Deep Learning for Extreme Multi-label Text Classification

Jingzhou Liu
Carnegie Mellon University
liujingzhou@cs.cmu.edu

Wei-Cheng Chang
Carnegie Mellon University
wchang2@andrew.cmu.edu

Yuexin Wu
Carnegie Mellon University
yuexinw@andrew.cmu.edu

Yiming Yang
Carnegie Mellon University
yiming@cs.cmu.edu

ABSTRACT

Extreme multi-label text classification (XMTC) refers to the problem of assigning to each document its most relevant subset of class labels from an extremely large label collection, where the number of labels could reach hundreds of thousands or millions. The huge label space raises research challenges such as data sparsity and scalability. Significant progress has been made in recent years by the development of new machine learning methods, such as tree induction with large-margin partitions of the instance spaces and label-vector embedding in the target space. However, deep learning has not been explored for XMTC, despite its big successes in other related areas. This paper presents the first attempt at applying deep learning to XMTC, with a family of new Convolutional Neural Network (CNN) models which are tailored for multi-label classification in particular. With a comparative evaluation of 7 state-of-the-art methods on 6 benchmark datasets where the number of labels is up to 670,000, we show that the proposed CNN approach successfully scaled to the largest datasets, and consistently produced the best or the second best results on all the datasets. On the Wikipedia dataset with over 2 million documents and 500,000 labels in particular, it outperformed the second best method by 11.7% $\sim$ 15.3% in precision@K and by 11.5% $\sim$ 11.7% in NDCG@K for K = 1,3,5.

1 INTRODUCTION

Extreme multi-label text classification (XMTC), the problem of finding each document its most relevant subset of labels from an extremely large space of categories, becomes increasingly important due to the fast growing of internet contents and the urgent needs for organizational views of big data. For example, Wikipedia has over a million of curator-generated category labels, and an article often has more than one relevant label: the web page of "potato" would be tagged with the class labels of "Solanum", "Root vegetables", "Crops originating from South America", etc. The classification system for Amazon shopping items, as another example, uses a large hierarchy of over one million categories for the organization of shopping items, and each item typically belongs to more than one relevant categories. Solving such multi-label classification problems in an extremely large scale presents open challenges for machine learning research.

Multi-label classification is fundamentally different from the traditional binary or multi-class classification problems which have been intensively studied in the machine learning literature. Binary classifiers treat class labels as independent target variables, which is clearly sub-optimal for multi-label classification as the dependencies among class labels cannot be leveraged. Multi-class classifiers rely on the mutually exclusive assumption about class labels (i.e., one document should have one and only one class label), which is wrong in multi-label settings. Addressing the limitations of those traditional classification methods by explicitly modeling the dependencies or correlations among class labels has been the major focus of multi-label classification research [7, 11, 13, 15, 42, 48]; however, scalable solutions for problems with hundreds of thousands or even millions of labels have become available only in the past few years[5, 36]. Part of the difficulty in solving XMTC problems is due to the extremely severe data sparsity issue. XMTC datasets typically exhibit a power-law distribution of labels, which means a substantial proportion of the labels have very few training instances associated with them. It is, therefore, difficult to learn the dependency patterns among labels reliably. Another significant challenge in XMTC is that the computational costs in both training and testing of mutually independent classifiers would be practically prohibiting when the number of labels reaches hundreds of thousands or even millions.

Significant progress has been made in XMTC recently. Several approaches have been proposed to deal with the huge label space and to address the scalability and data sparsity issues. Most of these approaches fall into two categories: target-embedding methods and tree-based ensemble methods. Let us briefly outline below the key ideas of these two categories and discuss the third category in addition, i.e., the deep learning methods which have made significant impact in multi-class classification problems but have not been explored in XMTC. Comparing the accomplishments and limitations of these three categories of work leads to the design of our proposed work in this paper and the aimed contributions.

1.1 Target-Embedding Methods

Target-embedding methods aim to address the data sparsity issue in training XMTC classifiers by finding a set of low-dimensional embeddings of the label vectors for training instances in the target space. Suppose the training data is given as n pairs of feature vectors and label vectors $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{n}$ where $\boldsymbol{x}_i \in \mathbb{R}^D$ and $\boldsymbol{y}_i \in \{0,1\}^L$, D is the number of features and L is the number of labels. Notice

SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan

DOI: http://dx.doi.org/10.1145/3077136.3080834

that L can be extremely large in XMTC, which means that learning a reliable mapping from arbitrary $\boldsymbol{x}$ to relevant $\boldsymbol{y}$ is difficult from limited training data. Instead, if we can effectively compress the label vectors from L-dimension to $\hat{L}$-dimension via a linear or nonlinear projection, then we can use standard classifiers (such as Support Vector Machines) to efficiently learn a reliable mapping from the feature space to the compressed label space. For classifying a new document we also need to project the predicted embedding of $\boldsymbol{y}$ back to the original high-dimensional space. The projection of target label vectors to their low-dimensional embeddings is called the compression process, and the projection back to the high-dimensional space is called the decompression process. Many variants of the target embedding methods have been proposed [3, 6, 9, 10, 14, 19, 20, 24, 40, 46, 50]. These methods mainly differ in their choices of compression and decompression techniques[36] such as compressed sensing[19, 24], Bloom filters[10], Singular Value Decomposition[39], landmark labels[3], output codes[50], etc. Among those methods, SLEEC [5] is considered representative as it outperformed competing methods on some benchmark datasets. We will provide more precise descriptions of this method in Section 2 as it is a strong baseline in our comparative evaluation (Section 4) in this paper.

1.2 Tree-based Ensemble Methods

Another category of efforts that has improved the state of the art of XMTC in recent years are the new variants of tree-based ensemble methods [1, 36, 41]. Similar to those in classical decision-tree learning, the new methods induce a tree structure which recursively partitions the instance space or sub-spaces at each non-leaf node, and has a base classifier at each leaf node which only focuses on a few active labels in that node. Different from traditional decision trees, on the other hand, the new methods learn a hyperplane (equivalent to using a weighted combination of all features) to split the current instance space at each node, instead of selecting a single feature based on information gain (as in classical decision trees) for the splitting. The hyperplane-based induction is much less greedy than the single-feature based induction of decision trees, and hence potentially more robust for extreme classification with a vast feature space. Another advantage of the tree-based methods is that the prediction time complexity is typically sublinear in the training-set size, and would be logarithmic (as the best case) if the induced tree is balanced. For enhancing the robustness of predictions, most tree-based methods learn an ensemble of trees, each of which is induced based on a randomly selected subset of features at each level of the tree. The top-performing method in this category is FastXML [36], for which we will provide a detailed description in Section 2 as it is a strong baseline in our comparative evaluation (Section 4).

1.3 Deep Learning for Text Classification

It should be noted that all the aforementioned methods are based on bag-of-word representations of documents. That is, words are treated as independent features out of context, which is a fundamental limitation of those methods. How to overcome such a limitation of existing XMTC methods is an open question that has not been studied in sufficient depth. Deep learning models, on the other hand, have achieved great successes recently in other related domains by automatically extracting context-sensitive features from raw text. Those areas include various tasks in natural language understanding [37], language modeling [33], machine translation[38], and more. In multi-class text classification in particular, which is closely related to multi-label classification but restricting each document to having only one label, deep learning approaches have recently outperformed linear predictors (e.g., linear SVM) with bag-of-word based features as input, and become the new state-of-the-art. The strong deep learning models in multi-class text classification include the convolutional neural network by [25] (CNN), the recurrent neural network by [27] (RNN), the combination of CNN and RNN by [49], the CNN with attention mechanism by [2, 43] and the Bow-CNN model by [21, 22]. Although some of those deep learning models were also evaluated on multi-label classification datasets [21], those methods are designed for multi-class settings, not taking multi-label settings into account in model optimization. We will provide more details about the CNN-Kim [25] and Bow-CNN [21] models in Section 2 as the representative deep learning models (which are applied to but not tailored for XMTC) in our comparative evaluation (Section 4).

The great successes of deep learning in multi-class classification and other related areas raise an important question for XMTC research, i.e., can we use deep learning to advance the state of the art in XMTC? Specifically, how can we make deep learning both effective and scalable when both the feature space and label space are extremely large? Existing work in XMTC has not offered the answer; only limited efforts have been reported in this direction, and the solutions have not scaled to extremely large problems [26, 34, 44, 47].

1.4 Our New Contributions

Our primary goal in this paper is to use deep learning to enhance the state of the art of XMTC. We accomplish this goal with the following contributions:

- We re-examine the state of the art of XMTC by conducting a comparative evaluation of 7 methods which are most representative in target-embedding, tree-based ensembling and deep learning approaches to XMTC, on 6 benchmark datasets where the label space sizes are up to 670,000.
- We propose a new deep learning method, namely XML-CNN, which combines the strengths of existing CNN models and goes beyond by taking multi-label co-occurrence patterns into account in both the optimization objective and the design of the neural network architecture, and scales successfully to the largest XMTC benchmark datasets.
- Our extensive experiments show that XML-CNN consistently produces the best or the second best results among all the competing methods on all of the 6 benchmark datasets. On the Wikipedia dataset with over 2 million documents and 500,000 labels in particular, our proposed method outperforms the second best method by 11.7%~15.3% in precision@K and by 11.5%~11.7% in NDCG@K for K = 1,3,5.

The rest of the paper is organized as follows. Section 2 outlines six existing competitive methods which will be used (together with our XML-CNN) in our comparative evaluation. Section 3 describes

the new XML-CNN method. Section 4 reports our extensive experiments and results, followed by conclusion in Section 5.

2 EXISTING COMPETITIVE METHODS

We outline six methods, including the most representative methods in XMTC and some successful deep learning methods which are designed for mutli-class text classification but also applicable to XMTC with minor adaptations. Later in Section 4 we will compare those methods empirically against our proposed XML-CNN method (Section 3).

2.1 SLEEC

SLEEC [5] is most representative for target-embedding methods in XMTC. It consists of two steps of learning embeddings and kNN classification. SLEEC learns $\hat{L}$-dimensional embeddings $\boldsymbol{z}_i \in \mathbb{R}^{\hat{L}}$ for the original L-dimensional label vectors $\boldsymbol{y}_i \in \{0,1\}^L$ that non-linearly capture label correlations by preserving pairwise distances between only closest label vectors, i.e. $d(\boldsymbol{z}_i, \boldsymbol{z}_j) \approx d(\boldsymbol{y}_i, \boldsymbol{y}_j)$ only if i is among j's k nearest neighbors under some distance metric $d(\cdot,\cdot)$. Then regressors $V \in \mathbb{R}^{\hat{L}\times D}$ are learned such that $V\boldsymbol{x}_i \approx \boldsymbol{z}_i$ with ℓ_1 regularization on $V\boldsymbol{x}_i$, which results in sparse solutions.

At prediction time, for a novel document $\boldsymbol{x}^* \in \mathbb{R}^D$ SLEEC performs a kNN search for its projection $z^* = V\boldsymbol{x}^*$ in the $\hat{L}$-dimensional embedding space. To speed up the kNN search, SLEEC groups training data into many clusters, and learns embeddings for each separate cluster, then kNN search is performed only within the cluster this novel document belongs to. Since clustering high dimensional data is usually unstable, an ensemble of SLEEC models are induced with different clusterings to boost prediction accuracy.

2.2 FastXML

FastXML [36] is considered the state-of-the-art tree-based method for XMTC. It learns a hierarchy of training instances and optimizes an NDCG-based objective at each node of the hierarchy. Specifically, a hyperplane parameterized by $w \in \mathbb{R}^D$ is induced at each node, which splits the set of documents in the current node into two subsets; the ranking of the labels in each of the two subsets are jointly learned. The key idea is to have the documents in each subset sharing similar label distribution, and to characterize the distribution using a set-specific ranked list of labels. This is achieved by jointly maximizing NDCG scores of the ranked label lists in the two sibling subsets. In practice, an ensemble of multiple induced trees are learned to improve the robustness of predictions.

At prediction time, each test document is passed from the root to a leaf node in each induced tree, and the label distributions in all the reached leaves are aggregated for the test document. Suppose T trees are induced, H is the average height of the trees, and $\hat{L}$ is the average number of labels per leaf node. The prediction cost is approximately $O(TDH + T\hat{L} + \hat{L}\log\hat{L})$, which is dominated by $O(TDH)$ when $\hat{L}$ is small. If the trees are near balanced, then $H = \log N \approx \log L$, and prediction cost is approximately $O(TD\log L)$, which is logarithmic in the number of labels.

2.3 FastText

FastText [23] is a simple yet effective deep learning method for multi-class text classification. A document representation is constructed by averaging the embeddings of the words that appear in the document, upon which a softmax layer is applied to map the document representation to class labels. This approach was inspired by the recent work on efficient word representation learning, such as skip-gram and CBOW[32]. It ignores word order in the construction of document representations, and uses a linear softmax classifier. This simplicity makes FastText very efficient to train yet achieving state-of-the-art performances on several multi-class classification benchmarks, and often is several orders of magnitude faster than other competing methods [23]. However, simply averaging input word embeddings with the shallow architecture for document-to-label mapping might limit its success in XMTC, as in XMTC, document presentations need to capture much richer information for successfully predicting multiple correlated labels and discriminating them from enormous numbers of irrelevant labels.

2.4 CNN-Kim

CNN-Kim [25] is one of the first attempts of applying convolutional neural networks to text classification. CNN-Kim constructs a document vector with the concatenation of its word embeddings, and then t filters are applied to this concatenated vector in the convolution layer to produce t feature maps, which are in turn fed to a max-over-time pooling layer to construct a t-dimensional document representation. This is followed by a fully-connected layer with L softmax outputs, corresponding to L labels. In practice CNN-Kim has shown excellent performance in multi-class text classification, and is a strong baseline in our comparative evaluations.

2.5 Bow-CNN

Bow-CNN [21] (Bag-of-word CNN) is another strong method in multi-class classification. It represents each small text region (several consecutive words) using a bag-of-word indicator vector (called the one-hot vector). Denoting by D the size of the feature space (the vocabulary), a D-dimensional binary vector is constructed for each region, where the i-th entry is 1 iff the i-th word in the vocabulary appears in the that text region. The embeddings of all regions are passed through a convolutional layer, followed by a special dynamic pooling layer that aggregates the embedded regions into a document representation, and then fed to a softmax output layer.

2.6 PD-Sparse

PD-Sparse [45] is a recently proposed max-margin method designed for extreme multi-label classification. It does not fall into the three categories in Section 1 (target-embedding methods, tree-based methods, and deep learning methods). In PD-Sparse, a linear classifier is learned for each label with ℓ_1 and ℓ_2 penalties on the weight matrix associated with this label. This results in a solution extremely sparse in both the primal and dual spaces, which is desirable in terms of both time and memory efficiency in XMTC. PD-Sparse proposes a Fully-Corrective Block-Coordinate Frank-Wolfe training algorithm that exploits sparsity in the solution and achieves sub-linear training time w.r.t the number of primal and dual variables, but prediction time is still linear w.r.t the number of

labels. [45] has shown that PD-Sparse outperforms 1-vs-all SVM and logistic regression on multi-label classification, with significantly reduced training time and model size.

3 PROPOSED METHOD

Our model architecture (XML-CNN), shown in Figure 1, is based on CNN-Kim [25], the CNN model for multi-class text classification described in 2.4.

Similar to other CNNs[21, 25], our model learns a rich number of feature representations by passing the document through various convolutional filters. The key attributes of our model lie in the following connected layers. Specifically, our model adopts a dynamic max pooling scheme that captures more fine-grained features from different regions of the document. We furthermore utilize a binary cross-entropy loss over sigmoid output that is more tailored for XMTC. An additional hidden bottleneck layer is inserted between pooling and output layer to learn compact document representations, which reduces model size as well as boosts model performance. In the following we describe our model in detail.

Let $\boldsymbol{e}_i \in \mathbb{R}^k$ be the k-dimensional word embedding corresponding to the i-th word in the current document, $i = 1, .., m$. The whole document is represented by the concatenation of its word embeddings $\boldsymbol{e}_{1:m} = [\boldsymbol{e}_1, .., \boldsymbol{e}_m] \in \mathbb{R}^{km}$. In general, the text region from the i-th word to the j-th word is represented by $\boldsymbol{e}_{i:j} = [\boldsymbol{e}_i, .., \boldsymbol{e}_j] \in \mathbb{R}^{k(j-i+1)}$. A convolution filter $\boldsymbol{v} \in \mathbb{R}^{kh}$ is applied to a text region of h words $\boldsymbol{e}_{i:i+h-1}$ to produce a new feature:

$$c_i = g_c(\boldsymbol{v}^T \boldsymbol{e}_{i:j+h-1})$$

where g_c is the nonlinear activation function for this convolution layer, such as sigmoid or ReLU. We omit all bias terms for simplicity. All c_i's together form a feature map $\boldsymbol{c} = [c_1, ..., c_m] \in \mathbb{R}^m$ associated with the applied filter $\boldsymbol{v}$. Here we pad the end of the document to obtain m features if the length is short. Multiple filters with different window sizes are used in a convolution layer to capture rich semantic information. Suppose t filters are used and the t resulting feature maps are $\boldsymbol{c}^{(1)}, .., \boldsymbol{c}^{(t)}$. Then a pooling operation $P(\cdot)$ is applied to each of these t feature maps to produce t p-dimensional vectors $P(\boldsymbol{c}^{(i)}) \in \mathbb{R}^p$. We will discuss the choice of $P(\cdot)$ in Section 3.1. The output of pooling layer is followed by a fully connected bottleneck layer with h hidden units and then an output layer with L units corresponding to the scores assigned to each label, denoted by $\boldsymbol{f} \in \mathbb{R}^L$:

$$\boldsymbol{f} = \boldsymbol{W}_o g_h(\boldsymbol{W}_h[P(\boldsymbol{c}^{(1)}), .., P(\boldsymbol{c}^{(t)})]) \quad (1)$$

Here $\boldsymbol{W}_h \in \mathbb{R}^{h \times tp}$ and $\boldsymbol{W}_o \in \mathbb{R}^{L \times h}$ are weight matrices associated with the bottleneck layer and output layer; g_h is the element-wise activation functions applied to the bottleneck layer. The key attributes that make our model especially suited for XMTC are the pooling operation, loss function, and the hidden bottleneck layer between pooling and output layer. We will verify that each of these three components contributes to performance improvement in XMTC through an ablation test in Section 4.4. In the remainder of this section, we introduce these three key attributes of our model.

3.1 Dynamic Max Pooling

In previous CNN models for text classification, including CNN-Kim, a max-over-time[12] pooling scheme is usually adopted. This simply means taking the maximum element of a feature map: $P(\boldsymbol{c}) = \hat{c} = \max\{\boldsymbol{c}\}$. The idea is to capture the most important feature, i.e. the entry with the largest value, in each feature map. Using max-over-time pooling, each filter generates a single feature, so the output of the pooling layer is $[P(\boldsymbol{c}^{(1)}), .., P(\boldsymbol{c}^{(t)})] = [\hat{c}^{(1)}, .., \hat{c}^{(t)}] \in \mathbb{R}^t$. However, one drawback of max-over-time pooling is that for each filter, only one value, i.e. the largest value in the feature map, is chosen to carry information to subsequent layers, which is especially problematic when the document is long. Moreover, this pooling scheme does not capture any information about the position of the largest value.

In our model, we adopt a dynamic max pooling scheme, which is similar to [8, 21]. Instead of generating only one feature per filter, p features are generated to capture richer information. For a document with m words, we evenly divide its m-dimensional feature map into p chunks, each chunk is pooled to a single feature by taking the largest value within that chunk, so that information about different parts of the document can be received by the top layers. Under this pooling scheme, each filter produces a p-dimensional feature (assuming m is dividable by p):

$$P(\boldsymbol{c}) = [\max\{\boldsymbol{c}_{1:\frac{m}{p}}\}, .., \max\{\boldsymbol{c}_{m-\frac{m}{p}+1:m}\}] \in \mathbb{R}^p$$

which captures both important features and position information about these important features.

3.2 Loss Function

The most straightforward adaptation from the mult-class classification problems to multi-label ones would be to extend the traditional cross-entropy loss. Specifically, [16–18, 25] consider the extended cross-entropy loss function as

$$\min_{\Theta} \quad -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{L} y_{ij}\log(\hat{p}_{ij}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j\in \boldsymbol{y}_i^+}\frac{1}{|\boldsymbol{y}_i^+|}\log(\hat{p}_{ij}),$$

where Θ denotes model parameters, $\boldsymbol{y}_i^+$ denotes the set of relevant labels of instance i and $\hat{p}_{ij}$ is the model prediction for instance i on label j , through a softmax activation:

$$\hat{p}_{ij} = \frac{\exp(f_j(\boldsymbol{x}_i))}{\sum_{j'=1}^{L}\exp(f_{j'}(\boldsymbol{x}_i))}.$$

Another intuitively reasonable objective for XMTC is rank loss [47] that minimizes the number of mis-ordered pairs of relevant and irrelevant labels, i.e. it aims to assign relevant labels with higher scores than irrelevant labels. However, rank loss has shown to be inferior to binary cross-entropy loss (BCE) over sigmoid activation when applied to multi-label classification datasets in a simple feed-forward neural network[34]. The binary cross-entropy objective can be formulated as:

$$\min_{\Theta} \quad -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{L}\left[y_{ij}\log(\sigma(f_{ij})) + (1-y_{ij})\log(1-\sigma(f_{ij}))\right] \quad (2)$$

where σ is sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

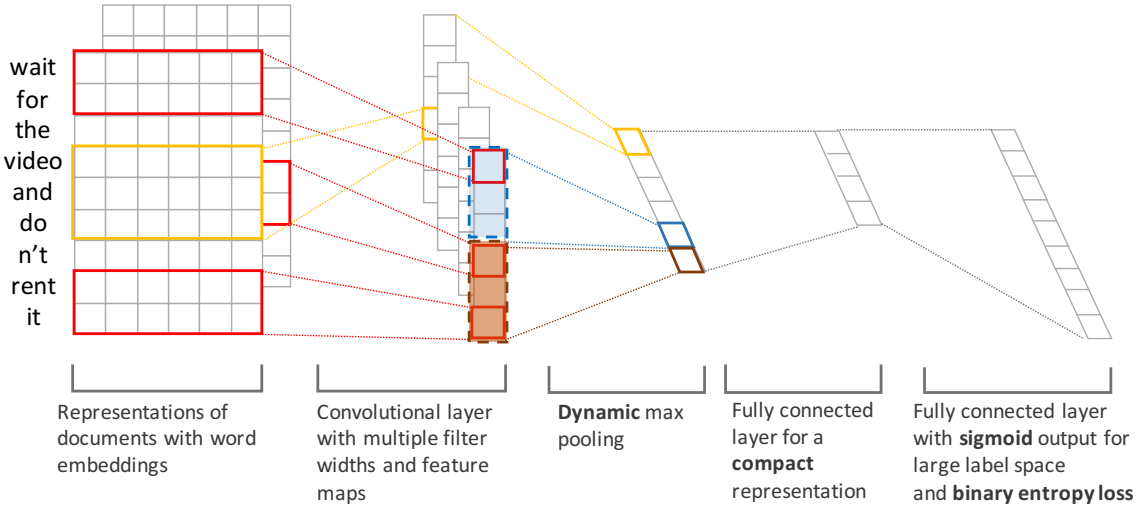

Figure 1: Proposed architecture XML-CNN **with an example sentence.**

We find that the BCE loss is more suited for multi-label problems and outperforms cross-entropy loss in our experiments (Section 4.4). Therefore, we adopt the BCE loss for our final model.

3.3 Hidden Bottleneck Layer

Unlike CNN-Kim [25] or Bow-CNN [21] where pooling layer and output layer are directly connected, we propose to add a fully-connected hidden layer with h units between pooling and output layer, referred to as the hidden bottleneck layer, as the number of its hidden units is far less than the pooling and output layer. The reason for this is two-fold. First, with pooling layer directly connected to output layer, the number of parameters between these two layers is $O(pt \times L)$. When the document is long, and the number of labels is large, more filters and more chunks in pooling operation are needed to obtain good document representations, and in XMTC setting, L can be up to millions, so model size of $O(pt \times L)$ might not fit into a common GPU memory. With an additional hidden bottleneck layer inserted between pooling and output layer, the number of parameters reduces to $O(h \times (pt + L))$, which can be an order of magnitude less than without this bottleneck layer. Second, without this hidden bottleneck layer, the model only has one hidden layer of non-linearity, which is not expressive enough to learn good document representations and classifiers. Experiments in Section 4 show that this hidden bottleneck layer does help to learn better document representations and improve prediction accuracy.

4 EXPERIMENTS

In this section we report our evaluation of the proposed method and the six competing methods introduced in Section 2 on XMTC benchmark datasets, and compare both the effectiveness and scalability of those methods. We also analyze the contribution of each component of our method via an ablation test.

4.1 Datasets

We used six benchmark datasets, including two small-scale datasets RCV1 [29] (103 labels) and EUR-Lex [31] (3865 labels), two medium-scale datasets Amazon-12K [30] (12,277 labels) and Wiki-30K [51] (29,947 labels), and two large-scale datasets Amazon-670K [28] (670,091 labels) and Wiki-500K[1] (501,069 labels). The dataset statistics are summarized in Table 1.

The TF-IDF features and the class labels for all of these datasets are available in the Extreme Classification Repository[1]. In our experiments, the non-deep learning methods (FastXML, SLEEC and PD-Sparse) used the vectors of TF-IDF features to represent input documents, while the deep learning methods (FastText, Bow-CNN and CNN-Kim and XML-CNN) require raw text of documents as input, which we obtained from the authors of the repository[1]. We removed words that do not have corresponding TF-IDF features from the raw text so that the feature set (vocabulary) of each dataset is the same in the experiments for both the deep learning and non-deep learning methods. Each dataset has an official training set and test set, which we used as they are. We reserved 25% of the training data as the validation set for model selection (i.e., for tuning hyper-parameters); the remaining 75% is used for training the models.

4.2 Evaluation Metrics

In extreme multi-label classification datasets, even though the label spaces are very large, each instance only has very few relevant labels (see Table 1). This means that it is important to present a short ranked list of potentially relevant labels for each test instance for user's attention, and to evaluate the quality of such ranked lists with an emphasis on the relevance of the top portion of each list. For these reasons rank-based evaluation metrics have been commonly used for comparing XMTC methods, including the precision at top K (P@K) and the Normalized Discounted Cummulated Gains

[1] https://manikvarma.github.io

datasets	N	M	D	L	$\bar{L}$	$\tilde{L}$	$\bar{W}$	$\hat{W}$
RCV1	23,149	781,265	47,236	103	3.18	729.67	259.47	269.23
EUR-Lex	15,449	3,865	171,120	3,956	5.32	15.59	1,225.20	1,248.07
Amazon-12K	490,310	152,981	135,895	12,277	5.37	214.45	230.25	224.67
Amazon-670K	490,449	153,025	135,895	670,091	5.45	3.99	230.20	224.62
Wiki-30K	12,959	5,992	100,819	29,947	18.74	8.11	2,246.58	2,210.10
Wiki-500K	1,646,302	711,542	2,381,304	501,069	4.87	16.33	750.64	751.42

Table 1: Data Statistics: N is the number of training instances, M is the number of test instances, D is the total number of features, L is the total number of class labels, $\bar{L}$ is the average number of label per document, $\tilde{L}$ is the average number of documents per label, $\bar{W}$ is the average number of words per document in the training set, $\hat{W}$ is the average number of words per document in the test set.

(NDCG@K) [1, 5, 36, 41, 45]. We follow such convention and use these two metrics in our evaluation in this paper, with $k = 1, 3, 5$. Denoting by $\boldsymbol{y} \in \{0,1\}^L$ as the vector of true labels of an document, and $\hat{\boldsymbol{y}} \in \mathbb{R}^L$ as the system-predicted score vector for the same document, the metrics are defined as:

$$\text{P@}k = \frac{1}{k} \sum_{l \in r_k(\hat{\boldsymbol{y}})} y_l$$

$$\text{DCG@}k = \sum_{l \in r_k(\hat{\boldsymbol{y}})} \frac{y_l}{\log(l+1)}$$

$$\text{NDCG@}k = \frac{\text{DCG@}k}{\sum_{l=1}^{\min(k, \|\boldsymbol{y}\|_0)} \frac{1}{\log(l+1)}}$$

where $r_k(\hat{\boldsymbol{y}})$ is the set of rank indices of the truly relevant labels among the top-k portion of the system-predicted ranked list for a document, and $\|\boldsymbol{y}\|_0$ counts the number of relevant labels in the ground truth label vector $\boldsymbol{y}$. P@K and NDCG@K are calculated for each test document and then averaged over all the documents.

4.3 Main Results

The performance of all methods in P@K and NDCG@K on all six datasets are summarized in Tables 2 and 3. Each line compares all the methods on a specific dataset, where the best score is in boldface. We conducted paired t-tests to compared the performance score of each method against the best one in the same line, where the number of documents in each test set is the number of trials.

As we can see in Table 2, XML-CNN has the best scores in 11 out of the 18 lines in the table, and each of the 11 scores is statistically significantly better than the 2nd best score in the corresponding line. On the other hand, the target-embedding method SLEEC is the best in 4 lines, the deep learning method Bow-CNN is the best in 2 lines, and the tree-based ensemble method FastXML is the best in only 1 line. Similar observation can be found in Table 3 of the NDCG@K scores, where XML-CNN has the best results in 12 out of the 18 lines. On the Wiki-500K dataset with over 2 million documents and 500,000 labels in particular, XML-CNN outperformed the second best method by 11.7% $\sim$ 15.3% in P@K and by 11.5% $\sim$ 11.7% in NDCG@K for K = 1,3,5. In the lines XML-CNN does not have the best score, it has the 2nd best score.

Why did XML-CNN perform particularly strong on the datasets of RCV1, Amazon-12K and Wiki-500K? We notice that these datasets have a higher number of training instances per class than that in other datasets (see the column of $\tilde{L}$ in Table 1). This confirms our intuition about deep learning: it typically requires more training data in order to achieve advantageous performance than other methods, and XML-CNN successfully took this advantage on these datasets. As for why other deep learning methods especially CNN-Kim performed much worse on the other hand, we will analyze this via the ablation test in Section 4.4.

As for why SLEEC, the leading target-embedding method, had the strongest performance on EUR-Lex and Wiki-30K in particular, we notice that these two datasets have much longer documents than other datasets (see column of $\bar{W}$ in Table 1). In other words, the feature vectors of documents in these two datasets are denser and possibly information-richer than those in other datasets. As we described in Section 2, SLEEC uses a linear regression model to establish the mapping from documents to label vectors in a dimension reduced target space. But why such a relatively simple linear mapping should lead to better predictions for long documents than the more complex deep learning or tree-based models is not clear at this point – answering this question requires future research.

An interesting observation is that all the methods performed relatively well on RCV1, and each method has much stronger performance on this dataset than itself on other datasets. Notice that this dataset has a much smaller number of labels (103 only) than those of other datasets, and the smallest number of labels (3.18) per document. These means that this dataset is the least challenging one among the six included in this study.

Another interesting observation is that FastText and CNN-Kim had significantly worse results than other methods on Amazon-670K (with 670K labels) and Wiki-500K (with 500K labels). The extremely huge label spaces mean that the data sparsity issue would be severe, especially for the long tail of rare labels. Both FastText and CNN-Kim are designed for multi-class classification and not for modeling label correlations, and hence suffered significantly in XMTC when the label spaces are extremely large. Other models like PD-Sparse and Bow-CNN would have a similar weakness but

we cannot examine them empirically as they failed to scale up on these two datasets.

4.4 Ablation Test

Given that CNN-Kim and XML-CNN are closely coupled in the sense that the latter is an extension of the former by adding a few components that especially suit XMTC, it would be informative to empirically examine the impact of each new component in XML-CNN via an ablation test. The results are shown in Figure 2, where CNN-Kim is the original model designed for multi-class problems; suffix v1 denotes the model obtained by replacing the original loss function of CNN-Kim by the binary cross-entropy loss over sigmoid output (Section 3.2); suffix v2 denotes the model obtained by inserting the hidden bottleneck layer to model v1; suffix v3 denotes the model obtained by adding the dynamic pooling layer to model v2. That is, v3 is the full version of our XML-CNN model. We show results on three representative datasets in terms of their category sizes (small, medium and large scale). Results on the other three datasets demonstrate similar trends and we omit them here due to space limit.

Overall, we can see that each of the three new components improved the performance of XML-CNN. First, the new loss function certainly is more suitable for dealing with multi-label problems. We also found that the hidden bottleneck layer not only consistently boosted the performance of XML-CNN, but also improved the scalability as it compresses the output of the pooling layer into a smaller one and makes XML-CNN scalable on the largest dataset. Last but not least, dynamic max-pooling layer plays a crucial role in our model, leading to significant improvement of XML-CNN over CNN-Kim, indicating its effectiveness in extracting richer information from the context of documents.

4.5 Efficiency and Scalability

The training and testing time in clock-time seconds for all the methods on the six benchmark datasets are summarized in Table 4. Notice that four of those methods (FastXML, SLEEC, PD-Sparse and FastText) are implemented for running on CPU, while the remaining deep learning methods (Bow-CNN, CNN-Kim and XML-CNN) were implemented for running on GPU. The left portion of Table 4 presents the time in seconds on CPU for the former group (running time using a single thread is reported), and the right portion presents the time in seconds on GPU for the latter group. Although the two groups are not directly comparable, they are informative in a practical sense. For the CPU runs we used Intel(R) Xeon(R) CPU E5-2630v3 2.40GHz with 192 GB memory, and for the GPU runs we used Nvidia GTX TITAN X.

In addition, we also ran all the methods on subsets of the Wiki-500K data, which were obtained by randomly sampling 1%, 5%, 10% and 25% of its 501,069 labels and corresponding documents. Figure 3 summarizes the training time on CPU or GPU in seconds, depending on the methods. Three methods (SLEEC, Bow-CNN and PD-Sparse) are not scalable to the fullset and the larger subsets of Wiki-500K because that their excessive memory requirements exceed the limits of our machines, and hence the early stop of their curves. In order to fill in the training-time slot for SLEEC in Table 4 on Wiki-500K, we had to reduce the feature space to 5000 instead of the full dimension of D=2,381,304; otherwise it could not fit into our memory. Thus the listed CPU seconds (209,735) in this table does not reflect the full complexity of SLEEC in training.

Combining Table 4 and Figure 3 we have the following points:

(1) For scalability comparison, time complexities on the largest datasets, e.g., Wiki-500K and Amazon-670K, are most informative. Only four methods, i.e., XML-CNN, CNN-Kim, FastXML and FastText successfully scaled to these extremely large problems; the remaining methods failed for memory issues;
(2) Both XML-CNN and FastXML (the 4th best method on average in this study) scaled well with both extremely large feature spaces and label spaces;
(3) SLEEC (the 2nd best method on average in this study) scaled well with extremely large label spaces (e.g., on Amazon-670K) but suffered from extremely large feature spaces (e.g., on Wiki-500K);
(4) CNN-Kim has comparable time complexities as XML-CNN in both training and testing, but CNN-Kim performed substantially worse on all of the datasets;
(5) FastText is less efficient on EUR-Lex, Wiki-30K as its training cost is proportional to document length on average;
(6) Bow-CNN is relatively fast in training if it were not running into memory issues. This is partly due to its implementation[2]; also, it has a dynamic pooling layer directly connected to output layer, resulting in $O(ptL)$ memory (as we discussed in Section 3.3), where t, p are the number of filters and pooling chunks, which can be an order of magnitude larger than that of XML-CNN ($O(h(pt + L))$) and CNN-Kim ($O(tL)$).

4.6 Experimental Details

In our proposed model XML-CNN, we used rectified linear units as activation functions, one-dimensional convolutional filters with the window sizes of 2, 4, 8; the number of feature maps for each filter was 128 for small datasets (RCV1, Wiki-30K, EUR-Lex, Amazon-12K) and 32 for large datasets (Wiki-500K, Amazon-670K), dropout rate was $p = 0.5$, and the number of hidden units of the bottleneck layer was 512. These hyper-parameters were fixed across all datasets.

For all the six baseline methods (Section 2) except for CNN-Kim, we used the author-provided implementations [1][2][3]; XML-CNN and CNN-Kim are implemented in THEANO[4]. For SLEEC, the number of learners was set to 15, embedding dimension was set to 100 for small datasets and 50 for large datasets as suggested in [5]. For FastXML, the number of trees was set to 50 and hyper-parameter $C_\delta = C_r = 1.0$ as suggested in [36]. For PD-Sparse, the tuning parameter C was set to 1.0. All other hyper-parameters of these methods were chosen on the validation set.

For word embeddings in deep learning models (except for Bow-CNN, which uses one-hot vectors as input) we used pre-trained 300-dimensional Glove vectors [35].

[2] http://riejohnson.com/cnn_download.html
[3] https://github.com/facebookresearch/fastText

		Methods						
Datasets	Metrics	FastXML	SLEEC	PD-Sparse	FastText	Bow-CNN	CNN-Kim	XML-CNN
RCV1	*P*@1	94.62	95.35	95.16	95.40	96.40	93.54	**96.86**
	P@3	78.40	79.51	79.46	79.96	**81.17**	76.15	81.11
	P@5	54.82	55.06	55.61	55.64	**56.74**	52.94	56.07
EUR-Lex	*P*@1	68.12	**78.21**	72.10	71.51	64.99	42.84	76.38
	P@3	57.93	**64.33**	57.74	60.37	51.68	34.92	62.81
	P@5	48.97	**52.47**	47.48	50.41	42.32	29.01	51.41
Amazon-12K	*P*@1	94.58	93.61	88.80	82.11	92.77	90.31	**95.06**
	P@3	78.69	79.13	70.69	71.94	69.85	74.34	**79.86**
	P@5	62.26	63.54	55.70	58.96	48.74	58.78	**63.91**
Amazon-670K	*P*@1	**35.59**	34.54	-	8.93	-	15.19	35.39
	P@3	31.87	30.89	-	9.07	-	13.78	**31.93**
	P@5	28.96	28.25	-	8.83	-	12.64	**29.32**
Wiki-30K	*P*@1	83.26	**85.96**	82.32	66.19	81.09	78.93	84.06
	P@3	68.74	73.13	66.96	55.44	50.64	55.48	**73.96**
	P@5	58.84	62.73	55.60	48.03	35.98	45.06	**64.11**
Wiki-500K	*P*@1	49.27	53.60	-	8.66	-	23.38	**59.85**
	P@3	33.30	34.51	-	7.32	-	11.95	**39.28**
	P@5	25.63	25.85	-	6.54	-	8.59	**29.81**

Table 2: Results in P@K – bold face indicates the best method in each line; * denotes the method (if any) whose score is not statistically significantly different from the best one; '-' denotes the methods which failed to scale due to memory issues.

		Methods						
Datasets	Metrics	FastXML	SLEEC	PD-Sparse	FastText	Bow-CNN	CNN-Kim	XML-CNN
RCV1	*G*@1	94.62	95.35	95.16	95.40	96.40	93.54	**96.88**
	G@3	89.21	90.45	90.29	90.95	92.04*	87.26	**92.22**
	G@5	90.27	90.97	91.29	91.68	**92.89**	88.20	92.63
EUR-Lex	*G*@1	68.12	**78.21**	72.10	71.51	64.99	42.84	76.38
	G@3	60.66	**67.86**	61.33	63.32	55.03	36.95	66.28
	G@5	56.42	**61.57**	55.93	58.56	49.92	33.83	60.32
Amazon-12K	*G*@1	94.58	93.61	88.80	82.11	92.77	90.31	**95.06**
	G@3	88.26	88.61	80.48	79.90	80.96	83.87	**89.48**
	G@5	84.89	86.46	77.81	79.36	73.01	81.21	**87.06**
Amazon-670K	*G*@1	**35.59**	34.54	-	8.93	-	15.19	35.39
	G@3	33.36	32.70	-	9.51	-	14.60	**33.74**
	G@5	31.76	31.54	-	9.74	-	14.12	**32.64**
Wiki-30K	*G*@1	83.26	**85.96**	82.32	66.19	81.09	78.93	84.06
	G@3	72.07	76.10*	70.54	57.87	57.26	60.52	**76.35**
	G@5	64.34	68.14	61.74	52.12	45.28	51.96	**68.94**
Wiki-500K	*G*@1	49.27	53.60	-	8.66	-	23.38	**59.85**
	G@3	41.64	43.64	-	8.43	-	15.45	**48.67**
	G@5	40.16	41.31	-	8.86	-	13.64	**46.12**

Table 3: Results in NDCG@K – bold face indicates the best method in each line; * denotes the method (if any) whose score is not statistically significantly different from the best one; '-' denotes the methods which failed to scale due to memory issues.

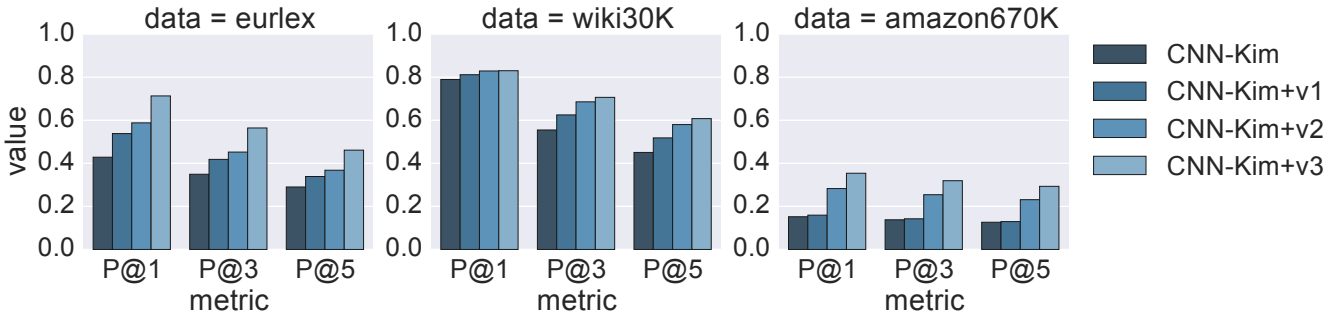

Figure 2: Result of the ablation test – v1 uses the new loss function for multi-label classification over CNN-Kim; v2 denotes the insertion of the additional hidden layers over v1, and v3 denotes the insertion of the dynamic max pooling layer over v2 (hence the full version of XML-CNN).

	CPU								GPU					
	FastXML		SLEEC		PD-Sparse		FastText		Bow-CNN		CNN-Kim		XML-CNN	
dataset	train	test	train	test	train	test	train	test	train	test	train	test	train	test
RCV1	552	615	888	1,764	20	15	858	184	804	85	3,720	92	2,340	89
EUR-Lex	756	55	4,617	16	322	0.4	19,518	7	1,665	4	660	0.6	1,020	0.7
Amazon-12K	27,949	1,015	90,925	2,506	2,399	15	20,790	499	11,097	135	41,460	34	48,000	45
Amazon-670K	47,342	1,300	63,991	2,856	-	-	19,353	26,746	-	-	138,060	889	188,040	2,476
Wiki-30K	2,047	21	2,646	0.6	238	60	6,354	20	2,377	20	1,020	3	5,280	9
Wiki-500K	192,741	8,972	209,735	20,002	-	-	721,444	84,442	-	-	494,400	4,534	422,040	16,511

Table 4: Training and testing time (seconds) of 7 methods on 6 datasets

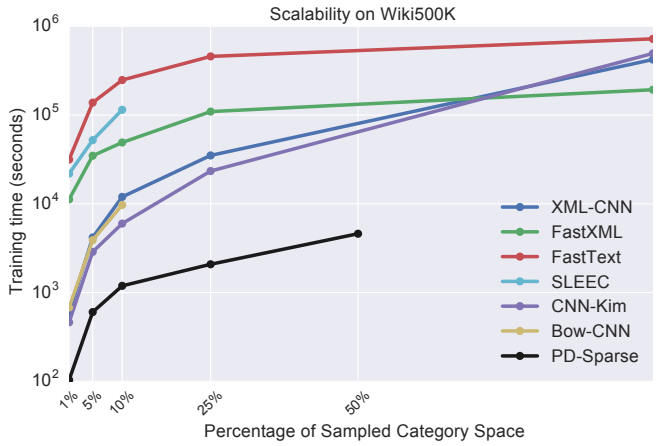

Figure 3: Scalability results on Wiki-500K.

5 CONCLUSION

This paper presented a new deep learning approach to extreme multi-label text classification, and evaluated the proposed method in comparison with other state-of-the-art methods on six benchmark datasets where the label-set sizes are up to 670K. Our proposed method XML-CNN goes beyond the deep learning methods for binary or multi-class classification by using a dynamic max pooling scheme that captures richer information from different regions of the document, a binary cross-entropy loss that is more suitable for multi-label problems, and a hidden bottleneck layer for better representations of documents as well as for reducing model size. The advantageous performance of XML-CNN over other competing methods is evident as it obtained the best or second best results on all the benchmark datasets in our comparative evaluation. We hope this study is informative for XMTC research.

ACKNOWLEDGMENTS

We thank the reviewers for their helpful comments. This work is supported in part by the National Science Foundation (NSF) under grant IIS-1546329.

REFERENCES

[1] 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 *Proceedings of the 22nd international conference on World Wide Web*. ACM, 13–24.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Krishnakumar Balasubramanian and Guy Lebanon. 2012. The landmark selection method for multiple output prediction. *arXiv preprint arXiv:1206.6479* (2012).

[4] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conf.* 1–7.

[5] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*. 730–738.

[6] Wei Bi and James Tin-Yau Kwok. 2013. Efficient Multi-label Classification with Many Labels.. In *ICML (3)*. 405–413.

[7] Matthew R Boutell, Jiebo Luo, Xipeng Shen, and Christopher M Brown. 2004. Learning multi-label scene classification. *Pattern recognition* 37, 9 (2004), 1757–1771.

[8] Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. 2015. Event Extraction via Dynamic Multi-Pooling Convolutional Neural Networks.. In *ACL (1)*. 167–176.

[9] Yao-Nan Chen and Hsuan-Tien Lin. 2012. Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*. 1529–1537.

[10] Moustapha M Cisse, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari. 2013. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*. 1851–1859.

[11] Amanda Clare and Ross D King. 2001. Knowledge discovery in multi-label phenotype data. In *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 42–53.

[12] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12, Aug (2011), 2493–2537.

[13] André Elisseeff and Jason Weston. 2001. A kernel method for multi-labelled classification. In *Advances in neural information processing systems*. 681–687.

[14] Chun-Sung Ferng and Hsuan-Tien Lin. 2011. Multi-label Classification with Error-correcting Codes.. In *ACML*. 281–295.

[15] Johannes Fürnkranz, Eyke Hüllermeier, Eneldo Loza Mencía, and Klaus Brinker. 2008. Multilabel classification via calibrated label ranking. *Machine learning* 73, 2 (2008), 133–153.

[16] Sayan Ghosh, Eugene Laksana, Stefan Scherer, and Louis-Philippe Morency. 2015. A multi-label convolutional neural network approach to cross-domain action unit detection. In *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*. IEEE, 609–615.

[17] Yunchao Gong, Yangqing Jia, Thomas Leung, Alexander Toshev, and Sergey Ioffe. 2013. Deep convolutional ranking for multilabel image annotation. *arXiv preprint arXiv:1312.4894* (2013).

[18] Matthieu Guillaumin, Thomas Mensink, Jakob Verbeek, and Cordelia Schmid. 2009. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 309–316.

[19] Daniel Hsu, Sham Kakade, John Langford, and Tong Zhang. 2009. Multi-Label Prediction via Compressed Sensing.. In *NIPS*, Vol. 22. 772–780.

[20] Shuiwang Ji, Lei Tang, Shipeng Yu, and Jieping Ye. 2008. Extracting shared subspace for multi-label classification. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 381–389.

[21] Rie Johnson and Tong Zhang. 2015. Effective use of word order for text categorization with convolutional neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologie*. 103–112.

[22] Rie Johnson and Tong Zhang. 2015. Semi-supervised convolutional neural networks for text categorization via region embedding. In *Advances in neural information processing systems*. 919–927.

[23] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759* (2016).

[24] Ashish Kapoor, Raajay Viswanathan, and Prateek Jain. 2012. Multilabel classification using bayesian compressed sensing. In *Advances in Neural Information Processing Systems*. 2645–2653.

[25] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1746–1751.

[26] Gakuto Kurata, Bing Xiang, and Bowen Zhou. 2016. Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence. In *Proceedings of NAACL-HLT*. 521–526.

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

[28] Jure Leskovec and Andrej Krevl. 2015. {SNAP Datasets}:{Stanford} Large Network Dataset Collection. (2015).

[29] 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.

[30] Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 165–172.

[31] Eneldo Loza Mencia and Johannes Fürnkranz. 2008. Efficient pairwise multilabel classification for large-scale problems in the legal domain. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 50–65.

[32] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[33] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*, Vol. 2. 3.

[34] Jinseok Nam, Jungi Kim, Eneldo Loza Mencía, Iryna Gurevych, and Johannes Fürnkranz. 2014. Large-scale multi-label text classificationfi!?revisiting neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 437–452.

[35] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global Vectors for Word Representation.. In *EMNLP*, Vol. 14. 1532–1543.

[36] Yashoteja Prabhu and Manik Varma. 2014. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 263–272.

[37] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, Vol. 1631. Citeseer, 1642.

[38] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[39] Farbound Tai and Hsuan-Tien Lin. 2012. Multilabel classification with principal label space transformation. *Neural Computation* 24, 9 (2012), 2508–2542.

[40] Jason Weston, Samy Bengio, and Nicolas Usunier. 2011. Wsabie: Scaling up to large vocabulary image annotation. (2011).

[41] Jason Weston, Ameesh Makadia, and Hector Yee. 2013. Label Partitioning For Sublinear Ranking.. In *ICML (2)*. 181–189.

[42] Yiming Yang and Siddharth Gopal. 2012. Multilabel classification with meta-level features in a learning-to-rank framework. *Machine Learning* 88, 1-2 (2012), 47–68.

[43] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

[44] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. 2017. Learning Deep Latent Spaces for Multi-Label Classification. (2017).

[45] Ian EH Yen, Xiangru Huang, Kai Zhong, Pradeep Ravikumar, and Inderjit S Dhillon. 2016. PD-Sparse: A Primal and Dual Sparse Approach to Extreme Multiclass and Multilabel Classification. (2016).

[46] Hsiang-Fu Yu, Prateek Jain, Purushottam Kar, and Inderjit S Dhillon. 2014. Large-scale Multi-label Learning with Missing Labels.. In *Proceedings of the 31th International Conference on Machine Learning*. 593–601.

[47] Min-Ling Zhang and Zhi-Hua Zhou. 2006. Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering* 18, 10 (2006), 1338–1351.

[48] Min-Ling Zhang and Zhi-Hua Zhou. 2007. ML-KNN: A lazy learning approach to multi-label learning. *Pattern recognition* 40, 7 (2007), 2038–2048.

[49] Rui Zhang, Honglak Lee, and Dragomir Radev. 2016. Dependency sensitive convolutional neural networks for modeling sentences and documents. *arXiv preprint arXiv:1611.02361* (2016).

[50] Yi Zhang and Jeff G Schneider. 2011. Multi-Label Output Codes using Canonical Correlation Analysis.. In *AISTATS*. 873–882.

[51] Arkaitz Zubiaga. 2012. Enhancing navigation on wikipedia with social tags. *arXiv preprint arXiv:1202.5469* (2012).