Multi-task Neural Networks for Hierarchical Text Classification

Yunfei Luo
UMass Amherst
yunfeiluo@umass.edu

Yuyang Liu
UMass Amherst
yuyliu@umass.edu

Rukai Cai
UMass Amherst
rukaicai@umass.edu

Abstract

Text classification plays a significant role in real-world applications, including but not limited to data management, recommendation systems, and search engines, and serves as a commonplace in the field of Natural Language Processing. In the real world, it is not always the case that an application of text categorization is asked to assign exactly one category to each piece of texts to be classified; instead, such application is realized in a hierarchical manner, which is why this project focuses on hierarchical as opposed to single-label text classification. To approach this task, we apply multi-task learning, which has been proved more flexible than the single-task learning in the literature, and neural networks, which have been shown to be powerful in various fields. We provide a general platform for hierarchical text classification with which we conducted experiments on several deep neural network structures and achieved considerable performance, along with a performance comparison between these networks on three data sets: Amazon product reviews, DBPedia, and Coronavirus tweets.

1 Introduction

Within the billions of applications people nowadays are using every day, mobile or tabletop, offline or online, a place where classification evinces its necessity is in the recommendation systems, of which the versatility has been proved by its capability of serving for a wide range of functionalities. In applications designed for entertainment, for instance, to align with the interests of the users, media items such as articles, musics, pictures, videos, and so on that are related to the users' history are often prompted to them. Under this setting, having labels of classes for these media items can undoubtedly help with the search engine, since, by specifying a classification method on the whole data set, the application will have a better understanding on what to search for within which domain. Meanwhile, the users will have an easier way to reach the items they are targeting at in oceans of information, further facilitating the procedure for the application to learn the users' preferences.

However, there is a myriad of new articles, musics, pictures, and videos uploaded to the internet every day, making it inefficient and almost impossible for people to label every single media item by hand. It is, therefore, crucial to have a model that can perform automatic classification for the purpose of convenience and effectiveness in data management.

This project cuts into the field of automatic classification and follows previous works within this field from the aspect of text classification, not only because text classification is a commonplace in the field of Natural Language Processing, but also because classification problems of the other types of media items, especially those mentioned above, can mostly be somehow converted to a text classification problem.

While talking about classification, what people usually have in mind is a one-to-one correspondence between a certain set of elements to be classified and another certain set of elements to be used as labels. "Sex assigned at birth" is one of the most salient examples of this like, where newborn babies are each assigned a label of male or female, or maybe others depending on the country and its gender policies. The key point here is that, in most cases, it is not allowed for someone to be assigned both male and female, and that is the key feature of single-label classification. Traditional classification methods such as those that involve Naive Bayes are designed typically to overcome classification tasks that share the above features.

Notwithstanding, in the real world, it is not always the case that an application of text categorization is asked to assign exactly one category or level of category to each piece of texts to be classified. Instead, these applications are, more often than not, realized in a hierarchical manner. For example, a piece of news concerning the results of a boxing tournament may be classified so that it belongs to a class labeled, say, “Boxing,” which, apparently, should not keep it from being classified also as “Sports” news. That is why this project focuses on hierarchical as opposed to single-label text classification. Our goal is to build a model that can have greater theoretical significance in the real world.

To approach a hierarchical text classification task or maybe to approach any task within the field of computer science in general, it is common that people build and train only one model. It is reasonable in the sense that people are more interested in finding out the ”best” strategy with respect to solving a problem, but this single-task pattern has been proved not flexible enough when scaling on the number of tasks by Caruana (1997), who proposed a model where certain parts of it are trained for several tasks but are combined into a single block. Specifically, the world Caruana (1997) is picturing is one where the models targeting at different tasks, somehow, share some of their parameters. Depending on this proposed structure, a fair amount of research has been conducted. More pieces of empirical evidence, such as the branching works by Guo et al. (2020) and the personalized model for student stress prediction by Shaw et al. (2019), indicate that similar tasks that are trained simultaneously could, in fact, be benefited from each other. The improvement in the performance, the acceleration in the time for convergence, and the better applicability of their models are all significant indications. These pieces of evidence establish why an approach to hierarchical text classification featuring neural networks that apply multi-task learning is not a redundant move and may lead us to broader and faster applications.

In a nutshell, in this project, we are applying multi-task learning, which has been proved more flexible than the single-task learning in the literature, and neural networks, which have been shown to be powerful in various fields, to perform hierarchical text classification.

2 Review of Literature

Cerri et al. (2014) demonstrated the applicability of neural networks on hierarchical text classification and proposed a local-based approach, where the prediction scores of the model for classifying current level of labels are used as the input to the model for classifying the next level of labels. They showed that this approach could achieve better performance than traditional methods such as decision tree, but a drawback of this method is that several neural networks are built separately, and the networks responsible for lower levels of labels do not have the information of the original input. Furthermore, because the authors were focusing more on revealing the improvement over the traditional methods by applying standard neural networks, a large space of potential improvements are left for future researching works.

Zhang et al. (2016) proposed a novel character-level as opposed to word-level One-Hot word embedding method that achieved considerable performance on the classification task. A shining point of this work worthy to highlight is that this method gives zero out-of-vocabulary words during the testing. The disadvantages, though, are that the big length of input sequences and the small size of the vocabulary make the model cost more time and iterations to converge. Also, it is hard for the model to be benefited from transfer learning as models that feature other types of embeddings can be, such as the popular Word2Vec proposed by Mikolov et al. (2013) and GloVe proposed by Pennington et al. (2014).

Conneau et al. (2017) proposed a deep convolutional neural network structure for text classification. This model adopted the ideas from ResNet He et al. (2015) and from VGG Simonyan and Zisserman (2015). A small size of kernels for convolutional filters is used, and residual connections are added to address the degradation problem. With a similar motivation, Kim et al. (2017) proposed a deep recurrent neural network structure, where the residual connections are added between the stacked LSTM layers. These models with a deep structure are empirically remarkable for feature extraction in NLP tasks.

Lu et al. (2018) introduced a multi-task learning approach to the sentiment classification for texts of different levels of positiveness and negativeness. They showed that multi-task neural networks with variational auto-encoder can effectively improve the classification accuracy by some other relative tasks. Although this work has only conducted experiments on the sentiment classification task, the authors introduced the multi-task structure as an approach worthy to try. Their work has also guided the future research in applying the multi-task model to the scenarios with tens or even hundreds of categorical classes.

Peng et al. (2018) indicated the applicability of convolutional neural network on hierarchical text classification. They introduced the hierarchical dependencies between the labels to the classifier with recursive regularization, previously proposed by Gopal and Yang (2013). The only drawback of this work, in our opinion, is that it is not clear how the proposed model can deal with cases where the hierarchical dependencies are not given, which are more generally the case in the real world.

3 Method

3.1 Multi-Task Neural Networks

The fundamental structure of our proposed model is guided by Lu et al. (2018) and an overview of this model is shown in Figure 1. Let N be the number of training data, L be the number of tokens in each sample, V be the vocabulary size, and D be the embedding dimension. Before we feed data into the model, each token should be transformed to its index in the vocabulary. An $N \times L$ matrix is then fed to the first embedded layer, which will convert each input index to its corresponding vector. As a result, an $N \times L \times D$ matrix will be fed to the next layers, which are our feature extractor. See more discussions about the feature extractors to be applied and experimented on in the later subsections.

According to the multi-task learning approach, our classifier, which is the processing layers of feature extractors, consists of a shared hidden linear layer and a task-specific hidden layer for each task. In our case, each task corresponds to a classification task at a different level. We follow the fully connected layer structure proposed by Zhang et al. (2016) and Conneau et al. (2017), who applied 2048 units in each hidden layer.

The loss function for the multi-task network is the weighted sum of loss from different tasks. In our case, the loss function for each task is a softmax operation. Let L_i and C_i be the loss function and the the number of classes, respectively, of a task indexed i, and W_i be the weight matrix in the output layer with a shape of $2048 \times C_i$. Imagine that we have a sample X_j with a ground-truth label $y_{i,\,j}$, then the softmax loss is defined as:

$$L_{i,\,j}(X,\,W_i,\,C_i) = -\log \frac{\exp(W_{i,\,y_{i,\,j}} X_j)}{\sum_{k=0}^{C_i} \exp(W_{i,k} X_j)}$$

Let t be the number of task. Then, our final loss for sample X_j is:

$$L_j(X) = \sum_{i=0}^{t} \lambda_i L_{i,\,j}(X,\,W_i,\,C_i)$$

Where λ_i is the weight assigned for the task indexed i. Under the scope of this project, we treat those coefficients as hyper-parameters. That is to say, we tend to assign larger values to tasks with more classes.

3.1.1 Fully Connected Neural Network

According to the structure of a standard shallow neural network, a fully connected layer in our case consists of a single hidden layer of size H. The 3D input X with shape $N \times L \times D$ is first flattened to a 2D matrix with shape $N \times (L \times D)$. Then, the output X' of the hidden layer is:

$$X' = \max(0,\,WX + b)$$

Where W is an $(L \times D) \times H$ 2D weight matrix that maps the input X to the hidden space, and b is the bias term, or the intercept in terms of linear regression. The $\max$ operation is the ReLU activation function. X' will have a shape of $N \times H$, and it will be passed to the Multi-task classifier to yield final predictions.

3.1.2 Shallow Convolutional Neural Network

1D convolutional neural network has been shown to be useful in the task of extracting features from texts. In our scenario, the initial channel is the embedded dimension, and the kernel size corresponds to the number of consecutive tokens. A stride of 1 is equivalent to extracting features from n-grams, where n is exactly the kernel size.

Our shallow convolutional neural network is shown in Figure 2. It consists of a convolution layer with a kernel size of 3, a stride of 1, and 64

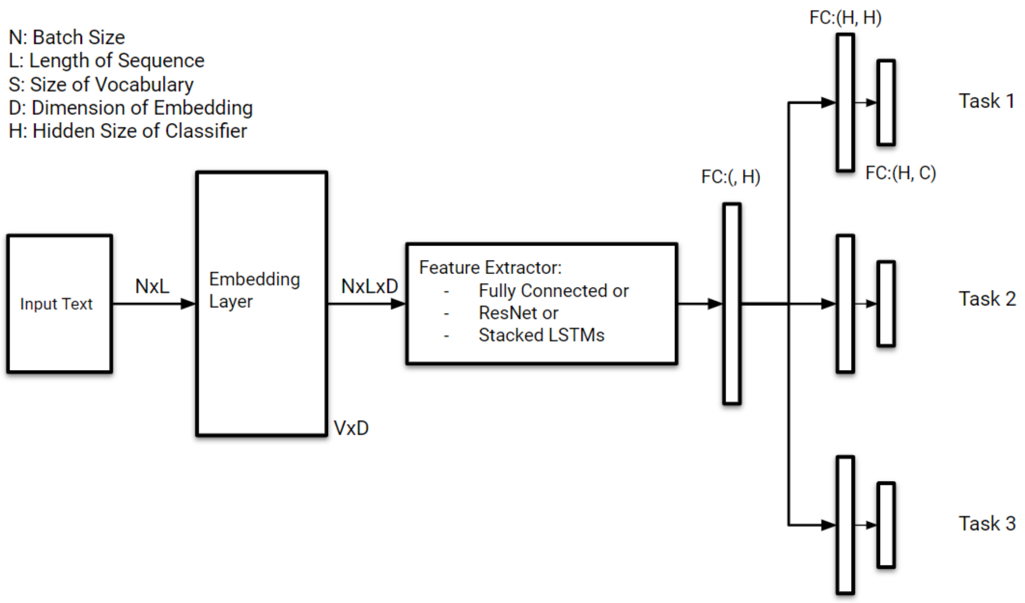

Figure 1: Model architecture.

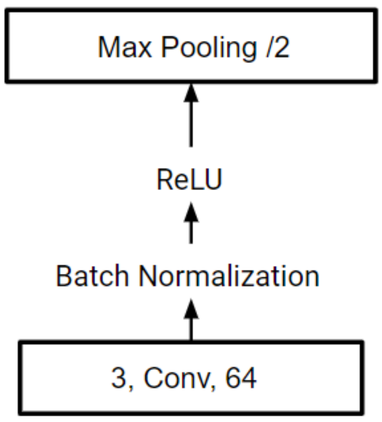

Figure 2: Shallow convolution blocks.

filters, followed by batch normalization, ReLU activation function, and a max pooling layer of kernel size 3 and stride 2. In the scenario of text processing, the pooling layer tends to be an operation that extracts the most representative features from the feature maps across the sequence.

3.1.3 Deep Convolutional Neural Network

We followed and implemented the deep neural network structure proposed by Conneau et al. (2017), as shown in Figure 3, and the pattern of each convolutional block within the deep neural network structure follows the design of residual networks by He et al. (2015). The residual connections are added to address the degradation problem. There are 17 convolutional layers followed by a pooling and fully connected layer in our feature extractor, which is similar to ResNet18. The only differences between the structure of deep networks Conneau et al. (2017) and the structure of the original ResNet He et al. (2015) are that the downsampling layers are replaced by pooling layers and that the average pooling layer is replaced by a k-max pooling layer.

3.1.4 Long Short-Term Memory (LSTM)

LSTM is a type of Recurrent Neural Network (RNN). In addition to the plain RNN, LSTM will keep one more state other than the hidden state, called cell state. Cell state provides the ability to remember some information in the contexts that are far away from the current time step. Each LSTM unit consist of input gate, output gate, and forget gate. These three gates control the flow of information into and out of the states.

3.1.5 Deep Residual LSTM

We followed and implemented the deep neural network structure proposed by Kim et al. (2017). Each LSTM layer takes the output from previous layers as input. The initial hidden and cell states are the last states from the previous layer,

Data sets	#S	#labels	#V	#OOV	#train	#val	#test	Seq Len
APR	40k	6, 50, 147	42k	21k	24k	8k	8k	96
DBPedia	338k	9, 70, 219	618k	196k	241k	36k	61k	160
CoVT	45k	3, 5	70k	31k	37k	4k	4k	32

Table 1: Statistics of data sets.

except for the first layer. Our implementation of this model consists of eight LSTM layers, where the residual connections are added every two layers on the middle six layers to address the degradation problem.

3.1.6 Batch Normalization

For the purpose of efficient convergence, we apply batch normalization Ioffe and Szegedy (2015) to every layer except for the embedding layer and for the final output layer. Batch normalization accelerates the convergence by removing the internal shift so that the model will tempt to adjust the parameters to fit batches with different distribution.

3.2 Word2Vec Embedding

Word2Vec Mikolov et al. (2013) is a predictive method that maps a token to a vector space. The method we use for pre-training is skip-gram, where the goal of the fit model is to predict the words around a given word. All our experiments concerning Word2Vec embeddings use 32 dimensions.

4 Data sets

As stated in the introduction, the data sets we are conducting experiments on are Amazon product reviews, DBPedia, and Coronavirus tweets.

The original samples in the data sets are not acceptably clean. We droped samples having too few characters. We also droped samples that belong to labels having too few samples. As for dealing with words in the texts, our pre-processing pipeline consists of tokenization and stop-words removal. For splitting the data that are not already split, we use 60% of the entire data set for training, 20% for validation, and 20% for testing.

Table 1 represents the number of samples, number of labels of each level, vocabulary size, number of Out-of-Vocabulary words, and the number of samples for training, validation, and testing. The i-th number in the #Labels column represents the number of labels at level i. "Seq Len" denotes the fixed length of each sample.

4.1 Amazon Product Reviews (APR)

There are 50k Amazon product reviews (APR) in the original data set Kashnitsky (2020), belong to one of the 6 L1 labels, 64 L2 labels, and 510 L3 labels. The higher the level, the more specific the labels. We first drop the sample with number of characters less than 32. Then we concatenate the 3 levels of labels together and drop the samples that belong to those combined labels with number of samples less than 64. After this washing process, we have around 40k number of samples left, along with 6 L1 labels, 50 L2 labels, and 147 L3 labels. More detail has been shown in Table 1.

Because the product reviews are usually informal sentences, the tokenizer we choose for this data set is the TweetTokenizer from NLTK Bird and Loper (2004).

4.2 DBPedia

The original data set Ofer (2019) has 338k samples overall, belonging to one of the 9 L1 labels, 70 L2 labels, and 219 L3 labels. Likewise, the higher the level, the more specific the labels. No data were dropped as this is a benchmark data set, and it represents a nice distribution.

The tokenizer chosen for this data set is the NLTK word tokenizer, preceded by punctuation removal and followed by stop words removal.

4.3 Coronavirus Tweets (CoVT)

The data set of Miglani (2020) originally has a training set of 41k samples set and a testing samples set of 4k samples. We split 4k samples from the original training sample set as validation sample set.

Unlike the other two data sets, this data set is annotated with sentimental labels with only two hierarchical layers having 3 and 5 labels each. The L1 labels are "positive," "neutral," and "negative," and the L2 ones are "extreme positive," "positive," "neutral," "negative," and "extreme negative." The average length for each sample is around 15. So, we picked 16 and 32 as our sequence lengths, and

	APR, 147 classes		DBPedia, 219 classes		CoVT, 5 classes	
Models	Single	Multi-task	Single	Multi-task	Single	Multi-task
FC-Net	17.03	**19.16**	83.17	**85.11**	35.91	**38.70**
ConvNet	25.72	**26.77**	90.57	**90.68**	37.78	**41.91**
LSTM	32.60	**41.53**	91.28	**92.26**	46.36	**53.55**

Table 2: Testing accuracies in percentage of all the models at finest level of labels with and without multi-task learning. “Single” denotes the vanilla model with single output head; “multi-task” denotes the multi-task model that we proposed with multi-head output heads; “FC-Net” denotes the fully connected neural network; “ConvNet” denotes the convolutional neural network. The number of classes is shown at the top of the table.

	APR			DBPedia			CoVT	
Models	L1.	L2.	L3.	L1.	L2.	L3.	L1.	L2.
Shallow FC-Net	59.02	31.07	19.16	96.71	91.13	85.11	59.03	38.70
Shallow ConvNet	70.52	43.04	26.77	97.52	93.86	90.68	61.29	38.52
Deep ConvNet	75.53	49.46	34.13	97.39	94.28	91.07	63.21	41.91
Shallow LSTM	75.79	51.95	**41.53**	**97.67**	**94.92**	**92.26**	**70.35**	**53.55**
Deep LSTM	**77.96**	**53.56**	41.12	96.72	93.71	90.39	59.89	39.57

Table 3: Testing performance of all the models at each level. “Li.” denotes the testing accuracy in percentage of the i-th level label. Naming of the models are same with that in Table 2. The number of classes at different levels of these data sets have been shown in Table 1.

the model performs better with 32, which becomes our final decision.

As a collection of tweets, this data set consists mostly of informal writings. We chose NLTK regular expression tokenizer and NLTK stopwords removal with no samples being dropped.

5 Results

In Table 2 are the results from our preliminary experiments. It provides the comparison of the performance achieved at a certain level of the hierarchical labels by using and not using multi-task structure with different neural networks as feature extractor.

As the preliminary results verify the effectiveness of the multi-task model, we conduct further experiments for exploring the possible improvements. In Table 3 are the complete experiment results. The best result in each column is boldfaced.

For all the training, we use mini-batch SGD optimizer with a momentum of 0.9. We use a batch size of 64 for Amazon product reviews and Coronavirus tweets, and of 128 for DBPedia. The initial learning rates are fixed to 1e-2. The models converge in around 30 epochs on Amazon product reviews and in around 15 epochs on DBPedia and Coronavirus tweets. We use the overall accuracy as our evaluation metric.

6 Analysis and Discussion

Our experiments have shown that using both the convolutional neural network and LSTM for feature extraction can perform better in most cases than using the baseline model, namely the fully connected neural network. From Figure 4, we can observe that LSTM networks converge with lowest error rate, while stacked LSTM networks did not seem to have brought much improvement. On DBpedia, given sufficient training samples, both shallow and deep convolutional neural networks converge with a very close error rate. Contrastingly, on the Amazon product views, where some labels have a poor number of samples, whereas deep convolutional neural networks can still achieve considerable performance, the shallow ones get stuck at some point and converge with a relatively high error rate. Fully connected neural networks, as the baseline model, converge with poor performance on both data sets and are in no competetion with the other neural networks mentioned thus far.

Now, let us look into feature extraction. When the model’s feature extractor is set to be a fully connected network, the features it extracts will be

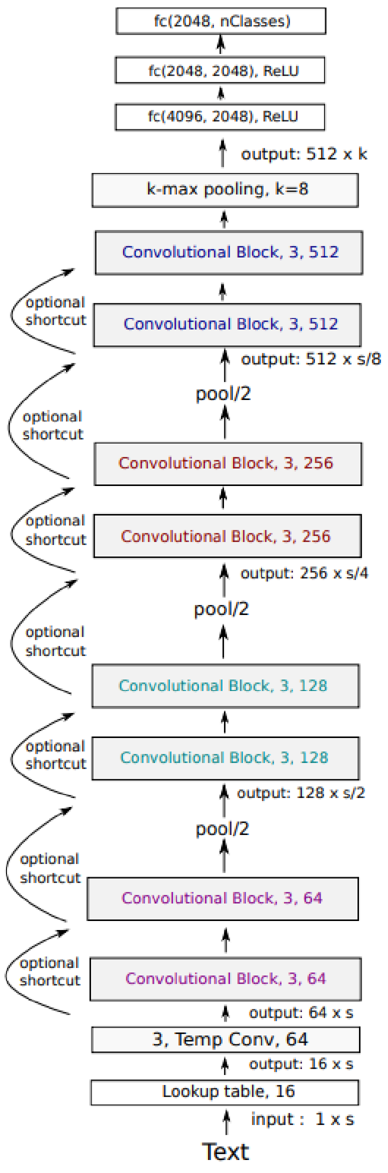

Figure 3: VDCNN by Conneau et al. (2017).

those that are global, meaning that all the tokens in a sentence to be classified will be evaluated simultaneously before being fed to another layer. In contrast, convolutional neural networks extract local features and evaluate the tokens in a similar manner to that of n-grams. Although there are no solid observations in the literature about whether global or local features are more efficient, in our case, the empirical results show that local features are better. The reason for this could be that the tokens have relatively stronger relations with surrounded tokens than with words that are far away from them. What is more, languages in general can be fickle, and some of them can arrange words fairly flexibly. By extracting local features, wherever the sets of tokens that are most informative in the sentence are, they can still be caught by the convolutional filters. Fully connected layers, on the other hand, will treat all the tokens equally and reduce the generalization of the model for the variety of sentences.

Let us be frank: Texts are complex. Although convolutional neural networks seem effective for extracting useful features from texts, it is undeniable that words in the sentence *can* have strong dependencies with words that are far from them. Convolutional neural networks cannot extract information resembling this kind of dependencies, but long short-term memory network (LSTM) can. LSTM, as one of the most popular recurrent neural networks, treats texts as a series of data and aims to solve the problem for missing information of long dependencies belonging. The recurrent layer will process the tokens in a sentence one by one and the hidden states of each step depends on its previous states. Such a structure enables the model to extract the features of the overall pattern of the sentences. It should be recalled at this point that, since the tokens in each sentence is supposedly coherent, local features cannot represent the entire pattern, and that although fully connected neural networks do consider the sentence as a whole, it cannot extract information of the flows within tokens. This could possibly explain why LSTM indicates its robustness over other models in our experiments.

Then, let us consider the number of parameters of the models. Table 4 shows the number of parameters for each model. The fully connected neural network, as the baseline model, has the most number of parameters, whereas LSTM has a significantly smaller number of parameters.

Models	#params
Shallow FC-Net	$\sim$43M
Shallow ConvNet	$\sim$24M
Deep ConvNet	$\sim$25M
Shallow LSTM	$\sim$0.3M
Deep LSTM	$\sim$0.6M

Table 4: Number of parameters.

As a summarization, a key question concerning all the models on which we have conducted experiments can be raised: What will be the trade-off if these models are being deployed?

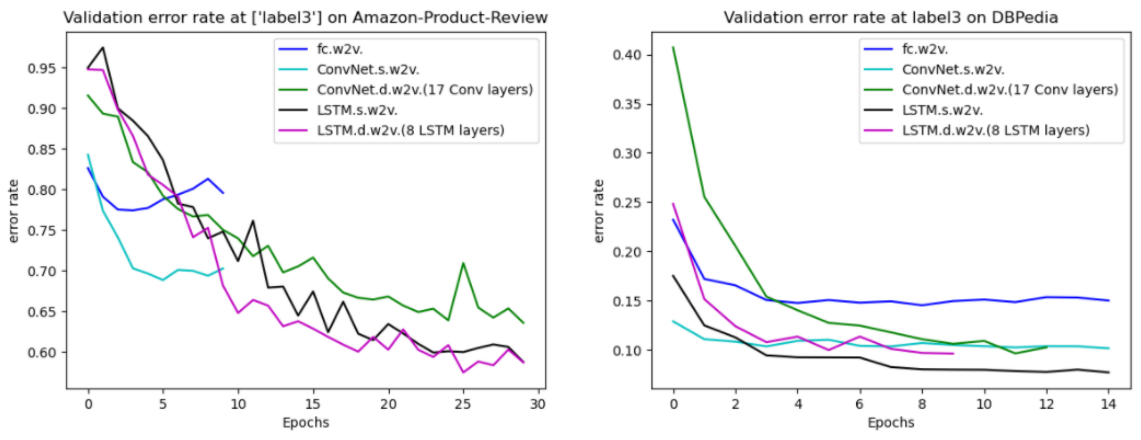

Figure 4: Validation error rate versus training epochs.

LSTM, as a recurrent neural network, is the most robust and stable model among all the models we experimented. On the Amazon product reviews data set, some of the labels have very poor number of samples. The least one only got 64 samples for training. In such a scenario, the LSTM model still achieved a considerable performance.

Convolutional Neural Network could be benefited directly from having deeper layers. In addition, as shown in the results from the DBPedia data set, when a sufficient amount of training data is given, the deep convolutional model could achieve accuracy very close to the best LSTM model we have. In addition, with convolutional layers, the model has more space for potential improvements, such as stacking more layers and enlarging the data set by data augmentation. Another obvious disadvantage of recurrent layers is that it cannot be benefited much from parallel computing as can the convolutional layers. Although LSTM seems to lose the competition, it has several significant strengths.

In closing this section, according to the empirical results, recurrent neural networks such as LSTM are so stable that they can achieve considerable results even under a very poor condition. As a light model, LSTM is efficient to train, and easy to deploy. To the contrary, convolutional neural networks such as ResNet require a lot more resources for training. But in a long-term view, the convolutional model is more appropriate in terms of sustainable development and has a lot of potential improvements.

7 Conclusion and Future Work

This article provides an empirical overview of applying multi-task neural network to the hierarchical text classification task. The empirical results indicate the robustness and effectiveness of this model. As we have presented, the model that could master hierarchical text classification can serve as a great feature to be added to the internet community. The society would be implicitly benefited from various aspects, including but not limited to convenience and effectiveness.

As for future work, we want to explore more structures for feature extraction such as attention mechanism and transformers. In addition to exploring the improvements in model performance, we also want to test the applicability of the multi-task model on various tasks. We hope to see more possible benefits made by these methods.

References

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Association for Computational Linguistics, Barcelona, Spain, pages 214–217.

Rich Caruana. 1997. Multitask learning. *Machine Learning* 28(1):47–75.

Ricardo Cerri, Rodrigo Barros, and André de Carvalho. 2014. Hierarchical multi-label classification using local neural networks. *Journal of Computer and System Sciences* 80(1):39–56.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. 2017. Very deep convolutional networks for text classification. arXiv:1606.01781v2 [cs.CL].

Siddharth Gopal and Yiming Yang. 2013. Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, New York, NY, USA, pages 257–265.

Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. 2020. Learning to branch for multi-task learning. arXiv:2006.01895v2 [cs.LG].

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. arXiv:1512.03385v1 [cs.CV].

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167v3 [cs.LG].

Yury Kashnitsky. 2020. Hierarchical text classification, version 1. Retrieved November 26, 2020 from https://www.kaggle.com/kashnitsky/hierarchical-text-classification/version/1.

Jaeyoung Kim, Mostafa El-Khamy, and Jungwon Lee. 2017. Residual LSTM: Design of a deep recurrent architecture for distant speech recognition. arXiv:1701.03360v3 [cs.LG].

Guangquan Lu, Xishun Zhao, Jian Yin, Weiwei Yang, and Bo Li. 2018. Multi-task learning using variational auto-encoder for sentiment classification. *Pattern Recognition Letters* 132(1):115–122.

Aman Miglani. 2020. Coronavirus tweets NLP — text classification, version 1. Retrieved November 26, 2020 from https://www.kaggle.com/datatattle/covid-19-nlp-text-classification/version/1.

Tomas Mikolov, Kai Chen, Gregory Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv:1301.3781v3 [cs.CL].

Daniel Ofer. 2019. DBPedia, version 2. Retrieved November 26, 2020 from https://www.kaggle.com/danofer/dbpedia-classes/version/2.

Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. 2018. Large-scale hierarchical text classification with recursively regularized deep graph-CNN. In *Proceedings of the 2018 World Wide Web (WWW) Conference*. International World Wide Web Conferences Steering Committee, Geneva, Switzerland, pages 1063–1072.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Lingusitics, Doha, Qatar, pages 1532–1543.

Abhinav Shaw, Natcha Simsiri, Iman Deznaby, Madalina Fiterau, and Tauhidur Rahaman. 2019. Personalized student stress prediction with deep multitask network. arXiv:1906.11356v1 [cs.LG].

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556v6 [cs.CV].

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2016. Character-level convolutional networks for text classification. arXiv:1509.01626v3 [cs.LG].