Deep Learning for Character-based Information Extraction

Yanjun Qi[1], Sujatha Das G[2], Ronan Collobert[3], and Jason Weston[4]

[1] Department of Computer Science, University of Virginia, yanjun@virginia.edu
Machine Learning Department, NEC Labs America
[2] Computer Science Department, Penn State University, sujatha.das@gmail.com
[3] IDIAP Research Institute, Switzerland, ronan@collobert.com
[4] Google, New York USA, jaseweston@gmail.com

Abstract. In this paper we introduce a deep neural network architecture to perform information extraction on character-based sequences, e.g. named-entity recognition on Chinese text or secondary-structure detection on protein sequences. With a task-independent architecture, the deep network relies only on simple character-based features, which obviates the need for task-specific feature engineering. The proposed discriminative framework includes three important strategies, (1) a deep learning module mapping characters to vector representations is included to capture the semantic relationship between characters; (2) abundant online sequences (unlabeled) are utilized to improve the vector representation through semi-supervised learning; and (3) the constraints of spatial dependency among output labels are modeled explicitly in the deep architecture. The experiments on four benchmark datasets have demonstrated that, the proposed architecture consistently leads to the state-of-the-art performance.

1 Introduction

In this paper, we focus on the information extraction (IE) tasks which aim to automatically extract information about pre-specified types of events, entities or relationships from unstructured or semi-structured machine-readable documents of sequences. The sequence data we target in this paper refers to the general family of character-based natural language strings. For instance, it could be Chinese text or Japanese text in which no space exists to mark word boundary between characters. Not limited to natural languages, the target sequence could also be a protein sequence which describes the primary structure of a protein using a linear string of amino acid letters.

Several IE tasks have been identified as important for end applications such as information retrieval or text summarization. For instance, the syntactic-level part-of-speech (POS) tagging, the semantic-level named entity extraction (NER) and the word segmentation (WS) tasks are the fundamental building blocks (Table 1) for processing Chinese [6]. Similarly, the functional tagging of each amino acid on the protein sequences is the core endeavor of computational biology. One classic task, secondary structure (SS) prediction [7] aims to predict each amino acid's secondary structure label which provides useful intermediate information for a protein's three-dimensional structure. All these tasks could be treated as

Characters	克	林	顿	总	统	前	往	中	东
WS	B	I	E	B	E	B	E	B	E
POS	B-NR	I-NR	E-NR	B-NN	E-NN	B-VV	E-VV	B-NR	E-NR
NER	B-PER	I-PER	E-PER	O	O	O	O	B-LOC	E-LOC

Table 1. An example of information extraction tagging on a sample sequence of Chinese text. Three character-based IE tasks are included, (1) WS: word segmentation; (2) POS: part-of-speech tagging and (3) NER: name entity recognition.

a labeling of each atomic element (e.g. Chinese character, or amino acid) in the sequence, into one of the multiple classes for the task of interest.

Differently from Romance language with words as the basic unit, Chinese text has no space to mark word boundary between characters. Tagging Chinese through assigning labels to characters (Table 1) has been shown to be a simple but effective formulation [9]. While early studies were mostly based on hand-crafted rules, recent systems have used supervised machine learning techniques such as, Hidden Markov Models (HMM), Maximum Entropy (ME) Models, Support Vector Machines (SVM), and Conditional Random Fields (CRF). The top systems from SIGHAN bakeoff competitions [6] were mostly based on the CRF and ME models. Analogous to Chinese text, the primary representation of a protein sequence includes a linear string of letters where each letter represents a certain amino acid and the string has no special letter to mark the boundary between functional segments. Many automated SS prediction methods have been described in the scientific literature [7], including for instance, neural networks, HMM and dynamic Bayesian networks. However, the above-mentioned systems mostly relied on rich sets of task-specific and hand-crafted features, which require time-consuming feature engineering and are hard to adapt to similar tasks.

Lately, "deep learning" [4] grows to bring in a lot of attentions and has won many pattern recognition contests. Our paper is motivated by a recent work from Collobert et al. [2] who demonstrated that a unified deep neural network architecture provides the state-of-art performance on multiple English Natural Language Processing (NLP) tasks using simple task-independent features. We adapt and modify this architecture to provide a unified framework for character-based IE tagging tasks by learning a hierarchy of representations given very basic inputs of character features. Three important components are employed in our framework, including *vector representation learning* of characters, semi-supervised *language modeling* and sentence-level training of output label dependencies.

2 Method

The proposed framework provides a unified end-to-end system that, given a string of characters, it provides several layers of feature extractions and predicts labels for each character. Table 1 provides a sample sequence of Chinese text and three sample IE outputs. Features relevant to the target IE task are learned automatically by backpropagation in the deep layers of the network model.

2.1 Learning Character to Vector Representation

The deep neural network is characterized by two specialized layers: (1) a character embedding layer and (2) a segment/window feature extraction layer, followed

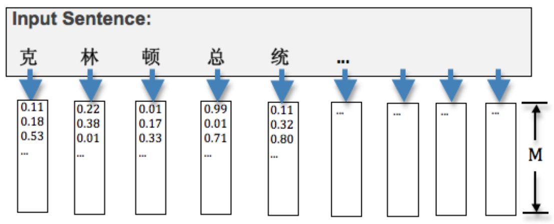


Fig. 1. An embedding module to learn vector representations for each input character.

by a series of classical neural network layers. The whole neural network architecture is displayed as Figure S1 in Supplementary [1] . The very first layer projects each character into a real-valued vector, in a M-dimensional latent embedding space (Figure 1). Here M is a hyperparameter to be chosen by the user. Within a finite character dictionary $\mathcal{D}$, each character $c_i \in \mathcal{D}$ is embedded into the feature space using a $M \times |\mathcal{D}|$ projection matrix W, such that W_{c_i}, the column vector of W at the index of c_i, is the vector representation corresponding to character c_i. Thus, using the first layer of this architecture, an input character sequence $\{c_1, c_2, \ldots c_n\}$ is transformed into a series of real valued vectors $\{W_{c_1}, W_{c_2}, \ldots W_{c_n}\}$. The parameters of the projection matrix W are learned automatically as part of the neural network training through back-propagation. The encoding weights in W are randomly initialized from a centered, uniform distribution. The resulting embedding after training is optimal for our target IE task, in the sense that it optimizes the objective cost on the training set for the specific task. Essentially this first embedding module is a special convolution neural network layer which applies the same projection W on each character in $\mathcal{D}$ regardless the position of characters in text sequences.

The second layer performs a sliding window operation on the character sequence which aggregates the output of the first layer into blocks corresponding to a fixed window of size k. To label a complete sentence, we slide the window along the sentence, labeling one character at a time. This means each character in the sequence is described through itself and its neighboring characters. The remaining layers comprise a standard, fully connected multi-layer perceptron network with L layers of hidden units. Each hidden layer l learns to map its input to a hidden feature space (through parameter U^l), and the last layer V utilizes a softmax function to map its input to the output tag (label) space. This layer's outputs are ensured to be positive and sum to 1, thus we can interpret the outputs of the whole network as probabilities for each class [2]. Essentially the network includes the following set of parameters to be estimated, $\Theta = \{W, \boldsymbol{U^1}, \boldsymbol{U^2}, ..., \boldsymbol{U^{L-1}}, \boldsymbol{U^L}\}$.

Assuming we are given a set of training examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1\ldots N}$, $\boldsymbol{x}_n$ represents a window of characters, y_n describes the label of middle character in $\boldsymbol{x}_n$, and $f(\boldsymbol{x}_n)$ represents the predicted output of $\boldsymbol{x}_n$ by the whole network. Then the deep model is trained to find the best Θ by minimizing the negative log-likelihood (NLL) loss, i.e. $E(\Theta) := \sum_{n=1}^{N} NLL_{\Theta}(f(\boldsymbol{x}_n), y_n)$ over the training samples. $E(\Theta)$ is optimized using backpropagation through Stochastic gradient descent (SGD). In SGD, a random example $(\boldsymbol{x}, y)$ is sampled from the training set and then a gradient descent step is applied to update the network parameter Θ as: $\Theta \leftarrow \Theta - \lambda \frac{\partial E(\Theta, \boldsymbol{x}, y)}{\partial \Theta}$, where λ is the learning rate hyperparameter.

2.2 Improving Representation Learning with Unlabeled Sequences

Manually labeling character-based sequences, i.e. to obtain tag label for each character, could be quite time-consuming, since it requires very detailed annotation on the character-level (Table 1). Information hiddlen inside unlabeled text has been shown to be helpful for supervised IE tagging [2]. Thus, we employ to add a so-called semi-supervised "language modeling" (LM) task [2] using unlabeled character sequences (abundant from internet, e.g. Wikipedia Chinese corpus, or Swissprot protein sequence database).

This step relies on the key observation that individual characters carry significant semantic information hidden in the unlabeled data. This translates to the basic idea that the target LM task will learn to force two character pieces with similar semantic meanings to have closer representations and two pieces with dissimilar meanings to have distant representations in a learned feature space. Intuitively, all length-k character windows from an unlabeled corpus could be labeled as positive examples, and negative fragments are generated by randomly substituting the middle character in each window. Thus, each pair of these (positive, negative) segments build up a corpus of dissimilar character pieces. A deep network, similar as the one for IE tagging, is then built to learn the representations based on the pairs of dissimilar segments from the above corpus. Again, the character embedding layer and all parameters of the subsequent network layers are automatically trained by backpropagation. Unlike supervised IE tagging using NLL loss, the LM task is trained with a margin ranking cost: $\sum_{\mathbf{s} \in \mathcal{S}} \sum_{c \in \mathcal{D}} \max(0, 1 - f'(\mathbf{s}) + f'(\mathbf{s}^c))$ where $\mathcal{S}$ is the set of positive character segments, and $\mathbf{s}^c$ is a pseudo-negative segment where its middle character has been replaced by a random character c in the dictionary $\mathcal{D}$. $f'(\cdot)$ represents the output of the deep network architecture for LM task, which indicates a function that projects its input segment to a value output. The output scalar value describes how likely a given character segment exists in the unlabeled corpus in our setup. Essentially, we are learning the network weights to rank positive character segments above synthetic negative segments. Then we utilize a popular deep learning strategy—pretraining, to connect the unsupervised LM task and the supervised tagging tasks. That is, parameter Θ' learned from LM is used as the initialization for parameter Θ for supervised IE.

2.3 Discriminative Training of Sentence-Level Label Dependency

So far, our deep framework uses a labeling-per-character strategy without exploiting the dependencies among the targeted tag labels. This assumes that the output label for each position in a character-based sequence can be predicted independently from nearby positions. Empirically, this assumption fails largely for IE tagging. For example, clearly some tags are less likely to follow another set of tags (Table 1) (e.g. O tag is less likely to follow B-PER tag for NER), and certain groups of tags are highly likely to appear as neighbors (e.g. B-PER and E-PER for NER). Thus we models the dependencies between target tags through discriminative forward training [2]. Basically it scores a whole candidate sentence by aggregating the predicted scores of each involved character and also weighting the scores by a transition matrix $[A]_{i,j}$. This parameter $[A]_{i,j}$ captures the degree of likelihood for jumping from i to j tags in successive characters. Similar to Θ, weights of A are also automatically trained by backpropagation. The sentence tagging layer with A parameter is put on the top of the basic model $f(\cdot)$, achieving a revised deep model $f''(\cdot)$. The final tagger $f''(\cdot)$ is trained to find the best tag path which optimizes each whole training sentence's output score. The prediction of the tag path is efficiently implemented through a Viterbi-like algorithm (space limit; details in [2]).

3 Experiments

Data: We demonstrate the power of deep learning architecture on four benchmark data sets for character-based IE tagging, (1), WS on Chinese tree bank (CTB) data (LDC2007T36) [8]. (2), POS on the Chinese tree bank (CTB) data [8]; (3), NER on "CITYU" data from SIGHAN3 bakeoff[6]. (4), CB513, the standard protein SS benchmark, is used [3] for the SS evaluation. All data details of the four sets (e.g. sample size, encoding of output tags, et al.) are in Supplementary [1] due to page limitation.

Metric: For WS, we compute the character-level precision, recall, and equally weighted F-measure (only F1 reported). When evaluating POS and NER, we compute overall phrase-level precision, recall, and F1-measure (only F1 reported). For the protein SS task, we use the standard "Q-score", which represents the *accuracy* evaluated at the amino acid level.

Setup: The evaluation of WS and POS is carried through 10-folds cross validation. Following the same setup as (Zhang&Clark 2008) [10], we have evenly partitioned CTB into ten groups, and used nine groups for training and the rest for testing. The NER evaluation is carried through the benchmark provided train-test split. The SS evaluation follows the standard seven-folds cross validation on this set. The Torch [2] deep learning toolbox is used for coding the architecture.

Hyperparameter: Furthermore, the deep framework requires the specification of multiple hyperparameters. This includes the size k of the character window, the size h of the hidden layers, and the learning rate λ. We considered $k \in \{3, 5, 7, 9, 11, 13\}$, $h \in \{80, 100, 120, 150, 200\}$, and $\lambda \in \{0.001, 0.01, 0.03, 0.05, 0.1\}$. The parameter for the embedding layer, M, was chosen from $\{15, 30, 50, 70\}$. The hyperparameter selection was based on the cross-validation results on training. The semi-supervised language model was trained using the freely available Chinese wikipedia corpus and the swissprot protein sequence database.

Result Comparison: Table 2 summaries multiple variations of the proposed architecture and the performance comparison to the state-of-the-art systems. We systematically test the incremental combinations of multiple strategies: (1). "c1": character to vector embedding; (2). "lm": unsupervised "language model" of

Configuration / Task	**WS-Chinese**	**POS-Chinese**	**NER-Chinese**	**SS-Protein***
1. c1	94.73	86.74	80.61	74.5
2. c1+lm	95.57	86.93	81.79	74.8
3. c1+vit	95.38	88.41	85.81	77.6
4. c1+lm+vit	96.07	88.81	86.99	77.8
5. c1+lm+c2	95.98	88.48	83.51	~
6. c1+lm+c2+vit	**96.62**	89.39	87.24	~
7. c1+lm+c2+vit+ws	~	**93.27**	88.88	**80.3***
Previous Best	95.9 [10]	91.9 [10]	**89.00** [6]	80.0 [5]
Previous Second Best	95.1 [10]	91.3 [10]	88.61 [6]	~

Table 2. Performance comparison on four character-based IE tagging tasks. Different combinations of the proposed strategies are compared (in percentage %). "*": for SS task, we also add the classic PSI-Blast feature in all combinations. For the 7th setup, SS is added with a multitasking strategy [7], instead of "ws+c2".

character. (3). "vit": sentence-level label dependency; (4). "ws": word segmentation added as discrete features [2]; (5). "c2": another embedding extraction layer for learning vector representations for character bigrams. It is clear that the final system (with the combination of all strategies) has improved over the state-of-the-art performance on WS and POS and has achieved the state-of-the-art predictive level for NER and SS tasks (detailed discussion in [1]).

Conclusion: We have proposed a deep learning framework for character-based information extraction on Chinese and protein sequences. As a flexible and robust prediction system, this architecture has achieved the state-of-art performance on four benchmark data. Our methodology could easily be adapted to additional character-based tagging tasks, such as Japanese NER.

References

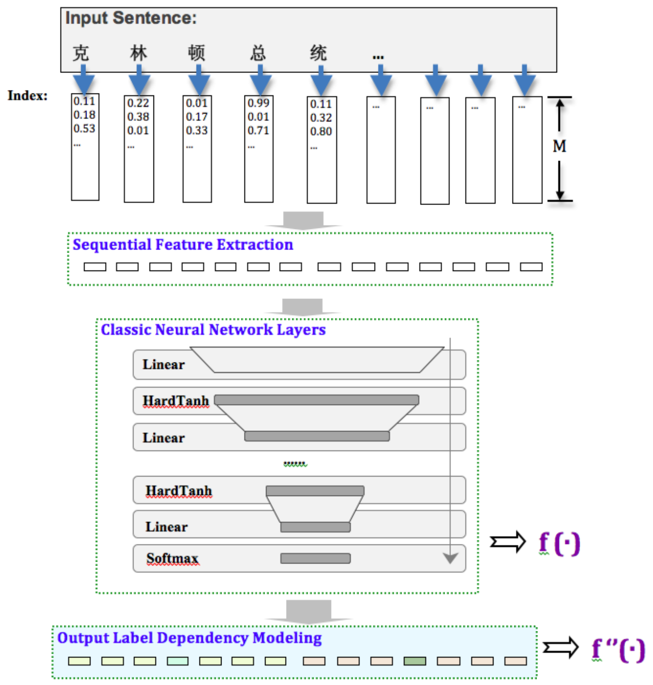
1. : Supplementary. `http://www.cs.cmu.edu/~qyj/zhSenna/` (Dec. 2013)
2. Collobert, R., Weston, J., Bottou, L., Michael, Kavukcuoglu, Kuksa: Natural language processing (almost) from scratch. JMLR **12** (2011) 2493–2537
3. Cuff, J.A., Barton, G.J.: Evaluation and improvement of multiple sequence methods for protein secondary structure prediction. Proteins **34** (1999) 508–519
4. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (2006) 504–507
5. Kountouris, P., Hirst, J.D.: Prediction of backbone dihedral angles and protein secondary structure using support vector machines. BMC Bioinf. **10**(437) (2009)
6. Levow, G.A.: The third international chinese language processing bakeoff: Word segmentation and named entity recognition. In: Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing. Volume 117., Sydney: July (2006)
7. Qi, Y., Oja, M., Weston, J., Noble, W.S.: A unified multitask architecture for predicting local protein properties. PLoS ONE **7**(3) (03 2012) e32235
8. Xue, N., Xia, F., Chiou, F.D., Palmer, M.: Penn chinese treebank: Phrase structure annotation of a large corpus. Natural Language Engineering **11** (2005) 207–238
9. Xue, N., et al.: Chinese word segmentation as character tagging. Computational Linguistics and Chinese Language Processing **8**(1) (2003) 29–48
10. Zhang, Y., Clark, S.: Joint word segmentation and pos tagging using a single perceptron. In: Proceedings of the 46th Annual Meeting of ACL. (2008) 888–896

Supplementary for ECIR 2014 Paper “Deep Learning for Character-based Information Extraction on Chinese and Protein Sequence”

Yanjun Qi Sujatha Das G Ronan Collobert Jason Weston

1 System

Figure 1: The basic deep learning system for character-based IE tagging.



2 Experiments

2.1 Data Sets:

Table 1 summarizes some statistics of the datasets we used in experiments. (1) The CTB data we used for WS and POS is from Chinese Treebank 6.0 (LDC2007T36), released in 2007, encompasses 2,036 text files, containing 28,295 sentences, 781,351 words and about $1.3M$ Chinese characters. (2). The CITYU NER data was from SIGHAN3 [6], which includes around $1.8M$ NE-labeled Chinese characters. (3). The CB513 data for SS task consists of 513 unrelated proteins with known 3D structure. Totally the CB513 includes about $84k$ amino acid characters labeled with SS target tags [8].

Table 1: Summary of datasets used in experiment

Dataset/Task	#Chars	#UniqueChars	#Sent
POS(CTB)	1,288,840	4,447	28,295
WS(CTB)	1,287,159	4,696	28,295
NER(CITYU)	1,816,417	4,678	43,734
SS (CB513)	83,707	25	497

The setup for WS and POS tasks used the 10-folds cross validation (CV), following the same configuration in (Zhang&Clark 2008) [9]. We have evenly partitioned CTB into ten groups, and used nine groups for training and the rest for testing. The partition of CTB is provided in Table 2 The train-test split for NER task was from SIGHAN3. The setup for SS task is the same as the standard seven-folds cross validation on CB513 data. The evaluation metrics are averaged per cross-validation test fold in all CV setting.

For the first three tasks related to Chinese text, we construct a list of characters from all three Chinese datasets to form a dictionary of 11951 characters. We convert all non-Chinese characters (such as English words) to lowercase and represented all numeric tokens by “NUMBER”. As shown in the performance comparison table, we also try to learn the vector representations for the character bigrams for Chinese IE. The dictionary of character bigrams is constructed similarly as the character unigram dictionary. We just

Table 2: The ten folds uniform partition of CTB (LDC2007T36) for POS and WS tasks.

Fold Index	Start	End
1	0001	0203
2	0204	0525
3	0526	0767
4	0768	1054
5	1055	2126
6	2127	2329
7	2330	2532
8	2533	2735
9	2736	2938
10	2939	3145

rank all the possible bigrams in the three datasets and use the top (most frequent) $30,000$ from the list. For the last SS task, it uses a small dictionary containing only the possible amino acid letters.

2.2 Output Tag Label:

Table 3: Summary of Output Labels

Task	#Labels	Example Tags
NER	14	B-LOC, I-LOC, S-ORG
POS	107	S-NR, B-NN, E-VV, B-DT
WS	5	B, I, E,S
CB513	4	H, B, C

Table 3 gives the number (second column) of target labels used for each task. We employ the IOBES coding scheme for WS, POS and NER tasks. This makes WS task has 5 different tag classes, NER has 14 different tagging labels, and similarly, POS tagging has 107 different target classes to predict for. The label of CB513 data utilizes a 3-letter alphabet $\{H, B, C\}$ in the standard way (H = alpha helix, B = residue in isolated beta bridge and C = Coil).

2.3 Hyperparameter:

The deep framework requires the specification of multiple hyperparameters. This includes the size k of the character window, the size h of the hidden layers, and the learning rate λ. We considered $k \in \{3, 5, 7, 9, 11, 13\}$, $h \in \{80, 100, 120, 150, 200\}$, and $\lambda \in \{0.001, 0.01, 0.03, 0.05, 0.1\}$. The parameter for the embedding layer, M, was chosen from $\{15, 30, 50, 70\}$. The hyperparameter selection was based on the cross-validation results on training. The semi-supervised language model was trained using the freely available Chinese wikipedia corpus [1] and the swissprot protein sequence database [2]. For the three tasks related to Chinese IE, we obtained the best results through setting $k = 3$, $h = 150$, the embedding size $M = 50$, the learning rate for $f(\cdot)$ as 0.05 and the learning rate for the output dependency modeling as 0.01. For SS task, the preferred hyperparameter setup is $k = 13$, $M = 15$, $h = 85$.

The software and methods were implemented using the Torch5 [3] machine learning library. Torch is implemented in C and Lua scripting language. In our basic model, for simplicity, we restrict the classical NN part of our deep neural network to one single hidden linear layer and one output linear layer.

2.4 Result Comparison

As pointed out by Table 2 in the main draft, the proposed architecture includes three strategies, using which we assume to improve the ability of tagging character sequences through, (1) learning embedding representation for characters in the deep framework, (2) including the semi-supervised "language model" ("lm") task, and (3) discriminative training of sentence-level label dependency (termed as "vit", since the prediction step uses viterbi algorithm). Accordingly, we systematically test the incremental combinations of these strategies the Table 2 of the main draft . The first configuration "c1") provides the performance of the most basic setup where only the embedding of character unigrams is utilized. The second configure "c1+lm" adds "language model" component on the top of "c1" case, which clearly improves the tagging ability on all tasks. For instance, WS tagging gets improved F1 measure from 94.73% to 95.57% (about 1 percent increasing). The third configuration "c1+vit" describes the result when combining the unigram embedding and discriminative training of tag dependencies (using "vit" term to label this component). The "tag dependencies" strategy dramatically improves the tagging ability on all three tasks. Especially on Chinese NER, the "c1+vit" configuration achieves the

F1 score 85.81%, a big jump from 80.61% when using "c1" embedding alone. Furthermore, the 4_{th} configuration "c1+lm+vit" improves even more when combining the three strategies of character embedding, language modeling and tag dependency training. In the fifth "c1+lm + c2" and the sixth configuration "c1+lm + c2+vit", we added another embedding feature extraction layer for representing the character bigrams (the same operation as unigram embedding, just on each character bigram). Compared to the state-of-the-art results in the last two rows, our sixth model has already beats the best WS result provided in [9] with about 0.7% improvements. For POS task, our final model is the seventh configuration "c1+c2+lm+vit+ws" which combines all the three functional components plus using word-segmentation as features. Note that the word segmentation is added as one extra new discrete features (i.e. again with a new embedding layer). This clearly beats the top system [9] (with the same 10folds CV setup) in the literature. For the NER task, the seventh configuration beats the second top system, and is slightly lower than the best ranked system from SIGHAN3 [6]. For SS, the seventh configure does not have WS features (of course) and also does not use the character bigram features. Instead, it adds a multitasking strategy which has been proposed in [7]. The extra strategy clearly improves our system even more, which makes SS predictions reach the state-of-the-art performance [5].

3 Discussion

We have described a deep learning framework for multiple character-based information extraction tasks. Our work experimentally proves the power of this architecture and its multiple important techniques. Lastly, we just realize that there is a recent paper from Zheng et al. [10] that uses a very similar method as our paper. The method is almost the same as Collobert et al [4]. Different from Collobert et al [4] using words as the basic units, our paper and Zheng et al. [10] use character as the primary basic unit. Zheng et al. [10] focused primarily on Chinese POS and WS. Different from [10], our framework has been further extended to Chinese NER and SS tasks. Our experimental results also improve the state-of-art on POS and WS.

References

[1] Chinese wikipedia, `http://zh.wikipedia.org`.

[2] swissprot protein sequence databas, `http://www.uniprot.org/downloads`.

[3] Torch. `http://torch5.sourceforge.net/`.

[4] Ronan Collobert, Jason Weston, Léon Bottou, Michael, Kavukcuoglu, and Kuksa. Natural language processing (almost) from scratch. *JMLR*, 12:2493–2537, 2011.

[5] P. Kountouris and J. D. Hirst. Prediction of backbone dihedral angles and protein secondary structure using support vector machines. *BMC Bioinf.*, 10(437), 2009.

[6] Gina-Anne Levow. The third international chinese language processing bakeoff: Word segmentation and named entity recognition. In *Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing*, volume 117. Sydney: July, 2006.

[7] Yanjun Qi, Merja Oja, Jason Weston, and William Stafford Noble. A unified multitask architecture for predicting local protein properties. *PLoS ONE*, 7(3):e32235, 03 2012.

[8] Burkhard Rost and Chris Sander. Combining evolutionary information and neural networks to predict protein secondary structure. *Proteins: Structure, Function, and Bioinformatics*, 19(1):55–72, 1994.

[9] Yue Zhang and Stephen Clark. Joint word segmentation and pos tagging using a single perceptron. In *Proceedings of the 46th Annual Meeting of ACL*, pages 888–896, 2008.

[10] Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. Deep learning for Chinese word segmentation and POS tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 647–657, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.