SaSAKE: Syntax and Semantics Aware Keyphrase Extraction from Research Papers

T Y S S Santosh[1], **Debarshi Kumar Sanyal**[2],
Plaban Kumar Bhowmick[1], **Partha Pratim Das**[1]
[1]IIT Kharagpur, Kharagpur – 721302, India
[2]Indian Association for the Cultivation of Science, Kolkata – 700032, India
santoshtyss@gmail.com, debarshisanyal@gmail.com,
plaban@cet.iitkgp.ac.in, ppd@cse.iitkgp.ac.in

Abstract

Keyphrases in a research paper succinctly capture the primary content of the paper and also assist in indexing the paper at a concept level. Given the huge rate at which scientific papers are published today, it is important to have effective ways of automatically extracting keyphrases from a research paper. In this paper, we present a novel method, Syntax and Semantics Aware Keyphrase Extraction (SaSAKE), to extract keyphrases from research papers. It uses a transformer architecture, stacking up sentence encoders to incorporate sequential information, and graph encoders to incorporate syntactic and semantic dependency graph information. Incorporation of these dependency graphs helps to alleviate long-range dependency problems and identify the boundaries of multi-word keyphrases effectively. Experimental results on three benchmark datasets show that our proposed method SaSAKE achieves state-of-the-art performance in keyphrase extraction from scientific papers.

1 Introduction

Keyphrases are words or phrases that capture important concepts of a document. The task of keyphrase extraction, i.e., automatically extracting a collection of keyphrases from a document, has attracted considerable attention from the research community due to its pivotal importance in various applications like text document retrieval (Jones and Staveley, 1999; Sanyal et al., 2019), document categorization (Hulth and Megyesi, 2006; Hammouda et al., 2005), opinion mining (Berend, 2011) and summarization (Qazvinian et al., 2010; Zhang et al., 2004). Keyphrase extraction from research papers is especially important due to their ability to concisely capture the main tenets of the complex scholarly documents. Although it is common for authors to specify keyphrases in research papers, they are not present in all publications. In these cases, it is beneficial to automatically infer the most relevant keyphrases.

Traditional methods for keyphrase extraction follow a two-step procedure where important phrases from the document are extracted as potential keyphrase candidates by heuristic rules (Medelyan et al., 2009; Witten et al., 2005; Le et al., 2016), and then the extracted candidate phrases are ranked either by unsupervised approaches (Bougouin et al., 2013; Erkan and Radev, 2004; Le et al., 2016; Mihalcea and Tarau, 2004) or supervised approaches (Medelyan et al., 2009; Witten et al., 2005). They typically label each candidate phrase independently without taking into account the dependencies that could potentially exist between neighbouring labels, and they also ignore the semantic meaning of the text. To overcome the above stated limitation, recently (Gollapalli et al., 2017) formulated keyphrase extraction as a *sequence labeling task* and used linear-chain Conditional Random Fields for this task. However, this approach does not explicitly take into account the long-term dependencies and semantics of the text. More recently, to capture both the semantics of the text as well as the dependencies among the labels of neighboring words, (Alzaidy et al., 2019) used a deep learning-based approach called BiLSTM-CRF that combines a bi-directional Long Short-Term Memory (BiLSTM) layer, which models the sequential input text, with a Conditional Random Field (CRF) layer, which captures the dependencies in the output.

Proceedings of the 28th International Conference on Computational Linguistics, pages 5372–5383
Barcelona, Spain (Online), December 8-13, 2020

While the above models are promising, they still suffer from some limitations which we address in this work. For example, consider the sentence in Fig.1. The previous models (Santosh et al., 2020; Alzaidy et al., 2019; Gollapalli et al., 2017) fail to recognize "recurrent Elman neural network" as a keyphrase. This can be attributed to two reasons. Firstly, due to the long-range dependency between the words "we", "extracted" and "network", the model fails to understand the importance of the phrase with respect to the sentence. Secondly, due to their inability to detect the boundaries of phrase accurately, they fail to predict multi-word keyphrases correctly.

We notice long-range dependencies such as those illustrated in the first limitation are shortened in *semantic dependency graphs* in which nodes represent words and edges represent semantic relations between the words, thereby representing predicate-argument relations between content words in a sentence. In Fig.1, lower pink edges show the semantic dependency graph using DM (DELPH-IN Minimal Recursion Semantics) representation schema (Ivanova et al., 2012) shortening the distance between those long-range dependencies. These properties of a semantic dependency graph allow neural network models to capture long-range semantic dependencies effortlessly (Ivanova et al., 2012). To address the second limitation, we posit that *syntactic dependency graphs* in which nodes represent words and edges represent syntactic relations between the words provide the ability to find the boundaries of keyphrases accurately. In Fig.1, the upper red arrows indicate the syntactic dependency relations among words, providing the ability to recognize the boundaries of the keyphrase "recurrent Elman neural network" with the help of the syntactic labels that act as important cues to identify such multi-word keyphrases in the sentence. Though historically, feature based approaches (Nguyen and Kan, 2007; Kim and Kan, 2009) did rely on syntactic and semantic dependency information to extract various features, the latest generation of keyphrase extraction models put syntax and semantics aside in favour of neural sequence models which outperformed syntactically-driven feature engineering methods. We believe that one of the reasons for this choice is the lack of simple and effective methods to incorporate syntactic and semantic dependency information into sequential neural networks.

In this paper, we propose *SaSAKE*, an acronym for *S*yntax *a*nd *S*emantic *A*ware *K*eyphrase *E*xtraction from research papers based on transformer architecture (Vaswani et al., 2017) which stacks up *sentence encoders* to incorporate sequential information and *graph encoders* to incorporate syntactic and semantic dependency graph information. Firstly, the sentence encoder reads the sentence producing context-level representations for each word through multi-head self-attention mechanism that is then fed as input to both the syntactic and the semantic graph encoders along with their corresponding dependency graphs. Then the semantic (syntactic) graph encoder captures the semantic (syntactic) relations among words and enhances the context-level representations incorporating semantic (syntactic) dependency information to produce semantics (syntax)-aware representations. We adopt a multi-head graph attention mechanism in graph encoders to aggregate the information of the relation triples (head, type, tail) into the corresponding head and tail nodes to construct a semantics/syntax-aware representation. However, the propagation of semantic and syntactic relations from distant nodes may introduce noise into the representations. So, we employ an *aggregation layer* that balances the influence of the local contextual information and the additional information obtained from the syntactic and the semantic relations, thereby producing more balanced final representations for each word that are then passed to a *CRF layer* which models the dependencies among the labels for prediction. Experimental results on three datasets of research papers show that SaSAKE outperforms previous state-of-the-art approaches for keyphrase extraction.

2 Related Work

Keyphrase extraction has been an active area of research since two decades (Frank et al., 1999). Unsupervised approaches score the extracted potential candidate phrases based on graph-based ranking algorithms (Page et al., 1999) wherein each word in the document is mapped to a node in the graph and the connecting edges in the graph represent the association patterns among the words in the document. Then, the scores of the individual words are estimated using various graph centrality measures and a combination of other heuristic rules based on tf-idf scores, word co-occurrence measures, extraction of specific lexical patterns and clustering. Such methods include TextRank (Mihalcea and Tarau, 2004),

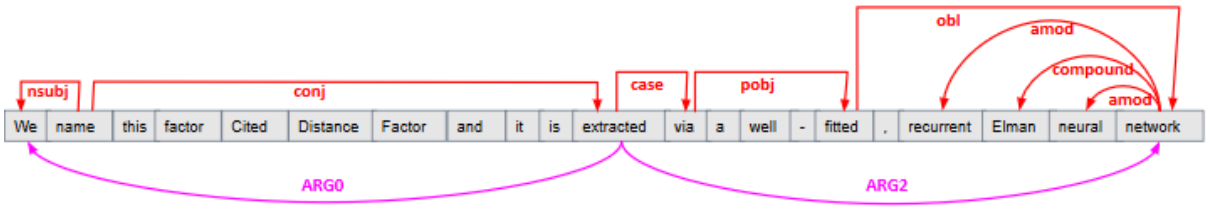


Figure 1: An example sentence from KP20K dataset annotated with syntactic and semantic dependency graphs. The upper red and lower pink dependency edges represent the syntactic and the semantic relations, respectively. Other dependency edges are omitted from display.

LexRank (Erkan and Radev, 2004), TopicRank (Bougouin et al., 2013), SGRank (Danesh et al., 2015) and SingleRank (Wan and Xiao, 2008). On the other hand, supervised approaches use binary classification to label the extracted candidate phrases as keyphrases or non-keyphrases based on textual features such as term frequencies (Hulth and Megyesi, 2006) and syntactic properties (Nguyen and Kan, 2007). Additionally, (Caragea et al., 2014) incorporate external knowledge like document citation context to mine features.

(Gollapalli et al., 2017) was the first to formulate keyphrase extraction as a sequence labeling task and used Conditional Random Fields (CRF) on the extracted features. (Alzaidy et al., 2019) integrated CRF with BiLSTM to capture local contextual information within the sentence. (Sahrawat et al., 2019) expanded the BiLSTM-CRF approach by using contextualized word embeddings. (Zhou et al., 2020) proposed a multi-level memory network with CRFs, which represents the memory network with two different levels (i.e., sentence level and document level) to capture the long-range contextual information. (Santosh et al., 2020) incorporated additional long-range contextual information from the document using document-level attention mechanism into the BiLSTM-CRF approach. There is another line of work that deals with generation of keyphrases that are even absent in the document (Meng et al., 2017; Yuan et al., 2018; Chen et al., 2018; Chen et al., 2019). In the present work, we will focus on keyphrase extraction. To the best of our knowledge, no existing work has attempted integrating semantic and syntactic dependency graphs into neural models for keyphrase extraction. We adopt graph neural networks as the graph encoder to leverage syntactic and semantic dependency graphs.

3 Problem Definition

Given a sentence as the sequence of words $X = [x_1, x_2, \cdots, x_n]$ as input, we parse the input sentence X to obtain the syntactic dependency graph $G_1 = (V_1, E_1)$ and the semantic dependency graph $G_2 = (V_2, E_2)$ using existing tools, where V_1, V_2 denote the sets of nodes and E_1, E_2 denote the sets of edges in the two cases, respectively. Each edge in G_1 represents a triple (head, type, tail) where head $\in V_1$, tail $\in V_1$ and type represents a syntactic relation in $\{nsubj, nmod, dobj, \ldots\}$. Each edge in G_2 represents a triple (head, type, tail) where head $\in V_2$, tail $\in V_2$ and type represents a semantic relation $\{ARG1, ARG2, compound, \ldots\}$ as per DM annotation scheme. Keyphrase extraction task is formulated as a sequence labeling task as follows: Given an input sentence X, its syntactic dependency graph G_1 and its semantic dependency graph G_2, our aim is to output a sequence $y = [y_1, y_2, \ldots, y_n]$ where each y_i is a label from the set {KP,NKP}. Here KP denotes that the word x_i is a keyphrase word and NKP denotes otherwise. Every longest sequence of KP words in a sentence constitutes a keyphrase.

4 Our Approach: SaSAKE

Our proposed approach, SaSAKE consists of the following components: (1) Sentence Encoder (2) Syntactic Graph Encoder (3) Semantic Graph Encoder (4) Aggregation Layer and (5) Label Sequence Prediction Layer. The sentence encoder takes a sequence of words as input to produce a context level representation of the words. Then the syntactic and semantic graph encoders work in parallel taking the context level representation of the words along with the syntactic and the semantic dependency graphs to produce syntax-aware and semantics-aware representation of the words, respectively. The aggregation layer combines the obtained syntax-aware and semantics-aware representations; it also helps to mitigate

the influence of the distant supporting information as we believe that the prediction should be based primarily on the local context. Finally, the above representations are fed into a CRF layer which acts as a decoder to predict the label, KP or NKP, associated with each word.

4.1 Sentence Encoder

We build the sentence encoder using the transformer architecture (Vaswani et al., 2017). It produces context-level representation of words of the input sentence X. It is composed of a stack of N_1 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism and the second is a fully connected feed-forward network. A transformer contains no recurrence or convolution. Therefore, to obtain input representations that use the order of the sequence, we add positional encoding to the learned input embeddings of the first layer $l_i^0 = e_i + p_i$ where p_i, e_i denote the positional encoding and the input embedding, respectively, corresponding to the i^{th} word in the input sequence x_i.

Each multi-head self-attention sub-layer allows the model to jointly attend to information from different representation subspaces at different positions, which is calculated with the help of multiple heads. For each head, we initially project them and then apply scaled dot-product attention to the queries of dimension d_k, keys of dimension d_k and values of dimension d_v by computing the dot products of the query with all keys, then dividing each by $\sqrt{d_k}$ and finally, applying the softmax function to obtain the weights on the values.

$$\text{MultiHeadS}(Q, K, V) = \text{Concat}(head_1, \ldots, head_M)W^O \tag{1}$$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2}$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{3}$$

where W_i^Q, W_i^K, W_i^V and W^O represent trainable matrices for projection and M is a hyperparameter denoting the number of heads. In the fully connected feed-forward network sub-layer, it consists of two linear transformations with a ReLU activation in between.

$$\text{FFN}(x) = \max(0, yW_1 + b_1)W_2 + b_2 \tag{4}$$

where W_1, W_2 represent trainable matrices and b_1, b_2 represent trainable bias vectors. We employ a residual connection (He et al., 2016) around each of the two sub-layers, followed by layer normalization (Lei Ba et al., 2016). That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{Sublayer}(x))$, where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself.

Summarizing, the sentence encoder for the k^{th} layer works as follows:

$$t = \text{LayerNorm}(l^{k-1} + \text{MultiHeadS}(l^{k-1}, l^{k-1}, l^{k-1})) \quad l^k = \text{LayerNorm}(t + FFN(t)) \tag{5}$$

where $k \in [1, N_1]$, and the final contextual-level representations l^{N_1} are fed into the semantic and the syntactic graph encoders in parallel.

4.2 Syntactic & Semantic Graph Encoder

We will describe syntactic graph encoder in brief. The semantic graph encoder is exactly identical to it with the only difference being the consideration of the semantic dependency graph in place of the syntactic dependency graph. A syntactic graph encoder takes context-level representations along with syntax dependency graphs as input to produce syntax-aware representations. It is composed of a stack of N_2 identical layers. Each layer has two sub-layers. The first is a multi-head graph-attention mechanism and the second is a fully connected feed-forward network. We also employ a residual connection around each of the two sub-layers, followed by layer normalization as previously done in sentence encoder. We initialize the input of each node to the first multi-head graph attention layer with contextual representation obtained from sentence encoder.

As in sentence encoder, we calculate multi-head graph attention by jointly attending to information from different representations from different heads. For calculating each head representation, we use

graph attention mechanism proposed by (Veličković et al., 2017) to incorporate the information obtained from syntactic dependency graph which is directed and also contains labels. For each relation triple (head,type,tail), we obtain the representation of the head and the tail by concatenating the node representations of the head h_{head}, the tail h_{tail} and the learnable type representation h_{type}, and then pass it through a linear transformation followed by a nonlinear activation function as follows.

$$g_{head} = \mathrm{ReLU}((h_{head}, h_{type}, h_{tail})W_1 + b_1) \tag{6}$$

$$g_{tail} = \mathrm{ReLU}((h_{head}, h_{type}, h_{tail})W_2 + b_2) \tag{7}$$

where W_1, W_2 are trainable weight matrices and b_1, b_2 are trainable vectors. We obtain the representation of head and tail using above mentioned method for every relation. Then we aggregate the information for each node considering all the representations obtained using the semantic relations in the graph corresponding to that node. Initially we calculate the attentive weight α_{ij} of each representation g_i corresponding to node h_j obtained as follows

$$\alpha_{ij} = \frac{\exp((g_i W_K)^T (h_j W^Q))}{\sum_{r \in N(j)} \exp((g_r W_k)^T (h_j W^Q))} \tag{8}$$

where W^K, W^Q are trainable matrices and N(j) represents nodes in the neighbourhood of node j whose node representation is h_j. Finally we obtain the updated representation h_j^* of node h_j using attention mechanism as follows.

$$h_j^* = \sum_{i \in N(j)} \alpha_{ij} h_i W^V \tag{9}$$

where W^V represents trainable matrix. Similar to sentence encoder, we concatenate several representations using multi-head operation as follows:

$$\mathrm{MultiHeadG}(h_j) = \mathrm{Concat}(h_j^{*1}, h_j^{*2}, \ldots, h_N^{*3}) \tag{10}$$

where N represents the number of heads and h_j^{*p} represents the output of the graph attention mechanism obtained by $head_p$.
Summarizing the graph encoder for k^{th} layer which works as follows:

$$r = \mathrm{LayerNorm}(g^{m-1} + \mathrm{MultiHeadG}(g^{m-1}))) \quad g^m = \mathrm{LayerNorm}(r + \mathrm{FFN}(r)) \tag{11}$$

where $m \in [1, N_2]$. Thus we obtain the final syntax-aware representation g^{N_2}. Similarly we obtain the final semantics-aware representation h^{N_3} through application of a similar graph encoder mechanism which is composed of a stack of N_3 identical layers.

4.3 Aggregation Layer

Through employment of syntactic and semantic graph encoders, syntactic and semantic relation information are propagated along the neighbourhood nodes. Such propagation helps to spread the information mitigating the issue of long-range dependency. But leveraging this additional long distance syntactic and semantic information has a downside of introducing noise into the representations. To alleviate this problem, we use a bi-directional LSTM (Hochreiter and Schmidhuber, 1997) that balances the influence of the local contextual information and the additional information obtained from the syntactic and the semantic relations. We concatenate the syntax-aware and the semantics-aware representations of each word and feed it as input to the bi-directional LSTM incorporating the information for the context words on both the directions where the forward LSTM reads the sequence from x_1 to x_n and the backward LSTM reads the sequence from x_n to x_1, producing final representations $b_1, b_2, \ldots, b_n$ corresponding to the n words in the sentence X.

4.4 Label Sequence Prediction Layer

To model the dependency among the output labels, we use Conditional Random Field (CRF) that treats the output labels as random variables forming a Markov Random Field conditioned upon the input $b_1, b_2, \ldots, b_n$ obtained from the aggregation layer.

5 Experiments

5.1 Datasets

We use three publicly available datasets of research papers namely KP20k (Meng et al., 2017), KDD and WWW (Gollapalli et al., 2017) to evaluate our model. KP20k dataset contains the metadata for 567,830 papers with distinct splits of train, validation, and test sets referred to as KP527K, KP20K-V and KP20K respectively. In all our experiments, we use KP527K for training, KP20K-V for tuning hyperparameters and KP20K, KDD and WWW for testing the model. Table 1 presents the detailed statistics of the datasets.

Table 1: Statistics of the dataset

Statistic	KP527K	KP20k-V	KP20K	KDD	WWW
Number of documents	527,830	20,000	20,000	755	1,330
Number of sentences	4,686,986	176,930	177,278	7,768	12,288
Number of keyphrases	2,806,381	106,181	105,523	3,093	6,405
Number of tokens in keyphrases	5,458,743	205,586	207,073	12,181	6,119

5.2 Implementation Details

We parse the input sentences with state-of-the-art semantic dependency parser (Wang et al., 2018) which employs a neural transition-based parser, using a variant of list-based arc-eager transition algorithm. We obtain the syntactic dependency graph using (Dozat and Manning, 2016) which uses BiLSTM-based approach with biaffine classifiers to predict arcs and labels. In the training stage, we choose the top 50,000 frequent words to form the predefined vocabulary and set the embedding dimension to 768. We also set the dimensions for hidden states to 768. We set the number of heads for multi-head self-attention in sentence encoder, multi-head graph attention in syntactic graph encoder and multi-head graph attention in semantic graph encoder to 4, 8, 8, respectively. We set the number of layers of sentence encoder N_1, syntactic graph encoder N_2 and semantic graph encoder N_3 to 3, 4 and 4, respectively based on hyperparameter tuning experiments which are described in later sections. We adopt the Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of 0.0001 and weight decay $\epsilon = 10^{-4}$. We also employ a dropout rate of 0.5 to prevent overfitting.

5.3 Baselines and Evaluation Metrics

We compare SaSAKE with the following baselines: DAKE (Santosh et al., 2020), MLM-CRF (Zhou et al., 2020), Bi-LSTM-CRF (Alzaidy et al., 2019), CRF (Gollapalli et al., 2017), copy-RNN (Meng et al., 2017), KEA (Witten et al., 2005), Tf-Idf, TextRank (Mihalcea and Tarau, 2004) and SingleRank (Wan and Xiao, 2008). Tf-Idf, TextRank and SingleRank are unsupervised extractive approaches while KEA, Bi-LSTM-CRF, CRF, DAKE, MLM-CRF and SaSAKE follow supervised extractive approach. Copy-RNN is a generative model based on sequence-to-sequence learning along with a copying mechanism. Following previous works, we evaluate the predictions of each model against the author-input gold standard keyphrases and report percentages of Precision, Recall and F1-score. For the unsupervised models and the sequence-to-sequence learning model, we report the performance at top-5 predicted keyphrases since top-5 showed highest performance in the previous works for these models.

5.4 Performance Evaluation

Table 2 shows the results of SaSAKE in comparison to various baselines. From Table 2 , we observe that supervised methods perform better than unsupervised methods. Among supervised methods, we observe that deep learning-based approaches perform better than the traditional feature-based approaches. This indicates the importance of understanding the semantics of the text for keyphrase extraction. BiLSTM-CRF yields better results in terms of the F1-score over CRF indicating that the combination of BiLSTM, which is effective in capturing the semantics of the textual content, and CRF, which captures the dependencies among the output labels, helped boost the performance in identifying keyphrases. DAKE

Table 2: Performance of different keyphrase extraction algorithms.

	KP20K			KDD			WWW		
Method	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Tf-Idf	8.97	13.49	10.77	8.90	10.00	9.40	8.30	10.20	9.20
TextRank	15.29	23.01	18.37	5.80	7.10	6.20	5.10	6.50	5.60
SingleRank	8.42	12.70	10.14	8.80	10.90	9.50	7.70	10.30	8.60
KEA	15.14	22.78	18.19	13.57	15.25	13.86	11.39	14.50	12.42
copyRNN	27.71	**41.79**	33.29	11.47	14.72	12.89	8.59	11.8	9.94
CRF	66.67	10.04	17.46	64.89	22.11	32.98	55.76	18.69	27.99
BiLSTM-CRF	64.19	24.66	35.63	64.33	28.43	39.43	57.83	31.85	41.08
MLM-CRF	-	-	-	37.87	27.71	32.00	32.51	24.17	27.73
DAKE	68.21	30.66	42.30	68.42	31.21	42.86	60.15	33.68	43.18
SaSAKE	**70.48**	36.23	**47.85**	**71.08**	**38.58**	**49.74**	**65.18**	**38.22**	**48.18**
SaSAKE–SynSem	65.24	26.38	37.56	65.72	30.27	41.44	58.37	32.08	41.40
SaSAKE–Sem	68.11	32.76	44.24	69.26	35.18	46.64	63.18	36.38	46.17
SaSAKE–Syn	67.86	32.18	43.65	69.42	34.95	46.49	61.95	36.27	45.75
SaSAKE–AL	69.17	35.19	46.64	70.17	37.22	48.64	64.76	37.67	47.63
SaSAKE–CRF	68.66	34.52	45.94	69.88	36.46	47.91	63.83	36.52	46.45

performs better than BiLSTM-CRF approaches demonstrating that additional supporting contextual information obtained from the document helps in identifying keyphrases by effectively capturing the semantics. We observe that our model, SaSAKE outperforms all the baselines. This can be attributed to several distinctive features of SaSAKE: (i) the sentence encoder designed using a transformer architecture incorporates contextual representations more effectively; (ii) the syntactic graph encoder helps the model to identify the boundaries of multi-word phrases effectively; and (iii) the semantic graph encoder helps the model to capture long-range dependencies effectively. We will study the effect of these factors in more detail in the following sections.

5.5 Ablation Study

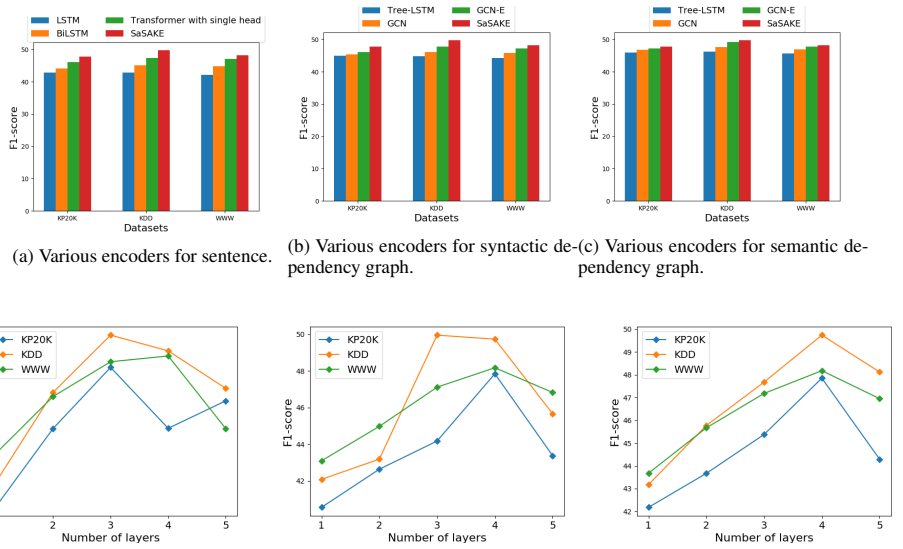
To understand the effectiveness of the architectural components of SaSAKE, we derive five variants of our model to carry out an ablation study. We derive (i) SaSAKE–SynSem by removing the graph encoders at both the syntactic and the semantic levels, (ii) SaSAKE–Syn by removing only the syntactic graph encoder, (iii) SaSAKE–Sem by removing only the semantic graph encoder, (iv) SaSAKE–AL by removing aggregation layer (i.e., simply concatenating the representations obtained from syntactic and semantic graph encoder) (v) SaSAKE-CRF by removing CRF layer and using softmax for prediction. Table 2 presents the results of our variants. From Table 2, we observe that SaSAKE–SynSem performs better than BiLSTM-CRF approach demonstrating the superiority of the transformer architecture used in sentence encoder compared to BiLSTM to capture context-level representations. SaSAKE–Syn and SaSAKE–Sem perform better than SaSAKE–SynSem showing that the incorporation of semantic and syntactic dependency information helps to improve the performance by capturing long-range dependencies effectively. Incorporation of syntactic and semantic graph encoder did show improvement but it is pushed even further when we add the aggregation layer (SaSAKE–AL), which helps to mitigate the noisy influence of learnt long-range dependencies and helps to refine according to the local context of each word. When CRF is removed from SaSAKE, we observe that its performance falls showing that the CRF layer successfully captures the label dependencies.

5.6 Effectiveness of Sentence Encoder

Variants of Sentence Encoder: In this section, we replace our sentence encoder, which is a transformer with multi-head self-attention, with other encoders, namely, LSTM , Bi-directional LSTM (Hochreiter and Schmidhuber, 1997) and transformer with single head self-attenion to study the effect of our encoder in capturing context-level information. Figure 2a presents the F1-score of these variants on the three datasets. From Fig. 2a, we observe that BiLSTM performs better than LSTM; it captures the context information from both the directions unlike only from a single direction in an LSTM. We also observe that the transformer architecture performs better than the LSTM ones showing the superiority of transformers in capturing the context effectively. Finally, we observe that the multi-head self-attention

mechanism performs better than the single-head one, allowing the model to jointly attend to information from different representation sub-spaces at different positions using multiple heads.

Effects on Number of Layers: In this section, we will study the influence of number of layers in the sentence encoder on the performance. We report the F1-score on three datasets varying the number of layers from 1 to 5 in Fig. 2d. From Fig. 2d, we observe that the performance rises till layer 3 in case of KDD and KP20K dataset and then begins to fall as the number of layers increases. In case of WWW, it increases till layer 4 and then takes a downward path but the marginal improvement (with respect to layer) is less; so we have fixed the number of layers in the sentence encoder to three across all datasets. The performance does not continue to increase due to the incremental complexity and the decreasing generalization capability of our model with the growing number of layers.



(a) Various encoders for sentence.

(b) Various encoders for syntactic dependency graph.

(c) Various encoders for semantic dependency graph.

(d) Number of layers in Sentence Encoder.

(e) Number of layers in Syntactic Graph Encoder.

(f) Number of layers in Semantic Graph Encoder.

Figure 2: Analysis of various components in our model, SaSAKE.

5.7 Effectiveness of Syntactic Graph Encoder

Variants of Syntax Encoder: We study the effectiveness of our syntax graph encoder designed using graph attention mechanism by comparing it with other syntax encoders like Tree-LSTM (Chen et al., 2016), GCN (Kipf and Welling, 2016) without incorporation of edge label and direction, and GCN-E (Marcheggiani and Titov, 2017) which incorporates edge direction information as incoming or outgoing but no edge labels. From Fig. 2b, we observe that GCN performs better than Tree-LSTM because GCN considers surrounding nodes whereas Tree-LSTM captures dependencies across unbounded paths in a tree which may turn out to be noisy for our task of keyphrase extraction. We observe that GCN-E performs better than GCN demonstrating the importance of edge direction information. Our model which incorporates the direction as well as label information of the edges using graph attention mechanism identifies the boundaries of multi-word phrases more effectively.

Effects of Number of Layers: We conduct experiments to study the effect of the number of layers of the syntactic graph encoder on the performance. Fig. 2e shows the F1-score with 1 to 5 layers on the three datasets. From Fig. 2e, we observe that the performance of our model on KP20K and WWW dataset first improves with the increase in the number of encoder layers upto 4 layers and then drops as the number of layers further increases. On the other hand, we have noticed for the KDD dataset, the curve took a downward trend from layer 3 but the downfall is not much pronounced in layer 4. So, we have set the number of layers in the syntactic graph encoder as four for uniformity across datasets. Since the graph encoder passes information into the local neighborhood of any node, successive operations on the dependency tree allows it to pass information to the farthest node, and the problem of overfitting takes effect when the layer count rises beyond a threshold, explaining the curve after layer 4 in the figure.

The improvements in latency (and area) obtained by Cartesian genetic programming are validated using a professional FPGA design tool.	
BiLSTM-CRF	genetic programming
DAKE	genetic programming
SaSAKE	Cartesian genetic programming
We present a notion of eta-long beta-normal term for the typed lambda calculus with sums and prove, using Grothendieck logical relations, that every term is equivalent to one in normal form	
BiLSTM-CRF	lambda calculus
DAKE	typed lambda calculus
SaSAKE	typed lambda calculus, Grothendieck logical relations

Table 3: Examples of the extracted keyphrases by our approach and other models. Phrases highlighted in green boxes are gold-standard keyphrases.

5.8 Effectiveness of Semantic Graph Encoder

Variants of Semantic Encoder: We study the effectiveness of our semantic graph encoder by comparing it with other semantic encoders, namely Tree-LSTM (Chen et al., 2016), GCN (Kipf and Welling, 2016) and GCN-E (Marcheggiani and Titov, 2017) as described in Sec. 5.7. We observe that our proposed method which adopts a multi-head graph attention mechanism in a graph encoder to aggregate the information of the relation triples (head, type, tail) into the corresponding head and tail nodes to construct a semantics-aware representation captures long-range dependencies better than the previous approaches.

Effects of Number of Layers: To study the influence of the number of layers in the semantic graph encoder on the performance, we report the F1-score with 1 to 5 layers on the three datasets. As shown in Fig. 2f, the performance of our model first improves with the increase in the number of layers upto 4 for all the datasets and then drops as the number of layers further increases. This is due to the overfitting problem described above for the case of the syntactic graph encoder.

5.9 Case Study

We perform a case study to better understand the model performance. In Table 3, we show two example sentences with gold-standard keyphrases highlighted with green boxes, along with the keyphrases extracted by our model, SaSAKE, and the baselines BiLSTM-CRF and DAKE. In the first example, the word 'Cartesian' associated with 'genetic programming' is not identified as part of keyphrase by BiLSTM-CRF and DAKE but SaSAKE could identify it because of the syntactic dependency the word 'Cartesian' possesses with 'genetic programming'. In second example, BiLSTM-CRF could not identify the complete phrase 'typed lambda calculus' whereas DAKE could identify it due to the information it obtained from the other sentences in the document. Our model SaSAKE could identify it due to the syntactic dependency 'typed' possesses with 'lambda calculus'. DAKE and BiLSRM-CRF failed to extract the phrase 'Grothendieck logical relations' . On the other hand, our model could extract it due to the semantic dependency it possesses with 'prove' and 'present'. More interestingly, although 'Grothendieck' is an out-of-vocabulary term, our model was able to identify it as part of a keyphrase exploiting the syntactic connection this word has with 'logical relations'. So SaSAKE even helps to alleviate out-of-vocabulary problem to some extent using dependency relations provided explicitly.

6 Conclusion

In this paper, we proposed SaSAKE - a deep neural architecture to extract keyphrases from a research paper based on transformer architecture. It employs a sentence encoder to incorporate sequential information, and graph encoders to incorporate syntactic as well as semantic dependency graph information that helps to capture long-range dependencies and identify the boundaries of keyphrases effectively. It outperforms several competing models on three standard benchmark datasets. In future, we plan to use contextualized representations of words like BERT and its variants and also, compare our model with other abstractive keyphrase generation algorithms. We also intend to incorporate additional supporting information which can be obtained from citation contexts and citation graphs.

Acknowledgement

This work is supported by *National Digital Library of India* Project sponsored by Ministry of Human Resource Development, Government of India at IIT Kharagpur.

References

Rabah Alzaidy, Cornelia Caragea, and C Lee Giles. 2019. Bi-LSTM-CRF sequence labeling for keyphrase extraction from scholarly documents. In *Proceedings of the World Wide Web Conference*, pages 2551–2557. ACM.

Gábor Berend. 2011. Opinion expression mining by exploiting keyphrase extraction. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*. Asian Federation of Natural Language Processing.

Adrien Bougouin, Florian Boudin, and Béatrice Daille. 2013. TopicRank: Graph-based topic ranking for keyphrase extraction. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*, pages 543–551.

Cornelia Caragea, Florin Adrian Bulgarov, Andreea Godea, and Sujatha Das Gollapalli. 2014. Citation-enhanced keyphrase extraction from research papers: a supervised approach. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1435–1446.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, and Hui Jiang. 2016. Enhancing and combining sequential and tree LSTM for natural language inference. *arXiv preprint arXiv:1609.06038*.

Jun Chen, Xiaoming Zhang, Yu Wu, Zhao Yan, and Zhoujun Li. 2018. Keyphrase generation with correlation constraints. *arXiv preprint arXiv:1808.07185*.

Wang Chen, Yifan Gao, Jiani Zhang, Irwin King, and Michael R Lyu. 2019. -guided encoding for keyphrase generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6268–6275.

Soheil Danesh, Tamara Sumner, and James H Martin. 2015. Sgrank: Combining statistical and graphical methods to improve the state of the art in unsupervised keyphrase extraction. In *Proceedings of the 4th Joint Conference on Lexical and Computational Semantics*, pages 117–126.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

Günes Erkan and Dragomir R Radev. 2004. LexRank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479.

Eibe Frank, Gordon W Paynter, Ian H Witten, Carl Gutwin, and Craig G Nevill-Manning. 1999. Domain-specific keyphrase extraction.

Sujatha Das Gollapalli, Xiao-Li Li, and Peng Yang. 2017. Incorporating expert knowledge into keyphrase extraction. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*.

Khaled M Hammouda, Diego N Matute, and Mohamed S Kamel. 2005. Corephrase: keyphrase extraction for document clustering. In *Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 265–274. Springer.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

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

Anette Hulth and Beáta B Megyesi. 2006. A study on automatically extracted keywords in text categorization. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 537–544. Association for Computational Linguistics.

Angelina Ivanova, Stephan Oepen, Lilja Øvrelid, and Dan Flickinger. 2012. Who did what to whom? a contrastive study of syntacto-semantic dependencies. In *Proceedings of the 6th Linguistic Annotation Workshop*, pages 2–11.

Steve Jones and Mark S Staveley. 1999. Phrasier: a system for interactive document retrieval using keyphrases. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–167. ACM.

Su Nam Kim and Min-Yen Kan. 2009. Re-examining automatic keyphrase extraction approaches in scientific articles. In *Proceedings of the Workshop on Multiword Expressions: Identification, Interpretation, Disambiguation and Applications (MWE 2009)*, pages 9–16.

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

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Tho Thi Ngoc Le, Minh Le Nguyen, and Akira Shimazu. 2016. Unsupervised keyphrase extraction: Introducing new kinds of words to keyphrases. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, pages 665–671. Springer.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*.

Olena Medelyan, Eibe Frank, and Ian H Witten. 2009. Human-competitive tagging using automatic keyphrase extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3*, pages 1318–1327. Association for Computational Linguistics.

Rui Meng, Sanqiang Zhao, Shuguang Han, Daqing He, Peter Brusilovsky, and Yu Chi. 2017. Deep keyphrase generation. *arXiv preprint arXiv:1704.06879*.

Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into text. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 404–411.

Thuy Dung Nguyen and Min-Yen Kan. 2007. Keyphrase extraction in scientific publications. In *Proceedings of the International Conference on Asian Digital Libraries*, pages 317–326. Springer.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Vahed Qazvinian, Dragomir R Radev, and Arzucan Ozgur. 2010. Citation summarization through keyphrase extraction. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 895–903.

Dhruva Sahrawat, Debanjan Mahata, Mayank Kulkarni, Haimin Zhang, Rakesh Gosangi, Amanda Stent, Agniv Sharma, Yaman Kumar, Rajiv Ratn Shah, and Roger Zimmermann. 2019. Keyphrase extraction from scholarly articles as sequence labeling using contextualized embeddings. *arXiv preprint arXiv:1910.08840*.

Tokala Yaswanth Sri Sai Santosh, Debarshi Kumar Sanyal, Plaban Kumar Bhowmick, and Partha Pratim Das. 2020. Dake: Document-level attention for keyphrase extraction. In *Proceedings of the European Conference on Information Retrieval*, pages 392–401. Springer.

Debarshi Kumar Sanyal, Plaban Kumar Bhowmick, Partha Pratim Das, Samiran Chattopadhyay, and T. Y. S. S. Santosh. 2019. Enhancing access to scholarly publications with surrogate resources. *Scientometrics*, 121(2):1129–1164.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Xiaojun Wan and Jianguo Xiao. 2008. Single document keyphrase extraction using neighborhood knowledge. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 2*, pages 855–860.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A neural transition-based approach for semantic dependency graph parsing. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Ian H Witten, Gordon W Paynter, Eibe Frank, Carl Gutwin, and Craig G Nevill-Manning. 2005. KEA: practical automated keyphrase extraction. In *Design and Usability of Digital Libraries: Case Studies in the Asia Pacific*. IGI Global.

Xingdi Yuan, Tong Wang, Rui Meng, Khushboo Thaker, Peter Brusilovsky, Daqing He, and Adam Trischler. 2018. One size does not fit all: Generating and evaluating variable number of keyphrases. *arXiv preprint arXiv:1810.05241*.

Yongzheng Zhang, Nur Zincir-Heywood, and Evangelos Milios. 2004. World wide web site summarization. *Web Intelligence and Agent Systems: An International Journal*, 2(1):39–53.

Tao Zhou, Yuxiang Zhang, and Haoxiang Zhu. 2020. Multi-level memory network with crfs for keyphrase extraction. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 726–738. Springer.