Model Knowledge Injection for Aspect-Based Sentiment Classification

Danique Thaens and Flavius Frasincar(✉)[0000−0002−8031−758X]

Erasmus University Rotterdam
PO Box 1738, NL-3000 DR
Rotterdam, the Netherlands
571216dt@student.eur.nl, frasincar@ese.eur.nl

Abstract. The number of online reviews has increased exponentially with the growing use of the Web and social media platforms. To handle the large amount of reviews, machine learning methods are needed to analyze these opinions properly. Aspect-Based Sentiment Classification (ABSC) attempts to determine the sentiment of each aspect within a text, to give a more in-depth overview of the opinion. In this work, a new model called LCR-Rot-hop-Kfont++ is introduced. This model injects knowledge into the state-of-the-art LCR-Rot-hop++ model inspired by previous work on Kformer. We evaluate the model using the SemEval-2015 and SemEval-2016 datasets, for which knowledge is injected during training, testing, and both training and testing using a domain-specific ontology. We show that injecting knowledge improves the accuracy of ABSC. For smaller datasets, it is advised to inject knowledge at training time, while for larger datasets it is advised to inject knowledge at testing time.

Keywords: ABSC · Knowledge injection · Ontology · Hybrid method

1 Introduction

In the last few decades, the Web has become an imperative part of daily life. New, easy ways have been created to search for information, view media, and share your opinion. This provides a rich source of information for companies to find improvements in their business. For smaller companies, going through each review manually is still possible. However, for larger businesses and organisations these opinions often come in high quantities. Hence, novel ways are needed to deduce their meaning. This field is called sentiment analysis [11].

It is most common that sentiment analysis is conducted on a document or sentence level, hence labelling the whole document or sentence as having one sentiment [25]. However, for companies or organisations, aiming to enhance their business, it is important to gain detailed information to identify high-quality parts of the company and those features which need improvement. Consequently, in recent years, Aspect-Based Sentiment Analysis (ABSA), which is responsible for detecting aspects and their related sentiment, has become more popular [25].

ABSA consists of two sub-tasks: Aspect Detection (AD) [20] and Aspect-Based Sentiment Classification (ABSC) [2], where the first is concerned with identifying and extracting all the aspects from a text and the latter focuses on classifying the sentiment towards these extracted targets [16]. This sentiment is often represented as polarity and usually has three categories: positive, negative, and neutral. This work takes the aspects as given and focuses solely on ABSC.

Many methods to classify sentiments have been created in the past. These methods can be divided into three categories: knowledge-based methods, machine learning methods, and hybrid methods, which combine the previous two methods [16]. [17] has demonstrated that knowledge-based methods and machine learning methods can complement each other, leading to improved performance. The hybrid models come in two forms, either the two types of methods are combined in one step or they are sequentially applied in two steps.

Many of the hybrid methods apply a two-step approach. The first step of these methods often uses a knowledge-based approach based on a domain-specific ontology. A domain-specific ontology represents here the concepts, their relationships, and the corresponding sentiments within a specific domain. If the first step is unable to classify, a machine learning method is deployed. One of the methods that provides the best results is the HAABSA++ model [21]. This approach combines a domain-specific ontology in the first step with the state-of-the-art LCR-Rot-hop++ neural network in the second step.

The other type of hybrid method is a one-step method, where knowledge is injected into a machine learning model. This means knowledge from an external source is added to the method to give the model more information. Often, such a method focuses on injecting knowledge into the input sentence before the neural network is utilized or training the neural network on extra data [3]. A newer area of research is on injecting knowledge directly into the neural network. One of these methods is called Kformer and focuses on the feed-forward layers of transformers for knowledge injection [24]. [24] showed that this method outperforms other tested methods that only inject knowledge during input for two knowledge-intensive tasks different from ABSC.

This work contributes to existing research by creating a new model that combines two state-of-the-art approaches applied to ABSC. The new model called LCR-Rot-hop-Kfont++ combines the LCR-Rot-hop++ neural network with the promising knowledge injection method Kformer. Kformer has not been tested on an ABSC task yet, nor has it been combined with another neural network like LCR-Rot-hop++. Moreover, in Kformer, textbooks are used as external knowledge, while in our work a domain-specific ontology is used for knowledge injection. This means that not only the concepts but also their relationships and sentiments are included. The embeddings from the ontology are initialized through OWL2Vec* [4]. This means that for each lexicalization in the ontology, OWL2Vec* creates a vector, that reflects the content and positioning of the lexicalization within the ontology. The Python code is made freely available at `https://github.com/DaniqueT/LCR-Rot-hop-Kfont_plus_plus`.

The remaining work is organised in the following way. Section 2 discusses relevant previous research. Section 3 briefly describes the data used. Section 4 provides the details of the proposed method. Section 5 analyses its performance, while Sect. 6 provides our conclusions and discusses suggestions for future work.

2 Related Works

In the related works we concentrate on the one-step hybrid approaches, as these are the focus of our research. For these methods knowledge can be injected at the input, model, or output level.

Input Knowledge Injection. Injecting knowledge at the input level means enhancing the original text. This is done by adding knowledge from an external resource before the sentence is processed by the neural network. The most simple method is to transform all important concepts from a knowledge graph into a string. This string is then concatenated with the initial text [3]. [3] showed that adding this information before using BERT outperforms regular BERT.

[12] designed a more intricate way of injecting knowledge into the input sequence, which is called Knowledge-enabled BERT (K-BERT). This model injects triplets of a knowledge graph into the original text by using soft-positioning and a visibility matrix. These are needed to limit the amount and effect of knowledge injected into the model. Too much knowledge can lead to knowledge noise, which indicates that the meaning of the original input is changed [12]. Both [3] and [12] find that K-BERT significantly outperforms BERT on a variety of NLP tasks. Especially on smaller datasets, K-BERT seems to perform well [3].

One promising avenue is first using a transformer with knowledge injection at the input level for sentence embedding and afterwards an additional neural network for classification. For example, [6] created the LCR-Rot-hop-ont++ model, which uses K-BERT in combination with the LCR-Rot-hop++ model [21]. Moreover, for the knowledge injection, [6] uses a domain-specific ontology instead of a knowledge graph in an attempt to fit the model even better to the domain data. The authors conclude that their model always outperforms the LCR-Rot-hop++ model without knowledge injection and outperforms the two-step HAABSA++ approach [21] for smaller datasets. This again demonstrates the usefulness of knowledge injection, especially for limited data.

Model Knowledge Injection. Another approach, rather than altering the input text, is to incorporate the knowledge into one of the neural network layers during the analysis. One of the methods exploring this is Kformer [24]. In this method, the authors make use of knowledge neurons found in the Feed Forward Network (FFN) of the transformer, which store implicit knowledge [5]. [24] aims to combine this stored model knowledge with external information by injecting knowledge into the FFN layers of the transformer. [24] finds that injecting knowledge directly into the FFN layers outperforms other approaches that inject

knowledge at the input. This indicates that the neural network is now able to make use of both knowledge stored in the model and external knowledge.

However, the layer in which the knowledge is injected can influence the results. On both tasks tested by [24] the model is enhanced through knowledge injection in the bottom three or the top three layers, while injecting into intermediate layers is beneficial in one case and detrimental in the other. In general, the authors conclude that the top three layers are best able to capture semantic information.

Output Knowledge Injection. A last option is to process the external knowledge in parallel to a transformer model [3]. [13] embeds the text to classify using BERT and concatenate that output with additional features from outside knowledge. This concatenated output is then combined in a MultiLayer Perceptron (MLP) after which it goes to the final output layer. This model outperforms the BERT baseline significantly [13] and even outperforms K-BERT on larger datasets [3]. Hence, this method seems promising but only appears to obtain better results than other knowledge injection models on larger datasets. As our dataset is limited in size, we will not explore this approach further.

3 Data

To evaluate the performance of the model, two different datasets from the SemEval workshop series are used. Specifically, the model is evaluated for the SemEval-2015 Task 12 [15] and the SemEval-2016 Task 5 Subtask 1 [14] datasets. These were specially designed to test the performance of ABSC models. The datasets contain restaurant reviews where each review has a variable number of sentences with one or more aspects within each sentence. Each sentiment towards these aspects is classified as either positive, negative, or neutral.

Each sentence contains one or multiple aspects which can be categorized. For each of these aspects, a polarity is determined. Moreover, the target in these sentences can be implicit. The sentiments associated to implicit targets are more difficult to classify and will not be the focus of this work. Hence, they will be removed from the dataset and left for further research.

The frequency distribution of the two datasets with respect to their polarities, after removing the implicit targets, is shown in Table 1. The SemEval-2015 and SemEval-2016 datasets are quite similar, in both the positive polarity is most frequent, followed by the negative polarity and then by the neutral polarity. SemEval-2016 is the bigger data set containing about 47% more training data and 9% more testing data compared to SemEval-2015. In SemEval-2016, the training and testing data are relatively similar in their frequency distribution. In SemEval-2015 training and test data differ slightly more with less positive and more negative sentiments in the test data compared to the training data.

For the knowledge injection, a domain-specific ontology is used, created for the restaurant domain [18]. This ontology contains concepts belonging to the restaurant domain and the relations between these concepts. In this work, the

Table 1. Distribution of the sentiment polarities in the SemEval-2015 and SemEval-2016 datasets.

	Training Data				Test Data			
	Positive	Neutral	Negative	Total	Positive	Neutral	Negative	Total
SemEval 2015	75.3%	2.8%	21.9%	1279	59.1%	6.1%	34.8%	597
SemEval 2016	70.2%	3.8%	26.0%	1880	74.3%	4.9%	20.8%	650

focus is exclusively only on 0-hops in the ontology. This means that for each word in a sentence that also appears in the ontology, synonyms of that specific word are used.

Lastly, the pre-trained BERT base model [7] is used to obtain embeddings for the data. This model contains 12 transformer layers, 768 hidden dimensions, and 12 self-attention heads. Moreover, a feedforward size of 3072 is used, which is exactly 4 times larger than the hidden size. For consistency, the dimension size of the knowledge representations is set to 100 and is expanded to 400 before concatenating it with the transformer representations.

4 Methodology

In this paper, we propose a methodology that combines Kformer and LCR-Rot-hop++ called LCR-Rot-hop-Kfont++. Kformer performs very well as a knowledge injection model [24], but has not been tested for ABSC yet. Furthermore, LCR-Rot-Hop++ is a state-of-the-art method for ABSC and could possibly even perform better with this added knowledge. We inject knowledge using Kformer methodology into BERT to create sentence embeddings, which are then classified by the LCR-Rot-hop++ model. In Fig. 1 the entire model is shown. In Sect. 4.1 the Kformer method combined with BERT is explained and in Sect. 4.2 the details of the LCR-Rot-hop++ network are provided.

4.1 Kformer

For the knowledge injection into the model, the Kformer method [24] is used together with the BERT model [7]. This approach builds the contextual word embeddings as output which afterwards can be used in the LCR-Rot-hop++ model. BERT is a multi-layer bidirectional transformer encoder [22]. This transformer encoder has two main layers in each of its L transformer layers: a Multi-Head Attention layer followed by an FFN layer. The FFN layer consists of two linear transformations together with a GELU activation function [10]. In this layer, the knowledge is injected. For this knowledge injection, a domain-specific ontology is used displaying the most relevant concepts and their connections. The full FFN layer with knowledge injection can be seen in Fig. 2.

The final output of the two linear layers in the FFN is:

$$\underset{n\times d}{\mathrm{FFN}(H^l)} = f(\underset{n\times d}{H^l} \times \underset{d\times d_m}{U^l}) \times \underset{d_m\times d}{V^l}\ , \tag{4.1}$$

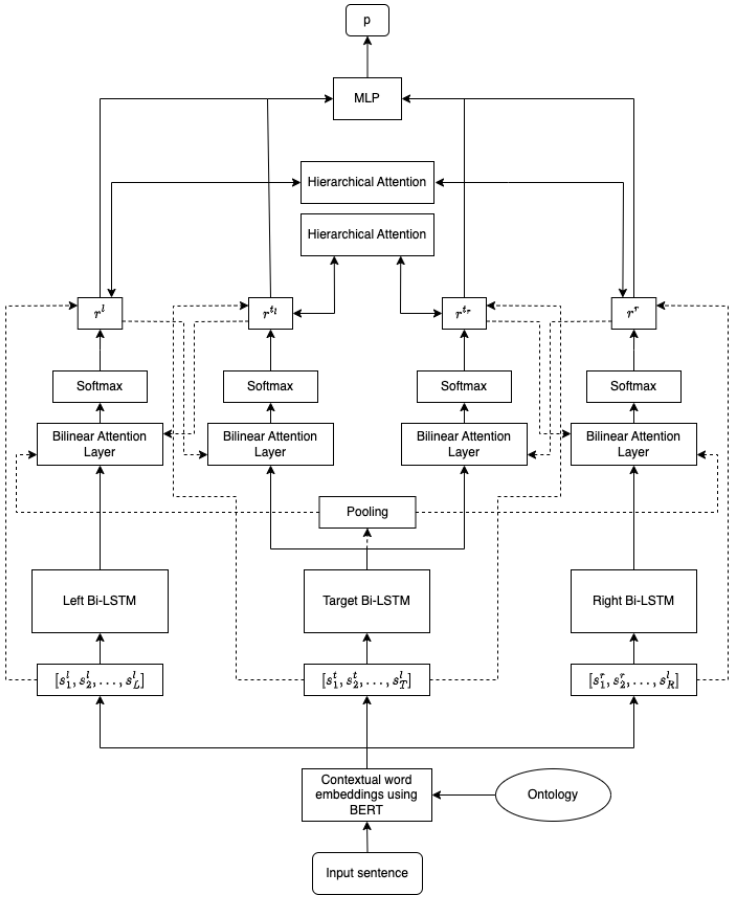

Fig. 1. The complete LCR-Rot-hop-Kfont++ model.

where the final attention output of the layer l is $H^l \in R^{n \times d}$ with n representing the size of the input, and d is the size of the hidden states. $U^l \in R^{d \times d_m}$ and $V^l \in R^{d_m \times d}$ are parameter matrices of the first and second linear layers, respectively, where d_m represents the intermediate size of the transformer within the FFN layer and d is the size of the hidden states and the size of the word embeddings. f is the non-linear GELU function [10].

However, the output of the FFN layer will change because of the knowledge injection. For the knowledge injection, the top N-related concepts are used, this N can be varied to obtain better performance. In this case, we use a domain-specific ontology [18] to obtain the related concepts. The closeness of these concepts to the original word is defined by their k-hop, where each hop means traversing the ontology one more class, with the ontology containing M classes in total. Hence, $hop = 0$ represents only injecting synonyms, since no traversing is done.

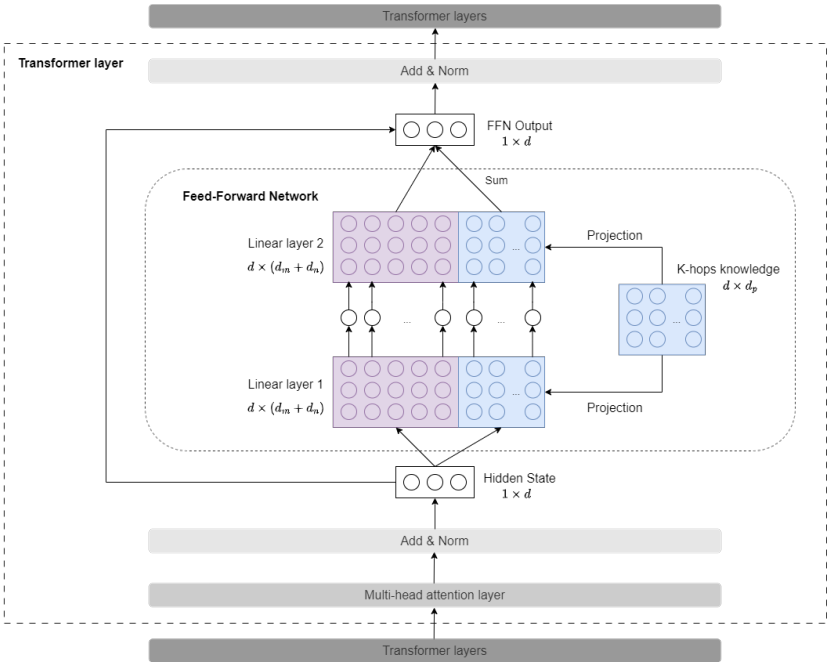

Fig. 2. The FFN layer with knowledge injection.

Differently than [24], each knowledge concept k has its lexical representations tokenized into n tokens that are later embedded into $k_1, k_2, ..., k_n$. The embeddings are initialized using OWL2Vec* [4]. Moreover, these embeddings are updated and learned during the training phase, if this is where knowledge is injected. If knowledge is only injected during testing, random vector embeddings are utilized. To finally represent the external knowledge, the weighted average of these token embeddings is taken. For the 0-hop case, the simple average is used:

$$\underset{d_p \times 1}{k} = \text{Avg}(\underset{d_p \times 1}{k_1}, \underset{d_p \times 1}{k_2}, ..., \underset{d_p \times 1}{k_n}). \tag{4.2}$$

where d_p represents the size of the knowledge token embeddings ($d_p = 100$ in our case).

When k-hops are used the weight of the knowledge concept will decrease the more hops are taken, since this external concept is less closely related to the original. This leads to the following equation:

$$\underset{d_p \times 1}{k} = \sum_{i \in hops} \underset{1 \times 1}{w_i} \sum_{j \in syn(i)} \underset{d_p \times 1}{k_{ij}}, \tag{4.3}$$

where i represents the hops from the original concept, j represents the synonyms used in the layer of hop i, and $k \in R^{d_p}$. The weight can be defined in the following way:

$$w_i = \begin{cases} 1 & \text{if } i = 0 \\ e^{-(n_{hops}+\gamma)} & \text{otherwise,} \end{cases} \tag{4.4}$$

where n_{hops} represents the number of hops in the ontology and γ is a hyperparameter. In this work, only 0-hops are explored, hence solely direct synonyms are used.

This knowledge first needs to be mapped to the vector spaces of the layer l in which it is injected. Both of the two linear layers within the FFN layer are used to project the knowledge:

$$\underset{d_n \times d}{\phi_u^l} = \Pr_u \underset{d_p \times 1}{k} = \underset{d_n \times d_p}{W_u^l} \times \underset{d_p \times d}{[k]_d} \tag{4.5}$$

$$\underset{d_n \times d}{\phi_v^l} = \Pr_v \underset{d_p \times 1}{k} = \underset{d_n \times d_p}{W_v^l} \times \underset{d_p \times d}{[k]_d} \tag{4.6}$$

where Pr represents the knowledge projection, $[k]_d$ represents the repetitiom d times of the vector k, W_u^l, $W_v^l \in R^{d_n \times d_p}$ are the weights of the two linear layers with u representing the first layer and v the second layer. These matrices are randomly initialized and updated during fine-tuning ($d = 768$ in our case). After this projection, the projected knowledge is concatenated to the end of the linear layer and the expanded $U_E^l \in R^{d \times (d_m + d_n)}$ and $V_E^l \in R^{(d_m+d_n) \times d}$ are obtained ($d_n = 400$ and $d_m = 3072$ in our case). This leads to the final output of the FFN layer with knowledge injection:

$$\begin{aligned} \underset{n \times d}{\text{FFN}(H^l)} &= f(\underset{n \times d}{H^l} \times \underset{d \times (d_m+d_n)}{U_E^l}) \times \underset{(d_m+d_n) \times d}{V_E^l} \\ &= f(\underset{n \times d}{H^l} \times [\underset{d \times d_m}{U^l} : \underset{d \times d_n}{\phi_u^{l'}}]) \times [\underset{d_m \times d}{V^l} : \underset{d_n \times d}{\phi_v^l}] \end{aligned} \tag{4.7}$$

where : represents the concatenation operator.

This injection can be done in any of the L layers in the transformer. The information will be processed and aggregated by the following transformer layers. In this work, the knowledge is injected into the top three layers of the model since [24] achieved good performance with this. We use the pre-trained BERT Base model which has 12 layers, 12 attention heads in each of its multi-head attention layers, and 768 hidden states which is also the size of the word embeddings ($L = 12, A = 12, H = 768$). As suggested by [7], the final representation of the words is computed by summing the word embeddings of the final four layers. This means that for word i the final output would be:

$$\underset{d \times 1}{\text{BERT}_i} = \sum_{l=9}^{12} \underset{d \times 1}{H_i^l}. \tag{4.8}$$

4.2 LCR-Rot-hop++

The LCR-Rot-hop++ network is deployed after the embeddings with BERT and Kformer are completed and this model gives the final classification. For the LCR-Rot-hop++ network, the same methodology is used as described by [21] and [23]. The first step is to split the N words of the sentence embeddings S $= [s_1, s_2, ..., s_N]$ into three parts of length L, T, and R which are the left context $[s_1^l, s_2^l, ..., s_L^l]$, target phrase $[s_1^t, s_2^t, ..., s_T^t]$, and right context $[s_1^r, s_2^r, ..., s_R^r]$, respectively. These parts are used as input for three bi-directional Long Short-Term memory modules (bi-LSTMs), namely a left-, centre-, and right-bi-LSTM that model the left context, target phrase, and right context separately. The output of these bi-LSTMs are the hidden states $[h_1^l, h_2^l, ..., h_L^l]$ for the left context, $[h_1^t, h_2^t, ..., h_T^t]$ for the target phrase, and $[h_1^r, h_2^r, ..., h_R^r]$ for the right context.

Then, a two-step rotary attention mechanism is applied over these three parts to capture the surrounding context and highlight the most important words in the target and context. This rotary attention mechanism is applied iteratively three times since [23] found this to be the optimal number. This results in four representations of the input: left context, right context, left-aware target representation, and right-aware target representation. Next, each step of the model will be explained in more detail.

Step 1: Target2Context Attention Mechanism. The first step focuses on finding the most important words from both the left and right contexts. In the first iteration, a representation of the target phrase is needed. The r^{t_p} target vector is calculated using average pooling:

$$\underset{2d\times 1}{r^{t_p}} = \text{pooling}([\underset{2d\times 1}{h_1^t}, \underset{2d\times 1}{h_2^t}, ..., \underset{2d\times 1}{h_T^t}]), \tag{4.9}$$

where d is the size of the word embeddings. During the following iterations, based on the context considered, r^{t_p} is replaced by either r^{t_l} or r^{t_r}, representing the left- or right-aware target representation, respectively.

Next, a representation of the left and right contexts is obtained through a score function f. This is achieved by utilizing the representation of the target phrase (in iteration one, the average target pooling r^{t_p}) and the hidden states of all the context words. For example, for the left context, the score function is:

$$f(\underset{1\times 1}{h_i^l}, r^{t_p}) = \tanh(\underset{1\times 2d}{h_i^{l'}} \times \underset{2d\times 2d}{W_c^l} \times \underset{2d\times 1}{r^{t_p}} + \underset{1\times 1}{b_c^l}), \tag{4.10}$$

where h_i^l are the hidden states, for $i = 1, ..., L$, W_c^l is a weight matrix, and b_c^l a bias term. Then, a softmax function is used with f as input that normalises the scores. The attention normalised scores α_i^l for the left context are calculated as follows:

$$\alpha_i^l = \frac{\exp(f(h_i^l, r^{r_p}))}{\sum_{j=1}^{L} \exp(f(h_j^l, r^{r_p}))}. \tag{4.11}$$

Lastly, the hidden states weighted by the normalised attention scores are used to compute the context representation. For example, for the left context:

$$\underset{2d\times 1}{r^l} = \sum_{i=1}^{L} \underset{1\times 1}{\alpha_i^l} \times \underset{2d\times 1}{h_i^l} \ . \tag{4.12}$$

The right context representation r^r can be computed similarly as in Eqs. 4.10, 4.11, and 4.12, by replacing h_i^l by h_i^r.

Step 2: Context2Target Attention Mechanism. In the second step of the rotary attention mechanism, context-aware target representations are obtained using the previously calculated context representations from step 1. The approach is quite similar to step 1. As before, the score function f is calculated. Now, instead of using the target representation, the context representations are used. Hence, for the left context, the formula becomes:

$$\underset{1\times 1}{f(\ h_i^t\ , r^l)} = \tanh(\ \underset{1\times 2d}{h_i^{t'}} \times \underset{2d\times 2d}{W_t^l} \times \underset{2d\times 1}{r^l} + \underset{1\times 1}{b_t^l}\ ), \tag{4.13}$$

where h_i^t are the hidden states for the words in the target phrase for $i = 1, ..., T$, W_t^l is a weight matrix, and b_t^l is a bias term. Then the normalized attention scores α_i^{tl} are computed using a softmax function:

$$\alpha_i^{tl} = \frac{\exp(f(h_i^t, r^l))}{\sum_{j=1}^{T} \exp(f(h_j^t, r^l))}. \tag{4.14}$$

Lastly, the left-aware target representation can be computed by multiplying the hidden states of the context with their respective attention scores:

$$\underset{2d\times 1}{r^{tl}} = \sum_{i=1}^{T} \underset{1\times 1}{\alpha_i^{tl}} \times \underset{2d\times 1}{h_i^t} \ . \tag{4.15}$$

To obtain the right-aware target representation, Eqs. 4.13, 4.14, and 4.15 are repeated with r^l replaced by r^r.

Hierarchical Attention. The rotary attention mechanism creates an output of four vector representations: left context, right context, left-aware target, and right-aware target. However, these are computed using only local information from the sentence. We provide a higher level of representation by including the information on the sentence level as well. This layer is called hierarchical attention, in which each vector of the previous two steps is updated. This hierarchical attention is applied separately in each of the iterations of the rotary attention mechanism. It is used on the intermediate context and target vector pairs. Firstly, an attention function f is used for each of these vectors:

$$\underset{1\times 1}{f(\ v^i\ )} = \tanh(\ \underset{1\times 2d}{v^{i'}} \times \underset{2d\times 1}{W} + \underset{1\times 1}{b}\ ), \tag{4.16}$$

where $v^i \in \{r^r, r^l, r^{t_r}, r^{t_l}\}$, for $i = 1, 2, 3, 4$, is the representation i of the input sentence, W is a weight matrix, and b is the added bias. Next, new attention scores are computed for each v^i in a context and target pair using f:

$$\alpha^i = \frac{exp(f(v^i)}{\sum_{j=1}^{2} exp(f(v^j))}. \tag{4.17}$$

Now the new representation of each of the old context and target representations $(r^r, r^l, r^{t_r}, r^{t_l})$ can be computed:

$$\underset{2d\times 1}{v^i_{new}} = \underset{2d\times 1}{v^i_{old}} \times \underset{1\times 1}{\alpha^i}\,. \tag{4.18}$$

Classification and Training. To obtain the final representation of the full sentence, the left context representation r^l, right context representation r^r, left-aware target representation r^{t_l}, and the right-aware target representation r^{t_r} are concatenated:

$$\underset{8d\times 1}{v} = [\ \underset{2d\times 1}{r^l}\ ;\ \underset{2d\times 1}{r^{t_l}}\ ;\ \underset{2d\times 1}{r^{t_r}}\ ;\ \underset{2d\times 1}{r^r}\ ]. \tag{4.19}$$

A linear layer then transforms this representation into a vector of length $|C|$ depicting the sentiment prediction vector p, where C represents the sentiment classes. Lastly, a softmax layer predicts the final sentiment polarity of the target phrase:

$$\underset{|C|\times 1}{p} = \text{softmax}(\ \underset{|C|\times 8d}{W_c} \times \underset{8d\times 1}{v} + \underset{|C|\times 1}{b_c}\ ), \tag{4.20}$$

where p is the conditional probability distribution, W_c is a weight matrix, and b_c is a bias term.

The model is trained using the back-propagation algorithm that minimizes the cross-entropy loss function with $L2$ regularization:

$$\underset{1\times 1}{L} = -\sum_j \underset{|C|\times 1}{y_j} \times \log(\ \underset{|C|\times 1}{p_j}\ ) + \lambda||\Theta||^2, \tag{4.21}$$

where y_j is the vector of the true sentiment classification, p_j is the vector containing the predicted sentiment classifications, both for the j-th observation. The weight of the $L2$ regularization term is depicted by λ, and Θ is a parameter set which contains the LSTM parameters, the LCR-Rot-hop++ parameters $\{W_c^l, W_c^r, W_t^l, W_t^r, b_c^l, b_c^r, b_t^l, b_t^r\}$, and the Kformer parameters $\{W_u^l, W_v^l\}$.

A uniform distribution is used to initialize all the weight matrices and biases and updated through the use of stochastic gradient descent with a momentum term. Moreover, the hyperparameters of the model are tuned, specifically the learning rate, dropout rate, L2 regularization term λ, and the momentum term. Of the training data, 80% of the data is used for tuning and the rest for validation. To speed up the hyperparameter tuning, a Tree-structured Parzen Estimator (TPE) algorithm is used [1].

Lastly, the training of the LCR-Rot-hop++ model has changed slightly compared to the method used in HAABSA++ [21]. Previously, the embeddings were created by BERT before the model was trained. Now, to ensure the weights of the knowledge injection matrices can also be learned and tuned, creating the embeddings using BERT is added to the model and these embeddings are also included in the back-propagation. Moreover, the knowledge from the domain ontology is initialized using OWL2Vec* [4]. When knowledge is injected during training times these vectors are updated. However, when knowledge injection is only done during testing, random vector embeddings are used.

5 Results

In this section, we present the results of our LCR-Rot-hop-Kfont++ model. The model is evaluated on the SemEval-2015 and SemEval-2016 datasets using three different variants: with knowledge injection during training, with knowledge injection during testing, and with knowledge injection during both training and testing. These were compared to the benchmark model LCR-Rot-hop++.

First, the hyperparameters of the models are tuned, after which the models are trained using 100 epochs. Moreover, an early stopping rule was implemented. This means that when for 20 epochs the validation loss did not decrease, the training was stopped to prevent overfitting. The knowledge injection takes place in the top three layers of the transformer: layers 10 through 12. This has been chosen because [24] concluded in their research that these layers are usually the best performing and are best able to capture semantic information.

The testing accuracy of the benchmark and our knowledge injection models are displayed in Table 2. Starting with the SemEval-2015 dataset, it can be seen that knowledge injection does improve accuracy. Both knowledge injection during training and knowledge injection in training and testing outperform the LCR-Rot-hop++ benchmark. Knowledge injection only during training is superior and achieved a 0.84 percentage points higher accuracy than the model without knowledge injection. However, only injecting knowledge during testing should not be done for SemEval-2015. This results in a 2.17 percentage points lower out-of-sample accuracy compared to LCR-Rot-hop++.

In the larger dataset of SemEval-2016, the results differ. Now, knowledge injection during testing outperforms the other models with a higher accuracy of 1.39 percentage points compared to the second-best. In this case, injecting knowledge during training leads to the lowest accuracy.

These differences between the models could be explained by how the knowledge is injected and by the size of the datasets. For the smaller dataset SemEval-2015, injecting knowledge only during testing decreases its performance. An explanation for this is that many words injected by the domain ontology, have not appeared in the training data before. Hence, words that the model has not yet learned are injected, possibly changing the meaning of the sentence which makes it more difficult for the model to classify.

In larger datasets like SemEval-2016, the above problem is not present, since there is a bigger chance that words from the sentiment ontology are already included in the training data and hence the model recognizes these. Moreover, when knowledge is only injected during testing, the vector embeddings of the domain ontology knowledge are not updated, instead random vector embeddings are used. [19] have demonstrated that random embeddings can actually outperform trained and updated embeddings. This effect is especially present for larger datasets [19], hence explaining why these random vectors perform better for the SemEval-2016 dataset than for the SemEval-2015 dataset.

For larger datasets, when only injecting knowledge in training, the problem is the other way around. As the results for SemEval-2016 demonstrate, injecting knowledge during training for larger datasets decreases the accuracy. This is due to knowledge noise, which causes the model to incorrectly learn input semantics. The effect of such an incorrect meaning is less big for injecting knowledge during testing. This is a local knowledge noise since the model only predicts the wrong sentiment for that specific sentence but does not train the model using this incorrect meaning. However, when this happens in training, the model uses this input to train the model and it could impact all further predictions. It is more likely that such knowledge noise happens with larger datasets since more sentences are used for training. This could explain why injecting knowledge during training works well for SemEval-2015 but decreases the models' performance for SemEval-2016.

Table 2. Out-of-sample accuracy of LCR-Rot-hop++ and LCR-Rot-hop-Kfont++ for the SemEval-2015 and SemEval-2016 datasets.

	SemEval-2015	SemEval-2016
LCR-Rot-hop++	79.56%	85.69%
LCR-Rot-hop-Kfont++ (train)	**80.40%**	85.54%
LCR-Rot-hop-Kfont++ (test)	77.39%	**87.08%**
LCR-Rot-hop-Kfont++ (train + test)	80.07%	85.69%

6 Conclusion

In this work, we have proposed a new ABSC model: LCR-Rot-hop-Kfont++. We inject knowledge from a domain ontology into the state-of-the-art LCR-Rot-hop++ model for ABSC inspired by previous work on Kformer. When only considering accuracy, knowledge injection generally leads to improved performance. However, the dataset size affects when knowledge injection is most effective. In smaller datasets, knowledge injection should only be done during training so the model can learn a larger amount of words. These extra knowledge can be utilized during testing to classify more sentiments correctly. For larger datasets, on the

other hand, it is better to inject knowledge only during testing since this prevents training on knowledge noise. Moreover, having a larger training dataset increases the chance that injected knowledge during testing was already seen during training and thus boosts the performance of the model.

In this research, only synonyms of the words in the sentence are added, also called 0-hop knowledge injection. Hence for further research, we want to test if 1-hop, 2-hops, or even more hops could give more context to the sentence and would be able to improve the model's accuracy. Moreover, only knowledge was injected into the top layers of the transformer: layers 10 through 12. Exploring the injection of knowledge in other layers would be an intriguing area for further work. Lastly, we would like to explore knowledge injection in sentiment mining models that use argumentation theory [8] or cross-domain similarities [9].

References

1. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: 25th Annual Conference on Neural Information Processing Systems (NIPS 2011). pp. 2546–2554. Curran Associates, Inc. (2011)
2. Brauwers, G., Frasincar, F.: A survey on aspect-based sentiment classification. ACM Computing Surveys **55**(4), 65:1–65:37 (2023)
3. Cadeddu, A., Chessa, A., De Leo, V., Fenu, G., Motta, E., Osborne, F., Recupero, D.R., Salatino, A., Secchi, L.: A comparative analysis of knowledge injection strategies for large language models in the scholarly domain. Engineering Applications of Artificial Intelligence **133**, 108166 (2024)
4. Chen, J., Hu, P., Jimenez-Ruiz, E., Holter, O.M., Antonyrajah, D., Horrocks, I.: OWL2Vec*: Embedding of OWL ontologies. Machine Learning **110**(7), 1813–1845 (2021)
5. Dai, D., Dong, L., Hao, Y., Sui, Z., Chang, B., Wei, F.: Knowledge neurons in pretrained transformers. In: 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022). pp. 8493–8502. ACL (2022)
6. Dekker, R., Gielisse, D., Jaggan, C., Meijers, S., Frasincar, F.: Knowledge injection for aspect-based sentiment classification. In: 34th International Conference on Database and Expert Systems Applications, (DEXA 2023). LNCS, vol. 14147, pp. 173–187. Springer (2023)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019). pp. 4171–4186. ACL (2019)
8. Dragoni, M., da Costa Pereira, C., Tettamanzi, A.G., Villata, S.: SMACk: An argumentation framework for opinion mining. In: 25th International Joint Conference on Artificial Intelligence (IJCAI 2016). pp. 4242–4243. AAAI Press (2016)
9. Dragoni, M., Petrucci, G.: A fuzzy-based strategy for multi-domain sentiment analysis. International Journal of Approximate Reasoning **93**, 59–73 (2018)
10. Hendrycks, D., Gimpel, K.: Bridging nonlinearities and stochastic regularizers with Gaussian error linear units. arXiv preprint arXiv:1606.08415 (2016)
11. Liu, B.: Sentiment analysis: Mining opinions, sentiments, and emotions. Cambridge University Press, second edn. (2020)

12. Liu, W., Zhou, P., Zhao, Z., Wang, Z., Ju, Q., Deng, H., Wang, P.: K-BERT: Enabling language representation with knowledge graph. In: 34th AAAI Conference on Artificial Intelligence (AAAI 2020). pp. 2901–2908. AAAI Press (2020)
13. Ostendorff, M., Bourgonje, P., Berger, M., Moreno-Schneider, J., Rehm, G., Gipp, B.: Enriching BERT with knowledge graph embeddings for document classification. In: 15th Conference on Natural Language Processing (KONVENS 2019). pp. 307–314. University of Erlingen-Nuremberg (2019)
14. Pontiki, M., Galanis, D., Papageorgiou, H., Androutsopoulos, I., Manandhar, S., Al-Smadi, M., Al-Ayyoub, M., Zhao, Y., Qin, B., De Clercq, O., et al.: SemEval-2016 task 5: Aspect based sentiment analysis. In: 10th International Workshop on Semantic Evaluation (SemEval 2016). pp. 19–30. ACL (2016)
15. Pontiki, M., Galanis, D., Papageorgiou, H., Manandhar, S., Androutsopoulos, I.: SemEval-2015 task 12: Aspect based sentiment analysis. In: 9th International Workshop on Semantic Evaluation (SemEval 2015). pp. 486–495. ACL (2015)
16. Schouten, K., Frasincar, F.: Survey on aspect-level sentiment analysis. IEEE Transactions on Knowledge and Data Engineering **28**(3), 813–830 (2016)
17. Schouten, K., Frasincar, F.: Ontology-driven sentiment analysis of product and service aspects. In: 15th International Semantic Web Conference (ESWC 2018). LNCS, vol. 10843, pp. 608–623. Springer (2018)
18. Schouten, K., Frasincar, F., de Jong, F.: Ontology-enhanced aspect-based sentiment analysis. In: 17th International Conference on Web Engineering (ICWE 2017). LNCS, vol. 10360, pp. 302–320. Springer (2017)
19. Tokarchuk, E., Niculae, V.: The unreasonable effectiveness of random target embeddings for continuous-output neural machine translation. In: 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2024). pp. 653–662. ACL (2024)
20. Truşcă, M.M., Frasincar, F.: Survey on aspect detection for aspect-based sentiment analysis. Artifical Intelligence Review **56**(5), 3797–3846 (2022)
21. Truşcă, M.M., Wassenberg, D., Frasincar, F., Dekker, R.: A hybrid approach for aspect-based sentiment analysis using deep contextual word embeddings and hierarchical attention. In: 20th Conference on Web Engineering (ICWE 2020). LNCS, vol. 12128, pp. 365–380. Springer (2020)
22. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: 31st International Conference on Neural Information Processing Systems (NIPS 2017). p. 6000–6010. Curran Associates, Inc. (2017)
23. Wallaart, O., Frasincar, F.: A hybrid approach for aspect-based sentiment analysis using a lexicalized domain ontology and attentional neural models. In: 16th Extended Semantic Web Conference (ESWC 2019). LNCS, vol. 11503, pp. 363–378. Springer (2019)
24. Yao, Y., Huang, S., Dong, L., Wei, F., Chen, H., Zhang, N.: Kformer: Knowledge injection in transformer feed-forward layers. In: 11th International Conference on Natural Language Processing and Chinese Computing (NLPCC 2022). LNCS, vol. 13551, pp. 131–143. Springer (2022)
25. Zhang, W., Li, X., Deng, Y., Bing, L., Lam, W.: A survey on aspect-based sentiment analysis: Tasks, methods, and challenges. IEEE Transactions on Knowledge and Data Engineering **35**(11), 11019–11038 (2023)