NeuralClassifier: An Open-source Neural Hierarchical Multi-label Text Classification Toolkit

Liqun Liu*, Funan Mu*, Pengyu Li, Xin Mu, Jing Tang, Xingsheng Ai, Ran Fu, Lifeng Wang, Xing Zhou
Advertising and Marketing Services, Tencent Inc.
{liqunliu, marvinmu, perrypyli, anmarsmu, jamesjtang, felixai, ranfu, fandywang, leostarzhou}@tencent.com

Abstract

In this paper, we introduce NeuralClassifier, a toolkit for neural hierarchical multi-label text classification. NeuralClassifier is designed for quick implementation of neural models for hierarchical multi-label classification task, which is more challenging and common in real-world scenarios. A salient feature is that NeuralClassifier currently provides a variety of text encoders, such as FastText, TextCNN, TextRNN, RCNN, VDCNN, DPCNN, DRNN, AttentiveConvNet and Transformer encoder, etc. It also supports other text classification scenarios, including binary-class and multi-class classification. Built on PyTorch[1], the core operations are calculated in batch, making the toolkit efficient with the acceleration of GPU. Experiments show that models built in our toolkit achieve comparable performance with reported results in the literature.

1 Introduction

Text classification is an important task in Natural Language Processing with many applications, such as web search, information retrieval, ranking and document classification (Deerwester et al., 1990; Pang et al., 2008). As a result of the great success of deep neural networks, a series of classification models based on neural networks that achieve very good performance in practice have been proposed (Kim, 2014; Lai et al., 2015; Joulin et al., 2016; Conneau et al., 2016; Liu et al., 2016; Johnson and Zhang, 2017; Vaswani et al., 2017; Yin and Schütze, 2017; Wang, 2018; Qiao et al., 2018; Guo et al., 2019).

The problem of Hierarchical Multi-label Classification (HMC) is a branch of classification problem. It is a more challenging classification problem in real-world scenarios. Unlike traditional flat and single-label text classification, it aims at considering the interrelationships among labels and classifying the text document into multiple labels, which are organized into a hierarchical structure of tree or DAG (Directed Acyclic Graph). Regularizing the deep architecture with the dependency among labels adopted by the existing solutions (Gopal and Yang, 2013; Peng et al., 2018) is more naturally for solving hierarchical multi-label text classification problem, especially for large scale datasets. It has a wide variety of real-world applications such as question answering (Qu et al., 2012), online advertising (Agrawal et al., 2013), and scientific literature organization (Peng et al., 2016).

```
"task_info": {
    "label_type": "multi-label",
    "hierarchical": true,
    "hierar_taxonomy": "rcv1.taxonomy",
    "hierar_penalty": 0.0001
}
"model_name": "TextCNN",
"device": "cuda",
"checkpoint_dir": "checkpoint/",
"data": {
    "train_files": ["/data/rcv1_train.json"],
    "validate_files": ["/data/rcv1_val.json"],
    "test_files": ["/data/rcv1_test.json"],
}
```

Figure 1: Configuration file segment.

There exist several open-source statistical hierarchical or multi-label text classification toolkits, such as scikit-multilearn[2], sklearn-hierarchical-classification[3], which provide users with various hierarchical or multi-label classification modules based on scikit-learn's interfaces and conventions. On the other hand, there is limited choice for neural hierarchical multi-label text classification toolkits. Although many researchers have released their codes along with their hierarchical or multi-

*Equal contribution
[1]https://pytorch.org/
[2]https://github.com/scikit-multilearn/scikit-multilearn
[3]https://github.com/globality-corp/sklearn-hierarchical-classification

Toolkits	Neural	Multi-label	Hierarchical	Feature Richness	Model Richness
scikit-multilearn	×	✓	×	×	✓
sklearn-hierarchical-classification	×	✓	✓	×	✓
HDLTex	✓	×	✓	×	×
HR-DGCNN	✓	✓	✓	×	×
NeuralClassifier	✓	✓	✓	✓	✓

Table 1: Toolkit Comparison.

```
Macro_average:
Overall: P: 0.926780, R: 0.926938, F1: 0.926859,
level_1: P: 1.000000, R: 0.983000, F1: 0.991427,
level_2: P: 0.988596, R: 0.972668, F1: 0.980567,
level_3: P: 0.946699, R: 0.937501, F1: 0.942077,
level_4: P: 0.925209, R: 0.926333, F1: 0.925771,

Micro_average:
Overall: P: 0.967359, R: 0.978000, F1: 0.972650,
         right: 978, predict: 1011, standard: 1000
level_1: P: 1.000000, R: 0.983000, F1: 0.991427,
         right: 983, predict: 983, standard: 1000
level_2: P: 0.979042, R: 0.981000, F1: 0.980020,
         right: 981, predict: 1002, standard: 1000
level_3: P: 0.973604, R: 0.980573, F1: 0.977076,
         right: 959, predict: 985, standard: 978
level_4: P: 0.962560, R: 0.976716, F1: 0.969586,
         right: 797, predict: 828, standard: 816
```

Figure 2: An illustration of evaluation outputs. "level 1" to "4" are the results of each level in hierarchical classification. Evaluation metrics are macro and micro.

label text classification papers (Kowsari et al., 2017; Peng et al., 2018), but the implementations are mostly focused on specific model structures and specific tasks, which greatly limit their extensions for other similar tasks.

In this paper, we introduce an open-source toolkit, NeuralClassifier[4], a neural hierarchical multi-label text classification toolkit based on PyTorch. It is designed for solving the hierarchical multi-label text classification problem with effective and efficient neural models. It provides a variety of models and features, users can utilize a comfortable configuration file with neural feature design and utilization. We take the layerwise implementation, which includes input layer, embedding layer, encoder layer and output layer. To our best knowledge, our work is the first neural hierarchical multi-label text classification toolkit with rich models. For the details, we give a summary comparison with existing toolkits in Table 1. NeuralClassifier is:

• **Rich in models and features:** An important feature of our work is that, compared with existing toolkits, NeuralClassifier reimplements a very large number of the state-of-the-art text encoders, including FastText (Joulin et al., 2016), TextCNN (Kim, 2014), TextRNN (Liu et al., 2016), RCNN (Lai et al., 2015) , VDCNN (Conneau et al., 2016), DPCNN (Johnson and Zhang, 2017) , AttentiveConvNet (Yin and Schütze, 2017), DRNN (Wang, 2018), Transformer encoder (Vaswani et al., 2017), Star-Transformer encoder (Guo et al., 2019). Meanwhile, NeuralClassifier involves a variety of useful features or widgets, i.e., word-based and char-based input, optimizers, loss functions, embedding methods and attention mechanisms, etc. All those above can be configured through a configuration file. Figure 1 shows a segment of configuration file. Note that users can configure different text encoders and features through the configuration file, and can easily modify the source code to achieve more advanced developments.

• **Suitable for almost all text classification tasks:** NeuralClassifier is designed for hierarchical and multi-label classification, which naturally also supports binary-class and multi-class classification, so it can be considered a universal toolkit for text classification tasks. Especially in hierarchical multi-label classification task, the taxonomy can be organized in the form of a tree or DAG, and instances are multi-labeled during training and testing. It also provides a complete evaluation mechanism. An illustration with results is shown in Figure 2. Users can choose their task types only through a comfortable configuration file without any code work.

• **Effective and efficient:** We conduct the experiments based on a variety of models and features provided by NeuralClassifier. Experiments show models built in our toolkit output comparable performance with reported results in the literature. Furthermore, NeuralClassifier is implemented using batch calculation that can be accelerated using GPU. Our experiments demonstrate that NeuralClassifier is an effective and efficient toolkit.

The rest of this paper is organized as follows: Section 2 describes the detail of architecture of

[4]Code is available at https://github.com/liqunhit/NeuralClassifier

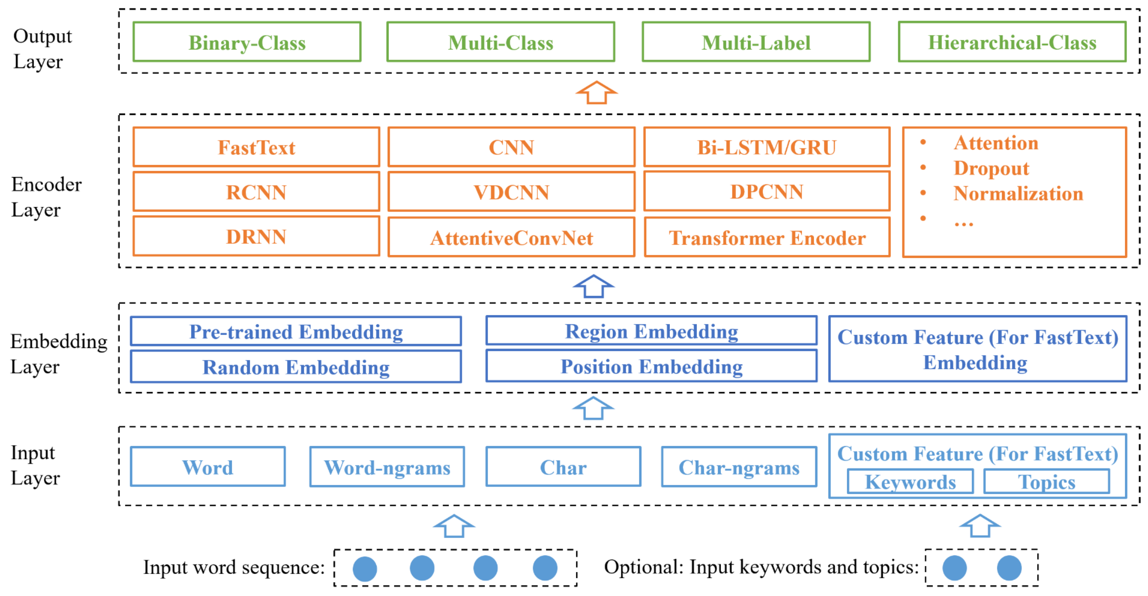

Figure 3: Architecture of NeuralClassifier. There are four layers: an input layer, an embedding layer, an encoder layer and output layer.

NeuralClassifier. The experimental evaluations and results are discussed in Section 3. Finally, Section 4 concludes this paper.

2 NeuralClassifier Architecture

The framework of NeuralClassifier is shown in Figure 3. It is composed of four layers: *input layer*, *embedding layer*, *encoder layer* and *output layer*. At the first layer (*input layer*), the input word sequence will be organized and processed as words, characters, or corresponding n-grams. For FastText, custom features such as keywords and topics are also supported. The embedding of input data will be generated at the embedding layer, subsequently be encoded at encoder layer. On top of the system, the different loss functions are constructed in the output layer to serve the different real-world tasks, i.e., binary-class, multi-class, multi-label and hierarchical multi-label classification. The user can deploy it through a comfortable configuration file without any code work. Note that a salient feature is that users can easily utilize and integrate any widgets in the NeuralClassifier to construct their own structure to satisfy any requirements.

The following will describe the pertinent details of the four layers and the user interface.

2.1 Layer Units

• **Input Layer.** The input text sequence will be processed at input layer. Input text sequence in the form of token (`word`) can be processed into words and characters, along with their n-grams. Custom feature inputs such as keywords and topics are also supported when the text encoder is FastText. All the above can be flexibly configured by the users. Besides, reading input data can be accelerated with multiple processes. See Figure 4 for an example of input data.

```
{
    "doc_label": ["Computer--MachineLearning--DeepLearning",
                  "Neuro--ComputationalNeuro"],
    "doc_token": ["I", "love", "deep", "learning"],
    "doc_keyword": ["deep learning"],
    "doc_topic": ["AI", "Machine learning"]
}
```

Figure 4: An example of input data. Multiple levels of hierarchy are separated with '--'.

• **Embedding Layer.** Various embeddings are processed at this layer. There are four embedding types can be chosen, which are random embedding, pre-trained embedding, region embedding and position embedding. Embedding can be initialized randomly or from a pre-trained embedding input. Region embedding (Qiao et al., 2018) is a supervised enhanced word embedding method that the representation of a word or char has two parts, the embedding of the word itself, and a weighting matrix to interact with the local context, referred to as local context unit. Position embedding (Vaswani et al., 2017) is an embedding method that considers position information in the input sequence.

• **Encoder Layer.** We reimplement a very large number of state-of-the-art text encoders at encoder layer, including FastText, TextCNN, TextRNN, RCNN, VDCNN, DPCNN, DRNN, Transformer encoder, Star-Transformer encoder and AttentiveConvNet. Each encoder has its own hyperparameters that can be configured by users.

• **Output Layer.** This layer determines the specific classification tasks, including binary-class, multi-class, multi-label and hierarchical-class. For the single-label (binary-class and multi-class) classification task, we provide three candidate loss functions, which are `SoftmaxCrossEntopy`, `BCLoss` and `SoftmaxFocalLoss` (Lin et al., 2017). For the hierarchical multi-label classification task, we use `BCELoss` or `SigmodFocalLoss` as the loss function for multi-label classification and add a recursive regularization (Gopal and Yang, 2013; Peng et al., 2018) for hierarchical classification. Using this regularization framework, we can incorporate the hierarchical dependencies between the class-labels into the regularization structure of the parameters thereby encouraging classes nearby in the hierarchy to share similar model parameters. In addition, such a regularization approach is more suitable for large-scale hierarchical multi-label classification task. Users can easily use above functions through the configuration file.

2.2 User Interface

NeuralClassifier provides abundant configuration interfaces, including the common settings, input settings, training settings and network structure settings. Through the configuration file, users can construct most state-of-the-art neural hierarchical multi-label text classification models. `JSON` is used as the configuration file format.

The configuration file has four major parts:

• **Common settings** include the type of classification tasks, which are single-labeled and multi-labeled, whether it is hierarchical (`task_info`), which running device to use (`device`), the specified model (`model_name`), directories of input and output data (`data`), how many subprocesses to use for data loading (`num_worker`), etc.

• **Input settings** include various configurations about input data, such as maximum input sequence length (`max_token_len`), minimum input token count (`min_token_count`), dictionary size (`max_token_dict_size`), pre-trained embeddings of input data (`token_pretrained_file`), etc.

• **Training settings** include the batch size (`batch_size`), type of loss function (`loss_type`), optimizer (`optimizer_type`), learning rate (`learning_rate`), number of epochs (`num_epochs`), which GPUs to use (`visible_device_list`), etc.

• **Network structure settings** specify which text encoders to use, such as TextCNN, TextRNN, RCNN, Transformer, etc. For each text encoder, there are corresponding hyperparameters that can be configured. Take TextCNN as an example, users can configure the size and number of convolution kernels and the number of tops in the pooling (`kernel_sizes`, `num_kernels`, `top_k_max_pooling`).

2.3 Extension

Users can write their own custom modules on all those layers, and self-defined modules can be integrated into the toolkit easily. For example, if a user wants to implement a new classifier model, he/she only needs to implement the part at encoder layer. All the other network structures can be used and controlled through the configuration file.

3 Evaluation

In this section, we conduct several experiments to evaluate the performance of NeuralClassifier using datasets from two public benchmarks, namely, RCV1 (Lewis et al., 2004) and Yelp[5]. The experiments consist of three parts: (1) Results of using rich models and features in Section 3.1; (2) influence of hierarchical information in Section 3.2; (3) speed with batch size in Section 3.3.

3.1 Results of using rich models and features

We use the ability of various models and features provided by our toolkit to illustrate the performance of NeuralClassifier on hierarchical multi-label text classification problem. Concretely, we select a best model through coarse-grained experiments on each of the two benchmarks and fix it, and then fine-tune the features and hyperparameters, such as model structures, input representations, activation functions, optimizers, learning rate, etc. The best performance models[6] are as

[5]https://www.yelp.com/dataset/challenge

[6]Note that the released settings are preliminary attempt so far, we will continue to update the optimization of parameter selection.

Models	RCV1 Micro-F1	Yelp Micro-F1
HR-DGCNN (Peng et al., 2018)	0.7610	–
HMCN (Wehrmann et al., 2018)	0.8080	0.6640
Our best models	**0.8099**	**0.6704**

Table 2: Results on the two benchmarks.

Encoders	RCV1		Yelp	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1
TextCNN	0.7608	0.4649	0.6179	0.3851
TextRNN	0.7883	0.4972	**0.6704**	0.4059
RCNN	**0.8099**	**0.5332**	0.6569	0.3951
FastText	0.6887	0.2701	0.5386	0.1817
DRNN	0.7601	0.4523	0.6174	0.3835
DPCNN	0.7439	0.4141	0.5671	0.2393
VDCNN	0.7263	0.3860	0.6079	0.3427
AttentiveConvNet	0.7533	0.4373	0.6367	0.4040
Region embedding	0.7780	0.4888	0.6601	**0.4514**
Transformer	0.7603	0.4274	0.6533	0.4121
Star-Transformer	0.7668	0.4840	0.6293	0.3977

Table 3: Results of different text encoders.

follows: (1) RCNN with two-layers Bi-GRU and one-layer CNN for RCV1 dataset (input = word, optimizer = Adam, learning rate = 0.008); (2) TextRNN with one-layer Bi-GRU for Yelp dataset (input = word, optimizer = Adam, learning rate = 0.008). Table 2 shows the results of the best models on the two benchmarks. The best models can achieve comparable results with the state-of-the-art HMC models. The results shows the effectiveness of our implementation, and usability of a variety of models and features. Table 3 shows performances of different text encoders. In particular, we can use different combinations of strategies to guide the setup of model for better performance in the real applications.

3.2 Influence of hierarchical information

Hierarchical classification problems can also be solving by flat methods, which regard the hierarchical classification as a flat classification, regardless of the hierarchical relationship between labels. As mentioned before, our toolkit is configurable, we can easily set different loss functions by configuration. In this section, we discuss the influence of hierarchical information. Table 4 shows the results of setting the HMC loss function (Hierarchical) and the traditional multi-label loss function (Flat). As can been seen from the results, hierarchical information can greatly improve performance. It also demonstrates the effectiveness of our implementation.

Encoders	Hierarchical		Flat	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1
TextCNN	0.7608	0.4649	0.7367	0.4224
TextRNN	0.7883	0.4972	0.7546	0.4505
RCNN	**0.8099**	**0.5332**	**0.7955**	**0.5123**
FastText	0.6887	0.2701	0.6865	0.2816
DRNN	0.7601	0.4523	0.7506	0.4450
DPCNN	0.7439	0.4141	0.7423	0.4261
VDCNN	0.7263	0.3860	0.7110	0.3593
AttentiveConvNet	0.7533	0.4373	0.7511	0.4286
Region embedding	0.7780	0.4888	0.7640	0.4617
Transformer	0.7603	0.4274	0.7602	0.4339
Star-Transformer	0.7668	0.4840	0.7618	0.4745

Table 4: Results of hierarchical and flat classification on RCV1.

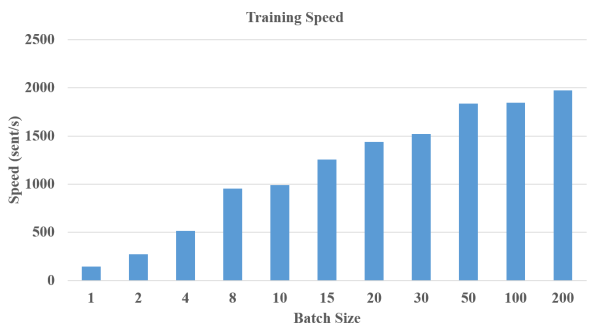

Figure 5: Results of speed with batch size.

3.3 Speed with Batch Size

As NeuralClassifier is implemented on bathed calculation, it can be greatly accelerated through parallel computing through GPU. We test the system speed on training process on RCV1 using a Nvidia Tesla P40 GPU. As shown in Figure 5, the training speed can be significantly accelerated through a large batch size, demonstrating the efficiency of our implementation.

4 Conclusion

This paper presents NeuralClassifier, an open-source neural hierarchical multi-label text classification toolkit. NeuralClassifier provides a large variety of text encoders and features. Users can design their models for different text classification tasks easily through the configuration file. We conduct a series of experiments and the results show that models built on NeuralClassifier can achieve state-of-the-art results with an efficient running speed.

References

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*, pages 13–24.

Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. 2016. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

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*, pages 257–265.

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. 2019. Star-transformer. *arXiv preprint arXiv:1902.09113*.

Rie Johnson and Tong Zhang. 2017. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 562–570.

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

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. 2017. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 364–371.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *29th AAAI Conference on Artificial Intelligence*.

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*, 5:361–397.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2980–2988.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*.

Bo Pang, Lillian Lee, et al. 2008. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135.

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 Conference on World Wide Web*, pages 1063–1072.

Shengwen Peng, Ronghui You, Hongning Wang, Chengxiang Zhai, Hiroshi Mamitsuka, and Shanfeng Zhu. 2016. Deepmesh: deep semantic representation for improving large-scale mesh indexing. *Bioinformatics*, 32(12):i70–i79.

Chao Qiao, Bo Huang, Guocheng Niu, Daren Li, Daxiang Dong, Wei He, Dianhai Yu, and Hua Wu. 2018. A new method of region embedding for text classification. In *International Conference on Learning Representations*.

Bo Qu, Gao Cong, Cuiping Li, Aixin Sun, and Hong Chen. 2012. An evaluation of classification models for question topic categorization. *Journal of the American Society for Information Science and Technology*, 63(5):889–903.

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

Baoxin Wang. 2018. Disconnected recurrent neural networks for text categorization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2311–2320.

Jonatas Wehrmann, Ricardo Cerri, and Rodrigo Barros. 2018. Hierarchical multi-label classification networks. In *International Conference on Machine Learning*, pages 5225–5234.

Wenpeng Yin and Hinrich Schütze. 2017. Attentive convolution. *arXiv preprint arXiv:1710.00519*.