Large-Scale Hierarchical Text Classification with Recursively Regularized Deep Graph-CNN

Hao Peng[1,4], Jianxin Li[1,4], Yu He[1,4], Yaopeng Liu[1,4], Mengjiao Bao[1,4], Lihong Wang[3], Yangqiu Song[2], and Qiang Yang[2]
[1]Beijing Advanced Innovation Center for Big Data and Brain Computing, Beihang University;
[2] Department of Computer Science and Engineering, HKUST;
[3]National Computer Network Emergency Response Technical Team/Coordination Center of China;
[4]State Key Laboratory of Software Development Environment, Beihang University
[1,3,4]Beijing, China; [2]Clear Water Bay, Hong Kong;
penghao,lijx,heyu,liuyp,baomj@act.buaa.edu.cn;wlh@isc.org.cn;yqsong,qyang@cse.ust.hk

ABSTRACT

Text classification to a hierarchical taxonomy of topics is a common and practical problem. Traditional approaches simply use bag-of-words and have achieved good results. However, when there are a lot of labels with different topical granularities, bag-of-words representation may not be enough. Deep learning models have been proven to be effective to automatically learn different levels of representations for image data. It is interesting to study what is the best way to represent texts. In this paper, we propose a graph-CNN based deep learning model to first convert texts to graph-of-words, and then use graph convolution operations to convolve the word graph. Graph-of-words representation of texts has the advantage of capturing non-consecutive and long-distance semantics. CNN models have the advantage of learning different level of semantics. To further leverage the hierarchy of labels, we regularize the deep architecture with the dependency among labels. Our results on both RCV1 and NYTimes datasets show that we can significantly improve large-scale hierarchical text classification over traditional hierarchical text classification and existing deep models.

CCS CONCEPTS

• **Information retrieval Retrieval tasks and goals**; *Design Methodology*; *Clustering and classification*; *Natural Language Processing*; • **Machine Learning Supervised learning**; • **Machine learning approaches** *Deep Convolutional Neural Networks*;

KEYWORDS

Hierarchical Text Classification; Recursive Regularization; Graph-of-words; Deep Learning; Deep Convolutional Neural Networks;

1 INTRODUCTION

Topical text classification is a fundamental text mining problem for many applications, such as news classification [18], question answering [28], search result organization [11], online advertising [2], etc. When there are many labels, hierarchical categorization of texts has been recognized as a natural and effective way to organize texts and it has been well studied in the past two decades [5, 6, 13, 30, 39, 43, 44]. Most of the above traditional approaches represent text as sparse lexical features such as bag-of-words (BOW) and/or n-grams due to simplicity and effectiveness [1]. Different kinds of feature engineering, such as none-phrases or key-phrases, were shown no significant improvement on themselves, while the majority voting over the results of different features are significantly better [36].

Recently, deep learning has been proven to be effective to perform end-to-end learning of hierarchical feature representations, and has made groundbreaking progress on object recondition in computer vision and speech recognition problems [24]. Two popular deep learning architectures have attracted more attention for text data, i.e., recurrent neural networks (RNNs) [3, 17][1] and convolutional neural networks (CNNs) [8, 25]. RNNs are more powerful on short messages or word level syntactics or semantics [3]. When they are applied to long documents, hierarchical RNNs can be developed [40]. However, hierarchical RNNs assume that the documents and sentences are considered as natural boundaries for the definition of the hierarchy where only regular texts and formal languages satisfy this constraint. Different from RNNs, CNNs use convolutional masks to sequentially convolve over the data. For texts, a simple mechanism is to recursively convolve the nearby lower-level vectors in the sequence to compose higher-level vectors [8]. This way of using CNNs simply evaluates the semantic compositionality of consecutive words, which corresponds to the n-grams used in traditional text modeling [1]. Similar to images, such convolution can naturally represent different levels of semantics shown by the text data. Higher level represents semantics captured by larger "n"-grams.

For document-level topical classification of texts, the sequential information of words might not be as important as it is for language models [3] or sentiment analysis [45]. For example, when we write "I love this restaurant! I think it is good. It has great sandwich. But the service may not be very efficient sine there are always a lot of people...", we can easily identify it's topic as "food" but sentiment analysis should be conducted more carefully since there is a word

[1]Here we ignore the discussion of recursive neural networks [38] since it requires knowing the tree structure of text, which is not as efficient as the others when dealing with large scale data.

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04..
https://doi.org/10.1145/3178876.3186005

"but." For topic classification, the key words, phrases, and their composition are more important. In this case, rather than sequential information, the *non-consecutive phrases* and *long-distance word dependency* are more important for computing the composition of semantics. For example, in a document, the words "restaurant" and "sandwich" may not co-occur in a small window. However, "menu" may co-occur with both of them somewhere else in the document, and the composition of all of three words is a very strong signal to classify the document to be "food" related topics. Therefore, a more appropriate way of modeling non-consecutive and long-distance semantics is expected for text topical classification.

In this paper, we propose a Hierarchically Regularized Deep Graph-CNN (HR-DGCNN) framework to tackle the above problems with the following considerations.

Input. Instead of viewing long documents as sequences, we first convert them to graphs. A natural way to construct the graph is based on word co-occurrence, i.e., if two words co-occur in a small window of texts, we build an edge between them. Then given a constructed graph, any sub-graphs can be regarded as long distance n-grams [34]. For each node of the graph, we use a pre-trained vector based on word2vec [32] as input features. In this way, our input can be regarded as a graph of vectors. Although word2vec optimization has been proven to be identical to co-occurrence matrix factorization under mild conditions [26], it is still preferable to explicitly represent documents as graphs, since for upper level convolution, the longer distance co-occurrence of words (which corresponds convolution over sub-graphs) can be explicitly computed and evaluated.

Convolution Layers. For lower intermediate layers, we follow the graph normalization approach [33] to make the following convolution operators possible. This graph normalization can be regarded as a local operator to convert a graph to a sorted sequence, where the order is based on the importance of the node on the graph. Other graph convolution approaches are discussed in the related work (Section 2). For the upper intermediate layers, we generally follow the well defined AlexNet [22] and VGG [37] networks for ImageNet classification. Different from image data, which at most has three channels, i.e., RGB values, word embeddings have much more channels. A typical word embedding can have 50 dimensions. In this way, the input tensor for convolution is slightly different from images, and thus, we coordinately modify the configuration of all the following convolution layers to make the feature representation more effective.

Output. For large scale hierarchical text classification, there have been many existing studies to design better output cost functions [12, 46]. Here, we use the cross entropy objective function to determine labels and adopt the simple but effective recursive regularization framework proposed in [13]. The idea is if the two labels are parent and child in the hierarchy, we assume that the classification from these two labels to other labels are similar. In the global view of the hierarchy, it means the children label classifiers should inherit the parent classifier. To handle large-scale labels, we also use a tree cut algorithm to automatically divide the trees into parts, and conquer the regularized models for different parts.

In the experiments, we compare our proposed approach with state-of-the-art methods, including traditional algorithms and deep learning approaches. We use two benchmark datasets to demonstrate both effectiveness and efficiency. RCV1 [27] dataset contains 23,149 training news articles and 784,446 testing news articles with 103 classes. NYTimes[2] contains 1,855,658 news articles in 2,318 categories. The results showed that our approach is very promising to work on large scale hierarchical text topical classification problems.

The contributions of this paper can be highlighted as follows. First, we introduce a Deep Graph-CNN approach to text classification. There have been proof that bag-of-graph representation [34] and CNN representation [8] are effective for text topic classification. However, this is the first attempt to show Graph-CNN is even more effective. Second, for large scale hierarchical text classification, we demonstrate that recursive regularization can also be applied to deep learning. This can be a general framework for deep learning applied to classifications problems when classifying data into a hierarchy of labels. Third, we use two benchmark datasets to demonstrate the efficiency and effectiveness of our algorithm. They are with either large test set, large label set, or large training set.

The rest of the paper is organized as follows. We first review the related work in Section 2. Then we introduce the detailed input and architecture of our algorithm in Sections 3 and 4. Then we show the experiments in Section 5. Finally, we conclude this paper in Section 6. Our system is publicly available at https://github.com/HKUST-KnowComp/DeepGraphCNNforTexts.

2 RELATED WORK

In this section, we briefly review the related work in following two categories.

2.1 Traditional Text Classification

Tradition text classification uses feature engineering (e.g., extracting features beyond BOW) and feature selection to obtain good features for text classification [1]. Dimensionality reduction can also be used to reduce the feature space. For example, Latent Dirichlet Allocation [4] has been used to extract "topics" from corpus, and then represent documents in the topic space. It can be better than BOW when the feature numbers are small. However, when the size of words in vocabulary increases, it does not show advantage over BOW on text classification task [4]. There is also existing work on converting texts to graphs [34, 42]. Similar to us, they used co-occurrence to construct graphs from texts, and then they either applied similarity measure on graph to define new document similarities [42] or applied graph mining algorithms to find frequent sub-graphs in the corpus to define new features for text [34]. Both of them showed some positive results for small label space classification problems, and the cost of graph mining is more than our approach which simply performs breadth-first search.

For hierarchical classification with large label space, many efforts have been put on how to leverage the hierarchy of labels to reduce to time complexity or improve the classification results. For example, top-down classification mechanism has been shown to be more efficient than bottom-up classification (or flat classification which treats each label in the leaf as a category) when there are many labels [30, 44]. Moreover, the parent-child dependency of labels in the hierarchy can also be used to improve the model. For

[2] https://catalog.ldc.upenn.edu/ldc2008t19

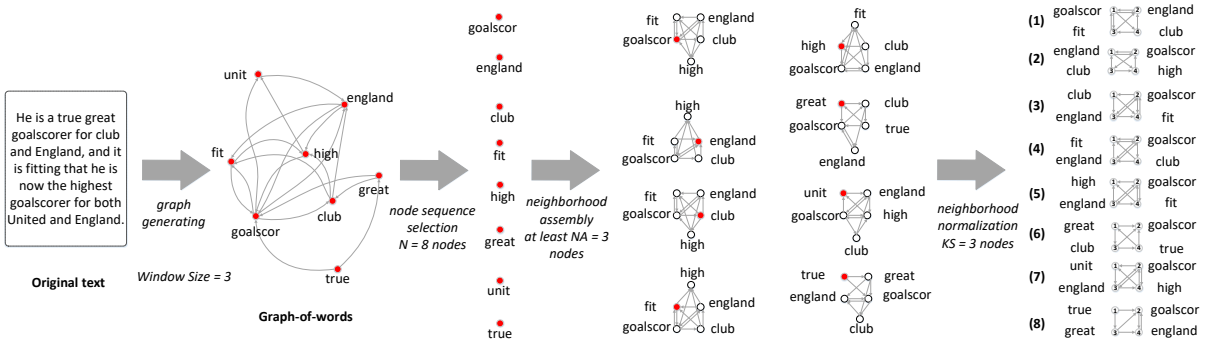

Figure 1: Illustration of document to graph. Starting from original raw text, we first build the graph of words based on word co-occurrence with sliding window of size three words. Then we select a sequence of nodes from the graph based on the rank of each node. For each node in the sequence, we find a sub-graph containing four nodes and normalize the neighborhoods to make each sub-graph consistent for further Graph-CNN operations.

example, some hierarchical cost-sensitive loss can be developed [5]. The idea of transfer learning can also be borrowed to improve each of the classifiers in the hierarchy [43]. Recently, a simpler recursive regularization of weight vectors of linear classifiers in the hierarchical model has been developed, and shown to be the state-of-the-arts in large-scale hierarchical text classification problems [13, 14].

2.2 Deep Learning for Text Classification

As the introduction mentioned, there have been RNN and CNN models applied to texts. Most of them are language models, which are inspired by [3]. The advantage of working with language models is that it can be trained in an unsupervised manner, and then the features extracted for words can be used for many downstream NLP tasks [8]. For text classification, both RNNs and CNNs have been applied. For example, hierarchal RNN has been proposed for long document classification [40] and later attention model is also introduced to emphasize important sentences and words [45]. CNNs have also been proposed to perform text classification. For example, Kalchbrenner et al. [19] used a dynamic CNN for sentence classification, and showed significant improvements over traditional tasks such as sentiment and question type (raw texts) classification. Then Kim [20] applied a much simpler CNN to sentences classification and got competitive results. Zhang et al. [48] and Conneau et al. [9] used a character level CNN with very deep architecture to compete with traditional BOW or n-gram models. The combination of CNNs and RNNs are also developed which shows improvements over topical and sentiment classification problems [23]. All of the above text classification models still dealt with small label space, i.e., at most 20 labels. Two recent papers mentioned that they can handle large scale label space. [29] used similar way to convert multi-label classification problem to be a set of multiple binary classification problems and they still used simple CNN model as [20] did. [47] adopted a general setting for large-scale label set classification but the text data they used have been preprocessed, which means the characteristics of texts is not considered.

Graph-CNN has been developed recently inspired by the success of CNN on object detection tasks [22, 37] since one can imagine image lattice as a very regular graph. However, extension to arbitrary graphs is not trivial, since the convolution operator cannot be applied to any size of sub-graphs. Thus, two general ways of Graph-CNN have been considered. One way is to consider a global graph which consists of all nodes for all data samples and develop a global convolution operator for the graph [10, 16, 21]. However, this way requires a huge input if the nodes of the graphs can be represented as high dimensional vectors. The other way is to use a local operator [33] to convolve each sub-graph. This is more similar to the idea of CNNs used for images. The local operator requires the sub-graphs to be of fixed size (or smaller than a size constraint) and the nodes in arbitrary sub-graphs to be ordered, which could be non-optimal. We introduce this local graph convolution approach to texts, where the graph is constructed based on word co-occurrence.

3 DOCUMENTS AS GRAPHS

In this section, we introduce how we process text documents into graphs, and represent each document as a graph of vectors. Formally, we denote the set of labels as $\mathcal{L} = \{l_i | i = 1, 2 \ldots, K\}$, where K is the number of labels. Since we focus on hierarchical classification, the labels have parent-children relationship. Thus, we also denote label $l_i^{(j)}$ $(j = 1, \ldots, K_i)$ as the children of l_i and K_i is the number of children of l_i. We denote the training set as $\mathcal{D} = \{d_s, T_s\}_{s=1}^{M}$, where M is the number of instances in $\mathcal{D}$, d_s is a document, T_s is the label set of d_s and $T_s \subseteq \mathcal{L}$.

3.1 Word Co-occurrence Graph

To convert a document to a graph, a natural way is to use the word co-occurrence. As Figure 1 shows, for a simple sentence, we convert it as a graph using word co-occurrence. We split each document to be sentences and further tokens using Stanford CoreNLP tool[3]. We also obtain the stems of each token using Stanford CoreNLP. To remove noise, we first remove stop words such as "the," "a," etc., provided by the RCV1-v2 data [27]. Then we use a fixed-size sliding window to count the word co-occurrence. For example, for the sentence in Figure 1, for the word "fitting," we first perform stemming to get "fit," and check a window of two previous words and build an edge from fit to each of the words in the window.

[3] http://stanfordnlp.github.io/CoreNLP/

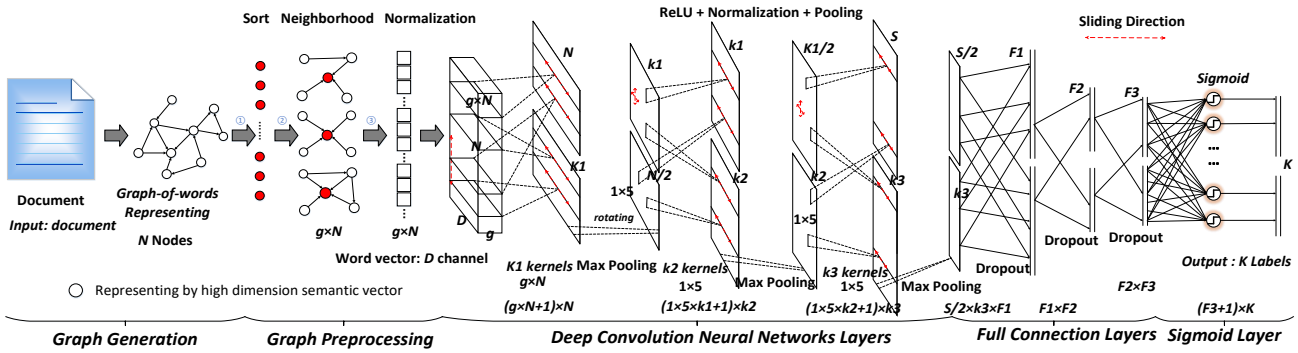

Figure 2: A typical configuration of Deep Graph-CNN for text classification.

3.2 Sub-graph of Words

When applying CNN to images, a convolution mask with fixed size (e.g., 11×11 pixels used in AlexNet [22]) is applied to local patches on the image to extract low-level features, e.g., edges. The combinations of convolved features are further convolved to obtain higher-level feature representations, e.g., parts and objects. Similar to images, we want to apply convolution masks to the sub-graphs on word co-occurrence graph (corresponding to local image patches). The combination of distant n-grams mined from word co-occurrence graph can compose low-level semantics, e.g., sub-topics. By further convolving with the output of convolution, we can obtain higher-level semantics, e.g., super-topics. To enable the convolution of sub-graphs, we need to select and normalize all the potential sub-graphs so that we can have a consistent way of convolution. Here we follow the general approach proposed in [33]. Given a graph of words, we use the following two steps to obtain the normalized the sub-graphs for further processing.

Sub-graph Generation. We sort all nodes in a graph based on their degrees (number of neighborhoods of a node). If the degrees are the same, we sort the nodes by their occurrences in the document. If the occurrences are further the same, we use their connections with neighbors to sort, i.e., the number of co-occurrence with neighborhoods. In this case, we can sort all the nodes in the graph and select N most important nodes that we think may affect the topic classification. After obtaining the nodes, we use a simple breadth-first-search algorithm to expand a sub-graph for each selected node. We set a number of least sub-graph size, i.e., g. If at the current depth the sub-graph is smaller than size g, we expand one more tier to see whether the size is satisfied. After this step, we will obtain N sub-graphs, each of which contains at least g nodes (if exists). For example, in the middle of Figure 1 we generated eight sub-graphs.

Sub-graph Normalization. Given a sub-graph, we want to have an order of nodes for a convolution mask to convolve. Thus, a labeling of nodes is expected to make the convolution consistent over all sub-graphs and across documents. An optimal labeling is defined as follows. Suppose graphs G and G' with g nodes are in a collection of graphs $\mathcal{G}$. Given the labeling s of a graph G, we can construct an adjacency matrix $\mathbf{A}^s(G)$. Then an optimal labeling is defined as

$$s^* = \arg\min_s E[D_{\mathbf{A}}(\mathbf{A}^s(G), \mathbf{A}^s(G')) - D_G(G, G')], \quad (1)$$

where $D_{\mathbf{A}}(\cdot,\cdot)$ is a distance measure of two matrices, such as $||\mathbf{A} - \mathbf{A}'||_{L1}$, and $D_G(\cdot,\cdot)$ is a distance measure of two graphs, such as edit distance on graphs. However, such labeling is NP-hard and thus we follow [33] to have an alternative labeling. Starting from the root, which is the node that triggered the sub-graph in previous step, we first follow breadth-first-search to use the depth to rank the nodes. Then in the same tier (depth) of the graph based spanning tree, we use the degree to rank the nodes. Then if two nodes in the same tier have the same degree, we further use other factors to break the tier, such as the edges used in previous step. Then after this step, we have g nodes for each sub-graph. For the sub-graphs with more than g nodes in the previous step, we simply use the rank filter out them. For the sub-graphs with less than g nodes, we add some dummy nodes disconnected to any nodes in the graph. We can easily see that, in this way, the normalization applied to a 2-D lattice such as an image, will be exactly the same as the way CNN's first layer is applied to images. The complexity of this sub-graph normalization is given by [33]. Practically, the complexity can be at most $O(Ng^2)$ [33] where N is the selected node number and g is the size of sub-graphs. The example of graph normalization is also shown in the last column of Figure 1.

3.3 Graphs of Embeddings

To incorporate as much semantic information as possible, instead of representing each node in the graph (a word in a document) as an ID, we follow other CNN and RNN algorithms reviewed in Section 2 to use word embeddings as input. We use word2vec [31, 32] trained by the larger corpus, i.e., Wikipedia, with CBOW model, where the window size is set to be five and the dimension of word embeddings is set to be 50 for all the experiments in this paper. Other parameters for word2vec are set to be default values. In this way, we have a graph of embeddings as input. Then the convolution introduced in the next section will be operated over sub-graphs of embeddings.

4 HIERARCHICALLY REGULARIZED DEEP GRAPH-CNN

We design the architecture with several convolutional layers. The number of convolutional layers can be adjusted and based on the size of datasets and total labels. Here we illustrate the architecture with a typical configuration, which is shown in Figure 2.

4.1 Convolutional Layers

The first convolutional layer takes the feature space of size $N \times g \times D$ as input, where N number of selected and normalized sub-graphs, g is the size of receptive field (size of the sub-graphs), and D is the dimension of word embeddings. For example, in Figure 2, we have $g = 5$ and $D = 50$. We use a $g \times D$ kernel to convolve the $N \times g \times D$ input tensor. This kernel serves as a composition of semantics of each input sub-graph to have a higher level semantic meaning. Then for all the N input sub-graphs, we use $k1$ kernels to convolve the same way to generate a $N \times k1$ matrix. After that, we use a max pooling layer to generate a $N/2 \times k1$ matrix, which means we select half of the sub-graphs which can better represent the topics for further processing. Then we use a 5×1 kernel to convolve the $N/2$ sub-graphs to obtain higher level of semantics (combination of sub-graphs). Here we use $k2$ such 5×1 kernels to generate $k2$ dimensions. Thus, we have a $k1 \times k2$ matrix. Till now, we have successfully convolved over different dimensions of word embeddings, different words in each sub-graph, and different sub-graphs in the documents. Then we can further convolve to have composition of higher-level semantics. For example, in Figure 2, we use a max polling layer after $k1 \times k2$ matrix to generate a $k1/2 \times k2$ matrix, followed by $k3$ 1×5 kernel with sliding step 3 to generate a $S \times k3$ matrix. This can be regarded as a composition of the original $k1$ different $g \times D$ kernels applied to $N \times g \times D$ input tensor. Then we apply a max pooling layer to generate a $S/2 \times k3$ matrix as the input to the fully connected layers. These two steps ($S \times k3$ and $S/2 \times k3$ matrices) are empirically set. Throughout the convolutional layers, we use ReLU as activation function to speed up the process of training and avoid over-fitting.

4.2 Fully Connected and Output Layers

To perform classification, we add three fully connected layers. These layers are mostly used to deal with nonlinearity of classification. As shown in Figure 2, we have $F1 = 2048$, $F2 = 1024$, and $F3 = 1024$ (or 512) for the final three layers. Here we apply dropout to avoid over-fitting where the dropout rate is set to be 0.5. Empirically dropout roughly doubled the number of iteration of convergence, but it improves the robustness of the network and boosts the prediction accuracy. The final layer is further connected to a set of K Sigmoid functions, which correspond to the K labels in the hierarchy. Given a set of M labeled documents and $m = 1, \ldots, M$, our model optimizes the cross-entropy between the true label distribution and the predicted distribution:

$$H = -\sum_{m=1}^{M}\sum_{k=1}^{K} l_k(d_m)\log P_k(d_m) + (1 - l_k(d_m))\log(1 - P_k(d_m)), \tag{2}$$

where $l_k(d_m)$ is binary label to indicate whether document d_m belongs to label k, and $P_k(d_m)$ refers to the probability of neural network prediction of label k.

4.3 Recursive Regularization

If we simply treat each label as an independent decision, we can use Eq. (2) to train the neural network. However, as we mentioned, most of the practical problems have a hierarchy of labels. In the hierarchy, the parent label is a super-topic of the children labels. In this case, introducing dependency among labels can significantly improve the classification, because when the leaf nodes has less training examples, the decision can be regularized by its parent. Thus, similar to [13], we use a recursive regularization over the final fully connected layer. As a simplification, the hierarchical dependencies between labels encourage the parameters of labels to be similar, if two labels are close in the hierarchy. For example, in Figure 3, there is edge between 'Computing' and "Artificial intelligence," so the parameters of the two labels could be similar to each other. Formally, we denote w_i as parameters in the final fully connected layer for label l_i in the hierarchy, and we denote $\mathcal{W} = \{w_{l_i} : l_i \in \mathcal{L}\}$. $l_i^{(j)}$ refers to a children of l_i. Then we use the following regularization term to regularize the parameters of the final fully connected layer:

$$\lambda(\mathcal{W}) = \sum_{l_i \in \mathcal{L}} \sum_{l_i^{(j)}} \frac{1}{2} ||w_{l_i} - w_{l_i^{(j)}}||^2. \tag{3}$$

We denote our algorithm Hierarchically Regularized Deep Graph-CNN (HR-DGCNN) using the following loss function to optimize the parameters:

$$J = H + C\lambda(\mathcal{W}), \tag{4}$$

where C is the penalty parameter.

4.4 Recursive Hierarchical Segmentation

To handle large-scale label hierarchies, we use a recursive hierarchical segmentation to divide and conquer the original problem. In this way, we can perform training over each sub-problem separately and significantly reduce the learning complexity for each sub-problem. An example of recursive hierarchical segmentation is shown in Figure 4. Suppose we want to have a segmentation of sub-trees containing at most five leaves. Then we perform depth-first and preorder traversal from root to leaf nodes with following steps. When traversing node A, the number of children leaf nodes is 5, so the sub-tree (i) can be firstly segmented. Then the sub-tree (i) is merged as one logical label as a leaf. When we backtrack to node B, the children leaf number also is 5. So it's divided into sub-tree (ii) and merged into one logical label as a leaf. According to the law of depth-first and preorder traversal, the sub-graph (iii) will also be divided, and merged into one logical label as a leaf. When the number of children leaf nodes is less than the threshold value, it will continue to traverse until no more nodes. So the sub-graph iv is divided and merged into one logical label. The children leaf nodes of F is 4, so we combine the nodes E and F to meet the requirement of having five leaf nodes. Although the hierarchy is divided into sub-graphs for parallel and distributed deep CNN models, the master program learns to classify top level nodes, such as the B, E, and F in Figure 4, and recursively calls other deep CNN models when testing.

5 EXPERIMENTS

In the experiments, we compare our proposed algorithms with state-of-the-art traditional hierarchical text classification as well as recently developed deep learning architectures on two datasets.

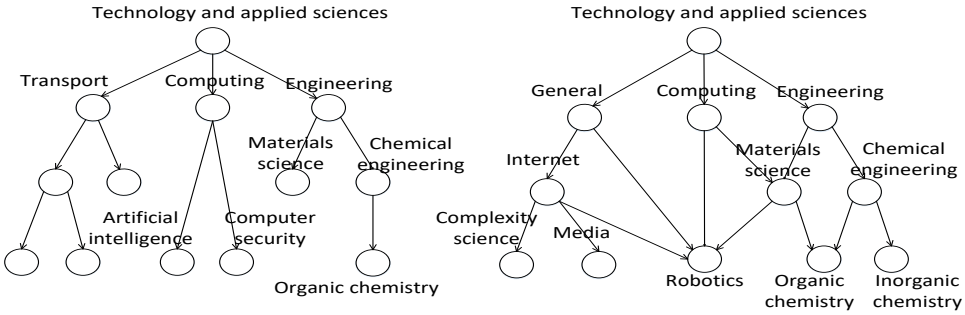

Figure 3: Label hierarchies. The left is a pure hierarchical structure of labels, and the right is graph structure of labels but has no cycles in the graph. Both are commonly used in practice and can be handled by recursive regularization.

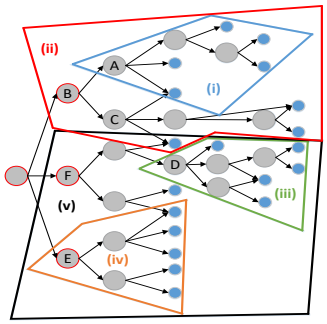

Figure 4: Illustration of Recursive Hierarchical Segmentation. The leaf nodes refer to class-labels, and the internal nodes are super topics. The hierarchy is recursively divided into 5 parts of sub-graphs.

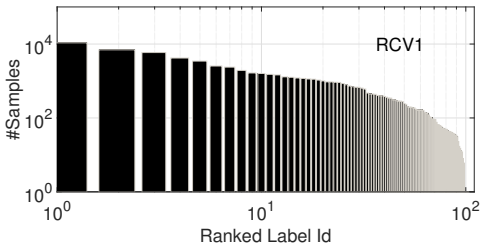

Figure 5: Label statistics of RCV1 training dataset.

Table 1: Dataset Statistics. The training/test split for RCV1 is done by [27]. The training/test split for NYTimes is done by ourselves, which is 90% for training and 10% for test. For both of the data, we randomly sample 10% from the training data as development sets.

Dataset	Training	Testing	Class-Labels	Depth	avg#Tokens
RCV1	23,149	784,446	103/137	6	240
NYTimes	1,670,093	185,566	2,318	10	629

5.1 Datasets and Evaluation Metrics

The two datasets we used are RCV1 and NYTimes datasets. A summarization of statistics of these two datasets is shown in Table 1.

• **RCV1** [27]. RCV1 dataset is a manually labeled newswire collection of Reuters News from 1996-1997. The news documents are categorized with respect to three controlled vocabularies: industries, topics, and regions. We use the topic-based hierarchical classification as it has been most popular in evaluation. There are 103 categories including all classes except for root in the hierarchy. According to [13–15], we also consider the 137 categories by adding some virtual classes.[4] On average, there are 225 documents per label for 103 labels for training. The distribution of RCV1 labels is shown in Figure 5.

• **NYTimes** [35]. This corpus contains nearly every article published in the New York Times between January 01, 1987 and June 19th, 2007. As a large scale corpus, NYTimes was widely used in document routing, document categorization, entity extraction, cross document coreference resolution, and information retrieval, etc. We use the standard of taxonomic classifier labeled in this corpus to test large-scale hierarchical text classification. On average, there are 720 documents per label for 2,318 labels for training. The distribution of NYTimes labels is shown in Figure 6.

We follow [13] to use Micro-F_1 and Macro-F_1 as our evaluation metrics for hierarchical classification.

• **Micro**-F_1 is a F_1 score considering overall precision and recall of all the labels. Let TP_t, FP_t, FN_t denote the true-positives, false-positives, and false-negatives for the t-th label in label set $\mathcal{L}$ respectively. Then micro-averaged F_1 is: $P = \frac{\sum_{t\in\mathcal{L}} TP_t}{\sum_{t\in\mathcal{L}} TP_t+FP_t}$, $R = \frac{\sum_{t\in\mathcal{L}} TP_t}{\sum_{t\in\mathcal{L}} TP_t+FN_t}$, $Micro-F_1 = \frac{2PR}{P+R}$.

• **Macro**-F_1 is another F_1 which evaluates averaged F_1 of all different class-labels in the hierarchy. $Macro-F_1$ gives equal weight to each label. Formally, macro-averaged F_1 is defined as: $P_t = \frac{TP_t}{TP_t+FP_t}$, $R_t = \frac{TP_t}{TP_t+FN_t}$, $Macro-F_1 = \frac{1}{|\mathcal{L}|}\sum_{t\in\mathcal{L}} \frac{2P_tR_t}{P_t+R_t}$.

5.2 Methods for Comparison

We compare both traditional hierarchical text classification baselines and modern deep learning based classification algorithms.

• Flat baselines: We used both Logistic Regression (**LR**) and Support Vector Machines (**SVM**) to learn from data. We call them

[4]If any internal node in the hierarchy had positive examples, we created a new leaf under it and re-assigned all instances to the leaf node. For all graph based dependencies, if there are two adjacent nodes both of which have training examples, we created an empty dummy node in between them. This is to prevent classes from getting directly regularized towards each other, but regularize towards their mean (the parameters of the dummy node) instead.

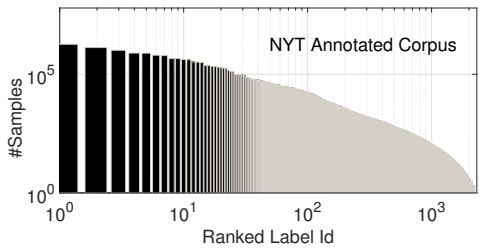

Figure 6: Label statistics of NYTimes training dataset.

Table 2: DGCNN Configurations. The convolutional layers parameters are denoted as "conv<receptive field size>-<number of channels>."

1 Layer	3 Layers	6 Layers
conv5-64	conv5-64	conv5-64
maxpool2	maxpool2	maxpool2
	conv5-128	conv5-128
	maxpool2	maxpool2
	conv5-256	conv5-256
	maxpool2	maxpool2
		conv5-512
		maxpool2
		conv5-512
		maxpool2
		conv5-1024
		maxpool2
FC-1024	FC-2048	FC-2048
FC-1024	FC-1024	FC-1024
FC-512	FC-512	FC-1024

flat baselines since we treat each leaf node as a label and train a multi-class classifier. Then for the internal nodes, we simply add all the ancestors of a leaf to be the labels to the document in the leaf.

• Hierarchical SVMs. We use the implementation of Hierarchical SVM (**HSVM**) [41] released by the authors and implement our version of Top-down Support Vector Machines (**TD-SVM**) [30].

• Hierarchical Regularization. We implemented both Hierarchically Regularized Logistic Regression (**HR-LR**) and Hierarchically Regularized Support Vector Machines (**HR-SVM**) [13, 14] according to the paper's introduction to the algorithms.

• Hierarchical RNN based Models. We compare two RNN models, i.e., Hierarchical Long Short-term Memory Network (**HLSTM**) [7] and Hierarchical Attention Network [45] (**HAN**). Both use RNN models to encode sentence level representation based on words, and then use RNN models to encode document level representation based on sentence representation. HAN further uses a global attention mechanism to attend useful words and sentences. They are originally used for document level sentiment classification. We used the implementation release along [7].

• CNNs based Models. We compare with **XML-CNN** model [29] which considered multi-label text classification using simple **CNN** model originally applied to sentence classification [20] and Recurrent Convolutional Neural Network model (**RCNN**) [23]. We also added experiments to test whether deeper models help. Here, we augment the simple CNN model [20] to be three layers and six layers, which we call deep CNN (**DCNN-3** and **DCNN-6**). The window size for CNN is 3 (the best setting shown in [20]), and for DCNN, all following layers were set to have convolution windows of size 5.

• Deep Graph-CNN Models. We implemented our proposed methods, i.e., Deep Graph Convolutional Neural Networks (**DGCNN**) and Hierarchically Regularized Deep Graph Convolutional Neural Networks (**HR-DGCNN**) with different numbers of convolutional layers, such as 1, 3, 6 layers. To construct the graph of words, we considered word co-occurrence in a window of size 5 for all the experiments. The configurations of different CNN layers are shown in Table 2,

Table 3: Comparison between stemmed and original words on RCV1 dataset.

Types	Model	Classes	Macro-F_1	Micro-F_1
Stemmed	DGCNN-3	103	**0.4322**	**0.7611**
Original	DGCNN-3	103	0.4143	0.7597

Table 4: Comparison among different sub-graph numbers (N) and normalized sub-graph sizes (d) on RCV1 dataset.

Types	Model	Classes	Macro-F_1	Micro-F_1
N=32	DGCNN-3	103	0.3191	0.6971
N=64	DGCNN-3	103	**0.4322**	**0.7611**
N=128	DGCNN-3	103	0.4326	0.7611
k=3	DGCNN-3	103	0.3919	0.7430
k=5	DGCNN-3	103	**0.4322**	**0.7611**
k=7	DGCNN-3	103	0.4318	0.7546

5.3 Experimental Settings

All of our experiments were run on 32 core Intel Xeon E5-2650-v3@2.30GHz, with 256GB RAM cluster and K80 GPUs. The operating system and software platforms are Debian 7.0, TensorFlow r0.12 and Python 3.4. For all the deep learning based models, we used public release of word2vec[5] to train 50[6] dimensional word embeddings over the 100 billion words from Wikipedia corpus based on CBOW model with window size of five (with other default parameters). We also generated vectors for both stemmed and original words for following graph-of-words generation using the same data. The common parameters of training the models were empirically set, such as batch size = 128, MOMENTUM = 0.9, Dropout = 0.5, moving average = 0.0005, and regularization weight decay = 0.00005, etc. Both data were trained on 90% of the training data shown in Table 1, and terminated based on the other 10%. For deep learning baselines, we used the same cost function as our DGCNN model for the final layer. We tried our best to use the best parameters either shown in the paper or the default parameters in the software. However, since most of the models were designed for sentiment classification, we found sometimes they may not work well on the two datasets we compared. Thus, we also put efforts on tuning the

[5]https://github.com/dav/word2vec

[6]Here, we consider balancing the computational complexity of deep learning model and the integrity of the word semantic expression.

Table 5: Comparison of results on RCV1 dataset.

Models	#Classes	Macro-F_1	Micro-F_1	Classes	Macro-F_1	Micro-F_1	Remarks
LR	103	0.3281	0.6921	137	0.5339*	0.8008*	We implemented TD-SVM and HR-LR/SVM. The numbers with *'s are the numbers directly copied from [13], which are mainly used to verify the correctness of our experiments. Note that the numbers of TD-SVM with 137 classes seem very similar to our TD-SVM results with 103 classes and have different behavior compared to others reported by [13].
SVM	103	0.3295	0.6905	137	0.5472*	0.8082*	
HSVM	103	0.3329	0.6932	137	–	–	
TD-SVM	103	0.3368	0.6964	137	0.3415*	0.7134*	
HR-LR	103	0.3217	0.7159	137	0.5581*	0.8123*	
HR-SVM	103	0.3858	0.7275	137	0.5656*	0.8166*	
HLSTM	103	0.3102	0.6726	137	0.3201	0.7019	For all the deep learning based baselines, we slightly modified the output as a set of binary classifications as Eq. (2) shows to handle many labels. DCNN-3 and DCNN-6 are models we developed to increase the depth of original architecture in [20] to see whether it helps.
HAN	103	0.3268	0.6964	137	0.3411	0.7211	
RCNN	103	0.2931	0.6859	137	0.3219	0.6952	
XML-CNN	103	0.3007	0.6955	137	0.3106	0.7149	
DCNN-3	103	0.3987	0.7323	137	0.5843	0.8169	
DCNN-6	103	0.3479	0.7158	137	0.5013	0.8072	
DGCNN-1	103	0.3631	0.7418	137	0.5495	0.8168	HR-based models improve non-HR-based models. Our best configuration (HR-DGCNN-3) improves the best baseline model (DCNN-3) by about 3 points (both F_1 scores) for 103 classes and about 7 points on Macro-F_1 and 1 point on Micro-F_1 for 137 classes.
DGCNN-3	103	0.4322	0.7611	137	0.6182	0.8175	
DGCNN-6	103	0.3905	0.7404	137	0.5637	0.8149	
HR-DGCNN-1	103	0.3682	0.7481	137	0.5649	0.8166	
HR-DGCNN-3	103	**0.4334**	**0.7618**	137	**0.6586**	**0.8255**	
HR-DGCNN-6	103	0.3992	0.7489	137	0.5623	0.8142	

models that did not work well. As a result, all the deep learning baselines are the best results to our best efforts. Before numerous experiments, we also fairly tested the difference between stemmed and original words on RCV1 data classification, under DGCNN with three convolutional layers (DGCNN-3). It is shown in Table 3 that both Micro-F_1 and Macro-F_1 are improved after stemming for better quality of word representations. Thus, we will present all the following classification results based on stemmed words for all models (the word embeddings are also trained based on stemmed words).

5.4 Performance on RCV1

For RCV1 data, with DGCNN and HR-DGCNN models, we set the number of selected nodes in graph-of-words N limited by 64, the size of selected and normalized sub-graphs to be 5, dimensionality of word vectors as 50, the other configurations to follow Table 2, and the numbers of classification neurons are 103/137 respectively, according to the introduction in Section 5.1. We tried different N's and k's shown in Table 4 and it shows that $N = 64$ and $k = 5$ can already show good enough results for RCV1 data.

The comprehensive comparison is shown in Table 5. For traditional text classification algorithms, we can see that SVM is better than LR. HSVM is comparable to TD-SVM, although HSVM uses more complicated structural output cost function. It seems simple top-down mechanism can do well on this dataset. HR-LR and HR-SVM are better than LR and SVM. In our implementation, the HR-SVM improvement is greater than HR-LR. Both show consistent results compared to the numbers shown in [13].

For deep learning based approaches, we can see that RNN based algorithms, HLSTM and HAN, are comparable to SVM and LR on 103 classes. However, they do not perform well on 137 classes. RCNN is even worse on both settings. We would presume that for fine-grained topical classification, recurrent models may not have advantage since it will compress the whole sequence of words as a dense vector for classification. RNNs have advantage for sentiment classification since it can model the dependencies of words like "very," "but," "not," etc. However, when we want to extract non-consecutive and long-distance semantics, it cannot capture such information. Note that in our experiments, HAN is better than HLSTM. This means that attention to particular topical words and boosting their weights can help final classification decision. For baseline CNN models, it is shown that XML-CNN does not perform very well on our tasks. However, deeper CNN models can significantly improve the performance (by 9 points of Macro-F_1 with 103 classes and 27 points with 137 classes).

For our DGCNN models, we tried different configurations shown in Table 2. From the results we can see that, hierarchical regularization indeed helps. For the 137 classes case, the improvement is very significant (by 4 points on Macro-F_1). This again verifies why the authors of [13] used an extended problem of RCV1 rather than the original 103 classes to apply recursive regularization. The DGCNN without hierarchical regularization is also better than XML-CNN and deeper CNN we implemented. The improvements are around 1-7 points for different settings and metrics. This shows that by representing the original text as a graph instead of a sequence, we can gain benefit from non-consecutive and long-distance semantics for topical text classification.

5.5 Performance on NYTimes

For NYTimes dataset, since the label space of NYTimes is large (Table 1), we divided the original problem by constraining the sub-problems to be at least 300 labels but no more than 600 labels. Then we got 9 such sub-problems. We applied this divide and conquer strategy for all the deep learning based models, while keeping SVM based models working on the original problem. For our DGCNN models, we set the number of selected nodes as 192, since from Table 1 we can see that on average there are more tokens in NYTimes than in RCV1 data. The size of sub-graphs of words is still limited to 5. The dimensionality of word embeddings is 50, and all the other configurations follow Table 2. The results shown in Table 7 have consistent conclusion as RCV1 dataset. The difference of NYTimes dataset is that it has much larger training data. Thus, we observe

Table 6: Comparison of training time based on GPU and CPU. (Test evaluations for all the models were performed by CPU.)

Types	Model	Datasets	1-Batch Time (sec.)	Train Time (hr.)	Test Time (hr.)	Macro-F_1	Micro-F_1
CPU	DGCNN-3	RCV1	1.416	12	2.5	0.4320	0.7611
GPU	DGCNN-3	RCV1	0.767	6	2.5	0.4322	0.7611
CPU	DGCNN-6	NYTimes	1.437	168	0.6	0.2985	0.6566
GPU	DGCNN-6	NYTimes	0.401	48	0.6	0.2991	0.6566

Table 7: Comparison of results on NYtimes dataset.

Models	Classes	Macro-F_1	Micro-F_1
SVM	2,318	0.2158	0.5217
HSVM	2,318	0.2187	0.5213
TD-SVM	2,318	0.2249	0.5404
HR-SVM	2,318	0.2571	0.6123
HLSTM	2,318	0.2141	0.5271
HAN	2,318	0.2217	0.5395
RCNN	2,318	0.2019	0.5311
XML-CNN	2,318	0.2001	0.5292
DCNN-3	2,318	0.2471	0.5793
DCNN-6	2,318	0.2669	0.6055
DGCNN-1	2,318	0.2147	0.5195
DGCNN-3	2,318	0.2791	0.6030
DGCNN-6	2,318	0.2991	0.6566
HR-DGCNN-1	2,318	0.2209	0.5293
HR-DGCNN-3	2,318	0.2807	0.6146
HR-DGCNN-6	2,318	**0.2995**	**0.6612**

that the deeper models perform better on this data (DCNN-6 and DGCNN-6 are better than DCNN-3 and DGCNN-3). Moreover, by comparing Figures 5 and 6, we can see that RCV1 dataset is more balanced. This is why the Macro-F_1's of NYTimes data are less than the ones of RCV1 data even when the Macro-F_1 scores are similar.

We also compare the results with and without hierarchical segmentation on NTYimes dataset, which is shown in Table 8. Although the Macro-F_1 and Micro-F_1 are close between recursive hierarchical segmentation and stand-alone DGCNN-6 models. The training time of stand-alone DGCNN-6 takes several days. The DGCNN-6 model can have up to 7.5 times speedup with our recursive hierarchical segmentation process.

Table 8: Comparsion of training time and results on NYTimes dataset. The evaluations for stand-alone (Native) and recursive hierarchical segmentation (RHS) programs were performed by DGCNN-6 and GPU.

Model	Macro-F_1	Micro-F_1	Training Time (days)
Native	0.2993	0.6481	15
RHS	0.2991	0.6566	2

Table 9: Number of parameters (in millions).

Dataset	1 Layer	3 Layers	6 Layers
RCV1	3.73	4.99	9.57
NYTimes	3.92	5.21	12.32

5.6 Time Consumption

We compared our models trained with different devices with different configurations, shown in Table 6. It results in that GPU can speed up the training time by about 2 times for RCV1 and 3.5 times for NYTimes, while the test errors are almost the same. Moreover, test time for RCV1 data is much more than NYTimes, because the test data of RCV1 is larger than NYTimes, according to Table 1. We also tested whether recursive hierarchical segmentation can help improve the efficiency of the model. As Table 8 shown, we can improve NYTimes training time more than seven times while maintaining the classification performance almost the same or even better.

5.7 Number of Parameters

In Table 9, we report the number of parameters for each dataset. All configurations follow the generic design presented in Table 2, while the size of graph-of-words employs the best performance in experiments. Specifically, here we have $N = 64, k = 5$ in RCV1 and $N = 192, k = 5$ in NYTimes. The number of parameters are shown in Table 9. Note that we divide the large-scale of label hierarchies of NYTimes dataset into 9 sub-problems. The scale of parameters of stand-alone DGCNN-6 in NYTimes was up to 105 millions.

6 CONCLUSIONS

In this paper, we present a deep graph CNN model to perform large-scale hierarchical text classification. We first convert bag-of-words to graph of words. Then we leverage the convolution power of semantic composition to generate text document representation for topic classification. The experiments compared to both traditional state-of-the-art text classification models as well as recently developed deep learning models show that our approach can significantly improve the results on two datasets, RCV1 and NYTimes. In the future, we plan to extend our deep graph CNN model to other complex text classification datasets and applications.

7 ACKNOWLEDGMENTS

The corresponding author is Jianxin Li. This work is supported by NSFC program (No.61472022,61772151,61421003) and partly by the Beijing Advanced Innovation Center for Big Data and Brain Computing. Yangqiu Song is supported by China 973 Fundamental R&D Program (No. 2014CB340304) and the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 26206717). Qiang Yang is supported by China 973 Fundamental R&D Program (No. 2014CB340304) and Hong Kong CERG projects 16211214, 16209715 and 16244616. We also thank the anonymous reviewers for their valuable comments and suggestions that help improve the quality of this manuscript.

REFERENCES

[1] Charu C. Aggarwal and ChengXiang Zhai. 2012. A Survey of Text Classification Algorithms. In *Mining Text Data*. 163–222.

[2] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. 2013. Multi-label Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages. In *WWW*. 13–24.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research* 3 (2003), 1137–1155.

[4] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3 (2003), 993–1022.

[5] Lijuan Cai and Thomas Hofmann. 2004. Hierarchical Document Categorization with Support Vector Machines. In *CIKM*. 78–87.

[6] Hao Chen and Susan Dumais. 2000. Bringing Order to the Web: Automatically Categorizing Search Results. In *CHI*. 145–152.

[7] Huimin Chen, Maosong Sun, Cunchao Tu, Yankai Lin, and Zhiyuan Liu. 2016. Neural sentiment classification with user and product attention. In *EMNLP*. 1650–1659.

[8] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* 12 (2011), 2493–2537.

[9] Alexis Conneau, Holger Schwenk, LoÃıc Barrault, and Yann Lecun. 2016. Very Deep Convolutional Networks for Text Classification. (2016).

[10] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*. 3837–3845.

[11] Susan Dumais and Hao Chen. 2000. Hierarchical classification of Web content. In *SIGIR*. ACM, 256–263.

[12] Eva Gibaja and Sebastián Ventura. 2015. A tutorial on multilabel learning. *ACM Computing Surveys (CSUR)* 47, 3 (2015), 52.

[13] Siddharth Gopal and Yiming Yang. 2013. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In *KDD*. 257–265.

[14] Siddharth Gopal and Yiming Yang. 2015. Hierarchical Bayesian inference and recursive regularization for large-scale classification. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 9, 3 (2015), 18.

[15] Siddharth Gopal, Yiming Yang, Bing Bai, and Alexandru Niculescu-Mizil. 2012. Bayesian models for large-scale hierarchical classification. In *NIPS*. 2411–2419.

[16] Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep Convolutional Networks on Graph-Structured Data. *CoRR* abs/1506.05163 (2015). http://arxiv.org/abs/1506.05163

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[18] Thorsten Joachims. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In *ECML*. 137–142.

[19] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *ACL*. 655–665.

[20] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*. 1746–1751.

[21] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

[23] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *AAAI*. 2267–2273.

[24] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep Learning. *Nature* 521 (2015), 436–444.

[25] Yann Lecun, LÃĺon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*. 2278–2324.

[26] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*. 2177–2185.

[27] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. 2004. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research* 5, Apr (2004), 361–397.

[28] Xin Li and Dan Roth. 2002. Learning question classifiers. In *ACL*. 1–7.

[29] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. Deep Learning for Extreme Multi-label Text Classification. In *SIGIR*. 115–124.

[30] Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. 2005. Support vector machines classification with a very large-scale taxonomy. *ACM SIGKDD Explorations Newsletter* 7, 1 (2005), 36–43.

[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *Computer Science* (2013).

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[33] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *ICML*. 2014–2023.

[34] François Rousseau, Emmanouil Kiagias, and Michalis Vazirgiannis. 2015. Text categorization as a graph classification problem. In *ACL*, Vol. 15. 107.

[35] Evan Sandhaus. 2008. The New York Times Annotated Corpus LDC2008T19. In *Linguistic Data Consortium*.

[36] Sam Scott and Stan Matwin. 1999. Feature Engineering for Text Classification. In *ICML*. 379–388.

[37] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

[38] Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *NIPS*. 801–809.

[39] Aixin Sun and Ee-Peng Lim. 2001. Hierarchical Text Classification and Evaluation. In *ICDM*. 521–528.

[40] Duyu Tang, Bing Qin, and Ting Liu. 2015. Document Modeling with Gated Recurrent Neural Network for Sentiment Classification. In *EMNLP*. 1422–1432.

[41] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research* 6, Sep (2005), 1453–1484.

[42] Wei Wang, Diep Bich Do, and Xuemin Lin. 2005. Term graph model for text classification. In *International Conference on Advanced Data Mining and Applications*. Springer, 19–30.

[43] Lin Xiao, Dengyong Zhou, and Mingrui Wu. 2011. Hierarchical classification via orthogonal transfer. In *ICML*. 801–808.

[44] Gui-Rong Xue, Dikan Xing, Qiang Yang, and Yong Yu. 2008. Deep classification in large-scale text hierarchies. In *SIGIR*. 619–626.

[45] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alexander J. Smola, and Eduard H. Hovy. 2016. Hierarchical Attention Networks for Document Classification. In *NAACL–HLT*. 1480–1489.

[46] Min-Ling Zhang and Zhi-Hua Zhou. 2014. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering* 26, 8 (2014), 1819–1837.

[47] Wenjie Zhang, Liwei Wang, Junchi Yan, Xiangfeng Wang, and Hongyuan Zha. 2017. Deep Extreme Multi-label Learning. *CoRR* abs/1704.03718 (2017).

[48] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NIPS*. 649–657.