Shortcut Sequence Tagging

Huijia Wu[1,3], Jiajun Zhang[1,2], 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. Deep stacked RNNs are usually hard to train. Recent studies have shown that shortcut connections across different RNN layers bring substantially faster convergence. However, shortcuts increase the computational complexity of the recurrent computations. To reduce the complexity, we propose the shortcut block, which is a refinement of the shortcut LSTM blocks. Our approach is to replace the self-connected parts (c_t^l) with shortcuts (h_t^{l-2}) in the internal states. We present extensive empirical experiments showing that this design performs better than the original shortcuts. We evaluate our method on CCG supertagging task, obtaining a 8% relatively improvement over current state-of-the-art results.

1 Introduction

In natural language processing, sequence 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 sequence 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 Long Short-Term Memory (Bi-LSTM) [5] become dominant in sequence tagging problems due to the superior performance [13, 25]. 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 refers to sequence length), the vertical hierarchy is often shallow, which may not be efficient at representing each token. Stacked LSTMs are deep in both dimensions, but become harder to train due to the feed-forward structure of stacked layers.

Shortcut connections (shortcuts, or skip connections) enable unimpeded information flow by adding direct connections across different layers [4, 7]. Recent works have shown the effectiveness of using shortcuts in deep stacked models [6, 21, 27]. These works share a common way of adding shortcuts as increments to the original network.

In this paper, we focus on the refinement of shortcut stacked models to reduce the computational cost. Concretely, we replace the self-connected parts in LSTM

cells with the gated shortcuts to simplify the updates, while preserving the recurrent information flow through cell outputs. We also investigate deterministic or stochastic gates to find the preferable way to control the shortcut connections.

We present extensive experiments on the Combinatory Category Grammar (CCG) supertagging task to compare shortcut block varieties, gating functions, and combinations of the blocks. Our model obtains the state-of-the-art results on CCG supertagging.

2 Recurrent Neural Networks for Sequence Tagging

Given a sequence $x = (x_1, \ldots, x_T)$, a recurrent neural network (RNN) computes the hidden states $h = (h_1, \ldots, h_T)$ and the outputs $y = (y_1, \ldots, y_T)$ 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 the recurrent information h_{t-1} and the inputs x_t are processed through the iteration function $f(\cdot, \cdot)$ to compute h_t.

There are many iteration functions to pass the information through the sequence. Long Short-Term Memory (LSTM) is one kind of function that preserves the hidden activations h_t for a long time. The computation of the recurrent flow in LSTM is:

$$c_t = c_{t-1} + \Delta \tag{3}$$

where c_t are the cells' internal states that used to iterate the recurrent information. Δ refer to the increments to the cells. Adding such increments through time make the gradient of the internal states more stable. Therefore, the short-term memory can be kept for a long time under this construction.

Another advantage of LSTM is the gating mechanism, which is used to avoid weight update conflicts. The activation of the gates decides when to keep or override information in the controlled units.

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

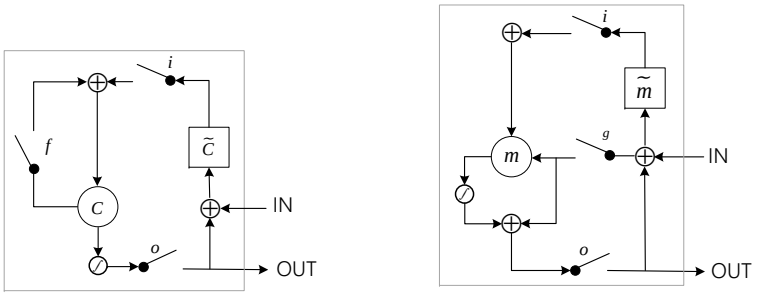
$$\mathcal{C} = -\frac{1}{N} \sum_{n=1}^{N} \log y_{t^n} \tag{4}$$

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.

3 Exploration of Shortcuts

3.1 Shortcut Blocks

The hidden units in stacked LSTMs can be divided into two parts: One is the hidden units within the same layer $\{h_t^l, t \in 1, \ldots, T\}$, which are connected through



(a) Long Short-Term Memory Block (b) Shortcut Block

Fig. 1: Illustration of (a) LSTM block and (b) shortcut block. (a) i, f and o represent input , forget and output gate, respectively. $\tilde{C}$ denotes the increments to the cell, and C denotes the cell's internal state. (b) g is the shortcut gate, $\tilde{m}$ denotes the cell inputs, and m denotes the internal state of the cell. Notice that (b) replaces the self-connected links (through f gate) in (a) with shortcut connections (through g gate).

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 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 from gradient vanishing/exploding problems, which make the gradients hard to pass through $\{1, \ldots, L\}$.

Shortcut connections make training much easier by adding a direct link between different layers. An intuitive explanation is that such links can make the error signal passing jump the layers, not just one by one. This behavior may lead to faster convergence and better generalization.

LSTM block [8] composes memory cells sharing the same input and output gate. He et al. [6] create a residual block which adds shortcut connections across different CNN layers. All these inspired us to build a shortcut block across different LSTM layers. Our shortcut block is mainly based on Wu et al. [27]. We start our construction from a special case of shortcuts: adding shortcuts to both c_t and h_t.

$$\begin{pmatrix} i \\ f \\ o \\ s \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \tag{5}$$

$$g = \text{sigm}(U^l h_t^{l-1} + V^l h_t^{l-2}) \tag{6}$$

$$c_t = f \odot c_{t-1} + i \odot s_t^l + g \odot h_t^{l-2} \tag{7}$$

$$h_t^l = o \odot \tanh(c_t) + g \odot h_t^{l-2} \tag{8}$$

We want to simplify the computation of the cell updates. Notice that the internal states contain three parts (Eq. 7): the self-connected parts c_{t-1}, the original cell inputs s_t^l, and the shortcuts h_t^{l-2}. We observe that there exists two kinds of recurrent information flow in the internal states: one is the recurrent flow along the horizontal direction, controlled by self-connected parts, the other is along the vertical direction, controlled by shortcuts.

We find that omit the self-connected parts $f_t \odot c_{t-1}$ in Eq.(7) is helpful to bring faster convergence and improve the performance. An intuitive explanation is that the input sequence and the output sequence are exactly matched in sequence tagging. Specifically, the input token $x_i, i \in \{1, \ldots, n\}$ in a sequence provides the most relevant information to predict y_i. We want to enhance the information flow in the vertical direction through shortcuts, while decreasing the horizontal flow controlled by the self-connected units. Based on the observations, we get the following construction:

$$\begin{pmatrix} i \\ o \\ s \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} W^l \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \tag{9}$$

$$\begin{aligned} g &= \text{sigm}(U^l h_t^{l-1} + V^l h_t^{l-2}) \\ m &= i \odot s_t^l + g \odot h_t^{l-2} \\ h_t^l &= o \odot \tanh(m) + g \odot h_t^{l-2} \end{aligned} \tag{10}$$

where h_t^{l-2} are the outputs from layer $l-2$. g is the gate which is used to access the shortcut connections h_t^{l-2} or block it (See Figure 1b).

Comparison with Wu et al. [27] Wu et al. [27] introduced gated shortcuts connecting to cell outputs:

$$\begin{aligned} c_t^l &= i \odot s_t^l + f \odot c_{t-1}^l \\ h_t^l &= o \odot \tanh(c_t) + g \odot h_t^{l-2} \end{aligned} \tag{11}$$

Comparison of (10) and (11) we can see the difference: Eq.(10) replaces the self-connected parts $f \odot c_{t-1}^l$ with shortcuts $g \odot h_t^{l-2}$ in the computation of the internal states.

Although the difference is tiny, our refinement is much easier to compute, since we do not need extra space to preserve the cell's internal state. Similarly, this behavior arises in the construction of Gated Recurrent Units (GRUs):

$$\begin{aligned} \tilde{h}_t^l &= \tanh(W^l h_t^{l-1} + U^l (r_t \odot h_{t-1}^l)) \\ h_t^l &= (1 - z_t) \odot h_{t-1}^l + z_t \odot \tilde{h}_t^l \end{aligned} \tag{12}$$

But the recurrent iterations in GRUs are very different from us. Our construction is built on the stacked LSTMs.

Comparison with LSTMs. LSTMs introduce a memory cell with a fixed self-connection to make the constant error flow (Figure 1a). LSTM compute the following increment to the self-connected cell at each time step:

$$c_t = c_{t-1} + s_t \tag{13}$$

Here we remove the multiplicative gates to simplify the explanation. The self-connected cells c_t can keep the recurrent information for a long time. s_t are the cell inputs.

In the shortcut block, we use the shortcuts to replace the self-connected parts. Our cell states become:

$$m = h_t^{l-2} + s_t \tag{14}$$

Although we ignore the self-connected parts in LSTM cells, it does not mean we throw away the recurrent information. As shown in Figure 1b, the connections from cell outputs to inputs preserve the recurrent information flow.

3.2 Gates Computation

Shortcut gates are used to make the skipped path deterministic [21] or stochastic [10]. We explore many ways to compute the shortcut gates (denoted by g_t^l). The simplest case is to use g_t^l as a linear operator. In this case, g_t^l is a weight matrix, and the element-wise product $g_t^l \odot h_t^{-l}$ in (10) becomes a matrix-vector multiplication:

$$g_t^l \odot h_t^{-l} := W^l h_t^{-l} \tag{15}$$

We can also get g_t^l under a non-linear mapping, which is similar to the computation of gates in LSTM:

$$g_t^l = \sigma(W^l h_t^{l-1}) \tag{16}$$

Here we use the output of layer $l-1$ to control the shortcuts. Notice that this non-linear mapping is not unique, we just show the simplest case.

Furthermore, inspired by the dropout [20] strategy, we can sample from a Bernoulli stochastic variable to get g_t^l. In this case, the gate is stochastic:

$$g_t^l \sim \text{Bernoulli}(p) \tag{17}$$

where g_t^l is a vector of independent Bernoulli random variables each of which has probability p of being 1. We can either fix p with a specific value or learn it with a non-linear mapping. For example, we can learn p by:

$$p = \sigma(H^l h_t^{l-1}) \tag{18}$$

At test time, h_t^{-l} is multiplied by p.

Discussion. The gates of LSTMs are essential parts to avoid weight update conflicts, which are also invoked by the shortcuts. In experiments, we find that using deterministic gates is better than the stochastic gates. We recommend using the logistic gates to compute g_t^l.

4 Neural Architecture for Sequence Tagging

Sequence 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 in the previous section, we only introduce the input and the output layer here.

4.1 Network Inputs

The inputs of the network are the distributed representation of each token in a sequence. Following Wu et al. [27], we use a local window approach together with a concatenation of word representations, character representations, and capitalization representations:

$$f_{w_t} = [L_w(w_t); L_a(a_t); L_c(c_w)] \tag{19}$$

where w_t, a_t represent the current word and its capitalization. $c_w := [c_1, c_2, \ldots, c_{T_w}]$, 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. $L_w(\cdot) \in \mathbb{R}^{|V_w| \times n}$, $L_a(\cdot) \in \mathbb{R}^{|V_a| \times m}$ and $L_c(\cdot) \in \mathbb{R}^{|V_c| \times r}$ are the look-up tables for the words, capitalization and characters, respectively. f_{w_t} represents the distributed feature of w_t. A context window of size d surrounding the current word is used as an input:

$$x_t = [f_{w_{t-\lfloor d/2 \rfloor}}; \ldots; f_{w_{t+\lfloor d/2 \rfloor}}] \tag{20}$$

where x_t is a concatenation of the context features.

4.2 Network Outputs

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

$$y_t = g(W^{hy}[\overrightarrow{h_t}; \overleftarrow{h_t}]) \tag{21}$$

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 tag in the tag set. W^{hy} is the hidden-to-output weight.

5 Experiments

5.1 Combinatory Category Grammar Supertagging

We evaluate our method on Combinatory Category Grammar (CCG) supertagging task, which is a sequence 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 refer to [22, 23].

Dataset and Pre-processing Our experiments are performed on CCGBank [9], which is a translation from Penn Treebank [16] 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 100-dimensional GolVe vectors [17]. 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, following [18] we initialize with random orthogonal matrices through SVD to avoid unstable gradients. All bias terms are initialized with zero vectors.

Hyperparameters. Our context window size is set to 3. The dimension of character embedding and capitalization embeddings are 5. The number of cells of the stacked bidirectional LSTM is also set to 465 for orthogonal initialization. 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 use stochastic gradient descent (SGD) algorithm with an initial learning rate 0.02 for training. The learning rate is then scaled by 0.5 when the following condition satisfied:

$$\frac{|e_p - e_c|}{e_p} <= 0.005 \text{ and } lr >= 0.0005$$

where e_p is the error rate on the validation set on the previous epoch. e_c is the error rate on the current epoch. An intuitive explanation of the rule is when the growth of the performance become lower, we need to use a smaller learning rate to adjust the weights. We use on-line learning in our experiments.

We use dropout to avoid overfitting. We add a binary dropout mask to the local context windows 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 $1 - p$.

Comparison with Other Systems We compare our methods with other systems. The comparison does not include any externally labeled data or POS tags. We present experiments trained on the training set and evaluated on the test set using the highest 1-best supertagging accuracy on the development set.

Table 1 shows deep stacked models perform better than other non-stacked models. Specically, our shortcut block achieves state-of-the-art results (95.12% on test set). Notice that 9 is the number of stacked Bi-LSTM layers. The total layer of the networks contains 11 (9 + 1 input-to-hidden layer + 1 hidden-to-output layer) layers. We find the 9 or 11 stacked depth are the proper choices for the task.

Table 1: 1-best supertagging accuracy on CCGbank. The "mixed block" indicates adding shortcut connections to both c_t^l and h_t^l, as shown in Eq. (7).

Model	**Dev**	**Test**
Clark and Curran [2]	91.5	92.0
MLP (Lewis et al. [15])	91.3	91.6
Bi-LSTM (Lewis et al. [14])	94.1	94.3
Elman-RNN (Xu et al. [28])	93.1	93.0
Bi-RNN (Xu et al. [29])	93.49	93.52
Bi-LSTM (Vaswani et al. [24])	94.24	94.5
9-stacked Bi-LSTM (Wu et al. [27])	94.55	94.69
9-stacked: mixed block (Ours)	94.72	95.08
9-stacked: shortcut block (Ours)	**94.93**	**95.12**

Exploration of Shortcuts To get a better understanding of the shortcuts proposed in Eq. (10), we experiment with its variants to compare the performance. Our analysis mainly focuses on three parts: the variants of shortcut blocks, gating functions. The comparisons are summarized as follows:

Shortcut Variants. Table 2 presents the shortcut variants. In the variants of mixed block (Eq.7), the shortcuts for both c_t^l and h_t^l obtains the highest accuracy (95.08%) on the test set, but with high computational cost. In the variants of shortcut block, the "no gate in m" case performs better than the others.

Table 2: Comparsion of shortcut variants. We use $\tilde{h}_t^l$ to represent the original cell output of LSTM block, which equals $o \odot \tanh(c_t^l)$, similar to $\tilde{c}_t^l := i \odot s_t^l + f \odot c_{t-1}$.

Case	**Variant**	**Dev**	**Test**
h_t^l updated [27]	with gate: $h_t^l = \tilde{h}_t^l + g \odot h_t^{l-2}$	94.51	94.67
mixed block (Eq.7)	no gate: $c_t^l = \tilde{c}_t^l + h_t^{l-2}, h_t^l = \tilde{h}_t^l + h_t^{l-2}$	93.84	93.84
	highway gate: $c_t^l = (1-g) \odot \tilde{c}_t^l + g \odot h_t^{l-2}$, $h_t^l = (1-g) \odot \tilde{h}_t^l + g \odot h_t^{l-2}$	94.49	94.62
	shortcuts for both c_t^l and h_t^l: $c_t^l = \tilde{c}_t^l + g_c \odot c_t^{l-2}$, $h_t^l = \tilde{h}_t^l + g_h \odot h_t^{l-2}$	94.72	94.98
shortcut block (Eq.10)	no gate in h_t^l: $h_t^l = o \odot \tanh(m) + h_t^{l-2}$	94.15	94.29
	no gate in m: $m = i \odot s_t^l + h_t^{l-2}$	94.77	94.97
	no shortcut in internal: $h_t^l = o \odot \tanh(i_t \odot s_t) + g \odot h_t^{l-2}$	93.83	94.01
	no shortcut in cell output: $h_t^l = o \odot \tanh(m)$	93.58	93.82

Gating Functions. Table 3 presents the comparison of several gating functions proposed in Section 3.2. We use the tanh function in the previous outputs to break the identity link. The result is 94.81% (Table 3), which is poorer 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.

We find the deterministic gates performs better than the stochastic gates. Further, non-linear mapping $g_t^l = \sigma(W^l h_t^{l-1})$ achieves the best test accuracy (Table 3, 94.79%), while other types such as linear or stochastic gates are not generalize well.

Table 3: Comparsion of gating functions.

Case	**Variant**	**Dev**	**Test**
scaled mapping	replace h_t^{l-2} with $\tanh(h_t^{l-2})$	94.60	94.81
linear mapping	$g_t^l \odot h_t^{-l} = w^l \odot h_t^{-l}$	92.07	92.15
non-linear mapping	$g_l^l = \sigma(W^l h_t^{l-1})$	**94.79**	**94.91**
	$g_l^l = \sigma(U^l h_{t-1}^l)$	94.21	94.56
	$g_l^l = \sigma(V^l h_t^{l-2})$	94.60	94.78
stochastic sampling	$g_t^l \sim \text{Bernoulli}(p)$, $p = 0.5$	91.12	91.47
	$g_t^l \sim \text{Bernoulli}(p)$, $p = \sigma(H^l h_t^{l-1})$	93.90	94.06

Comparison of Hyper-parameters As described in Section 4.1, we use a complex input encoding for our model. Concretely, we use a context window approach, together with character-level information to get a better representa-

tion for the raw input. We give comparisons for the system with/without this approaches while keeping the hidden and the output parts unchanged.

Table 4 shows the effects of the hyper-parameters on the task. We find that the model does not perform well (94.06%) without using local context windows. Although LSTMs can memorize recent inputs for a long time, it is still necessary to use a convolution-like operator to convolve the input tokens to get a better representation. Character-level information also plays an important role for this task (13% relatively improvement), but the performance would be heavily damaged if using characters only.

Table 4: Comparsion of hyper-parameters.

Case	**Variant**	**Dev**	**Test**
window size	$k = 0$	93.96	94.06
	$k = 5$	94.27	94.81
	$k = 7$	94.52	94.71
character-level	$l_w = 0$	93.59	93.71
	$l_w = 3$	94.21	94.41
	$l_w = 7$	94.43	94.75
character only	-	92.17	93.0

6 Related Work

Skip connections have been widely used for training deep neural networks. For recurrent neural networks, Schmidhuber [19]; El Hihi and Bengio [3] introduce deep RNNs by stacking hidden layers on top of each other. Graves [4]; Hermans and Schrauwen [7] propose the use of skip connections in stacked RNNs. However, the researchers have paid less attention to the analysis of various kinds of skip connections, which is our focus in this paper.

Recently, deep stacked networks have been widely used for applications. Srivastava et al. [21] and He et al. [6] 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. [11] extends the one dimensional LSTMs to many dimensional LSTMs, which provides a more general framework to construct deep LSTMs.

Zhang et al. [30] proposed highway LSTMs by introducing gated direct connections between internal states in adjacent layers. Zilly et al. [31] introduced recurrent highway networks (RHNs) which use a single recurrent layer to make RNN deep in a vertical direction. These works do not use skip connections, and the hierarchical structure is reflected in the LSTM internal states or cell outputs. Wu et al. [26, 27] proposed a similar architecture for the shortcuts in stacked Bi-LSTMs, we make a improvement to their design.

There are also some works using stochastic gates to transmit the information. Zoneout [12] provides a stochastic link between the previous hidden states and the current states, forcing the current states to maintain their previous values during the recurrent step. Chung et al. [1] proposes a stochastic boundary state to update the internal states and cell outputs. These stochastic connections are connected between adjacent layers, while our constructions of the shortcuts are mostly cross-layered. Also, the updating mechanisms of LSTM blocks are different.

7 Conclusions

In this paper, we propose the shortcut block as a basic architecture for constructing deep stacked models. We compare several gating functions and find that the non-linear deterministic gate performs the best. These explorations can help us to train deep stacked Bi-LSTMs successfully. Based on this shortcuts structure, we achieve the state-of-the-art results on CCG supertagging. Our explorations could easily be applied to other sequence processing problems, which can be modeled with RNN architectures.

References

1. Chung, J., Ahn, S., Bengio, Y.: Hierarchical multiscale recurrent neural networks. arXiv preprint arXiv:1609.01704 (2016)
2. Clark, S., Curran, J.R.: Wide-coverage efficient statistical parsing with ccg and log-linear models. Computational Linguistics 33(4), 493–552 (2007)
3. El Hihi, S., Bengio, Y.: Hierarchical recurrent neural networks for long-term dependencies. In: NIPS. vol. 400, p. 409. Citeseer (1995)
4. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 (2013)
5. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Networks 18(5), 602–610 (2005)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
7. Hermans, M., Schrauwen, B.: Training and analysing deep recurrent neural networks. In: Advances in Neural Information Processing Systems. pp. 190–198 (2013)
8. Hochreiter, S., Schmidhuber, J.: Lstm can solve hard long time lag problems. Advances in neural information processing systems pp. 473–479 (1997)
9. Hockenmaier, J., Steedman, M.: Ccgbank: a corpus of CCG derivations and dependency structures extracted from the penn treebank. Computational Linguistics 33(3), 355–396 (2007)
10. Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K.: Deep networks with stochastic depth. arXiv preprint arXiv:1603.09382 (2016)
11. Kalchbrenner, N., Danihelka, I., Graves, A.: Grid long short-term memory. arXiv preprint arXiv:1507.01526 (2015)
12. Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N.R., Goyal, A., Bengio, Y., Larochelle, H., Courville, A., et al.: Zoneout: Regularizing rnns by randomly preserving hidden activations. arXiv preprint arXiv:1606.01305 (2016)

13. Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., Dyer, C.: Neural architectures for named entity recognition. arXiv preprint arXiv:1603.01360 (2016)
14. Lewis, M., Lee, K., Zettlemoyer, L.: Lstm ccg parsing. In: Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics (2016)
15. Lewis, M., Steedman, M.: Improved CCG parsing with semi-supervised supertagging. Transactions of the Association for Computational Linguistics 2, 327–338 (2014)
16. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of english: The penn treebank. Computational linguistics 19(2), 313–330 (1993)
17. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: EMNLP. vol. 14, pp. 1532–43 (2014)
18. Saxe, A.M., McClelland, J.L., Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120 (2013)
19. Schmidhuber, J.: Learning complex, extended sequences using the principle of history compression. Neural Computation 4(2), 234–242 (1992)
20. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research 15(1), 1929–1958 (2014)
21. Srivastava, R.K., Greff, K., Schmidhuber, J.: Highway networks. arXiv preprint arXiv:1505.00387 (2015)
22. Steedman, M.: The syntactic process, vol. 24. MIT Press (2000)
23. Steedman, M., Baldridge, J.: Combinatory categorial grammar. Non-Transformational Syntax: Formal and Explicit Models of Grammar. Wiley-Blackwell (2011)
24. Vaswani, A., Bisk, Y., Sagae, K., Musa, R.: Supertagging with lstms. In: Proceedings of the Human Language Technology Conference of the NAACL (2016)
25. Wang, P., Qian, Y., Soong, F.K., He, L., Zhao, H.: Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. arXiv preprint arXiv:1510.06168 (2015)
26. Wu, H., Zhang, J., Zong, C.: A dynamic window neural network for ccg supertagging. national conference on artificial intelligence pp. 3337–3343 (2016)
27. Wu, H., Zhang, J., Zong, C.: An empirical exploration of skip connections for sequential tagging. international conference on computational linguistics pp. 203–212 (2016)
28. Xu, W., Auli, M., Clark, S.: CCG supertagging with a recurrent neural network. Volume 2: Short Papers p. 250 (2015)
29. Xu, W., Auli, M., Clark, S.: Expected f-measure training for shift-reduce parsing with recurrent neural networks. In: Proceedings of NAACL-HLT. pp. 210–220 (2016)
30. Zhang, Y., Chen, G., Yu, D., Yaco, K., Khudanpur, S., Glass, J.: Highway long short-term memory rnns for distant speech recognition. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5755–5759. IEEE (2016)
31. Zilly, J.G., Srivastava, R.K., Koutník, J., Schmidhuber, J.: Recurrent highway networks. arXiv preprint arXiv:1607.03474 (2016)