Text Window Denoising Autoencoder: Building Deep Architecture for Chinese Word Segmentation

Ke Wu, Zhiqiang Gao, Cheng Peng, and Xiao Wen

School of Computer Science & Engineering, Southeast University, Nanjing 210096, China

Abstract. Deep learning is the new frontier of machine learning research, which has led to many recent breakthroughs in English natural language processing. However, there are inherent differences between Chinese and English, and little work has been done to apply deep learning techniques to Chinese natural language processing. In this paper, we propose a deep neural network model: *text window denoising autoencoder*, as well as a complete pre-training solution as a new way to solve classical Chinese natural language processing problems. This method does not require any linguistic knowledge or manual feature design, and can be applied to various Chinese natural language processing tasks, such as Chinese word segmentation. On the PKU dataset of Chinese word segmentation bakeoff 2005, applying this method decreases the F1 error rate by 11.9% for deep neural network based models. We are the first to apply deep learning methods to Chinese word segmentation to our best knowledge.

Keywords: Deep Learning, Word Segmentation, Denoising Autoencoder, Chinese Natural Language Processing.

1 Introduction

Researchers have applied deep learning methods to sequence tagging problems in natural language processing (NLP) of *English*, such as chucking, named entity recognition, etc, and great results have been achieved [4]. The basic idea is to firstly construct a real-valued vector for common English words (these vectors are called word embeddings [1]), and then use a multi-layer neural network to process these vectors.

However, there are inherent differences between *Chinese* and *English*, therefore one cannot simply apply deep learning methods for English NLP to Chinese. There are two major differences: 1) The Chinese language is composed of characters, while English of words. According to our statistical experiments on Chinese and English Wikipedia corpus, it only takes a Chinese dictionary of roughly 5,000 characters to cover more than 99% of all characters used in the entire Chinese Wikipedia, compared to an English dictionary of more than 120,000 words to cover the same portion of words in English Wikipedia. This shows that there

G. Zhou et al. (Eds.): NLPCC 2013, CCIS 400, pp. 1–12, 2013.

are far less commonly used Chinese characters than English words. 2) Meanings in Chinese are conveyed by complex relationships between characters. Although Chinese characters have meanings themselves, they can still form words that have completely different meanings from the meanings represented by the characters alone. By using deep architectures, these kinds of complex relationships between Chinese characters can be represented and extracted as highly abstract features, therefore deep architectures are even more important in Chinese NLP.

Because of these differences, a complete pre-training is more beneficial for deep models for Chinese NLP. The current deep learning approaches for English NLP lack a complete pre-training solution for building deep architectures both theoretically and practically. For example, in [4], they only pre-trained the word embeddings, and the weights of the hidden layers were simply initialized randomly. Part of the reason is that due to the very large vocabulary of commonly used English words, most parameters in deep models for English NLP are in the word embeddings. That's not the case in deep models for Chinese NLP, so a proper pre-training of the hidden layers is even more necessary. Another issue is, although methods for pre-training English word embeddings by training a neural language model have been proposed, there is no explanation on why training a neural language model is a good way to pre-train the word embeddings, or its relationship with other commonly used pre-training methods.

In this paper, we follow the fundamental architectures used in deep learning methods for English NLP and apply them to Chinese. On top of that, we propose a *complete pre-training solution*. Note that our method can be used on any sequence tagging tasks in Chinese NLP, and in this paper we focus on Chinese word segmentation. Our major contributions are: 1) We use a different criterion to build Chinese neural language model, and get better convergence than the criterions commonly used in training English neural language model. 2) We explain that the training process of our neural language model is essentially the same as training a special denoising autoencoder on text window, which we call ***t**ext **w**indow de**n**oising **a**utoencoder (TINA)*. 3) We describe the method to stack text window denoising autoencoders as a way to pre-train deep neural networks for Chinese word segmentation.

This paper is organized as follows. Section 2 describes the neural network framework we used for Chinese word segmentation. In section 3 we propose our Chinese neural language model training criterion. In Section 4 we propose text window denoising autoencoder, and we also describe how to stack TINA and build pre-trained deep neural network models for Chinese word segmentation. Section 5 gives our experiments showing the effectiveness of the TINA model. Section 6 introduces related work briefly. We conclude our work in section 7.

2 Deep Neural Network Framework for Chinese Word Segmentation

We view Chinese word segmentation as a sequence tagging problem, which means to assign a tag to each Chinese character (we use the "BIU" tag schema [9]).

The input of the neural network model is a fixed size text window (a window of Chinese characters in a sentence), and the output is a probability distribution of tags for the character in the center of the text window. The neural network architecture [4] is shown in Figure 1. The *embedding layer* performs a matrix lookup operation, and finds the corresponding real-valued vector in Chinese character embeddings (a real-valued matrix) for each character in the input text window. These real-valued vectors will then be concatenated and input to the *hidden layers*, which are classical neural network layers. The *output layer* is a softmax layer, which ensures all of the output values are between 0 and 1, and that their sum is 1. It is a generalization of the logistic function to multiple variables, which is called softmax function. Each output node corresponds to a specific tag.

More formally, for each character $c \in \mathcal{D}$ ($\mathcal{D}$ is a Chinese character dictionary), there is a corresponding d-dimensional real-valued vector in the matrix $\mathbf{W} \in \mathfrak{R}^{d \times |\mathcal{D}|}$. This matrix is the Chinese character embeddings. The first layer of the neural network is a table lookup and concatenate operation: $L_{input}(\mathbf{c_1}, \ldots, \mathbf{c_s}) = (\mathbf{W} \cdot \mathbf{c_1}, \ldots, \mathbf{W} \cdot \mathbf{c_s})$, in which $\mathbf{c}$ is the one-hot representation of the character. The hidden layers take the form of classical neural network, and we choose tanh as the activation function: $L_{hidden} = \tanh(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$. The output layer is a softmax layer, in which the output of node i is:

$$L_{output_i} = \frac{\exp(\mathbf{w_i} \cdot \mathbf{x} + b_i)}{\sum_{j=1}^{n} \exp(\mathbf{w_j} \cdot \mathbf{x} + b_j)}. \tag{1}$$

We could train this neural network using standard backpropagation (without pre-training), but that will only lead to poor results even with very few hidden layers. According to recent researches in deep learning, pre-training can significantly improve the performance of deep neural networks. In the following sections, we introduce a complete pre-training solution, which pre-trains both the embedding layer and the hidden layers.

3 Pre-train Chinese Character Embeddings

Following the work in English deep NLP [4], we can pre-train the Chinese character embeddings by training a (slightly unconventional) Chinese neural language model. We want to build a neural network to predict the center character in a text window given its context (the characters in the text window except the one in the center), or more formally, $P(w|w_{-s/2}^{-1}, w_1^{s/2})$, where s denote the size of the text window, and w_a^b denotes characters from position a to b in the text window (position of the center character is 0). The basic structure of the neural network is the same as the one described in the previous section, but with a different output layer. One way to do it is to use a softmax output layer that has one output node per character for all the characters in the dictionary, thus ask the model to predict $P(w|w_{-s/2}^{-1}, w_1^{s/2})$ directly [1]. The problem with this method is that the whole model will get very large (the output layer has the same size as the dictionary), and therefore very hard to train.

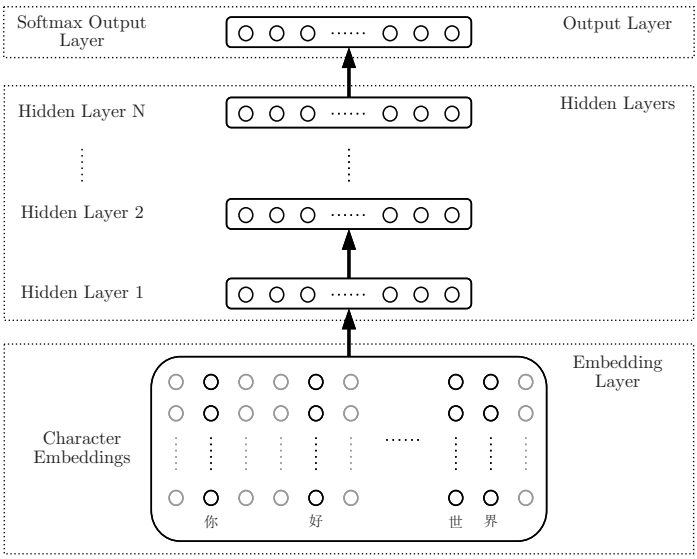

Fig. 1. The architecture of deep neural networks for Chinese NLP. The *embedding layer* maps the input text window into real-valued vectors by performing matrix lookup operation. The *hidden layers* are classical neural network layers. The *output layer* is a softmax layer, with each node corresponding to a specific tag ("BIU" for word segmentation).

Instead of predicting the probability of all the characters directly, we put forward a different training criterion. The model is given the context as well as a (random) character, and can estimate the probability that the given character is the correct one with the context. Or more formally, we try to model $P_c(c_0 = w_0|w_{-s/2}^{-1}, w_1^{s/2}, c_0)$. This is essentially the same as training a model to predict $P(w|w_{-s/2}^{-1}, w_1^{s/2})$ directly, because we can define that

$$P(w|w_{-s/2}^{-1}, w_1^{s/2}) = \frac{P_c(w = w_0|w_{-s/2}^{-1}, w_1^{s/2}, w)}{\sum_{c_i \in \mathcal{D}} P_c(c_i = w_0|w_{-s/2}^{-1}, w_1^{s/2}, c_i)}, \tag{2}$$

which means one can simply assign a fixed context to every character in the dictionary and let the model predict the probability that the character is the correct one, and then generate a joint probability on top of all the outputs.

The input of our neural language model is the context of the text window, as well as another character c_0. The output layer has only one sigmoid node, which outputs the probability $P_c(c_0 = w_0|w_{-s/2}^{-1}, w_1^{s/2}, c_0)$. The remaining layers are the same as the neural network framework introduced in the last section. This model is much smaller in terms of parameters, so it's much easier to train. The training only requires unsupervised data. Text windows are extracted

directly from a given Chinese corpus as positive examples, and for every positive example, a corresponding negative example is generated by replacing the center character in the text window with a random character. We then train the model by maximizing the following log-likelihood criterion for probability $P_c(c_0 = w_0|w_{-s/2}^{-1}, w_1^{s/2}, c_0)$:

$$\begin{aligned}\ell(\theta : \mathcal{T}) = &\sum_{\forall c_0 = w_0} \log P_c(c_0 = w_0|\theta, w_{-s/2}^{-1}, w_1^{s/2}, c_0) \\ &- \sum_{\forall c_0' \neq w_0} \log P_c(c_0' = w_0|\theta, w_{-s/2}^{-1}, w_1^{s/2}, c_0),\end{aligned} \tag{3}$$

where θ denote parameters in the model, and $\mathcal{T}$ denote all the text windows.

Note that optimizing the likelihood of $P_c(c_0 = w_0|w_{-s/2}^{-1}, w_1^{s/2}, c_0)$ is equivalent to optimizing the likelihood of $P(w|w_{-s/2}^{-1}, w_1^{s/2})$ according to our definition.

4 Pre-train Hidden Layers: (Stacked) Text Window Denoising Autoencoder

In this section, we describe our method for pre-training the hidden layers. First, we introduce text windows denoising autoencoder, then we describe how to stack TINA and build a deep neural network with both embedding layer and hidden layers pre-trained.

4.1 Neural Language Model as Text Window Denoising Autoencoder

The neural language model that we introduced in the last section is essentially a special denoising autoencoder, which we call ***t**ext **w**indow de**n**oising **a**utoencoder (TINA)*. A denoising autoencoder is consisted of two parts, namely encoder and decoder. The encoder processes noised data and produces real-valued vector as an "encode" (features) of the data. The decoder then processes the "encode" and try to reconstruct the clean data. Denoising autoencoders are trained by a reconstruction error criterion, which measures the error between the reconstructed clean data and the original data. Denoising autoencoders have already been shown very useful at building deep architectures [12].

The neural language model training process can be evaluated from a denoiser perspective. The data that we're trying to model is text window. We omit the center character as a way to introduce noise, and we ask the model to try to guess (reconstruct) the center character given its context as a denoise process. For example, consider a text window "你好吗世界", we give TINA the context of this text window (你好世界), and then ask the model to guess (reconstruct) the center character (吗). TINA does this by assigning each character a probability of being "the one" given the context. These probability can then be seen as a

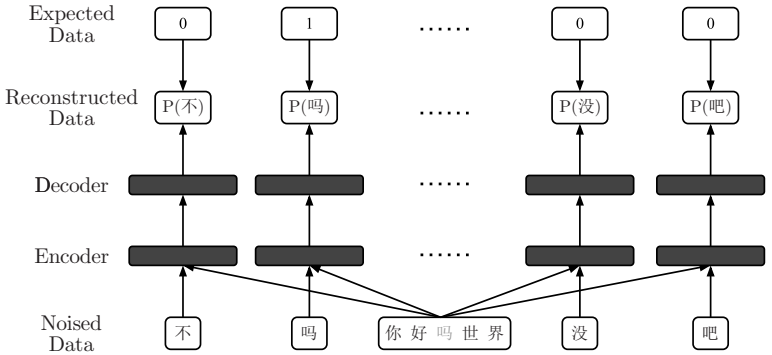

Fig. 2. An example of TINA. The text window is "你好吗世界". We give TINA the context of this text window (你好世界), and ask the model to guess (reconstruct) the center character (吗). TINA does this by assigning each character a probability of being "the one" given the context. These probability can then be regarded as a reconstruction of the one-hot representation of the original center character (吗).

reconstruction of the one-hot representation of the original center character (吗). This example is illustrated in Figure 2.

Formally, given an example of our neural language model with a single hidden layer:

$$\begin{aligned} L_1(\mathbf{c_1},\ldots,\mathbf{c_s}) &= L_{hidden}(L_{input}(\mathbf{c_1},\ldots,\mathbf{c_s})) \\ &= \tanh(\mathbf{w}\cdot(\mathbf{W}\cdot\mathbf{c_1},\ldots,\mathbf{W}\cdot\mathbf{c_s})+\mathbf{b}) \\ L_2(\mathbf{c_1},\ldots,\mathbf{c_s}) &= L_{output}(L_1(\mathbf{c_1},\ldots,\mathbf{c_s})) \\ &= \mathrm{sigmoid}(\mathbf{w}\cdot(L_1(\mathbf{c_1},\ldots,\mathbf{c_s})+\mathbf{b}), \end{aligned} \tag{4}$$

the encoder of the *text window denoising autoencoder* has the form:

$$encoder(\mathbf{x}) = (L_1(\mathbf{x},\mathbf{c_1}),\ldots,L_1(\mathbf{x},\mathbf{c_s})), \tag{5}$$

where $\mathbf{x} = (w_{-s/2}^{-1}, w_1^{s/2})$, which is a text window with the center character omitted, and the features are:

$$feature(\mathbf{x}) = (\mathbf{y_1},\ldots,\mathbf{y_n}), \tag{6}$$

where $\mathbf{y_i} = L_1(\mathbf{x},\mathbf{c_i})$. The decoder is:

$$decoder(\mathbf{y}) = (L_2(\mathbf{y_1}),\ldots,L_2(\mathbf{y_n})). \tag{7}$$

The (square) reconstruction error that this text window denoising autoencoder is minimizing against is:

$$E(\theta, w_{-s/2}^{-1}, w_0, w_1^{s/2}) = \sum_{\forall c_i \in \mathcal{D}} (r_i - 1_{\{c_i = w_0\}})^2, \tag{8}$$

Note that since:

$$r_i = L_2(\mathbf{y_i}) = P_c(c_i = w_0|\theta, w_{-s/2}^{-1}, c_i, w_1^{s/2}), \tag{9}$$

minimizing the reconstruction loss function of the text window denoising autoencoder is exactly the same as maximizing the log-likelihood criterion we proposed for the neural language model.

4.2 Building Deep Architecture

We have already explained that our neural language model can be seen as a text window denoising autoencoder, and it becomes natural to simply follow the stacking strategy of standard denoising autoencoders [12] and build a deep neural network for Chinese word segmentation with both embedding layer and hidden layers pre-trained.

Specifically, the stacking process goes as follows: 1) Train a TINA with a single hidden layer by training a neural language model with a single hidden layer using the method we described in the last section. 2) Remove the softmax output layer, which is the decoder, and add another hidden layer as well as a new softmax output layer. 3) Train this TINA with two hidden layers the same way as step 1, and after that the model becomes one layer deeper.

This stacking process can be repeated several times until a stacked TINA model with sufficient depth (hidden layers) has been built. Then one could simply throw away the last softmax layer, and add a new softmax layer according to the specific task (which in our case is word segmentation), and lastly use back-propagation to fine-tune this pre-trained deep neural network with supervised data.

It's worth mentioning that one could also train a neural language model with a single hidden layer, then fix the word embeddings and train a standard denoising autoencoder on top of it [11]. However, this method under performs our method, which is shown in section 5.

5 Experiments and Analysis

In this section, we demonstrate[1]: 1) The performance of text window denoising autoencoder as a language model. 2) The reconstruction performance of stacked TINAs. 3) The performance of the deep neural network pre-trained by stacked TINAs on Chinese word segmentation.

We use the dataset of Chinese word segmentation bakeoff 2005. There are four different annotated corpus in this dataset, and we use the two in simplified Chinese, namely PKU and MSR dataset. All corpus have already been split into training data and test data [5].

[1] Our source code is available at: `https://github.com/eirikrwu/tinybrain`

Table 1. Comparison of log rank score of different language models. Classical 3-gram, 5-gram models (with Katz backoff), and a neural language model trained with a margin-based loss are compared to a TINA model with 1 hidden layer. The TINA model has the best performance in terms of the log rank score.

Language Model	Log Rank Score
3-Gram (Katz backoff)	2.54
5-Gram (Katz backoff)	2.53
NLM with Margin Loss	2.48
TINA with 1 Hidden Layer	**2.44**

5.1 Text Window Denoising Autoencoder as a Language Model

In the first experiment, we demonstrate the performance of text window denoising autoencoder as a language model. It can also be seen as the reconstruction performance of TINA. The TINA model we trained for this experiment uses a 100-dimensional character embeddings, and only one hidden layer of 300 units. The text window size is 11, which means that we're training the model to predict the character given 5 characters before and 5 after. The corpus that we use in this experiment is the PKU dataset of Chinese word segmentation bakeoff 2005.

We measure the error of the model by log of the rank of the predicted word [4]. We ask our model to output a probability for every character in the dictionary given a text window context, and then we compute the rank of the probability of the correct character. We average the log of the rank over all the text windows in test data as the final log rank score.

We compare our TINA model with the classical 3-gram and 5-gram models, which are trained using Katz backoff [8]. We also trained a neural language model which has the same configuration as our TINA model, except using a margin-based loss function, which is a commonly used loss function in English deep NLP [4]. The results are shown in Table 1. Our TINA model has the best performance in terms of the log rank score.

5.2 Stacking

In the second experiment, we build a deep architecture by stacking TINA models. We use the PKU corpus as in the above experiment. The TINA models we trained in this experiment use a 50-dimensional character embeddings. The text window size is 5. We first train a TINA model with a single hidden layer that has 300 units. Then we gradually add more hidden layers that have the same size and activation function as the first hidden layer, and train them along the way. The log rank score of all the TINA models with different depth is shown in Table 2.

The results show that the performance in terms of log rank score keeps getting better as more hidden layers are added. These deep TINA models can be seen as

Table 2. Comparison of stacked TINA models with different number of hidden layers. The performance in terms of log rank score keeps getting better as the model getting deeper.

Number of Hidden Layers	Log Rank Score
1 Hidden Layers	2.61
2 Hidden Layers	2.52
3 Hidden Layers	**2.45**

pre-trained deep neural networks that can extract highly representational features (that's why the log rank keeps getting lower as the model getting deeper). In the next experiment, we'll use them to build deep neural networks for specific tasks to show that these features are indeed useful.

5.3 Chinese Word Segmentation

In the third experiment, we demonstrate the performance of stacking TINA models as a pre-training method for deep neural networks for Chinese word segmentation task. We use both PKU and MSR dataset of Chinese word segmentation bakeoff 2005.

We use character embeddings of size 50. When pre-trained character embeddings are needed, we use the ones trained in the first experiment. We use a Viterbi-like decoding algorithm to retrieve the most likely valid tag sequence after the per character tag probability has been estimated [9]. We compare various model configurations, as well as different training methods.

Final results are shown in Table 3. We first train a neural network with a single hidden layer of size 300, and with all parameters initialized randomly. This basic model achieves F1 score of 92.6% on the PKU dataset. We then train neural networks with three hidden layers of size 300, with random hidden layer initialization and TINA pre-training respectively. Additionally, we pre-train the hidden layers by fixing character embeddings and using a normal denoising autoencoder with 25% masking noise [12]. The results show that compared to random initialization, TINA pre-training decreases the F1 error rate by 11.9% on the PKU dataset. Compared to the fixing character embeddings and pre-training with standard denoising autoencoder method, the F1 error rate is decreased by 7.8% on the PKU dataset. We further add one more hidden layer of size 300 with TINA pre-training, and the F1 error rate further drops 3.4%. We get similar results on the MSR dataset. TINA pre-training decreases the F1 error rate of the three-hidden-layer model by 5.3% compared to random initialization, 4.2% compared to standard denoising autoencoder. However, on the MSR dataset no further improvement is observed after adding one more TINA pre-trained hidden layer. We believe that this is because the MSR dataset is "simpler" than PKU dataset (notice the higher baseline performance).

Table 3. Word segmentation performance of different model configurations and training methods. "50CE" denote 50-dimensional character embeddings. "xL * yU" denote neural networks with x hidden layers of size y. "r" denote random initialization. "p" denote pre-trained character embeddings. "f" denote pre-trained with normal denoising autoencoders on top of fixed character embeddings. "TINA" denote pre-trained with TINA. Deep neural networks pre-trained by TINA have the best performance.

Dataset	Model	Precision	$\text{Recall}_{\text{OOV}}$	$\text{Recall}_{\text{IV}}$	F1
PKU	Baseline	83.6%	5.9%	95.6%	86.9%
	50CE(r) + 1L * 300U(r)	93.5%	75.0%	92.7%	92.6%
	50CE(p) + 1L * 300U(r)	93.7%	75.9%	93.7%	93.2%
	50CE(p) + 3L * 300U(r)	93.7%	76.0%	93.9%	93.3%
	50CE(p) + 3L * 300U(f)	93.7%	76.3%	94.6%	93.6%
	50CE(p) + 3L * 300U(TINA)	94.4%	77.9%	94.8%	94.1%
	50CE(p) + 4L * 300U(TINA)	94.6%	76.6%	95.0%	**94.3**%
MSR	Baseline	91.2%	0%	98.1%	93.3%
	50CE(r) + 1L * 300U(r)	94.5%	64.0%	95.1%	94.4%
	50CE(p) + 1L * 300U(r)	95.1%	63.6%	96.1%	95.2%
	50CE(p) + 3L * 300U(r)	95.0%	63.9%	96.0%	95.1%
	50CE(p) + 3L * 300U(f)	95.2%	64.4%	96.0%	95.2%
	50CE(p) + 3L * 300U(TINA)	95.7%	65.0%	96.4%	95.6%
	50CE(p) + 4L * 300U(TINA)	95.6%	64.9%	96.4%	**95.6**%

6 Related Works

Existing approaches for basic Chinese NLP tasks are almost all based on linear classifiers like hidden markov model or conditional random fields on simple binary word observation features [13][18]. These methods often rely on task specific manual feature design, therefore poses the risk of over-engineering. More recently, semi-supervised methods that can leverage unlabelled corpus, as well as joint training methods that integrate multiple sequence tagging tasks into one system have been proposed [7][14][16][17].

Due to the feature-oriented pre-training process, deep learning methods do not require any manual feature design. One can think of a deep neural network as two parts: all the hidden layers is a feature extractor whose sole purpose is to extract high lever features from data, and the output layer is a simple (even liner) classifier that leverages the high level features extracted from the bottom layers to complete the task. Researchers have found that it helps a lot in a non-convex optimization problem if the model has already been put to a position near the optimal before the gradient based searching starts [3]. So with a proper pre-training that concentrates on feature extraction, it puts our model near the optimal position. Denoising autoencoder [12] and restricted boltzmann machine [10] are two commonly used methods for feature-oriented pre-training.

In the pursuit of a word representation that's more suitable for deep neural network than simple one-hot vectors, Bengio et al. firstly invented the English

word embeddings and neural language model [2]. Like most English deep learning NLP methods, our Chinese NLP architecture is also a follow up work on Bengio's work, so it's very similar to English deep learning NLP methods. Basically, we simply change the model input from window of the English words to Chinese characters. However, we use a different method to train the Chinese character embeddings. In addition, we proposed a method that relates to denoising autoencoder to pre-train deep neural networks for Chinese NLP tasks.

Collobert et al.'s training method for English word embeddings [4] is similar to our training method for Chinese character embeddings. However, they use a margin-based loss to train their model. Since there are too many words in English, a margin-based loss criterion will be a lot easier to optimize. We have less characters in Chinese, so we're able to use a better criterion. We choose the likelihood loss instead. Also, they did not explain why training a neural language model is a good way of pre-training the word embeddings.

Recently, researchers have done some work to apply deep learning methods to Chinese NLP. Yang et al. proposed a bilingual English-Chinese word alignment approach based on deep neural network [15]. To our best knowledge, we are the first to apply deep learning methods to Chinese word segmentation.

7 Conclusion and Future Works

We have proposed a deep neural network model: text window denoising autoencoder, as the building block to build deep architectures for Chinese word segmentation. This is a fundamentally different approach, which does not require any linguistic knowledge or manual feature design. We demonstrated that deep neural networks for Chinese word segmentation can be effectively trained with this model. We built a Chinese word segmentation system with TINA, and achieved good performance.

Although we've achieved improvements with a TINA pre-trained deep architecture, for now our best model still under perform the state of the art models on both dataset [13]. However, We think our method shows great potential. We've only tested a few possible model configurations due to time and resource limit, and it's very likely that better performance could be achieved by simply tweaking our model configuration (larger character embedding dimension, more hidden units, etc). We leave that as a future work. There also exists many tricks that can significantly boost the performance of deep neural network, among which a particular method called dropout training [6] shows the best promise. We will try to apply these methods to our model in the future.

References

1. Arisoy, E., Sainath, T.N., Kingsbury, B., Ramabhadran, B.: Deep neural network language models. In: Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT, pp. 20–28. Association for Computational Linguistics (2012)

2. Bengio, Y., Ducharme, R., Vincent, P.: A neural probabilistic language model. Advances in Neural Information Processing Systems, 932–938 (2001)
3. Bengio, Y.: Learning deep architectures for AI. Foundations and Trends® in Machine Learning 2(1), 1–127 (2009)
4. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. The Journal of Machine Learning Research 12, 2493–2537 (2011)
5. Emerson, T.: The second international chinese word segmentation bakeoff. In: Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing, vol. 133 (2005)
6. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012)
7. Jiang, W.B., Sun, M., Lv, Y.J., Yang, Y.T., Liu, Q.: Discriminative Learning with Natural Annotations: Word Segmentation as a Case Study. In: 51st Annual Meeting of the Association for Computational Linguistics (2013)
8. Katz, S.M.: Estimation of probabilities from sparse data for the language model component of a speech recogniser. IEEE Transactions on Acoustics, Speech, and Signal Processing 35(3), 400–401 (1987)
9. Low, J.K., Ng, H.T., Guo, W.: A maximum entropy approach to Chinese word segmentation. In: Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing, vol. 1612164 (2005)
10. Salakhutdinov, R., Hinton, G.E.: Deep boltzmann machines. In: Proceedings of the International Conference on Artificial Intelligence and Statistics, vol. 5(2), pp. 448–455. MIT Press, Cambridge (2009)
11. Socher, R., Pennington, J., Huang, E.H., Ng, A.Y., Manning, C.D.: Semi-supervised recursive autoencoders for predicting sentiment distributions. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 151–161. Association for Computational Linguistics (2011)
12. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. The Journal of Machine Learning Research 11, 3371–3408 (2010)
13. Wang, K., Zong, C., Su, K.Y.: Integrating Generative and Discriminative Character-Based Models for Chinese Word Segmentation. ACM Transactions on Asian Language Information Processing 11(2), 7 (2012)
14. Wang, Z.G., Zong, C.Q., Xue, N.W.: A Lattice-based Framework for Joint Chinese Word Segmentation, POS Tagging and Parsing. In: 51st Annual Meeting of the Association for Computational Linguistics (2013)
15. Yang, N., Liu, S.J., Li, M., Zhou, M., Yu, N.H.: Word Alignment Modeling with Context Dependent Deep Neural Network. In: 51st Annual Meeting of the Association for Computational Linguistics (2013)
16. Zeng, X.D., Wong, F.D., Chao, S.L., Trancoso, I.: Co-regularizing character-based and word-based models for semi-supervised Chinese word segmentation. In: 51st Annual Meeting of the Association for Computational Linguistics (2013)
17. Zhang, M., Zhang, Y., Che, W.X., Liu, T.: Chinese Parsing Exploiting Characters. In: 51st Annual Meeting of the Association for Computational Linguistics (2013)
18. Zhao, H., Huang, C.N., Li, M.: An improved Chinese word segmentation system with conditional random field. In: Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing, vol. 1082117, Sydney (2006)