HDLTex: Hierarchical Deep Learning for Text Classification

Kamran Kowsari*, Donald E. Brown[§‡], Mojtaba Heidarysafa[§],
Kiana Jafari Meimandi[§], Matthew S. Gerber[§‡], and Laura E. Barnes[§‡]
*Department of Computer Science, University of Virginia, Charlottesville, VA, USA
[§] Department of System and Information Engineering, University of Virginia, Charlottesville, VA, USA
[‡] Data Science Institute, University of Virginia, Charlottesville, VA, USA
{kk7nc, deb, mh4pk, kj6vd, msg8u, lbarnes}@virginia.edu

***Abstract*—Increasingly large document collections require improved information processing methods for searching, retrieving, and organizing text. Central to these information processing methods is document classification, which has become an important application for supervised learning. Recently the performance of traditional supervised classifiers has degraded as the number of documents has increased. This is because along with growth in the number of documents has come an increase in the number of categories. This paper approaches this problem differently from current document classification methods that view the problem as multi-class classification. Instead we perform hierarchical classification using an approach we call Hierarchical Deep Learning for Text classification (HDLTex). HDLTex employs stacks of deep learning architectures to provide specialized understanding at each level of the document hierarchy.**

***Index Terms*—Text Mining; Document Classification; Deep Neural Networks; Hierarchical Learning; Deep Learning**

I. Introduction

Each year scientific researchers produce a massive number of documents. In 2014 the 28,100 active, scholarly, peer-reviewed, English-language journals published about 2.5 million articles, and there is evidence that the rate of growth in both new journals and publications is accelerating [1]. The volume of these documents has made automatic organization and classification an essential element for the advancement of basic and applied research. Much of the recent work on automatic document classification has involved supervised learning techniques such as classification trees, naïve Bayes, support vector machines (SVM), neural nets, and ensemble methods. Classification trees and naïve Bayes approaches provide good interpretability but tend to be less accurate than the other methods.

However, automatic classification has become increasingly challenging over the last several years due to growth in corpus sizes and the number of fields and sub-fields. Areas of research that were little known only five years ago have now become areas of high growth and interest. This growth in sub-fields has occurred across a range of disciplines including biology (e.g., CRISPR-CA9), material science (e.g., chemical programming), and health sciences (e.g., precision medicine). This growth in sub-fields means that it is important to not just label a document by specialized area but to also organize it within its overall field and the accompanying sub-field. This is hierarchical classification.

Although many existing approaches to document classification can quickly identify the overall area of a document, few of them can rapidly organize documents into the correct sub-fields or areas of specialization. Further, the combination of top-level fields and all sub-fields presents current document classification approaches with a combinatorially increasing number of class labels that they cannot handle. This paper presents a new approach to hierarchical document classification that we call Hierarchical Deep Learning for Text classification (HDLTex).[1] HDLTex combines deep learning architectures to allow both overall and specialized learning by level of the document hierarchy. This paper reports our experiments with HDLTex, which exhibits improved accuracy over traditional document classification methods.

II. Related Work

Document classification is necessary to organize documents for retrieval, analysis, curation, and annotation. Researchers have studied and developed a variety of methods for document classification. Work in the information retrieval community has focused on search engine fundamentals such as indexing and dictionaries that are considered core technologies in this field [2]. Considerable work has built on these foundational methods to provide improvements through feedback and query reformulation [3], [4].

More recent work has employed methods from data mining and machine learning. Among the most accurate of these techniques is the support vector machine (SVM) [5]–[7]. SVMs use kernel functions to find separating hyperplanes in high-dimensional spaces. Other kernel methods used for information retrieval include string kernels such as the spectrum kernel [8] and the mismatch kernel [9], which are widely used with DNA and RNA sequence data.

SVM and related methods are difficult to interpret. For this reason many information retrieval systems use decision trees [3] and naïve Bayes [10], [11] methods. These methods are easier to understand and, as such, can support query

[1] HDLTex is shared as an open source tool at https://github.com/kk7nc/HDLTex

reformulation, but they lack accuracy. Some recent work has investigated topic modeling to provide similar interpretations as naïve Bayes methods but with improved accuracy [12].

This paper uses newer methods of machine learning for document classification taken from deep learning. Deep learning is an efficient version of neural networks [13] that can perform unsupervised, supervised, and semi-supervised learning [14]. Deep learning has been extensively used for image processing, but many recent studies have applied deep learning in other domains such as text and data mining. The basic architecture in a neural network is a fully connected network of nonlinear processing nodes organized as layers. The first layer is the input layer, the final layer is the output layer, and all other layers are hidden. In this paper, we will refer to these fully connected networks as Deep Neural Networks (DNN). Convolutional Neural Networks (CNNs) are modeled after the architecture of the visual cortex where neurons are not fully connected but are spatially distinct [15]. CNNs provide excellent results in generalizing the classification of objects in images [16]. More recent work has used CNNs for text mining [17]. In research closely related to the work in this paper, Zhang et al. [18] used CNNs for text classification with character-level features provided by a fully connected DNN. Regardless of the application, CNNs require large training sets. Another fundamental deep learning architecture used in this paper is the Recurrent Neural Network (RNN). RNNs connect the output of a layer back to its input. This architecture is particularly important for learning time-dependent structures to include words or characters in text [19]. Deep learning for hierarchical classification is not new with this paper, although the specific architectures, the comparative analyses, and the application to document classification are new. Salakhutdinov [20], [21] used deep learning to hierarchically categorize images. At the top level the images are labeled as animals or vehicles. The next level then classifies the kind of animal or vehicle. This paper describes the use of deep learning approaches to create a hierarchical document classification approach. These deep learning methods have the promise of providing greater accuracy than SVM and related methods. Deep learning methods also provide flexible architectures that we have used to produce hierarchical classifications. The hierarchical classification our methods produce is not only highly accurate but also enables greater understanding of the resulting classification by showing where the document sits within a field or area of study.

III. Baseline Techniques

This paper compares fifteen methods for performing document classification. Six of these methods are baselines since they are used for traditional, non-hierarchical document classification. Of the six baseline methods three are widely used for document classification: term-weighted support vector machines [22], multi-word support vector machines [23], and naïve Bayes classification (NBC). The other three are newer deep learning methods that form the basis for our implementation of a new approach for hierarchical document classification. These deep learning methods are described in Section V.

A. Support Vector Machines (SVMs)

Vapnik and Chervonenkis introduced the SVM in 1963 [24], [25], and in 1992 Boser et al. introduced a nonlinear version to address more complex classification problems [26]. The key idea of the nonlinear SVM is the generating kernel shown in Equation 1, followed by Equations 2 and 3:

$$K(x, x') =< \phi(x), \phi(x') > \tag{1}$$

$$f(x) = \sum_{x_i \in training} \alpha_i K(x, x_i) + b \tag{2}$$

$$\max_{\alpha_1,...,\alpha_n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} \alpha_j \alpha_k y_j y_k K(x_j, x_k) \tag{3}$$
$$\forall \alpha_i \geq 0 i \in 1, .., n.$$

Multi-Class SVM: Text classification using string kernels within SVMs has been successful in many research projects [27]. The original SVM solves a binary classification problem; however, since document classification often involves several classes, the binary SVM requires an extension. In general, the multi-class SVM (MSVM) solves the following optimization:

$$\min_{w_1, w_2, .., w_k, \zeta} \frac{1}{2} \sum_k w_k^T w_k + C \sum_{(x_i, y_i) \in D} \zeta_i \tag{4}$$

$$st. \quad w_{y_i}^T x - w_k^T x \leq i - \zeta_i, \quad \forall (x_i, y_i) \in D, k \in \{1, 2, ..., K\}, k \neq y_i \tag{5}$$

where k indicates number of classes, ζ_i are slack variables, and w is the learning parameter. To solve the MSVM we construct a decision function of all k classes at once [22], [28]. One approach to MSVM is to use binary SVM to compare each of the $k(k-1)$ pairwise classification labels, where k is the number of labels or classes. Yet another technique for MSVM is one-versus-all, where the two classes are one of the k labels versus all of the other $k-1$ labels.

Stacking Support Vector Machines (SVM): We use Stacking SVMs as another baseline method for comparison with HDLTex. The stacking SVM provides an ensemble of individual SVM classifiers and generally produces more accurate results than single-SVM models [29], [30].

B. Naïve Bayes classification

Naïve Bayes is a simple supervised learning technique often used for information retrieval due to its speed and interpretability [2], [31]. Suppose the number of documents is n and each document has the label $c, c \in \{c_1, c_2, ..., c_k\}$, where k is the number of labels. Naïve Bayes calculates

$$P(c \mid d) = \frac{P(d \mid c) P(c)}{P(d)} \tag{6}$$

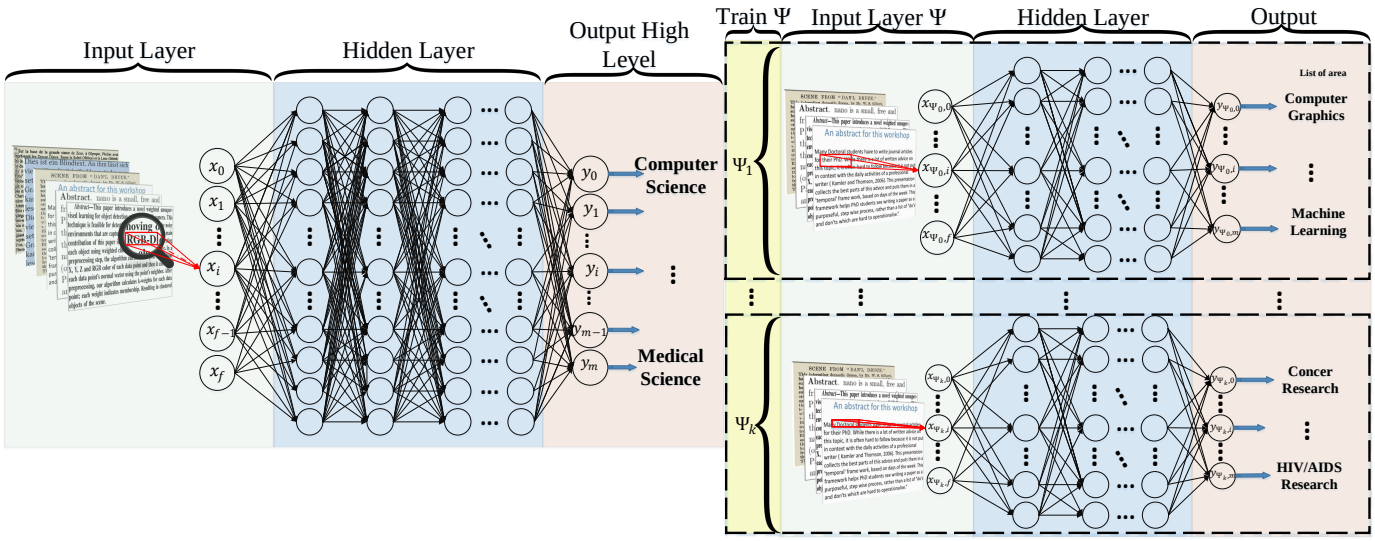

Fig. 1: HDLTex: Hierarchical Deep Learning for Text Classification. This is our Deep Neural Network (DNN) approach for text classification. The left figure depicts the parent-level of our model, and the right figure depicts child-level models defined by Ψ_i as input documents in the parent level.

where d is the document, resulting in

$$\begin{aligned} C_{MAP} &= arg\max_{c \in C} P(d \mid c)P(c) \\ &= arg\max_{c \in C} P(x_1, x_2, ..., x_n \mid c)P(c). \end{aligned} \tag{7}$$

The naïve Bayes document classifier used for this study uses word-level classification [11]. Let $\hat{\theta}_j$ be the parameter for word j, then

$$P(c_j \mid d_i; \hat{\theta}) = \frac{P(c_j \mid \hat{\theta})P(d_i \mid c_j; \hat{\theta}_j)}{P(d_i \mid \hat{\theta})}. \tag{8}$$

IV. Feature Extraction

Documents enter our hierarchical models via features extracted from the text. We employed different feature extraction approaches for the deep learning architectures we built. For CNN and RNN, we used the text vector-space models using 100 dimensions as described in Glove [32]. A vector-space model is a mathematical mapping of the word space, defined as

$$d_j = (w_{1,j}, w_{2,j}, ..., w_{i,j}..., w_{l_j,j}) \tag{9}$$

where l_j is the length of the document j, and $w_{i,j}$ is the Glove word embedding vectorization of word i in document j.

For DNN, we used count-based and term frequency–inverse document frequency (tf-idf) for feature extraction. This approach uses counts for N-grams, which are sequences of N words [33], [34]. For example, the text "In this paper we introduced this technique" is composed of the following N-grams:

- Feature count (1): { (In, 1) , (this, 2), (paper, 1), (we, 1), (introduced, 1), (technique, 1) }
- Feature count (2): { (In, 1) , (this, 2), (paper, 1), (we, 1), (introduced, 1), (technique, 1), (In this, 1), (This paper, 1), (paper we, 1),...}

Where the counts are indexed by the maximum N-grams. So Feature count (2) includes both 1-grams and 2-grams. The resulting DNN feature space is

$$\begin{aligned} f_{j,n} = &[x_{(j,0)}, ..., x_{(j,k-1)}, x_{j,\{0,1\}}, ..., \\ &x_{j,\{k-2,k-1\}}, ..., x_{j,\{k-n,...,k-1\}}] \end{aligned} \tag{10}$$

where f is the feature space of document j for n-grams of size $n, n \in \{0, 1, ..., N\}$, and x is determined by word or n-gram counts. Our algorithm is able to use N-grams for features within deep learning models [35].

V. Deep Learning Neural Networks

The methods used in this paper extend the concepts of deep learning neural networks to the hierarchical document classification problem. Deep learning neural networks provide efficient computational models using combinations of non-linear processing elements organized in layers. This organization of simple elements allows the total network to generalize (i.e., predict correctly on new data) [36]. In the research described here, we used several different deep learning techniques and combinations of these techniques to create hierarchical document classifiers. The following subsections provide an overview of the three deep learning architectures we used: Deep Neural Networks (DNN), Recurrent Neural Networks(RNN), and Convolutional Neural Networks (CNN).

A. Deep Neural Networks (DNN)

In the DNN architecture each layer only receives input from the previous layer and outputs to the next layer. The layers are fully connected. The input layer consists of the text features (see IV) and the output layer has a node for each classification label or only one node if it is a binary classification. This architecture is the baseline DNN. Additional details on this architecture can be found in [37].

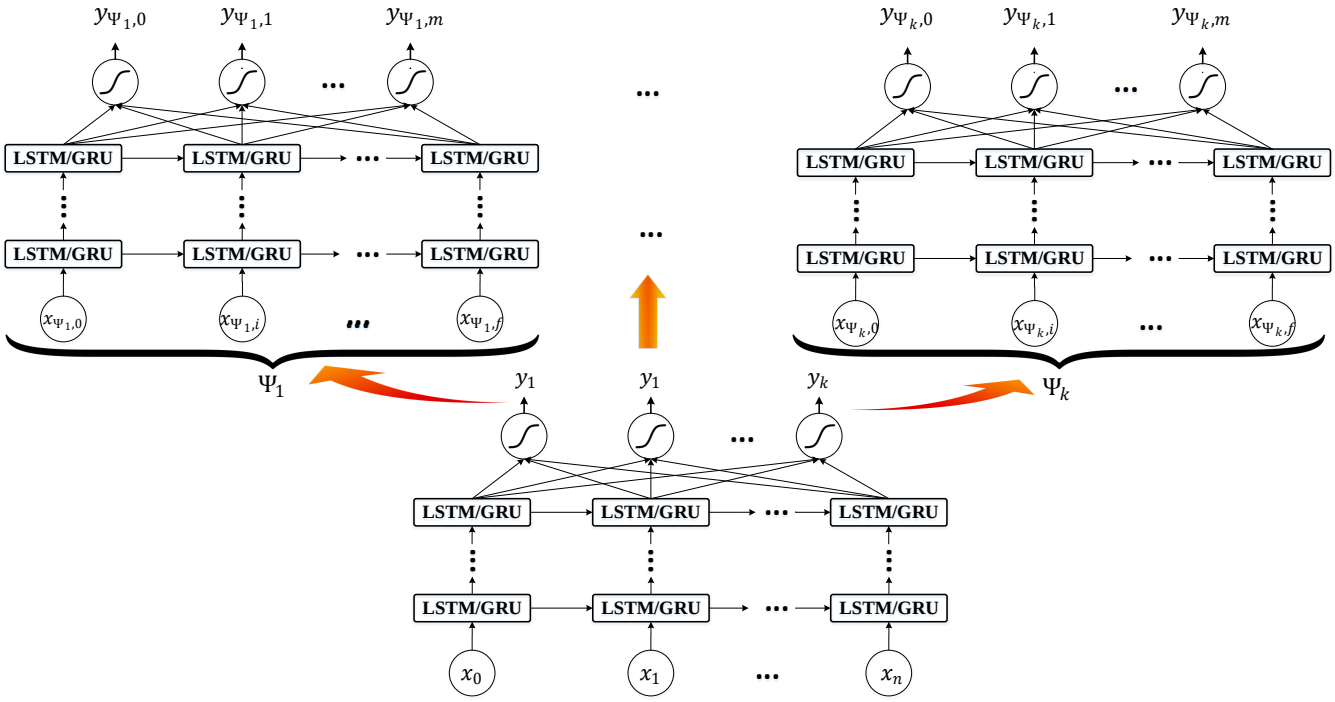

Fig. 2: HDLTex: Hierarchical Deep Learning for Text Classification. This is our structure of recurrent neural networks (RNN) for text classification. The left figure is the parent level of our text leaning model. The right figure depicts child-level learning models defined by Ψ_i as input documents in the parent levels.

This paper extends this baseline architecture to allow hierarchical classification. Figure 1 shows this new architecture. The DNN for the first level of classification (on the left side in Figure 1) is the same as the baseline DNN. The second level classification in the hierarchy consists of a DNN trained for the domain output in the first hierarchical level. Each second level in the DNN is connected to the output of the first level. For example, if the output of the first model is labeled *computer science* then the DNN in the next hierarchical level (e.g., Ψ_1 in Figure 1) is trained only with all *computer science* documents. So while the first hierarchical level DNN is trained with all documents, each DNN in the next level of the document hierarchy is trained only with the documents for the specified domain.

The DNNs in this study are trained with the standard back-propagation algorithm using both sigmoid (Equation 11) and ReLU (Equation 12) as activation functions. The output layer uses softmax (Equation 13).

$$f(x) = \frac{1}{1+e^{-x}} \in (0,1), \tag{11}$$

$$f(x) = \max(0,x), \tag{12}$$

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad \forall\, j \in \{1,\ldots,K\} \tag{13}$$

Given a set of example pairs $(x, y), x \in X, y \in Y$ the goal is to learn from the input and target spaces using hidden layers. In text classification, the input is generated by vectorization of text (see Section IV).

B. Recurrent Neural Networks (RNN)

The second deep learning neural network architecture we use is RNN. In RNN the output from a layer of nodes can reenter as input to that layer. This approach has advantages for text processing [38]. The general RNN formulation is given in Equation 14 where x_t is the state at time t and $\boldsymbol{u_t}$ refers to the input at step t.

$$x_t = F(x_{t-1}, \boldsymbol{u_t}, \theta) \tag{14}$$

We use weights to reformulate Equation 14 as shown in Equation 15 below:

$$x_t = \mathbf{W_{rec}}\sigma(x_{t-1}) + \mathbf{W_{in}}\mathbf{u_t} + \mathbf{b}. \tag{15}$$

In Equation 15, $\mathbf{W_{rec}}$ is the recurrent matrix weight, $\mathbf{W_{in}}$ are the input weights, $\mathbf{b}$ is the bias, and σ is an element-wise function. Again we have modified the basic architecture for use in hierarchical classification. Figure 2 shows this extended RNN architecture.

Several problems (e.g., vanishing and exploding gradients) arise in RNNs when the error of the gradient descent algorithm is back-propagated through the network [39]. To deal with these problems, long short-term memory (LSTM) is a special type of RNN that preserves long-term dependencies in a more effective way compared with the basic RNN. This

is particularly effective at mitigating the vanishing gradient problem [40].

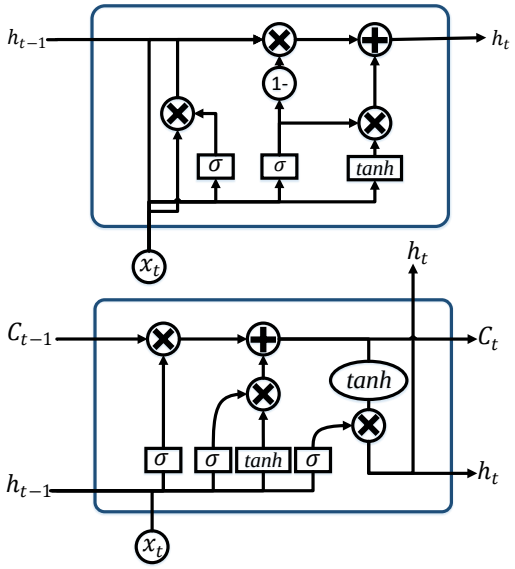

Fig. 3: The top sub-figure is a cell of GRU, and the bottom Figure is a cell of LSTM.

Figure 3 shows the basic cell of an LSTM model. Although LSTM has a chain-like structure similar to RNN, LSTM uses multiple gates to regulate the amount of information allowed into each node state. A step-by-step explanation the LSTM cell and its gates is provided below:

1) Input Gate:

$$i_t = \sigma(W_i[x_t, h_{t-1}] + b_i), \tag{16}$$

2) Candid Memory Cell Value:

$$\tilde{C}_t = \tanh(W_c[x_t, h_{t-1}] + b_c), \tag{17}$$

3) Forget Gate Activation:

$$f_t = \sigma(W_f[x_t, h_{t-1}] + b_f), \tag{18}$$

4) New Memory Cell Value:

$$C_t = i_t * \tilde{C}_t + f_t C_{t-1}, \tag{19}$$

5) Output Gate Values:

$$\begin{aligned} o_t &= \sigma(W_o[x_t, h_{t-1}] + b_o), \\ h_t &= o_t \tanh(C_t), \end{aligned} \tag{20}$$

In the above description, b is a bias vector, W is a weight matrix, and x_t is the input to the memory cell at time t. The i, c, f and o indices refer to input, cell memory, forget and output gates, respectively. Figure 3 shows the structure of these gates with a graphical representation.

An RNN can be biased when later words are more influential than the earlier ones. To overcome this bias convolutional neural network (CNN) models (discussed in Section V-C) include a max-pooling layer to determine discriminative phrases in text [41]. A gated recurrent unit (GRU) is a gating mechanism for RNNs that was introduced in 2014 [42]. GRU is a simplified variant of the LSTM architecture, but there are differences as follows: GRUs contain two gates, they do not possess internal memory (the C_{t-1} in Figure 3), and a second non-linearity is not applied ($tanh$ in Figure 3).

C. Convolutional Neural Networks (CNN)

The final deep learning approach we developed for hierarchical document classification is the convolutional neural network (CNN). Although originally built for image processing, as discussed in Section II, CNNs have also been effectively used for text classification [15]. The basic convolutional layer in a CNN connects to a small subset of the inputs usually of size 3×3. Similarly the next convolutional layer connects to only a subset of its preceding layer. In this way these convolution layers, called feature maps, can be stacked to provide multiple filters on the input. To reduce computational complexity, CNNs use pooling to reduce the size of the output from one stack of layers to the next in the network. Different pooling techniques are used to reduce outputs while preserving important features [43]. The most common pooling method is max-pooling where the maximum element is selected in the pooling window. In order to feed the pooled output from stacked featured maps to the next layer, the maps are flattened into one column. The final layers in a CNN are typically fully connected. In general during the back-propagation step of a CNN not only the weights are adjusted but also the feature detector filters. A potential problem of CNNs used for text is the number of channels or size of the feature space. This might be very large (e.g., 50K words) for text, but for images this is less of a problem (e.g., only 3 channels of RGB) [14].

D. Hierarchical Deep Learning

The primary contribution of this research is hierarchical classification of documents. A traditional multi-class classification technique can work well for a limited number classes, but performance drops with increasing number of classes, as is present in hierarchically organized documents. In our hierarchical deep learning model we solve this problem by creating architectures that specialize deep learning approaches for their level of the document hierarchy (e.g., see Figure 1). The structure of our Hierarchical Deep Learning for Text (HDLTex) architecture for each deep learning model is as follows:

DNN: 8 hidden layers with 1024 cells in each hidden layer.

RNN: GRU and LSTM are used in this implementation, 100 cells with GRU with two hidden layers.

CNN: Filter sizes of $\{3, 4, 5, 6, 7\}$ and max-pool of 5, layer sizes of $\{128, 128, 128\}$ with max pooling of $\{5, 5, 35\}$, the CNN contains 8 hidden layers.

All models used the following parameters: *Batch Size = 128*, *learning parameters* = 0.001, β_1*=0.9*, β_2*=0.999*, $\epsilon = 1e^{08}$, $decay = 0.0$, *Dropout=0.5 (DNN) and Dropout=0.25 (CNN and RNN).*

E. Evaluation

We used the following cost function for the deep learning models:

$$Acc(X) = \sum_{\varrho} \left[\frac{Acc(X_{\Psi_\varrho})}{k_\varrho - 1} \sum_{\Psi \in \{\Psi_1, .. \Psi_k\}} Acc(X_{\Psi_i}).n_{\Psi_k} \right] \quad (21)$$

where ϱ is the number of levels, k indicates number of classes for each level, and Ψ refers to the number of classes in the child's level of the hierarchical model.

F. Optimization

We used two types of stochastic gradient optimization for the deep learning models in this paper: RMSProp and Adam. These are described below.

RMSProp Optimizer: The basic stochastic gradient descent (SGD) is shown below:

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta, x_i, y_i) \quad (22)$$

$$\theta \leftarrow \theta - \left(\gamma\theta + \alpha \nabla_\theta J(\theta, x_i, y_i)\right) \quad (23)$$

For these equations, θ is the learning parameter, α is the learning rate, and $J(\theta, x_i, y_i)$ is the objective or cost function. The history of updates is defined by $\gamma \in (0, 1)$. To update parameters, SGD uses a momentum term on a rescaled gradient, which is shown in Equation (23). This approach to the optimization does not perform bias correction, which is a problem for a sparse gradient.

Adam Optimizer: Adam is another stochastic gradient optimizer, which averages over only the first two moments of the gradient, v and m, as shown below:

$$\theta \leftarrow \theta - \frac{\alpha}{\sqrt{\hat{v}} + \epsilon} \hat{m} \quad (24)$$

where

$$g_{i,t} = \nabla_\theta J(\theta_i, x_i, y_i) \quad (25)$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_{i,t} \quad (26)$$

$$m_t = \beta_2 v_{t-1} + (1 - \beta_2) g_{i,t}^2 \quad (27)$$

In these equations, m_t and v_t are the first and second moments, respectively. Both are estimated as $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ and $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$. This approach can handle the non-stationarity of the objective function as can RMSProp, but Adam can also overcome the sparse gradient issue that is a drawback in RMSProp [44].

VI. Results

A. Data

Our document collection had 134 labels as shown in Table I.[2] The target value has two levels, $k_0 \in \{1, .., 7\}$ which are $k_0 \in \{$ Computer Science, Electrical Engineering, Psychology, Mechanical Engineering, Civil Engineering, Medical Science, biochemistry$\}$ and children levels of the labels, k_ϱ, which contain $\{17, 16, 19, 9, 11, 53, 9\}$ specific topics belonging to k_0, respectively. To train and test the baseline methods described in Section III and the new hierarchical document classification methods described in Section V, we collected data and meta-data on $46,985$ published papers available from the *Web Of Science* [45], [46]. To automate collection we used Selenium [47] with *ChoromeDriver* [48] for the Chrome web browser. To extract the data from the site we used *Beautiful Soup* [49]. We specifically extracted the abstract, domain, and keywords of this set of published papers. The text in the abstract is the input for classification while the domain name provides the label for the top level of the hierarchy. The keywords provide the descriptors for the next level in the classification hierarchy. Table I shows statistics for this collection. For example, Medical Sciences is one of the top-level domain classifications and there are 53 sub-classifications within this domain. There are also over 14k articles or documents within the domain of health sciences in this data set.

TABLE I: Details of the document set used in this paper.

Domain	Number of Document	Number of Area
Biochemistry	5,687	9
Civil Engineering	4,237	11
Computer Science	6,514	17
Electrical Engineering	5,483	16
Medical Sciences	14,625	53
Mechanical Engineering	3,297	9
Psychology	7,142	19
Total	**46,985**	**134**

We divided the data set into three parts as shown in Table II. Data set $WOS - 46985$ is the full data set with 46,985 documents, and data sets $WOS-11967$ and $WOS-5736$ are subsets of this full data set with the number of training and testing documents shown as well as the number of labels or classes in each of the two levels. For dataset $WOS - 11967$, each of the seven level-1 classes has five sub-classes. For data set $WOS - 5736$, two of the three higher-level classes have four sub-classes and the last high-level class has three sub-classes. We removed all special characters from all three data sets before training and testing.

TABLE II: Details of three data sets used in this paper.

Data Set	Training	Testing	Level 1	Level 2
WOS-11967	8018	3949	7	35
WOS-46985	31479	15506	7	134
WOS-5736	4588	1148	3	11

B. Hardware and Implementation

The following results were obtained using a combination of central processing units (CPUs) and graphical processing units (GPUs). The processing was done on a $Xeon\ E5 - 2640\ (2.6GHz)$ with 32 cores and $64GB$ memory, and the GPU cards were $Nvidia\ Quadro\ K620$ and $Nvidia\ Tesla\ K20c$. We implemented our approaches in Python using the Compute Unified Device Architecture (CUDA), which is a parallel computing platform and

[2]WOS dataset is shared at http://archive.ics.uci.edu/index.php

TABLE III: HDLTex and Baseline Accuracy of three WOS datasets

	WOS-11967			WOS-46985			WOS-5736		
	Methods		Accuracy	Methods		Accuracy	Methods		Accuracy
Baseline	DNN		80.02	DNN		66.95	DNN		86.15
	CNN (Yang el. et. 2016)		83.29	CNN (Yang el. et. 2016)		70.46	CNN (Yang el. et. 20016)		88.68
	RNN (Yang el. et. 2016)		83.96	RNN (Yang el. et. 2016)		72.12	RNN (Yang el. et. 2016)		89.46
	NBC		68.8	NBC		46.2	NBC		78.14
	SVM (Zhang el. et. 2008)		80.65	SVM (Zhang el. et. 2008)		67.56	SVM (Zhang el. et. 2008)		85.54
	SVM (Chen el et. 2016)		83.16	SVM (Chen el et. 2016)		70.22	SVM (Chen el et. 2016)		88.24
	Stacking SVM		79.45	Stacking SVM		71.81	Stacking SVM		85.68
HDLTex	DNN	DNN	83.73	DNN	DNN	70.10	DNN	DNN	88.37
	91.43	91.58		87.31	80.29		97.97	90.21	
	DNN	CNN	83.32	DNN	CNN	71.90	DNN	CNN	90.47
	91.43	91.12		87.31	82.35		97.97	92.34	
	DNN	RNN	81.58	DNN	RNN	73.92	DNN	RNN	88.42
	91.43	89.23		87.31	84.66		97.97	90.25	
	CNN	DNN	85.65	CNN	DNN	71.20	CNN	DNN	88.83
	93.52	91.58		88.67	80.29		98.47	90.21	
	CNN	CNN	85.23	CNN	CNN	73.02	CNN	CNN	**90.93**
	93.52	91.12		88.67	82.35		98.47	92.34	
	CNN	RNN	83.45	CNN	RNN	75.07	CNN	RNN	88.87
	93.52	89.23		88.67	84.66		98.47	90.25	
	RNN	DNN	**86.07**	RNN	DNN	72.62	RNN	DNN	88.25
	93.98	91.58		90.45	80.29		97.82	90.21	
	RNN	CNN	85.63	RNN	CNN	74.46	RNN	CNN	90.33
	93.98	91.12		90.45	82.35		97.82	92.34	
	RNN	RNN	83.85	RNN	RNN	**76.58**	RNN	RNN	88.28
	93.98	89.23		90.45	84.66		97.82	90.25	

Application Programming Interface (API) model created by $Nvidia$. We also used Keras and TensorFlow libraries for creating the neural networks [50], [51].

C. Empirical Results

Table III shows the results from our experiments. The baseline tests compare three conventional document classification approaches (naïve Bayes and two versions of SVM) and stacking SVM with three deep learning approaches (DNN, RNN, and CNN). In this set of tests the RNN outperforms the others for all three WOS data sets. CNN performs second-best for three data sets. SVM with term weighting [22] is third for the first two sets while the multi-word approach of [23] is in third place for the third data set. The third data set is the smallest of the three and has the fewest labels so the differences among the three best performers are not large. These results show that overall performance improvement for general document classification is obtainable with deep learning approaches compared to traditional methods. Overall, naïve Bayes does much worse than the other methods throughout these tests. As for the tests of classifying these documents within a hierarchy, the HDLTex approaches with stacked, deep learning architectures clearly provide superior performance. For data set $WOS-11967$, the best accuracy is obtained by the combination RNN for the first level of classification and DNN for the second level. This gives accuracies of 94% for the first level, 92% for the second level and 86% overall. This is significantly better than all of the others except for the combination of CNN and DNN. For data set $WOS-46985$ the best scores are again achieved by RNN for level one but this time with RNN for level 2. The closest scores to this are obtained by CNN and RNN in levels 1 and 2, respectively. Finally the simpler data set $WOS-5736$ has a winner in CNN at level 1 and CNN at level 2, but there is little difference between these scores and those obtained by two other HDLTex architectures: DNN with CNN and RNN with CNN.

VII. Conclusions and Future Work

Document classification is an important problem to address, given the growing size of scientific literature and other document sets. When documents are organized hierarchically, multi-class approaches are difficult to apply using traditional supervised learning methods. This paper introduces a new approach to hierarchical document classification, HDLTex, that combines multiple deep learning approaches to produce hierarchical classifications. Testing on a data set of documents obtained from the Web of Science shows that combinations of RNN at the higher level and DNN or CNN at the lower level produced accuracies consistently higher than those obtainable by conventional approaches using naïve Bayes or SVM. These results show that deep learning methods can provide improvements for document classification and that they provide flexibility to classify documents within a hierarchy. Hence, they provide extensions over current methods for document classification that only consider the multi-class problem.

The methods described here can improved in multiple ways. Additional training and testing with other hierarchically structured document data sets will continue to identify architectures that work best for these problems. Also, it is possible to extend the hierarchy to more than two levels to capture more of the complexity in the hierarchical classification. For example, if keywords are treated as ordered then the hierarchy continues down multiple levels. HDLTex can also be applied to unlabeled documents, such as those found in news or other media outlets.

Scoring here could be performed on small sets using human judges.

REFERENCES

[1] M. Ware and M. Mabe, "The stm report: An overview of scientific and scholarly journal publishing," 2015.

[2] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1, no. 1.

[3] J. C. French, D. E. Brown, and N.-H. Kim, "A classification approach to boolean query reformulation," *JASIS*, vol. 48, no. 8, pp. 694–706, 1997.

[4] K. Kowsari, M. Yammahi, N. Bari, R. Vichr, F. Alsaby, and S. Y. Berkovich, "Construction of fuzzyfind dictionary using golay coding transformation for searching applications," *International Journal of Advanced Computer Science and Applications(IJACSA)*, vol. 6, pp. 81–87, 2015.

[5] T. Joachims, "Transductive inference for text classification using support vector machines," in *ICML*, vol. 99, 1999, pp. 200–209.

[6] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of machine learning research*, vol. 2, no. Nov, pp. 45–66, 2001.

[7] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems," *J. Mach. Learn. Res*, vol. 15, no. 1, pp. 3133–3181, 2014.

[8] C. S. Leslie, E. Eskin, and W. S. Noble, "The spectrum kernel: A string kernel for svm protein classification." in *Pacific symposium on biocomputing*, vol. 7, no. 7, 2002, pp. 566–575.

[9] E. Eskin, J. Weston, W. S. Noble, and C. S. Leslie, "Mismatch string kernels for svm protein classification," in *Advances in neural information processing systems*, 2003, pp. 1441–1448.

[10] A. McCallum, K. Nigam *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752. Citeseer, 1998, pp. 41–48.

[11] S.-B. Kim, K.-S. Han, H.-C. Rim, and S. H. Myaeng, "Some effective techniques for naive bayes text classification," *IEEE transactions on knowledge and data engineering*, vol. 18, no. 11, pp. 1457–1466, 2006.

[12] N. Van Linh, N. K. Anh, K. Than, and C. N. Dang, "An effective and interpretable method for document classification," *Knowledge and Information Systems*, vol. 50, no. 3, pp. 763–793, 2017.

[13] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[14] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *arXiv preprint arXiv:1412.1058*, 2014.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[17] J. Y. Lee and F. Dernoncourt, "Sequential short-text classification with recurrent and convolutional neural networks," *arXiv preprint arXiv:1603.03827*, 2016.

[18] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[19] L. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[20] R. Salakhutdinov, J. B. Tenenbaum, and A. Torralba, "Learning with hierarchical-deep models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1958–1971, 2013.

[21] G. B. Huang, H. Lee, and E. Learned-Miller, "Learning hierarchical representations for face verification with convolutional deep belief networks," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 2518–2525.

[22] K. Chen, Z. Zhang, J. Long, and H. Zhang, "Turning from tf-idf to tf-igm for term weighting in text classification," *Expert Systems with Applications*, vol. 66, pp. 245–260, 2016.

[23] W. Zhang, T. Yoshida, and X. Tang, "Text classification based on multi-word with support vector machine," *Knowledge-Based Systems*, vol. 21, no. 8, pp. 879–886, 2008.

[24] V. Vapnik and A. Y. Chervonenkis, "A class of algorithms for pattern recognition learning," *Avtomat. i Telemekh*, vol. 25, no. 6, pp. 937–945, 1964.

[25] A. Y. Chervonenkis, "Early history of support vector machines," in *Empirical Inference*. Springer, 2013, pp. 13–20.

[26] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[27] R. Singh, A. Sekhon, K. Kowsari, J. Lanchantin, B. Wang, and Y. Qi, "Gakco: a fast gapped k-mer string kernel using counting," 2017.

[28] J. Weston and C. Watkins, "Multi-class support vector machines," Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, May, Tech. Rep., 1998.

[29] A. Sun and E.-P. Lim, "Hierarchical text classification and evaluation," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001, pp. 521–528.

[30] F. Sebastiani, "Machine learning in automated text categorization," *ACM computing surveys (CSUR)*, vol. 34, no. 1, pp. 1–47, 2002.

[31] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *European conference on machine learning*. Springer, 1998, pp. 4–15.

[32] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *EMNLP*, vol. 14, 2014, pp. 1532–1543.

[33] W. B. Cavnar, J. M. Trenkle *et al.*, "N-gram-based text categorization," *Ann Arbor MI*, vol. 48113, no. 2, pp. 161–175, 1994.

[34] D. Glickman, "Comparing input words to a word dictionary for correct spelling," Feb. 5 1985, uS Patent 4,498,148.

[35] V. Kešelj, F. Peng, N. Cercone, and C. Thomas, "N-gram-based author profiles for authorship attribution," in *Proceedings of the conference pacific association for computational linguistics, PACLING*, vol. 3, 2003, pp. 255–264.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[37] J. Lanchantin, R. Singh, B. Wang, and Y. Qi, "Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks," *arXiv preprint arXiv:1608.03644*, 2016.

[38] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," http://karpathy.github.io/2015/05/21/rnn-effectiveness/, May 2015.

[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[40] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." *ICML (3)*, vol. 28, pp. 1310–1318, 2013.

[41] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification." in *AAAI*, vol. 333, 2015, pp. 2267–2273.

[42] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[43] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," *Artificial Neural Networks–ICANN 2010*, pp. 92–101, 2010.

[44] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] O. Poungpair, A. Pootong, P. Tongtawe, T. Songserm, P. Tapchaisri, and W. Chaicumpa, "Web of science®," *COMMUNICATIONS*, 2013.

[46] T. Reuters, "Web of science." 2012.

[47] S. Munzert, C. Rubba, P. Meißner, and D. Nyhuis, *Automated data collection with R: A practical guide to web scraping and text mining*. John Wiley & Sons, 2014.

[48] U. Gundecha, *Selenium Testing Tools Cookbook*. Packt Publishing Ltd, 2015.

[49] L. Richardson, "Beautiful soup," https://www.crummy.com/software/BeautifulSoup/, May 2017.

[50] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[51] F. Chollet *et al.*, "Keras: Deep learning library for theano and tensorflow," *https://keras.io/*, 2015.