A Dynamic Window Neural Network for CCG Supertagging

Huijia Wu,[1,3] Jiajun Zhang,[1,3] Chengqing Zong*[1,2,3]
[1]National Laboratory of Pattern Recognition, Institute of Automation, CAS
[2]CAS Center for Excellence in Brain Science and Intelligence Technology
[3]University of Chinese Academy of Sciences
{huijia.wu,jjzhang,cqzong}@nlpr.ia.ac.cn

Abstract

Combinatory Category Grammar (CCG) supertagging is a task to assign lexical categories to each word in a sentence. Almost all previous methods use fixed context window sizes to encode input tokens. However, it is obvious that different tags usually rely on different context window sizes. This motivates us to build a supertagger with a dynamic window approach, which can be treated as an attention mechanism on the local contexts. We find that applying dropout on the dynamic filters is superior to the regular dropout on word embeddings. We use this approach to demonstrate the state-of-the-art CCG supertagging performance on the standard test set.

Introduction

Combinatory Category Grammar (CCG) provides a connection between syntax and semantics of natural language. The syntax can be specified by derivations of the lexicon based on the combinatory rules, and the semantics can be recovered from a set of predicate-argument relations. CCG provides an elegant solution for a wide range of semantic analysis, such as semantic parsing (Zettlemoyer and Collins 2007; Kwiatkowski et al. 2010; 2011; Artzi, Lee, and Zettlemoyer 2015), semantic representations (Bos et al. 2004; Bos 2005; 2008; Lewis and Steedman 2013), and semantic compositions, all of which heavily depend on the supertagging and parsing performance. All these motivate us to build a more accurate CCG supertagger.

CCG supertagging is the task to predict the lexical categories for each word in a sentence. Existing algorithms on CCG supertagging range from point estimation (Clark and Curran 2007; Lewis and Steedman 2014) to sequential estimation (Xu, Auli, and Clark 2015; Lewis, Lee, and Zettlemoyer 2016; Vaswani et al. 2016), which predict the most probable supertag of the current word according to the context in a fixed size window. This fixed size window assumption is too strong to generalize. We argue this from two perspectives.

One perspective comes from the inputs. For a particular word, the number of its categories may vary from 1 to 130 in CCGBank 02-21 (Hockenmaier and Steedman 2007). We need to choose different context window sizes to meet different ambiguity levels. The other perspective is for the targets. There are about 21000 different words together with 31 different Part-Of-Speech(POS) tags which have the same category N/N. Using the same context window size for each word is obviously inappropriate.

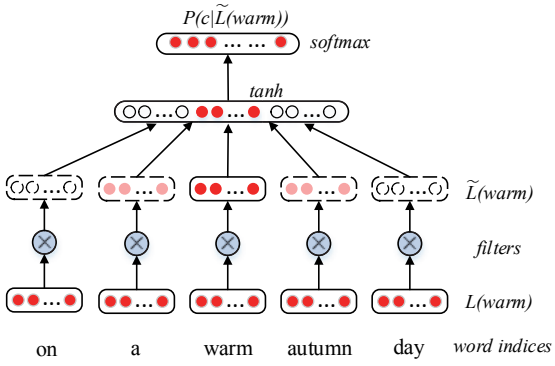


Figure 1: A dynamic window approach for supertagging using a multilayer perceptron. We add "filters" to each token in the context window. If one filter is closed to 0, then the corresponding token will be blocked, otherwise it will be passed as a normal input.

To overcome these problems, we notice that Xu et al. (2015) use dropout in the embedding layer to make the input contexts sparse. This method motivates us to get rid of the unnecessary information in the contexts automatically rather than use a pre-specified prior. Then we observe that the gating mechanism of long short-term memory (LSTM) blocks, especially the input gate, can determine when to enter into the block. All these inspire us to add a gate to each item in the context windows to make them sparse but informative.

This method can be interpreted as the attention mechanism (Bahdanau, Cho, and Bengio 2014), which focuses on parts of the memories when making decisions. From this perspective, the contexts of the current word are the memories, and the dynamic window is the attention. We focus on the contexts extracted from the attended windows to predict

*Corresponding author.

the corresponding lexical categories.

Figure 1 visualize this method. We add one logistic gate to each item in the local context. If the gate is close to zero, the corresponding features in the window will be ignored during forward pass. Moreover, we add a dropout mask on the gates to further improve its sparsity. Combining attention and dropout lead a significant performance improvement.

We evaluated our approach on multilayer perceptrons (MLPs) and and recurrent neural networks (RNNs), including vanilla forms (standard RNNs) and gated RNNs. The experiments show that the performance of these networks can obtain improvements using our method.

Background

Category Notation

CCG uses a set of lexical categories to represent constituents (Steedman 2000). In particular, a fixed finite set is used as a basis for constructing other categories, which is described in Table 1.

Category	Description
N	noun
NP	noun phrase
PP	prepositional phrase
S	sentence

Table 1: The description of basic categories used in CCG

The basic categories could be used to generate an infinite set $\mathbb{C}$ of functional categories by applying the following recursive definition:

- $N, NP, PP, S \in \mathbb{C}$
- $X/Y, X\backslash Y \in \mathbb{C}$ if $X, Y \in \mathbb{C}$

Each functional category specifies some arguments. Combining the arguments can form a new category according to the orders (Steedman and Baldridge 2011). The argument could be either basic or functional, and the orders are determined by the forward slash $/$ and the backward slash $\backslash$. A category X/Y is a forward functor which could accept an argument Y to the right and get X, while the backward functor $X\backslash Y$ should accept its argument Y to the left.

Neuron Based Supertaggers

CCG supertagging is an approach for assigning lexical categories for each word in a sentence. The problem can be formulated by $P(\mathbf{c}|\mathbf{w};\boldsymbol{\theta})$, where $\mathbf{w} = [w_1, \ldots, w_T]$ indicates the T words in a sentence, and $\mathbf{c} = [c_1, \ldots, c_T]$ indicates the corresponding lexical categories. Notice that the length of the words and categories are the same. We denote vectors with bolded font and matrices with capital letters. Bias terms in neural networks are omitted for readability.

Network Inputs. Network inputs are the representation of each token in a sequence. Our inputs include the concatenation of word representations, character representations, and capitalization representations. To reduce sparsity, all words are lower-cased, together with capitalization and character representations as inputs.

Formally, we can represent the distributed word feature $\mathbf{f}_{w_t}$ using a concatenation of these embeddings:

$$\mathbf{f}_{w_t} = [\mathbf{L}_w(w_t); \mathbf{L}_a(a_t); \mathbf{L}_c(\mathbf{c}_w)] \quad (1)$$

where w_t, a_t represent the current word and its capitalization. $\mathbf{c}_w := [c_1, c_2, \ldots, c_{T_w}]$, where T_w is the length of the word and $c_i, i \in \{1, \ldots, T_w\}$ is the i-th character for the particular word. $\mathbf{L}_w(\cdot) \in \mathbb{R}^{|V_w| \times n}$, $\mathbf{L}_a(\cdot) \in \mathbb{R}^{|V_a| \times m}$ and $\mathbf{L}_c(\cdot) \in \mathbb{R}^{|V_c| \times r}$ are the look-up tables for the words, capitalization and characters, respectively. $\mathbf{f}_{w_t} \in \mathbb{R}^{n+m+r}$ represents the distributed feature of w_t. A context window of size d surrounding the current word is used as an input:

$$\mathbf{x}_t = [\mathbf{f}_{w_{t-\lfloor d/2 \rfloor}}; \ldots; \mathbf{f}_{w_{t+\lfloor d/2 \rfloor}}] \quad (2)$$

where $\mathbf{x}_t \in \mathbb{R}^{(n+m+r) \times d}$ is the concatenation of the context features. We use it as the input of the network.

Network Outputs.

Since our goal is to assign CCG categories to each word, we use a *softmax* activation function $g(\cdot)$ in the output layer:

$$\mathbf{y}_t = g(\mathbf{W^{hy}} \mathbf{h}_t) \quad (3)$$

where $\mathbf{y}_t \in \mathbb{R}^K$ is a probability distribution over all possible categories. $y_k(t) = \frac{\exp(h_k)}{\sum_{k'} \exp(h_{k'})}$ is the k-th dimension of $\mathbf{y}_t$, which corresponds to the k-th lexical category in the lexicon. $\mathbf{h}_t \in \mathbb{R}^H$ is the output of the hidden layer at time t. $\mathbf{W^{hy}} \in \mathbb{R}^{K \times H}$ is the hidden-to-output weight.

Inputs to Outputs Mappings. Neuron based supertaggers model the inputs to outputs mappings using neural networks. Since CCG supertagging can either be treated as a point or a sequential estimation problem, which correspond to two kinds of neural networks: MLPs and RNNs, respectively. For simplicity, we will talk about gated RNNs only, which are special kinds of RNNs with logistic gates in the hidden units to control information flow. There are many kinds of gated RNNs, such as long short-term memory, (Hochreiter and Schmidhuber 1997) and gated recurrent unit (Cho et al. 2014). We focus on LSTMs only in this work.

LSTMs replace the hidden units in vanilla RNNs with complicated blocks, which are designed as:

$$\tilde{\mathbf{c}}_t = \sigma(\mathbf{W^{xc}} \mathbf{x}_t + \mathbf{W^{hc}} \mathbf{h}_{t-1}) \quad (4)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (5)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot f(\mathbf{c}_t) \quad (6)$$

where $\mathbf{x}_t$ and $\mathbf{h}_t$ are the input and the output of the block. $\mathbf{i}_t \in \mathbb{R}^H$, $\mathbf{f}_t \in \mathbb{R}^H$ and $\mathbf{o}_t \in \mathbb{R}^H$ denote the input gate, forget gate and output gate, respectively, which are logistic units to filter the information. $\mathbf{W^{xc}} \in \mathbb{R}^{H \times I}$ is the weight storing the input. $\mathbf{W^{hc}} \in \mathbb{R}^{H \times H}$ is the recurrent weight connecting the previous block outputs to the cells. $\mathbf{c}_t \in \mathbb{R}^H$ is the short-term memory state, which is used to store the history information. Based on the three gates, the information flow in $\mathbf{c}_t$ can be kept for a long time. $f(\cdot)$ is the non-linear mapping, here we use the hyperbolic tangent function $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$.

A Dynamic Window Approach

In this section we will introduce a dynamic window approach for supertagging. Specifically, we add logistic gates to each token in the context window to filter the unnecessary information. We can modify the Eq. (2) to:

$$\tilde{\mathbf{x}}_t = \mathbf{r}_t \otimes \mathbf{x}_t \tag{7}$$

$$:= [r_{t-\lfloor d/2 \rfloor}\mathbf{f}_{w_{t-\lfloor d/2 \rfloor}}; \ldots; r_{t+\lfloor d/2 \rfloor}\mathbf{f}_{w_{t+\lfloor d/2 \rfloor}}] \tag{8}$$

Here $\otimes$ denote an element-wise scalar-vector product. $\mathbf{r}_t := [r_{t-\lfloor d/2 \rfloor}, \ldots, r_{t+\lfloor d/2 \rfloor}] \in \mathbb{R}^d$ is a logistic gate to filter the unnecessary contexts. r_i and $\mathbf{f}_i, i \in \{t - \lfloor d/2 \rfloor, \ldots, t + \lfloor d/2 \rfloor\}$ is a scalar and a vector, respectively. Their product is defined as:

$$r\mathbf{f} := [rf_1, \ldots, rf_n]$$

One perspective for the filter gate is an attention model focusing on the necessary contexts. This effect can be visualized in Figure 1: If one component of $\mathbf{r}$, say r_i is 0, the corresponding word feature $\mathbf{f}_{w_i}$ will be removed or deactivated from the input.

Design of the Gates

We use a feed-forward neural network to learn $\mathbf{r}_t$:

$$\mathbf{r}_t = \sigma(\mathbf{W}^{\mathbf{xr}}\mathbf{x}_t) \tag{9}$$

where $\sigma(\cdot)$ is the sigmoid function defined as $\sigma(z) = \frac{1}{1+e^{-z}}$. This function is to make sure the values of $\mathbf{r}_t$ are between 0 and 1. $\mathbf{x}_t$ is network input, as defined in Eq. (2). $\mathbf{r}_t \in \mathbb{R}^d$ is the output of the dynamic window model, where d is the window size. $\mathbf{W}^{\mathbf{xr}} \in \mathbb{R}^{d \times I}$ is the weight to be learned.

One disadvantage of the sigmoid function is when a neuron is nearly saturated, its derivative becomes small, which makes the connecting weights change very slowly. If some neurons in $\mathbf{r}_t$ are saturated, which states will be stable and may not generalize well. To further improve the sparsity in the context window, we add a dropout mask (Srivastava et al. 2014) on $\mathbf{r}_t$:

$$l_i(t) \sim \text{Bernouli}(p) \tag{10}$$

$$\tilde{\mathbf{r}}_t = \mathbf{l}_t \odot \mathbf{r}_t \tag{11}$$

where $\mathbf{l}_t$ is a vector of independent Bernouli random variables $l_i(t)$, which has probability p of being 1. Since $\tilde{\mathbf{r}}_t$ acts on each word feature in the context window, this dropout can be viewed as drop on words directly (Dai and Le 2015; Iyyer, Manjunatha, and Boyd-Graber 2015). One minor difference is they use word dropout at the sentence level, and we use it at the dynamic window level.

To further explain it, let's consider a trivial case when all items in $\mathbf{r}_t$ are set to 1:

$$\mathbf{r}_t = [1, \ldots, 1] \tag{12}$$

Adding a dropout mask on such a $\mathbf{r}_t$ is equivalent to drop the words in the contexts randomly, which can be seen as a window approach with a random size.

Embedded into MLPs

For MLPs with this approach, we can use the filtered context as an input:

$$\mathbf{h}_t = f(\mathbf{W}^{\mathbf{xh}}\tilde{\mathbf{x}}_t) \tag{13}$$

$$\mathbf{y}_t = g(\mathbf{W}^{\mathbf{hy}}\mathbf{h}_t) \tag{14}$$

where $\tilde{\mathbf{x}}_t$ is defined in Eq. (7). We use the filtered context as an input to the hidden layer. $\mathbf{W}^{\mathbf{xh}} \in \mathbb{R}^{H \times I}$ and $\mathbf{W}^{\mathbf{hy}} \in \mathbb{R}^{K \times H}$ is the weight parameters of MLP.

Embedded into Vanilla RNNs

The similar approach can be applied to RNNs with slight modifications. For each hidden state we have two types of inputs: one is from the input layer, the other is from the hidden (Elman 1990) or the output layer (Jordan 1986). The recurrent weight may vanish or explode if its eigenvalues are deviated from 1. To avoid these problems, we add one gate to the recurrent input to reset it:

$$\tilde{\mathbf{r}}_t = \sigma(\mathbf{W}^{\mathbf{xr}}\mathbf{x}_t) \tag{15}$$

$$s_t = \sigma(\mathbf{W}^{\mathbf{xs}}\mathbf{x}_t) \tag{16}$$

where $s_t \in \mathbb{R}$ is a scalar between 0 and 1, which is used to reset the recurrent input. $\mathbf{W}^{\mathbf{xs}} \in \mathbb{R}^{1 \times I}$ is the corresponding hidden-to-output weight. Intuitively, if s_t is close to zero, the recurrent input $\mathbf{W}^{\mathbf{yc}}\mathbf{y}_{t-1}$ will be disappeared, which degenerates to a MLP.

Taken a Jordan-type RNN as an example, we have:

$$\tilde{\mathbf{h}}_t = f(\mathbf{W}^{\mathbf{xh}}\tilde{\mathbf{x}}_t + s_t\mathbf{W}^{\mathbf{yh}}\mathbf{y}_{t-1}) \tag{17}$$

$$\mathbf{y}_t = g(\mathbf{W}^{\mathbf{hy}}\tilde{\mathbf{h}}_t) \tag{18}$$

where $\tilde{\mathbf{h}}_t \in \mathbb{R}^H$ is the output of the hidden layer. $\tilde{\mathbf{x}}_t$ is the current input. $\mathbf{y}_{t-1} \in \mathbb{R}^K$ is the previous output of the output layer. $\mathbf{W}^{\mathbf{yh}} \in \mathbb{R}^{H \times K}$ is the recurrent weight from the previous output layer to the current hidden layer.

Embedded into Gated RNNs

For gated RNNs, we use a two-stacked bidirectional LSTM (Bi-LSTM) to model the task. The architecture can be defined as follows:

$$\overrightarrow{\mathbf{h}_t} = \text{LSTM}(\overrightarrow{\mathbf{x}_t}, \overrightarrow{\mathbf{h}_{t-1}}, \overrightarrow{\mathbf{c}_{t-1}}) \tag{19}$$

$$\overleftarrow{\mathbf{h}_t} = \text{LSTM}(\overleftarrow{\mathbf{x}_t}, \overleftarrow{\mathbf{h}_{t-1}}, \overleftarrow{\mathbf{c}_{t-1}}) \tag{20}$$

$$\mathbf{y}_t = g(\overrightarrow{\mathbf{h}_t}, \overleftarrow{\mathbf{h}_t}) \tag{21}$$

where LSTM$(\cdot)$ is the LSTM computation. $\overrightarrow{\mathbf{x}_t}$ and $\overleftarrow{\mathbf{x}_t}$ are the forward and the backward input sequence, respectively. The output of the two hidden layers $\overrightarrow{\mathbf{h}_t}$ and $\overleftarrow{\mathbf{h}_t}$ in a birectional LSTM are stacked on top of each other:

$$\mathbf{h}_t^l = f^l(\mathbf{h}_t^{l-1}, \mathbf{h}_{t-1}^l) \tag{22}$$

where $\mathbf{h}_t^l$ is the t-th hidden state of the l-th layer.

Discussion

The main idea of the attention mechanism is to focus on parts of the memories, which are used to store the information for prediction, such as the inputs or the hidden units. From this perspective, our dynamic window method can be seen as an attention-based system. Moreover, supertagging is a special kind of sequence-to-sequence problem, in which the input and the output sequence has the same length. Thus, we do not need to use an encoder to memorize the input and use another decoder to generate the output.

The difference between the two attention mechanisms lies in the type of memories. In the encoder-decoder architecture, the attention model is considered through a weighted average of the output of the encoder. The reason is that they use a encoder and a decoder to model the variable-length outputs, and the memories are the encoder hidden states, while in the supertagging problem, we only use a encoder to do the task, our memories are just the inputs.

Experiments

We divide our experiments into two steps: First we make comparisons with the existing approaches to test the performance of our models. The comparisons do not include any externally labeled data or POS labels. Then we describe quantitative results which validate the effectiveness of our dynamic window approach.

Dataset and Pre-Processing

Our experiments are performed on CCGBank (Hockenmaier and Steedman 2007), which is a translation from Penn Treebank (Marcus, Marcinkiewicz, and Santorini 1993) to CCG with a coverage 99.4%. We follow the standard splits, using sections 02-21 for training, section 00 for development and section 23 for the test. We use a full category set containing 1285 tags. All digits are mapped into the same digit '9', and all words are lowercased.

Network Configuration

Initialization.

There are two types of weights in our experiments: recurrent and non-recurrent weights. For non-recurrent weights, we initialize word embeddings with the pre-trained 200-dimensional GolVe vectors (Pennington, Socher, and Manning 2014). Other weights are initialized with the Gaussian distribution $\mathcal{N}(0, \frac{1}{\sqrt{\text{fan-in}}})$ scaled by a factor of 0.1, where *fan-in* is the number of units in the input layer. For recurrent weight matrices, we initialize with random orthogonal matrices through SVD (Saxe, McClelland, and Ganguli 2013)to avoid unstable gradients. Orthogonal initialization for recurrent weights is important in our experiments, which takes about 2% relative performance gain than other methods such as Xavier initialization (Glorot and Bengio 2010).

Hyperparameters.

For MLPs, we use a window size of 9. For vanilla RNNs and gated RNNs, a window size of 3 is enough to capture the local contexts. The dimension of the word embeddings is 200. The size of character embedding and capitalization embeddings are set to 5. We set the maximum number of a word's characters to 5 to eliminate complexity, which means we concatenate the leftmost 5 characters to the rightmost 5 characters as character representations. The number of cells of the stacked Bi-LSTM is set to 512. We also tried 400 cells or 600 cells and found this number did not impact performance so much. All stacked hidden layers have the same number of cells. The output layer has 1286 neurons, which equals to the number of tags in the training set with a RARE symbol.

Training.

We train the networks using the back-propagation algorithm, using stochastic gradient descent (SGD) algorithm with an fixed learning rate 0.02 for all layers. We also tried other optimization methods, such as momentum (Plaut and others 1986), Adadelta (Zeiler 2012), or Adam (Kingma and Ba 2014), but none of them perform as well as SGD. Gradient clipping is not used. We use on-line learning in our experiments, which means the parameters will be updated on every training sequences, one at a time.

We use a negative log-likelihood cost to evaluate the performance. Given a training set $\{(\mathbf{x}^n, \mathbf{t}^n)_{n=1}^N\}$, the objective function can be written as:

$$\mathcal{C} = -\frac{1}{N}\sum_{n=1}^{N} \log \mathbf{y}_{t^n} \tag{23}$$

where $t^n \in \mathbb{N}$ is the true target for sample n, and $\mathbf{y}_{t^n}$ is the t-th output in the *softmax* layer given the inputs $\mathbf{x}^n$.

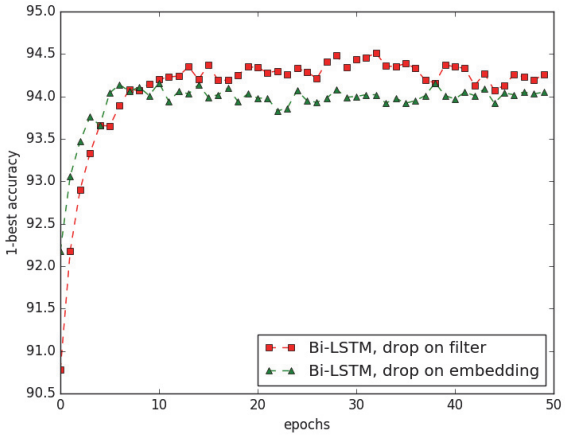


Figure 2: Comparison between dropout on the dynamic filter and dropout on word embeddings using a stacked Bi-LSTM on the development set. "Bi-LSTM, drop on filter" means the Bi-LSTM with dropout on the dynamic filter.

Regularization.

Dropout is the only regularizer in our model to avoid overfitting. We add a dropout mask to our filter gate $\mathbf{r}_t$ with a drop rate 0.5, which is helpful to improve the performance.

Figure 2 shows such the comparison of the filter dropout and the embedding dropout. Here the embedding dropout refers to dropout on $\mathbf{x}_t$. We can see that the filter dropout could improve the 1-best accuracy by about 0.3% than the embedding dropout. We also apply dropout to the output of the hidden layer with a 0.5 drop rate. At test time, weights are scaled with a factor $1 - p$.

Results on Supertagging Accuracy

We report the highest 1-best supertagging accuracy on the development set for final testing. Table 2 shows the comparisons of accuracy on CCGBank. We notice that stacked Bi-LSTM (with depth 2) performs the best than other neural based models, which are introduced in the previous section. For example, the Jordan-type RNN in Table 2 are described in Eq (17). Our dynamic window approach provides about 8% relative performance improvement. The character-level information (+ 4% relative accuracy) are helpful to improve the performance.

Model	Dev	Test
Clark and Curran (2007)	91.5	92.0
Lewis et al. (2014)	91.3	91.6
Lewis et al. (2016)	94.1	94.3
Xu et al. (2015)	93.1	93.0
Xu et al. (2016)	93.49	93.52
Vaswani et al. (2016)	94.24	94.5
MLP	92.06	92.28
Elman RNN	92.74	92.89
Jordan RNN	92.61	92.75
forward LSTM	93.39	93.51
Bi-LSTM	94.35	94.57
stacked Bi-LSTM (depth 2)	**94.5**	**94.71**
stacked Bi-LSTM (no dyn)	94.17	94.30
stacked Bi-LSTM (no char)	94.16	94.46

Table 2: 1-best supertagging accuracy on CCGbank. "no dyn" refers to the models that do not use the filter gates to concatenate the tokens in a context window, "no char" refers to the models that do not use the character-level information, "drop emb" refers to dropout on word embeddings rather than on dynamic filters.

On the Usage of Dynamic Filters

We discuss other types of the dynamic filters. One is to use a element-wise operation on $\mathbf{x}_t$. In this case, $\mathbf{r}_t$ is a matrix, and the operation in Eq. (8) becomes:

$$\tilde{\mathbf{x}}_t = \mathbf{r}_t \odot \mathbf{x}_t \tag{24}$$

$$:= [\mathbf{r}_{t-\lfloor d/2 \rfloor} \odot \mathbf{f}_{w_{t-\lfloor d/2 \rfloor}}; \ldots; \mathbf{r}_{t+\lfloor d/2 \rfloor} \odot \mathbf{f}_{w_{t+\lfloor d/2 \rfloor}}] \tag{25}$$

Here $\odot$ denote an element-wise product. The performance is 94.25% (Table 3, line 2), which shows that the gate is preferred to operate on words directly, rather than on word embeddings.

We can also use a MLP instead of a one-layer network to learn the dynamic filters:

$$\mathbf{u}_t = \sigma(\mathbf{W}^{\mathbf{xu}}\mathbf{x}_t)$$
$$\mathbf{r}_t = \sigma(\mathbf{W}^{\mathbf{ur}}\mathbf{u}_t) \tag{26}$$

where we add a hidden layer $\mathbf{u}_t \in \mathbb{R}^u$ to learn $\mathbf{r}_t$. But the performance is not good (94.43%, Table 3, line 3) with an extra computational cost.

The weighted average on the inputs $\sum_{i=1}^{d} r_i \mathbf{f_i}$ is a standard method for attention, which leads to a poor result of 94.15% (Table 3, line 4). This shows the gated concatenation is superior to the weighted average for sequence tagging.

Filters	Accuracy	Remark
one layer (origin)	94.7	$\mathbf{r}_t \in \mathbb{R}^d, \tilde{\mathbf{x}}_t = \mathbf{r}_t \otimes \mathbf{x}_t$
one layer (CNN)	94.25	$\mathbf{r}_t \in \mathbb{R}^I, \tilde{\mathbf{x}}_t = \mathbf{r}_t \odot \mathbf{x}_t$
two layer(MLP)	94.43	Eq. (26)
weighted average	94.15	$\tilde{\mathbf{x}}_t = \sum_{i=1}^{d} r_i \mathbf{f_i}$

Table 3: Comparisons of different dynamic filters using a stacked Bi-LSTM model. "two layer" refers to using a two layer feed-forward neural network to learn dynamic filters. $\mathbf{r}_t \odot \mathbf{x}_t$ is a element-wise product.

Effects of the Dynamic Window Approach

Figure 3 shows the effectiveness of our dynamic window approach on a stacked Bi-LSTM model. We can observe at first 10 epochs the Bi-LSTM + dyn performs worse than the original Bi-LSTM model due to filtered inputs, but after that the performance of the Bi-LSTM + dyn improves about 0.3%.

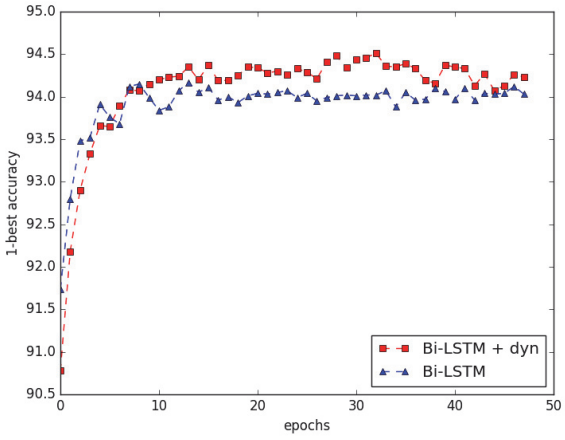
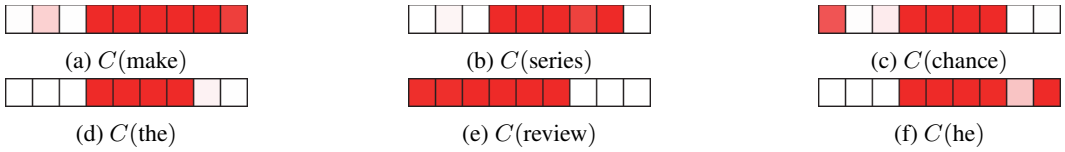


Figure 3: Comparison of a stacked Bi-LSTM with and without the dynamic window approach on the development set. "Bi-LSTM+dyn" denotes the stacked Bi-LSTM with the dynamic window approach.

Visualizations

Our model uses filter gates to dynamically choose the needed contexts. To understand this mechanism, we ran-



(a) $C(\text{make})$ (b) $C(\text{series})$ (c) $C(\text{chance})$

(d) $C(\text{the})$ (e) $C(\text{review})$ (f) $C(\text{he})$

Figure 4: Examples of the activations of filter gates for different contexts. (*red* indicates filters saturated near 1, *white* indicates filters unsaturated near 0), $C(\cdot)$ refers to the fixed context surrounding the center word.

domly choose some words to visualize the dynamic activities during training. All the visualizations are done using an MLP on CCGBank Section 02-21.

Figure 4 shows the different dynamic activities for the words. Each sub-figure has 9 blocks (a window size of 9), each of which shows an activation of one filter $l_i(t), i \in \{1, \ldots, 9\}$. After training to convergence, the items far away from the middle words are gradually removed from the contexts, while the attentions are gradually focused on the items nearby the central words. We can observe that for different words, their dynamic activities are different.

Related Work

Clark and Curran (2007) use a log-linear model to build the supertagger, using discrete feature functions for the targets based on the words and POS tags. The discrete property of the model makes the features independent of each other. Lewis and Steedman (2014) propose a semi-supervised supertagging model using a multi-layer perceptron (MLP) based on Collobert (2011) and conditional random field (CRF) proposed by Turian et al. (2010). Without using POS tags, they use the per-trained word embeddings with 2-character suffix and capitalization as features to represent the word. This distributed embedding encodes the word similarities and provides a better representation than log-linear models. However, MLP based supertagger ignores the sequential information, and their CRF based model can capture this but takes far more computational complexity than the MLP model due to the huge number of supertags.

The supertaggers based on log-linear and MLP are all point estimators, while CCG supertagging is more suitable to be treated as a sequential estimation problem due to long-range dependencies of the predicate-argument relations contained in lexical categories. Recently, Xu et al. (2015) design an Elman-type RNN to capture these dependencies, and use a fixed size window for each word as MLPs. The recurrent matrix in RNN can restore the historical information, which makes it outperform the MLP based model. But RNNs may suffer from the gradient vanishing/exploding problems and are not good at capturing long-range dependencies in practice. Vaswani et al. (2016) and Lewis et al. (2016) shows the effectiveness of Bi-LSTMs in supertagging, but they do not use a context window for the inputs. We only get 93.9% performance on the development set without using context windows. We find that a window size of 3 is needed in the stacked Bi-LSTM to get a better performance.

Our model can be treated as a marriage between attention mechanism and dropout. The most relevant attention-based models relating to our work is Wang et al. (2015), in which they use an attention model to find the relevant words within the context for predicting the center word. Their attention mechanism is similar to Bahdanau et al. (2014), while ours was not originally designed as a weighted average but a gated concatenation. Dropout on the dynamic window is similar to (Dai and Le 2015), which randomly drop words in the input sentences. Gal (2015) also use dropout on words, but using a fixed mask rather a random one.

Conclusion

We presented a dynamic window approach for CCG supertagging. Our model uses logistic gates to filter the context window surrounding the center word. This attention mechanism shows effectiveness on both MLPs and RNNs. We observed that using dropout on the dynamic window will greatly improve the generalization performance. We further visualized the activation of the filters, which is useful to help us understanding the dynamic activities. Although our work mainly focus on the CCG supertagging, this method can be easily applied to other sequence tagging tasks, such as POS tagging and named entity recognition (NER).

Acknowledgments

The research work has been funded by the Natural Science Foundation of China under Grant No. 61333018 No. 91520204 and supported by the Strategic Priority Research Program of the CAS under Grant No. XDB02070007. We thank the anonymous reviewers for their useful comments that greatly improved the manuscript.

References

Artzi, Y.; Lee, K.; and Zettlemoyer, L. 2015. Broad-coverage CCG semantic parsing with amr. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1699–1710. Association for Computational Linguistics.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bos, J.; Clark, S.; Steedman, M.; Curran, J. R.; and Hockenmaier, J. 2004. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th international conference on Computational Linguistics*, 1240. Association for Computational Linguistics.

Bos, J. 2005. Towards wide-coverage semantic interpretation. In *Proceedings of Sixth International Workshop on Computational Semantics IWCS*, volume 6, 42–53.

Bos, J. 2008. Wide-coverage semantic analysis with boxer. In *Proceedings of the 2008 Conference on Semantics in Text Processing*, 277–286. Association for Computational Linguistics.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Clark, S., and Curran, J. R. 2007. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics* 33(4):493–552.

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

Dai, A. M., and Le, Q. V. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, 3061–3069.

Elman, J. L. 1990. Finding structure in time. *Cognitive science* 14(2):179–211.

Gal, Y. 2015. A theoretically grounded application of dropout in recurrent neural networks. *arXiv preprint arXiv:1512.05287*.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, 249–256.

Hochreiter, S., and Schmidhuber, J. 1997. Lstm can solve hard long time lag problems. *Advances in neural information processing systems* 473–479.

Hockenmaier, J., and Steedman, M. 2007. Ccgbank: a corpus of CCG derivations and dependency structures extracted from the penn treebank. *Computational Linguistics* 33(3):355–396.

Iyyer, M.; Manjunatha, V.; and Boyd-Graber, J. 2015. Deep unordered composition rivals syntactic methods for text classification.

Jordan, M. I. 1986. Attractor dynamics and parallellism in a connectionist sequential machine.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kwiatkowski, T.; Zettlemoyer, L.; Goldwater, S.; and Steedman, M. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, 1223–1233. Association for Computational Linguistics.

Kwiatkowski, T.; Zettlemoyer, L.; Goldwater, S.; and Steedman, M. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1512–1523. Association for Computational Linguistics.

Lewis, M., and Steedman, M. 2013. Combining distributional and logical semantics. *Transactions of the Association for Computational Linguistics* 1:179–192.

Lewis, M., and Steedman, M. 2014. Improved CCG parsing with semi-supervised supertagging. *Transactions of the Association for Computational Linguistics* 2:327–338.

Lewis, M.; Lee, K.; and Zettlemoyer, L. 2016. Lstm ccg parsing. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Ling, W.; Chu-Cheng, L.; Tsvetkov, Y.; Amir, S.; Astudillo, R. F.; Dyer, C.; Black, A. W.; and Trancoso, I. 2015. Not all contexts are created equal: Better word representations with variable attention. EMNLP.

Marcus, M. P.; Marcinkiewicz, M. A.; and Santorini, B. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics* 19(2):313–330.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, 1532–43.

Plaut, D. C., et al. 1986. Experiments on learning by back propagation.

Saxe, A. M.; McClelland, J. L.; and Ganguli, S. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Steedman, M., and Baldridge, J. 2011. Combinatory categorial grammar. *Non-Transformational Syntax: Formal and Explicit Models of Grammar. Wiley-Blackwell*.

Steedman, M. 2000. *The syntactic process*, volume 24. MIT Press.

Turian, J.; Ratinov, L.; and Bengio, Y. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, 384–394. Association for Computational Linguistics.

Vaswani, A.; Bisk, Y.; Sagae, K.; and Musa, R. 2016. Supertagging with lstms. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Xu, W.; Auli, M.; and Clark, S. 2015. CCG supertagging with a recurrent neural network. *Volume 2: Short Papers* 250.

Xu, W.; Auli, M.; and Clark, S. 2016. Expected f-measure training for shift-reduce parsing with recurrent neural networks. In *Proceedings of NAACL-HLT*, 210–220.

Zeiler, M. D. 2012. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zettlemoyer, L. S., and Collins, M. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *EMNLP-CoNLL*, 678–687.