Entity-Aware Dependency-Based Deep Graph Attention Network for Comparative Preference Classification

Nianzu Ma[†], Sahisnu Mazumder[†], Hao Wang[♮], Bing Liu[†]
[†]Department of Computer Science, University of Illinois at Chicago, USA
[♮]School of Information Science and Technology, Southwest Jiaotong University, China
jingyima005@gmail.com, sahisnumazumder@gmail.com
cshaowang@gmail.com, liub@uic.edu

Abstract

This paper studies the task of comparative preference classification (CPC). Given two entities in a sentence, our goal is to classify whether the first (or the second) entity is preferred over the other or no comparison is expressed at all between the two entities. Existing works either do not learn entity-aware representations well and fail to deal with sentences involving multiple entity pairs or use sequential modeling approaches that are unable to capture long-range dependencies between the entities. Some also use traditional machine learning approaches that do not generalize well. This paper proposes a novel Entity-aware Dependency-based Deep Graph Attention Network (ED-GAT) that employs a multi-hop graph attention over a dependency graph sentence representation to leverage both the *semantic information* from word embeddings and the *syntactic information* from the dependency graph to solve the problem. Empirical evaluation shows that the proposed model achieves the state-of-the-art performance in comparative preference classification.

1 Introduction

Given a sentence that contains two entities of interest, the task of Comparative Preference Classification is to decide whether there is a comparison between the two entities and if so, which entity is preferred (Jindal and Liu, 2006a; Ganapathibhotla and Liu, 2008; Liu, 2012; Panchenko et al., 2019). For example, considering sentence s_1 (shown in Table 1), there is a comparison between the two underlined entities, and "golf" is preferred over "baseball". This sentence contains explicit comparative predicate "*easier*". The task seems straightforward but is quite challenging due to many counterexamples. For example, s_2 shows that "*better*" may not indicate a comparison. s_3, another counterexample, shows that "*slower*" indeed indicates a comparison, but not between "Perl" and "Python", but between "tools" and "K9".

ID	Sentences
s_1	<u>Golf</u> is *easier* to pick up than <u>baseball</u>.
s_2	I'm considering learning <u>Python</u> and more <u>PHP</u> if any of those would be *better*.
s_3	The <u>tools</u> based on <u>Perl</u> and <u>Python</u> is much *slower* under Windows than <u>K9</u>.

Table 1: Comparative sentence examples. Entities of interest are underlined in each sentence.

Problem statement. Given a sentence $s = \langle w_1, w_2, ..., e_1, ..., e_2, ...w_n\rangle$, where e_1 and e_2 are entities consisting of a single word or a phrase, and e_1 appears before e_2 in the sentence, our goal is to classify the comparative preference direction between these two entities into one of the three classes: {BETTER, WORSE, NONE}. BETTER (WORSE) means e_1 is preferred (not preferred) over e_2. NONE means that there is no comparative relation between e_1 and e_2.

Although closely related, Comparative Preference Classification (CPC) is different from Comparative Sentence Identification (CSI), which is a 2-class classification problem that classifies a sentence as a comparative or a non-comparative sentence. In previous work, Jindal and Liu (2006a) did CSI without considering *which two entities* are involved in a comparison. Tkachenko and Lauw (2015) employed some dependency graph features to approach the CSI task given two entities of interest. In this entity-aware case, syntactic features are crucial. However, not using word embeddings in the model makes the model harder to generalize with a good performance given various ways of expressing comparisons. Panchenko et al. (2019) gave the state-of-the-art result on the CPC task by using a pretrained sentence encoder to produce sentence embeddings as a feature for classification. However, this model is not entity-aware and does not use the dependency graph information.

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 5782–5788
July 5 - 10, 2020.

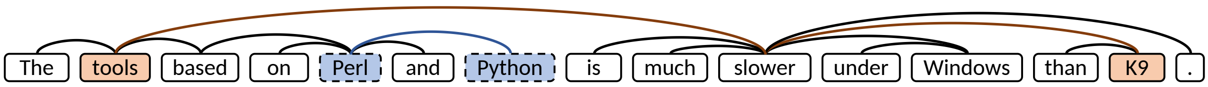

Figure 1: Dependency graph representation of a comparative sentence

For the CPC task, building a model that is entity-aware and also explicitly uses the dependency graph information is vital. We explain the reason as follows. For example, the dependency graph information gives a clue that the underlined entities in s_2 of Table 1 are not involved in a comparison, although there is a comparative indicator "*better*" in the sentence. s_3 (also refer to Figure 1) has two entity pairs, which make an entity-aware model necessary. The pair of entities, *tools* and *K9*, are far away from each other in the sequence. But in the dependency graph, they are just two hops away from each other and one hop away from the key comparative predicate "*slower*". For the pair of entities, *Perl* and *Python*, although both are sequentially near to the word "*slower*", the dependency graph information does not indicate they are involved in a comparison. We see that an entity-aware model can avoid the mistake of taking comparative predicates not associated with the entity pair as an evidence. Also, the dependency graph of a sentence contains important clues that can benefit the comparative preference classification. Methods, which are not entity-aware and do not model dependency structures, are not capable of dealing with the cases in s_2 and s_3.

To address the limitations of the previous models, we propose a novel Entity-aware Dependency-based Deep Graph Attention Network (ED-GAT) for comparative preference classification. We represent a sentence by its dependency graph. This Graph Attention Network (GAT) (Veličković et al., 2018) based model can naturally fuse word semantic information and dependency information within the model. By building a deep graph attention network stacking several self-attention layers, the model can effectively capture long-range dependencies, which is beneficial for identifying the comparison preference direction between two entities. We have applied this model on a real-world benchmark dataset, and the results show that incorporating the dependency graph information greatly helps this task. It outperforms strong and latest baselines, as discussed in the experiments.

2 Proposed Model

In this section, we first give a brief introduction to the GAT model. We then present the proposed ED-GAT model and discuss how to apply it to the CPC task.

2.1 Graph Attention Network (GAT)

The critical component of our model is the Graph Attention Network (GAT) (Veličković et al., 2018), which fuses the graph-structured information and node features within the model. Its masked self-attention layers allow a node to attend to neighborhood features and learn different attention weights for different neighboring nodes.

The node features fed into a GAT layer are $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_i, ...\boldsymbol{x}_n]$, $\boldsymbol{x}_i \in \mathbb{R}^F$, where n is the number of nodes, F is the feature size of each node. The attention mechanism of a typical GAT can be summarized by equation (1).

$$\begin{gathered}\boldsymbol{h}_i^{out} = \Big\|_{k=1}^{K} \sigma\left(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k \boldsymbol{W}^k \boldsymbol{x}_j\right) \\ \alpha_{ij}^k = \frac{exp(f((\boldsymbol{a}^k)^T[\boldsymbol{W}^k\boldsymbol{x}_i \parallel \boldsymbol{W}^k\boldsymbol{x}_j]))}{\sum_{c\in\mathcal{N}_i} exp(f((\boldsymbol{a}^k)^T[\boldsymbol{W}^k\boldsymbol{x}_i \parallel \boldsymbol{W}^k\boldsymbol{x}_c]))}\end{gathered} \tag{1}$$

Here, given the node feature vectors in GAT, node i attends over its 1-hop neighbors $j \in \mathcal{N}_i$. $\Big\|_{k=1}^{K}$ denotes the concatenation of K multi-head attention outputs, $\boldsymbol{h}_i^{out} \in \mathbb{R}^{F'}$ is the output of node i at the current layer, α_{ij}^k is the k-th attention between nodes i and j, $\boldsymbol{W}^k \in \mathbb{R}^{\frac{F'}{K}\times F}$ is linear transformation, $\boldsymbol{a}^k \in \mathbb{R}^{\frac{2F'}{K}}$ is the weight vector, and $f(\cdot)$ is LeakyReLU non-linearity function.

Overall, the input-output for a single GAT layer is summarized as $\boldsymbol{H}^{out} = GAT(\boldsymbol{X}, \boldsymbol{A}; \Theta^l)$. The input is $\boldsymbol{X} \in \mathbb{R}^{n\times F}$ and the output is $\boldsymbol{H}^{out} \in \mathbb{R}^{n\times F'}$, where n is the number of nodes, F is the node feature size, F' is GAT hidden size, and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix of the graph.

2.2 ED-GAT for CPC task

We use the dependency parser in (Chen and Manning, 2014) to convert a sentence into a dependency parse graph. Each word corresponds to a node in the graph. The node features are the word embedding vectors, denoted as $\boldsymbol{x}_i \in \mathbb{R}^F$ corresponding to node i. The input node feature matrix is $\boldsymbol{X} \in \mathbb{R}^{n\times F}$. Note that an entity is either a single word or a multi-word phrase. To treat each entity as

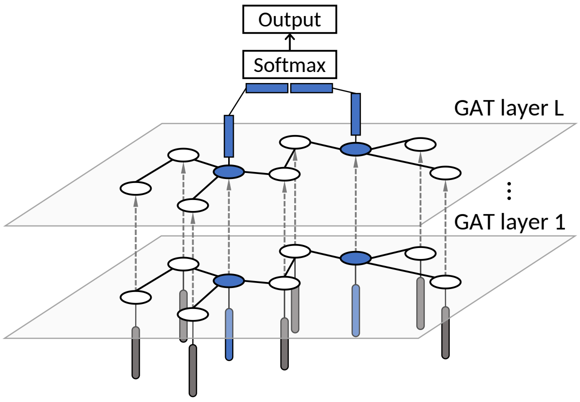

Figure 2: L layer ED-GAT model

one node, we replace the whole entity word/phrase with "EntityA" or "EntityB" before parsing. A multi-word entity embedding is obtained by averaging the embeddings of the words in the entity. We observe that for a given node in the dependency parse graph, both its parents and children contain useful information for the task. To make the ED-GAT model treat both its parents and children as neighbors, we simplify the original directed dependency graph into an undirected graph. The structure of the graph is encoded into an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. ED-GAT does not attend to all neighbors of a given node on an equal basis. The attention weights to the neighbors are automatically learned during training based on their usefulness to the task, regardless of whether they are parents or children in the dependency graph. The higher the attention weight given to a neighbor, the more useful this neighbor is to the task.

In a single GAT layer, a word or an entity in a graph only attends over the local information from 1-hop neighbors. To enable the model to capture long-range dependencies, we stack L layers to make a deep model, which allows information from L-hops away to propagate to this word. Our model is thus a deep graph attention network.

As illustrated in Figure 2, the stacking architecture is represented as $\boldsymbol{H}^{l+1} = GAT(\boldsymbol{H}^l, \boldsymbol{A}; \Theta^l)$, $l \geq 0$, $\boldsymbol{H}^0 = \boldsymbol{X}\boldsymbol{W}_0 + \boldsymbol{b}_0$. The output of the GAT layer l, $\boldsymbol{H}^l_{out} = GAT(\boldsymbol{H}^l, \boldsymbol{A}; \Theta^l)$, is the input for layer $(l+1)$, denoted by $\boldsymbol{H}^{l+1}$. $\boldsymbol{H}^0$ is the initial input. $\boldsymbol{W}_0 \in \mathbb{R}^{F \times F'}$ and $\boldsymbol{b}_0$ are the projection matrix and bias vector. For a L layer ED-GAT model, the output of the final layer is $\boldsymbol{H}^L_{out} \in \mathbb{R}^{n \times F'}$.

We use a mask layer to fetch the two hidden vectors from $\boldsymbol{H}^L_{out}$, which corresponds to the two entities of interest: $(\boldsymbol{h}_{e_1}, \boldsymbol{h}_{e_2}) = \text{Masklayer}(\boldsymbol{H}^L_{out})$. Next, we concatenate these two vectors as: $\boldsymbol{v} = [\boldsymbol{h}_{e_1} \parallel \boldsymbol{h}_{e_2}]$ and use a feed-forward layer with softmax function to project $\boldsymbol{v}$ into classes for prediction. Here using $\boldsymbol{h}_{e_1}$ and $\boldsymbol{h}_{e_2}$ makes the ED-GAT model entity-aware as they are the output of the nodes corresponding to entities e_1 and e_2, each of which attends over its neighbors' features in L hops in the graph and leverages both the word semantics and dependency structure information in learning. The ED-GAT model is trained by minimizing the standard cross-entropy loss over training examples.

3 Related Works

Many papers have been devoted to exploring comparisons in text. For the CSI task, early works include those in (Jindal and Liu, 2006a; Ganapathibhotla and Liu, 2008). More recently, Park and Blake (2012) employed handcrafted syntactic rules to identify comparative sentences in scientific articles. For other languages such as Korean and Chinese, related works include (Huang et al., 2008), (Yang and Ko, 2009) and (Zhang and Jin, 2012).

Other works are interested in identifying entities, aspects and comparative predicates in comparative sentences, e.g., (Jindal and Liu, 2006b), (Hou and Li, 2008), (Kessler and Kuhn, 2014), (Kessler and Kuhn, 2013), and (Feldman et al., 2007). Ganapathibhotla and Liu (2008) used lexicon properties to determine the preferred entities given the output of (Jindal and Liu, 2006b), which is quite different from our task.

There are also works related to product ranking using comparisons, such as those in (Kurashima et al., 2008), (Zhang et al., 2013), (Tkachenko and Lauw, 2014) and (Li et al., 2011). All these related works solve very different problems in comparison analysis than our CPC task.

Works in NLP that use Graph Neural Networks and dependency graph structures include (Huang and Carley, 2019), (Guo et al., 2019). But their tasks and models are different from ours.

4 Experiments

4.1 Dataset

We perform experiments using the benchmark CompSent-19 dataset (Panchenko et al., 2019), where each sentence has an entity pair (e_1, e_2) and its comparative preference label. The original dataset is split into an 80% training set and a 20% test set. During the experiment, we further

Dataset	Better	Worse	None	Total
Train	872(19%)	379(8%)	3,355(73%)	4,606
Dev	219(19%)	95(8%)	839(73%)	1,153
Test	273(19%)	119(8%)	1,048(73%)	1,440
Total	1,346	593	5,242	7,199

Table 2: Statistics of the CompSent-19 dataset

split the original training data by randomly sampling 20% for each label as the development set for model selection. The dataset statistics are given in Table 2. The model is trained only on the newly split training set. We use the class-based F1 score as the evaluation measure. F1(B), F1(W) and F1(N) represent F1 score for classes BETTER, WORSE and NONE respectively. F1-Micro is the average F1 score as in (Panchenko et al., 2019).

4.2 Model Implementation Details

The Stanford Neural Network Dependency Parser (Chen and Manning, 2014) is used to build the dependency parse graph for each sentence. In our experiment, we use two pretrained word embeddings: GloVe embeddings (Pennington et al., 2014)[1] and BERT embedding (Devlin et al., 2019)[2]. The input of BERT is formatted as the standard BERT input format, with "[CLS]" before and "[SEP]" after the sentence tokens. For this, we employ the BERT tokenizer to tokenize each word into word pieces (tokens). The output of the pretrained-BERT model is a sequence of embeddings, each of size 768, and corresponds to a word piece. We average the word piece embeddings of the original word to get the embedding for each word (node in the dependency graph). Note that, word embeddings are kept frozen and not fine-tuned by the subsequent model structure.

For the ED-GAT model, we set the hidden size as 300. The features of the nodes, which are the word embeddings, are first transformed into vectors of the hidden size and then fed into the ED-GAT model. We use 6 attention heads, training batch size of 32, Adam optimizer (Kingma and Ba, 2014) with learning rate 5e-4, word embedding dropout rate (Srivastava et al., 2014) 0.3 and GAT attention dropout rate 0. The implementation of the model is based on PyTorch Geometric (PyG) (Fey and Lenssen, 2019) and NVIDIA GPU GTX 1080 ti.

[1]http://nlp.stanford.edu/data/glove.840B.300d.zip

[2]For all our BERT related experiments, we use the pre-trained BERT model: `https://storage.googleapis.com/bert_models/2018_10_18/uncased_L-12_H-768_A-12.zip`

4.3 Compared Models

We compare models from the previous literature with several variations of our proposed model.

Majority-Class assigns the majority label in the training set to each instance in the test set.

SentEmbed given in (Panchenko et al., 2019) obtains sentence embeddings from a pretrained Sentence Encoder (Conneau et al., 2017; Bowman et al., 2015). The sentence embedding[3] is then fed to XGBoost (Chen and Guestrin, 2016) for classification. For a fair comparison, we also feed the sentence embedding into a linear layer. They are represented as $\textbf{SentEmbed}_{\textbf{XGBoost}}$ and $\textbf{SentEmbed}_{\textbf{Linear}}$.

SVM-Tree[4] given in (Tkachenko and Lauw, 2015) uses convolution kernel methods and dependency tree features to approach the CSI task. We use the one-vs-rest technique to adapt this model to our three-class CPC task.

WordEmbed-Avg first constructs a sentence embedding by averaging the word embeddings of all words in a sentence, and then feeds it to a linear classifier. **Glove-Avg** and **BERT-Avg**, respectively are the methods that use GloVe embeddings from GloVe.840B (Pennington et al., 2014) and static BERT embeddings (Devlin et al., 2019).

BERT-FT appends a linear classification layer on the hidden state corresponding to the first token "[CLS]" of the BERT sequence output and then fine-tunes the pretrained BERT weights on our task.

ED-GAT is the proposed model in this paper (Section 2.2). We use both GloVe embeddings and BERT embeddings. We use (L) to represent model variants with different numbers of layers and use the subscript to denote the type of embedding. For example, $\text{ED-GAT}_{\text{GloVe}}(8)$ is the ED-GAT model using GloVe embedding, and the depth of the model is 8 layers. We also add the $\textbf{LSTM}_{\textbf{BERT}}$ baseline, which uses the sequence output of a static BERT model to train an LSTM model. The final hidden vector is used for classification.

4.4 Results and Analysis

As we see in Table 3, the state-of-the-art (SOTA) baseline is $\text{SentEmbed}_{\text{XGBoost}}$. $\text{SentEmbed}_{\text{Linear}}$ performs much worse than $\text{SentEmbed}_{\text{XGBoost}}$. This result shows that XGBoost classifies sentence embeddings much better than a linear layer. Simply using word embedding average, GloVe-Avg

[3]https://github.com/facebookresearch/InferSent

[4]https://github.com/sitfoxfly/tree-svm

	Models	Micro.	F1(B)	F1(W)	F1(N)
Baselines	Majority-Class	68.95	0.0	0.0	81.62
	SVM-Tree	68.12	53.35	13.90	78.13
	SentEmbed$_{\text{Linear}}$	79.31	62.71	37.61	88.42
	SentEmbed$_{\text{XGBoost}}$	85.00*	75.00*	43.00*	92.00*
	Glove-Avg	76.32	48.28	20.12	86.34
	BERT-Avg	77.64	53.94	26.88	87.47
	LSTM$_{\text{BERT}}$	80.97	63.55	44.02	88.95
	BERT-FT	83.12	69.62	50.37	89.84
Proposed Models	ED-GAT$_{\text{GloVe}}$(8)	83.96	72.58	47.35	90.79
	ED-GAT$_{\text{GloVe}}$(9)	83.89	72.05	46.45	90.54
	ED-GAT$_{\text{GloVe}}$(10)	84.24	72.56	50.20	91.19
	ED-GAT$_{\text{BERT}}$(8)	**87.43**	**78.21**	56.14	**92.98**
	ED-GAT$_{\text{BERT}}$(9)	86.46	74.40	**58.72**	92.31
	ED-GAT$_{\text{BERT}}$(10)	86.18	77.35	53.33	92.23

Table 3: Comparison of baselines and ED-GAT variants. * indicates the result is from the original paper.

and BERT-Avg do not perform well. The result of LSTM$_{\text{BERT}}$ shows that using BERT embedding sequentially is not suitable for our task. BERT-FT fine-tunes BERT on our task, but its performance is below SOTA. During experiments, we also found that the performance of BERT-FT is unstable. The training process of the model quickly overfits the pretrained BERT weights.

For the ED-GAT model, we first tried to train embeddings only on this dataset by randomly initializing word embeddings as input. As expected, the results were significantly poorer than those using the pre-trained embeddings, in part because our training data is very small (see Table 2). As the baselines all use pretrained embeddings, we thus report the results of using pre-trained word embeddings in Table 3. When employing Glove embeddings, surprisingly, ED-GAT$_{\text{GloVe}}$(10) performs better than BERT-FT, which is based on a language model pretrained on a huge corpus. We also tried to employ word2vec[5] for ED-GAT. It got very similar results to those using the GloVe embeddings. The Micro-F1 scores of using word2vec embeddings for the number of layers 8, 9, and 10 are 83.12, 83.33, and 84.86, respectively. To be concise, we did not include these results in Table 3.

Our model also uses the static BERT embedding, which further improves the result. Using static BERT embedding avoids overfitting. On the one hand, it incorporates the rich semantic information with the BERT pretrained weights. On the other hand, ED-GAT's ability to leverage dependency graph features greatly helps the model in capturing

[5]GoogleNews-vectors-negative300.bin.gz (https://code.google.com/archive/p/word2vec/)

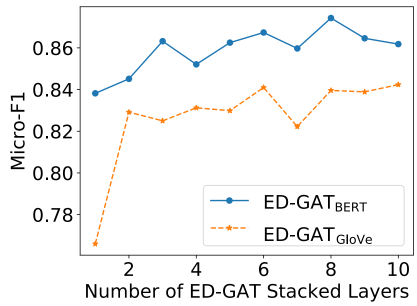

Figure 3: Effects of the number of layers in ED-GAT

the comparison between the entities and classifying the preference direction. Our ED-GAT$_{\text{BERT}}$(8) reports the new state-of-the-art results for CPC task considering F1-Micro and all class-wise F1.

Effects of Model Depth. From Figure 3, we see that increasing the number of stacked layers improves the performance of the model. For ED-GAT$_{\text{GloVe}}$, as GloVe does not contain the context information, the GAT structure based on the dependency graph greatly improves the result. Even the 2-layer model achieves a good result. ED-GAT$_{\text{BERT}}$ does not have the same effect because the BERT embedding already contains rich semantic information. But still, when the number of layers increases, ED-GAT$_{\text{BERT}}$ becomes more powerful as it captures longer range dependencies.

5 Conclusion

This paper proposes a novel model called ED-GAT for Comparative Preference Classification. It naturally leverages dependency graph features and word embeddings to capture the comparison and to classify the preference direction between two given entities. Experimental results show that it outperforms all strong baselines and even BERT pretrained using a huge corpus.

Our future work aims to improve the CPC performance further. Apart from that, we also plan to design novel models to perform the related tasks of entity extraction and aspect extraction from comparative sentences. Performing all these tasks jointly in a multitask learning framework is a promising direction as well because it can exploit the shared features and the inherent relationships of these tasks to perform all tasks better.

Acknowledgments

This work was supported in part by two grants from National Science Foundation: IIS-1910424 and IIS-1838770, and a research gift from Northrop Grumman.

References

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*, pages 632–642.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *SIGKDD*, pages 785–794.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*, pages 670–680.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186.

Ronen Feldman, Moshe Fresco, Jacob Goldenberg, Oded Netzer, and Lyle Ungar. 2007. Extracting product comparisons from discussion boards. In *ICDM*, pages 469–474.

Matthias Fey and Jan E. Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Murthy Ganapathibhotla and Bing Liu. 2008. Mining opinions in comparative sentences. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 241–248.

Zhijiang Guo, Yan Zhang, and Wei Lu. 2019. Attention guided graph convolutional networks for relation extraction. In *ACL*, pages 241–251.

Feng Hou and Guo-hui Li. 2008. Mining chinese comparative sentences by semantic role labeling. In *Proceedings of International Conference on Machine Learning and Cybernetics*, pages 2563–2568.

Binxuan Huang and Kathleen Carley. 2019. Syntax-aware aspect level sentiment classification with graph attention networks. In *EMNLP-IJCNLP*, pages 5468–5476.

Xiaojiang Huang, Xiaojun Wan, Jianwu Yang, and Jianguo Xiao. 2008. Learning to identify comparative sentences in chinese text. In *Proceedings of Pacific Rim International Conference on Artificial Intelligence*, pages 187–198.

Nitin Jindal and Bing Liu. 2006a. Identifying comparative sentences in text documents. In *SIGIR*, pages 244–251.

Nitin Jindal and Bing Liu. 2006b. Mining comparative sentences and relations. In *AAAI*, pages 1331–1336.

Wiltrud Kessler and Jonas Kuhn. 2013. Detection of product comparisons-how far does an out-of-the-box semantic role labeling system take you? In *EMNLP*, pages 1892–1897.

Wiltrud Kessler and Jonas Kuhn. 2014. Detecting comparative sentiment expressions - a case study in annotation design decisions. In *Proceedings of the 12th edition of the KONVENS conference*, pages 165–170.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Takeshi Kurashima, Katsuji Bessho, Hiroyuki Toda, Toshio Uchiyama, and Ryoji Kataoka. 2008. Ranking entities using comparative relations. In *Proceedings of International Conference on Database and Expert Systems Applications*, pages 124–133.

Si Li, Z-J Zha, Zhaoyan Ming, Meng Wang, T-S Chua, Jun Guo, and Weiran Xu. 2011. Product comparison using comparative relations. In *SIGIR*, pages 1151–1152.

Bing Liu. 2012. *Sentiment analysis and opinion mining*. Morgan & Claypool Publishers.

Alexander Panchenko, Alexander Bondarenko, Mirco Franzek, Matthias Hagen, and C Biemann. 2019. Categorizing comparative sentences. In *Workshop on Argument Mining@ACL*, pages 136–145.

Dae Hoon Park and Catherine Blake. 2012. Identifying comparative claim sentences in full-text scientific articles. In *Workshop on detecting structure in scholarly discourse*, pages 1–9.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Maksim Tkachenko and Hady Lauw. 2015. A convolution kernel approach to identifying comparisons in text. In *ACL-IJCNLP*, pages 376–386.

Maksim Tkachenko and Hady W Lauw. 2014. Generative modeling of entity comparisons in text. In *CIKM*, pages 859–868.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

Seon Yang and Youngjoong Ko. 2009. Extracting comparative sentences from korean text documents using comparative lexical patterns and machine learning techniques. In *ACL-IJCNLP*, pages 153–156.

Runxiang Zhang and Yaohong Jin. 2012. Identification and transformation of comparative sentences in patent chinese-english machine translation. In *Proceedings of International Conference on Asian Language Processing*, pages 217–220.

Zhu Zhang, Chenhui Guo, and Paulo Goes. 2013. Product comparison networks for competitive analysis of online word-of-mouth. *ACM Transactions on Management Information Systems (TMIS)*, pages 20:1–20:22.