LibN3L:A Lightweight Package for Neural NLP

Meishan Zhang[1], **Jie Yang**[2], **Zhiyang Teng**[2], **Yue Zhang**[2]
1.School of Computer Science and Technology, Heilongjiang University, Harbin, China
2. Singapore University of Technology and Design
{meishan_zhang, yue_zhang}@sutd.edu.sg
{jie_yang, zhiyang_teng}@mymail.sutd.edu.sg

Abstract

We present a light-weight machine learning tool for NLP research. The package supports operations on both discrete and dense vectors, facilitating implementation of linear models as well as neural models. It provides several basic layers which mainly aims for single-layer linear and non-linear transformations. By using these layers, we can conveniently implement linear models and simple neural models. Besides, this package also integrates several complex layers by composing those basic layers, such as RNN, Attention Pooling, LSTM and gated RNN. Those complex layers can be used to implement deep neural models directly.

Keywords: neural network, sequence labeling, recurrent neural network

1. Introduction

Deep learning methods have received increasing research attention in natural language processing (NLP), with neural models being built for classification (Kalchbrenner et al., 2014), sequence labeling (Collobert et al., 2011), parsing (Socher et al., 2013; Dyer et al., 2015; Zhou et al., 2015; Weiss et al., 2015), machine translation (Cho et al., 2014), fine-grained sentiment analysis (Zhang et al., 2015) and other tasks. This surge of the interest gives rise to a demand of software libraries, which can facilitate research by allowing fast prototyping and modeling for experimentation.

For traditional methods such as conditional random fields (CRF) (Lafferty et al., 2001) and SVM (Vapnik, 1995), there has been various software toolkits, implemented in different programming languages, including Java, Python and C++. These toolkits offer a large degree of variety for building NLP models by using or adapting the machine learning algorithms. For deep learning, a number of software tools have been developed, including *theano* [1] *(Bergstra et al., 2010), caffe* [2] *(Jia et al., 2014), CNN*[3], *torch*[4] etc. These tools are based on different programming languages and design concepts. On the other hand, most of these libraries are not designed specifically for NLP tasks. In addition, many existing libraries define a complex class hierarchy, making it difficult for some users to use or adapt the modules.

We present another deep learning toolkit in C++, designed specifically for NLP applications. The main objective is to make it extremely light-weight, so as to minimize the effort in building a neural model. We take a layered approach, offering high-level models for classification and sequence labeling, such as neural CRF (Do et al., 2010), recurrent neural networks (RNN) (Graves, 2012) and long-short-term memories (LSTM) (Hochreiter and Schmidhuber, 1997), which are frequently used in NLP. On the other hand, we minimize encapsulation, implementing neural structures strictly abiding by their formal definitions, so as to make it easy to work directly with neural layers and facilitate extensions to existing network structures.

Our design is centralized in the structure of a neural layer, which performs the standard feed-forward function and back-propagation. We provide a wide range of built-in neural activation functions, and common operations such as concatenation, pooling, window function and embedding lookup, which are needed by most NLP tasks. We support flexible objective functions and optimization methods, such as max-margin, max likelihood criterions and AdaGrad (Duchi et al., 2011), and also verification functions such as gradient check. One uniqueness of our toolkit is the support of both dense continuous features and sparse indicator features in neural layers, making it convenient also to build traditional discrete models such as the perceptron, logistic regression and CRF, and to combine discrete and continuous features (Ma et al., 2014; Durrett and Klein, 2015; Zhang and Zhang, 2015).

Taking word segmentation, POS-tagging and name entity recognition (NER) as typical examples, we show how state-of-the-art discrete, neural and hybrid models can be built using our toolkit. For example, we show how a bidirectional LSTM model can be built for POS tagging in only 23-lines (12 for inference and 11 for back-propagation) of codes, which gives highly competitive accuracies on standard benchmarks.

2. Classes

2.1. Base Layers

Shown in Table 1, we provide several basic classes, which are widely used in neural networks and discrete machine learning algorithms, including atomic layers, pooling functions, loss functions and others. All classes have three interfaces, one for obtaining forward outputs, one for computing backward losses, and the last for update parameters.

Neural Layers The neural layers are single atomic layers used in neural networks, which support one, two or three input vectors. In Table 1, f can be any activation function,

[1]https://github.com/Theano/Theano
[2]http://caffe.berkeleyvision.org/
[3]https://github.com/clab/cnn
[4]http://torch.ch/

Atomic Layers	Dense	uni-layer: $y = f(Wx + b)$ bi-layer: $y = f(W_1x_1 + W_2x_2 + b)$ tri-layer: $y = f(W_1x_1 + W_2x_2 + W_3x_3 + b)$ tensor-layer: $y = f(x_1Tx_2 + b)$
	Discrete	uni-layer: $y = f(Wx)$
Pooling	$y = \sum_{i=1}^{n} \alpha_i \odot x_i$	$\begin{cases} \text{max: } \alpha_{i,j} = 1, \text{when } i = \arg\max_s(x_{s,j}), \text{otherwise } 0; \\ \text{min: } \alpha_{i,j} = 1, \text{when } i = \arg\min_s(x_{s,j}), \text{otherwise } 0; \\ \text{average: } \alpha_{i,j} = \frac{1}{n}; \\ \text{sum: } \alpha_{i,j} = 1. \end{cases}$
Loss Function	Classifier	max entropy (MAXENT) : $o, y \rightarrow \partial o$: $loss(o) = -y \log \mathrm{softmax}(o);$ $\partial o = \frac{\mathbf{d} loss(o)}{\mathbf{d} o}$
	Structural Learning	CRF, max likelihood (CRFML) : $o_1^n, y_1^n \rightarrow \partial o_1^n$: $loss(o_1^n) = -\log p(y_1^n \| o_1^n),$ where $p(.)$ can be computed via the forward-backward algorithm (Sutton and Mccallum, 2007); $\partial o_1^n = \frac{\mathbf{d} loss(o_1^n)}{\mathbf{d} o_1^n}$
		CRF, max margin (CRFMM) : $o_1^n, y_1^n \rightarrow \partial o_1^n$: $loss(o_1^n) = \max_{\hat{y}_1^n}(s(\hat{y}_1^n) + \delta(\hat{y}_1^n, y_1^n)) - s(y_1^n),$ where $\hat{y}_1^n$ is an answer sequence with one label for each position; $\partial o_1^n = \frac{\mathbf{d} loss(o_1^n)}{\mathbf{d} o_1^n}$
Others	LookupTable: E, specifying vector representations for one vocabulary.	
	Concatenation: $y = x_1 \oplus x_2 \oplus \cdots \oplus x_M$	
	Dropout: $y = m \odot x$, where m is a mask vector	
	Window function: $x_1^n \rightarrow y_1^n$, where $y_i = x_{i-c} \oplus \cdots \oplus x_i \oplus \cdots \oplus x_{i+c}$	

Table 1: Base classes.

RNN	$x_1^n \rightarrow y_1^n : y_j = f(Wx_j + Uy_{j\pm1} + b)$
GRNN	$x_1^n \rightarrow y_1^n$, where y_1^n is computed by: $r_j = \sigma(W_1x_j + U_1y_{j\pm1} + b_1)$ $\tilde{y}_j = f(W_2x_j + U_2(r_j \odot y_{j\pm1}) + b_2)$ $z_j = \sigma(W_3x_j + U_3y_{j\pm1} + b_3)$ $y_j = (\vec{1} - z_j) \odot y_{j\pm1} + z_j \odot \tilde{y}_j$
LSTM	$x_1^n \rightarrow y_1^n$, where y_1^n is computed by: $i_j = \sigma(W_1x_j + U_1y_{j\pm1} + V_1c_{j\pm1} + b_1)$ $f_j = \sigma(W_2x_j + U_2y_{j\pm1} + V_2c_{j\pm1} + b_2)$ $\tilde{c}_j = f(W_3x_j + U_3y_{j\pm1} + b_3)$ $c_j = i_j \odot \tilde{c}_j + f_j \odot c_{j\pm1}$ $o_j = \sigma(W_4x_j + U_4y_{j\pm1} + V_4c_j + b_4)$ $y_j = o_j \odot f(c_j)$
Attention Model	$x_1^n, a_1^n \rightarrow y,$ where y is computed by: $h_j = f(Wx_j + Ua_j + b)$ $\alpha_j = \exp(h_j)$ $z = \sum_{j=1}^{n} \alpha_j$ $y = \sum_{j=1}^{n} \frac{\alpha_j \odot x_j}{z}$

Table 2: Classes of neural network structures.

such as the simple id operation or non-linear functions including $\tanh$, $\mathrm{sigmoid}$ and $\exp$. For discrete features, we support only one vector input. A logistic regression classifier can be built using one single discrete layer.

Pooling Pooling functions are widely used to obtain fixed-dimensional output from sequential vectors of variable lengths. Commonly-used pooling techniques include *max*, *min* and *averaged* function. We implement *sum* pooling also.

Loss Function We offer three different loss functions, one for classification, based on the max-entropy principle and two for structural sequence labeling problems, based on the theory of CRF, with a max likelihood and a max margin objective, respectively.

Others To facilitate model building, we provide some useful classes such as lookup table, drop out, concatenation and window feature extraction. These functions are all shown in Table 1.

2.2. Network structures

Using basic classes, one can build advanced neural network structures in the literature. In this package, we implement four different neural networks, including a simple recurrent neural network (RNN), a gated recurrent neural network (GRNN), a long-short term memory neural network (LSTM) and an attention model. Their definitions are given in Table 2.

3. Evaluation

We show how to apply the package to building neural network models for Chinese word segmentation, POS tagging and NER. All three tasks are formalized as sequence labeling problems. The general framework is shown in Figure 1, where we collect input vectors (t_1^n) at the bottom for each word, and then add a windowlized layer to exploit surrounding information, obtaining x_1^n. Then, we apply two LSTM neural networks, one being computed from left

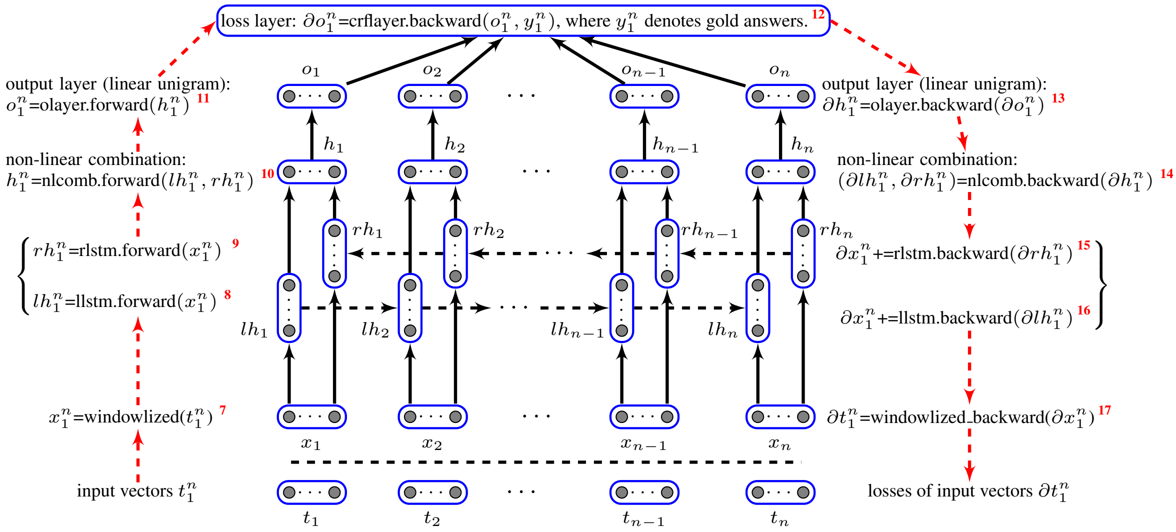

Figure 1: Neural framework for word segmentation, POS tagging and named entity recognition.

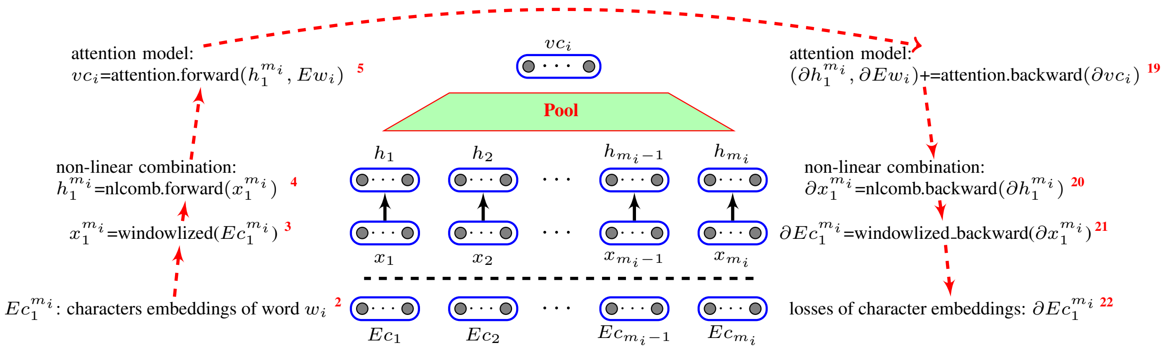

Figure 2: vector representation derived from character sequences.

to right lh_1^n and the other being computed from right to left rh_1^n. These two kinds of features are combined using a non-linear combination layer, giving h_1^n. Finally, we compute output vectors o_1^n, scoring different labels at each position.

During training, we run standard back-propagation. We choose CRF max-margin loss to compute the output losses ∂o_1^n. Then step by step, we compute the losses of h_1^n, lh_1^n, rh_1^n, x_1^n and t_1^n, aggregating losses for each parameter at each layer. Finally, we use Adagrad to update parameters for all layers.

Between segmentation, POS tagging and NER, the differences lie mainly in the input vectors t_1^n. For Chinese word segmentation, we use the concatenation of character unigram embeddings Ec_i and bigram embeddings $Ec_{i-1}c_i$ at each position as the input vector t_i. The character unigram and bigram embeddings are pretrained separately. For POS tagging, t_i consists of embedding Ew_i of the word w_i and its vector representation vc_i derived from its character sequence $c_1^{m_i}$ (m_i is the length of word w_i). vc_i is constructed according to neural network structures shown in Figure 2. For NER, t_i consists of three parts, including Ew_i, vc_i and the word's POS tag embedding Ep_i. The deep neural POS tagging model consists of only 23 lines of code, as marked by red superscripts in Table 3, Figure 2 and Figure 1.

Besides the neural models above, we also implement discrete models for the three tasks. The discrete features are extracted according to Liu et al. (2014), Toutanova et al. (2003) and Che et al. (2013) for word segmentation, POS tagging and NER, respectively. We simply apply the sparse atomic layer and exploit the same CRF max-margin for training model parameters. Finally, we make combinations of the discrete and neural models by aggregating their output vectors.

Results.

We conduct experiments on several datasets. For Chinese word segmentation, we exploit PKU, MSR and CTB60 datasets, where the training and testing corpus of PKU and MSR can be downloaded from BakeOff2005 website[5]. For POS tagging, we perform experiments on both English and Chinese datasets. For English, we follow Toutanova et al. (2003), using WSJ sections of 0-18 as the training dataset, section 19-21 as the development corpus and section 22-24 as the testing dataset. For Chinese, we use the same data set as Li et al. (2015). For NER, we follow Che et al. (2013) to

[5]http://www.sighan.org/bakeoff2005/. We split 10% of the training corpus as the development corpus. The training, development and testing sections corpus of CTB60 is the same as (Zhang et al., 2014).

Operation	Word Segmentation	POS Tagging	NER
Forward	Ec_i = uniCharE.lookup(c_i) Ec_ic_{i-1} = biCharE.lookup(c_ic_{i-1}) t_i = concat(Ec_i, Ec_ic_{i-1})	Ew_i = wordE.lookup(w_i) [1] vc_i = vector($c_1^{m_i}$) t_i = concat(Ew_i, vc_i) [6]	Ew_i = wordE.lookup(w_i) Ep_i = wordE.lookup(p_i) vc_i = vector($c_1^{m_i}$) t_i = concat(Ew_i, Ep_i, vc_i)
Backward	(∂Ec_i, ∂Ec_ic_{i-1}) = unconcat(∂t_i) uniCharE.backloss(c_i, ∂Ec_i) biCharE.backloss(c_ic_{i-1}, ∂Ec_ic_{i-1})	(∂Ew_i, ∂vc_i) = unconcat(∂t_i) [18] $\partial c_1^{m_i}$ = vector_backward(∂vc_i) wordE.backloss(w_i, ∂Ew_i) [23]	(∂Ew_i, ∂p_i, ∂vc_i) = unconcat(∂t_i) $\partial c_1^{m_i}$ = vector_backward(∂vc_i) posE.backloss(p_i, ∂Ep_i) wordE.backloss(w_i, ∂Ew_i)

Table 3: The obtaining of word representation.

Model	Chinese Word Segmentation									POS Tagging		NER					
	PKU			MSR			CTB60			English	Chinese	English			Chinese		
	P	R	F	P	R	F	P	R	F	Acc	Acc	P	R	F	P	R	F
Discrete	95.42	94.56	94.99	96.94	96.61	96.78	95.43	95.16	95.29	97.23	93.97	80.14	79.29	79.71	72.67	73.92	73.29
Neural	94.29	94.56	94.42	96.79	**97.54**	97.17	94.48	95.01	94.75	97.28	94.02	77.25	80.19	78.69	65.59	71.84	68.57
Hybrid	**95.74**	**95.12**	**95.42**	**97.01**	97.39	**97.20**	**95.68**	**95.64**	**95.66**	**97.47**	**95.07**	81.90	**83.26**	**82.57**	72.98	**80.15**	**76.40**
State-of-the-art	N/A	N/A	94.50	N/A	N/A	**97.20**	N/A	N/A	95.05	97.24	94.10	**82.95**	76.67	79.68	**76.90**	63.32	69.45

Table 4: Main results.

split Ontonotes 4.0 to get the English and Chinese datasets. Our experimental results are shown in Table 4. As can be seen for the table, our neural models give competitive results compared the state-of-the-art results on each task, which are Zhang and Clark (2007) for Chinese word segmentation, Toutanova et al. (2003) for English POS tagging, Li et al. (2015) for Chinese POS tagging and Che et al. (2013) for English and Chinese NER.

4. Code

Our code and examples in this paper is available under GPL at https://github.com/SUTDNLP/, including repositories of *LibN3L*, *NNSegmentation*, *NNPOSTagging* and *NNNamedEntity*.

5. Acknowledgments

We thank the anonymous reviewers for their constructive comments, which helped to improve the paper. This work is supported by the Singapore Ministry of Education (MOE) AcRF Tier 2 grant T2MOE201301, SRG ISTD 2012 038 from Singapore University of Technology and Design, and National Natural Science Foundation of China (NSFC) under grant 61170148.

6. Bibliographical References

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX.

Che, W., Wang, M., Manning, C. D., and Liu, T. (2013). Named entity recognition with bilingual constraints. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 52–62.

Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. *Syntax, Semantics and Structure in Statistical Translation*, page 103.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Do, T., Arti, T., et al. (2010). Neural conditional random fields. In *International Conference on Artificial Intelligence and Statistics*, pages 177–184.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Durrett, G. and Klein, D. (2015). Neural crf parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China, July. Association for Computational Linguistics.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. (2015). Transition-based dependency parsing with stack long short-term memory. In *ACL*, pages 334–343.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385. Springer.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. In *Proceedings of the 52nd ACL*, pages 655–665.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289.

Li, Z., Chao, J., Zhang, M., and Chen, W. (2015). Coupled sequence labeling on heterogeneous annotations: Pos tagging as a case study. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1783–1792, Beijing, China, July. Association for Computational Linguistics.

Liu, Y., Zhang, Y., Che, W., Liu, T., and Wu, F. (2014). Domain adaptation for crf-based chinese word segmentation using free annotations. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 864–874, Doha, Qatar, October. Association for Computational Linguistics.

Ma, J., Zhang, Y., and Zhu, J. (2014). Tagging the web: Building a robust web tagger with neural network. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 144–154.

Socher, R., Bauer, J., Manning, C. D., and Andrew Y., N. (2013). Parsing with compositional vector grammars. In *Proceedings of the 51st ACL*, pages 455–465.

Sutton, C. and Mccallum, A. (2007). An Introduction to Conditional Random Fields for Relational Learning. In Lise Getoor et al., editors, *Introduction to statistical relational learning*, chapter 4, page 93. MIT Press.

Toutanova, K., Klein, D., Manning, C., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. New York.

Weiss, D., Alberti, C., Collins, M., and Petrov, S. (2015). Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China, July. Association for Computational Linguistics.

Zhang, Y. and Clark, S. (2007). Chinese segmentation with a word-based perceptron algorithm. In *Proceedings of the 45th ACL*, pages 840–847.

Zhang, M. and Zhang, Y. (2015). Combining discrete and continuous features for deterministic transition-based dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1316–1321, Lisbon, Portugal, September. Association for Computational Linguistics.

Zhang, M., Zhang, Y., Che, W., and Liu, T. (2014). Character-level chinese dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1326–1336, Baltimore, Maryland, June. Association for Computational Linguistics.

Zhang, M., Zhang, Y., and Vo, D.-T. (2015). Neural networks for open domain targeted sentiment. In *Proceedings of the 2015 Conference on EMNLP*.

Zhou, H., Zhang, Y., Huang, S., and Chen, J. (2015). A neural probabilistic structured-prediction model for transition-based dependency parsing. In *Proceedings of the 53rd ACL*, pages 1213–1222.