An Empirical Exploration of Skip Connections for Sequential Tagging

Huijia Wu[1,3], Jiajun Zhang[1,3], and 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

In this paper, we empirically explore the effects of various kinds of skip connections in stacked bidirectional LSTMs for sequential tagging. We investigate three kinds of skip connections connecting to LSTM cells: (a) skip connections to the gates, (b) skip connections to the internal states and (c) skip connections to the cell outputs. We present comprehensive experiments showing that skip connections to cell outputs outperform the remaining two. Furthermore, we observe that using gated identity functions as skip mappings works pretty well. Based on this novel skip connections, we successfully train deep stacked bidirectional LSTM models and obtain state-of-the-art results on CCG supertagging and comparable results on POS tagging.

1 Introduction

In natural language processing, sequential tagging mainly refers to the tasks of assigning discrete labels to each token in a sequence. Typical examples include part-of-speech (POS) tagging and combinatory category grammar (CCG) supertagging. A regular feature of sequential tagging is that the input tokens in a sequence cannot be assumed to be independent since the same token in different contexts can be assigned to different tags. Therefore, the classifier should have memories to remember the contexts to make a correct prediction.

Bidirectional LSTMs (Graves and Schmidhuber, 2005) become dominant in sequential tagging problems due to the superior performance (Wang et al., 2015; Vaswani et al., 2016; Lample et al., 2016). The horizontal hierarchy of LSTMs with bidirectional processing can remember the long-range dependencies without affecting the short-term storage. Although the models have a deep horizontal hierarchy (the depth is the same as the sequence length), the vertical hierarchy is often shallow, which may not be efficient at representing each token. Stacked LSTMs are deep in both directions, but become harder to train due to the feed-forward structure of stacked layers.

Skip connections (or shortcut connections) enable unimpeded information flow by adding direct connections across different layers (Raiko et al., 2012; Graves, 2013; Hermans and Schrauwen, 2013). However, there is a lack of exploration and analyzing various kinds of skip connections in stacked LSTMs. There are two issues to handle skip connections in stacked LSTMs: One is where to add the skip connections, the other is what kind of skip connections should be used to pass the information. To answer the first question, we empirically analyze three positions of LSTM blocks to receive the previous layer's output. For the second one, we present an identity mapping to receive the previous layer's outputs. Furthermore, following the gate design of LSTM (Hochreiter and Schmidhuber, 1997; Gers et al., 2000) and highway networks (Srivastava et al., 2015a; Srivastava et al., 2015b), we observe that adding a multiplicative gate to the identity function will help to improve performance.

In this paper, we present a neural architecture for sequential tagging. The input of the network are token representations. We concatenate word embeddings to character embeddings to represent the word and morphemes. A deep stacked bidirectional LSTM with well-designed skip connections is then used

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 203–212, Osaka, Japan, December 11-17 2016.

to extract the features needed for classification from the inputs. The output layer uses *softmax* function to output the tag distribution for each token.

Our main contribution is that we empirically evaluated the effects of various kinds of skip connections within stacked LSTMs. We present comprehensive experiments on the supertagging task showing that skip connections to the cell outputs using identity function multiplied with an exclusive gate can help to improve the network performance. Our model is evaluated on two sequential tagging tasks, obtaining state-of-the-art results on CCG supertagging and comparable results on POS tagging.

2 Related Work

Skip connections have been widely used for training deep neural networks. For recurrent neural networks, Schmidhuber (1992); El Hihi and Bengio (1995) introduced deep RNNs by stacking hidden layers on top of each other. Raiko et al. (2012); Graves (2013); Hermans and Schrauwen (2013) proposed the use of skip connections in stacked RNNs. However, the researchers have paid less attention to the analyzing of various kinds of skip connections, which is our focus in this paper.

The works closely related to ours are Srivastava et al. (2015b), He et al. (2015), Kalchbrenner et al. (2015), Yao et al. (2015), Zhang et al. (2016), and Zilly et al. (2016). These works are all based on the design of extra connections between different layers. Srivastava et al. (2015b) and He et al. (2015) mainly focus on feed-forward neural network, using well-designed skip connections across different layers to make the information pass more easily. The Grid LSTM proposed by Kalchbrenner et al. (2015) extends the one dimensional LSTMs to many dimensional LSTMs, which provides a more general framework to construct deep LSTMs.

Yao et al. (2015) and Zhang et al. (2016) propose highway LSTMs by introducing gated direct connections between internal states in adjacent layers and do not use skip connections, while we propose gated skip connections across cell outputs. Zilly et al. (2016) introduce recurrent highway networks (RHN) which use a single recurrent layer to make RNN deep in a vertical direction, in contrast to our stacked models.

3 Recurrent Neural Networks for Sequential Tagging

Consider a recurrent neural network applied to sequential tagging: Given a sequence $x = (x_1, \ldots, x_T)$, the RNN computes the hidden state $h = (h_1, \ldots, h_T)$ and the output $y = (y_1, \ldots, y_T)$ by iterating the following equations:

$$h_t = f(x_t, h_{t-1}; \theta_h) \tag{1}$$

$$y_t = g(h_t; \theta_o) \tag{2}$$

where $t \in \{1, \ldots, T\}$ represents the time. x_t represents the input at time t, h_{t-1} and h_t are the previous and the current hidden state, respectively. f and g are the transition function and the output function, respectively. θ_h and θ_o are network parameters.

We use a negative log-likelihood cost to evaluate the performance, which can be written as:

$$\mathcal{C} = -\frac{1}{N} \sum_{n=1}^{N} \log y_{t^n} \tag{3}$$

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

The core idea of Long Short-Term Memory networks is to replace (1) with the following equation:

$$c_t = f(x_t, h_{t-1}) + c_{t-1} \tag{4}$$

where c_t is the internal state of the memory cell, which is designed to store the information for much longer time. Besides this, LSTM uses gates to avoid weight update conflicts.

Standard LSTMs process sequences in temporal order, which will ignore future context. Bidirectional LSTMs solve this problem by combining both the forward and the backward processing of the input sequences using two separate recurrent hidden layers:

$$\overrightarrow{h_t} = \text{LSTM}(\overrightarrow{x_t}, \overrightarrow{h_{t-1}}, \overrightarrow{c_{t-1}}) \tag{5}$$

$$\overleftarrow{h_t} = \text{LSTM}(\overleftarrow{x_t}, \overleftarrow{h_{t-1}}, \overleftarrow{c_{t-1}}) \tag{6}$$

$$y_t = g(\overrightarrow{h_t}, \overleftarrow{h_t}) \tag{7}$$

where LSTM$(\cdot)$ is the LSTM computation. $\overrightarrow{x_t}$ and $\overleftarrow{x_t}$ are the forward and the backward input sequence, respectively. The output of the two hidden layers $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ in a birectional LSTM are connected to the output layer.

Stacked RNN is one type of deep RNNs, which refers to the hidden layers are stacked on top of each other, each feeding up to the layer above:

$$h_t^l = f^l(h_t^{l-1}, h_{t-1}^l) \tag{8}$$

where h_t^l is the t-th hidden state of the l-th layer.

4 Various kinds of Skip Connections

Skip connections in simple RNNs are trivial since there is only one position to connect to the hidden units. But for stacked LSTMs, the skip connections need to be carefully treated to train the network successfully. In this section, we analyze and compare various types of skip connections. At first, we give a detailed definition of stacked LSTMs, which can help us to describe skip connections. Then we start our construction of skip connections in stacked LSTMs. At last, we formulate various kinds of skip connections.

Stacked LSTMs without skip connections can be defined as:

$$\begin{pmatrix} i_t^l \\ f_t^l \\ o_t^l \\ s_t^l \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{tanh} \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \qquad \begin{aligned} c_t^l &= f_t^l \odot c_{t-1}^l + i_t^l \odot s_t^l \\ h_t^l &= o_t^l \odot \tanh(c_t^l) \end{aligned} \tag{9}$$

During forward pass, LSTM needs to calculate c_t^l and h_t^l, which is the cell's internal state and the cell outputs state, respectively. To get c_t^l, s_t^l needs to be computed to store the current input. Then this result is multiplied by the input gate i_t^l, which decides when to keep or override information in memory cell c_t^l. The cell is designed to store the previous information c_{t-1}^l, which can be reset by a forget gate f_t^l. The new cell state is then obtained by adding the result to the current input. The cell outputs h_t^l are computed by multiplying the activated cell state by the output gate o_t^l, which learns when to access memory cell and when to block it. "sigm" and "tanh" are the sigmoid and tanh activation function, respectively. $W^l \in \mathbb{R}^{4n \times 2n}$ is the weight matrix needs to be learned.

The hidden units in stacked LSTMs have two forms. One is the hidden units in the same layer $\{h_t^l, t \in 1, \ldots, T\}$, which are connected through an LSTM. The other is the hidden units at the same time step $\{h_t^l, l \in 1, \ldots, L\}$, which are connected through a feed-forward network. LSTM can keep the short-term memory for a long time, thus the error signals can be easily passed through $\{1, \ldots, T\}$. However, when the number of stacked layers is large, the feed-forward network will suffer the gradient vanishing/exploding problems, which make the gradients hard to pass through $\{1, \ldots, L\}$.

The core idea of LSTM is to use an identity function to make the constant error carrosel. He et al. (2015) also use an identity mapping to train a very deep convolution neural network with improved performance. All these inspired us to use an identity function for the skip connections. Rather, the gates of LSTM are essential parts to avoid weight update conflicts, which are also invoked by skip connections. Following highway gating, we use a gate multiplied with identity mapping to avoid the conflicts.

Skip connections are cross-layer connections, which means that the output of layer $l-2$ is not only connected to the layer $l-1$, but also connected to layer l. For stacked LSTMs, h_t^{l-2} can be connected to the gates, the internal states, and the cell outputs in layer l's LSTM blocks. We formalize these below:

Skip connections to the gates. We can connect h_t^{l-2} to the gates through an identity mapping:

$$\begin{pmatrix} i_t^l \\ f_t^l \\ o_t^l \\ s_t^l \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \left(W^l \; I^l \right) \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \\ h_t^{l-2} \end{pmatrix} \tag{10}$$

where $I^l \in \mathbb{R}^{4n \times n}$ is the identity mapping.

Skip connections to the internal states. Another kind of skip connections is to connect h_t^{l-2} to the cell's internal state c_t^l:

$$c_t^l = f_t^l \odot c_{t-1}^l + i_t^l \odot s_t^l + h_t^{l-2} \tag{11}$$
$$h_t^l = o_t^l \odot \tanh(c_t^l) \tag{12}$$

Skip connections to the cell outputs. We can also connect h_t^{l-2} to cell outputs:

$$c_t^l = f_t^l \odot c_{t-1}^l + i_t^l \odot s_t^l \tag{13}$$
$$h_t^l = o_t^l \odot \tanh(c_t^l) + h_t^{l-2} \tag{14}$$

Skip connections using gates. Consider the case of skip connections to the cell outputs. The cell outputs grow linearly during the presentation of network depth, which makes the h_t^l's derivative vanish and hard to convergence. Inspired by the introduction of LSTM gates, we add a gate to control the skip connections through retrieving or blocking them:

$$\begin{pmatrix} i_t^l \\ f_t^l \\ o_t^l \\ g_t^l \\ s_t^l \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \qquad \begin{aligned} c_t^l &= f_t^l \odot c_{t-1}^l + i_t^l \odot s_t^l \\ h_t^l &= o_t^l \odot \tanh(c_t^l) + g_t^l \odot h_t^{l-2} \end{aligned} \tag{15}$$

where g_t^l is the gate which can be used to access the skipped output h_t^{l-2} or block it. When g_t^l equals 0, no skipped output can be passed through skip connections, which is equivalent to traditional stacked LSTMs. Otherwise, it behaves like a feed-forward LSTM using gated identity connections. Here we omit the case of adding gates to skip connections to the internal state, which is similar to the above case.

Skip connections in bidirectional LSTM. Using skip connections in bidirectional LSTM is similar to the one used in LSTM, with a bidirectional processing:

$$\begin{aligned} \overrightarrow{c_t^l} &= \overrightarrow{f} \odot \overrightarrow{c_{t-1}^l} + \overrightarrow{i} \odot \overrightarrow{s_t^l} \\ \overrightarrow{h_t^l} &= \overrightarrow{o} \odot \tanh(\overrightarrow{c_t^l}) + \overrightarrow{g} \odot \overrightarrow{h_t^{l-2}} \end{aligned} \qquad \begin{aligned} \overleftarrow{c_t^l} &= \overleftarrow{f} \odot \overleftarrow{c_{t-1}^l} + \overleftarrow{i} \odot \overleftarrow{s_t^l} \\ \overleftarrow{h_t^l} &= \overleftarrow{o} \odot \tanh(\overleftarrow{c_t^l}) + \overleftarrow{g} \odot \overleftarrow{h_t^{l-2}} \end{aligned} \tag{16}$$

5 Neural Architecture for Sequential Tagging

Sequential tagging can be formulated as $P(t|w;\theta)$, where $w = [w_1, \ldots, w_T]$ indicates the T words in a sentence, and $t = [t_1, \ldots, t_T]$ indicates the corresponding T tags. In this section we introduce an neural architecture for $P(\cdot)$, which includes an input layer, a stacked hidden layers and an output layer. Since the stacked hidden layers have already been introduced, we only introduce the input and the output layer here.

5.1 Network Inputs

Network inputs are the representation of each token in a sequence. There are many kinds of token representations, such as using a single word embedding, using a local window approach, or a combination of word and character-level representation. Here our inputs contain the concatenation of word representations, character representations, and capitalization representations.

Word representations. All words in the vocabulary share a common look-up table, which is initialized with random initializations or pre-trained embeddings. Each word in a sentence can be mapped to an embedding vector w_t. The whole sentence is then represented by a matrix with columns vector $[w_1, w_2, \ldots, w_T]$. We use a context window of size d surrounding with a word w_t to get its context information. Following Wu et al. (2016), we add logistic gates to each token in the context window. The word representation is computed as $w_t = [r_{t-\lfloor d/2 \rfloor} w_{t-\lfloor d/2 \rfloor}; \ldots; r_{t+\lfloor d/2 \rfloor} w_{t+\lfloor d/2 \rfloor}]$, where $r_t := [r_{t-\lfloor d/2 \rfloor}, \ldots, r_{t+\lfloor d/2 \rfloor}] \in \mathbb{R}^d$ is a logistic gate to filter the unnecessary contexts, $w_{t-\lfloor d/2 \rfloor}, \ldots, w_{t+\lfloor d/2 \rfloor}$ is the word embeddings in the local window.

Character representations. Prefix and suffix information about words are important features in sequential tagging. Inspired by Fonseca et al. (2015) et al, which uses a character prefix and suffix with length from 1 to 5 for part-of-speech tagging, we concatenate character embeddings in a word to get the character-level representation. Concretely, given a word w consisting of a sequence of characters $[c_1, c_2, \ldots, c_{l_w}]$, where l_w is the length of the word and $L(\cdot)$ is the look-up table for characters. We concatenate the leftmost most 5 character embeddings $L(c_1), \ldots, L(c_5)$ with its rightmost 5 character embeddings $L(c_{l_w-4}), \ldots, L(c_{l_w})$. When a word is less than five characters, we pad the remaining characters with the same special symbol.

Capitalization representations. We lowercase the words to decrease the size of word vocabulary to reduce sparsity, but we need an extra capitalization embeddings to store the capitalization features, which represent whether or not a word is capitalized.

5.2 Network Outputs

For sequential tagging, we use a *softmax* activation function $g(\cdot)$ in the output layer:

$$y_t = g(W^{hy}[\overrightarrow{h_t}; \overleftarrow{h_t}]) \tag{17}$$

where y_t is a probability distribution over all possible tags. $y_k(t) = \frac{\exp(h_k)}{\sum_{k'} \exp(h_{k'})}$ is the k-th dimension of y_t, which corresponds to the k-th tags in the tag set. W^{hy} is the hidden-to-output weight.

6 Experiments

6.1 Combinatory Category Grammar Supertagging

Combinatory Category Grammar (CCG) supertagging is a sequential tagging problem in natural language processing. The task is to assign supertags to each word in a sentence. In CCG the supertags stand for the lexical categories, which are composed of the basic categories such as N, NP and PP, and complex categories, which are the combination of the basic categories based on a set of rules. Detailed explanations of CCG refers to (Steedman, 2000; Steedman and Baldridge, 2011).

The training set of this task only contains 39604 sentences, which is too small to train a deep model, and may cause over-parametrization. But we choose it since it has been already proved that a bidirectional recurrent net fits the task by many authors (Lewis et al., 2016; Vaswani et al., 2016).

6.1.1 Dataset and Pre-processing

Our experiments are performed on CCGBank (Hockenmaier and Steedman, 2007), which is a translation from Penn Treebank (Marcus et al., 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.

6.1.2 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 et al., 2014). Other weights are initialized with the Gaussian distribution

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
7-layers + skip output + gating	94.51	94.67
7-layers + skip output + gating (no char)	94.33	94.45
7-layers + skip output + gating (no dropout)	94.06	94.0
9-layers + skip output + gating	**94.55**	**94.69**

Table 1: 1-best supertagging accuracy on CCGbank. "skip output" refers to the skip connections to the cell output, "gating" refers to adding a gate to the identity function, "no char" refers to the models that do not use the character-level information, "no dropout" refers to models that do not use dropout.

$\mathcal{N}(0, \frac{1}{\sqrt{\text{fan-in}}})$ scaled by a factor of 0.1, where *fan-in* is the number of units in the input layer. For recurrent weight matrices, following Saxe et al. (2013) we initialize with random orthogonal matrices through SVD to avoid unstable gradients. Orthogonal initialization for recurrent weights is important in our experiments, which takes about 2% relative performance enhancement than other methods such as Xavier initialization (Glorot and Bengio, 2010).

Hyperparameters. For the word representations, we use a small window size of 3 for the convolutional layer. The dimension of the word representation after the convolutional operation is 600. The size of character embedding and capitalization embeddings are set to 5. The number of cells of the stacked bidirectional 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 equal 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 trained the 7-layer network for roughly 2 to 3 days on one NVIDIA TITAN X GPU using Theano [1] (Team et al., 2016).

Regularization. Dropout (Srivastava et al., 2014) is the only regularizer in our model to avoid overfitting. Other regularization methods such as weight decay and batch normalization do not work in our experiments. We add a binary dropout mask to the local context windows on the embedding layer, with a drop rate p of 0.25. We also apply dropout to the output of the first hidden layer and the last hidden layer, with a 0.5 drop rate. At test time, weights are scaled with a factor $1 - p$.

6.1.3 Results

Table 1 shows the comparisons with other models for supertagging. The comparisons do not include any externally labeled data and POS labels. We use stacked bidirectional LSTMs with gated skip connections for the comparisons, and report the highest 1-best supertagging accuracy on the development set for final testing. Our model presents state-of-the-art results compared to the existing systems. The character-level information (+ 3% relative accuracy) and dropout (+ 8% relative accuracy) are necessary to improve the performance.

[1] `http://deeplearning.net/software/theano/`

6.1.4 Experiments on Skip Connections

We experiment with a 7-layer model on CCGbank to compare different kinds of skip connections introduced in Section 4. Our analysis mainly focuses on the identity function and the gating mechanism. The comparisons (Table 2) are summarized as follows:

No skip connections. When the number of stacked layers is large, the performance will degrade without skip connections. The accuracy in a 7-layer stacked model without skip connections is 93.94% (Table 2), which is lower than the one using skip connections.

Various kinds of skip connections. We experiment with the gated identity connections between internal states introduced in Zhang et al.(2016), but the network performs not good (Table 2, 93.14%). We also implement the method proposed in Zilly et al. (2016), which we use a single bidirectional RNH layer with a recurrent depth of 3 with a slightly modification[2]. Skip connections to the cell outputs with identity function and multiplicative gating achieves the highest accuracy (Table 2, 94.51%) on the development set. We also observe that skip to the internal states without gate get a slightly better performance (Table 2, 94.33%) than the one with gate (94.24%) on the development set. Here we recommend to set the forget bias to 0 to get a better development accuracy.

Identity mapping. We use the sigmoid function to the previous outputs to break the identity link, in which we replace $g_t \odot h_t^{l-1}$ in Eq. (15) with $g_t \odot \sigma(h_t^{l-1})$, where $\sigma(x) = \frac{1}{1+e^{-x}}$. The result of the sigmoid function is 94.02% (Table 2), which is poor than the identity function. We can infer that the identity function is more suitable than other scaled functions such as sigmoid or tanh to transmit information.

Exclusive gating. Following the gating mechanism adopted in highway networks, we consider adding a gate g_t to make a flexible control to the skip connections. Our gating function is $g_t^l = \sigma(W_g^l h_{t-1}^l + U_g^l h_t^{l-2})$. Gated identity connections are essential to achieving state-of-the-art result on CCGbank.

Case	Variant	Dev	Test
H-LSTM, Zhang et al.(2016)	-	93.14	93.52
RHN, Zilly et al. (2016)	$L = 3$, with output gates	94.28	94.24
no skip connections	-	93.94	94.26
to the gates, Eq. (10)	-	93.9	94.22
to the internals	no gate, Eq. (11)	94.33	94.63
	with gate	94.24	94.52
to the outputs	no gate, Eq. (14)	93.89	93.98
	with gate, $b_f = 5$, Eq. (15)	94.23	94.81
	with gate, $b_f = 0$, Eq. (15)	94.51	94.67
	sigmoid mapping: $g_t \odot \sigma(h_t^{l-1})$	94.02	94.18

Table 2: Accuracy on CCGbank using 7-layer stacked bidirectional LSTMs, with different types of skip connections. b_f is the bias of the forget gate.

6.1.5 Experiments on Number of Layers

Table 3 compares the effect of the depth in the stacked models. We can observe that the performance is getting better with the increased number of layers. But when the number of layers exceeds 9, the performance will be hurt. In the experiments, we found that the number of stacked layers between 7 and 9 are the best choice using skip connections. Notice that we do not use layer-wise pretraining (Bengio et al., 2007; Simonyan and Zisserman, 2014), which is an important technique in training deep networks.

[2]Our original implementation of Zilly et a. (2016) with a recurrent depth of 3 fails to converge. The reason might be due to the explosion of s_L^t under addition. To avoid this, we replace s_L^t with $o_t * \tanh(s_L^t)$ in the last recurrent step.

Further improvements might be obtained with this method to build a deeper network with improved performance.

#Layers	Dev	Test
3	94.21	94.35
5	94.51	94.57
7	94.51	94.67
9	94.55	94.7
11	94.43	94.65

Table 3: Accuracy on CCGbank using gated identity connections to cell outputs, with different number of stacked layers.

6.2 Part-of-speech Tagging

Part-of-speech tagging is another sequential tagging task, which is to assign POS tags to each word in a sentence. It is very similar to the supertagging task. Therefore, these two tasks can be solved in a unified architecture. For POS tagging, we use the same network configurations as supertagging, except the word vocabulary size and the tag set size. We conduct experiments on the Wall Street Journal of the Penn Treebank dataset, adopting the standard splits (sections 0-18 for the train, sections 19-21 for validation and sections 22-24 for testing).

Model	Test
Søgaard (2011)	97.5
Ling et al. (2015)	97.36
Wang et al. (2015)	**97.78**
Vaswani et al. (2016)	97.4
7-layers + skip output + gating	97.45
9-layers + skip output + gating	97.45

Table 4: Accuracy for POS tagging on WSJ.

Although the POS tagging result presented in Table 4 is slightly below the state-of-the-art, we neither do any parameter tunings nor change the network architectures, just use the one getting the best development accuracy on the supertagging task. This proves the generalization of the model and avoids heavy work of model re-designing.

7 Conclusions

This paper investigates various kinds of skip connections in stacked bidirectional LSTM models. We present a deep stacked network (7 or 9 layers) that can be easily trained and get improved accuracy on CCG supertagging and POS tagging. Our experiments show that skip connections to the cell outputs with the gated identity function performs the best. Our explorations could easily be applied to other sequential processing problems, which can be modelled with RNN architectures.

Acknowledgements

The research work has been funded by the Natural Science Foundation of China under Grant No. 61333018 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

Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. 2007. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153.

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

Salah El Hihi and Yoshua Bengio. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, volume 400, page 409. Citeseer.

Erick R Fonseca, João Luís G Rosa, and Sandra Maria Aluísio. 2015. Evaluating word embeddings and a revised corpus for part-of-speech tagging in portuguese. *Journal of the Brazilian Computer Society*, 21(1):1–14.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471.

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

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

Michiel Hermans and Benjamin Schrauwen. 2013. Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 190–198.

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

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

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. 2015. Grid long short-term memory. *arXiv preprint arXiv:1507.01526*.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

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

Mike Lewis, Kenton Lee, and Luke Zettlemoyer. 2016. Lstm ccg parsing. In *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*.

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

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–43.

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

Tapani Raiko, Harri Valpola, and Yann LeCun. 2012. Deep learning made easier by linear transformations in perceptrons. In *AISTATS*, volume 22, pages 924–932.

Andrew M Saxe, James L McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

Jürgen Schmidhuber. 1992. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Anders Søgaard. 2011. Semisupervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 48–52. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015a. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015b. Highway networks. *arXiv preprint arXiv:1505.00387*.

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

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

Theano Development Team, Rami Alrfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frdric Bastien, Justin Bayer, and Anatoly Belikov. 2016. Theano: A python framework for fast computation of mathematical expressions.

Ashish Vaswani, Yonatan Bisk, Kenji Sagae, and Ryan Musa. 2016. Supertagging with lstms. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Peilu Wang, Yao Qian, Frank K Soong, Lei He, and Hai Zhao. 2015. Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv:1510.06168*.

Huijia Wu, Jiajun Zhang, and Chengqing Zong. 2016. A dynamic window neural network for ccg supertagging. *arXiv preprint arXiv:1610.02749*.

Wenduan Xu, Michael Auli, and Stephen Clark. 2015. CCG supertagging with a recurrent neural network. *Volume 2: Short Papers*, page 250.

Wenduan Xu, Michael Auli, and Stephen Clark. 2016. Expected f-measure training for shift-reduce parsing with recurrent neural networks. In *Proceedings of NAACL-HLT*, pages 210–220.

Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. 2015. Depth-gated lstm. In *Presented at Jelinek Summer Workshop on August*, volume 14, page 1.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. 2016. Highway long short-term memory rnns for distant speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5755–5759. IEEE.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2016. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*.