Neural Networks Leverage Corpus-wide Information for Part-of-speech Tagging

Yuta Tsuboi
IBM Research - Tokyo
yutat@jp.ibm.com

Abstract

We propose a neural network approach to benefit from the non-linearity of corpus-wide statistics for part-of-speech (POS) tagging. We investigated several types of corpus-wide information for the words, such as word embeddings and POS tag distributions. Since these statistics are encoded as dense continuous features, it is not trivial to combine these features comparing with sparse discrete features. Our tagger is designed as a combination of a linear model for discrete features and a feed-forward neural network that captures the non-linear interactions among the continuous features. By using several recent advances in the activation functions for neural networks, the proposed method marks new state-of-the-art accuracies for English POS tagging tasks.

1 Introduction

Almost all of the approaches to NLP tasks such as part-of-speech tagging and syntactic parsing mainly use sparse discrete features to represent local information such as word surfaces in a size-limited window. The non-linearity of those discrete features is often used in many NLP tasks since the simple conjunction (AND) of discrete features represents the co-occurrence of the features and is intuitively understandable. In addition, the thresholding of these combinatorial features by simple counts effectively suppresses the combinatorial increase of the parameters. At the same time, although global information had also been used in several reports (Nakagawa and Matsumoto, 2006; Huang and Yates, 2009; Turian et al., 2010; Schnabel and Schütze, 2014), the non-linear interactions of these features were not well investigated since these features are often dense continuous features and the explicit non-linear expansions are counterintuitive and drastically increase the number of the model parameters. In our work, we investigate neural networks used to represent the non-linearity of global information for POS tagging in a compact way.

We focus on four kinds of corpus-wide information: (1) *word embeddings*, (2) POS tag distributions, (3) *supertag* distributions, and (4) context word distributions. All of them are continuous dense features and we use a feed-forward neural network to exploit the non-linearity of these features. Although all of them except (3) have been used for POS tagging in previous work (Nakamura et al., 1990; Schmid, 1994; Schnabel and Schütze, 2014; Huang and Yates, 2009), we propose a neural network approach to capture the non-linear interactions of these features. By feeding these features into neural networks as an input vector, we can expect our tagger can handle not only the non-linearity of the N-grams of the same kinds of features but also the non-linear interactions among the different kind of features.

Our tagger combines a linear model using sparse high-dimensional features and a neural network using continuous dense features. Although Collobert et al. (2011) seeks to solve NLP tasks without depending on the feature engineering of conventional NLP methods, our architecture is more practical because it integrates the neural networks into a well-tuned conventional method. Thus, our tagger enjoys both the manually explored combinations of discrete features and the automatically learned non-linearity of the continuous features. We also studied some of the newer activation functions: Rectified Linear Units (Nair and Hinton, 2010), Maxout networks (Goodfellow et al., 2013), and L_p-pooling (Gulcehre et al., 2014; Zhang et al., 2014).

Deep neural networks have been a hot topic in many application areas such as computer vision and voice recognition. However, although neural networks show state-of-the-art results on a few semantic tasks (Zhila et al., 2013; Socher et al., 2013; Socher et al., 2011), neural network approaches have not performed better than the state-of-the-art systems for traditional syntactic tasks. Our neural tagger shows state-of-the-art results: 97.51% accuracy in the standard benchmark on the Penn Treebank (Marcus et al., 1993) and 98.02% accuracy in POS tagging on CoNLL2009 (Hajič et al., 2009). In our experiments, we found that the selection of the activation functions led to large differences in the tagging accuracies. We also observed that the POS tags of the words are effectively clustered by the hidden activations of the intermediate layer. This observation is evidence that the neural network can find good representations for POS tagging.

The remainder of this paper is organized as follows. Section 2 introduces our deterministic tagger and its learning algorithm. Section 3 describes the continuous features that represent corpus-wide information and Section 4 is about the neural network we used. Section 5 presents our empirical study of the effects of corpus-wide information and neural networks on English POS tagging tasks. Section 6 describes related work, and Section 7 concludes and suggests items for future work.

2 Transition-based tagging

Our tagging model is a deterministic tagger based on Choi and Palmer (2012), which is a one-pass, left-to-right tagging algorithm that uses well-tuned binary features.

Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_T) \in X^T$ be an input token sequence of length T and $\boldsymbol{y} = (y_1, y_2, \ldots, y_T) \in Y^T$ be a corresponding POS tag sequence of $\boldsymbol{x}$. We denote the predicted tags by a tagger as $\hat{\boldsymbol{y}}$ and the subsequence from r to t as $\boldsymbol{y}_r^t$. The prediction of the t-th tag is deterministically done by the classifier:

$$\hat{y}_t = \operatorname*{argmax}_{y \in Y} f_{\boldsymbol{\theta}}(\boldsymbol{z}_t, y), \tag{1}$$

where $f_{\boldsymbol{\theta}}$ is a scoring function with arbitrary parameters, $\boldsymbol{\theta} \in \mathbb{R}^d$, that are to be learned and $\boldsymbol{z}_t$ is an arbitrary feature representation of the t-th position using $\boldsymbol{x}$ and $\hat{\boldsymbol{y}}_1^{t-1}$ which is the prediction history of the previous tokens.

We extend Choi and Palmer (2012) in three ways: (1) an online SVM learning algorithm with L_1 and L_2 regularization, (2) continuous features for corpus-wide information, and (3) the composite function of a linear model for discrete features and a non-linear model for continuous features. Since (2) and (3) are the main topics of this paper, they are explained in detail in Sections 3 and 4 and we describe only (1) here.

First, our learning algorithm trains a multi-class SVM with L_1 and L_2 regularization based on *Follow the Proximally Regularized Leader* (FTRL-Proximal) (McMahan, 2011). In the k-th iteration, the parameter update is done by

$$\boldsymbol{\theta}^k = \operatorname*{argmin}_{\boldsymbol{\theta}} \sum_{l=1}^{k} \left(\boldsymbol{g}^l \cdot \boldsymbol{\theta} + \frac{1}{2\boldsymbol{\eta}^l} \left|\left| \boldsymbol{\theta} - \boldsymbol{\theta}^l \right|\right|_2^2 \right) + R(\boldsymbol{\theta}),$$

where $\boldsymbol{g}^k \in \mathbb{R}^d$ is a subgradient of the hinge loss function and $R(\boldsymbol{\theta}) = \boldsymbol{\lambda}_1 ||\boldsymbol{\theta}||_1 + \frac{\boldsymbol{\lambda}_2}{2} ||\boldsymbol{\theta}||_2^2$ is the composite function of the L_1 and L_2 regularization terms with hyper-parameters $\boldsymbol{\lambda}_1 \geq 0$ and $\boldsymbol{\lambda}_2 \geq 0$. To incorporate an adaptive learning rate scheduling, Adagrad (Duchi et al., 2010), we use per-coordinate learning rates for $\{i | 1 \leq i < d\}$:

$$\eta_i^k = \frac{\alpha_i}{\left(\beta_i + \sqrt{\sum_{l=1}^{k} (g_i^l)^2} \right)},$$

where $\alpha \geq 0$ and $\beta \geq 0$. Although the naive implementation may require $O(k)$ computation in the k-th iteration, FTRL-Proximal can be implemented efficiently by maintaining two length-d vectors, $\boldsymbol{m} = \sum_l^k \boldsymbol{g}^l - \frac{1}{2\boldsymbol{\eta}^l} \boldsymbol{\theta}^l$ and $\boldsymbol{n} = \sum_l^k (g_i^l)^2$ (McMahan et al., 2013).

Second, to overcome the error propagation problem, we train the classifier with a simple variant of the on-the-fly example generation algorithm from Goldberg and Nivre (2012). Since the scoring function refers to the prediction history, Choi and Palmer (2012) uses the gold POS tags, $\boldsymbol{y}_1^{t-1}$, to generate training examples, which means they assume all of the past decisions are correct. However, this causes error propagation problems, since each state depends on the history of the past decisions. Therefore, at the k-th iteration and the t-th position of the input sequence, we simply use the predictions of the previously learned classifiers to generate training examples, i.e.,

$$\hat{y}_{t-r} = \operatorname*{argmax}_{y \in Y} f_{\boldsymbol{\theta}_{k-r}}(\boldsymbol{z}_{t-r}, y)$$

for all $\{r | 1 \leq r < t - 1\}$. Although it is not theoretically justified, it empirically runs as a stochastic version of DAGGER (Ross et al., 2011) or SEARN (Daumé III et al., 2009) with the speed benefit of online learning.

Algorithm 1 Learning algorithm

```
function LEARN($\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{m}, \boldsymbol{n}, \boldsymbol{\theta}^k$)
    while ¬ stop do
        Select a random sentence $(\boldsymbol{x}, \boldsymbol{y})$
        for $t = 1$ to $T$ do
            $\boldsymbol{u}$=UPDATE($\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{m}, \boldsymbol{n}, \boldsymbol{\theta}^k$)
            $\hat{y_t} = \mathrm{argmax}_{y \in Y} f_{\boldsymbol{u}}(\boldsymbol{z}_t, y)$
            $\tilde{y} = \mathrm{argmax}_{y \neq y_t} f_{\boldsymbol{u}}(\boldsymbol{z}_t, , y)$
            if $f_{\boldsymbol{u}}(\boldsymbol{z}_t, y_t) - f_{\boldsymbol{u}}(\boldsymbol{z}_t, \tilde{y}) < 1$ then
                $\boldsymbol{g} = \partial_{\boldsymbol{u}} \ell(\boldsymbol{z}_t, y_t, \tilde{y})$ ▷ Subgradient
                For all $i \in I$ compute
                $\sigma_i = \left(\sqrt{n_i + g_i^2} - \sqrt{n_i}\right) / \alpha_i$
                $m_i \leftarrow m_i + g_i - \sigma_i u_i$
                $n_i \leftarrow n_i + g_i^2$
            end if
            $k \leftarrow k + 1$
        end for
    end while
    return $\boldsymbol{\theta}^k$
end function

function UPDATE($\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \boldsymbol{m}, \boldsymbol{n}, \boldsymbol{\theta}^k$)
    for $i \in I$ do
        $\theta_i^k = \begin{cases} 0 & \text{if } |m_i| \leq \lambda_1 \\ \frac{-m_i + \mathrm{sgn}(m_i)\lambda_1}{(\beta_i \lambda_2 + \sqrt{n_i})/\alpha_i + \lambda_2} & \text{otherwise} \end{cases}$
        $u_i \leftarrow \theta_i^k$
        if acceleration then
            $u_i \leftarrow \theta_i^k + \frac{k}{k+3}\left(\theta_i^k - \theta_i^{k-1}\right)$
        end if
    end for
    for $i \notin I$ do
        $u_i \leftarrow \theta_i^k \leftarrow \theta_i^{k-1}$
        ▷ Leaving all $\theta$ for inactive $i$ unchanged
    end for
    return $\boldsymbol{u}$
end function
```

Algorithm 1 summarizes our training process where $\ell(\boldsymbol{z}_t, y_t, \tilde{y}) := \max(0, 1 - f_{\boldsymbol{\theta}}(\boldsymbol{z}_t, y) + f_{\boldsymbol{\theta}}(\boldsymbol{z}_t, \tilde{y}))$ is the multi-class hinge loss (Crammer and Singer, 2001). I in Algorithm 1 is a set of parameter indexes that correspond to the non-zero features, so the update is sparse for sparse features. In addition, for the parameter update of the neural networks, we also use an accelerated proximal method (Parikh and Boyd, 2013), which is considered as a variant of the momentum methods (Sutskever et al., 2013). Although $\boldsymbol{u}$ and $\boldsymbol{\theta}$ are the same when the acceleration is not used, $\boldsymbol{u}$ in Algorithm 1 is an extrapolation step in the accelerated method. Although we do not focus on the learning algorithm in this work, the algorithm converges quite quickly and the speed is important because the neural network extension described later requires a hyper-parameter search which is computationally demanding.

3 Corpus-wide Information

Since typical discrete features indicate only the occurrence in a local context and do not convey corpus-wide statistics, we studied four kinds of continuous features for POS tagging to represent the corpus-wide information.

3.1 Word embeddings

Word embeddings, or distributed word representations, embed the words into a low-dimensional continuous space. Most of the neural network applications for NLP use word embeddings (Collobert et al., 2011; Socher et al., 2011; Zhila et al., 2013; Socher et al., 2013), and even for linear models, Turian et al. (2010) highlights the benefit of word embeddings on sequential labeling tasks.

In particular, in our experiments, we used two recently proposed algorithms, *word2vec* (Mikolov et al., 2013) and *glove* (Pennington et al., 2014), which are simple and scalable, although our method could use other word embeddings. Word2vec trains the word embeddings to predict the words surrounding each word, and glove trains the word embeddings to predict the logarithmic count of the surrounding words of each word. Thus, these embeddings can be seen as the distributed versions of the distributional features since the word vectors compactly represent the distribution of the context in which a word appears. We normalized the word embeddings to unit length and used the average vector of training vocabulary for the unknown tokens.

3.2 POS tag distribution

In a way similar to Schmid (1994), we use POS tag distribution over a training corpus. Each word is represented by a vector of length $|Y|$ in which the y-th element is the conditional probabilities with which that word gets the y-th POS tag. We also use the POS tag distributions of the affixes and spelling binary features used in Choi and Palmer (2012). We cite the definitions of these features.

1. Affix: $c_{:1}, c_{:2}, c_{:3}, c_{n:}, c_{n-1:}, c_{n-2:}, c_{n-3:}$ where c_* is a character string in a word. For example $c_{:2}$ is the prefix of length two of a word and $c_{n-1:}$ is the suffix of length two of a word.
2. Spelling: initial uppercase, all uppercase, all lowercase, contains 1/2+ capital(s) not at the beginning, contains a (period/number/hyphen).

The probabilities for a feature b is estimated with additive smoothing as

$$P(y|b) = \frac{C(b, y) + 1}{C(b) + |Y|}, \tag{2}$$

where $C(b)$ and $C(b, y)$ are the counts of b and co-occurrences of b and y, respectively. In addition, an extra dimension for sentence boundaries is added to the vector for word-forms. In total, the POS tag distributions for each word are encoded by a vector of dimension $|Y|+1+|Y|\times 14$ ($|Y|$ for lowercase simplified word-forms, 1 for sentence boundaries, $|Y| \times 7$ for affixes, and $|Y| \times 7$ for spellings).

3.3 Supertag distribution

We also use the distribution of *supertags* for dependency parsing. Supertags are lexical templates which are extracted from the syntactic dependency structures and suppertagging is often used for the pre-processing of a parsing task. Since the supertags encode rich syntactic information, we expect the supertag distribution of a word to also provide clues for the POS tagging. We used two types of supertags: One is the dependency relation label of the head of the word and the other is that of the dependents of the word. Following Ouchi et al. (2014), we added the relative position, left (L) or right (R), to the supertags. For example, a word has its dependents in the left direction with a label "nn" and in the right direction with a label "amod", so its supertag set for dependents is {"nn/L", "amod/R"}. A special supertag "NO-CHILD" is used for a word that has no dependent. Note that, although the Model 2 supertag set of Ouchi et al. (2014) is defined as the combination of head and dependent tags, we used them separately. The feature values for each word are defined in the same way as Equation 2 in Section 3.2. Since a word can have more than one dependent, the dependent supertag features are no longer multinomial distributions but we used them in that way. Note that, since the feature values are calculated using the tree annotations from training set, our tagger does not require any dependency parser at runtime.

3.4 Context word distribution

This is the simplest distributional features in which each word is represented by the distributions of its left and right neighbors. Although the context word distribution is similar to word embeddings, we believe they complement each other, as reported by Levy and Goldberg (2014). Following Schnabel and Schütze (2014), we restricted the set of indicator words to the 500 most frequent words in the corpus, and used two special feature entries: One is the marginal probability of the non-indicator words and the other is the probabilities of neighboring sentence boundaries. The conditional probabilities for left and right neighbors are estimated in the same way as Equation 2 in Section 3.2, and there are a total of $1,004$ dimensions of this feature for a word.

4 Neural Networks

The non-linearity of the discrete features has been exploited in many NLP tasks, since the simple conjunction of the discrete features is intuitive and the thresholding of these combinatorial features by their feature counts effectively suppresses the combinatorial increase of the parameters.

In contrast, it is not easy to manually tune the non-linearity of the continuous features. For example, it is not intuitive to design the conjunction features of two kinds of word embeddings, word2vec and glove. Although kernel methods have been used to incorporate non-linearity in prior research, they are rarely used now because their tagging speed is too slow (Giménez and Màrquez, 2003). Our solution is to introduce feed-forward neural networks to capture the non-linearity of the corpus-wide information.

4.1 Hybrid model

We designed our tagger as a hybrid of a linear model and a non-linear model. Wang and Manning (2013) reported that a neural network using both sparse discrete features and dense (low-

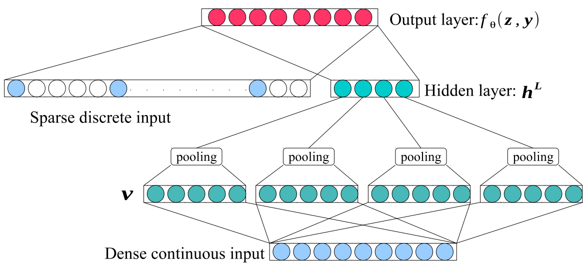


Figure 1: A hybrid architecture of a linear model and a neural network with a pooling activation function

dimensional) continuous features was worse than a linear model using the same two features. At the same time, they also reported that a neural network using only the dense continuous features outperformed a linear model using the same features. Based on their results, we applied neural networks only for the continuous features and used a linear model for the discrete features.

Formally, the scoring function (1) in Section 2 is defined as the composite function of two terms: $f(\boldsymbol{z}, y) := f_{\text{linear}}(\boldsymbol{z}, y) + f_{\text{nn}}(\boldsymbol{z}, y)$. The first f_{linear} is the linear model and the second f_{nn} is a neural network. Since this is a linear combination of two functions, the subgradient of the loss function required for Algorithm 1 is also the linear combination of subgradients of two functions, which means

$$\begin{aligned}\partial_{\boldsymbol{\theta}} \ell(\boldsymbol{z}_t, y_t, \tilde{y}) &= \partial_{\boldsymbol{\theta}} f_{\text{linear}}(\boldsymbol{z}_t, \tilde{y}) + \partial_{\boldsymbol{\theta}} f_{\text{nn}}(\boldsymbol{z}_t, \tilde{y}) \\ &\quad - \partial_{\boldsymbol{\theta}} f_{\text{linear}}(\boldsymbol{z}_t, y_t) - \partial_{\boldsymbol{\theta}} f_{\text{nn}}(\boldsymbol{z}_t, y_t)\end{aligned}$$

if $f_{\boldsymbol{\theta}}(\boldsymbol{z}_t, y_t) - f_{\boldsymbol{\theta}}(\boldsymbol{z}_t, \tilde{y}) < 0$.

First, the linear model can be defined as

$$f_{\text{linear}}(\boldsymbol{z}, y) := \boldsymbol{\theta}_{\text{d}} \cdot \boldsymbol{\phi}_{\text{d}}(\boldsymbol{z}, y),$$

where $\boldsymbol{\phi}_{\text{d}}(\boldsymbol{z}, y)$ is a feature mapping for the discrete part of z and a POS tag, and $\boldsymbol{\theta}_{\text{d}}$ is the corresponding parameter vector. Since this is a linear model, the gradient of this function is simply $\partial_{\boldsymbol{\theta}} f_{\text{linear}}(\boldsymbol{z}, y) = \boldsymbol{\phi}_{\text{d}}(\boldsymbol{z}, y)$.

Second, each hidden layer of our neural networks non-linearly transforms an input vector $\boldsymbol{h}'$ into an output vector $\boldsymbol{h}$ and we can say $\boldsymbol{h}'$ is the continuous part of $\boldsymbol{z}$ at the first layer. Let $\boldsymbol{h}^L$ be a hidden activation of the top layer, which is the non-linear transformation of the continuous part of $\boldsymbol{z}$. The output layer of the neural network is defined as

$$f_{\text{nn}}(\boldsymbol{z}, y) := \boldsymbol{\theta}_{\text{o}} \cdot \boldsymbol{\phi}_{\text{o}}(\boldsymbol{h}^L, y),$$

where $\boldsymbol{\phi}_{\text{o}}(\boldsymbol{h}, y)$ is a feature mapping for the hidden variables and a POS tag, and $\boldsymbol{\theta}_{\text{o}}$ is the corresponding parameter vector.

4.2 Activation functions

The hidden variables $\boldsymbol{h}$ are computed by the recursive application of a non-linear activation function. Since new styles of the activation functions were recently proposed, we review several activation functions here. Let $\boldsymbol{v} \in \mathbb{R}^{|V|}$ be the input of an activation function and each element is $v_j = \boldsymbol{\theta}_{\text{nn},j} \cdot \boldsymbol{h}' + \theta_{\text{bias},j}$, where $\boldsymbol{\theta}_{\text{nn},j}$ is the parameter vector for v_j and $\theta_{\text{bias},j}$ is the bias parameter for v_j. We also assume $\boldsymbol{v}$ is divided into groups of size G, and denote the j-th element of the i-th group as $\{v_{ij} | 1 \le i \le |V|/G \wedge 1 \le j \le G\}$. We studied three activation functions:

1. Rectified linear units (ReLUs) (Nair and Hinton, 2010):

$$h_j = \max(0, v_j) \text{ for all } \{j | 1 \le j \le |V|\}.$$

Note that a subgradient of ReLUs is

$$\frac{\partial h_j}{\partial \boldsymbol{\theta}} = \begin{cases} \frac{\partial v_j}{\partial \boldsymbol{\theta}} & \text{if } v_j > 0 \\ 0 & \text{otherwise.} \end{cases}$$

2. Maxout networks (MAXOUT) (Goodfellow et al., 2013):

$$h_i = \max_{1 \le j \le G} v_{ij} \text{ for all } \{i | 1 \le i \le \frac{|V|}{G}\}.$$

Note that a subgradient of MAXOUT is

$$\frac{\partial h_i}{\partial \boldsymbol{\theta}} = \frac{\partial v_{i\hat{j}}}{\partial \boldsymbol{\theta}}, \text{ where } \hat{j} = \operatorname*{argmax}_{1 \le j \le G} v_{ij}$$

3. Normalized L_p-pooling (L_p) (Gulcehre et al., 2014):

$$h_i = \left(\frac{1}{G} \sum_{j=1}^{G} |v_{ij}|^p \right)^{\frac{1}{p}} \text{ for all } \{i | 1 \le i \le \frac{|V|}{G}\}.$$

Note that a subgradient of L_p is

$$\frac{\partial h_i}{\partial \boldsymbol{\theta}} = \sum_{j=1}^{G} \frac{\partial v_{ij}}{\partial \boldsymbol{\theta}} \frac{v_{ij}|v_{ij}|^{p-2}}{G} \left(\frac{1}{G} \sum_{j=1}^{G} |v_{i,j}|^p \right)^{\frac{1}{p}-1}.$$

The activation inputs for each predefined group, $\{v_{1j}, \ldots, v_{Gj}\}$, are aggregated by a non-linear function in MAXOUT or L_p activation functions, while each input is transformed into a corresponding hidden variable in the ReLUs. When the number of parameters required for these activation functions is the same, the number of output variables $\boldsymbol{h}$ for MAXOUT and L_p is one-G-th smaller than that for ReLUs. Boureau et al. (2010) show pooling operations theoretically reduce the variance of hidden activations, and our experimental results also show MAXOUT and L_p perform better than the ReLUs with the same number of parameters. Note that MAXOUT is a special case of unnormalized L_p pooling when $p = \infty$ and $v_j > 0$ for all j (Zhang et al., 2014). Figure 1 summarizes the proposed architecture with a single hidden layer and a pooling activation function.

4.3 Hyper-parameter search

Finally, the subgradients of the neural network, $f_{\text{nn}}(\boldsymbol{z}, y)$, can be computed through standard back-propagation algorithms and we can apply them in Algorithm 1. However, many of the hyper-parameters have to be determined for the training of the neural networks, and two stages of random hyper-parameter searches (Bergstra and Bengio, 2012) are used in our experiments. Note that the parameters are grouped into three sets, $\boldsymbol{\theta}_d, \boldsymbol{\theta}_{\text{o}}, \boldsymbol{\theta}_{\text{nn}}$, and the same values for $\lambda_1, \lambda_2, \alpha, \beta$ are used for each parameter set.

In the first stage, we randomly select 32 combinations of λ_2 for f_{nn}, λ_1, λ_2 for f_{linear}, the epoch to start the L1/L2 regularizations, and the on and off the acceleration in Algorithm 1. Here are the candidates of three hyper-parameters:

1. λ_1: 0 for the update of f_{nn} and $\{0, 10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}, 1\}$ for the update of f_{linear};
2. λ_2: $\{0.1, 0.5, 1, 5, 10\}$ for the update of f_{nn} and $\{1, 5, 10, 50, 100\}$ for the update of f_{linear}; and
3. Epoch to start the regularizations: $\{0, 1, 2\}$.

In the second stage with each hyper-parameter combination above, we select 8 random combinations of α, β for both f_{linear} and f_{nn} and initial parameter ranges R for f_{nn}. Here are the candidates of the three hyper-parameters:

1. α: $\{0.01, 0.05, 0.1, 0.5, 1, 5\}$;
2. β: $\{0.5, 1, 5\}$;
3. R: $\{[-0.1, 0.1], [-0.05, 0.05], [-0.01, 0.01], [-0.005, 0.005]\}$.

The values of $\boldsymbol{\theta}$ for f_{nn} are uniformly sampled in the range of the randomly selected R. Note that, according to Goodfellow et al. (2014), the biases $\boldsymbol{\theta}_{\text{bias}}$ are initialized as 0 for MAXOUT and L_p, and uniformly sampled from a range $R + \max(R)$, i.e., always initialized with non-negative values. The best combination for the development set is chosen after training that uses random 20% of the training set at the second stage, and Algorithm 1 is terminated when the all token accuracy of the development data has been declining for 5 epochs at the first stage. In other words, 32×8 random combinations of α, β, and $\boldsymbol{\theta}$ for f_{nn} were tested.

Data Set	#Sent.	#Tokens	#Unknown
Training	38,219	912,344	0
Development	5,527	131,768	4,467
Testing	5,462	129,654	3,649

Table 1: Data set splits for PTB.

5 Experiments

5.1 Setup

Our experiments were mainly performed using the Wall Street Journal from Penn Treebank (PTB) (Marcus et al., 1993). We used tagged sentences from the parse trees (Toutanova et al., 2003) and followed the standard approach of splitting the PTB, using sections 0–18 for training, section 19–21 for development, and section 22–24 for testing (Table 1). In addition, we used the CoNLL2009 data sets with the training, development, and test splits used in the shared task (Hajič et al., 2009) for better comparison with a joint model of POS tagging and dependency parsing (Bohnet and Nivre, 2012).

Our baseline tagger was trained by Algorithm 1. As discrete features for our tagger, we used the same binary feature set as Choi and Palmer (2012) which is composed of (a) $1, 2, 3$-grams of the surface word-forms and their predicted/dominated POS tags, (b) the prefixes and suffixes of the words, and (c) the spelling types of the words. In the same way as Choi and Palmer (2012), we used lowercase simplified word-forms which appeared at least 3 times.

In addition to their binary features, we used continuous features which are the concatenation of the corpus-wide features in a context window. The window of size $w = 2s + 1$ is the local context centered around x_t: $x_{t-s}, \cdots, x_t, \cdots, x_{t+s}$. The experimental settings of each feature described in Section 3 are as follows.

Word embeddings
We used two word vectors: 300-dimensional vectors that were learned by word2vec using a part of the Google News dataset (around 100 billion tokens) [1], and 300-dimensional vectors that were learned by glove using a part of the Common Crawl dataset (840 billion tokens) [2]. For sentence boundaries, we use the vector of the special entry "`</s>`" for the word2vec embeddings and the zero vector for the glove embeddings.

POS tag distribution
The counts are calculated using training data.

Supertag distribution
In the experiments on PTB, we used the Stanford parser v2.0.4[3] to convert from phrase structures to dependency structures so that the dependency relation labels of the Stanford dependencies are used. The size of the supertag set is 85 for both heads and dependents in our experiments. In the experiments on CoNLL2009, the dependency structures and labels defined in CoNLL2009 are used and the size of supertag set is 99 for both heads and dependents.

Context word distribution
To count the neighboring words in our experiments, we used sections 0–18 of the Wall Street Journal and all of the Brown corpus from Penn Treebank (Marcus et al., 1993).

Since the training of the neural networks is computationally demanding, first, we trained the linear classifiers using Algorithm 1 to select the best window sizes for each corpus-wide information of Section 3. Then the best window size setting for the development set of PTB was used for training the neural networks described in Section 4.

	Window size					Accuracy (%)	
#	**w2v**	**glv**	**pos**	**stg**	**cw**	**All**	**Unk.**
1	-	-	-	-	-	97.15	86.81
2	3	-	-	-	-	97.36	88.96
3	-	3	-	-	-	97.34	89.55
4	3	3	-	-	-	97.40	90.44
5	3	3	3	-	1	97.44	90.17
6	3	3	3	1	1	97.44	**90.53**
7	3	3	3	3	1	**97.45**	90.22
8	3	3	6	-	1	97.41	90.51
9	3	3	6	3	1	97.44	90.15

Table 2: Feature and window size selection: development accuracies of all tokens (All) and unknown tokens (Unk.) of linear models trained on PTB (w2v: word2vec; glv: glove; pos: POS tag distribution; stg: supertag distribution; cw: context word distribution).

We fixed the group size at 8 for MAXOUT and L_p, and the number of hidden variables was chosen from $\{32, 48\}$ for MAXOUT and L_p and from $\{32, 64, 128, 256, 384\}$ for ReLUs according to all token accuracy on the development data of PTB. We report the POS tagging accuracy for both all of the tokens and only for the unknown tokens that do not appear in the training set.

5.2 Results

Table 2 shows the accuracies of the linear models on PTB with different window sizes for the continuous features. The window sizes of the word embeddings (word2vec and glove) in Section 3.1, POS tag distributions in Section 3.2, supertag distributions in Section 3.3, and context word distributions in Section 3.4 are shown in the columns of *w2v*, *glv*, *pos*, *stg*, and *cw*, respectively. Note that "-" denotes the corresponding feature was not used at all and the first row with all "-" denotes the results only using the original binary features from Choi and Palmer (2012). The window sizes in Table 2 are chosen mainly to investigate the effect of the word2vec embeddings, glove embeddings, and supertag distributions, since they had not previously been used for POS tagging.

The additions of the word embeddings improve all token accuracy by about 0.2 points according to the results shown in Nos. $1, 2, 3$. Although both word embeddings improved the accuracy of the unknown tokens, the gain of the glove embeddings (No. 3) is larger than that of the

[1]The pre-trained vectors are available at `https://code.google.com/p/word2vec`

[2]The pre-trained vectors are available at `http://nlp.stanford.edu/projects/glove/`

[3]`http://nlp.stanford.edu/software/lex-parser.shtml`

	Neural Network Settings			Development Set		Test Set	
#	**Activation functions**	**#Hidden**	**Group size (G)**	**All**	**Unk.**	**All**	**Unk.**
1	Linear model	-	-	97.45	90.22	97.46	91.39
2	ReLUs	384	1	97.45	90.87	97.42	91.04
3	$L_p(p=2)$	48	8	**97.52**	90.91	**97.51**	91.64
4	$L_p(p=3)$	32	8	97.51	90.91	**97.51**	91.53
5	MAXOUT	48	8	97.50	90.89	97.50	**91.67**
6	$L_p(p=2)$ (w/o linear part)	48	8	97.39	**91.18**	97.40	91.23

Table 3: Development and test accuracies of all tokens and unknown tokens (%) on PTB.

Tagger	**All**	**Unk.**
Manning (2011)	97.32	90.79
Søgaard (2011)	97.50	N/A
$L_p(p=2)$	97.51	91.64

(a) Test accuracies on PTB

Tagger	**All**	**Unk.**
Bohnet and Nivre (2012)	97.84	N/A
$L_p(p=2)$	98.02	92.01

(b) Test accuracies on CoNLL2009

Table 4: Test accuracies of all tokens and unknown tokens (%) comparing with the previously reported results

word2vec (No. 2). The reason for this difference in the two embeddings may be because the training data for the glove vectors is 8 times larger than that for the word2vec vectors. The usage of the two word embeddings shows further improvement in the tagging accuracy over single word embeddings (No. 4).

The addition of the POS tag distributions and the context word distributions improves all token accuracy (Nos. $5, 8$). The comparison between the results with stag="-" (Nos. $5, 8$) and stag $= \{1, 3\}$ (Nos. $6, 7, 9$) indicates the minor but consistent improvement by using the supertag distribution features in Section 3.3. Finally, the 7th window-size setting in Table 2 achieves the best all token accuracy among the linear models, so we chose this setting for the experiments with the neural networks.

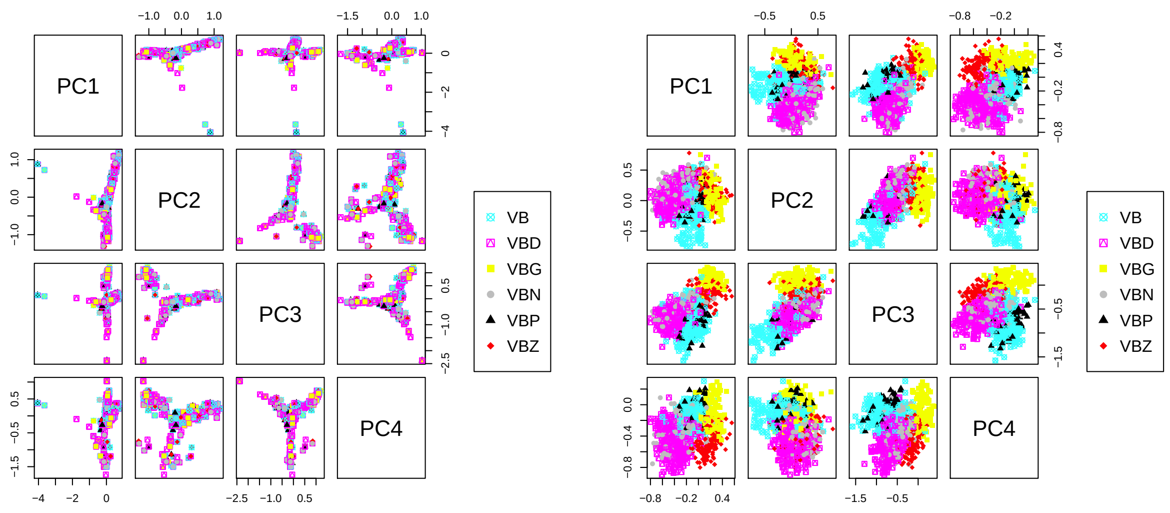
In Table 3, we compare the different settings of the neural networks with a single hidden layer [4] on the development set and test set from PTB. Neural networks with the MAXOUT and L_p (Nos. $3, 4, 5$) significantly outperform the best linear model (No. 1) [5], but the accuracy of the ReLUs (No. 2) was similar to that of the best linear model. According to these results, we argue that the activation function selection is important, although conventional research in NLP has used only a single activation function. It took roughly 7 times as long to learn the hybrid models than the linear model (No. 1). "$L_p(p = 2)$ (w/o linear part)" (No. 6) shows the result for a $L_p(p = 2)$ model which does not include the linear model f_{linear} for the binary features. Comparing the test results of No. 6 with that of No. 3, the proposed hybrid architecture of a linear model and a neural network enjoys the benefits of both models. Note that No. 6's accuracies of the unknown tokens are relatively competitive, and this may be because the continuous features for the neural network do not include word surfaces.

Since it shows the best accuracy for all tokens on the development set, we refer to $L_p(p = 2)$ with 48 hidden variables and the group size of 8 (No. 3 in Table 3) as our representative tagger and denote it as $L_p(p = 2)$ in the rest of discussion. In Table 4a, we compare our result with the previously reported results and we see that our tagger outperforms the current state-of-the-art systems on PTB for the accuracies of all tokens and unknown tokens.

In addition, since our tagger was trained using the dependency tree annotations as described in Section 3.3, we compare it with the results of Bohnet and Nivre (2012) which is also trained using both POS tag and dependency annotations. Although their focus is on the dependency pars-

[4]We leave the investigation of deeper neural networks as future work.

[5]For significance tests, we have used the *Wilcoxon matched-pairs signed-rank test* at the 95% confidence level dividing the data into 100 data pairs.



(a) PCA of the raw features

(b) PCA of the hidden activations of $L_p(p = 2)$

Figure 2: Scatter plots of verbs for all combinations between the first four principal components of the raw features and the activation of hidden variables.

ing, they report state-of-the art POS accuracies for many languages. Note that Bohnet and Nivre (2012) also used external resources. Table 4b gives the results for CoNLL2009 data set[6]. Our tagger outperform Bohnet and Nivre (2012), so we believe this is the highest POS accuracy ever reported for a tagger trained on this data set.

Finally, to visualize the learned representations, we applied principal components analysis (PCA) to the hidden activations $\boldsymbol{h}^L$ of the first $10,000$ tokens of the development set from PTB. We also performed PCA to the raw continuous inputs of the same data set. Figure 2 shows the data plots for all the combinations among the first four principal components. We plots only the verb tokens to make the plots easier to see. Figures 2a and 2b show the PCA results of the raw features and the hidden activations of $L_p(p = 2)$, respectively. Compared to Figure 2a, the tokens with the same POS tag are more clearly clustered in Figure 2b. This suggests the neural network learned the good representations for POS tagging and these hidden activations can be used as the input of the succeeding processes, such as parsing.

[6]The accuracies of our tagger on the development set of CoNLL2009 data are 97.76% for all tokens and 93.42% for unknown tokens.

6 Related Work

There is some old work on the POS tagging by neural networks. Nakamura et al. (1990) proposed a neural tagger that predicts the POS tag using a previous POS predictions. Schmid (1994) is most similar to our work. The inputs of his neural network are the POS tag distributions of a word and its suffix in a context window, and he reports a 2% improvement over a regular hidden Markov model. However, his tagger did not use the other kinds of corpus-wide information as we used.

Most of the recent studies on POS tagging use linear models (Suzuki and Isozaki, 2008; Spoustová et al., 2009) or other non-linear models, such as k-nearest neighbor (kNN) (Søgaard, 2011). One trend in these studies is model combinations. Suzuki and Isozaki (2008) combined generative and discriminative models, Spoustová et al. (2009) used the combination of three taggers to generate automatically annotated corpus, and Søgaard (2011) used the outputs of a supervised tagger and an unsupervised tagger as the feature space of the kNN. Our work also follows this trend since neural networks can be considered as non-linear integration of several linear classifiers.

Apart from POS tagging, some previous studies in parsing used the discretization method to handle the combination of continuous features. Bohnet and Nivre (2012) binned the difference of two continuous features in discrete steps of a predefined small interval. Bansal et al. (2014) used the conjunction of discretized features and studied two discretization methods: One is the binning of real values into discrete steps and the other is a hard clustering of continuous feature vectors. It is not easy to determine the optimal intervals for the binning method, and the clustering method is unsupervised so that the clusters are not guaranteed for good representations of the target tasks.

To capture rich syntactic information for Chinese POS tagging, Sun and Uszkoreit (2012) used the ensemble model of both a POS tagger and a constituency parser. Sun et al. (2013) improved the efficiency of Sun and Uszkoreit (2012) in which a single tagging model is trained using automatically annotated corpus generated by the ensemble tagger. Although the supertag distribution feature in Section 3.3 is a simple way to incorporate syntactic information, automatically parsed large corpora may make the estimate of the supertag distributions more accurate.

7 Conclusion and Future Work

We are studying a neural network approach to handle the non-linear interaction among corpus-wide statistics. For POS tagging, we used word embeddings, POS tag distributions, supertag distributions, and context word distributions in a context window. These features are beneficial, even for linear classifiers, but the neural networks leverage these features for improving tagging accuracies. Our tagger with Maxout networks (Goodfellow et al., 2013) or L_p-pooling (Zhang et al., 2014; Gulcehre et al., 2014) show the state-of-the-art results on two English benchmark sets.

Our empirical results suggest further opportunities to investigate continuous features not only for POS tagging but also for other NLP tasks. An obvious use case for continuous features is the N-best outputs with confidence values, which were predicted by the previous process in a NLP pipeline, such as the POS tags used for syntactic parsing. Another interesting extension is the use of on-the-fly features which reflect previous network states, although the neural networks in our current work do not refer to the prediction history. Recurrent neural networks (RNNs) may be a solution to represent the prediction history in a compact way, and Mesnil et al. (2013) reported that RNNs outperform conditional random fields (CRFs) on a sequential labeling task. They also show the superiority of bi-directional RNNs on their task, so the bi-directional RNNs may also be effective on the POS tagging, since bi-directional inferences were also used in earlier work (Tsuruoka and Tsujii, 2005).

It has a clear benefit over kernel methods in that the test-time computational cost of neural networks is independent from training data. However, although the test-time speed of original kernel methods is proportional to the number of training data, recent development of kernel approximation techniques achieve significant speed improvements (Le et al., 2013; Pham and Pagh, 2013). Since this work shows the non-linearity of continuous features should be exploited, those approximated kernel methods may also improve the tagging accuracies without sacrifice tagging speed.

Independent from our work, Ma et al. (2014) and Santos and Zadrozny (2014) also recently proposed neural network approaches for POS tagging. Ma et al. (2014)'s approach is similar to our approach, with a combination of a linear model and a neural network, although a direct comparison is not easy since their focus is the Web domain adaptation of POS tagging. Remarkably, they report n-gram embeddings are better than single word embeddings. Santos and Zadrozny (2014) proposed character-level embedding to capture the morphological and shape information for POS tagging. Although the reported accuracy (97.32%) on PTB data is lower than state of the art results, their approach is promising for morphologically rich languages. We may study the integration of these embeddings into our approach as future work.

References

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*. The Association for Computer Linguistics.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Com-*

putational Natural Language Learning (EMNLP-CoNLL), pages 1455–1465.

Y-Lan Boureau, Jean Ponce, and Yann LeCun. 2010. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 111–118.

Jinho D. Choi and Martha Palmer. 2012. Fast and robust part-of-speech tagging using dynamic model selection. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*, pages 363–367.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Koby Crammer and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

Hal Daumé III, John Langford, and Daniel Marcu. 2009. Search-based structured prediction. *Machine Learning Journal*, 75(3):297–325, June.

John C. Duchi, Elad Hazan, and Yoram Singer. 2010. Adaptive subgradient methods for online learning and stochastic optimization. In *Proceedings of the Conference on Learning Theory (COLT)*, pages 257–269.

Jesús Giménez and Lluís Màrquez. 2003. Fast and accurate part-of-speech tagging: The svm approach revisited. In *Proceedings of Recent Advances in Natural Language Processing (RANLP)*, pages 153–163.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of International Conference on Computational Linguistics (COLING)*, pages 959–976.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. 2013. Maxout networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1319–1327.

Ian J. Goodfellow, Mehdi Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio. 2014. An empirical investigation of catastrophic forgeting in gradient-based neural networks. In *Proceedings of International Conference on Learning Representations (ICLR)*.

Caglar Gulcehre, Kyunghyun Cho, Razvan Pascanu, and Yoshua Bengio. 2014. Learned-norm pooling for deep feedforward and recurrent neural networks. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Conference on Computational Natural Language Learning (CoNLL)*, pages 1–18.

Fei Huang and Alexander Yates. 2009. Distributional representations for handling sparsity in supervised sequence-labeling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (ACL/IJCNLP)*, pages 495–503.

Quoc V. Le, Tamás Sarlós, and Alexander J. Smola. 2013. Fastfood - computing Hilbert space expansions in loglinear time. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 244–252.

Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL)*.

Ji Ma, Yue Zhang, Tong Xiao, and Jingbo Zhu. 2014. Tagging the Web: Building a robust web tagger with neural network. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*. The Association for Computer Linguistics.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proceedings of Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, pages 171–189.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1222–1230.

H. Brendan McMahan. 2011. Follow-the-regularized-leader and mirror descent: Equivalence theorems and L1 regularization. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 525–533.

Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Proceedings of Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 3771–3775.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 3111–3119.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 807–814.

Tetsuji Nakagawa and Yuji Matsumoto. 2006. Guessing parts-of-speech of unknown words using global information. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*.

Masami Nakamura, Katsuteru Maruyama, Takeshi Kawabata, and Kiyohiro Shikano. 1990. Neural network approach to word category prediction for English texts. In *Proceedings of International Conference on Computational Linguistics (COLING)*, pages 213–218.

Hiroki Ouchi, Kevin Duh, and Yuji Matsumoto. 2014. Improving dependency parsers with supertags. In *Proceedings of Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 154–158.

Neal Parikh and Stephen P. Boyd. 2013. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: global vectors for word representation. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*.

Ninh Pham and Rasmus Pagh. 2013. Fast and scalable polynomial kernels via explicit feature maps. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 239–247.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 627–635.

Cicero Dos Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1818–1826.

Helmut Schmid. 1994. Part-of-speech tagging with neural networks. In *Proceedings of International Conference on Computational Linguistics (COLING)*, pages 172–176.

Tobias Schnabel and Hinrich Schütze. 2014. FLORS: Fast and simple domain adaptation for part-of-speech tagging. *Transactions of the Association for Computational Linguistics*, 2:15–26, February.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 801–809.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Chris Manning, Andrew Ng, and Chris Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods on Natural Language Processing (EMNLP)*, pages 1631–1642.

Anders Søgaard. 2011. Semi-supervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*, pages 48–52.

Drahomíra johanka Spoustová, Jan Hajič, Jan Raab, and Miroslav Spousta. 2009. Semi-supervised training for the averaged perceptron pos tagger. In *Proceedings of Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 763–771.

Weiwei Sun and Hans Uszkoreit. 2012. Capturing paradigmatic and syntagmatic lexical relations: Towards accurate Chinese part-of-speech tagging. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*, pages 242–252.

Weiwei Sun, Xiaochang Peng, and Xiaojun Wan. 2013. Capturing long-distance dependencies in sequence models: A case study of Chinese part-of-speech tagging. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*, pages 180–188.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. 2013. On the importance of initialization and momentum in deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1139–1147.

Jun Suzuki and Hideki Isozaki. 2008. Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*, pages 665–673.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 252–259.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. 2005. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pages 467–474.

Joseph P. Turian, Lev-Arie Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (ACL)*, pages 384–394.

Mengqiu Wang and Christopher D. Manning. 2013. Effect of non-linear deep architecture in sequence labeling. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*.

Xiaohui Zhang, Jan Trmal, Daniel Povey, and Sanjeev Khudanpur. 2014. Improving deep neural network acoustic models using generalized maxout networks. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Alisa Zhila, Wen tau Yih, Christopher Meek, Geoffrey Zweig, and Tomas Mikolov. 2013. Combining heterogeneous models for measuring relational similarity. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1000–1009.