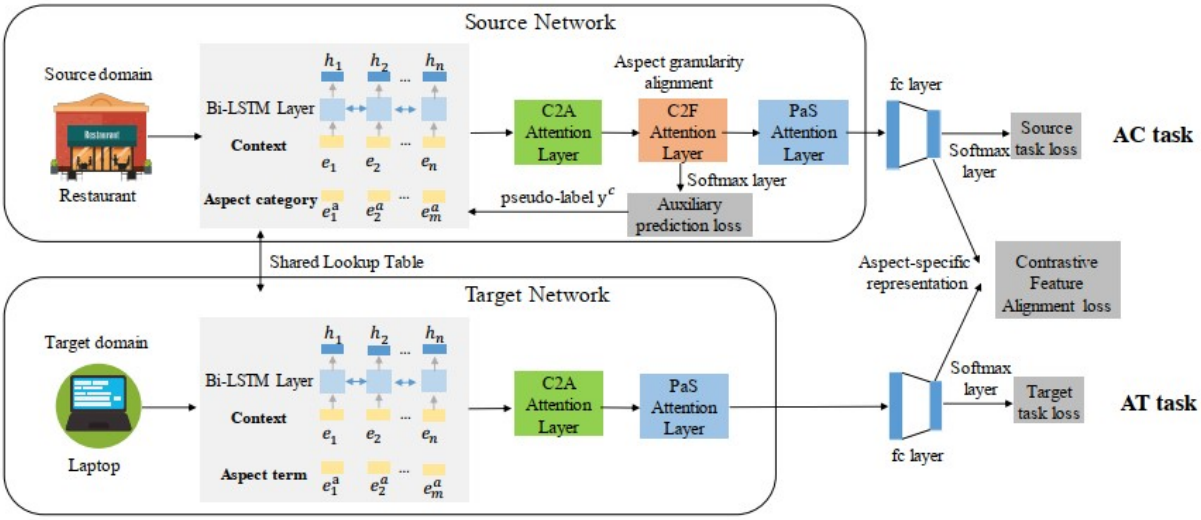


Figure 24: MGALN Architecture. *Source:* [49, p.5]

No matter from which task a sentence-aspect (term) pair is, both parts are embedded separately. For this model, sentence and context are used equally and have length n. The sentence embeddings $e = \{e_1, \ldots, e_n\}$ are then fed to a BiLSTM yielding contextualized representations $h = \{h_1, \ldots, h_n\}$. Together with the aspect embeddings $e^a = \{e^a_1, \ldots, e^a_m\}$, they are used to calculate the relevance scores from Equation (3.2) for Context2Aspect (C2A) attention as

$$M(i, j) = \tanh(W_a[h_i; e^a_j] + b_a).$$

They indicate the alignment between a context word i and an aspect word j. Weights and biases are denoted by W_a and b_a. The individual aspect-level attention weights $\delta(i)$ given the i-th word are received according to Equation 3.3 by applying softmax row-wise on $M(i, j)$. Then, the C2A attention weight is defined as $\alpha = \frac{1}{n}\sum_{i=1}^{n} \delta(i)$. Based on them, context-aware aspect representations, i.e. the attention outputs, are calculated as

$$h^a_* = \sum_{j=1}^{m} \alpha_j e^a_j,$$

where $* \in \{s, t\}$ stands for source or target domain.

Now, the process differs for the two domains: The representations of the source domain h_s^a are passed to an additional Coarse2Fine (C2F) attention layer. This includes an auxiliary pseudo-label task, where the aspect term a^s is seen as a pseudo-label y^c of the aspect category c. The goal is to predict the pseudo-category label of the aspect term based on the attention of h_s^a with respect to the context. This attention mechanism is computed with relevance scores

$$z_i^f = (u_f)^\top \tanh(W_f[h_i; h_s^a] + b_f).$$

Then, the attention weights β^f are obtained via softmax and the weighted sum of h_i is the output v^a. In this equation, W_f, u_f and b_f are layer-specific weights and biases. Feeding the attention output v^a to a softmax, results in the predicted pseudo-label $\hat{y}_k^c$ which is trained by minimizing the corresponding cross-entropy loss $\mathcal{L}_{aux}$. As there may not exist an explicit aspect term for every aspect category, a fusion gate F is added which controls how strongly h_s^a and v^a influence a source aspect representation r_s^a. It is computed via

$$\begin{aligned} F &= sigmoid(W[v^a; h_s^a] + b) \text{ and} \\ r_s^a &= F \odot h_s^a + (a - F) \odot W' v^a \end{aligned}$$

with weights W, W' and bias b.

The next layer is *Position-aware Sentiment* (PaS) attention which is again applied on both source and target representations r_s^a and $r_t^a := h_t^a$. Here, the position information for the target domain is included via a so-called *target position relevance* between a word i and the aspect term. It is defined as

$$p_i^t = \begin{cases} 1 - \frac{m_0 - i}{n}, & if\ i < m_0 \\ 0, & if\ m_0 \leq i \leq m_0 + m \\ 1 - \frac{i - (m_0 + m)}{n}, & if\ i > m_0 + m, \end{cases}$$

where sentence and aspect term length are denoted by n and m and the index of the first aspect word by m_0. Since the aspect category usually does not appear directly in a sentence in ACSC-labeled data, an alternative calculation has to be made for the source domain data. To do so, a location matrix is set up by

$$L_{ii'} = 1 - \frac{|i - i'|}{n}$$

with $i, i' \in [1, n]$ denoting the proximity of all words to each other. Then, the *source position relevance* for a word i is $p_i^s = L_i \beta^f$. Eventually, the PaS attention relevance scores are obtained by

$$z_i^o = (u_o)^\top \tanh(W_o[h_i; r_*^a] + b_o),$$

with weights u_o, W_o and b_o. The attention weights are calculated as softmax over $p_i^* z_i^o$ and the outputs v^o are the weighted sum of h_i. These are passed to a fully-connected layer and a softmax for sentiment classification. The model is optimized on the source or target domain by minimizing the corresponding cross-entropy losses $\mathcal{L}_{sen}^s$ or $\mathcal{L}_{sen}^t$.

Additionally, *Contrastive Feature Alignment* (CFA) is employed to cover up the difference between the domains. The two models for source and target domain are parameterized by g_s and g_t, respectively. Semantic Alignment (SA) is used to obtain identical distributions $\mathbb{P}(g_s(X^s))$ and $\mathbb{P}(g_t(X^t))$ for different domains, but same classes. In contrast to that, Semantic Separation (SS) makes sure that the two probability distributions are as diverse as possible for both different domains and classes. Then, the CFA loss is defined as

$$\mathcal{L}_{cfa} = \sum_{k,k'} \omega(g_s(x_k^s, a_k^s), g_t(x_{k'}^t, a_{k'}^t))$$

with a contrastive function for either semantic alignment or separation

$$\omega(u, v) = \begin{cases} ||u - v||^2, & if\ y_k^s = y_{k'}^t \\ max(0, D - ||u - v||^2), & if\ y_k^s \neq y_{k'}^t \end{cases}$$

with a degree of separation $D = 1$.

The source and target losses are defined as

$$\mathcal{L}_{src} = \mathcal{L}_{sen}^s + \mathcal{L}_{aux} + \lambda \mathcal{L}_{cfa} + \rho \mathcal{L}_{reg}^s \text{ and}$$
$$\mathcal{L}_{tar} = \mathcal{L}_{sen}^t + \lambda \mathcal{L}_{cfa} + \rho \mathcal{L}_{reg}^t$$

with hyperparameters λ, ρ and L2 regularization $\mathcal{L}_{reg}^*$. The source loss also includes an auxiliary loss $\mathcal{L}_{aux}$ for task alignment. The first stage of the two-step training procedure is about training the source network on the source domain by optimizing $\mathcal{L}_{src}$ without the term $\lambda \mathcal{L}_{cfa}$. Then, the source network is initialized with this whole pretrained version and the target network with the BiLSTM, C2A and PaS modules of the pretrained source network. In the second stage, $\mathcal{L}_{src}$ and $\mathcal{L}_{tar}$ are minimized in an alternating way.

BERT-PT

Although *BERT Post-Training (BERT-PT)* in [89] was mainly designed for the task of Question Answering, it can be used for ATE and ATSC as well. Focusing only on the latter, the architecture is the following: The BERT representation for the `[CLS]` token

which is the aspect-aware encoding of the whole sentence is forwarded to a fully-connected layer and a softmax. The model is trained by minimizing cross-entropy loss.

The more important part of BERT-PT is the employment of post-training methods on a pretrained BERT before fine-tuning. The reason for this is that for direct fine-tuning on the desired task may suffer from limited size of data and thus might result in struggling with respect to domain or task issues. Their post-training approach works jointly on domain knowledge (DK) data and machine reading comprehension (MRC) data. The first data set shares the domain, but is not labeled, which helps to understand the domain, while the other has flipped properties, which makes the model focus on the task. To post-train on domain knowledge, BERT's pretraining objectives Masked Language Modelling and Next Sentence Prediction are used. Thus, its loss is the sum of both objective losses: $\mathcal{L}_{DK} = \mathcal{L}_{MLM} + \mathcal{L}_{NSP}$. For post-training on task-aware knowledge, the SQuAD (cf. [69]) data set is taken. The corresponding loss is called $\mathcal{L}_{MRC}$ and similarly calculated as for the task of Question Answering for which we refer the reader to the paper mentioned above. Summing over both post-training losses leads to the combined post-training loss $\mathcal{L} = \mathcal{L}_{DK} + \mathcal{L}_{MRC}$ which should be minimized.

BERT-ADA

Based on the BERT model, the authors of [70] experiment with *domain adaptation* which is why their model is called BERT-ADA. The widely known domains Restaurants and Laptops were chosen for this procedure.

BERT-ADA consists of self-supervised domain-specific fine-tuning ("Language Model fine-tuning/LM") and supervised task-specific fine-tuning for ATSC ("Train") which take place after the regular pretraining of BERT. In order to evaluate a model trained in this way, a third step of testing ("Test") with its own data is needed as usual. This process can be depicted as

$$D_{LM} \rightarrow D_{Train} \rightarrow D_{Test},$$

where D stands for the corpus used in the corresponding step. In case of Laptops, for Language Model fine-tuning Amazon Laptop Reviews (cf. [26]) are chosen, while for Restaurants data from Yelp[1] is taken. The corpora for training the ATSC task D_{Train} are the SemEval-14 data sets.

[1] `https://www.yelp.com/dataset`

Creating all combinations of domains across all three steps leads to three categories:

- In-Domain Training: $D_{LM} \rightarrow T \rightarrow T$
- Cross-Domain Training: $D_{LM} \rightarrow S \rightarrow T \; with \; S \neq T$
- Joint-Domain Training: $D_{LM} \rightarrow (S \cup T) \rightarrow T \; with \; S \neq T$

A special case of Cross-Domain Training is Cross-Domain Adaptation with $T \rightarrow S \rightarrow T$. As there are three options for the first fine-tuning step, there are also three models to start with: BERT-ADA Laptop, BERT-ADA Restaurant and BERT-ADA Joint, where the last is the union of Laptops and Restaurant data sets. Language Model fine-tuning is algorithmically the same as pretraining BERT with Masked Language Modelling and Next Sentence Prediction tasks and thus similar to BERT-PT.

DomBERT

Profiting from domain knowledge is also the main idea behind DomBERT in [90]. DomBERT focuses on learning domains that are helpful for pretraining. It can be used for ATE, ATSC and in a combined way.

Given a set of source domains, DomBERT aims to choose those that are relevant for the target domain t and then performs Masked Language Modelling on their training examples with the corresponding loss $\mathcal{L}_{MLM}$. Thereby, embeddings for each domain are learned by the auxiliary task of Domain Classification. For this task, each text document is labeled with a domain tag l. A training example is built by aggregating texts from the same domain until the maximum input length for a BERT model is reached. Assume we have n source domains and one target domain, which leads to a total number of $n + 1$ domains. Having chosen the Amazon review data set (cf. [26]) and Yelp data[2] as auxiliary data sets, the authors had $n = 4679$. As DomBERT is based on BERT, the model works with BERT's outputs h, precisely with the output for the `[CLS]` token $h_{\texttt{[CLS]}}$. It is the document-level representation of an example and used for calculating the logits for all domains by

$$\hat{l} = D \; (W h_{\texttt{[CLS]}} + b)$$

with weights W and b. Factor $D = d_t \circ d_1 \circ \ldots \circ d_n$ is the concatenation of the $n + 1$ domain embeddings. For optimization, we simply use cross-entropy loss on domain tags l and $\hat{l}$ which yields $\mathcal{L}_{CLS}$. The following regularization is added to increase the diversity

[2] `https://www.yelp.com/dataset`, version 2019

of domain embeddings:

$$\Delta = \frac{1}{|D|^2} \; ||\cos(D, D^\top) - I||_2^2.$$

The final loss to be optimized then is $\mathcal{L} = \lambda \; \mathcal{L}_{MLM} + (1 - \lambda) \; \mathcal{L}_{CLS} + \Delta$.

Due to the auxiliary domain classification task, DomBERT contains a data sampler which draws examples from the target domain and other relevant domains and uses them for future learning. To do so, the target domain should have the highest probability for being sampled. This probability for a domain i is calculated using cosine similarity *cos* within

$$\mathbb{P}_i = \frac{\exp(\cos(d_t, d_i)/\tau)}{\sum_{j=0}^{n+1} \exp(\cos(d_t, d_j)/\tau)}$$

with temperature τ balancing between highly-ranked and long-tailed domains.

4.1.6 Models based on Local Context Focus (LCF)

A common assumption in terms of ATSC is that the sentiment of an aspect depends more strongly on words that are closer to the aspect than on those that are further away. To model this assumption, [95] introduced a *Local Context Focus (LCF)* mechanism which was picked up by others.

LCF-GloVe/-BERT

A Local-Context-Focus mechanism means that, in order to identify the sentiment of an aspect, an additional focus is set on words that are close to the aspect. To determine the local context, the *Semantic-Relative Distance (SRD)* was introduced in [95]. For every word in a sequence, it is defined as $SRD_i = |i - P_a| - \lfloor \frac{m}{2} \rfloor$, where i and P_a refer to the position of the corresponding word and the aspect, respectively. The length of the aspect is denoted by m. As far as we understand, positions and length m are counted with respect to words. So, the SRD basically measures the distance between words and aspects. Its value is calculated for each word around the aspect. If it comes below a certain threshold or is equal to it, this word is said to be part of the local context of the aspect. To clarify this, we use the following example: *"The staff was very friendly. However, it is too expensive."* Take "staff" as the aspect of choice. We label the positions for all words from 1 to 8, which results in the position of the aspect $P_a = 2$. The aspect itself has a length of $m = 1$, which reduces the last part of the formula to zero. The SRDs

for the words in our review can be seen in Table 5. Setting the SRD threshold $\alpha = 3$, "The", "was", "very" and "friendly" belong to the local context of the aspect "staff".

Word	The	staff	was	very	friendly	However	it	is	too	expensive
SRD	1	-	1	2	3	4	5	6	7	8

Table 5: SRD values for an example sentence.

The authors suggested two variants of embeddings which are part of the complete architecture depicted in Figure 25: The basic architecture LCF-GloVe consists of GloVe embeddings (cf. [62]) and a pre-feature extractor, whereas in the LCF-BERT variant these two are substituted by a BERT layer. The pre-feature extractor (PFE) consists of multi-head self-attention (see Section 3.4.1) followed by a tanh and position-wise convolution transformation as it is defined in Section 4.1.3. Different to the AEN models, the activation function in Equation (4.1) is a ReLU here.

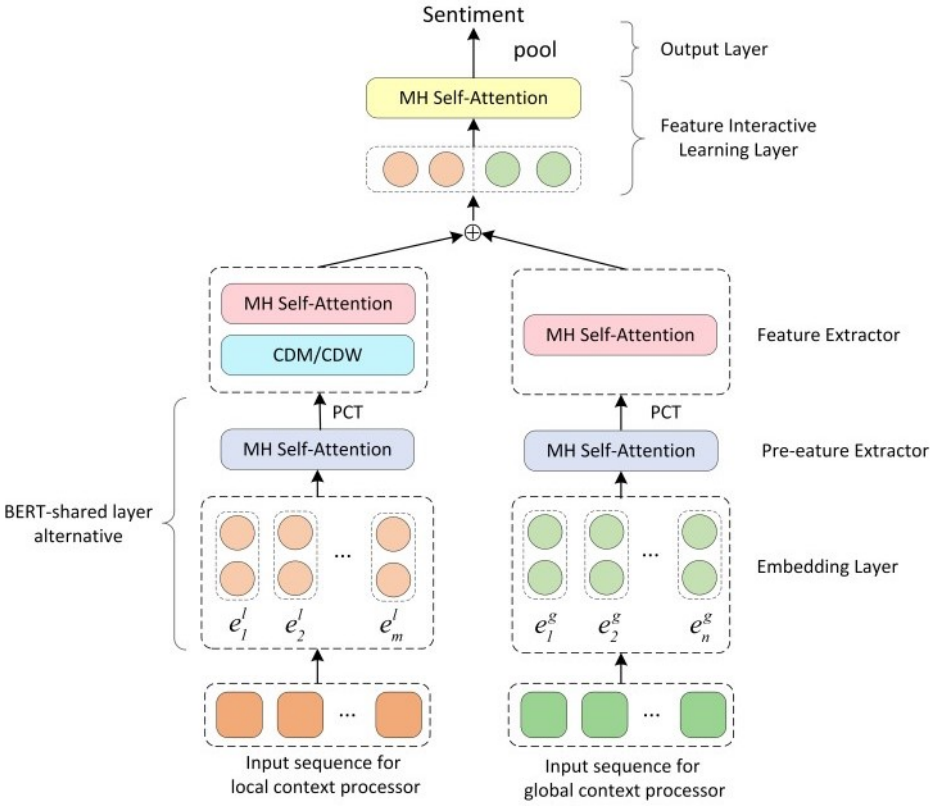


Figure 25: LCF Architecture. *Source:* [95, p.5]

However, ATSC is not only based on the local context, but also on the global context of the aspect. Depending on the chosen design, out-of-local-context words are not taken into account, in case of a *Context Features Dynamic Mask Layer (CDM)*, or with smaller weights, in case of a *Context Features Dynamic Weighted Layer (CDW)*. CDM simply masks out those words that have an SRD value higher than the threshold α. The weights of the out-of-local-context words for CDW are determined by $\frac{SRD_i - \alpha}{n}$, where n is the sequence length.

The outputs of the feature extractors are then concatenated, fed to a dense and then to a multi-head self-attention layer. This procedure takes place in the so-called *feature interactive learning layer*. The final classification output is obtained by applying a pooling and a softmax layer. The model is trained by minimizing cross-entropy loss with L2 regularization.

LCF-ATEPC

Also in [93], the idea of LCF was taken up; yet, they turned it into an approach for both ATE and ATSC as shown in Figure 26. For each input token, two labels are assigned: The first indicates whether this token is at the beginning (B_{ASP}), inside (I_{ASP}) or outside (O) of an aspect term; the second indicates the polarity. This is the joint labeling scheme from Section 2.2; yet, extracted aspect terms are not directly used for polarity classification. Thus, applying aspect extraction first and then sentiment classification would make this approach a pipeline method. However, we now focus only on how the single-task method works.

The architecture of LCF-ATEPC strongly resembles the one of LCF-BERT in Figure 25. One difference appears within the Local Context Feature Generator, more precisely in CDW: The weights for out-of-local-context words are now calculated as $\frac{n - SRD_i - \alpha}{n}$. Additionally, the authors also experimented with computing outputs of both CDM and CDW, concatenating them and doing a linear transformation before applying multi-head self-attention. Like before, a tanh is added on top of the multi-head self-attention. This type of model is called *fusion* and more successful. The remaining parts of ATSC seem to be the same as in the original LCF architecture, unless Figure 26 does not depict multi-head self-attention for global context. Moreover, this output is fed to a softmax which predicts whether the corresponding token is part of an aspect term. The authors also experimented with a BERT model that was post-trained on the chosen domain as it was proposed in [70]. This clearly improves the performance.

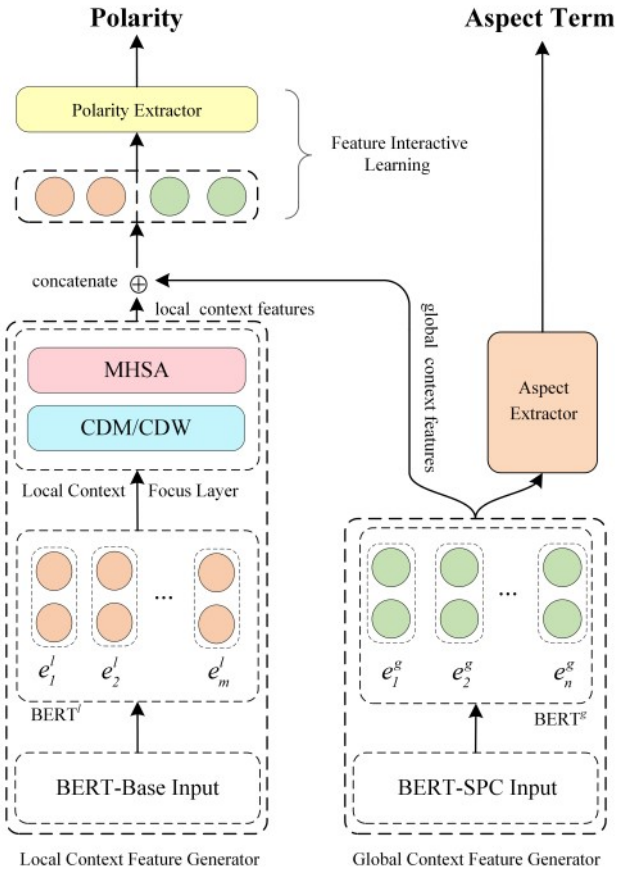


Figure 26: LCF-ATEPC Architecture. *Source:* [93, p.5]

LCFS-ASC

The idea of the local context focus mechanism was also followed in [63] and modified with respect to the Semantic Relative Distance. As it only counts the distances between words so far, they replaced it by a *Syntactic Relative Distance (SyRD)*. It is defined as the number of steps of the shortest path of a dependency parsing tree between two words. Taking their example for which the dependency parsing tree is shown in Figure 27, we illustrate the computation of SyRD: As $SyRD(amplifier, loudly) = 2$ and $SyRD(sound, loudly) = 3$, the SyRD of the entire aspect term "Sound Amplifier" is 2.5. Thus, their approach for ATSC is called *Local Context Focus on Syntax (LCFS-ASC)*. They also developed a new approach for ATE which can be used before ATSC within a pipeline. However, since this part is independent of the ATSC part, we will forego its details here and refer the reader to the paper.

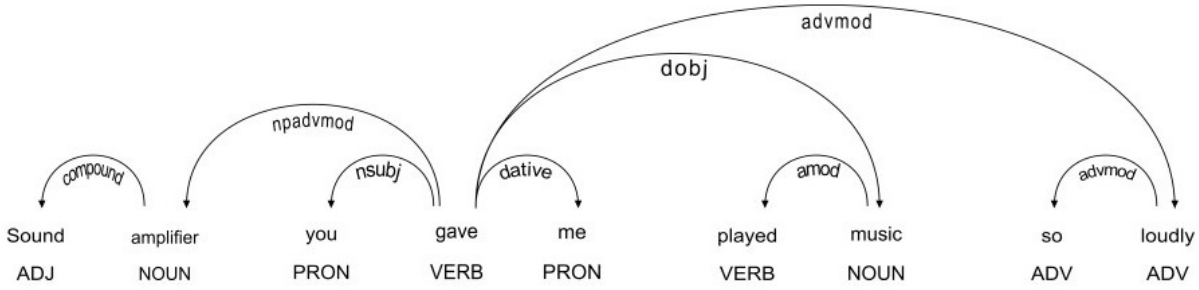


Figure 27: Example of a Dependency Parsing Tree. *Source:* [63, p.3216]

4.1.7 Methods based on Graphs

Of course, [63] were not the only ones to exploit a dependency tree when building models for ATSC. Therefore, the complete next section is dedicated to these models. LCFS-ASC, however, employs it within the Local Context mechanism which is why that approach has been part of the previous section.

AdaRNN

One of the first to do so were [17] who proposed an RNN- and syntax-based approach called *Adaptive Recursive Neural Network (AdaRNN)*. Besides, they also introduced the Twitter data set (details provided in the Appendix) which is equally popular as the SemEval-14 data sets.

For each sentence, a dependency tree has to be given. It is converted into a tree where the aspect term acts as the root node. During this process, the sentiments of the context are propagated recursively to the aspect term, starting with those words directly connected to the target. The key feature of AdaRNN is to employ multiple composition functions that are selected based on linguistic tags and combined vectors. These composition functions are used for sentiment propagation. The composition result of a parent node is defined as

$$v = f\left(\sum_{h=1}^{C} \mathbb{P}(g_h|v_l, v_r, e) g_h(v_l, v_r)\right), \tag{4.3}$$

where v_l, v_r are left and right child vectors and f a non-linear function. The external feature vector e encodes the relation type in a binary way. The compositions functions $g_1, \ldots, g_C$ are linear transformations with weights $W_1, \ldots, W_C$ and biases $b_1, \ldots, b_C$. Their

distribution is given by

$$\begin{bmatrix} \mathbb{P}(g_1|v_l, v_r, e) \\ \dots \\ \mathbb{P}(g_C|v_l, v_r, e) \end{bmatrix} = softmax \left(\beta S \begin{bmatrix} v_l \\ v_r \\ e \end{bmatrix} \right) \tag{4.4}$$

with S determining which composition function to use and hyperparameter β. In this way, the representations for all words in a sentence are calculated. The representations of the root node are then forwarded to a softmax layer for classification. For optimization, L2-regularized cross-entropy loss is to be minimized.

PhraseRNN

PhraseRNN, proposed in [58], is an extension of AdaRNN and employs information from both dependency and constituent trees. Different to AdaRNN, not a list of global composition functions, but special composition functions for inner-phrase (G) and outer-phrase (H) words are chosen.

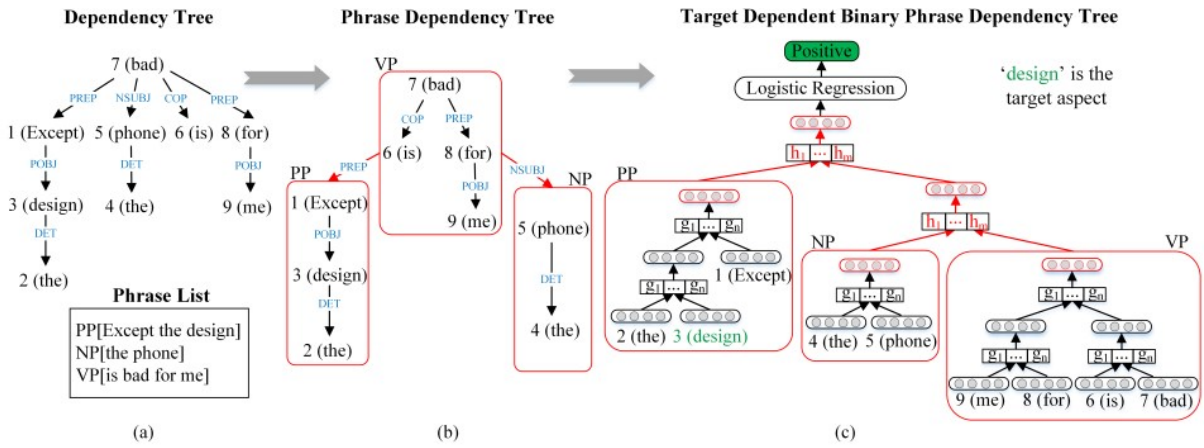


Figure 28: PhraseRNN Architecture. *Source:* [58, p.2511]

The procedure which is shown in Figure 28 is the following: At first, a constituent tree is utilized to extract phrases from each sentence. With phrases simply parts of the sentence are meant, e.g. noun phrases (NP) or verb phrases (VP). Based on these phrases P, a so-called *phrase dependency tree* is generated from the given dependency tree $T = (V, E)$ with vertices V and relation edges E. The resulting tree has the form $pT = (pV, pE)$ with pV being a set of subtrees of T and $pE = \{(r_{ji}, T_i, T_j)\}$ denoting the set of relations r_{ji} between subtrees. This tree is then again converted into another tree bpT which stands for

target dependent binary phrase dependency tree. It takes the target word v_t as additional input. For the detailed tree transformation algorithms, we refer to the paper.

Similarly to Equation (4.3) of AdaRNN, based on this tree, the representation of an inner-phrase parent node v_{in} is calculated as

$$v_{in} = f\left(\sum_{i=1}^{n} \mathbb{P}(g_i|v_l, v_r, e_{in}) g_i(v_l, v_r)\right)$$

with the same components as above, besides the external feature vector e_{in} which will be explained in the following paragraph. Also the probability distribution of the g_i's follows Equation (4.4) from AdaRNN except for the new definition of e_{in} again. For an outer-phrase parent node, the representations v_{out} and the probability distributions are obtained analogously, substituting e_{in} with e_{out} and g_i with h_i.

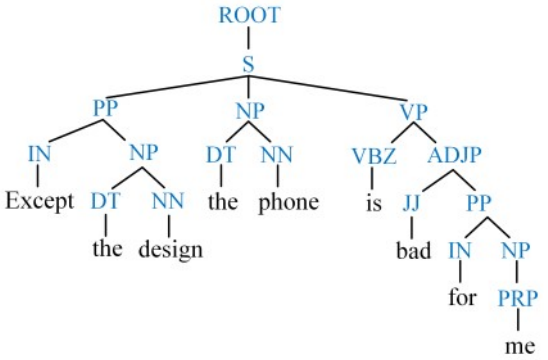


Figure 29: Constituent Tree Example. *Source:* [58, p.2510]

Both external features $e_{in/out,i}$ contain the labels of the left and right child nodes as well as the dependency relation of node v_i: $e_{in/out,i} = (label_l, label_r, DepType_i)$. The labels $label_l, label_r$ are the POS tags of the corresponding node in the constituent tree in Figure 29 if it is a leaf word; otherwise the label is the tag of the lowest common parent of descendants of that node. For instance, the label is "NP" for the node created from "the" and "design". The dependency relation is taken from the original dependency tree, i.e. the direct relationship between the child nodes if they are leaves. If not, it is the relationship between their head words. For example, it is "COP" for "is" and "bad", "POBJ" for "for" and "me" and "PREP" for their overall parent. The model is trained minimizing the L2-regularized sum of the mean of the negative log likelihood.

SynATT

Also the SynATT model, proposed in [24], works with syntactic distances. The basis of their model is an attention-based LSTM to which they added two components. The overall architecture can be found in Figure 30.

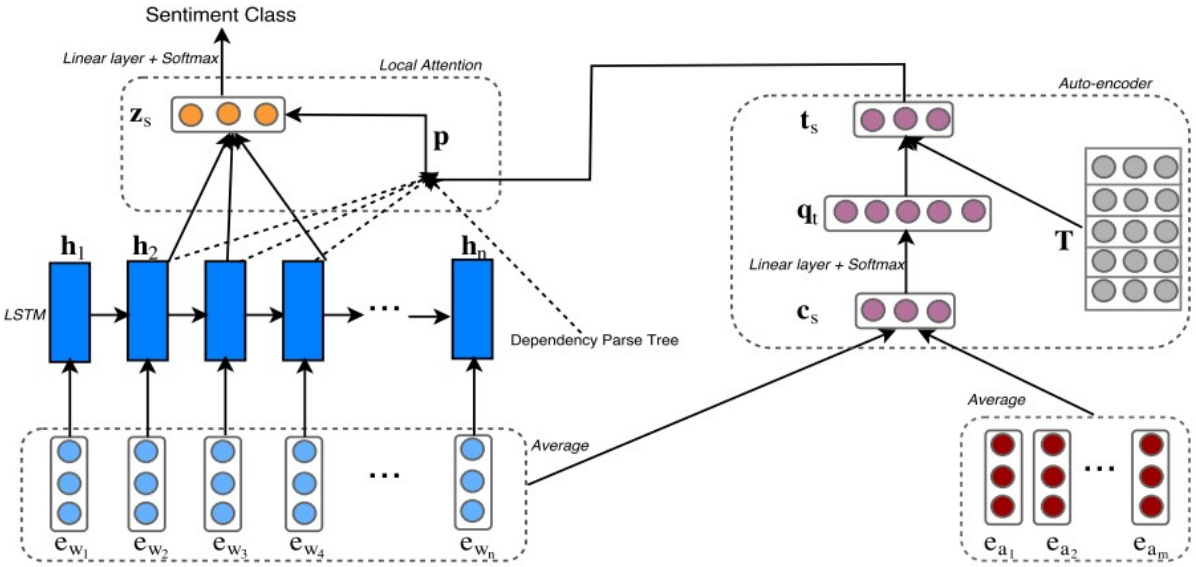


Figure 30: SynATT Architecture. *Source:* [24, p.1124]

In a standard attention-based LSTM, the representation of a sentence is the weighted sum of LSTM outputs h: $z_s = \sum_{i=1}^{n} p_i h_i$ which has the same form as the attention output Equation (3.4). The attention weights p_i are computed following Equation (3.3) as softmax of relevance scores d_i. They are based on the aspect term representations t_s which are usually the average over the embedded aspect term e_{a_i}.

One novelty of SynATT is the formula of t_s which is calculated as

$$t_s = T^\top q_t \text{ with } q_t = softmax(W_t c_s + b_t)$$
$$\text{and } c_s = average\left(\frac{1}{m}\sum_{i=1}^{m} e_{a_i}, \frac{1}{n}\sum_{j=1}^{n} e_{w_j}\right).$$

The last line c_s incorporates both aspect term and context information and q_t are weights for the aspect embeddings matrix T which is randomly initialized. Weights and biases are denoted by W_t and b_t and e_{w_j} are word embeddings. This procedure is depicted in the right half of Figure 30.

The second novelty of SynATT concerns the attention weights p_i that now also take the syntax of the sentence into account. The new weights are calculated as

$$p_i = \frac{d_i}{\sum_j d_j} \text{ with } d_i = \begin{cases} \frac{1}{2^{l_i-1}} \exp(f_{score}(h_i, t_s))), & if\ l_i \in [1, ws] \\ 0, & \text{otherwise} \end{cases}$$

with $f_{score}(h_i, t_s) = \tanh(h_i^\top W_a t_s)$ and weights W_a. The location l_i is the distance of a word to the aspect term counting the number of paths between them in the dependency tree (like in LCFS-ASC in Section 4.1.6). The window size ws indicates in which range the context words are taken into account for the attention mechanism.

The model is trained by minimizing a loss composed of several objective functions:

$$\mathcal{L}(\theta) = J(\theta) + \lambda_u U(\theta) + \lambda_r R(\theta).$$

with hyperparameters λ_r and λ_r. The cross-entropy loss of sentiment classification is denoted by $J(\theta)$ and $U(\theta)$ is introduced to have high quality aspect embeddings. It can be seen as the reconstruction error of an autoencoder (cf. [72]) and it is defined as

$$U(\theta) = -\sum_{(s,a)\in D} \log(\min(\epsilon, CosSim(t_s, c_s))).$$

The trainable parameters are denoted by θ and $CosSim$ is the cosine similarity. Besides, D stands for the set of training samples of a sentence-aspect pair (s, a) and $\epsilon = 10^{-7}$. In the overall loss, the third term is added to make each aspect embedding more unique and it is thus defined as

$$R(\theta) = ||(T_{norm} T_{norm}^\top - I)^2||$$

with T_{norm} being the L2-normalized form of T.

SDGCN

SDGCN stands for *Sentiment Dependencies with Graph Convolutional Networks*. This approach in [98] includes the relationship between several aspects from which additional information regarding sentiments can be retrieved. For instance, take a look at the example *"The setting is romantic, but the food is horrible."* with aspect terms "setting" and "food". The first is clearly positive and as it is connected to the second aspect via "but", it makes sense for "food" to be negative. This kind of relationship is used here to improve predictions.

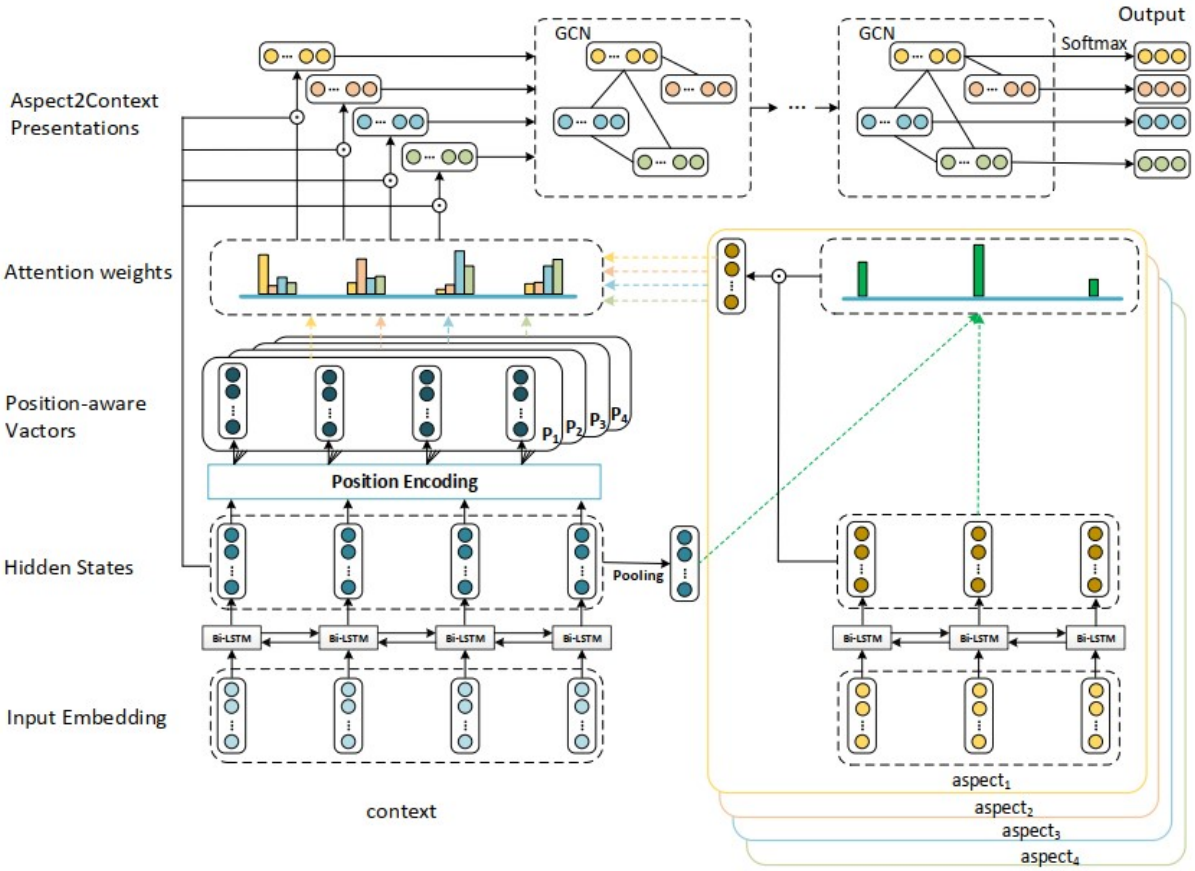


Figure 31: SDGCN Architecture. *Source:* [98, p.3]

As Figure 31 shows, at first each word is embedded using either GloVe (cf. [62]) or BERT embeddings, resulting in SDGCN-GloVe and SDGCN-BERT, respectively. The embeddings for the whole sentence (here called context) and for the aspect terms are then fed separately to two BiLSTMs yielding hidden representations $H^c = [h_1^c, \ldots, h_N^c]$ and $H^{a_i} = [h_1^{a_i}, \ldots, h_{M_i}^{a_i}]$ with sentence length N and aspect term length M_i. The relative distance between a word t and an aspect i is defined by

$$d_t^{a_i} = \begin{cases} 1, & if\ dis = 0 \\ 1 - \frac{dis}{N}, & if\ 1 \leq dis \leq s \\ 0, & if\ dis > s, \end{cases}$$

where dis is the absolute distance between the word and the aspect term with $dis = 0$ if the word itself is part of the aspect term. A constant threshold is denoted by s. The relative distance is used to identify those words that are closer to a specific aspect term,

just like in local context focus models. Based on them, position-aware representations are computed by

$$P^{a_i} = P_i = [p_1^{a_i}, \ldots, p_N^{a_i}] \text{ with } p_t^{a_i} = d_t^{a_i} h_t^c.$$

Then, a bidirectional attention mechanism is applied, consisting of context to aspect and aspect to context attention. With the first, new aspect representations are obtained based on the context. This is what is shown on the right side of Figure 31. The relevance scores from Equation (3.2) are calculated as

$$f_{ca}(\bar{h^c}, h_t^{a_i}) = \bar{h^c}^\top W_{ca} h_t^{a_i}$$

with weights W_{ca} and $\bar{h^c}$ being the output of average pooling of H^c. For attention weights of $h_t^{a_i}$ and the output m^{a_i}, calculations follow Equations (3.3) and (3.4). The second attention mechanism, however, yields aspect-specific context representations $x^{a_i} = x_i$ according to a similar procedure. Relevance scores are defined as

$$f_{ac}(m^{a_i}, p_t^{a_i}) = m^{a_i \top} W_{ac} p_t^{a_i}$$

with corresponding weights W_{ac}. Equations (3.3) and (3.4) again lead to attention weights for h_t^c and outputs x_i. Stacking the representations based on all K aspect terms together results in a sentence representation $X = [x_1, \ldots, x_K]$.

Then, this works as input to a GCN (see Section 3.5.1). As its basis, two undirected sentiment graphs are generated in which a node represents an aspect and an edge the corresponding sentiment dependency relation. In the *adjacent-relation graph*, each aspect is only connected to its direct aspect neighbors, whereas in the *global-relation graph* all aspects are connected to each other. Experiments showed that the latter performed slightly better. For a node v, the representation of the l-th GCN layer is updated by

$$x_v^l = ReLU\left(\sum_{u \in N(v)} W_{cross} x_u + b_{cross}) + ReLU(W_{self} x_v^{l-1} + b_{self})\right)$$

where W_{cross}, W_{self} are weights, b_{cross}, b_{self} biases and $N(v)$ is the set of neighbors of v. This formula is an adaptation of Equation (3.5). In each layer, the node representations of the previous layer are taken as input. The nodes x_i^L of the last layer L are then fed to a fully-connected layer and a softmax. Training is done by minimizing L2-regularized cross-entropy loss.

CDT

The CDT model, which is short for *Convolution over a Dependency Tree* and which was proposed in [77], is also based on a Graph Convolutional Network (GCN). Its architecture is presented in Figure 32.

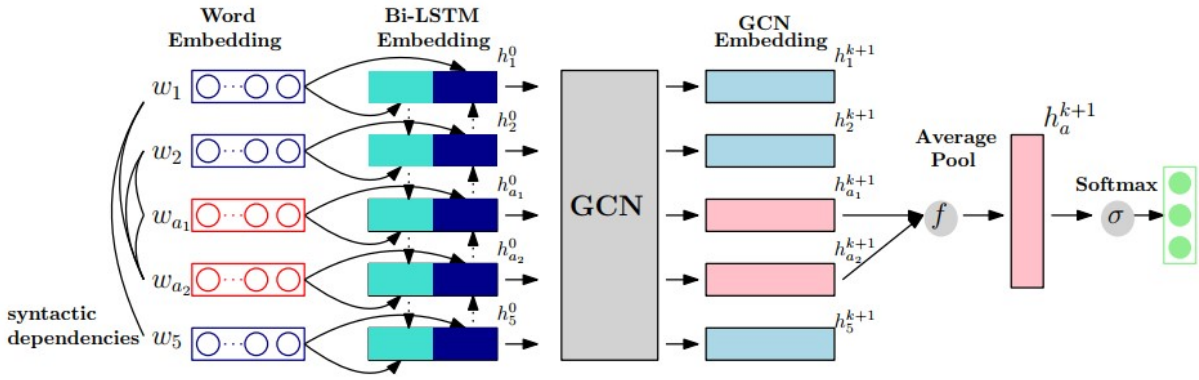


Figure 32: CDT Architecture. *Source:* [77, p.250]

At first, an embedded sentence is fed to a BiLSTM which outputs the representations $\{h_1^0, \ldots, h_n^0\}$. They build the basis for a dependency tree which is a graph G with n nodes. Each node stands for a word which is represented by the corresponding output of the BiLSTM, whereas each edge stands for a syntactic dependency path. The dependency tree is represented by an adjacency matrix

$$A = \begin{cases} 1, & \text{if node i is connected to node j} \\ 0, & \text{otherwise} \end{cases} \tag{4.5}$$

specifying the relationship between the nodes. As usual in a GCN, they are updated by aggregating the information which is propagated along the tree. The update for a single node embedding has the following form which is based on Equation (3.5):

$$h_i^{(k+1)} = ReLU\left(\sum_{j=1}^{n} c^i A_{ij}(W^{(k)} h_j^{(k)} + b^{(k)})\right). \tag{4.6}$$

Here, $c^i = 1/d_i = (\sum_{j=1}^{n} A_{ij})^{-1}$ is a normalization constant with d_i indicating the degree of node i. Furthermore, $h_j^{(k)}$ is the hidden state representation of node j in layer k and $W^{(k)}$ and $b^{(k)}$ are weights and biases. After $K+1$ layers, average pooling is applied on the aspect vectors $h_{a_1}^{K+1}, \ldots, h_{a_l}^{K+1}$ in order to retain most of the information, resulting in aspect-based representations $h_a^{(K+1)}$. They are then passed to a linear and a softmax

layer for classification.

The model is trained by minimizing the cross-entropy loss

$$\mathcal{L}(\theta_1, \theta_2) = -\sum_{(a,s)\in D}\sum_{c\in C} y_c((a,s)) \log \hat{y}_c((a,s))$$

with the set of all aspect-sentence pairs D, C sentiment classes, true labels $y_c((a,s))$ and the corresponding predictions $\hat{y}_c((a,s))$. The parameters of the BiLSTM and the GCN are denoted by θ_1, θ_2, respectively.

ASGCN

The *Aspect-Specific Graph Convolutional Network (ASGCN)* introduced in [97] partly resembles the CDT model. Its architecture is presented in Figure 33.

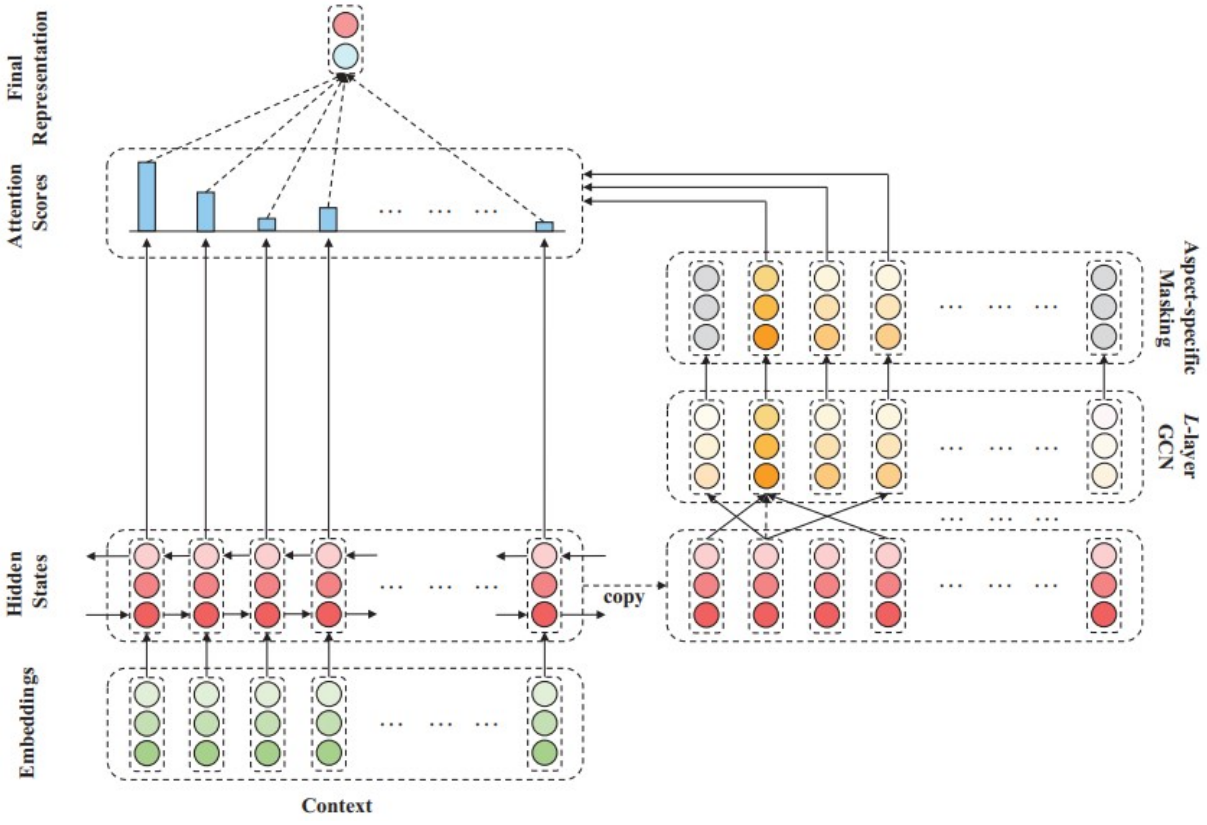


Figure 33: ASGCN Architecture. *Source:* [97, p.4571]

Again, each embedded sentence is fed to a BiLSTM to receive hidden state vectors $H^c = \{h_1^c, \ldots, h_n^c\}$. Like before, aspect-oriented features are obtained by applying a GCN on the dependency tree that is initialized with H^c. Here, two variants of the dependency trees are chosen: ASGCN-DG employs un-directional graphs, whereas ASGCN-DT uses directional trees. This difference is denoted in the corresponding adjacency matrix A (defined in Equation (4.5)) which is more sparse for ASGCN-DT. Additionally, the diagonals of all adjacency matrices are ones, i.e. each word is adjacent to itself. This is similar to the modified adjacency matrix of the original GCN in Section 3.5.1. Almost completely analogously to Equation (4.6), the node update is executed by

$$h_i^l = ReLU\left(b^l + \frac{1}{d_i + 1}\sum_{j=1}^{n} A_{ij} W^l g_j^{l-1}\right),$$

where g_j^{l-1} is the representation of token j from layer $l-1$ and $d_i = \sum_{j=1}^{n} A_{ij}$. As usual, W^l and b^l denote weights and bias. Before a representation h_i^l is forwarded to the next GCN layer, it receives a position-aware transformation $g_i^l = q_i h_i^l$ that assigns higher position weights to closer context words. It is employed to decrease noise and bias introduced by the dependency parsing process. The position weights are defined as

$$q_i = \begin{cases} 1 - \frac{\tau+1-i}{n}, & if\ 1 \leq i < \tau + 1 \\ 0, & if\ \tau + 1 \leq i \leq \tau + m \\ 1 - \frac{i-\tau-m}{n}, & if\ \tau + m < i \leq n \end{cases}$$

where the aspect term has length m and starts at position $\tau + 1$. The output of the GCN with L layers is denoted as $H^L = \{h_1^L, \ldots, h_n^L\}$.

Similar to the CDT model, aspect-oriented features H_{mask}^L are obtained by masking out the hidden state vectors that do not belong to an aspect word. Yet, a clear difference to CDT is the incorporation of an attention mechanism to identify semantically relevant words of the aspect terms. The relevance scores of Equation (3.2) are computed as

$$\beta_t = \sum_{i=1}^{n} (h_t^c)^\top h_i^L = \sum_{i=\tau+1}^{\tau+m} (h_t^c)^\top h_i^L.$$

The attention weights α_t are obtained following Equation (3.3) and lead to attention outputs $r = \sum_{t=1}^{n} \alpha_t h_t^c$. Applied on them, a linear layer and a softmax yield the classification probabilities. The model is trained minimizing cross-entropy loss with L2-regularization.

TD-GAT

A Graph Attention Network (GAT, see Section 3.5.2) serves as basis for the *Target-Dependent GAT (TD-GAT)* model in [31]. First, an embedded sentence is turned into an un-directed graph by a dependency parser (cf. [6]). In case of multi-word aspect terms, the aspect term is substituted by a placeholder before feeding the sentence to the dependency parser. Its local feature vector is the mean of the embeddings of all aspect term words. All word representations are updated exploiting the graph structure in every GAT layer as described in Section 3.5.2. The only difference is that a Leaky ReLU is applied on top of the relevance scores before calculating the attention weights as their softmax. For the last layer L, concatenation is used and results in representations H_L.

In addition to the GAT, an LSTM is employed to model the connections from words to aspect terms across layers. The idea is that the higher the layer, the more information from words further away is incorporated. Thus, the TD-GAT model can be summarized by the update rules

$$H_{l+1}, C_{l+1} = LSTM(GAT(H_l))$$

with state initializations (H_l, C_l). The first states are $H_0, C_0 = LSTM(XW_p + [b_p]_N)$ with zeros as starting points. Here, C_l is the memory cell of the LSTM in layer l and W_p and $[b_p]_N$ are weights and biases, the latter stacked N times. The stacked embeddings vectors are denoted by X. It is also possible to use a GRU instead of an LSTM. The output of this architecture with L layers is a representation for the aspect term node. It is fed to a linear and a softmax layer for classification. It is trained by minimizing cross-entropy loss with L2 regularization.

RGAT

One of the latest approaches, the *Relational Graph Attention Network (RGAT)*, was introduced in [4]. It is based on the three components contextual encoder, RGAT encoder and classifier which are shown in Figure 34.

The input of RGAT is a triplet $\langle \mathcal{T}, \mathcal{S}, \mathcal{G} \rangle$, where $\mathcal{T}$ stands for the aspect term of length m appearing in a sentence $\mathcal{S}$ of length n. The syntactic graph over $\mathcal{S}$ is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{A}, \mathcal{R})$ with the node or word set $\mathcal{V}$, an adjacency matrix $\mathcal{A}$ (as defined in Equation (4.5)) and a label matrix $\mathcal{R}$. Each entry $\mathcal{R}_{ij}$ contains the label of the corresponding $\mathcal{A}_{ij}$ if it does not equal zero.

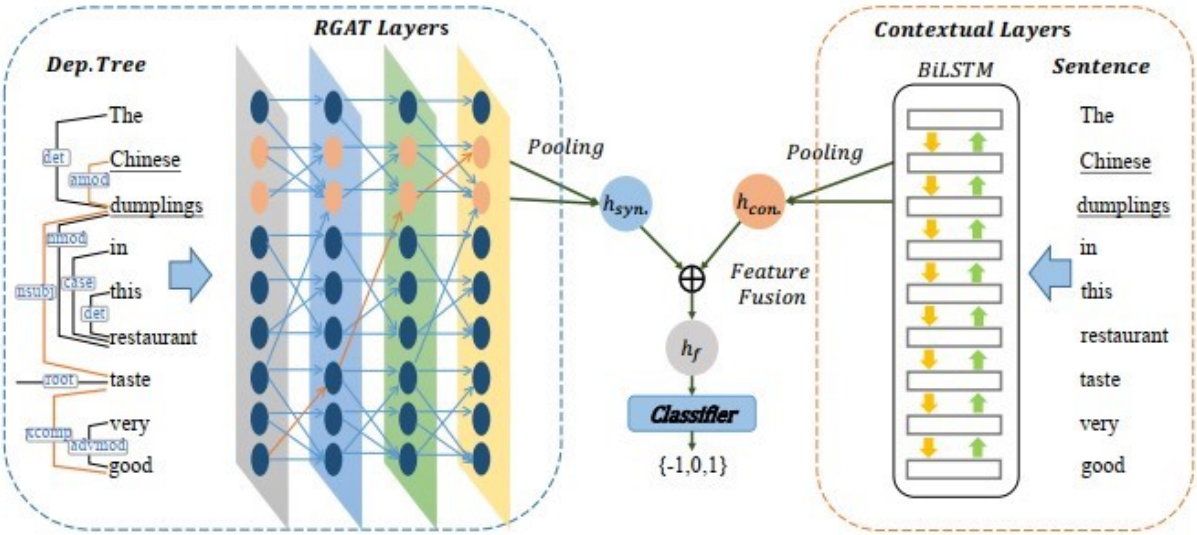


Figure 34: RGAT Architecture: RGAT encoder, classifier and contextual encoder. *Source:* [4, p.4]

The contextual encoder basically is a model being able to do feature learning and aggregation, like a BiLSTM or BERT. It outputs a contextual vector $(h_i, \ldots, h_{i+m-1})$ for the aspect term. The RGAT encoder, however, creates syntax-aware aspect term embeddings $(\hat{h}_i, \ldots, \hat{h}_{i+m-1})$. The difference to the original GAT is that the RGAT encoder can work with a labeled graph. In several so-called *RGAT layers* (as shown in Figure 35), node-aware and relation-aware attention are calculated and applied in a combined way. To do so, the entries of the label matrix $\mathcal{R}_{ij}$ are first transformed into vectors $r_{ij} \in \mathbb{R}^{d_r}$ with the dimension of relation embeddings d_r. They are used to calculate *(not normalized) relation-aware attention weights*

$$e_{ij}^R = \begin{cases} f(h_i^{l-1}, r_{ij}), & if\ j \in \mathcal{N}(i) \\ -\inf, & otherwise \end{cases}$$

with an attention function f and the index set $\mathcal{N}(i)$ of neighbors of word i induced by $\mathcal{A}$. They are calculated in addition to the *(not normalized) node-aware attention weights* e_{ij}^N which are the same as in the normal GAT (see Section 3.5.2). The final *normalized attention scores* are defined by

$$\alpha_{ij} = \frac{\exp(e_{ij}^N + e_{ij}^R)}{\sum_{j' \in \mathcal{N}(i)} \exp(e_{ij}^N + e_{ij}^R)},$$

almost identically as in the usual GATs. Then, the feature vector of a word i is updated

with weights W_{Vr}^{l} via

$$\hat{h}_i^l = ||_{k=1}^{Z} \sigma \left(\sum_{j \in \mathcal{N}(i)} \hat{\alpha}_{ij}^{lz} \ \left(W_V^{lz} h_j^{l-1} + W_{Vr}^{l} r_{ij} \right) \right),$$

which is similar to Equation (3.6).

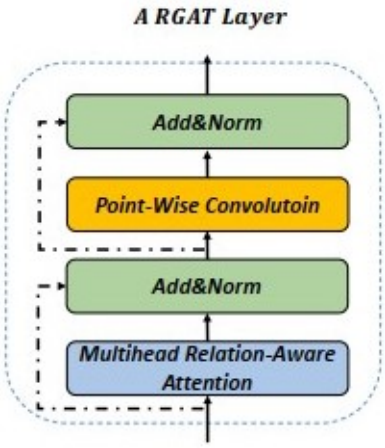


Figure 35: RGAT Layer. *Source:* [4, p.4]

Repeating this procedure in L RGAT layers yields syntax-aware aspect term representations $(\hat{h}_i^L, \ldots, \hat{h}_{i+m-1}^L)$ for which we drop the superscript L for simplification. As well as on the contextual representations $(h_i, \ldots, h_{i+m-1})$, average pooling is applied on them to receive global representations

$$h_{con} = pool(h_i, \ldots, h_{i+m-1}) \text{ and } h_{syn} = pool(\hat{h}_i, \ldots, \hat{h}_{i+m-1}).$$

This is similar to the operation in Equation (3.7) of the original GAT. The representations are then combined by a fusion mechanism based on gates which is defined as

$$h_f = g \odot h_{syn} + (1 - g) \odot h_{con} \text{ with } g = \sigma(W_g[h_{syn}; h_{con}] + b_g),$$

where W_g and b_g are weights and bias. For classification, the fused representation is fed to a linear and a softmax layer. Training is executed by minimizing L2-regularized cross-entropy loss.

4.1.8 Models based on Capsule Networks

Capsules Networks were proposed in [28,73] for the field of Computer Vision. In this context, so-called *capsules* are responsible for recognizing certain implicit entities in images. Each capsule does internal calculations and returns a probability that the corresponding entity appears in the image. The original CapsNet consists of two capsule layers: one to store low-level feature maps, the other to calculate classification probabilities where each capsule refers to one class. This model is able to memorize more information than a CNN and to control the amount of information flowing from one layer to another by so-called *dynamic routing*. For more detailed information, we refer to the papers mentioned initially. The following approaches have Capsules Networks as their basis and modify them for the ATSC task.

T(rans)Cap

Capsule Networks and Transfer Learning were combined in [8] to form their method called *Transfer Capsule Network (T(rans)Cap)*. Transfer Learning is employed by utilizing knowledge from document-level tasks $\mathcal{T}_D$ for the ATSC task $\mathcal{T}_A$. The model architecture, in particular the four layers, is depicted in Figure 36.

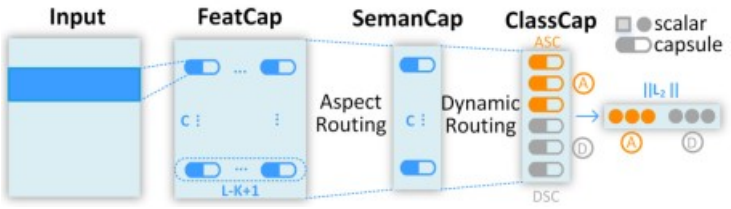


Figure 36: T(rans)Cap Architecture. *Source:* [8, p.549]

At first, the input is transformed into numerical vectors using two lookup layers. The first applies simple pretrained embeddings, in this case GloVe (cf. [62]), which are denoted as $\{e_1, \ldots, e_L\} \in \mathbb{R}^{d_w \times L}$. The second captures the absolute distance between each word and the aspect term. This only holds for $\mathcal{T}_A$, whereas these position embeddings $\{l_1, \ldots, l_L\} \in \mathbb{R}^{d_l \times L}$ for $\mathcal{T}_D$ are simply zeros. The final representation $X = \{x_1, \ldots, x_L\} \in \mathbb{R}^{d_h \times L}$ is the position-wise concatenation of both embeddings.

Then, the so-called *FeatCap* layer selects K-gram features from these embeddings and receives feature vectors or capsules $r \in \mathbb{R}^{d_p \times (L-K+1)}$ by applying convolutions:

$$r_i = X_{i:i+K} * F + b.$$

Here, $F \in \mathbb{R}^{d_p \times (d_h \times K)}$ stands for the kernel group with d_p convolutional kernels of size $(d_h \times K)$. Each kernel group F belongs to a semantic category, which is why C different kernel groups are utilized in the same way, leading to the output of the FeatCap layer $R = [r_1, \ldots, r_C] \in \mathbb{R}^{C \times d_p \times (L-K+1)}$.

In the *SemanCap layer*, so-called *aspect routing weights* a_i are calculated for the words in a K-sized window around an aspect term by

$$a_i = \sigma(X_{i:i+K} * F_a + T_a e_a + b_a).$$

This fusing convolution is only used for $\mathcal{T}_A$, with kernel $F_a \in \mathbb{R}^{d_h \times K}$, transfer matrix $T_a \in \mathbb{R}^{1 \times d_w}$, bias b_a and the pretrained embedding e_a for the aspect term. By this operation, the aspect information regarding its context is compressed. An aspect routing weight of zero means a blockage of the corresponding feature capsule. As document-level corpora $\mathcal{C}_D$ do not contain aspect terms, their aspect routing weight is always set to 1. In order to maintain a unified procedure, a helper function is defined by

$$g_i(X) = \begin{cases} a_i, & if\ X \in \mathcal{C}_A \\ 1, & if\ X \in \mathcal{C}_D \end{cases}$$

with $\mathcal{C}_A$ and $\mathcal{C}_D$ being the corpora of $\mathcal{T}_A$ and $\mathcal{T}_D$, respectively. Applying g_i to the whole sentence X results in $g \in \mathbb{R}^{1 \times (L-K+1)}$. As in the FeatCap layer, this procedure is repeated C times, yielding $G = [g_1, \ldots, g_C] \in \mathbb{R}^{C \times 1 \times (L-K+1)}$. With these weights, the feature capsules are routed to *aspect-customized feature capsules* $P = R \odot G \in \mathbb{R}^{C \times d_p \times (L-K+1)}$. As P does not capture sentence-level, but local features and it contains many capsules, the next layer may not learn robust representations. Thus, all feature capsules in the same channel are aggregated horizontally into so-called *semantic capsules*

$$U = \max_{t=1}^{C \times d_p} P_t.$$

The length of each semantic capsule u_i represents the probability that its semantic meaning occurs in the corresponding input. Thus, we apply the so-called *squash* function

$$squash(u_i) = \frac{||u_i||^2}{1 + ||u_i||^2} \frac{u_i}{||u_i||} \tag{4.7}$$

in order to have normalized results in $[0, 1]$.

In the last layer, called *ClassCap*, class capsules based on sentiment classes are built. In this layer, aspect-level and document-level tasks differ from each other. First, a prediction

vector of a semantic capsule i towards a class capsule j is calculated as

$$\hat{u}_{j|i} = W_{ij} u_i$$

with weights W_{ij}. Based on this equation, the dynamic routing procedure is executed r times, leading to a final representation for each class capsule j. It consists of the following steps applied in exactly this order. Starting with a coupling coefficient

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})}, \tag{4.8}$$

the vector representation of class capsule j is $s_j = \sum_i c_{ij} \hat{u}_{j|i}$. Applying Equation (4.7) on it, the final representation v_j of class capsule j is calculated. Then, the agreement coefficient $\alpha_{ij} = \hat{u}_{j|i} v_j$ is updated, which is needed for the log prior probability b_{ij} in Equation (4.8). It denotes the probability of a semantic capsule i passing to a class capsule j, thus indicating how intense their connection is. Initialized with zero, its update step is $b_{ij} \leftarrow b_{ij} + \alpha_{ij}$.

As we have already mentioned, the length of a class capsule represents the probability of the corresponding sentiment. This means that the active class should have the longest capsule. Therefore, for each class capsule j a margin loss is defined by

$$\mathcal{L}_j = Y_j \max(0, m^+ - ||v_j||)^2 + \lambda(1 - Y_j) \max(0, ||v_j|| - m^-)^2 \tag{4.9}$$

with $Y_j = 1$ if the sentiment is present in class capsule j, otherwise zero. Following [73], we have $m^+ = 0.9$, $m^- = 0.1$ and $\lambda = 0.5$. The loss for a single task $T \in \{A, D\}$ is $\mathcal{L}_T = \sum_{j=1}^{J} \mathcal{L}_j$ and the entire loss of the model is $\mathcal{L} = \mathcal{L}_A + \gamma \mathcal{L}_D$ with $\gamma \in [0, 1]$.

CapsNet(-BERT)

A variation of Capsule Networks and its combination with BERT was introduced in [34]. It is called *CapsNet(-BERT)* and its architecture is presented in Figure 37. It is possible to use CapsNet also for ACSC, yet here we focus on the ATSC variant only. CapsNet consists of four layers which are an embedding, an encoding and two capsule layers.

In the first one, the sentence is converted to embeddings, where aspect term embeddings a are the average of the embeddings of all aspect term words. The aspect-aware sentence embedding E^{sa} is defined as $E_i^{sa} = [E_i; a]$ with the concatenation operator $[;]$. These representations are turned into contextualized ones using bidirectional GRUs (see Section 3.3) and residual connections (see Section 3.4.2): $H = BiGRU(E^{sa}) + E^{sa}$.

In the primary capsule layer, primary capsules $P = [p_1, \ldots, p_n]$ and an aspect capsule c are defined as

$$p_i = squash(W^p h_i + b^p) \text{ and } c = squash(W^a a + b^a)$$

with weights W^p, W^a and biases b^p, b^a and the squash function from Equation (4.7). In order to avoid unstable training due to varying sentence lengths, *aspect aware normalization* is proposed by the authors. This also solves the problems of saturated squash activations with high confidence for all categories for long sentences and low confidence for short sentences. To do so, normalized primary capsule weights u are generated by applying softmax and weights W^n:

$$u_i = \frac{\exp(p_i W^n c)}{\sum_{j=1}^{n} \exp(p_j W^n c)}.$$

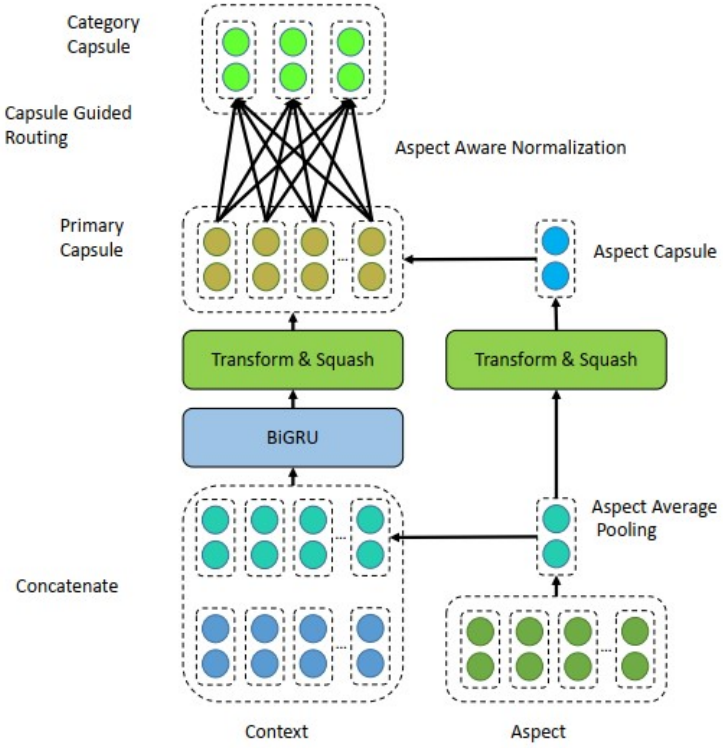


Figure 37: CapsNet Architecture. *Source:* [34, p.6282]

Instead of the dynamic routing mechanism which was proposed in [73] and used for T(rans)Cap in Section 4.1.8, a *capsule-guided routing mechanism* is applied. The reason for the substitution is that the interaction makes training inefficient and the routing process is not guided by any upper layer information. Here, prior knowledge about the sentiment categories is stored in sentiment capsules to direct the routing process. A

sentiment matrix $G \in \mathbb{R}^{C \times d}$ is initialized with averaged embeddings of sentiment words for C sentiment categories and d-dimensional sentiment embeddings. Sentiment capsules $Z = [z_1, \ldots, z_C]$ and routing weights w_{ij} are calculated with weights W^r via

$$z_i = squash(G_i)$$
$$w_{ij} = \frac{\exp(p_i W^r z_j)}{\sum_{k=1}^{n} \exp(p_i W^r z_k)}.$$

In the category capsule layer, the final category capsules are defined as

$$v_j = squash(s \sum_{i=1}^{n} w_{ij} u_i p_i),$$

where s is a trainable parameter scaling the connection weights. Based on them, the loss is calculated like for TCap beforehand, with the only difference that $\lambda = 0.6$. To create CapsNet-BERT, the embedding and encoding layers are substituted by a pretrained BERT model.

4.2 ATE+ATSC Methods

The models in this section are not only capable of predicting sentiments of aspects, but also of identifying the aspect terms themselves. They can be grouped into collapsed, joint and pipeline models depending on the labeling scheme (see Section 2.2). Their reported F1 Micro Scores are depicted in Figure 38 with those models highlighted that are going to be part of the practical analysis in Section 5. These are GRACE [53] for the pipeline and BERT+TFM [47] for the collapsed approaches. The graph shows that the topic of combined models is a rather young one, starting only in 2019. Additionally, the amount of models is much smaller than for the previous task. All the numbers are additionally listed in Table 6.

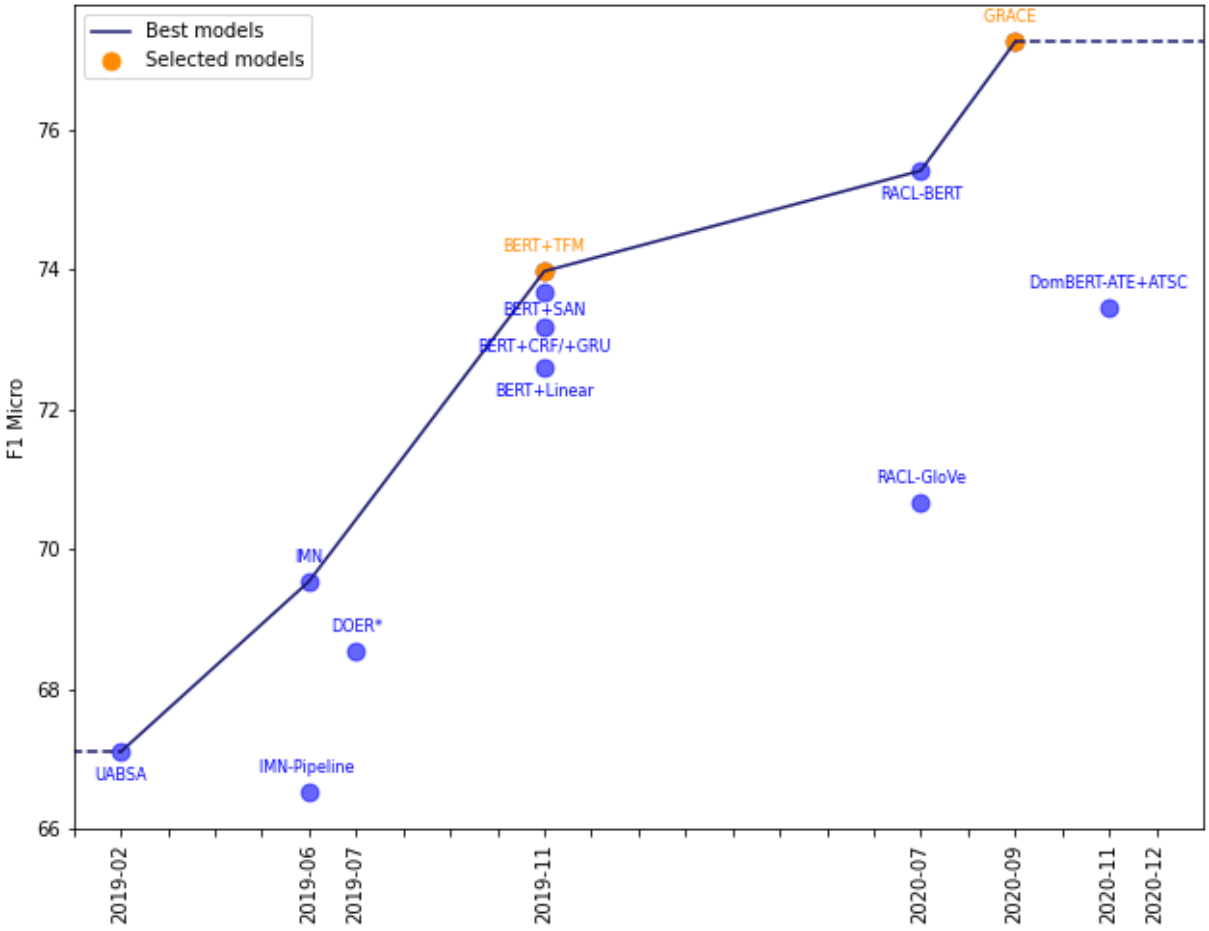


Figure 38: Development of reported ATE+ATSC F1 Micro Scores on SemEval-14 Restaurants. An asterisk indicates that the corresponding values are not taken from the original papers.

F1 MICRO		**Data Set**	SemEval-14	
		Domain	Restaurant	Laptop
Model Category	**Model**	**Value Calculation**		
Extra data	DomBERT-ATE+ATSC	mean of 10 runs	73.45	66.21
Pipeline Models	GRACE		**77.26**	**70.71**
	IMN-Pipeline (ATE + ATSC)	mean of 5 runs	66.53	56.02
	SPAN-Pipeline			68.06
Joint Models	LCM-Gate			62.42
	DOER	mean of 10 runs	68.55 [4]	60.35
	IMN (ATE + ATSC)	mean of 5 runs	69.54	58.37
	RACL-GloVe (ATE + ATSC)	mean of 5 runs	70.67	60.63
	RACL-BERT (ATE + ATSC)	mean of 5 runs	**75.42**	63.40
	SPAN-joint			**64.59**
Collapsed Models	UABSA*		67.10	57.90
	SPAN-collapsed			48.66
	BERT+Linear*	mean of 5 runs	72.61	60.43
	BERT+GRU*	mean of 5 runs	73.17	**61.12**
	BERT+SAN*	mean of 5 runs	73.68	60.49
	BERT+TFM*	mean of 5 runs	**73.98**	60.80
	BERT+CRF*	mean of 5 runs	73.17	60.78

Table 6: Reported F1 Micro Scores for 3-class ATE+ATSC. Sources different from the original papers are marked accordingly. For all models except from DomBERT, only the implementation indicated that the F1 Scores are calculated with "micro" average. Bold printed values indicate the best model of one category, underlined ones the best performance on the corresponding data set. An asterisk marks that values are taken from the GitHub repository instead of the paper due to corrected re-evaluation.

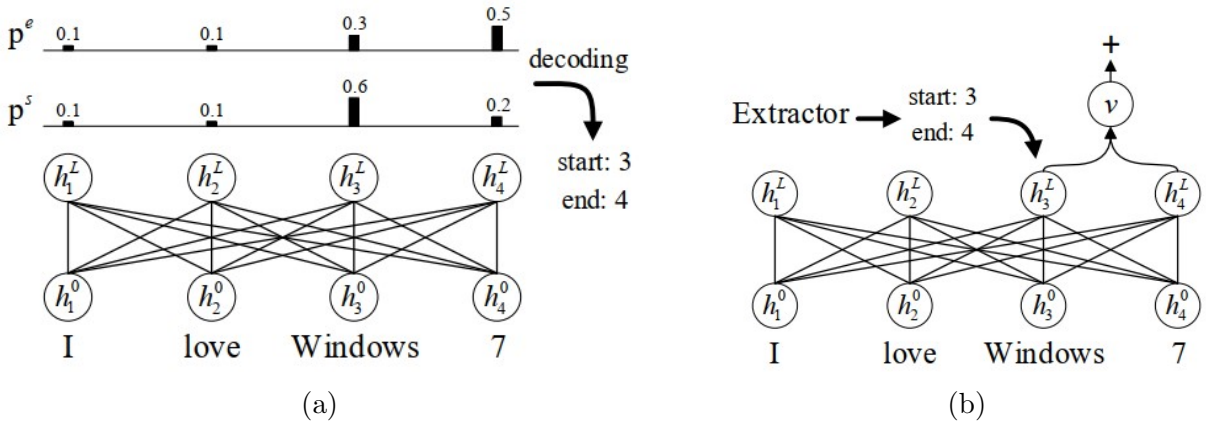
4.2.1 Models with a Pipeline Architecture

The characteristic of a pipeline model is that it starts with the ATE task and performs it completely before it continues with ATSC. Moreover, there are no information from ATSC used for ATE.

SPAN

The model introduced in [30] follows the idea of labeling word spans, more precisely their beginning and their end positions, instead of tagging words. Thus, it is simply called

SPAN. On the spans, a typical pipeline model is applied which first extracts the aspect terms and then classifies their polarities based on the span information. An advantage of this procedure is that all target words can be processed before the sentiment is predicted which avoids label inconsistency of multi-word aspect terms.



(a) (b)

Figure 39: Building blocks of SPAN: (a) Multi-target extractor (b) Polarity Classifier. *Source:* [30, p.3]

The specialized architecture is placed on top of a BERT model with L layers. Its output representations h^L are forwarded to a multi-target extractor which is shown in Figure 39a. Its goal is to extract aspect term candidates by indicating their start and end positions. The confidence scores for start and end positions are defined as

$$g^s = w_s h^L \text{ and } g^e = w_e h^L$$

with weights w_s, w_e. Their corresponding probability distributions p^s, p^e are calculated as the softmax of the scores. Aspect term extraction is trained by minimizing the summed negative log likelihood

$$\mathcal{L} = -\sum_{i=1}^{n+2} y_i^s \log(p_i^s) - \sum_{j=1}^{n+2} y_j^e \log(p_j^e).$$

A multi-span decoding algorithm is introduced to determine the final aspect term spans per sentence: The top-M indices from the scores g^s and g^e are identified at first and stored in S and E, respectively. Each pair $r_l = (s_i, e_j)$ denotes a candidate span and has its own heuristic-regularized score $u_l = g^s_{s_i} + g^e_{e_j} - e_j - s_i + 1$. Both r_l and u_l are added to the corresponding lists R and$U$, respectively. The end position has to appear later in the sentence than the start position and the sum of their scores has to be greater than a threshold γ. The span r_l with the maximum score u_l is removed from R and added to the

output set O. Any other span that overlaps with r_l is also discarded from R. We repeat this procedure until R is empty or the best K aspect term spans have been identified.

Within the polarity classifier (see Figure 39b), an attention mechanism is applied on a target span $r \in O$, modifying the attention Equations (3.3) and (3.4) like this:

$$\alpha = softmax(w_\alpha h^L_{s_i:e_j}) \text{ and } v = \sum_{t=s_i}^{e_j} \alpha_{t-s_i+1} h^L_t,$$

where weights are denoted by w_α. Consequently, the polarity score for an aspect term span is defined as

$$g^p = W_p \tanh(W_v v)$$

and the corresponding probability p^p as its softmax. Here, we have weights W_p, W_v. The polarity classifier is trained to minimize

$$\mathcal{J} = -\sum_{i=1}^{k} y_i^p \log(p_i^p),$$

where k denotes the number of sentiment classes and y_i^p the true sentiment.

The authors did not only set up a pipeline approach, but they also investigated slightly different variants for joint and collapsed labeling. For the Pipeline approach, both models are trained independently by minimizing $\mathcal{L}$ in the first step and $\mathcal{J}$ in the other. Then they are stacked one after another for inference. In contrast to that, the joint variant incorporates a shared BERT model with two separate output layers. It is trained by minimizing the sum $\mathcal{L} + \mathcal{J}$. For the collapsed model, the labels are aggregated into (*start position sentiment*, *end position sentiment*). For the example of Figure 39, this would be $(3+, 4+)$, indicating that the aspect term consists of words three and four and that it has a positive sentiment. The multi-target extractor outputs three sets of probabilities for start and end positions, one for each sentiment. Also one optimization objective for each sentiment class is defined. At inference time, on each set of scores the multi-span decoding algorithm is applied and the final prediction is received by merging the output sets O^+, O^- and O^0 of the three polarities. The building blocks of SPAN can also be used separately for the single tasks ATE and ATSC.

MTL

Another model that can not only be a pipeline approach but also a collapsed one was proposed in [3]. As it is entitled with *Multi-task learning*, we refer to these models as MTL-Pipeline and MTL-collapsed in order to be consistent with our terminology, whereas the authors named them end-to-end and joint. In contrast to the other pipeline approaches, MTL is not based on BERT.

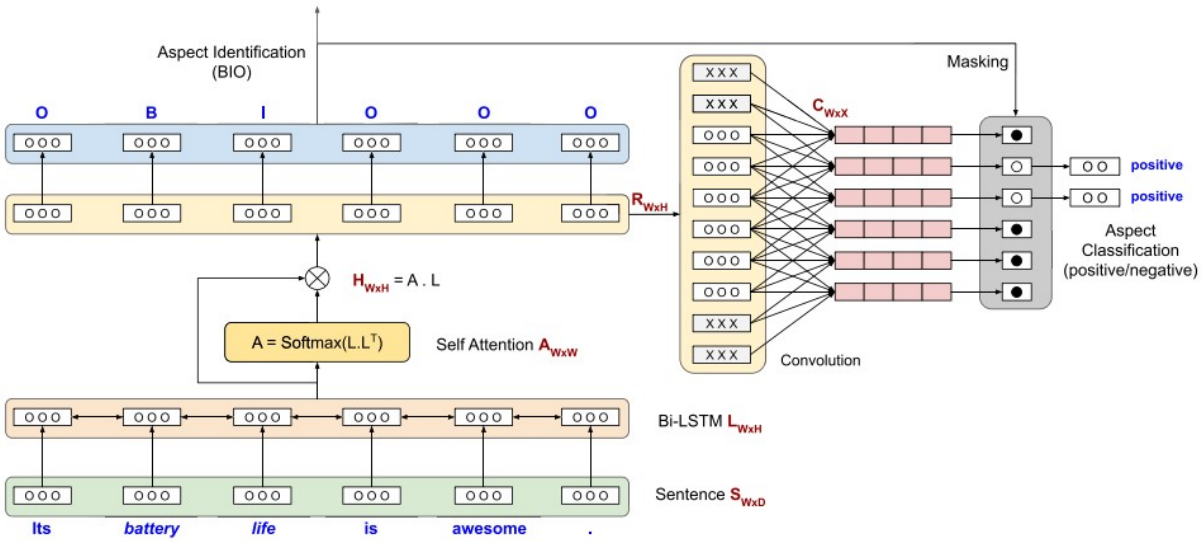


Figure 40: MTL-pipeline Architecture. *Source:* [3, p.250]

Figure 40 shows the architecture of the pipeline model. At first, the input sentence is embedded and fed to a BiLSTM. After that, a self-attention mechanism is applied on the BiLSTM outputs L to learn the relationship between words or tokens. This helps to label the beginning, inside and outside an aspect term correctly. The attention weights of Equation (3.3) are calculated as $A = softmax(LL^\top)$ and the attention outputs as $R = AL$. Then, a softmax layer is used for the final aspect term classification with labels B, I, O: $Prediction^{Aspect} = softmax(R)$.

The second part, ATSC, is built upon a CNN layer that takes the attention outputs R as its input: $C = Conv1D(R)$. The labels from the ATE task are utilized as masks, because only B- and I-labeled words should receive a sentiment tag. It is calculated as a softmax about the masked convolution outputs. Possible sentiments are negative, neutral, positive and conflict.

For the collapsed approach, the two tasks are merged into a single sequence labeling task, yet no details about the exact procedure are given. For both types of models, however,

in case the sentiment for a multi-word aspect term is not clear, the sentiment with the maximum count in the sentence is chosen. If there is a tie, the sentiment of the first token is taken.

GRACE

GRACE, a *Gradient Harmonized and Cascaded Labeling* model in [53], also belongs to the group of pipeline approaches. The aspect term label set is $T^e = \{B, I, O\}$ and the sentiment label set T^c consists of positive, negative, neutral, conflict and other. GRACE's architecture is shown in Figure 41. The left block depicts a pretrained BERT model which shares its first l layers between the ATE and the ATSC task. The remaining layers $l+1, \ldots, L$, where L is the maximum number of BERT layers, are only used for ATE. They are followed by a classification layer for aspect terms. These classification outputs then are the inputs of a Transformer decoder which does sentiment classification (right block). The decoder differs from the original one in [80] due to applying multi-head attention without masking in its first sublayer. Its second sublayer takes the representations of BERT's l-th layer as key and value vectors. The outputs of the decoder, the new sentiment representations, are denoted as G_c. The principle of using the first set of labels as input for the second is called *Cascaded labeling*. It is their way to deal with interactions between different aspect terms and the reason for us to categorize GRACE as a pipeline model.

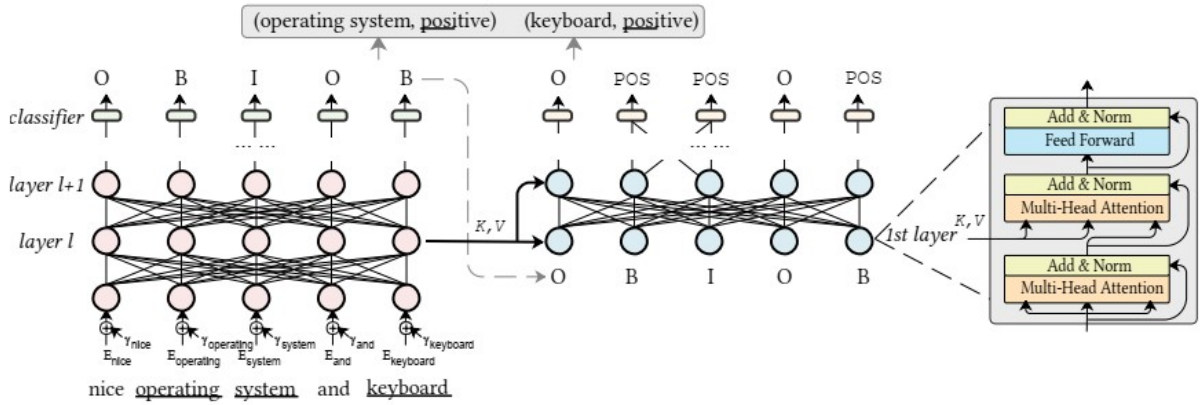


Figure 41: GRACE Architecture. *Source:* [53, p.3]

The authors experimented with more extensions to deal with some general issues. Their ablation studies show that it is best to include them in the model. A new feature they introduced is the modification of the loss: They include gradient harmonization to cope with imbalanced labels during training. Actually, the label "O" appears more often than

the other labels. So, instead of the basic cross-entropy loss

$$\mathcal{L}_\tau = -\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{t_i^\tau \in T^\tau\}} (\log p_i^\tau)^\top \text{ with}$$
$$p^\tau = softmax(M^\tau w^\tau) \text{ and } M^\tau = \begin{cases} H^L, \ if & \tau = e, \\ G_c, \ if & \tau = c \end{cases},$$

with $\tau \in \{e, c\}$ indicating the ATE (e) or ATSC (c) task and trainable weights w^τ, they implemented the following loss:

$$\mathcal{L}_\tau = -\frac{1}{n}\sum_{i=1}^{n} \beta_{t_i^\tau} \mathbb{1}_{\{t_i^\tau \in T^\tau\}} (\log p_i^\tau)^\top \text{ with } \beta_{t_i^\tau} = \frac{N^\tau}{\rho(g_{t_i^\tau})}.$$

The total number of labels is N^τ and the gradient norm $g = |\frac{\partial \mathcal{L}}{\partial z}| = |p - \hat{t}|$. Moreover, z is the output of the model, $\mathcal{L}$ is the cross-entropy loss, $\hat{t}$ are the true labels and p are the predictions. The gradient density ρ is approximated in a more complicated manner for which we refer to [53].

Additionally, *Virtual Adversarial Training (VAT)* is used to make GRACE more robust to input noise. Small perturbations r are added to input embeddings E which is also shown in Figure 41. The corresponding loss is computed via

$$\begin{aligned} \mathcal{L}_{VAL} &= \frac{1}{n}\sum_{i=1}^{n} D_{KL}\ (p(\cdot|E;\Theta)\ ||\ (p(\cdot|E^*;\Theta))) \text{ with} \\ E^* &= E + r,\ r = \epsilon g/||g||_2, \\ g &= \nabla_{E'} D_{KL}\ (p(\cdot|E;\hat{\Theta})\ ||\ (p(\cdot|E';\hat{\Theta}))), \\ E' &= E + \xi d, \end{aligned}$$

where $D_{KL}(p(\cdot|\cdot)||(p(\cdot|\cdot)))$ is the Kullbach-Leibler divergence of a conditional probability p, Θ the current parameters with $\hat{\Theta}$ being a corresponding constant set, ϵ and ξ are hyperparameters and $d \sim \mathcal{N}(0, I)$. Finally, both losses for ATE and ATSC and the VAT-loss are summed up to the overall loss for training.

Furthermore, post-training was applied on the pretrained BERT model using Amazon (cf. [26]) and Yelp[3] data sets, just like the authors of BERT-PT and DOER did. They also added a mechanism to create a consistent polarity label. The underlying problem is that multi-word aspect terms may receive contradicting sentiments. As experiments did

[3] `https://www.yelp.com/dataset`

not prove an increasing performance, we refer the reader to the details in the paper. Note that GRACE can be also employed to carry out the ATE task only.

4.2.2 Models with a Joint Labeling Scheme

As its characteristic, joint labeling has two separate lists of labels indicating aspects and polarities. Some of the following models can not only be employed on the combined task, but also perform ATSC alone. For them, F1 Micro Scores are given in Table 7.

F1 MICRO		**Data Set**	SemEval-14	
		Domain	Restaurant	Laptop
Model Category	**Model**	**Value Calculation**		
Joint Models	DOER (ATSC)		64.50 [4]	60.18 [4]
	RACL-GloVe (ATSC)	mean of 5 runs	74.46	71.09
	RACL-BERT (ATSC)	mean of 5 runs	**81.61**	**73.91**

Table 7: Reported F1 Micro Scores for 3-class-ATSC. Sources different from the original papers are marked accordingly. Bold printed values indicate the best model.

IMN

The *Interactive Multi-Task Network*, called IMN, was proposed in [25]. Its general structure relies on CNNs and it is shown in Figure 42. Besides ATE and ATSC, two document-level tasks are included as well, which resembles the idea of TCap (see Section 4.1.8).

An input sentence is first fed to a feature extractor f_{θ_s}, where the words are embedded and then forwarded to m^s CNN layers. This feature extractor is shared by all tasks. Its output is a vector $h^{s(t)}$, where $t = 0$ right after the initialization, and it serves as input for the specific tasks. The important ones for us are ATE (here: AE) and ATSC (here: AS) and they are summarized as aspect-level tasks. The goal of the first one is to find all aspect and opinion terms and mark them with the labels from $Y^{ae} = \{BA, IA, BP, IP, O\}$ indicating beginning and inside of aspect or opinion terms, or other.

The ATE feature extractor $f_{\theta_{ae}}$ consists of m^{ae} CNN layers whose outputs h_i^{ae} are concatenated with the word embeddings and the initial representations $h_i^{s(0)}$. They are then fed to a decoder, which is basically a fully-connected layer and a softmax for classification. The ATSC feature extractor is similar, yet there are the following differences: In addition

to the m^{as} CNN layers, there is a self-attention layer, where the model can access the outputs $\hat{y}^{ae}$ from the ATE block. The self-attention matrix A is computed as softmax of the following scores:

$$score_{ij}^{(i \neq j)} = (h_i^{as} W^{as} (h_j^{as})^\top) \frac{1}{|i-j|} P_j^{op},$$

where P_j^{op} is the predicted probability of the j-th word being part of an opinion term. It is the sum of the predicted probabilities of words with BP or IP labels in y_j^{ae}. Furthermore, W^{as} and h_i^{as} are weights and h_j^{as} are the outputs of the ATSC-CNNs. The diagonal elements of A are zeros. The outputs of the self-attention layer are $h_i'^{as} = \sum_{j=1}^{n} A_{ij} h_j^{as}$. Concatenated with $h_i^{s(0)}$, they form the final task-specific representation that is fed to the same decoder as ATE. For multi-word aspect terms, the predicted label of the first token determines the polarity of the whole aspect term.

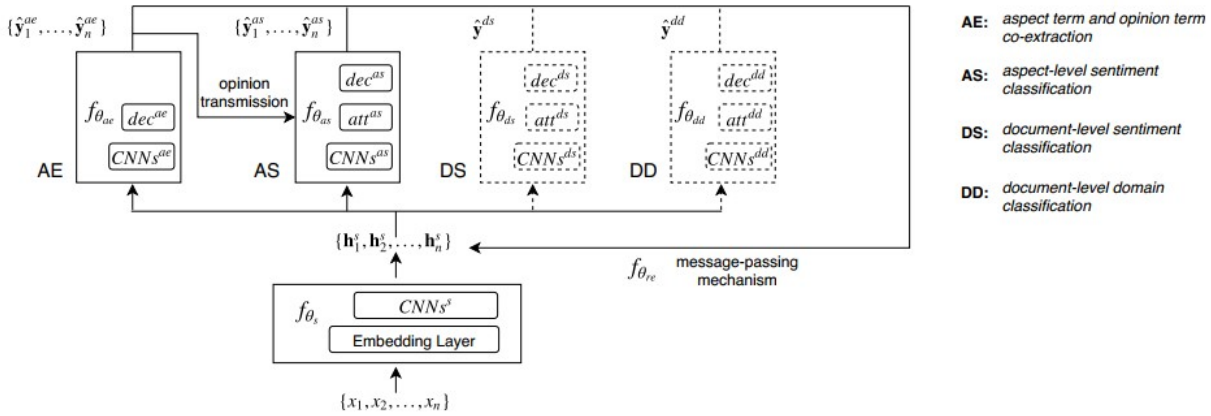


Figure 42: IMN Architecture. *Source:* [25, p.3]

As already mentioned, document-level tasks are included to alleviate the problem of not enough training data. These are document-level sentiment (DS) and domain (DD) classification. Their blocks have the same structure as the ATSC block, i.e. m^o CNN layers with outputs h_i^o followed by an attention layer and the same decoder as above. The attention weights are calculated as $a_i^o = softmax(h_i^o)$ and lead to the final document representation $h^o = \sum_{i=1}^{n} a_i^o h_i^o$. Throughout the whole procedure, $o \in \{ds, dd\}$ indicates the corresponding document-level task.

A novelty of IMN is the *message passing mechanism* which uses aggregated predictions of different tasks of previous layers to update the shared representations $h^{s(t)}$. So, the new hidden representation is given by

$$h_i^{s(t)} = f_{\theta_{re}} \left(h_i^{s(t-1)}; \hat{y}_i^{ae(t-1)}; \hat{y}_i^{as(t-1)} \right); \hat{y}_i^{ds(t-1)}; a_i^{ds(t-1)}; a_i^{dd(t-1)}$$

with the concatenation operator [;] and the fully-connected layer plus ReLU activation $f_{\theta_{re}}$.

IMN is trained alternating between aspect- and document-level instances after some epochs of pretraining with document-level tasks only. The loss for aspect-level instances is

$$\mathcal{L}(\theta_s, \theta_{ae}, \theta_{as}, \theta_{ds}, \theta_{dd}, \theta_{re}) = \frac{1}{N_a} \sum_{i=1}^{N_a} \frac{1}{n_i} \sum_{j=1}^{n_i} l(y_{i,j}^{ae}, \hat{y}_{i,j}^{ae(T)}) + l(y_{i,j}^{as}, \hat{y}_{i,j}^{as(T)})$$

with the maximum number of iterations T of the message passing mechanism, the number of aspect-level training instances N_a, n_i tokens in the i-th sequence and gold labels y_i. Cross-entropy loss is denoted by l. For document-level instances, this loss is used:

$$\mathcal{L}(\theta_s, \theta_{ds}, \theta_{dd}) = \frac{1}{N_{ds}} \sum_{i=1}^{N_{ds}} l(y_i^{ds}, \hat{y}_i^{ds}) + \frac{1}{N_{dd}} \sum_{i=1}^{N_{dd}} l(y_i^{dd}, \hat{y}_i^{dd}).$$

Analogously, N_{ds} and N_{dd} denote the number of document-level training instances for DS and DD and y_i are the corresponding gold labels. Note that the message passing mechanism is not used for document-level instances.

DOER

In contrast to IMN which relies on CNNs, an RNN-based approach named *Dual cross-shared RNN (DOER)* was proposed in [54]. It can also perform both ATE and ATSC tasks. Designed as two sequence labeling problems, to each word of the input an aspect term tag out of $\{B, I, O\}$ and a sentiment tag from the set $\{NEG, POS, NEU, CON, O\}$ should be assigned. Predictions are made by executing the six steps of Figure 43.

At first, *double embeddings* (cf. [88]) are applied on the input. They consist of general-purpose h_g and domain-specific embeddings h_d which are simply concatenated. These embeddings are then fed to stacked dual RNNs (i.e. one stacked RNN for ATE and ATSC each) having so-called *Bidirectional Residual Gated Units (BiReGU)* as layers. Figure 44a shows how a ReGU works which is similar to, yet different from GRU and LSTM (see Section 3.3): A forget gate $f_t = \sigma(W_f x_t + U_f c_{t-1})$ controls how much information from the previous memory state c_{t-1} is forwarded to the current one

$$c_t = (1 - f_t) \odot c_{t-1} + f_t \odot \tanh(W_i x_t),$$

where W_f, U_f and W_i are weights. This means that the forget gate is responsible for the

information flow between two time steps. A residual gate $o_t = \sigma(W_o x_t + U_o c_{t-1})$ decides how much information from c_t is taken to the new hidden state h_t:

$$h_t = (1 - o_t) \odot c_t + o_T \odot \tilde{h}_t.$$

For the candidate state $\tilde{h}_t$, we have $\tilde{h}_t = x_t$ if the sizes of c_t and x_t are the same and $\tilde{h}_t = \tanh(W_x x_t)$ otherwise. The residual gate thus controls the information processing between two layers. Note that σ can be any logistic function. The output of a BiReGU is the concatenation of the last hidden states $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ of the unidirectional ReGUs.

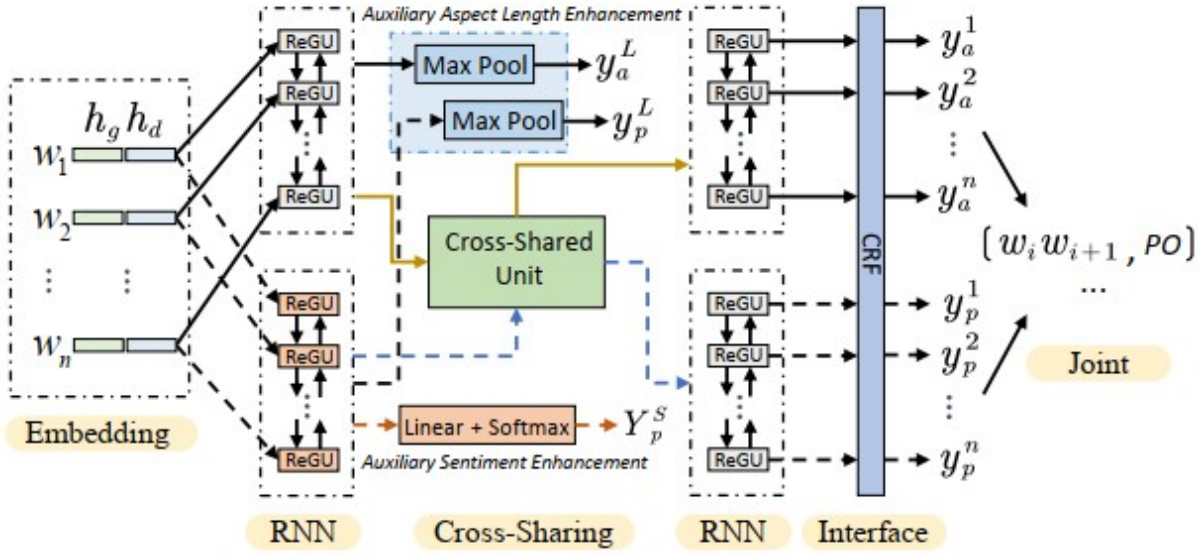


Figure 43: DOER Architecture. *Source:* [54, p.593]

After the RNNs, the interaction between ATE and ATSC is determined by a *Cross-shared Unit (CSU)* which is shown in Figure 44b. First, a composition vector is calculated by

$$\alpha_{ij}^M = f_m(h_i^m, h_j^{\bar{m}}) = \tanh((h_i^m)^\top G^m h_j^{\bar{m}}),$$

where $M \in \{A, P\}$ and $m, \bar{m} \in \{a, p\}$ are the indices of ATE (A,a) and ATSC (P,p) and m and $\bar{m}$ are always the opposite of each other. Weights are denoted by G^m. Based on this composition vector, the scalar attention score is

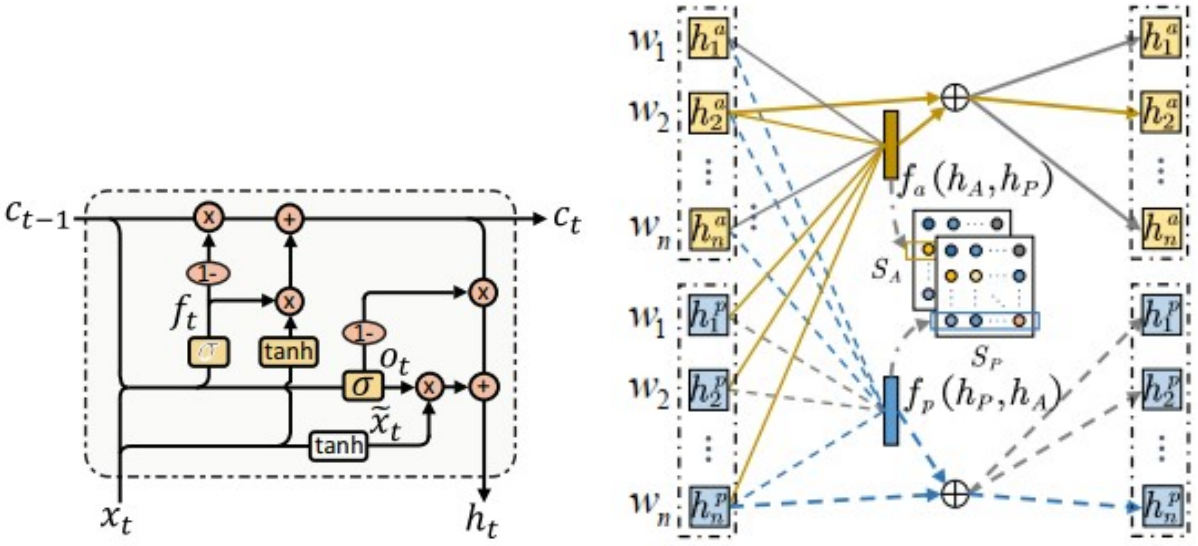
$$S_{ij}^M = v_m^\top \alpha_{ij}^M$$

with weight v_m. The higher the S_{ij}^A score, the higher the correlation is between an aspect term i and the sentiment representation of the word with index j. A higher S_{ij}^P score states the contrary, which is a higher correlation between the polarity of i and the aspect

term representation based on the j-th word. These attention scores are used to improve the representations of ATE and ATSC by

$$h_M = h_M + softmax(S^M)h_{\bar{M}},$$

which are again fed to an RNN layer. Then, in the inference layer, a linear-chain CRF (see Section 3.1) is applied and for the generation of labels the Viterbi algorithm (cf. [19]) is used. To output joint labels, the aspect term labels work as boundaries for the sentiment labels. In case of multi-word aspect term, the most frequent polarity tag is taken for the whole aspect term or - if there is a tie - the label of the first token (just like in MTL).



(a) (b)

Figure 44: Building blocks of DOER: (a) Cross-shared Unit (b) Residual Gated Unit. *Source:* [54, p.593]

Additionally, two auxiliary tasks are performed to improve predictions. The first is *Auxiliary Aspect Length Enhancement* (see the blue box in Figure 43) and it is about predicting the average aspect term length during training. To do so, the output of the first RNN layer $h_A^{l_1}$ is turned into $\tilde{h}_A$ by max pooling. The prediction of the average aspect term length is

$$z_{u_A} = \sigma(W_{u_A}^\top \tilde{h}_A)$$

with a weight w_{u_A}. Predictions are trained by minimizing the corresponding loss

$$\mathcal{L}_{u_A} = ||z_{u_A} + \hat{z}_u||^2 \tag{4.10}$$

with $\hat{z}_u$ being the average length of an aspect term after applying global normalization.

The same procedure is used for ATSC with the corresponding loss $\mathcal{L}_{u_P}$.

The other auxiliary task is *Auxiliary Sentiment Enhancement* and it is depicted by the red box in Figure 43. Here, a sentiment lexicon is used to train a classifier to assign a positive, negative or no sentiment to each word. The predictions are made with this formula

$$z_i^s = softmax(W_s^\top h_i^{p,l_1}),$$

where h_i^{p,l_1} is the ATSC output of the first RNN layer and W_s are weights. This classifier is trained by minimizing the cross-entropy loss for each sentence:

$$\mathcal{L}_s = -\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{\hat{y}_i^S \in \hat{Y}^P\}}\ log(z_i^S)^\top. \tag{4.11}$$

Consequently, the complete model is trained by minimizing the joint loss with L2 regularization, i.e.

$$\mathcal{L} = \mathcal{L}_a + \mathcal{L}_p + \mathcal{L}_{u_A} + \mathcal{L}_{u_P} + \mathcal{L}_s + \frac{\lambda}{2}||\Theta||^2,$$

where Θ denotes the parameter set and λ the regularization parameter. The first two losses are the negative log likelihoods for each of the tasks, whereas the other losses come from Equations (4.10) and (4.11).

RACL

Like IMN, [9] introduced an approach which is able to perform ATE, ATSC and joint modelling. It also divides the overall ABSA task into the three subtasks aspect term extraction (ATE, here: AE), opinion term extraction (OE) and aspect term sentiment classification (ATSC, here: SC). Their method is called *Relation Aware Collaborative Learning (RACL)* model and shown in Figure 45a.

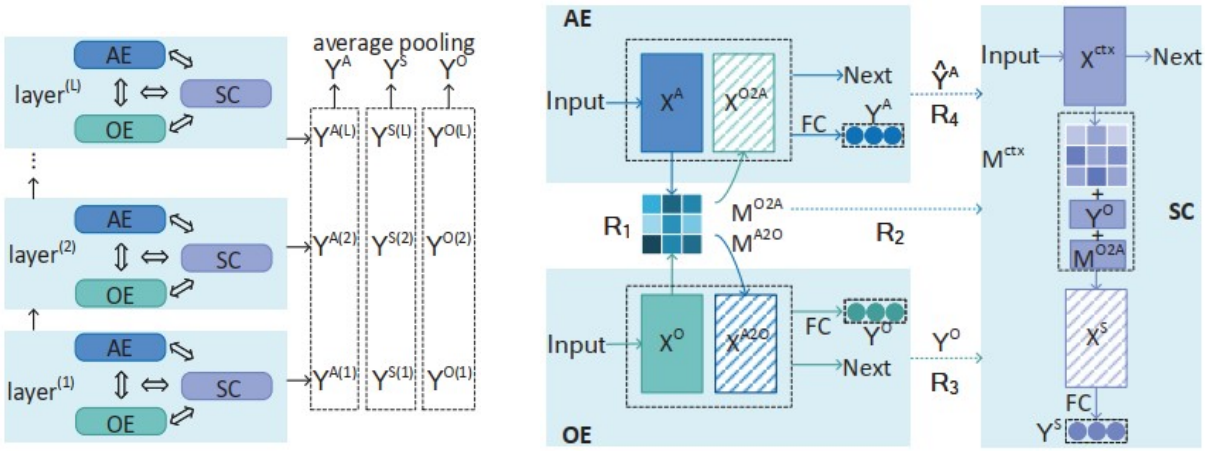
As depicted in Figure 45b, a single RACL layer consists of three modules, one for each subtask. They share the same input representations $E = \{e_1, \ldots, e_n\} \in \mathbb{R}^{d_w \times n}$ which are either GloVe embeddings (cf. [62]) or the outputs of a pretrained BERT model. Thus, the final models are either called RACL-GloVe or RACL-BERT. Following the shared-private scheme of [12,51], the input representations are transformed into task-invariant and task-specific features. The first ones, denoted as $H = \{h_1, \ldots, h_n\} \in \mathbb{R}^{d_h \times n}$, are received by applying a fully-connected layer on each embedded sentence E. The AE-oriented features $X^A \in \mathbb{R}^{d_c \times n}$ and OE-oriented features $X^O \in \mathbb{R}^{d_c \times n}$ are constructed by feeding H to two CNNs F^A and F^O separately. For the SC task, contextual features are encoded as

$X^{ctx} = F^{ctx}(H) \in \mathbb{R}^{d_h \times n}$ with another CNN F^{ctx}. In order to get the semantic relation between aspect terms and their context, an attention mechanism is used with h_i as query vector. The relevance scores here indicate the strength of the dependency between a query word i and a context word j and they are calculated as

$$ds_{i,j}^{(i \neq j)} = ((h_i)^\top \times X_j^{ctx})(\log_2(2 + |i - j|))^{-1}.$$

The second factor, a modification of the absolute distance between two words, is added to give words closer to the aspect term higher weights. Following Equation (3.3) the attention weights $M_{i,k}^{ctx}$ are computed using the dependency strength. Analogously to Equation (3.4), the attention output representing the SC-oriented features is

$$X_i^S = \sum_{j=1}^{n} M_{i,j}^{ctx} X_j^{ctx}. \tag{4.12}$$



(a) (b)

Figure 45: Architecture of RACL: (a) General Overview (b) A single RACL layer. *Source:* [9, p.3688]

In order to profit from relationships between the subtasks, the authors proposed four interactive relations which are also part of Figure 45b. The dyadic relation between AE and OE is modelled by R_1. So-called *useful clues* X_i^{O2A} from OE to AE and, vice versa, X_i^{A2O} can be extracted with a similar attention mechanism as was used for the SC-oriented features. To do so, the semantic relation between words in AE and OE is calculated via

$$sr_{i,j}^{(i \neq j)} = (X_i^A)^\top \times X_j^O \tag{4.13}$$

and normalized with respect to j via Equation (3.3) to obtain $M_{i,j}^{O2A}$. The useful clues

$X_i^{O2A} = \sum_{j=1}^{n} M_{i,j}^{O2A} X_j^O$ are concatenated with the AE-oriented features X^A via $[;]$ and turned into aspect term tag predictions by

$$Y^A = softmax(W^A[X^A; X^{O2A}])$$

with a transformation matrix $W^A \in \mathbb{R}^{3\times 2d_c}$. The opinion tags Y^O are predicted completely in the same manner using the transposition of Equation (4.13). As a word w_i can only be an aspect term or an opinion term in a certain sentence, this has to be assured of in the loss function

$$\mathcal{L}^R = \sum_{i=1}^{n} \max\left(0, \mathbb{P}(y_i^A \in \{B, I\}) + \mathbb{P}(y_i^O \in \{B, I\}) - 1.0\right). \tag{4.14}$$

The second relation R_2 is the triadic one between SC and R_1, denoting the influence of R_1 to SC as

$$M_{i,k}^{ctx} \leftarrow M_{i,k}^{ctx} + M_{i,k}^{O2A},$$

where $M_{i,k}^{O2A}$ represents R_1. The difference between $M_{i,k}^{ctx}$ and $M_{i,k}^{O2A}$ is that the first models the dependency between aspect terms and their contexts from the point of view of sentiment classification and the latter from the point of view of ATE.

Relation R_3 is located between SC and OE, giving higher weights to extracted opinion terms during sentiment prediction. This works similarly to R_2 by updating $M_{i,k}^{ctx}$, here with the predictions Y^O:

$$M_{i,k}^{ctx} \leftarrow M_{i,k}^{ctx} + \mathbb{P}(y_i^O \in \{B, I\})(\log_2(2 + |i - j|))^{-1}.$$

After this update step, Equation (4.12) should be re-evaluated. Now sentiments for identified aspect terms can be predicted by

$$Y^S = softmax(W^S[H; X^S])$$

with a transformation matrix $W^S \in \mathbb{R}^{3\times 2d_h}$.

The last relation R_4, taking place between AE and SE, works with AE results to supervise SC training. Actually, the true aspect term tags $\hat{Y}^A$ are incorporated in the following way:

$$y_i^S \leftarrow \mathbb{1}(\hat{y}_i^A) y_i^S.$$

If word w_i is an aspect term, the indicator functions evaluates to 1.

In order to capture sufficient features of the input, several RACL layers are stacked as

shown in Figure 45a. In the first layer, $X^{ctx(1)}$, $[X^{A(1)}; X^{O2A(1)}]$ and $[X^{O(1)}; X^{A2O(1)}]$ are encoded. They serve as input for the second layer, where $X^{ctx(2)}$, $X^{A(2)}$ and $X^{O(2)}$ are calculated. This procedure is repeated until the last layer L is reached. Then, the final predictions are obtained by average pooling over all layers:

$$Y^T = avg([Y^{T(1)}, \ldots, Y^{T(L)}]$$

with $T \in \{AE, OE, SC\}$.

For each subtask, the cross-entropy loss is defined as

$$\mathcal{L}^T = -\sum_{i=1}^{n}\sum_{j=1}^{J} \hat{y}_{ij}^T log(y_{ij}^T)$$

with sequence length n, J label categories and ground truth labels $\hat{y}_{ij}^T$. Together with the Hinge loss of Equation (4.14), the subtask losses form the combined loss $\mathcal{L} = \sum_{T\in\{AE,OE,SC\}} \mathcal{L}^T + \lambda\mathcal{L}^R$ which is to be minimized in order to train the model.

LCM

Some issues of existing models were raised in [82]. They introduced an approach with separated sets of labels that are used to correct each other if they do not agree. This is why their model is called *Label Correction Model (LCM)*. The aspect boundary labels Y^B follow the BIOES-scheme which is different from the approaches we encountered so far. The sentiment label set Y^S contains positive, negative, neutral and outside.

The architecture has three building blocks as shown in Figure 46: Encoder, Add/Gate Mechanisms and Conditional Random Fields (CRFs, see Section 3.1). The input sentences embedded with WordPiece (cf. [86]) are fed to BERT which serves as encoder resulting in representations H^L. Both sentiment and aspect boundary label sequences of the input are also turned into embeddings H^S and H^B, respectively. For a simpler notation, we write H^{set} with $set \in S, B$. These embeddings can be used together with the hidden states H^L in two different ways to calculate combined representations of input labels and sequences. For the *Add mechanism*, H^L and H^{set} are simply added, yielding H_{add}^{Lset}, while the *Gate mechanism* is more complex. First, a so-called gate matrix G^{set} is calculated by

$$G^{set} = \sigma(H^L * H^{set}),$$

where σ stands for the sigmoid activation. The gate matrix is then used to compute

combined representations

$$H_{gate}^{Lset} = H^L \odot G^{set} + H^{set} \odot (1 - G^{set}).$$

This means that for each entry in H_{gate}^{Lset}, H^L and H^{set} are combined in a weighted sum. A high value of the so-called *2-Norm Gate Value* $||G_i^{set}||_2^3$ indicates that more information from H^L is forwarded to the combined representation, i.e. predictions are made based on more contextual information. On the contrary, the lower the 2-Norm Gate Value, the more the other label type is integrated into predictions as the context alone does not provide enough information.

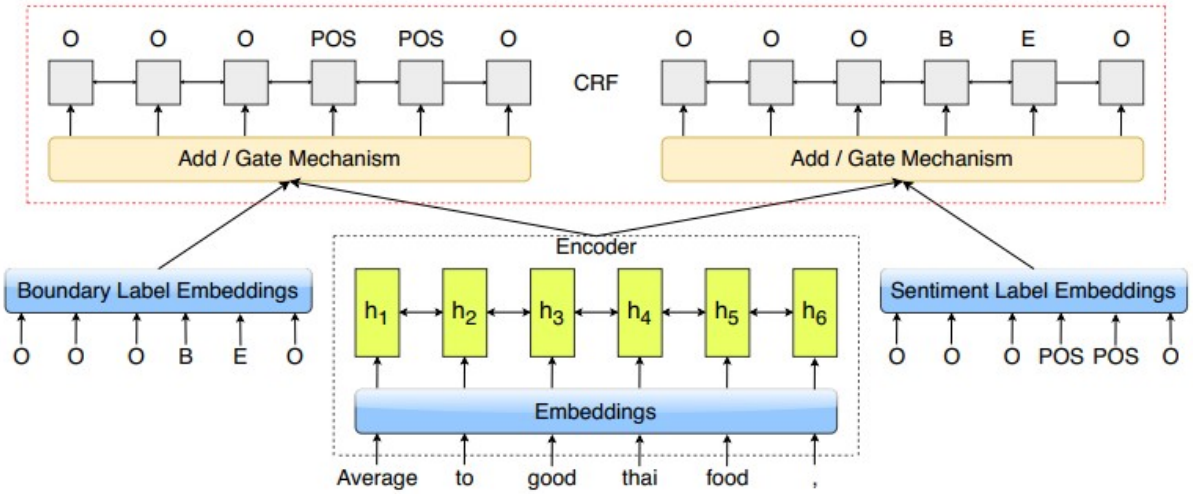


Figure 46: LCM Architecture. *Source:* [82, p.823]

For label predictions, two CRFs are applied on H_{add}^{Lset} or H_{gate}^{Lset} that output the entire label sequences at once. During training, the log-likelihood $\mathcal{L} = \mathbb{P}(Y^B|X) + \mathbb{P}(Y^S|X)$ is to be maximized. For test-time predictions, the Viterbi algorithm (cf. [19]) is applied to obtain the optimal labels. Note that the Add/Gate mechanism and CRFs are run once during training, but can be repeated several times for testing. For training, the model uses the real sentiment labels and the hidden state of the sequence to predict the aspect boundary label sequence. At test time, only the hidden states are used for the first predictions, which in turn are then additional input for further predictions.

4.2.3 Models with a Collapsed Labeling Scheme

What tells collapsed models from joint models apart is the unified tagging scheme that incorporates both aspect term tags and sentiments labels in one set of labels. We already

got to know some approaches that incorporate a collapsed variant, yet in this section we are going to see collapsed models only.

BERT-E2E-ABSA

A rather simple way of performing ATE+ATSC with collapsed labels is stacking a classification layer on top of BERT. This is the approach followed in [47] and officially called *BERT-E2E-ABSA*. Depending on the top layer, we name it *BERT+X*.

BERT+Linear BERT+Linear simply adds a linear layer and a softmax on the BERT representations. On the following three models, exactly these two layers are placed on top for classification.

BERT+GRU In BERT+GRU, the task-specific representations are calculated by a GRU (see Section 3.3) applied on BERT's output. Layer normalization is added after each multiplication with weights to stabilize training.

BERT+SAN Self-Attention is added to BERT in BERT+SAN, where keys, queries and values are weighted products of the BERT representations. The output is a layer-normalized sum of the BERT outputs and the self-attention.

BERT+TFM Also BERT+TFM is based on the addition of self-attention, more precisely of another transformer layer. This means that the representations of BERT-SAN $\hat{H}^L$ are fed into a point-wise feed-forward network and its outputs are summed up with $\hat{H}^L$. Again, layer normalization is applied.

BERT+CRF BERT+CRF employs a linear-chain CRF layer on BERT's representations. The goal of a CRF layer (see Section 3.1) is to identify the globally most likely tag sequence. To do so, sequence-level scores are defined as

$$s(x, y) = \sum_{t=0}^{T} M^A_{y_t, y_{t+1}} + \sum_{t=1}^{T} M^P_{t, y_t}$$

and the likelihood $p(y|x)$ is calculated as their softmax over all possible tag sequences. These scores correspond to the weighted sum of feature functions F_k in Equation (3.1). The transition matrix M^A indicates how strongly adjacent predictions are dependent from each other and M^P denotes the linear transformation of BERT representations. The sequence with the maximum score works as the output of the model; it is received by applying the Viterbi algorithm (cf. [19]).

UABSA

A more complicated collapsed approach is a *Unified Model for ABSA (UABSA)* in [46]. It consists of two stacked LSTMs cells $LSTM^{\mathcal{T}}$ and $LSTM^{S}$ and three special components for *Boundary Guidance (BG)*, *Sentiment Consistency (SC)* and *Opinion-Enhanced Target Word Detection (OE)*. Figure 47 shows how these building blocks are connected to each other.

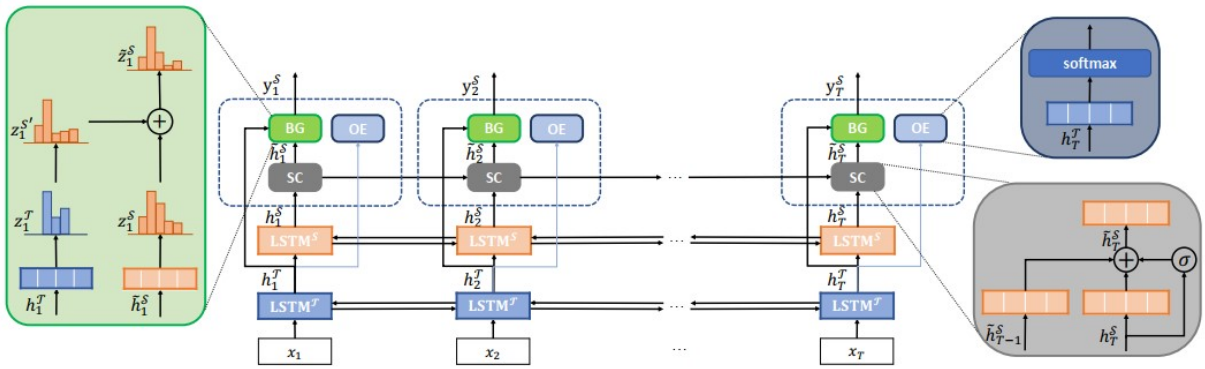


Figure 47: UABSA Architecture. *Source:* [46, p.3]

The upper $LSTM^S$ is responsible for the entire ATE+ATSC task which also includes outputting a label sequence. It is predicted using a softmax layer. The lower $LSTM^{\mathcal{T}}$, however, does auxiliary work by predicting boundary tags of targets. They are used to improve the predictions of the upper LSTM. This means that the lower LSTM produces the first part of the unified label according to the BIOES-scheme which restricts the unified tags to those that start with the correct letter. Both LSTMs are connected in such a way that the hidden representations from $LSTM^{\mathcal{T}}$ can be directly forwarded to the upper LSTM. In formulas, it looks like this:

$$h_t^{\mathcal{T}} = [\overrightarrow{LSTM^{\mathcal{T}}}(x_t); \overleftarrow{LSTM^{\mathcal{T}}}(x_t)], \tag{4.15}$$

$$h_t^{S} = [\overrightarrow{LSTM^{S}}(h_t^{\mathcal{T}}); \overleftarrow{LSTM^{S}}(h_t^{\mathcal{T}})]. \tag{4.16}$$

Here, we have time $t \in 1, \ldots, T$, input x_t and the hidden representations $h_t^{\mathcal{T}}$ and h_t^S of the corresponding LSTMs. (Model-based) Probability scores $z_t^{\mathcal{T}}$ for the boundary tags and the unified tags are obtained by

$$z_t^{\mathcal{T}} = \mathbb{P}(y_t^{\mathcal{T}}|x_t) = softmax(W^{\mathcal{T}} h_t^{\mathcal{T}}), \tag{4.17}$$

$$z_t^S = \mathbb{P}(y_t^S|h_t^{\mathcal{T}}) = softmax(W^S h_t^S), \tag{4.18}$$

where $W^{\mathcal{T}}$ and W^S are weights.

At this point, the extra components come into play: The first one is the BG component which is colored green in Figure 47. It turns the label restrictions, also called constraints, into a transition matrix which is initialized with

$$W_{i,j}^{tr} = \begin{cases} \frac{1}{|B_i|}, & if\ j \in B_i, \\ 0, & otherwise. \end{cases}$$

The boundary tag is denoted as i, the unified tag as j and the set of possible unified tags as B_i. The transition matrix is used to calculate transition-based/boundary-based sentiment scores

$$z_t^{S'} = (W^{tr})^\top z_t^{\mathcal{T}}.$$

As these scores might be not very informative, e.g. if the boundary tagger makes predictions with low confidence, proportion scores α_t are introduced. Relying on the confidence c_t, they are defined as

$$\alpha_t = \epsilon c_t \text{ with } c_t = (z_t^{\mathcal{T}})^\top z_t^{\mathcal{T}}$$

and the maximum proportions ϵ of the scores $z_t^{S'}$ in the tagging decisions. The final scores then are a combination of boundary-based and model-based unified tagging scores

$$\tilde{z}_t^S = \alpha_t z_t^{S'} + (1 - \alpha_t) z_t^S.$$

It may appear for multi-word aspect terms that the predictions are not consistent among all the words. To avoid this issue, the SC component was designed (marked in gray in the graphic). The underlying idea is to predict a unified tag based on the features from its own time step and those from the previous time step. Thus, a gating mechanism is used:

$$\tilde{h}_t^S = g_t \odot h_t^S + (1 - g_t) \odot \tilde{h}_{t-1}^S \text{ with } g_t = \sigma(W^g h_t^S + b^g),$$

where W^g and b^g are weight and bias.

The third component OE which is symbolized in blue in Figure 47 is responsible for a more robust boundary tagger. This goal should be reached by taking a different look at the training data. The underlying hypothesis is that a word counts "as a target word if there is at least one opinion word within the context window of fixed-size [...] of this word" (cf. [46]). An auxiliary classifier on token-level is set up to determine whether a word is a target word or not:

$$y_t^O = \arg\max_y z_t^O \text{ with } z_t^O = softmax(W^O h_t^{\mathcal{T}}).$$

The boundary representation $h_t^{\mathcal{T}}$ here include labels indicating a target word. Weights are denoted by W^O. The complete model is trained by minimizing the sum over the cross-entropy losses for the tree tasks $S, \mathcal{T}$ and O.

ADS-SAL

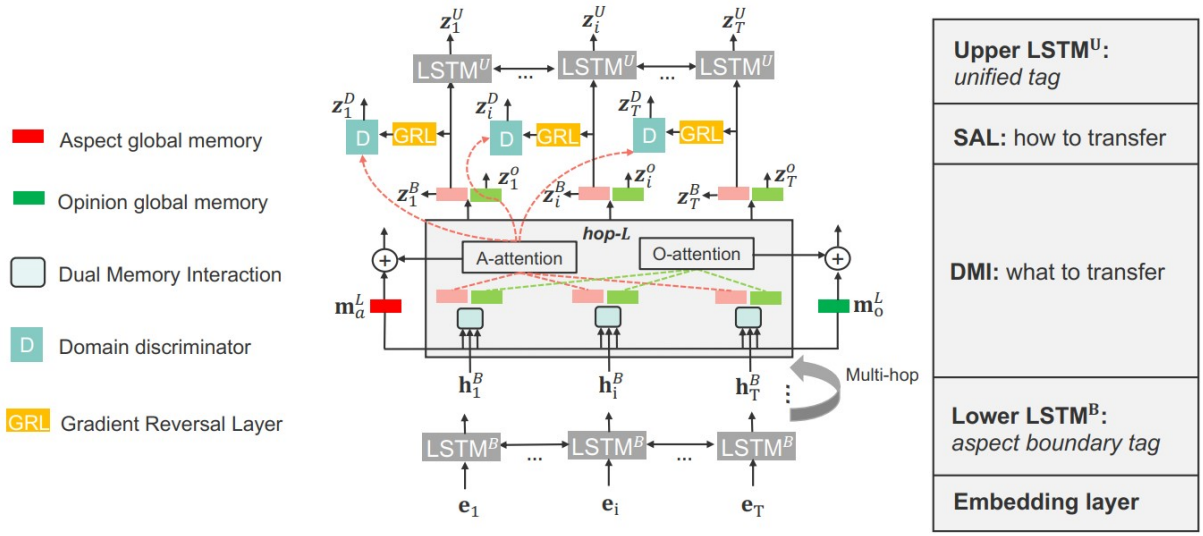


Figure 48: SAL Architecture. *Source:* [48, p.4592]

Based on *Selective Adversarial Training (SAL)*, [48] introduced their new approach ADS-SAL, where "ADS" stands for aspect detection and sentiment. As depicted in Figure 48, it resembles UABSA in its underlying architecture of two stacked BiLSTMs. Like before, the upper $LSTM^U$ outputs the collapsed labels, whereas the lower $LSTM^B$ is responsible for predicting aspect term boundaries. Their representations are denoted as h_i^B and h_i^U and obtained by Equations (4.15) and (4.16). Note that U stands for upper (instead of S) and B stands for the lower LSTM (instead of $\mathcal{T}$) now. The model-based probability scores for aspect boundary tags z_i^B and unified tags z_i^U are calculated as in

Equations (4.17) and (4.18), with the only difference that explicit bias terms within the softmax are mentioned now. The loss for both tags with true labels y_i^Q is summarized as

$$\mathcal{L}_{\mathcal{M}} = \sum_{D_s} \sum_{Q \in B,U} \sum_{i=1}^{T} l(z_i^Q, y_i^Q) \tag{4.19}$$

Different to UABSA, now a *Global-Local Memory Interaction (GLMI)* is introduced which is responsible for the interaction between a global memory m and a local memory h_i. In order to have both memories in the same space, local and global information are combined in the transformed local memory $\tilde{h}_i = h_i + ReLU(W[h_i; m] + b)$ with the concatenation operator $[;]$. The amount of correlation between m and $\tilde{h}_i$ is measured by the correlation vector

$$r_i = m^\top G \tilde{h}_i^B =: f(h_i, m; \Theta, G)$$

where the latter is just a parameterization. We have $\Theta = \{W, b\}$ and G models the latent relations.

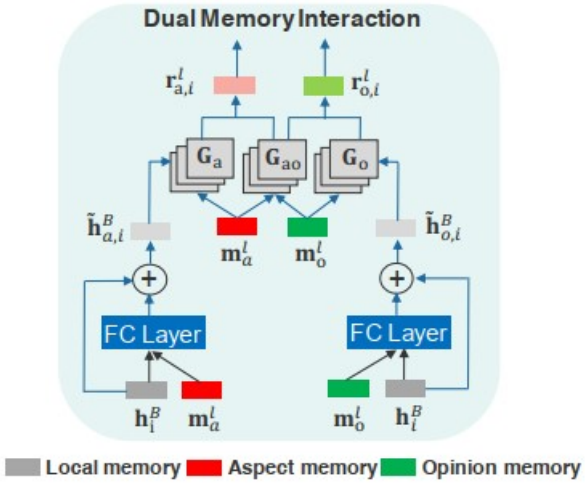


Figure 49: Dual Memory Interaction (DMI). *Source:* [48, p.4593]

Between the LSTMs, a multi-hop *Dual Memory Interaction (DMI)* is added. Its details are depicted in Figure 49. Analogously to the GLMI, we define a global aspect memory m_a and a global opinion memory m_o. The hidden states of the lower LSTM $H^B = \{h_i^B\}_{i=1}^T$ serve as local memories. In each hop l, two correlation vectors for aspect and opinion co-detection are calculated via

$$r_{a,i}^l = [f(h_i^B, m_a^l; \Theta_a, G_a); f(h_i^B, m_o^l; \Theta_o, G_{ao})]$$
$$r_{o,i}^l = [f(h_i^B, m_o^l; \Theta_o, G_o); f(h_i^B, m_a^l; \Theta_a, G_{ao}^\top)]$$

with the composition tensors G_a, G_o and G_{ao} for the latent relations within aspects, within opinions and between aspects and opinions. These vectors are then turned into aspect (A-attention) and opinion attention (O-attention) weights by

$$\alpha_{p,i}^{l} = \frac{\exp(W_p r_{p,i}^{l})}{\sum_{j=1}^{T} \exp(W_p r_{p,j}^{l})}$$

with $p \in \{a, o\}$ and weights W_p. The attentions are then used to update the global aspect and opinion memories of the next hop of with the current local memories: $m_p^{l+1} = m_p^l + \sum_{i=1}^{T} \alpha_{p,i}^{l} h_i^B$. In the last hop L, $r_{a,i}^L$ is fed to $LSTM_U$ for aspect detection and $r_{o,i}^L$ is taken to predict whether a word is an opinion word by probability scores

$$z_i^O = p(y_i^O | r_{o,i}^L) = softmax(W_O r_{o,i}^L + b_O). \tag{4.20}$$

The corresponding loss is called opinion detection loss and with true labels y_i^O and cross-entropy loss l it is defined as

$$\mathcal{L}_O = \sum_{D_s \cup D_t} \sum_{i=1}^{T} l(z_i^O, y_i^O).$$

Afterwards, a domain discriminator is used to determine the domain label y_i^D of each word. This is part of the *Selective Adversarial Learning (SAL)* which is used for declaring words with high probabilities as aspects. To do so, a *Gradient Reversal Layer (GRL)* is defined as $R_\lambda(x) = x$ with a reversal gradient $\frac{\partial R_\lambda(x)}{\partial x} = -\lambda I$ and adaptation rate λ. The GRL is applied on the correlation vector $r_{a,i}^L$ before the probability scores over the domain labels are predicted by

$$z_i^D = p(y_i^D | r_{a,i}^L) = softmax(W_D R_\lambda(r_{a,i}^L) + b_D).$$

The selective domain adversarial loss is defined as

$$\mathcal{L}_D = \sum_{D_s \cup D_t} \sum_{i=1}^{T} \alpha_{a,i}^L l(z_i^D, y_i^D), \tag{4.21}$$

where l stands for cross-entropy loss, D_s for labeled source data and D_t for unlabeled target data.

Training is divided into two stages: During the discriminative stage, the sum $\mathcal{L}_\mathcal{M} + \rho\mathcal{L}_O$ with hyperparameter ρ is minimized, while in the domain-invariant stage the optimization objective is the minimum of $\mathcal{L}_D$. This alternating procedure makes training more stable.

EI

Another collapsed labeling approach was introduced in [45]. It focuses on extracting both explicit (E) and implicit (I) structures that are relevant for the task of ATSC. Thus is called EI.

For the explicit structures, they made up a tagging scheme with three types of labels B_p, A_p and $E_{\epsilon,p}$. The goal of this scheme is to separate a sentence into spans to each of which a polarity is assigned. Within these spans, they distinguish between aspect terms and other words. Each word can have multiple labels. The index $p \in \{+, -, 0\}$ denotes the polarity and $\epsilon \in \{B, M, E, S\}$ stands for beginning, middle, end of and one-word aspect term. In case of B_p, the corresponding word is located before the aspect term or it is the first word of it. So to speak, the opposite, i.e. the word after the aspect term or its last word is denoted by A_p. The third label type $E_{\epsilon,p}$ indicates that the current word is both part of a sentiment span and part of the aspect term, where its exact position is specified by ϵ. A sentiment span ends when an A_p-labeled word is followed by a $B_{p'}$-labeled word; the first word belongs to a span with sentiment p, the other to another span with sentiment p'. Some words, especially conjunctions, are often not clearly assignable to one span or another. Thus, there are many possibilities for labeling a single sentence. In order to make this more clear, we refer to Figure 50, where the words of one sentence are labeled in two different ways. Neighboring tags are connected to each other by arrows. They are red if they cross a span border and blue otherwise.

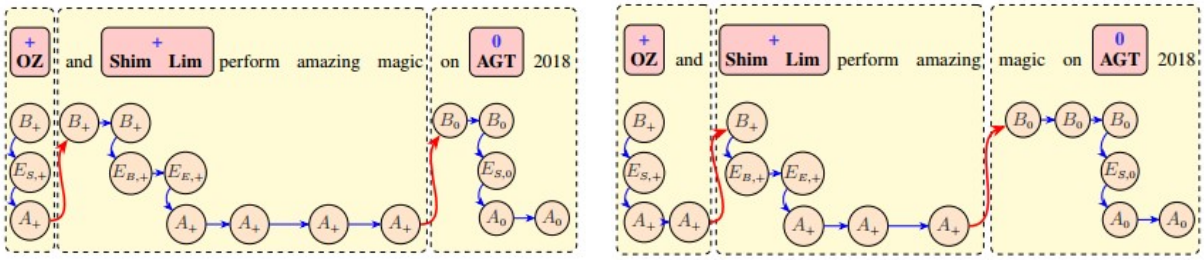


Figure 50: Labeling Examples of EI. *Source:* [45, p.2f]

The *explicit structure* has the following form: The probability of an output y given a sentence x is defined as

$$p(y|x) = \frac{\sum_h \exp(s(x, y, h))}{\sum_{y',h'} \exp(s(x, y', h'))}$$

with a latent variable h providing all combinations of sentiment spans for (x, y). The

sentence-level score function is given by

$$s(x, y, h) = \sum_{e \in E(x,y,h)} \phi_x(e),$$

where $E(x, y, h)$ denotes the set of all edges and $\phi_x(e)$ the edge-level score functions. The remaining model resembles a neural CRF as proposed in [16, 61].

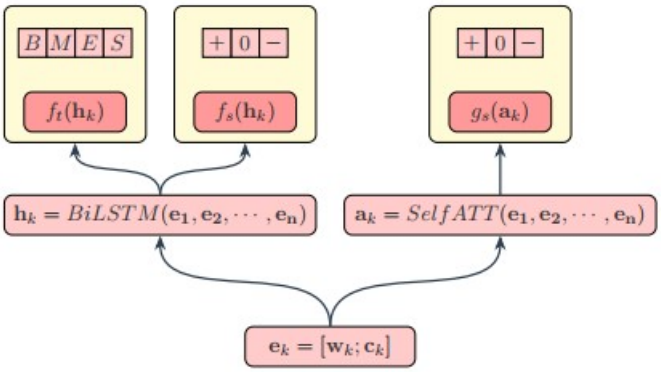


Figure 51: EI Architecture for implicit structures. *Source:* [45, p.4]

The edge-level score functions already belong to the *implicit structures* and are illustrated in Figure 51. Their calculation depends on the edge types:

$$\begin{aligned}
\phi_x(E^k_{\epsilon,p} E^{k+1}_{\epsilon',p}) &= f_t(h_k)_\epsilon \\
\phi_x(E^k_{\epsilon,p} A^k_p) &= f_t(h_k)_\epsilon \\
\phi_x(B^k_p B^{k+1}_p) &= f_s(h_k)_p \\
\phi_x(A^k_p A^{k+1}_p) &= f_s(h_k)_p \\
\phi_x(A^k_p B^{k+1}_{p'}) &= f_s(h_k)_p.
\end{aligned}$$

Two separate linear layers f_t and f_s are applied on h_k in order to obtain target and sentiment scores for each possible tag, i.e. $\{B, M, E, S\}$ and $\{+, -, 0\}$, respectively. The representations $h_k = BiLSTM(e_1, \ldots, e_n)$ are the outputs of a BiLSTM which takes embeddings $e_k = [w_k; c_k]$ consisting of word embeddings w_k and character embeddings c_k as inputs.

Additional edge-level score functions (also depicted in Figure 51) have the following form:

$$\begin{aligned}
\phi_x(B^{k_1}_p E^{k_1}_{\epsilon,p}) &= g_s(a_{k_1})_p \\
\phi_x(E^{k_2}_{\epsilon,p} A^{k_2}_p) &= g_s(a_{k_2})_p.
\end{aligned}$$

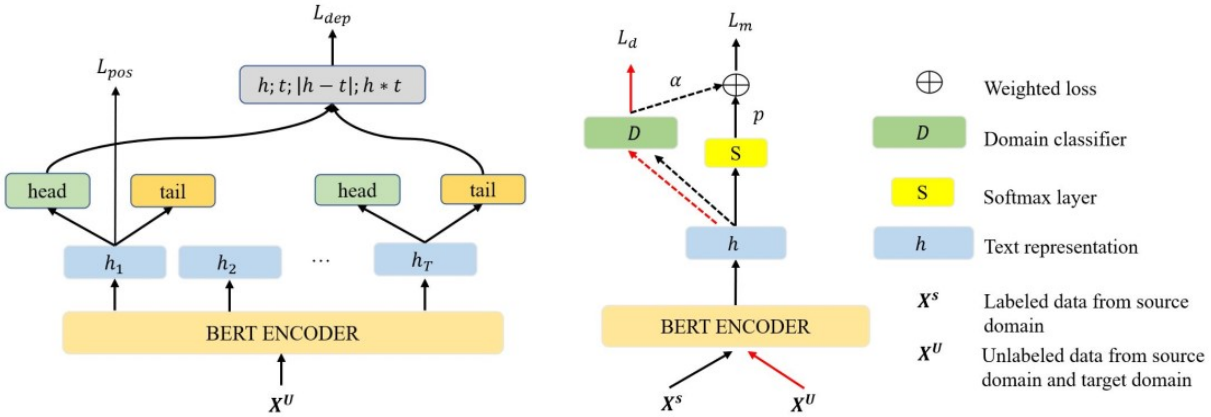
They return scores for $\{+, -, 0\}$ for the edges at the beginning k_1 and the end k_2 of the aspect terms. Their inputs a_k denoting the implicit structure between the corresponding word w_k and the context are the outputs of the following self-attention mechanism (see Section 3.4.1):

$$a_k = \sum_{j=1}^{n} \alpha_{k,j} e_j \text{ with } \alpha_{k,j} = softmax_j(\beta_{k,j}),$$
$$\beta_{k,j} = U^\top ReLU(W[e_k; e_j] + b).$$

The attention matrix is denoted by U, weights by W and bias by b. This procedure was inspired by [42].

BERT-UDA

BERT-UDA in [22] is a BERT-based approach relying on a *unified domain adaptation (UDA)* framework with two components for Feature- and Instance-Based Domain Adaptation. The authors introduce these two building blocks to reduce feature and instance discrepancy between different domains.



(a) (b)

Figure 52: Components of BERT-UDA: (a) Feature-Based Domain Adaptation Component (b) Instance-Based Domain Adaptation Component. *Source:* [22, p.7038]

We start with describing the *Feature-Based Domain Adaptation* component to which Figure 52a corresponds. Each word is embedded using four different types of embeddings which are word, segment, absolute position and POS tag embeddings. They are fed to a BERT layer and result in context-aware representations H. Masked POS tagging

is performed on them which means that 25% of the tokens and their tags are replaced with `[MASK]`, similarly to training BERT. For the masked tokens, a POS tag should be predicted. The probabilities p_i^{pos} of POS tag types are calculated as a softmax of a linear transformation of h_i. For optimization, the loss is defined as

$$L_{pos} = \sum_{D_U} \sum_{i}^{T} \mathbb{1}_{\{token\ i\ is\ masked\}} l(p_i^{pos}, y_i^{pos}) \tag{4.22}$$

where y_i^{pos} indicates the true POS tag, D_U the set of unlabeled instances of the target domain and l the cross-entropy loss. Furthermore, syntactic relations in H are identified by

$$h_i^{head} = \tanh(W_1 h_i + b_1) \text{ and } h_i^{tail} = \tanh(W_2 h_i + b_2)$$

which are the representations of head and child or tail tokens in a dependency tree. Weights and biases are denoted by W_1, W_2 and b_1, b_2. So, for words i and j that are connected in the dependency tree, their dependency relation is the following concatenation:

$$o_{ij} = [h_i^{head}; h_j^{tail}; h_i^{head} - h_j^{tail}; h_i^{head} \odot h_i^{tail}].$$

The corresponding probabilities p_{ij}^{dep} are received by applying a linear transformation and a softmax on o_{ij}. Optimization is done on

$$L_{dep} = \sum_{D_U} \sum_{i}^{T} \sum_{j}^{T} \mathbb{1}(ij) l(p_{ij}^{dep}, y_{ij}^{dep}), \tag{4.23}$$

where $\mathbb{1}(ij)$ equals 1 if tokens i and j are connected via a direct edge in the dependency tree. The entire loss for feature-based domain adaptation is

$$L_{feature} = L_{pos} + L_{dep}. \tag{4.24}$$

Instance-based Domain Adaptation (see Figure 52b) strives to cover the differences between domains by re-weighting the instances in the source domain. In each sentence, words can be distinguished as domain-invariant and domain-specific. For each word, a word-level domain classifier is trained which predicts whether a word belongs to the source or target domain. The domain distribution probabilities p_i^D are obtained by applying a linear transformation and a softmax on H. The loss for this classifier is

$$L_d = \sum_{D_U} \sum_{i=1}^{T} l(p_i^D, y_i^D)$$

with ground truth y_i^D. During the main task, the ratio of word-level target-domain and

source-domain probabilities $\frac{p_{i,t}^D}{p_{i,s}^D}$ is used as weight for each word. Thus, L_d only optimizes those weights and biases that are needed for the linear transformation when calculating p_i^D. Training the main task is based on the loss

$$L_m = \sum_{D_S} \sum_{i}^{T} \alpha_i l(p_i^m, y_i^m) \text{ with } p_i^m = softmax(W_m h_i + b_m).$$

Word-level weights α_i are retrieved by re-normalization of $\frac{p_{i,t}^D}{p_{i,s}^D}$ of which the components are delivered by the domain classifier. The composite loss for instance-based domain adaptation is

$$L_{instance} = L_m + L_d. \tag{4.25}$$

There are two ways to conduct overall model training, sequentially and jointly. In the first case, initially a feature space is learned by optimizing Equation (4.24); then instance-based domain adaptation is executed by minimizing Equation (4.25). For joint training, the four losses are summed up in the following way, referring to Equations (4.22) and (4.23):

$$L = L_m + L_d + L_{pos} + \lambda L_{dep}.$$

Here, λ denotes a hyperparameter. Ablation studies showed that in most of the cases, joint training outperformed sequential training. Thus, the values reported in Table 13 are those from joint training. They used two different BERT models for BERT-UDA: the original uncased BERT-Base (in our table: BERT-B) and the post-trained BERT from [89] (BERT-E). Like in the cross-domain case of BERT-ADA (see Section 4.1.5), training is done on the source domain and evaluation on the target domain.

5 Experiments

Having gathered all the theoretical knowledge about model architectures and categorization, we selected six models to be re-evaluated on five different data sets. These data sets as well as their preparation will be explained in detail in this section. The goal is to establish comparability between the models and to examine whether reported results can be reproduced. We will also give an overview about the reasons for choosing the models, their implementations and the results. Our code is available on our GitHub page[4].

5.1 Selected Approaches

The idea was to choose one model for most of the categories from the previous section for our experiments. We did not cover all of them due to various reasons. They include missing opinion term labels for MAMS or ARTS, missing or too large additional data or too long training times. For our selected approaches, the procedure is the following: At first, we take the implementation as given by the authors and try to reproduce their results. We leave the hyperparameters as they are stated there. Then, we adapt their code for the remaining data sets and their modifications, sticking closely to the hyperparameters for already used data. The biggest change we made was increasing the number of training epochs drastically and adding an early stopping mechanism to all of them. We had a *Tesla V100 PCIe 16GB* GPU at hand to speed up model training.

MGATN We chose MGATN for our practical part as it is reported to be the best of the RNN-based models on SemEval-14 data sets (for SemEval-14 Restaurants, see Figure 11). As there is no public implementation by its authors, we refer to this collection[5] of ABSA methods. We slightly modified the here introduced early stopping mechanism and then implemented it into the other models. The best model during training process was chosen

[4] `https://github.com/el-ma-le/atsc-experiments-official`
[5] `https://github.com/songyouwei/ABSA-PyTorch`

based on the best validation accuracy and/or F1 Macro Score - like for all other ATSC models.

CapsNet-BERT CapsNet-BERT seems to outperform all capsule networks with respect to their accuracy on SemEval-14 Restaurant data. Additionally, it performed second-best on MAMS as Figure 53 shows. Thus, we chose it for our experiments. Its code can be found here[6].

RGAT-BERT The best performance in the group of graph-based models is RGAT-BERT which is also the best of all approaches on MAMS data in terms of both accuracy and F1 Macro (see Figure 53). Additionally, it is one of the latest approaches. Its implementation can be found here[7]. In comparison to the two models from above, the manually created accuracy score was not the same as the one from Scikit-learn[8], which is why we substituted the original metric to keep the metrics across the models aligned. During the data transformation process, we selected the stanza tokenizer (see [67]) as it provided us with the necessary syntax information. We decided against the Deep Biaffine Parser[9] that was used by the authors since it did not produce the syntactic dependency relation tags and head IDs (see Section 2.3) we needed.

LCF-ATEPC We chose LCF-ATEPC for the group of LCF-based models as it has reached the highest F1 Macro Scores and accuracy on SemEval-14 data of all approaches (see Figure 11). Yet, this only holds for the variant that is trained using additional domain adaptation which we were not able to reproduce due to missing pretrained models. Thus, we decided to go for the second best, LCF-ATEPC-Fusion using the official implementation of LCF-ATEPC from here[10]. During our experiments, the authors of LCF-ATEPC started building a new repository[11] based on the existing code which we did not use as it was still subject to changes. LCF-ATEPC is a joint approach which can be trained simultaneously for aspect extraction and aspect term polarity classification. Yet, the extracted aspects are not used for the ATSC task in this case. Thus, this method can be seen as a single-task model which is why we report the ATSC measures only and no ATE measures.

[6] https://github.com/siat-nlp/MAMS-for-ABSA
[7] https://github.com/muyeby/RGAT-ABSA
[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
[9] https://github.com/yzhangcs/parser
[10] https://github.com/yangheng95/LCF-ATEPC
[11] https://github.com/yangheng95/pyabsa

BERT+TFM In order to determine which kind of F1 Score is reported in the paper, we decided to run BERT-E2E-ABSA methods in our practical analysis. Besides, they seemed to be the best models on Laptop data among all collapsed models. The variant of our choice was TFM as it was slightly better than the rest (see Figure 38). The implementation is taken from this repository[12]. As our F1 Micro Scores were much closer to the given values as our F1 Macro Scores, we assume that they report F1 Micro. Due to misunderstandings of the SemEval14-16 data sets, the authors announced new performance measures for the restaurant data sets on their GitHub page which we include in our tables. Model selection was based on F1 Micro and Macro Scores, which were calculated based on *(start position, end position, polarity)*-triples for each identified aspect. Due to the collapsed labeling scheme, these scores account for both ATE and ATSC. The calculation based on triples was included in the other ATE+ATSC models as well.

GRACE GRACE appears to be the best of the pipeline models in literature. Furthermore, it is the best ATE+ATSC model on SemEval-14 Restaurants and Laptops as Figure 38 shows. It includes a post-training step of the pretrained BERT model using Yelp and Amazon data. The authors provide this post-trained model on their GitHub page[13] where we also took the implementation from. Model selection was done based on ATSC F1 Micro and F1 Macro Score as well as on ATE F1 Micro Score; their calculations were slightly adjusted in order to match the calculation of those from BERT+TFM.

5.2 Data Sets

All data sets used throughout this thesis are publicly available. We give an exploratory overview about their characteristics based on our own analyses. The sizes and numbers may slightly vary from some papers due to different approaches in data selection and preprocessing.

5.2.1 SemEval-14

SemEval-14 Restaurants The most frequently used data set in ABSA is about restaurants in New York. It was presented first in [20] and adapted later by many others. It probably received the most publicity being the basis for the restaurant data sets of the

[12] `https://github.com/lixin4ever/BERT-E2E-ABSA`
[13] `https://github.com/ArrowLuo/GRACE`

Semantic Evaluation (SemEval) Conferences in 2014-2016 [64–66]. Here, we solely focus on SemEval-14 as these are the data sets most suitable for our task and the ones for which the most evaluation results exist. A subset of the restaurant data from [20] was chosen as training data and several kinds of labels for specific subtasks were added. The testing data were collected by the authors of [64] themselves and labeled in the same way. The train and test sets (as well as other data sets for the competition) are available here[14]. Each row contains one English sentence giving the author's opinion about a certain restaurant in New York. As these data sets were designed for both ATSC and ACSC tasks, there are sentences that only have labels for one of the tasks, i.e. aspect terms or aspect categories, each with corresponding polarities. We are only interested in ATSC which is why in Table 8 we also state the data set sizes for which we removed sentences without any extracted aspect terms. We also deleted duplicate sentences in the training set. Interestingly, there are three more sentences in the training set than reported in [64]. For each identified aspect term within a sentence, the polarity is given by positive, negative, neutral or conflict. Yet, we removed conflict polarities due to their small amount. Together with the corrections mentioned before, this reduced the amount of training and test sentences usable for our task from 3,044 to 1,978 and 800 to 600, respectively. Table 9 shows the distribution of aspect terms per sentence after data cleaning. The subset of sentences that contain at least two aspect terms with different sentiments has a length of 320 in the training set and of 80 in the test set. For each sentence, in addition to the sentence itself and the aspect terms with polarities, the start and end position of each aspect term are given.

Data Set	**Subset**	**Original Sentences in total**	**Sentences without Duplicates**	**Sentences for 3-class ATSC**	**Multi-Sentiment Sentences**	**Aspect Terms in total**	**Positive Aspect Terms**	**Negative Aspect Terms**	**Neutral Aspect Terms**	**Removed Conflict Aspect Terms**
SemEval-14 Restaurants	Training	3,044	3,038	1,978	320	3,605	2,161	807	637	91
	Test	800	800	600	80	1,120	728	196	196	14
SemEval-14 Laptops	Training	3,048	3,036	1,460	166	2,317	988	866	463	45
	Test	800	800	411	38	638	341	128	169	16
ARTS Restaurants	Test	2,784	2,784	2,784	206	3,528	1,952	1,103	473	0
ARTS Laptops	Test	1,576	1,576	1,576	74	1,877	883	587	407	0
MAMS Restaurant	Training	4,297	4,297	4,297	4,297	11,186	3,380	2,764	5,042	0
	Validation	500	500	500	500	1,332	403	325	604	0
	Test	500	500	500	500	1,336	400	329	607	0

Table 8: Number of sentences, aspect terms and aspect term polarities per data set. Multi-Sentiment sentences are those with at least two different polarities after removing "conflict" polarity. Aspect Terms in total also exclude "conflict".

[14] http://metashare.ilsp.gr:8080/repository/browse/semeval-2014-absa-restaurant-reviews-train-data/479d18c0625011e38685842b2b6a04d72cb57ba6c07743b9879d1a04e72185b8/ and http://metashare.ilsp.gr:8080/repository/browse/semeval-2014-absa-test-data-gold-annotations/b98d11cec18211e38229842b2b6a04d77591d40acd7542b7af823a54fb03a155/

Data Set	Subset	1	2	3	4	5	6	7	8	9	10	11	12	13
SemEval-14 Restaurants	Training	966	565	262	105	29	15	5	3	1	0	0	0	0
	Test	282	186	78	31	14	3	1	0	0	0	0	0	1
SemEval-14 Laptops	Training	895	350	139	42	10	6	3	1	1	0	0	0	1
	Test	258	101	34	10	6	1	0	0	0	0	0	0	0
ARTS Restaurants	Test	2,289	348	85	41	15	3	1	0	0	0	1	0	1
ARTS Laptops	Test	1,360	159	37	13	6	1	0	0	0	0	0	0	0
MAMS Restaurant	Training	0	2,568	1,169	364	126	48	13	6	1	1	1	0	0
	Validation	0	285	136	55	16	5	2	0	0	1	0	0	0
	Test	0	264	173	45	10	5	0	1	0	1	1	0	0

Table 9: Number of sentences with precisely 1 to 13 identified aspect terms per data set after removing duplicates, ACSC-only sentences and "conflict" polarities.

SemEval-14 Laptops The second domain of SemEval-14 data is Laptops. The data were collected and annotated in [64] for the task of aspect ATE and ATSC. The train data are available here[15], while the test data are combined with those of the Restaurants above. The corresponding statistics are shown in Tables 8 and 9. Again, there are three additional sentences in comparison to [64] and there were duplicate sentences in the training data which we deleted. Removing those sentences without any identified aspects or conflict polarities only, we end up with 1,460 instead of 3,048 original rows of data in the training set and 411 instead of 800 in the test set. The number of sentences with two aspect terms with different sentiments is 166 in the training and 38 in the test set.

5.2.2 MAMS

The authors of [64] were not the only ones to take the data of [20] as basis and develop a data set for ATSC. A so-called *Multi-Aspect Multi-Sentiment (MAMS)* data set for the restaurant domain was created in [34] who criticized existing data sets for not being adequate for ATSC. Their main point was that those original data sets mainly contain sentences with only one aspect or several aspects with the same sentiment. This would make sentiment prediction far too easy: It would not be more difficult than sentence-level sentiment prediction. In order to deal with this issue, they only allowed sentences of [20] in their data sets that had at least two aspects with differing sentiments. Additionally, they only selected sentences with a maximum of 70 words. A similar approach was followed in [91] who extracted a so-called "hard subset" of the SemEval-14 test data consisting only of samples with multiple aspects with opposite polarities. Yet, their data set did not get very popular so far.

[15] `http://metashare.ilsp.gr:8080/repository/browse/semeval-2014-absa-laptop-reviews-train-data/94748ff4624e11e38d18842b2b6a04d7ca9201ec33f34d74a8551626be122856`

Having prepared a data set for each of ATSC and ACSC, we now focus on the one for ATSC which is provided in this repository[16]. Its characteristics can be found in Tables 8 and 9. Note that they did not take aspects with a conflicting polarity into account. The MAMS data sets have the same structure as the SemEval-14 data sets, yet the official MAMS set is the only data collection with an extra validation set. Its proportion with respect to the union of training and validation data is about ten percent.

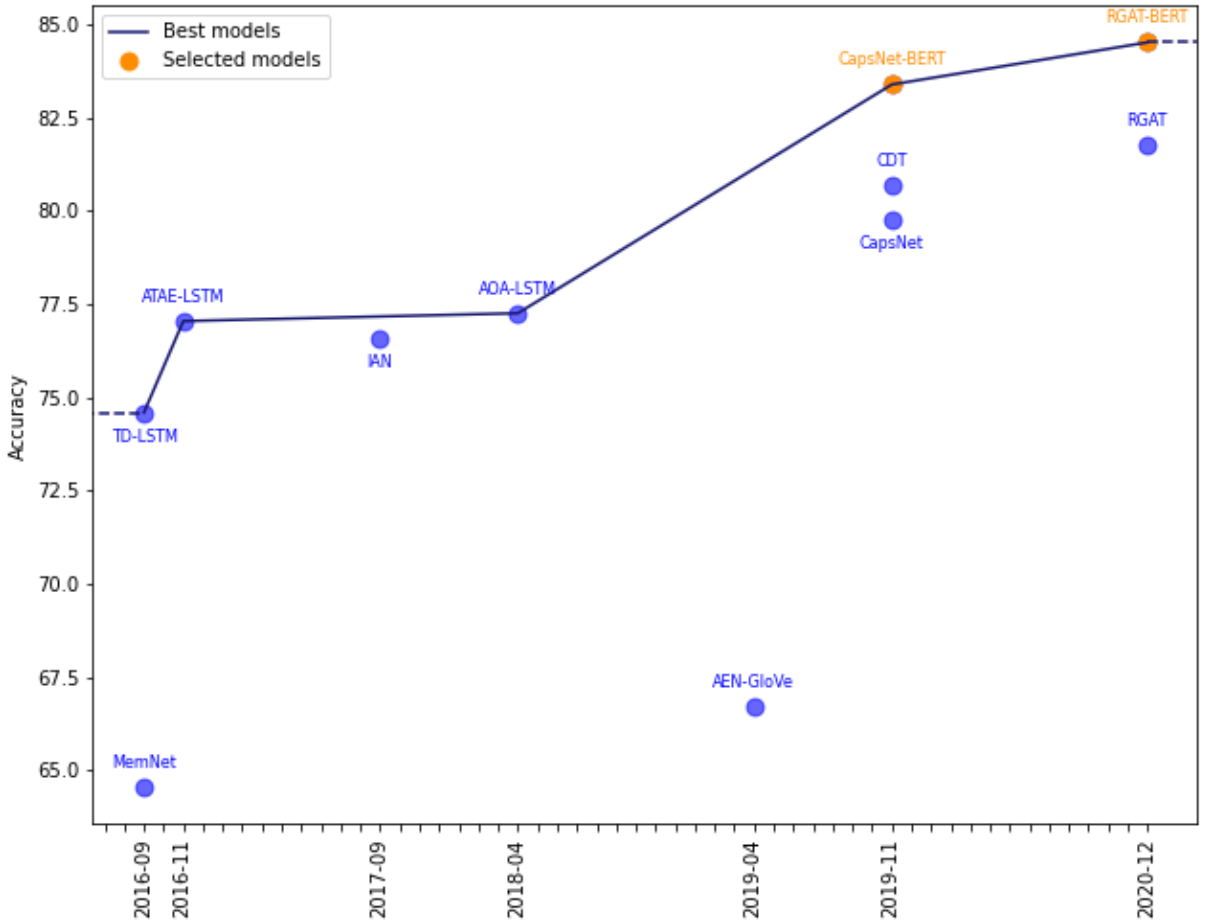


Figure 53: Development of reported ATSC Accuracy on MAMS. All values except for RGAT(-BERT) and CDT are taken from the MAMS paper.

The authors of MAMS did not explicitly state which performance measure they used. However, comparing the results for other methods, e.g. AOA-LSTM or AEN, makes us conclude that they report accuracy. We applied the same logic to the number of classes

[16] https://github.com/siat-nlp/MAMS-for-ABSA

for their SemEval results, which is why we assume that they only tested on the three classes positive, negative and neutral. The reported accuracy measures on MAMS are shown in Figure 53.

5.2.3 ARTS

Another issue regarding the quality of SemEval data sets was raised in [87]: They question whether the existing data sets are good enough to test the aspect robustness of a model, i.e. whether the model is able to correctly identify the words corresponding to the chosen aspect term and predict its sentiment based only on them. Using their examples *"Tasty burgers and crispy fries"* and *"Terrible burgers, but crispy fries"*, this means that a model should predict a positive polarity for "burgers" in the first sentence and a negative one in the second. In order to test this, it is necessary to have sentences containing aspects with differing polarities. Thus, the authors created an automatic generation framework that takes SemEval-14 test data as input and yields an *Aspect Robustness Test Set (ARTS)*. They used three different strategies to enrich the existing test set: The first one, REVTGT (="reverse target"), aims to reverse the sentiment of the chosen aspect term (also called target aspect). This is reached by flipping the opinion using antonyms or adding negation words like "not". Additionally, conjunctions may be changed in order to make sentences sound more fluent, as it was done in the example above. Another way to improve the test set is REVNON (="reverse non-target") which changes the sentiment of non-target aspects if they have the same sentiment as the target aspect or exaggerates it if it already is of a differing polarity. Taking the same example again, this may lead to *"Tasty burgers, but terrible fries"*. The third strategy is called ADDDIFF (="add different sentiment") and adds non-target aspects with an opposite sentiment to confuse the model. These aspects are selected from a set of aspects collected from the whole data set and put at the end of the sentence. This may create the following variant of our example: *"Tasty burgers, terrible fries and poor service"*.

As ARTS data only work as test sets, models have to be trained on SemEval training sets. The test sets for both restaurants and laptops can be found in this repository[17] and their statistics in Tables 8 and 9. They show that the focus of ARTS is totally different from the one of MAMS: Whereas the latter strives to provide sentences with lots of aspects with contradicting sentiments, the aim of ARTS is to make variations of the existing sentences. In contrast to the data sets mentioned before, each ARTS row contains only one aspect term, its polarity and its start and end position, but sentences may appear more than once to cover for different aspects. Sentiments can be positive, negative and

[17] `https://github.com/zhijing-jin/ARTS_TestSet`

neutral. Preparing the data set for CapsNet-BERT, we noticed that the start and end positions of some aspect terms were not correct. We changed them in order to make the code work properly. We also deleted duplicates.

5.3 Data Preparation

As we have already discovered in the previous section that each data set has been designed with its own motivation, they have different properties regarding task specificity, label sets and number of aspect terms. Now we describe our procedure to process the data in such a way that they have the same structure and same content for each model.

Data Structure Although almost all models rely on the same data from SemEval-14, they require different input formats which were determined by the corresponding authors. This means that not only the file types are different, but also the structure of the content. Table 10 gives an overview about these differences. *Sequence Tagging* means that each token receives a label, not only aspect terms. In order to fulfill our comparability goal, we re-created these data formats ourselves in order to make sure the same information is used for all models. Details can be found in Section 5.1.

Model	**Format**	**Aspects**	**Labels**
MGATN	xml.seg	one aspect per sentence	aspect term, polarity - in case of multiple aspects, duplicate sentences
CapsNet-BERT	xml	multiple aspects per sentence	aspect term, polarity, to, from (both character numbers)
RGAT-BERT	json	multiple aspects per sentence	token, pos, head, deprel, aspect term, polarity, to, from (both token numbers)
BERT+TFM	txt	multiple aspects per sentence	sequence tagging collapsed ("T"+sentiment)
GRACE	txt	multiple aspects per sentence	sequence tagging joint & collapsed: Token, POS-Tag, Chunk, Term Label, Sentiment Label, Collapsed Label
LCF-ATEPC	dat	multiple aspects per sentence	sequence tagging joint, per sentence the polarity only for one aspect term is given - in case of multiple aspects, duplicate sentences

Table 10: Overview about the different input formats for our models.

Duplicates The SemEval-14 training sets and the ARTS Restaurants test set included some duplicate lines. As we have already mentioned, we removed them.

Corrections The majority of the data sets contained a few incorrect labels. They were often incorrect in terms of the position of the aspects or their spelling. Which data sets needed to be corrected, can be found in Table 11.

Data Set	Subset	Original	Removed Duplicates	Removed Conflict	Corrected terms & Positions
SemEval-14	Restaurant Train	xml	yes	yes	yes
	Restaurant Test	xml	-	yes	-
	Laptop Train	xml	yes	yes	yes
	Laptop Test	xml	-	yes	-
ARTS	Restaurant Test	json	yes	-	yes
	Laptop Test	json	-	-	yes
MAMS	Train	xml	-	-	yes
	Validation	xml	-	-	-
	Test	xml	-	-	yes

Table 11: Overview about required preprocessing for data sets. Blank cells mean that the corresponding task was not necessary.

Full Data vs. ATSC Data Often, only a subset of the data is labeled for ATSC, i.e. aspect terms and their polarities are marked. These numbers are shown in Figure 8, columns 2 and 3. For ATSC, there is no benefit on training on data without aspect terms. For consistency, we removed all reviews without aspects from our data sets.

Three-class vs. Four-class Classification While the original SemEval-14 data sets contain negative, positive, neutral and conflict sentiment labels, in the others only the first three sentiment categories are used. These differences can also be observed in the architectures of some methods. As the number of conflict labels is rather small (as shown in Table 8), we removed them from the data.

Train-Validation Split Unlike MAMS data, the SemEval data come without an official validation set. Thus, we split the SemEval training data five times into training and validation sets with a 90%-10%-ratio like in [46]. On every train-validation pair and for each model, we perform five runs. For each of these combinations, the resulting trained model is then evaluated on the same test. Several runs are necessary in order to receive more reliable performance results.

5.4 Evaluation Metrics

For both ATE and ATSC, we can measure the model performance with two different statistics. However, it is important to note that only models within these two categories can be compared with each other, but not between those two. The reason behind this is that ATSC-only models are given the true aspect terms, whereas combined methods

identify them by themselves. This results in a discrepancy with respect to the set of aspect terms for which sentiment classification is to be made.

Accuracy The most widely used performance measure is accuracy. It is simply the percentage of correctly classified samples with respect to all samples (cf. [23, p.103]). More formally, it can be written as

$$Acc = \frac{TP + TN}{TP + TN + FP + FN},$$

where TP stands for the number of true positive samples, TN for that of true negatives, FP for that of false positives and FN for that of false negatives (cf. [71, p.38]).

F1 Score In multi-class classification, however, it is often more sensible to use the F1 Score instead (cf. [71, p.38]) as the class proportions may vary strongly. This behavior can be taken into account by the F1 Score. It is based on Recall R and Precision P, given as

$$R = \frac{TP}{TP + FN} \text{ and } P = \frac{TP}{TP + FP},$$

and defined as

$$F1 = \frac{2PR}{P + R}.$$

There are several calculation variants (cf. [1]) of which the most prominent are:

- micro: globally, i.e. count all TPs, FNs and FPs no matter which label they belong to.
- macro: label-wise, i.e. calculate F1 for each label and average them without weights.
- weighted: label-wise and weighted, i.e. calculate F1 for each label and average them using the number of true samples per label as weight. This takes label imbalance into account.

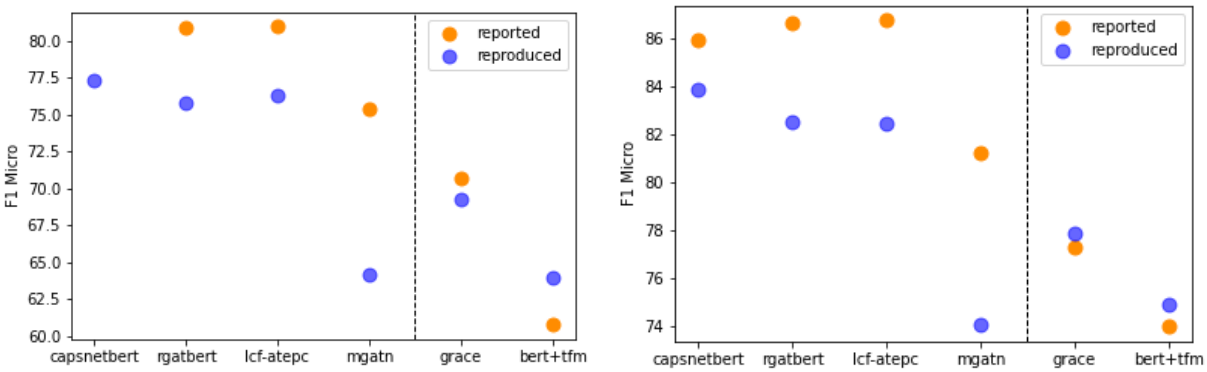
In order to produce easily comparable results, we are going to monitor both accuracy and all of these F1 Scores.

Aspect Robustness Score (ARS) The Aspect Robustness Score (ARS) was introduced in [87] in order to measure how well models can deal with variations of sentences like in the ARTS data set. Therefore, they see a sentence and all its variations as a unit for which a prediction is only correct if all the predictions for all variations are correct. These units and their corresponding predictions are then fed to a regular accuracy score. This score was used in the corresponding paper for ARTS data. For them, we calculate both ARS and the usual metrics.

5.5 Results

Our detailed quantitative results can be found in Tables 15 and 16 in the appendix. Here, we first give a general overview, then we will show plots and draw an individual conclusion for each model. Furthermore, we also discuss data-specific observations.

5.5.1 General Outcome



(a) SemEval-14 Laptops

(b) SemEval-14 Restaurants

Figure 54: Comparison of reported and reproduced performance. The reproduced value is the mean of all five runs for each of the five splits, i.e. 25 runs per model in total. Note that absolute performance of GRACE and BERT+TFM cannot be compared to the other models due to different tasks. No F1 Micro Score was reported for CapsNet-BERT on SemEval-14 Laptops.

Results not reproducible In general, reported values were not reproducible in an exact way. Figure 54 shows that our results except for BERT+TFM tend to be lower

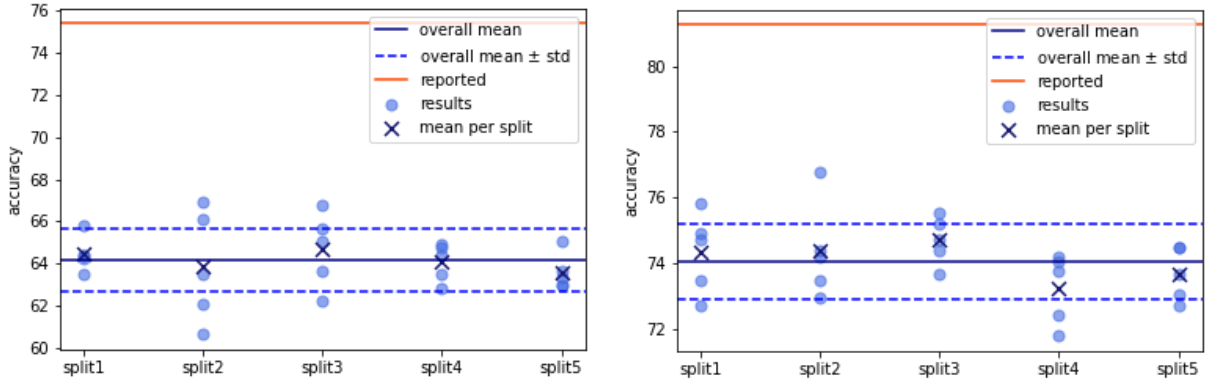
than the reported ones for both SemEval-14 Restaurants and Laptops. We do not give a similar figure for MAMS or ARTS as there are not enough reported values to form a good graph. The insight that reported values usually cannot be reached is also shared in [57], although they tested other models in a different setup.

Effect of multiple runs It is very interesting to see how different runs can lead to rather broad ranges of results, although having done only five training runs per model and data split. This is especially important when it comes to model comparison, where also small improvements with respect to the state of the art are often seen as a big success.

Effect of splits There is no clear sign that a certain split has a definitely better performance than another throughout all the models. We can observe that sometimes the variance within a split is definitely smaller than in another one, but there is no pattern. This may lead to the conclusion that the data sets are homogeneous enough so that the exact splits have no impact on model performance.

Reasons for gaps in performance results Results differing from the reported values can be explained by various reasons. First, we often do not know how the reported values were created, i.e. whether the authors took the best or an average value of their runs. For instance, Figure 61a shows that in average, our results are lower than the reported value. Yet, if only the best run is taken into consideration, we surpassed the reported score. Second, our data usually are not identical to the data sets used for the original papers due to the preprocessing steps we explained beforehand. Also, training and validation splits are probably different from ours. For some models, we needed additional syntactical information which we inferred from other packages than indicated (or none were given). Third, hyperparameter configurations are often not totally clear. Mainly, we took those that were chosen in the implementations. Consequently, it is not surprising that we were not able to exactly reproduce given results.

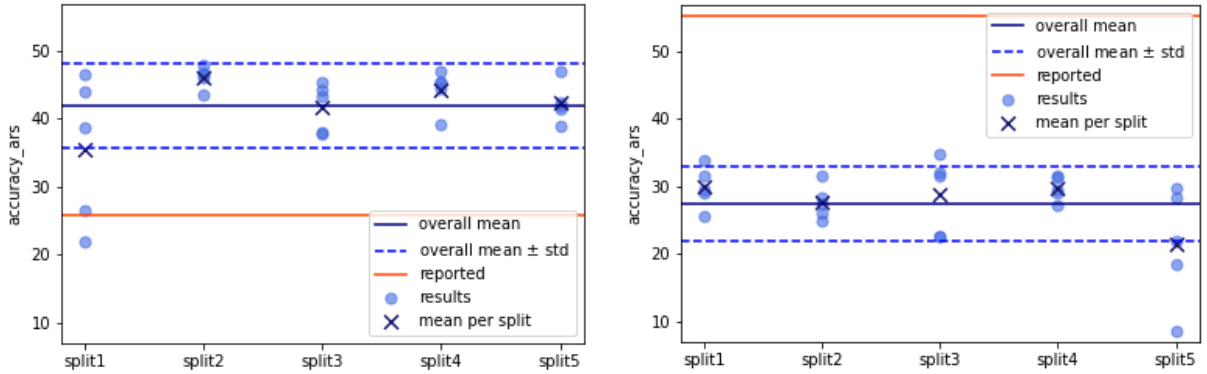
5.5.2 Model-specific Observations



(a) SemEval-14 Laptops (b) SemEval-14 Restaurants

Figure 55: Accuracy of MGATN.

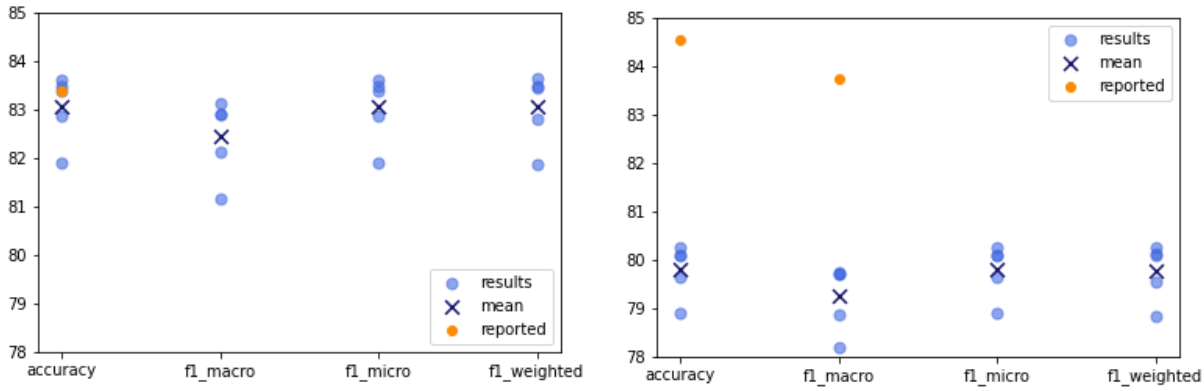
MGATN For MGATN, we were not able to reach the reported values by far. Figure 55 shows that our results are around five to ten percentage points below the reported accuracies for Laptops and Restaurants, respectively. A reason for this behavior might be that we could not use the official implementation of the authors. In terms of ARS Accuracy on ARTS Restaurants, MGATN was the only model that reached only a single-digit value which means that it is not good at dealing with perturbed sentences. Furthermore, our point of view that RNN-based methods perform worse than BERT-based methods (as shown by the reported values in Figure 11) was confirmed: As Table 15 illustrates the values of MGATN usually lie ten to 15 (sometimes even 20) percentage points below those of the other ATSC methods.



(a) ARTS Laptops (b) ARTS Restaurants

Figure 56: ARS Accuracy of CapsNet-BERT.

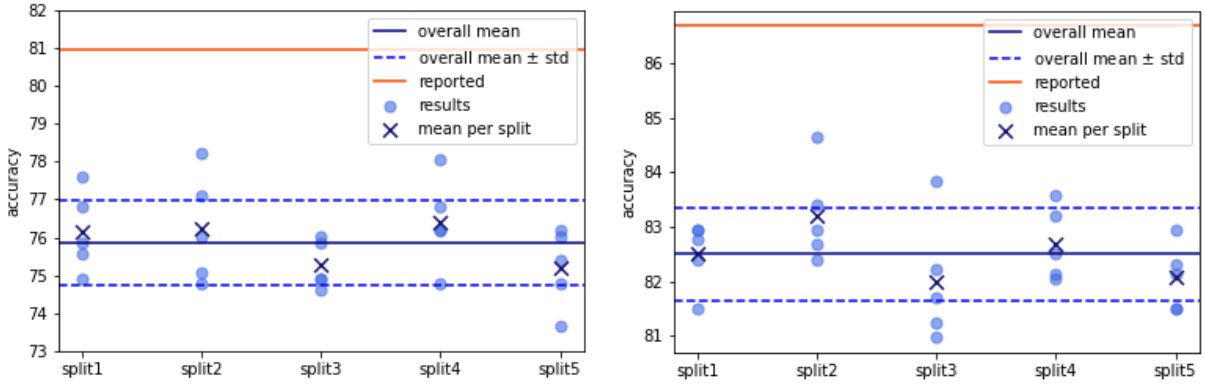
CapsNet-BERT Comparing all the selected models on the ATSC task, CapsNet-BERT performed best on all data sets regarding all the metrics except for ARS Accuracy on ARTS Restaurant data. Yet, our attention is mainly drawn towards ARTS and MAMS results, also because there is only one reported value for SemEval-14 data. In terms of MAMS, our results are scattered around the reported value as Figure 57a shows. For ARTS, however, it seems as if the reported ARS accuracy for Laptops matched our result for Restaurants, and vice versa, as Figure 56 illustrates. As far as we can tell, we did not mix up the data sets during our calculations which makes this quite peculiar.



(a) CapsNet-BERT (b) RGAT-BERT

Figure 57: Results on MAMS.

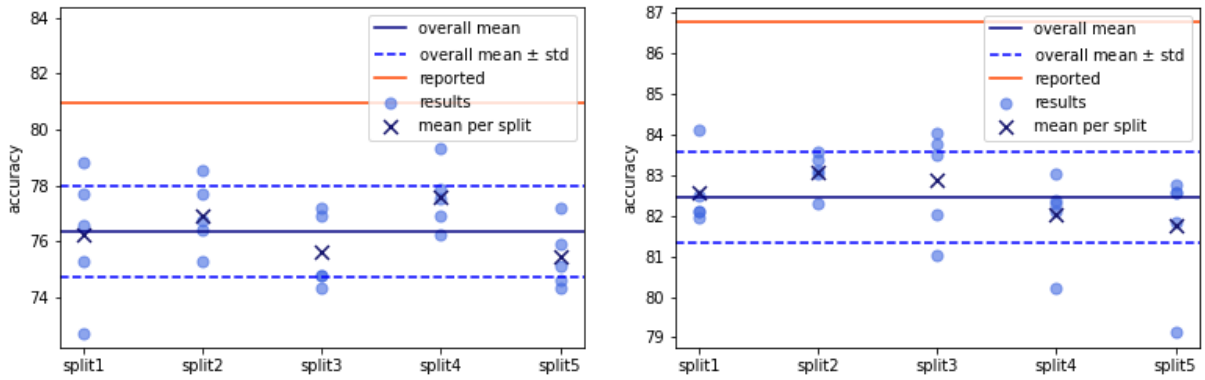
RGAT-BERT For both SemEval-14 and MAMS data we missed the reported values by around five percentage points as Figures 58 and 57b show. ARTS Restaurants is the only data set on which the best ARS Accuracy was not reached by CapsNet-BERT, but RGAT-BERT.



(a) SemEval-14 Laptops (b) SemEval-14 Restaurants

Figure 58: Accuracy of RGAT-BERT.

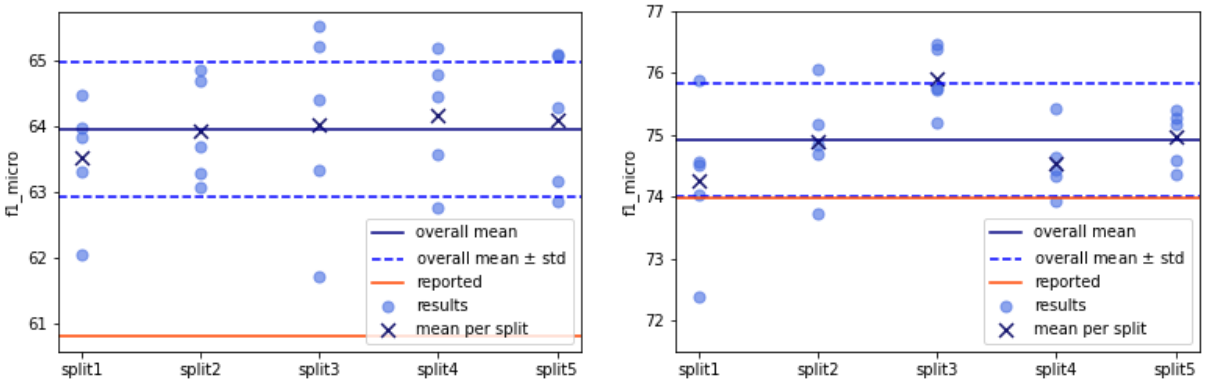
LCF-ATEPC Our experiments resulted in on average about five percentage points lower accuracies for LCF-ATEPC than were reported. This can be seen in Figure 59. Yet, LCF-ATEPC reached the best ARS Accuracy value on ARTS Restaurant data in our analysis.



(a) SemEval-14 Laptops (b) SemEval-14 Restaurants

Figure 59: Accuracy of LCF-ATEPC.

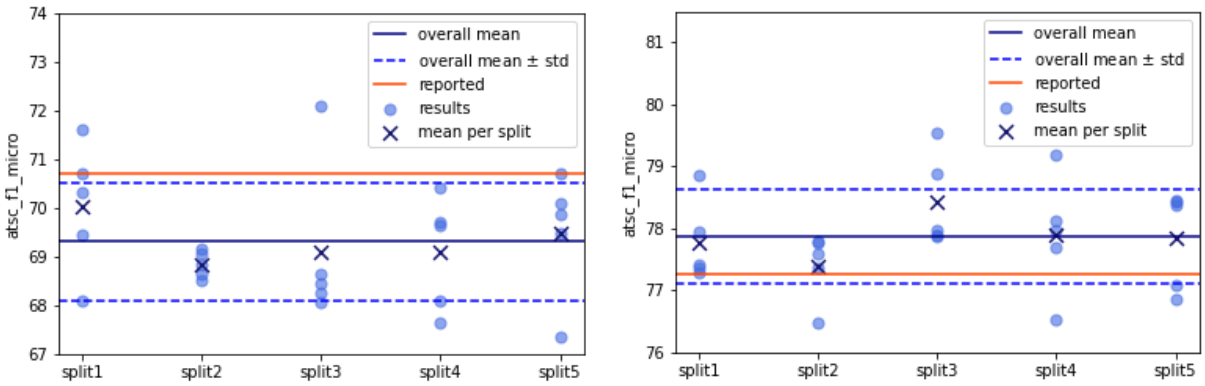
BERT+TFM In contrast to the majority of the other models, for BERT+TFM our runs surpassed the reported values on SemEval-14 data. As Figure 60 indicates this holds for all runs for the Laptop domain and in average for the Restaurant domain.



(a) SemEval-14 Laptops (b) SemEval-14 Restaurants

Figure 60: F1 Micro Scores of BERT+TFM.

GRACE Running GRACE, we were able to produce results in the scope of the reported values, regarding SemEval-14 Restaurants our results were even better in average. This is shown in Figure 61. In the ATE+ATSC task, GRACE outperformed BERT+TFM on all data sets except for MAMS.



(a) SemEval-14 Laptops (b) SemEval-14 Restaurants

Figure 61: ATSC F1 Micro Scores of GRACE.

5.5.3 Data-specific Results

SemEval-14 In general, the performances on the Restaurants data set are better than on the Laptops data set, usually within a range of five to ten percentage points. This behavior was not only observed by us, but can also be seen in the reported values in Figure 11. The only exception for differences between the domains can be found in the ATE F1 Micro Score for GRACE which is caused by the different task.

MAMS As explained in Section 5.2, MAMS and SemEval-14 Restaurants rely on the same source of original restaurant reviews. Yet, the MAMS data set is a more complicated version with more and usually differing aspects, which is why it makes sense that our observed performance on MAMS lies below that of SemEval-14 Restaurants. This behavior is also common among the reported values. Although we had the official validation set at hand here, reproduced and reported performance were not much closer to each other in average than for data sets where we constructed the splits by ourselves. The only exception is CapsNet-BERT where there is only a difference of 0.35 percentage points for accuracy, speaking of the overall mean.

ARTS Unlike for the SemEval-14 data sets, the performance values on ARTS do not differ that much from each other comparing Restaurant and Laptop data sets. Usually, the differences account to up to five percentage points for both ATSC and ATE+ATSC regarding the overall mean. Maybe the domain-specific results are more similar because the same enrichment methods have been applied to both of them which stronger aligned their levels of complexity with respect to the task. Referring to the ARS Accuracy, we can observe that this is a very strict metric, never resulting in an overall mean higher than 45%. Since mainly different models were tested in [87], it is hard to check whether this outcome is caused by our implementation of the ARS or these models in general perform badly with respect to this score.

6 Conclusion

In this work, we presented a thorough overview about existing models for aspect-term sentiment classification (and partly also for aspect term extraction), yet no claim is made to completeness. Categorization of approaches revealed that models are based on a variety of ideas to which the future will probably add many more. Starting with rather simple RNN- or CNN-based models, models evolved by including attention mechanism, local context focus or graphs. With the introduction of BERT, a new level of performance could be reached. In recent times, the task of identifying aspects became more relevant which is why more approaches deal with both tasks.

Our practical part revealed that reproducing reported results precisely is not possible, at least for the subset of our selected models. We decided to run MGATN for the group of RNN-based models, LCF-ATEPC as it was the overall best, RGAT-BERT as it was the latest, CapsNet-BERT for the capsule networks, GRACE for the group of pipeline approaches and BERT+TFM for the collapsed models. A tendency towards lower results is clearly visible in our experiments, sometimes even five to ten percentage points lower than the original values. The only exception was BERT+TFM for which we surpassed the given values. The reasons for these observations may lay in the data preprocessing step, in the hyperparameters or in the absence of a convention on which values to report (best or mean of several runs). Such a convention indicating a common reporting procedure combined with already prepared data sets with all possible labels could improve the comparability between models a lot. Also a huge practical meta-analysis of all models on several data sets would clarify the situation.

The discovery of models hardly being comparable based on their performance measures is a very important one from our point of view. When new models are proposed, one of the main aspects during their evaluation is the improvement with respect to the state of the art. But when the performance of a single model can vary between single runs, the question is which results to take into account for model rankings.

Based on our experiments, CapsNet-BERT outperformed the other ATSC models with a varying margin depending on data set and metric. Furthermore, it is easy to handle as no additional labels or syntax tags need to be added by external packages. Thus,

we recommend using this model to someone who wants to choose from our selection. In contrast to CapsNet-BERT, MGATN performed clearly worse than the other ATSC models, which we expected as typical for RNN-based approaches. For ATE+ATSC, our comparison revealed the superiority of GRACE which is probably justified by its more complex architecture.

In terms of data sets, it is not that easy to make a recommendation: SemEval-14 data are common benchmark data sets, yet ARTS and MAMS have their justification based on their motivation for more complicated data sets. In general, models perform best on Restaurant data sets from SemEval-14 and MAMS. Consequently, for simple comparisons against other models, the first ones should be sufficient (ignoring the comparability issues for the moment). When interested in the challenge of analyzing more complicated sentences, however, ARTS and MAMS data should be chosen. Regarding efficiency, applying models on SemEval-14 data first and then on ARTS would be the best choice as both are trained on the SemEval-14 training sets. In doing so, one longer lasting step of training could be skipped.

Anyway, as this is a rather young field of research, we are interested to see what the future brings and how methods can be improved any further.

7 Outlook

While we gave a rather broad overview in the theoretical part, the practical part still holds a lot of possibilities to be pursued in the future. Here, we collect the ideas that came to our minds during the work on our own experiments.

Cross Domain Experiments We have seen that there are some models like MGALN and BERT-ADA (see Section 4.1.5) or ADS-SAL and BERT-UDA (see Section 4.2.3) which are designed to do cross-domain learning, i.e. training on one domain, e.g. Restaurants, and testing on another, e.g. Laptops. Although this might lead to worse results than in-domain training, the idea behind this concept is important for practical reasons. Often, labeled data are very rare in practice and it is expensive to do labeling. Thus, having enough labeled at hand, would reduce annotation work to a minimum. Consequently, it would be very nice to know which models are suitable for this type of learning. Few experiments on this topic were already made in [57], yet there are enough model-data combinations left for further analyses.

Data Sets Recently, more ABSA data sets have been constructed, like *Men's T-Shirts* and *Television* in [57]. For SemEval-15, [66] also provided a test set for the hotels domain. Evaluating the existing models on these data sets could reveal whether the models perform well on all data sets, or it depends on the domain. Another new data set is the so-called YASO data set introduced in [59]. It consists of several domains which would make the task more interesting, but also less realistic as real-world applications of ATSC methods may focus on a single domain only.

Languages Pursuing the idea of new data sets even further, may lead to non-English data sets, for instance the GermEval-17 data set in [85]. It provides German texts about "Deutsche Bahn". Testing our models on this data set would introduce a new quality criterion as the models are usually only trained and tested on English data. Based on the idea of analyzing a German, one could also extend this idea and compare approaches

between different languages. For this task, however, appropriate data sets would be needed which are still not available for the majority of languages.

BERT Variants It would also be interesting to substitute the BERT basis of a model with DistilBERT [74], RoBERTa [52], AlBERT [43] or other variants of BERT. Each of them was created with a special motivation which may also be beneficial for the ATSC task: DistilBERT is much smaller, whereas RoBERTa was pretrained on much more data and for AlBERT changes in the model architecture were made. For instance, the performance results of models based on BERT and RoBERTa were compared in [13].

Different Basic Models BERT is a very popular model, especially in our case as basis for more complex ones. Yet, GPT [5], Electra [11], XLNet [94] and T5 [68] have also been published recently with similarly promising or even better results on benchmark tasks. Thus, researching about ABSA methods based on them may be interesting as well. Maybe it is even possible to achieve another increase in performance as the one from RNN-based to BERT-based models.

Multi-task extras A new labeling scheme for joint ATE+ATSC was introduced in [37]. Whereas in this work, the aspect label is assigned to the aspect term as usual, for the polarity label an extra token is created: After the aspect term, `[sentiment]` is inserted, like in the following example:

Word	Price	[sentiment]	was	affordable
Label	B	positive	O	O

Table 12: ATE+ATSC Labeling Example following [37].

Incorporating this labeling scheme into existing models could result into different performances which would be interesting to examine. Another idea regarding multi-task models comes from [92] who generated a unified framework to solve all ABSA subtasks in one run. Maybe it is possible to transfer this scheme to other models as well.

Weakly/Semi-/Unsupervised Models Few methods have been developed so far to solve the ABSA tasks in a weakly, semi- or unsupervised way. Yet, this could be helpful as labeling is always an expensive and time-consuming task. For instance, [27], [33] and [60] proposed some ideas on how this topic could be approached. Further work could give an

overview on how these models are structured, how they perform and how they can be improved in the future.

List of Figures

1 Example for a constituent tree. *Source:* [14, p.82] . . . 6
2 Example for a dependency tree. *Source:* [14, p.82] . . . 6

3 CNN Architecture. *Source:* [14, p.29] . . . 8
4 RNN Architecture . . . 9
5 LSTM Architecture. *Source:* [96, p.355] . . . 11
6 GRU Architecture. *Source:* [96, p.349] . . . 12
7 Attention Architecture. *Source:* [96, p.404] . . . 14
8 Multi-head attention architecture. *Source:* [96, p.415] . . . 15
9 Transformer Architecture. *Source:* [80, p.3] . . . 16
10 A single GCN layer. *Source:* [97, p.4569] . . . 18

11 Reported ATSC Accuracy on SemEval-14 Restaurants . . . 21
12 GCAE Architecture for ATSC. *Source:* [91, p.5] . . . 24
13 TD-LSTM Architecture. *Source:* [78, p.2] . . . 25
14 TC-LSTM Architecture. *Source:* [78, p.3] . . . 25
15 ATAE-LSTM Architecture. *Source:* [83, p.610] . . . 26
16 IAN Architecture. *Source:* [55, p.2] . . . 27
17 RAM Architecture. *Source:* [7, p.454] . . . 28
18 Attention-over-Attention LSTM Architecture. *Source:* [32, p.3] . . . 30
19 MGATN Architecture. *Source:* [18, p.3436] . . . 31
20 MemNet Architecture. *Source:* [79, p.3] . . . 33
21 AEN Architecture. *Source:* [76, p.3] . . . 35
22 Variants of PH-SUM: (a) P-SUM (b) H-SUM. *Source:* [39, p.3] . . . 37
23 BAT Architecture. *Source:* [38, p.3] . . . 38
24 MGALN Architecture. *Source:* [49, p.5] . . . 39
25 LCF Architecture. *Source:* [95, p.5] . . . 45
26 LCF-ATEPC Architecture. *Source:* [93, p.5] . . . 47
27 Example of a Dependency Parsing Tree. *Source:* [63, p.3216] . . . 48
28 PhraseRNN Architecture. *Source:* [58, p.2511] . . . 49
29 Constituent Tree Example. *Source:* [58, p.2510] . . . 50
30 SynATT Architecture. *Source:* [24, p.1124] . . . 51

31 SDGCN Architecture. *Source:* [98, p.3] . . . 53
32 CDT Architecture. *Source:* [77, p.250] . . . 55
33 ASGCN Architecture. *Source:* [97, p.4571] . . . 56
34 RGAT Architecture . . . 59
35 RGAT Layer. *Source:* [4, p.4] . . . 60
36 T(rans)Cap Architecture. *Source:* [8, p.549] . . . 61
37 CapsNet Architecture. *Source:* [34, p.6282] . . . 64
38 Reported ATE+ATSC F1 Micro on SemEval-14 Restaurants . . . 66
39 Building blocks of SPAN . . . 68
40 MTL-pipeline Architecture. *Source:* [3, p.250] . . . 70
41 GRACE Architecture. *Source:* [53, p.3] . . . 71
42 IMN Architecture. *Source:* [25, p.3] . . . 74
43 DOER Architecture. *Source:* [54, p.593] . . . 76
44 Building blocks of DOER . . . 77
45 Architecture of RACL . . . 79
46 LCM Architecture. *Source:* [82, p.823] . . . 82
47 UABSA Architecture. *Source:* [46, p.3] . . . 84
48 SAL Architecture. *Source:* [48, p.4592] . . . 86
49 Dual Memory Interaction (DMI). *Source:* [48, p.4593] . . . 87
50 Labeling Examples of EI. *Source:* [45, p.2f] . . . 89
51 EI Architecture for implicit structures. *Source:* [45, p.4] . . . 90
52 Components of BERT-UDA . . . 91

53 Reported ATSC Accuracy on MAMS . . . 99
54 Comparison of reported and reproduced values . . . 104
55 Accuracy of MGATN. . . . 106
56 ARS Accuracy of CapsNet-BERT. . . . 106
57 Results on MAMS. . . . 107
58 Accuracy of RGAT-BERT. . . . 107
59 Accuracy of LCF-ATEPC. . . . 108
60 F1 Micro Scores of BERT+TFM. . . . 108
61 ATSC F1 Micro Scores of GRACE. . . . 109

62 Reported ATSC Accuracy on SemEval-14 Laptops . . . 130
63 Reported ATSC F1 Macro on SemEval-14 Laptops . . . 131
64 Reported ATSC F1 Macro on SemEval-14 Restaurants . . . 132

List of Tables

1 ABSA Names 3
2 Labeling Examples 5

3 Accuracy 22
4 F1 Macro Score 23
5 SRD Example 45
6 F1 Micro Scores for ATE+ATSC 67
7 F1 Micro Scores for ATSC 73

8 Data Set Statistics 97
9 Aspect Term Statistics 98
10 Model Input Formats 101
11 Data Preprocessing Requirements 102

12 Labeling Example 114
13 Cross-Domain Results 129
14 Data Set Twitter Statistics 133
15 ATSC Results 135
16 ATE+ATSC Results 136

Bibliography

[1] scikit learn: sklearn.metrics.f1_score. `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html`, (accessed: 04.03.2021).

[2] spaCy: Linguistic features. `https://spacy.io/usage/linguistic-features`, (accessed: 12.07.2021).

[3] Md Shad Akhtar, Tarun Garg, and Asif Ekbal. Multi-task learning for aspect term extraction and aspect sentiment classification. *Neurocomputing*, 398:247–256, 2020.

[4] Xuefeng Bai, Pengbo Liu, and Yue Zhang. Investigating typed syntactic dependencies for targeted sentiment classification using graph attention neural network. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:503–514, 2020.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[6] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, 2014.

[7] Peng Chen, Zhongqian Sun, Lidong Bing, and Wei Yang. Recurrent attention network on memory for aspect sentiment analysis. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 452–461, 2017.

[8] Zhuang Chen and Tieyun Qian. Transfer capsule network for aspect level senti-

ment classification. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 547–556, 2019.

[9] Zhuang Chen and Tieyun Qian. Relation-aware collaborative learning for unified aspect-based sentiment analysis. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3685–3694, 2020.

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

[11] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. 2020.

[12] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25th International Conference on Machine Learning*, page 160–167, 2008.

[13] Junqi Dai, Hang Yan, Tianxiang Sun, Pengfei Liu, and Xipeng Qiu. Does syntax matter? A strong baseline for aspect-based sentiment analysis with roberta. *CoRR*, abs/2104.04986, 2021.

[14] Li Deng and Yang Liu, editors. *Deep Learning in Natural Language Processing*. Springer, Berlin, Heidelberg, 2018.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2019.

[16] Trinh–Minh–Tri Do and Thierry Artieres. Neural conditional random fields. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 9:177–184, 2010.

[17] Li Dong, Furu Wei, Chuanqi Tan, Duyu Tang, Ming Zhou, and Ke Xu. Adaptive recursive neural network for target-dependent Twitter sentiment classification. *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 49–54, 2014.

[18] Feifan Fan, Yansong Feng, and Dongyan Zhao. Multi-grained attention network for

aspect-level sentiment classification. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3433–3442, 2018.

[19] G. David Forney. The viterbi algorithm. *Proceedings of the IEEE*, pages 268–278, 1973.

[20] Gayatree Ganu, Noemie Elhadad, and Amélie Marian. Beyond the stars: Improving rating predictions using review text content. *Twelfth International Workshop on the Web and Databases (WebDB 2009)*, 2009.

[21] Yoav Goldberg. *Neural Network Methods for Natural Language Processing*, volume 37 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool, San Rafael, CA, 2017.

[22] Chenggong Gong, Jianfei Yu, and Rui Xia. Unified feature and instance based domain adaptation for aspect-based sentiment analysis. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7035–7045, 2020.

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

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

[25] Ruidan He, Wee Sun Lee, Hwee Tou Ng, and Daniel Dahlmeier. An interactive multi-task learning network for end-to-end aspect-based sentiment analysis. *CoRR*, abs/1906.06906, 2019.

[26] Ruining He and Julian McAuley. Ups and downs. *Proceedings of the 25th International Conference on World Wide Web*, 2016.

[27] Carlos Henríquez, Freddy Briceño, and Dixon Salcedo. Unsupervised model for aspect-based sentiment analysis in spanish. *IAENG International Journal of Computer Science*, 46, 2019.

[28] Geoffrey E. Hinton, Alex Krizhevsky, and Sida D. Wang. Transforming auto-encoders. *ICANN*, 2011.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–1780, 1997.

[30] Minghao Hu, Yuxing Peng, Zhen Huang, Dongsheng Li, and Yiwei Lv. Open-domain targeted sentiment analysis via span-based extraction and classification. *CoRR*, abs/1906.03820, 2019.

[31] Binxuan Huang and Kathleen M. Carley. Syntax-aware aspect level sentiment classification with graph attention networks. *CoRR*, abs/1909.02606, 2019.

[32] Binxuan Huang, Yanglan Ou, and Kathleen M. Carley. Aspect level sentiment classification with attention-over-attention neural networks. *CoRR*, abs/1804.06536, 2018.

[33] Jiaxin Huang, Yu Meng, Fang Guo, Heng Ji, and Jiawei Han. Weakly-supervised aspect-based sentiment analysis via joint aspect-sentiment topic embedding. *CoRR*, abs/2010.06705, 2020.

[34] Qingnan Jiang, Lei Chen, Ruifeng Xu, Xiang Ao, and Min Yang. A challenge dataset and effective models for aspect-based sentiment analysis. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6280–6285, 2019.

[35] Hyun jung Park, Minchae Song, and Kyung-Shik Shin. Deep learning models and datasets for aspect term sentiment classification: Implementing holistic recurrent attention on target-dependent memories. *Knowledge-Based Systems*, 187:104825, 2020.

[36] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Draft for third edition, 2020. `https://web.stanford.edu/~jurafsky/slp3/ed3book_dec30 2020.pdf`.

[37] Bamba Kane, A. Jrad, A. Essebbar, Ophélie Guinaudeau, V. Chiesa, Ilhem Quénel, and S. Chau. Cnn-lstm-crf for aspect-based sentiment analysis: A joint method applied to french reviews. *ICAART*, 2021.

[38] Akbar Karimi, Leonardo Rossi, and Andrea Prati. Adversarial training for aspect-based sentiment analysis with bert. 2020.

[39] Akbar Karimi, Leonardo Rossi, and Andrea Prati. Improving bert performance for aspect-based sentiment analysis. 2020.

[40] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[41] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

[42] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, 2016.

[43] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

[44] Yann Lecun. Generalization and network design strategies. *Connectionism in perspective*, 1989.

[45] Hao Li and Wei Lu. Learning explicit and implicit structures for targeted sentiment analysis. *CoRR*, abs/1909.07593, 2019.

[46] Xin Li, Lidong Bing, Piji Li, and Wai Lam. A unified model for opinion target extraction and target sentiment prediction. 2019.

[47] Xin Li, Lidong Bing, Wenxuan Zhang, and Wai Lam. Exploiting BERT for end-to-end aspect-based sentiment analysis. *CoRR*, abs/1910.00883, 2019.

[48] Zheng Li, Xin Li, Ying Wei, Lidong Bing, Yu Zhang, and Qiang Yang. Transferable end-to-end aspect-based sentiment analysis with selective adversarial learning. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4590–4600, 2019.

[49] Zheng Li, Ying Wei, Yu Zhang, Xiang Zhang, Xin Li, and Qiang Yang. Ex-

ploiting coarse-to-fine task transfer for aspect-level sentiment classification. *CoRR*, abs/1811.10999, 2018.

[50] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[51] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Adversarial multi-task learning for text classification. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1–10, 2017.

[52] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[53] Huaishao Luo, Lei Ji, Tianrui Li, Nan Duan, and Daxin Jiang. Grace: Gradient harmonized and cascaded labeling for aspect-based sentiment analysis. 2020.

[54] Huaishao Luo, Tianrui Li, Bing Liu, and Junbo Zhang. DOER: dual cross-shared RNN for aspect term-polarity co-extraction. *CoRR*, abs/1906.01794, 2019.

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

[56] Takeru Miyato, Andrew M. Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. 2017.

[57] Rajdeep Mukherjee, Shreyas Shetty, Subrata Chattopadhyay, Subhadeep Maji, Samik Datta, and Pawan Goyal. Reproducibility, replicability and beyond: Assessing production readiness of aspect based sentiment analysis in the wild. 2021.

[58] Thien Nguyen and Kiyoaki Shirai. Phrasernn: Phrase recursive neural network for aspect-based sentiment analysis. pages 2509–2514, 2015.

[59] Matan Orbach, Orith Toledo-Ronen, Artem Spector, Ranit Aharonov, Yoav Katz, and Noam Slonim. Yaso: A new benchmark for targeted sentiment analysis. 2020.

[60] Aitor García Pablos, Montse Cuadros, and German Rigau. W2VLDA: almost unsupervised system for aspect based sentiment analysis. *CoRR*, abs/1705.07687, 2017.

[61] Jian Peng, Liefeng Bo, and Jinbo Xu. Conditional neural fields. *Advances in Neural Information Processing Systems*, 22, 2009.

[62] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[63] Minh Hieu Phan and Philip O. Ogunbona. Modelling context and syntactical features for aspect-based sentiment analysis. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3211–3220, 2020.

[64] Maria Pontiki, Dimitrios Galanis, John Pavlopoulos, Harris Papageorgiou, Ion Androutsopoulos, and Suresh Manandhar. Semeval-2014 task 4: Aspect based sentiment analysis. *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 27–35, 2014.

[65] Maria Pontiki, Dimitris Galanis, Haris Papageorgiou, Ion Androutsopoulos, Suresh Manandhar, Mohammad AL-Smadi, Mahmoud Al-Ayyoub, Yanyan Zhao, Bing Qin, Orphée De Clercq, Véronique Hoste, Marianna Apidianaki, Xavier Tannier, Natalia Loukachevitch, Evgeniy Kotelnikov, Nuria Bel, Salud María Jiménez-Zafra, and Gülşen Eryiğit. SemEval-2016 task 5: Aspect based sentiment analysis. *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 19–30, 2016.

[66] Maria Pontiki, Dimitris Galanis, Haris Papageorgiou, Suresh Manandhar, and Ion Androutsopoulos. SemEval-2015 task 12: Aspect based sentiment analysis. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 486–495, 2015.

[67] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.

[68] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[69] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016.

[70] Alexander Rietzler, Sebastian Stabinger, Paul Opitz, and Stefan Engl. Adapt or get left behind: Domain adaptation through BERT language model finetuning for aspect-target sentiment classification. *CoRR*, abs/1908.11860, 2019.

[71] Sebastian Ruder. *Neural Transfer Learning for Natural Language Processing*. PhD thesis, National University of Ireland, Galway, 2019.

[72] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, pages 318–362, 1986.

[73] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.

[74] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[75] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, 2016.

[76] Youwei Song, Jiahai Wang, Tao Jiang, Z. Liu, and Y. Rao. Attentional encoder network for targeted sentiment classification. *ArXiv*, abs/1902.09314, 2019.

[77] Kai Sun, Richong Zhang, Samuel Mensah, Yongyi Mao, and Xudong Liu. Aspect-level sentiment analysis via convolution over dependency tree. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5679–5688, 2019.

[78] Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. Target-dependent sentiment classification with long short term memory. *CoRR*, abs/1512.01100, 2016.

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

[80] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[81] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. 2017.

[82] Qianlong Wang and Jiangtao Ren. Label correction model for aspect-based sentiment analysis. pages 822–832, 2020.

[83] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based LSTM for aspect-level sentiment classification. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, 2016.

[84] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. 2015.

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

[86] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[87] Xiaoyu Xing, Zhijing Jin, Di Jin, Bingning Wang, Qi Zhang, and Xuanjing Huang. Tasty burgers, soggy fries: Probing aspect robustness in aspect-based sentiment analysis. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3594–3605, 2020.

[88] Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. Double embeddings and cnn-based sequence labeling for aspect extraction. *CoRR*, abs/1805.04601, 2018.

[89] Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. BERT post-training for review reading comprehension and aspect-based sentiment analysis. *CoRR*, abs/1904.02232, 2019.

[90] Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. Dombert: Domain-oriented language model for aspect-based sentiment analysis. 2020.

[91] Wei Xue and Tao Li. Aspect based sentiment analysis with gated convolutional networks. *CoRR*, abs/1805.07043, 2018.

[92] Hang Yan, Junqi Dai, Tuo ji, Xipeng Qiu, and Zheng Zhang. A unified generative framework for aspect-based sentiment analysis. 2021.

[93] Heng Yang, Biqing Zeng, Jianhao Yang, Youwei Song, and Ruyang Xu. A multi-task learning model for chinese-oriented aspect polarity classification and aspect term extraction. *CoRR*, abs/1912.07976, 2020.

[94] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

[95] Biqing Zeng, Heng Yang, Ruyang Xu, Wu Zhou, and Xuli Han. Lcf: A local context focus mechanism for aspect-based sentiment classification. *Applied Sciences*, 9:3389, 2019.

[96] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2021. `https://d2l.ai`.

[97] Chen Zhang, Qiuchi Li, and Dawei Song. Aspect-based sentiment classification with aspect-specific graph convolutional networks. *CoRR*, abs/1909.03477, 2019.

[98] Pinlong Zhao, Linlin Hou, and Ou Wu. Modeling sentiment dependencies with graph convolutional networks for aspect-level sentiment classification. *CoRR*, abs/1906.04501, 2019.

[99] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *CoRR*, abs/1506.06724, 2015.

Appendix

Appendix A: Reported Results

		Domain	**Beauty/Hotels → Rest14**		**Rest/Beauty/Hotels → Lap14**		**Rest/Beauty/Hotels → Twitter**	
Model Category	**Model**	**Metric**	**Accuracy**	**F1 Macro**	**Accuracy**	**F1 Macro**	**Accuracy**	**F1 Macro**
Extra data	MGALN	average of AC-AT pairs	81.49	71.48	76.21	71.42	74.62	73.53
		Domain	**Lap14 → Rest14**		**Rest14 → Lap14**			
Model Category	**Model**	**Metric**	**Accuracy**	**F1 Macro**	**Accuracy**	**F1 Macro**		
Extra data	BERT-ADA Laptop	mean of 9 runs	80.68	72.93	**77.92**	**72.99**		
	BERT-ADA Restaurant	mean of 9 runs	**83.68**	72.91	76.16	70.46		
	BERT-ADA Joint	mean of 9 runs	82.23	**73.03**	75.91	69.84		
		Domain	**Lapt14+Rest14 → Rest14**		**Lapt14+Rest14 → Lap14**			
Model Category	**Model**	**Metric**	**Accuracy**	**F1 Macro**	**Accuracy**	**F1 Macro**		
Extra data	BERT-ADA Laptop	mean of 9 runs	86.22	79.79	**80.23**	75.77		
	BERT-ADA Restaurant	mean of 9 runs	**87.89**	81.05	79.14	74.93		
	BERT-ADA Joint	mean of 9 runs	87.69	**81.20**	79.94	**78.74**		
		Domain	**Rest14-16 → Lap14**	**Lap14 → Rest14-16**				
Model Category	**Model**	**Metric**	**F1 Micro**	**F1 Micro**				
Collapsed	ADS-SAL	mean of 5 runs	34.13	43.04				
	BERT-B-UDA	mean of 5 runs	33.68	45.46				
	BERT-E-UDA	mean of 5 runs	**43.95**	**49.52**				

Table 13: Accuracy and F1 Macro Scores for 3-class-ATSC of models trained in a cross-domain way and with reported values. Bold printed values indicate the best model of one category.

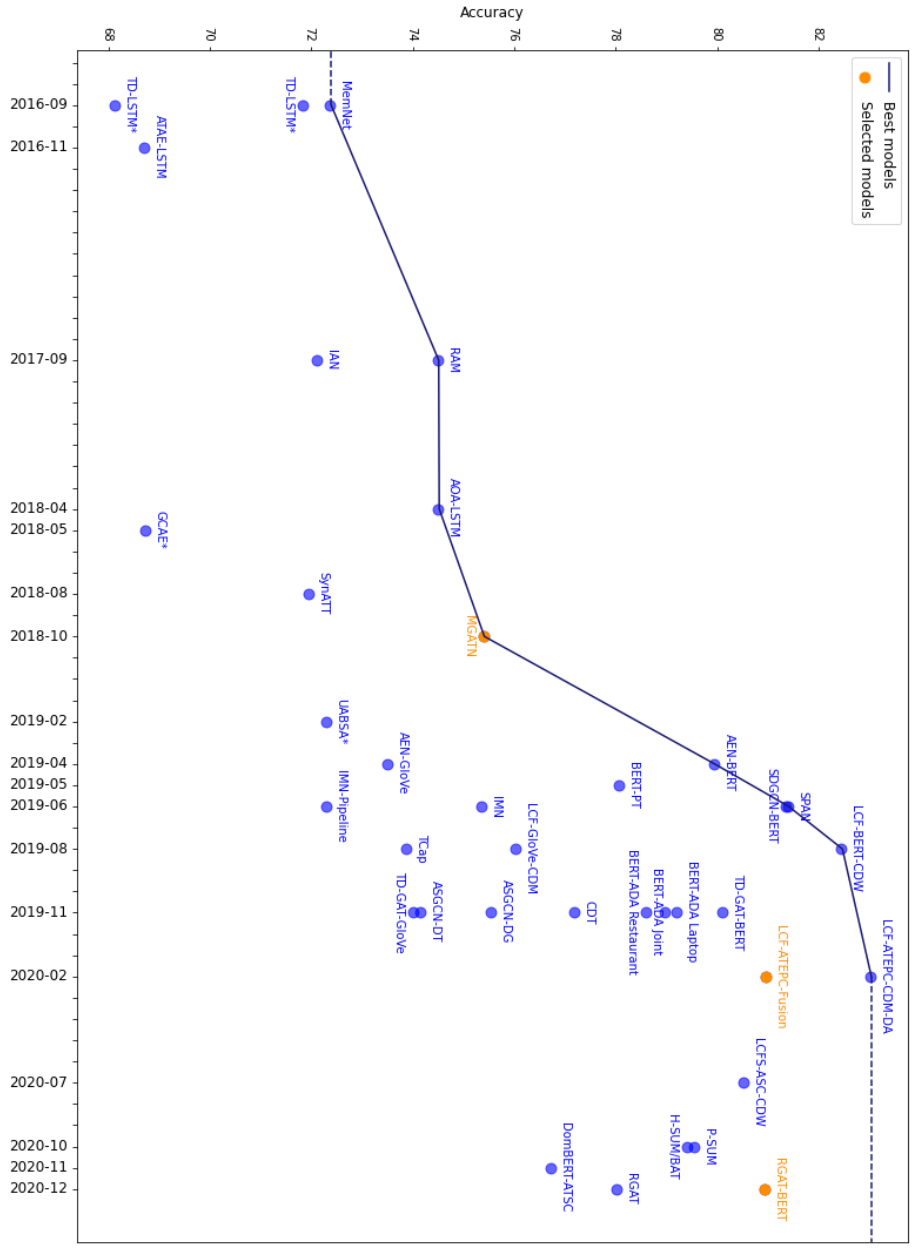


Figure 62: Development of reported ATSC Accuracy on SemEval-14 Laptops. An asterisk indicates that the corresponding values are not taken from the original papers.

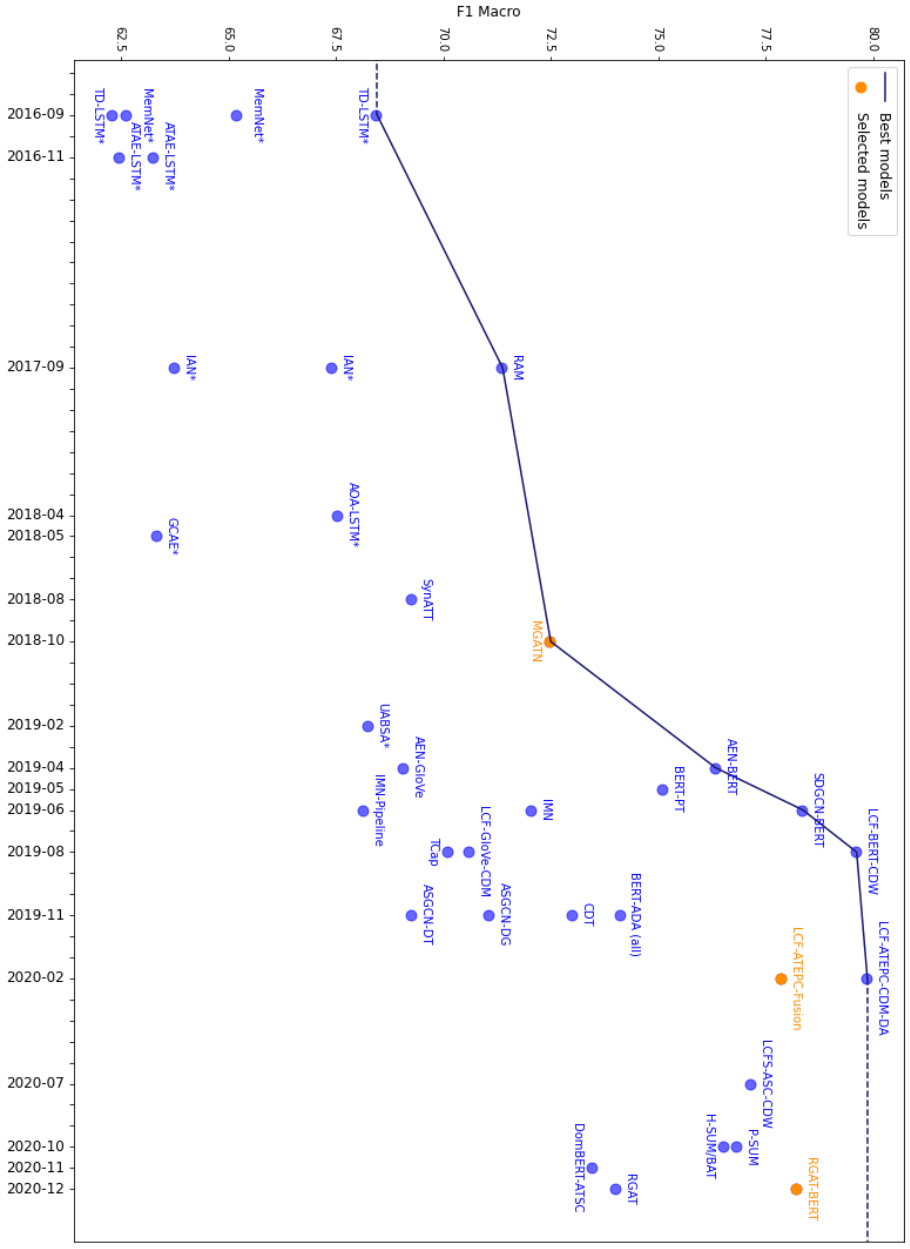


Figure 63: Development of reported ATSC F1 Macro on SemEval-14 Laptops. An asterisk indicates that the corresponding values are not taken from the original papers.

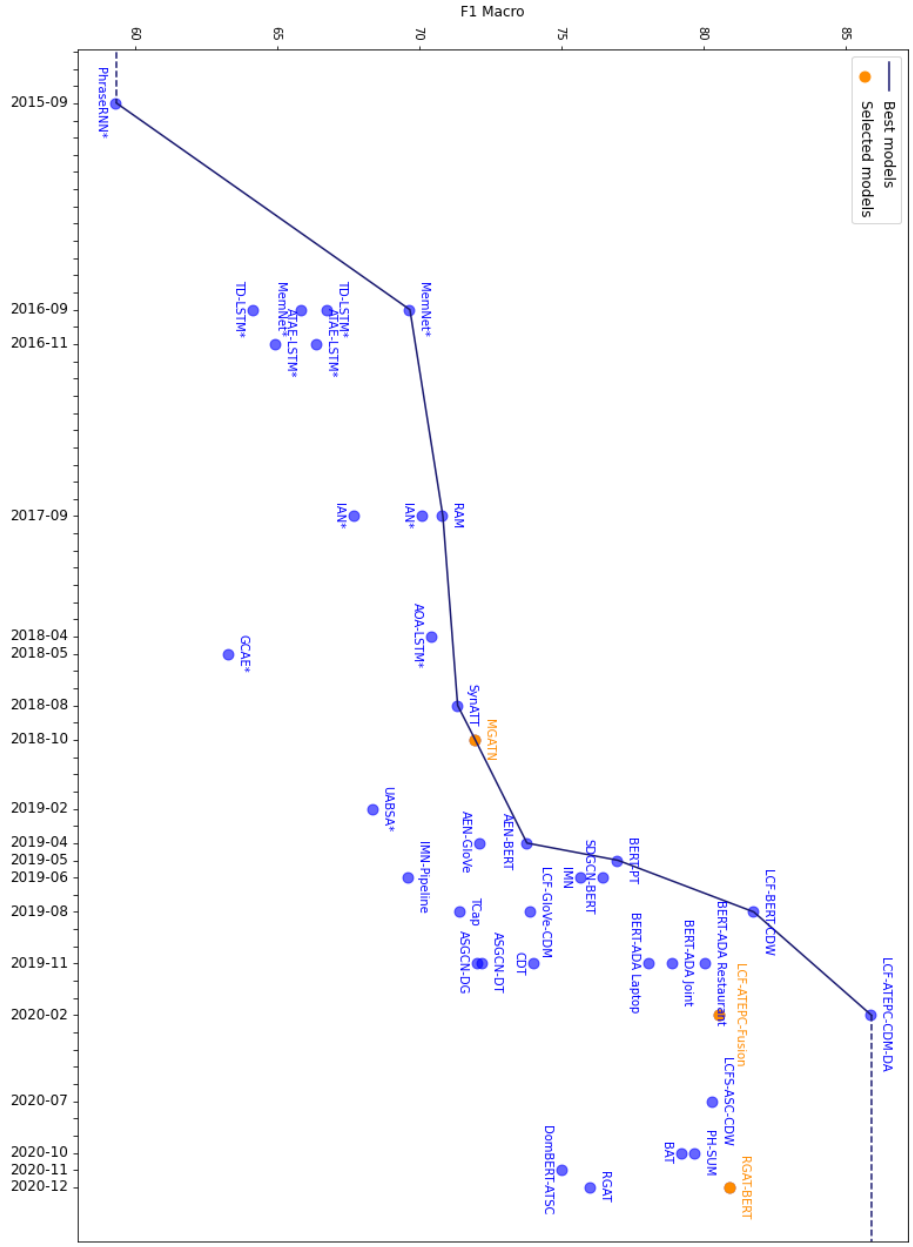


Figure 64: Development of reported ATSC F1 Macro on SemEval-14 Restaurants. An asterisk indicates that the corresponding values are not taken from the original papers.

Appendix B: Twitter Data

In order to receive a data set based on Tweets, [17] queried the Twitter API using names of people, companies and products as keywords. For instance, the top three aspects in terms of frequency are "Britney Spears", "Lindsay Lohan" and "Harry Potter" in both training and test set. These words simultaneously act as aspect terms which were classified as negative, neutral and positive manually. The training set consists of 6,243 unique sentences. The reported number of 6,248 is higher due to duplicates. The test set includes 692 sentences and no duplicates. The authors made sure that both data sets are balanced by providing class proportions of 25%, 50% and 25% for positive, neutral and negative, respectively. The exact statistics can be found in Tables 14. As the table shows, each sentence only contains exactly one aspect term. This makes ATSC rather easy, since the sentiment based on the whole sentence could be taken as the aspect-based sentiment. Thus, we did to use this data set for our practical evaluations.

Source & Domain	Subset	Sentences in total	Sentences w/o duplicates	Sentences labeled for ATSC	Aspect Terms in total	Positive Aspect Terms	Negative Aspect Terms	Neutral Aspect Terms	Conflict Aspect Terms
Twitter	Training	6,248	6,243	6,243	6,243	1,56	1,56	3,123	0
	Test	692	692	692	692	173	173	346	0

Table 14: Number of sentences, aspect terms and aspect term polarities in the Twitter data set. Multi-Sentiment sentences are those with at least two different polarities.

Appendix C: Our Results

The following tables show the quantitative results of our experiments. For SemEval-14, five train-validation splits were created out of the original training set. On each split pair, five runs were performed which lead to split-specific means and standard deviations. In the overall mean and deviation, all runs of all splits were included. Consequently, they are based on 25 values for SemEval-14 and ARTS data and five values for MAMS data (as there were no splits applied).

Metric	**Model**	**SemEval-14 Restaurant**						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
Accuracy = F1 Micro	MGATN	74.32 (±1.24)	74.36 (±1.47)	74.70 (±0.73)	73.23 (±1.07)	73.66 (±0.81)	74.05 (±1.14)	81.25
	RGAT-BERT	82.52 (± 0.60)	83.21 (±0.88)	82.00 (±1.13)	82.70 (±0.67)	82.09 (±0.60)	82.50 (±0.86)	86.68
	CapsNetBERT	84.46 (±0.84)	84.07 (±0.92)	84.68 (±0.87)	83.46 (±0.63)	82.77 (±1.40)	83.89 (±1.13)	85.93
	LCF-ATEPC	82.56 (±0.89)	83.09 (±0.49)	82.87 (±1.28)	82.01 (±1.06)	81.78 (±1.52)	82.46 (±1.13)	86.77
F1 Macro	MGATN	62.04 (±2.37)	60.48 (±2.78)	61.34 (±0.99)	59.05 (±3.13)	57.15 (±3.70)	60.01 (±3.08)	71.94
	RGAT-BERT	72.88 (±0.68)	75.00 (±1.72)	72.86 (±2.21)	73.59 (±2.27)	72.39 (±0.81)	73.34 (±1.79)	80.92
	CapsNetBERT	76.21 (±1.59)	76.85 (±0.87)	77.02 (±1.66)	74.50 (±1.06)	72.43 (±4.07)	75.40 (±2.66)	-
	LCF-ATEPC	73.33 (±2.34)	75.17 (±0.38)	74.03 (±2.85)	73.22 (±1.58)	71.38 (±2.76)	73.43 (±2.36)	80.54
F1 Weighted	MGATN	72.83 (±1.56)	71.91 (±1.81)	72.53 (±0.48)	71.08 (±1.75)	70.03 (±2.23)	71.68 (±1.84)	-
	RGAT-BERT	81.03 (±0.54)	82.42 (±1.11)	81.09 (±1.37)	81.80 (±1.32)	80.76 (±0.67)	81.42 (±1.15)	-
	CapsNetBERT	83.50 (±1.00)	83.65 (±0.75)	83.98 (±1.09)	82.48 (±0.71)	81.02 (±2.44)	82.93 (±1.65)	-
	LCF-ATEPC	83.86 (±0.73)	83.80 (±0.70)	83.97 (±0.89)	82.88 (±1.09)	83.61 (±1.37)	83.63 (±0.99)	-

Metric	**Model**	**SemEval-14 Laptop**						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
Accuracy = F1 Micro	MGATN	64.48 (±0.85)	63.86 (±2.66)	64.67 (±1.78)	64.08 (±0.88)	63.61 (±0.85)	64.14 (±1.49)	75.39
	RGAT-BERT	76.14 (±1.05)	76.24 (±1.43)	75.27 (±0.63)	76.39 (±1.19)	75.20 (±1.02)	75.85 (±1.13)	80.94
	CapsNetBERT	76.21 (±1.01)	77.52 (±1.80)	77.49 (±1.13)	77.55 (±1.22)	77.84 (±1.70)	77.32 (±1.41)	-
	LCF-ATEPC	76.22 (±2.37)	76.93 (±1.24)	75.61 (±1.35)	77.58 (±1.16)	75.44 (±1.16)	76.36 (±1.62)	80.97
F1 Macro	MGATN	56.98 (±0.92)	56.36 (±3.09)	55.82 (±2.29)	56.81 (±2.87)	56.93 (±2.05)	56.58 (±2.21)	72.47
	RGAT-BERT	70.54 (±1.54)	70.86 (±2.51)	69.49 (±1.13)	71.94 (±1.62)	70.59 (±1.23)	70.68 (±1.73)	78.2
	CapsNetBERT	70.76 (±1.87)	72.92 (±2.45)	72.68 (±1.72)	72.56 (±2.43)	73.39 (±3.21)	72.46 (±2.37)	-
	LCF-ATEPC	70.23 (±3.60)	72.43 (±0.89)	70.20 (±1.58)	73.34 (±1.72)	70.63 (±2.07)	71.37 (±2.37)	77.86
F1 Weighted	MGATN	63.71 (±0.66)	63.20 (±2.63)	62.52 (±1.87)	63.22 (±2.30)	63.50 (±1.48)	63.23 (±1.79)	-
	RGAT-BERT	75.16 (±1.26)	75.37 (±1.87)	74.38 (±1.00)	76.14 (±1.32)	74.99 (±0.97)	75.21 (±1.34)	-
	CapsNetBERT	75.29 (±1.47)	77.20 (±2.09)	76.97 (±1.38)	76.73 (±2.00)	77.43 (±2.59)	76.72 (±1.95)	-
	LCF-ATEPC	77.33 (±1.93)	77.08 (±1.72)	76.43 (±1.37)	77.74 (±0.99)	75.59 (±1.23)	76.84 (±1.56)	-

Metric	**Model**	**MAMS**						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
Accuracy = F1 Micro	MGATN	-	-	-	-	-	61.95 (±3.17)	-
	RGAT-BERT	-	-	-	-	-	79.79 (±0.55)	84.52
	CapsNetBERT	-	-	-	-	-	83.04 (±0.70)	83.39
	LCF-ATEPC	-	-	-	-	-	78.94 (±0.56)	-
F1 Macro	MGATN	-	-	-	-	-	59.25 (±3.78)	-
	RGAT-BERT	-	-	-	-	-	79.24 (±0.69)	83.74
	CapsNetBERT	-	-	-	-	-	82.44 (±0.81)	-
	LCF-ATEPC	-	-	-	-	-	78.43 (±0.64)	-
F1 Weighted	MGATN	-	-	-	-	-	61.24 (±3.53)	-
	RGAT-BERT	-	-	-	-	-	79.77 (±0.59)	-
	CapsNetBERT	-	-	-	-	-	83.04 (±0.74)	-
	LCF-ATEPC	-	-	-	-	-	78.94 (±0.50)	-

Metric	**Model**	**ARTS Restaurant**						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
Accuracy = F1 Micro	MGATN	57.19 (±1.42)	57.61 (±2.47)	58.04 (±1.91)	57.74 (±1.01)	58.45 (±0.57)	57.81 (±1.54)	-
	RGAT-BERT	72.32 (±0.83)	73.20 (±1.52)	72.57 (±2.37)	71.38 (±1.54)	72.44 (±1.09)	72.38 (±1.54)	-
	CapsNetBERT	78.80 (±1.17)	78.38 (±0.75)	78.91 (±1.98)	78.80 (±0.77)	75.23 (±5.86)	78.02 (±2.98)	-
	LCF-ATEPC	73.59 (±0.55)	73.92 (±1.43)	74.88 (±1.58)	71.11 (±3.27)	73.13 (±0.90)	73.32 (±2.09)	-
F1 Macro	MGATN	47.03 (±0.76)	43.15 (±6.16)	43.17 (±7.18)	45.96 (±1.69)	43.13 (±2.40)	44.49 (±4.40)	-
	RGAT-BERT	63.53 (±2.11)	66.20 (±2.04)	64.77 (±3.19)	62.99 (±3.07)	63.70 (±1.27)	64.24 (±2.51)	-
	CapsNetBERT	71.22 (±1.36)	71.94 (±0.65)	71.63 (±2.65)	71.02 (±1.32)	65.87 (±7.49)	70.34 (±4.06)	-
	LCF-ATEPC	64.94 (±1.38)	66.82 (±1.76)	66.55 (±2.61)	62.91 (±2.71)	63.84 (±0.99)	65.01 (±2.39)	-
F1 Weighted	MGATN	54.89 (±0.81)	52.59 (±3.92)	52.79 (±5.22)	55.02 (±0.25)	52.96 (±1.44)	53.65 (± 2.96)	-
	RGAT-BERT	70.96 (±1.15)	72.65 (±1.66)	72.03 (±2.49)	70.61 (±2.07)	71.41 (±1.16)	71.53 (±1.79)	-
	CapsNetBERT	78.12 (±1.19)	78.29 (±0.48)	78.55 (±1.85)	78.19 (±0.84)	74.20 (±6.39)	77.47 (±3.25)	-
	LCF-ATEPC	74.74 (±0.37)	74.41 (±1.36)	75.83 (±1.34)	72.04 (±3.37)	74.70 (±0.91)	74.34 (±2.07)	-
ARS Accuracy	MGATN	9.13 (±1.42)	9.50 (±2.51)	10.00 (±3.03)	9.90 (±1.00)	9.57 (±0.67)	9.62 (±1.81)	-
	RGAT-BERT	35.17 (±3.16)	36.47 (±3.02)	35.47 (±4.52)	33.33 (±3.31)	35.73 (±3.14)	35.23 (±3.34)	-
	CapsNetBERT	29.96 (±3.11)	27.70 (±2.60)	28.75 (±5.70)	29.74 (±1.84)	21.43 (±8.50)	27.52 (±5.57)	55.36
	LCF-ATEPC	39.16 (±1.66)	40.30 (±3.24)	40.10 (±3.89)	34.02 (±6.20)	39.16 (±3.12)	38.55 (±4.28)	-

Metric	Model	ARTS Laptop						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
Accuracy = F1 Micro	MGATN	52.31 (±0.20)	52.14 (±1.56)	52.29 (±1.20)	52.19 (±0.83)	52.83 (±0.77)	52.35 (±0.96)	-
	RGAT-BERT	65.81 (±3.23)	64.66 (±5.33)	66.31 (±1.68)	68.25 (±1.35)	66.31 (±2.56)	66.27 (±3.12)	-
	CapsNetBERT	66.68 (±6.17)	72.51 (±0.73)	70.80 (±2.32)	71.97 (±1.48)	71.84 (±1.85)	79.77 (±3.60)	-
	LCF-ATEPC	69.38 (±1.78)	67.57 (±2.58)	68.99 (±0.74)	69.45 (±2.12)	67.50 (±1.56)	68.58 (±1.91)	-
F1 Macro	MGATN	46.58 (±0.76)	46.86 (±2.05)	44.91 (±1.69)	46.81 (±2.63)	48.41 (±1.57)	46.71 (±2.03)	-
	RGAT-BERT	60.30 (±4.14)	59.96 (±5.90)	61.46 (±1.73)	64.37 (±1.69)	62.75 (±2.62)	61.77 (±3.68)	-
	CapsNetBERT	61.61 (±6.59)	68.53 (±1.71)	66.57 (±3.09)	67.36 (±2.66)	68.29 (±3.51)	66.47 (±4.38)	-
	LCF-ATEPC	63.90 (±2.70)	63.79 (±3.44)	64.19 (±1.64)	66.02 (±2.87)	63.81 (±1.99)	64.34 (±2.53)	-
F1 Weighted	MGATN	50.54 (±0.45)	50.67 (±1.20)	49.60 (±1.30)	50.83 (±1.70)	52.10 (±1.00)	50.75 (±1.37)	-
	RGAT-BERT	64.30 (±3.69)	63.47 (±5.71)	65.23 (±1.58)	67.60 (±1.52)	65.73 (±2.70)	65.27 (±3.43)	-
	CapsNetBERT	65.34 (±6.43)	71.89 (±1.18)	70.02 (±2.69)	70.96 (±2.11)	71.31 (±2.61)	69.91 (±4.00)	-
	LCF-ATEPC	70.71 (±1.68)	68.02 (±2.25)	69.94 (±0.60)	69.79 (±1.80)	67.96 (±1.59)	69.28 (±1.89)	-
ARS Accuracy	MGATN	11.68 (±0.83)	12.12 (±1.43)	11.14 (±1.78)	12.41 (±1.34)	13.87 (±0.93)	12.24 (±1.52)	-
	RGAT-BERT	34.31 (±6.26)	31.68 (±10.32)	34.84 (±3.83)	39.17 (±2.18)	34.01 (±6.34)	34.80 (±6.36)	-
	CapsNetBERT	35.52 (±10.83)	46.13 (±1.61)	41.75 (±3.66)	44.33 (±3.01)	42.34 (±2.90)	42.01 (±6.21)	25.86
	LCF-ATEPC	41.98 (±2.42)	37.77 (±4.95)	40.69 (±0.75)	40.94 (±4.09)	37.08 (±3.60)	39.69 (±3.73)	-

Table 15: Our performance results (mean ± standard deviation) for ATSC models. For SemEval-14 Restaurants and Laptops as well as for MAMS, no ARS Accuracy is measured.

Metric	Model	SemEval-14 Restaurant						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
F1 Micro	BERT+TFM	74.27 (±1.25)	74.90 (±0.84)	75.90 (±0.53)	74.55 (±0.54)	74.96 (±0.46)	74.91 (±0.91)	73.98
	GRACE	77.78 (±0.65)	77.40 (±0.54)	78.43 (±0.75)	77.90 (±0.95)	77.84 (±0.80)	77.87 (±0.76)	77.26
F1 Macro	BERT+TFM	66.71 (±1.52)	67.16 (±1.39)	69.37 (±0.73)	66.49 (±0.84)	67.63 (±1.20)	67.47 (±1.50)	-
	GRACE	72.05 (±0.88)	71.40 (±0.99)	72.41 (±1.22)	72.13 (±1.35)	71.36 (±1.49)	71.87 (±1.18)	-
Precision	BERT+TFM	74.25 (±1.46)	74.72 (±1.00)	76.04 (±0.86)	74.29 (±0.35)	75.46 (±0.85)	74.95 (±1.14)	-
	GRACE	76.25 (±0.79)	76.08 (±0.90)	77.17 (±0.82)	76.86 (±0.87)	76.35 (±0.83)	76.54 (±0.87)	-
Recall	BERT+TFM	74.30 (±1.30)	75.10 (±1.01)	75.78 (±0.57)	74.82 (±0.90)	74.48 (±1.07)	74.90 (±1.06)	-
	GRACE	79.37 (±0.75)	78.78 (±0.22)	79.75 (±0.87)	78.99 (±1.12)	79.41 (±0.83)	79.26 (±0.82)	-
ATE F1 Micro	GRACE	87.88 (±0.60)	88.29 (±0.30)	88.38 (±0.42)	88.64 (±0.41)	88.66 (±0.53)	88.37 (±0.51)	-

Metric	Model	SemEval-14 Laptop						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
F1 Micro	BERT+TFM	63.53 (±0.93)	63.92 (±0.81)	64.03 (±1.56)	64.16 (±0.99)	64.09 (±1.05)	63.95 (±1.03)	60.80
	GRACE	70.04 (±1.33)	68.84 (±0.27)	69.10 (±1.68)	69.10 (±1.17)	69.49 (±1.28)	69.31 (±1.21)	70.71
F1 Macro	BERT+TFM	56.92 (±2.33)	57.04 (±2.39)	57.92 (±2.66)	58.62 (±1.31)	58.09 (±1.49)	57.72 (±2.03)	-
	GRACE	65.29 (±1.90)	64.00 (±0.39)	64.95 (±2.42)	64.51 (±0.98)	65.06 (±1.57)	64.76 (±1.55)	-
Precision	BERT+TFM	65.57 (±1.16)	65.69 (±0.65)	65.19 (±1.61)	65.48 (±0.77)	65.35 (±1.02)	65.46 (±1.02)	63.23
	GRACE	69.77 (±1.47)	68.19 (±0.35)	68.18 (±1.78)	68.64 (±1.60)	68.63 (±1.31)	68.68 (±1.41)	72.38
Recall	BERT+TFM	61.65 (±1.38)	62.26 (±1.37)	62.94 (±1.79)	62.90 (±1.31)	62.90 (±1.33)	62.53 (±1.42)	58.64
	GRACE	70.32 (±1.27)	69.52 (±0.47)	70.06 (±1.69)	69.58 (±0.82)	70.38 (±1.38)	69.97 (±1.16)	69.12
ATE F1 Micro	GRACE	85.99 (±1.51)	85.18 (±0.60)	85.40 (±0.59)	85.98 (±0.72)	85.68 (±0.65)	85.64 (±0.87)	87.93

Metric	Model	MAMS						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
F1 Micro	BERT+TFM	-	-	-	-	-	64.94 (±1.47)	-
	GRACE	-	-	-	-	-	63.48 (±0.60)	-
F1 Macro	BERT+TFM	-	-	-	-	-	65.54 (±1.43)	-
	GRACE	-	-	-	-	-	64.59 (±0.61)	-
Precision	BERT+TFM	-	-	-	-	-	65.01 (±1.90)	-
	GRACE	-	-	-	-	-	62.63 (±0.98)	-
Recall	BERT+TFM	-	-	-	-	-	64.93 (±2.42)	-
	GRACE	-	-	-	-	-	64.37 (±0.86)	-
ATE F1 Micro	GRACE	-	-	-	-	-	75.96 (±0.42)	-

Metric	Model	ARTS Restaurant						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
F1 Micro	BERT+TFM	39.80 (±0.78)	39.34 (±0.44)	39.76 (±0.41)	39.29 (±0.56)	39.28 (±1.01)	39.50 (±0.66)	-
	GRACE	61.86 (±1.53)	63.22 (±1.04)	62.80 (±1.28)	62.44 (±1.71)	63.82 (±2.38)	62.83 (±1.66)	-
F1 Macro	BERT+TFM	36.83 (±0.90)	36.13 (±0.47)	36.80 (±0.50)	36.04 (±0.76)	36.19 (±1.27)	36.40 (±0.84)	-
	GRACE	55.91 (±2.11)	57.22 (±1.11)	56.89 (±1.80)	56.40 (±2.03)	57.18 (±3.46)	56.72 (±2.10)	-
Precision	BERT+TFM	28.21 (±0.62)	27.83 (±0.39)	28.22 (±0.28)	27.77 (±0.46)	27.97 (±0.56)	28.00 (±0.48)	-
	GRACE	60.76 (±1.67)	62.20 (±1.41)	61.63 (±1.62)	61.68 (±1.46)	62.56 (±2.38)	61.76 (±1.71)	-
Recall	BERT+TFM	67.55 (±1.17)	67.17 (±0.99)	67.33 (±0.85)	67.17 (±0.86)	66.01 (±2.72)	67.05 (±1.47)	-
	GRACE	63.02 (±1.65)	64.30 (±0.93)	64.02 (±1.00)	63.24 (±2.02)	65.14 (±2.38)	63.94 (±1.73)	-

ARS Accuracy	BERT+TFM	37.53 (±1.97)	35.60 (±2.25)	35.07 (±2.59)	35.83 (±2.43)	34.30 (±2.81)	35.67 (±2.94)	-
	GRACE	34.71 (±2.98)	38.39 (±3.00)	37.70 (±2.49)	36.78 (±3.81)	40.69 (±4.11)	37.66 (±3.64)	-
ATE F1 Micro	GRACE	50.53 (±0.32)	50.81 (±0.25)	50.78 (±0.26)	50.87 (±0.14)	51.02 (±0.33)	50.83 (±0.29)	-
Metric	**Model**	**ARTS Laptop**						
		Split 1	Split 2	Split 3	Split 4	Split 5	Overall	Reported
F1 Micro	BERT+TFM	34.56 (±1.88)	34.55 (±1.61)	35.06 (±1.64)	35.80 (±.075)	35.50 (±0.39)	35.09 (±1.36)	-
	GRACE	65.90 (±1.75)	64.63 (±3.57)	63.16 (±1.97)	64.36 (±2.47)	64.67 (±1.10)	64.54 (±2.30)	-
F1 Macro	BERT+TFM	31.70 (±2.60)	31.34 (±2.02)	32.44 (±2.22)	33.37 (±0.55)	33.12 (±0.64)	32.39 (±1.84)	-
	GRACE	63.98 (±1.92)	61.54 (±3.97)	60.24 (±2.27)	61.56 (±3.10)	61.90 (±1.85)	61.85 (±2.79)	-
Precision	BERT+TFM	25.91 (±1.29)	25.85 (±0.99)	26.06 (±1.00)	26.56 (±0.53)	26.41 (±0.15)	26.16 (±0.86)	-
	GRACE	66.81 (±2.20)	65.43 (±3.99)	63.83 (±2.04)	65.23 (±3.14)	65.41 (±2.23)	65.34 (±2.75)	-
Recall	BERT+TFM	51.91 (±3.33)	52.14 (±3.33)	53.62 (±3.45)	54.90 (±1.32)	54.15 (±1.42)	53.34 (±2.78)	-
	GRACE	65.03 (±1.48)	63.89 (±3.37)	62.51 (±1.96)	63.54 (±2.08)	64.00 (±1.34)	63.79 (±2.14)	-
ARS Accuracy	BERT+TFM	23.60 (±4.29)	23.26 (±4.83)	24.87 (±4.12)	26.91 (±2.10)	26.23 (±2.47)	24.97 (±3.70)	-
	GRACE	38.80 (±3.90)	36.40 (±3.85)	33.20 (±1.79)	32.80 (±3.03)	36.40 (±4.56)	35.52 (±3.97)	-
ATE F1 Micro	GRACE	52.97 (±0.53)	52.64 (±0.59)	52.62 (±0.36)	53.08 (±0.49)	52.82 (±0.37)	52.83 (±0.47)	-

Table 16: Our performance results (mean ± standard deviation) for ATE+ATSC models. For SemEval-14 Restaurants and Laptops as well as for MAMS, no ARS Accuracy is measured.

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

This paper was not previously presented to another examination board and has not been published.

Munich, 22.07.2021

Elisabeth Lebmeier