DEPENDENT BIDIRECTIONAL RNN WITH SUPER-LONG SHORT-TERM MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work, we first conduct mathematical analysis on the memory decay of three RNN cells; namely, the simple recurrent neural network (SRN), the long short-term memory (LSTM) and the gated recurrent unit (GRU). Based on the analysis, we propose a new design, called the super-long short-term memory (SLSTM), to extend the memory length of a cell. Next, we present an efficient RNN architecture, called the dependent bidirectional recurrent neural network (DBRNN), for the sequence-in-sequence-out (SISO) problem. Finally, the superior performance of the DBRNN architecture with the SLSTM cell is demonstrated by experimental results.

1 INTRODUCTION

The recurrent neural network (RNN) has proved to be an effective solution for natural language processing (NLP) through the advancement in the last three decades (Elman, 1990; Jordan, 1997). At the cell level of a RNN, the long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) and the gated recurrent unit (GRU) (Cho et al., 2014) are often adopted by a RNN as its low-level building cell. Being built upon these cells, various RNN architectures have been proposed to solve the sequence-in-sequence-out (SISO) problem. To name a few, there are the bidirectional RNN (BRNN) (Schuster & Paliwal, 1997), the encoder-decoder architecture (Cho et al., 2014; Sutskever et al., 2014; Vinyals et al., 2015; Bahdanau et al., 2015) and the deep RNN (Pascanu et al., 2014). Although the LSTM and the GRU were designed to enhance the memory length of RNNs and avoid the gradient vanishing/exploding issue (Hochreiter & Schmidhuber, 1997; Razvan et al., 2013; Bengio et al., 1994), a good understanding of their memory length is still lacking. The first objective of this research is to analyze the memory length of three RNN cells – the simple RNN (SRN) (Elman, 1990; Jordan, 1997), the long short-term memory (LSTM) and the gated recurrent unit (GRU). This will be conducted in Sec. 2. Based on this understanding, we propose a new design, called the super-long short-term memory (SLSTM), to extend the memory length of a cell in Sec. 3.

As to the macro RNN architecture, one popular choice is the BRNN. Since the elements in BRNN output sequences should be independent of each other (Schuster & Paliwal, 1997), the BRNN cannot be used to solve dependent output sequence problem alone. Nevertheless, most language tasks do involve dependent output sequences. The second choice is the encoder-decoder system, where the attention mechanism has been introduced (Vinyals et al., 2015; Bahdanau et al., 2015) to improve its performance furthermore. As shown later in this work, the encoder-decoder system is not an efficient learner. Here, to take advantages of both the encoder-decoder and the BRNN and overcome their drawbacks, we propose a new architecture called the dependent bidirectional recurrent neural network (DBRNN), which will be elaborated in Sec. 4. Furthermore, we conduct a series of experiments on the part of speech (POS) tagging and the dependency parsing (DP) problems in Sec. 5 to demonstrate the superior performance of the DBRNN architecture with the SLSTM cell. Finally, concluding remarks are given and future research direction is pointed out in Sec. 6.

2 MEMORY ANALYSIS OF SRN, LSTM AND GRU

For a large number of NLP tasks, we are concerned with finding semantic patterns from the input sequence. It was shown by Elman (1990) that the RNN builds an internal representation of semantic patterns. The memory of a cell characterizes its ability to map an input sequence of certain length

into such a representation. It was reported by Gers et al. (2000) that a SRN only memorized sequences of length between 3-5 units while a LSTM could memorize sequences of length longer than 1000 units. In this section, we study the memory of the SRN, LSTM and GRU.

2.1 MEMORY OF SRN

The SRN is described by the following two equations:

$$h_t = W_h h_{t-1} + W_{in} X_t, \tag{1}$$

$$Y_t = f(h_t), \tag{2}$$

where subscript t is the index of the time unit, $W_h \in \mathbb{R}^{N \times N}$ is the weight matrix for hidden state vector $h_{t-1} \in \mathbb{R}^N$, $W_{in} \in \mathbb{R}^{N \times M}$ is the weight matrix of input vector $X_t \in \mathbb{R}^M$, $Y_t \in \mathbb{R}^N$ and $f(\cdot)$ is an element-wise non-linear function. Usually, $f(\cdot)$ is a hyperbolic-tangent or a sigmoid function. Note that we can account for the bias terms by augmenting vectors h_t and X_t with one more dimension and adjusting weight matrices W_h and W_{in} accordingly in a straightforward manner. Thus, although the system of equations given in Eqs. (1) and (2) is simple, it is generic.

By induction, h_t can be rewritten as

$$h_t = W_h^t h_0 + \sum_{k=1}^{t} W_h^{t-k} W_{in} X_k, \tag{3}$$

where h_0 is the initial internal state of the SRN. Typically, we assume an initial rest state by setting it to be a column vector of zeros; i.e., $h_0 = \underline{0}$. Then, Eq. (3) becomes

$$h_t = \sum_{k=1}^{t} W_h^{t-k} W_{in} X_k. \tag{4}$$

Let $\lambda_{\max}$ be the largest singular value of W_h. Then, we have

$$||W_h^{t-k} X_k|| \leq ||W_h||^{|t-k|} ||X_k|| = \lambda_{\max}^{|t-k|} ||X_k||. \tag{5}$$

Hence, the contribution of X_k, $k < t$, to h_t decays in form of $\lambda_{\max}^{|t-k|}$. We conclude that SRN's memory decays exponentially with its memory length $|t-k|$. Clearly, the memory will explode if $|\lambda_{\max}| > 1$. This means that W_h cannot be chosen arbitrarily.

2.2 MEMORY OF LSTM

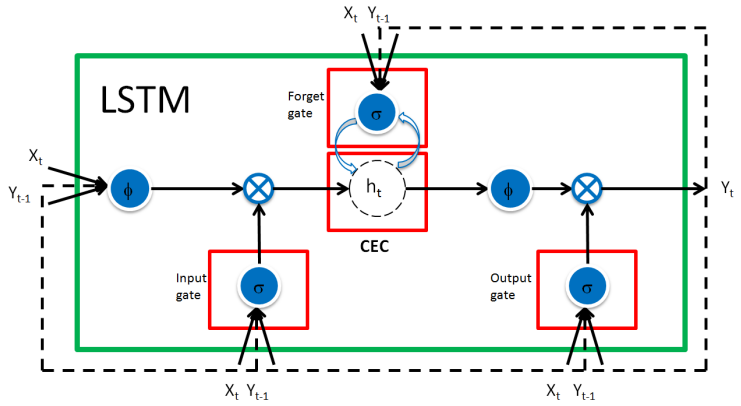


Figure 1: The diagram of a LSTM cell.

By following the work of Hochreiter & Schmidhuber (1997), we plot the diagram of a LSTM cell in Fig. 1. In this figure, ϕ, σ and $\otimes$ denote the hyperbolic tangent function, the sigmoid function and the multiplication operation, respectively. All of them operate in an element-wise fashion. The

LSTM has an input gate, an output gate, a forget gate and a constant error carousal (CEC) module. Mathematically, the LSTM cell can be written as

$$h_t = \sigma(W_f I_t) h_{t-1} + \sigma(W_i I_t)\phi(W_{in} I_t), \tag{6}$$

$$Y_t = \sigma(W_o I_t)\phi(h_t), \tag{7}$$

where $h_t \in \mathbb{R}^N$, column vector $I_t \in \mathbb{R}^{(M+N)}$ is a concatenation of the current input, $X_t \in \mathbb{R}^M$, and the previous output, $Y_{t-1} \in \mathbb{R}^N$ (*i.e.,* $I_t^T = [X_t^T, Y_{t-1}^T]$). Furthermore, W_f, W_i, W_o and W_{in} are weight matrices for the forget gate, the input gate, the output gate and the input, respectively. Their dimensions are defined by their corresponding inputs and outputs.

Under the assumption $h_0 = \underline{0}$, the hidden state vector of the LSTM can be derived by induction as

$$h_t = \sum_{k=1}^{t} \underbrace{\left[\prod_{j=k+1}^{t} \sigma(W_f I_j)\right]}_{\text{forget gate}} \sigma(W_i I_k)\phi(W_{in} I_k), \tag{8}$$

where $\prod$ denotes the element-wise multiplication. By setting $f(\cdot)$ in Eq. (2) to a hyperbolic-tangent function, we can compare outputs of the SRN and the LSTM below:

$$\text{SRN:} \qquad Y_t^{SRN} = \phi\left(\sum_{k=1}^{t} W_h^{t-k} W_{in} X_k\right), \tag{9}$$

$$\text{LSTM:} \qquad Y_t^{LSTM} = \sigma(W_o I_t)\phi\left(\sum_{k=1}^{t} \underbrace{\left[\prod_{j=k+1}^{t} \sigma(W_f I_j)\right]}_{\text{forget gate}} \sigma(W_i I_k)\phi(W_{in} I_k)\right). \tag{10}$$

We see from the above that W_h^{t-k} and $\prod_{j=k+1}^{t} \sigma(W_f I_j)$ play the same memory role for the SRN and the LSTM, respectively.

If W_f in Eq. (10) is selected such that

$$\min |\sigma(W_f I_j)| \geq \lambda_{\max}, \quad \forall \lambda_{\max} \in [0, 1),$$

then

$$\left|\prod_{j=k+1}^{t} \sigma(W_f I_j)\right| \geq \lambda_{\max}^{|t-k|}. \tag{11}$$

As given in Eqs. (5) and (11), the impact of input I_k on output Y_t in the LSTM lasts longer than that of input X_k in the SRN. This is the case if an appropriate weight matrix, W_f, of the forget gate is selected.

2.3 Memory of GRU

The GRU was originally proposed for neural machine translation (Cho et al., 2014). It provides an effective alternative for the LSTM. Its operations can be expressed by the following four equations:

$$z_t = \sigma(W_z X_t + U_z h_{t-1}), \tag{12}$$

$$r_t = \sigma(W_r X_t + U_r h_{t-1}), \tag{13}$$

$$\tilde{h}_t = \phi(W X_t + U(r_t \otimes h_{t-1})), \tag{14}$$

$$h_t = z_t h_{t-1} + (1 - z_t)\tilde{h}_t, \tag{15}$$

where X_t, h_t, z_t and r_t denote the input vector, the hidden state vector, the update gate vector and the reset gate vector, respectively, and W_z, W_r, W, are trainable weight matrices. Its hidden state is also its output, which is given in Eq. (15). If we simplify the GRU by setting U_z, U_r and U to zero matrices, then we can obtain the following simplified GRU system:

$$z_t = \sigma(W_z X_t), \tag{16}$$

$$\tilde{h}_t = \phi(W X_t), \tag{17}$$

$$h_t = z_t h_{t-1} + (1 - z_t)\tilde{h}_t. \tag{18}$$

For the simplified GRU with the initial rest condition, we can derive the following by induction:

$$h_t = \sum_{k=1}^{t} \Big[\underbrace{\prod_{j=k+1}^{t} \sigma(W_z X_j)}_{\text{update gate}} \Big] (1 - \sigma(W_z X_k)) \phi(W X_k). \tag{19}$$

By comparing Eqs. (8) and (19), we see that the update gate of the simplified GRU and the forget gate of the LSTM play the same role. One can control the memory decay behavior of the GRU by choosing the weight matrix, W_z, of the update gate carefully.

3 Super-Long Short-Term Memory (SLSTM)

As discussed above, the LSTM and the GRU have longer memory by introducing the forget and the update gates, respectively. However, their memory still fades quickly since their memory decay rate is proportional to the product of a sequence of small numbers. In this section, we attempt to design super-long short-term memory (SLSTM) cells and propose two new cell models:

- SLSTM-I: the super-long short-term memory (SLSTM) with input weight vector $c_i \in \mathbb{R}^N$, $i = 1, \cdots, t-1$, where weights c_i and c_j (with $i \neq j$) are independent.
- SLSTM-II: the SLSTM-I with no forget gate.

These two cells are depicted in Figs. 2 (a) and (b), respectively.

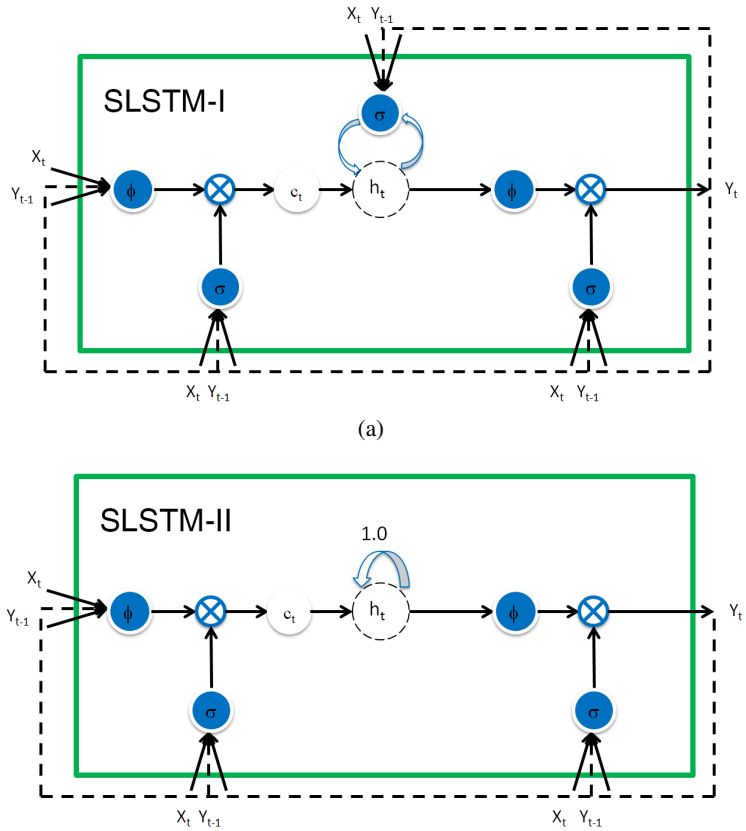


Figure 2: The diagrams of (a) the SLSTM-I cell and (b) the SLSTM-II cell.

The SLSTM-I cell can be described by

$$h_t = \sigma(W_f I_t)h_{t-1} + c_t\sigma(W_i I_t)\phi(W_{in} I_t), \tag{20}$$

$$Y_t = \sigma(W_o I_t)\phi(h_t + b). \tag{21}$$

With $h_0 = \underline{0}$, we can get

$$h_t = \sum_{k=1}^{t} c_k \Big[\prod_{j=k+1}^{t} \sigma(W_f I_j) \Big] \sigma(W_i I_k)\phi(W_{in} I_k), \tag{22}$$

$$Y_t = \sigma(W_o I_t)\phi\Big(h_t + b \Big), \tag{23}$$

where $b \in \mathbb{R}^N$ is a trainable bias vector. The SLSTM-II cell can be written as

$$h_t = h_{t-1} + c_t\sigma(W_i I_t)\phi(W_{in} I_t), \tag{24}$$

$$Y_t = \sigma(W_o I_t)\phi(h_t + b). \tag{25}$$

It can be further simplified as

$$h_t = \sum_{k=1}^{t} c_k\sigma(W_i I_k)\phi(W_{in} I_k), \tag{26}$$

$$Y_t = \sigma(W_o I_t)\phi\Big(h_t + b \Big). \tag{27}$$

As shown above, we introduce weight vector, c_i, $i = 1, \cdots, t-1$, to the SLSTM-I and the SLSTM-II to increase or decrease the impact of input I_i in the sequence. The major difference between the SLSTM-I and the SLSTM-II is that fewer parameters are used in the SLSTM-II than those in the SLSTM-I. The numbers of parameters used by different RNN cells are compared in Table 1, where $X_t \in \mathbb{R}^M$, $Y_t \in \mathbb{R}^N$ and $t = 1, \cdots, S$.

Table 1: Comparison of Parameter Numbers.

Cell	**Number of Parameters**
LSTM	$4N(M + N + 1)$
GRU	$3N(M + N + 1)$
SLSTM-I	$4N(M + N + 1) + N(S + 1)$
SLSTM-II	$3N(M + N + 1) + N(S + 1)$

4 Proposed Dependent BRNN (DBRNN) Architecture

We investigate the macro RNN architecture and propose the dependent BRNN (DBRNN) in this section. Our proposal is inspired by the pros and cons of two RNN architectures – the bidirectional RNN (BRNN) architecture (Schuster & Paliwal, 1997) and the encoder-decoder architecture (Cho et al., 2014). In the following, we will first examine the BRNN and the encoder-decoder in Sec. 4.1 and, then, propose the DBRNN in Sec. 4.2.

4.1 BRNN and Encoder-Decoder

Most NLP problems can be formulated as an estimation problem. The probabilistic model for the BRNN can be written as

$$\hat{Y_t^f} = \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\{X_i\}_{i=1}^{t}), \tag{28}$$

$$\hat{Y_t^b} = \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\{X_i\}_{i=t}^{S}), \tag{29}$$

$$\hat{Y_t} = \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\hat{Y_t^f}, \hat{Y_t^b}), \tag{30}$$

where P is the probability density function, S is the length of the input sequence. f and b denote forward and backward respectively. So $\hat{Y_t^f}$ and $\hat{Y_t^b}$ are the forward and the backward predictions of Y_t, respectively. As shown in Eq. (30), the final prediction of the BRNN is the pooling of different expert opinions - one is from the forward prediction while the other is from the backward prediction. Due to this bidirectional design, the BRNN can fully utilize the information of the entire input sequence to predict each individual output element. On the other hand, the BRNN does not utilize the predicted output in predicting Y_t. This makes elements in the predicted sequence $\{\hat{Y}_t\}_{t=1}^{S}$ independent of each other.

This problem can be handled using the encoder-decoder model in form of

$$\hat{Y}_t = \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\{\hat{Y}_i\}_{i=1}^{t-1}, \{X_i\}_{i=1}^{S}). \tag{31}$$

However, unlike the BRNN, the encoder-decoder architecture does not pool expert opinions, making it vulnerable to previous erroneous predictions in the forward path. Recently, the BRNN has been introduced in the encoder by Bahdanau et al. (2015), yet this design still does not address the erroneous prediction problem.

4.2 DBRNN Architecture and Training

Being motivated by observations in Sec. 4.1, we propose the DBRNN architecture to fulfill the following objectives:

$$\hat{Y_t^f} = \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\{\hat{Y_i^f}\}_{i=1}^{t-1}, \{X_i\}_{i=1}^{S}), \tag{32}$$

$$\hat{Y_t^b} = \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\{\hat{Y_i^b}\}_{i=t+1}^{S}, \{X_i\}_{i=1}^{S}), \tag{33}$$

$$\hat{Y}_t = \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\hat{Y_t^f}, \hat{Y_t^b}) \approx \underset{Y_t}{\operatorname{argmax}}\, P(Y_t|\{\hat{Y}_i\}_{i=1}^{S}, \{X_i\}_{i=1}^{S}). \tag{34}$$

The DBRNN architecture is shown in Fig. 3. It consists of a lower and an upper BRNN branches. The lower one pools the expert opinions from the input while the upper one pools the forward and backward predictions from the output. At each time step, the input to the forward and the backward parts of the upper BRNN is the concatenated forward and backward outputs from the lower BRNN branch.

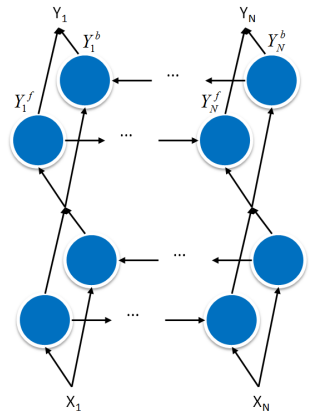


Figure 3: The DBRNN architecture.

The DBRNN architecture can be described mathematically as follows. We use $X = \{x_1, ..., x_S\}$, $x_i \in \mathbb{R}^M$, and $Y = \{y_1, ..., y_S\}$, $y_i \in \mathbb{R}^N$, to denote the input and output sequences, respectively, where N, S and M are the same as those in Table 1. Let $C(\cdot)$ be the cell function. The input, X, is fed into the forward and backward RNN of the lower BRNN branch as

$$Z_t^f = C_l^f(x_t, h_{l(t-1)}^f), \quad Z_t^b = C_l^b(x_t, h_{l(t+1)}^b), \quad Z_t = \begin{bmatrix} Z_t^f \\ Z_t^b \end{bmatrix}, \tag{35}$$

where h denotes the cell hidden state and l denotes the lower BRNN. The final output, Z_t, of the lower BRNN is the concatenation of the output, Z_t^f, of the forward RNN and the output, Z_t^b, of the backward RNN. Similarly, the upper BRNN generates the final output Y_t as

$$Y_t^f = C_u^f(Z_t, h_{u(t-1)}^f), \quad Y_t^b = C_u^b(Z_t, h_{u(t+1)}^b), \quad Y_t = W^f Y_t^f + W^b Y_t^b, \tag{36}$$

where u denotes the upper BRNN, Y_t is a linear combination of Y_t^f and Y_t^b, and W^f and W^b are trainable weight matrices. To generate forward prediction $\hat{Y_t^f}$ and backward prediction $\hat{Y_t^b}$, the forward and backward paths of the upper BRNN branches are trained separately with the target sequence and the reversed target sequence, respectively. The results of the upper forward and backward RNN are then combined to generate the final result.

There are three errors: prediction error of $\hat{Y_t^f}$ denoted by e_f, prediction error of $\hat{Y_t^b}$ denoted by e_b and prediction error of $\hat{Y_t}$ denoted by e. To train this network, e_f is back propagated through time to the upper forward RNN and the lower BRNN, e_b is back propagated through time to the upper backward RNN and the lower BRNN, and e is back propagated through time to the entire architecture.

It is worthwhile to compare the DBRNN and the solution in Cheng et al. (2016). Both of them have a bidirectional design for the output. However, there exist three main differences. First, the DBRNN is a general design for the sequence-in-sequence-out (SISO) problem without being restricted to dependency parsing. The target sequences in training $\hat{Y_t^f}$, $\hat{Y_t^b}$ and $\hat{Y_t}$ are the same for the DBRNN. In contrast, the solution in Cheng et al. (2016) has different target sequences. Second, the attention mechanism is used by Cheng et al. (2016) but not in the DBRNN. Third, The encoder-decoder design is adopted by in Cheng et al. (2016) but not in the DBRNN.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We conduct experiments on two problems: part of speech (POS) tagging and dependency parsing (DP). The POS tagging task is an easy one which requires shorter memory while the DP task needs much longer memory and has more complex relations between the input and the output.

In the experiments, we compare the performance of four RNN architectures under two scenarios: 1) $I_t = X_t$, and 2) $I_t^T = [X_t^T, Y_{t-1}^T]$. The four RNN architectures are the basic one-directional RNN (basic RNN), the BRNN, the sequence-to-sequence (a variation of encoder-decoder) RNN, and the DBRNN with four cell designs (LSTM, GRU, SLSTM-I and SLSTM-II). When $I_t = X_t$, we do not include the GRU cell since it inherently demands $I_t^T = [X_t^T, Y_{t-1}^T]$. For the DBRNN, we show the results for Y_t (denoted by "DBRNN combined") and Y_t^f (denoted by "DBRNN forward"), which is the prediction from the forward path of the upper BRNN branch. We do not include the result for Y_t^b, which is the prediction from the backward path of the upper BRNN branch since the performance of the backward RNN path of the upper BRNN branch is poorer.

The training dataset used for both problems are from the Universal Dependency 2.0 English branch (UD-English). It contains 12543 sentences and 14985 unique tokens. The test dataset for both experiments is from the test English branch (gold, en.conllu) of CoNLL 2017 shared task development and test data.

In the experiment, the lengths of the input and the target sequences are fixed. Sequences longer than the maximum length will be truncated. If the sequence is shorter than the fixed length, a special pad symbol will be used to pad the sequence. The input to the POS tagging and the DP problems are the stemmed and lemmatized sequences (column 3 in CoNLL-U format). The target sequence for POS tagging is the universal POS tag (column 4). The target sequence for DP is the interleaved dependency relation to the headword (relation, column 8) and its position (column 7). As a result, the length of the actual target sequence (rather than the preprocessed fixed-length sequence) for DP is twice of the length of the actual input sequence.

The input is first fed into a trainable embedding layer (Bengio et al., 2003) before it is sent to the actual network. Table 2 shows the detailed network and training specifications. It is important

Table 2: Network and training details

Input/output sequence fixed length	100/100
Number of RNN layers	1
Embedding layer vector size	512
Number of RNN cells	512
Batch size	20
Training steps	14000
Learning rate	0.5
Training optimizer	AdaGrad(Duchi, 2011)
Maximum gradient norm	5

to point out that we do not finetune network parameters or apply any engineering trick for the best possible performance since our main goal is to compare the performance of the LSTM, GRU, SLSTM-I and SLSTM-II four cells under various macro-architectures.

5.2 Experimental Results

The results of the POS tagging problem with $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ are shown in Tables 3 and 4, respectively. Among all possible combinations, the DBRNN with the LSTM cell has the highest accuracy (89.16%) for $I_t = X_t$ while the DBRNN with the GRU cell has the (89.74%) has the highest accuracy for $I_t^T = [X_t^T, Y_{t-1}^T]$.

Table 3: POS tagging test accuracy, $I_t = X_t$ (%)

	LSTM	SLSTM-I	SLSTM-II
BASIC RNN	**85.38**	85.30	84.35
BRNN (Schuster & Paliwal, 1997)	**88.49**	82.84	79.14
Seq2seq (Sutskever et al., 2014)	25.83	24.87	**31.43**
DBRNN Combined	**89.16**	83.69	81.08
DBRNN Forward	**88.93**	83.54	81.08

Table 4: POS tagging test accuracy, $I_t^T = [X_t^T, Y_{t-1}^T]$ (%)

	LSTM	GRU	SLSTM-I	SLSTM-II
BASIC RNN	86.98	**87.09**	85.57	85.56
BRNN	88.94	**89.26**	83.48	82.57
Seq2seq	24.73	33.79	34.09	**52.96**
DBRNN Combined	89.67	**89.74**	84.25	84.41
DBRNN Forward	89.46	**89.53**	84.05	84.44

The results of the DP problem with $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ are shown in Tables 5 and 6, respectively. The SLSTM-I and SLSTM-II cells perform better than the LSTM and the GRU cells. Among all possible combinations, the DBRNN combined with SLSTM-I has the best performance. It has an accuracy of 54.39% and 60.30% for the former and the latter, respectively. Also, the basic RNN often outperforms BRNN for the DP problem as shown in Tables 5 and 6. This can be explained by that the basic RNN can access the entire input sequence when predicting the latter half of the output sequence since the target sequence is twice as long as the input. The other reason is that the BRNN can easily overfit when predicting the headword position.

Table 5: DP test accuracy, $I_t = X_t$ (%)

	LSTM	SLSTM-I	SLSTM-II
BASIC RNN	15.14	38.36	**42.52**
BRNN	14.74	**39.24**	35.78
Seq2seq	24.37	30.04	**35.67**
DBRNN Combined	25.26	**54.39**	53.80
DBRNN Forward	25.71	**53.67**	52.61

Table 6: DP test accuracy, $I_t^T = [X_t^T, Y_{t-1}^T]$ (%)

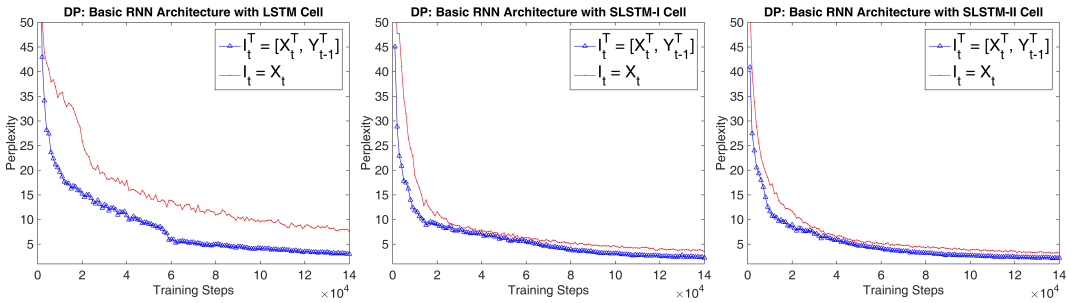
	LSTM	GRU	SLSTM-I	SLSTM-II
BASIC RNN	44.12	47.49	54.52	**56.02**
BRNN	32.46	27.83	**54.14**	47.72
Seq2seq	27.67	29.94	40.85	**48.73**
DBRNN Combined	56.89	51.32	**60.30**	58.28
DBRNN Forward	58.30	53.17	**59.81**	58.05

We see from Tables 3 - 6 that the two DBRNN architectures outperform other architectures in both POS tagging and DP problems regardless of used cells. This shows the superiority of introducing the expert opinion pooling from both the input and the predicted output.

Furthermore, the proposed SLSTM-I and SLSTM-II outperform the LSTM and the GRU by a significant margin for complex language tasks. This demonstrates that the scaling factor in the SLSTM-I and the SLSTM-II does help the network retain longer memory. For the POS tagging problem, the SLSTM-I and the SLSTM-II do not perform as well as the GRU or the LSTM. This is probably due to the shorter memory requirement of this simple task. The SLSTM cells are over-parameterized and, as a result, they converge slower and tend to overfit the training data.

The performance of the SLSTM-I and the SLSTM-II cells is close except for the sequence-to-sequence architecture, where the SLSTM-II cell performs particularly well. This has to do with the removing of the forget gate in the SLSTM-II. The hidden state h_t of the SLSTM-II is more expressive in representing patterns over a longer distance. Since the sequence-to-sequence design relies on the expressive power of the hidden state, the SLSTM-II does have an advantage.

We compare the convergence behavior of $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ with the LSTM, the SLSTM-I and the SLSTM-II cells for the DP problem in Fig. 4. We see that the SLSTM-I and the SLSTM-II do not behave very differently between $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ as the LSTM does. This shows the effectiveness of the SLSTM-I and the SLSTM-II design regardless of the input. More performance comparison will be provided in the Appendix.



(a) (b) (c)

Figure 4: Training perplexity of the basic RNN with $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ for the DP problem.

6 CONCLUSION AND FUTURE WORK

The memory decay behavior of the LSTM and the GRU was investigated and explained by mathematical analysis. Although the memory of the LSTM and the GRU fades slower than that of the SRN, it may not be long enough for complicated language tasks such as dependency parsing. To enhance the memory length, two cells called the SLSTM-I and the SLSTM-II were proposed. Furthermore, we introduced a new RNN architecture called the DBRNN that has the merits of both the BRNN and the encoder-decoder. It was shown by experimental results that the DBRNN with SLSTM-I and SLSTM-II outperforms other designs by a significant margin for complex language tasks. The DBRNN design is superior for both simple and complex language tasks. There are interesting issues to be further explored. For example, is the SLSTM cell also helpful in more sophisticated RNN architectures such as the deep RNN? Is it possible to make the DBRNN deeper and better? They are left for future study.

REFERENCES

Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *In Proceedings of The International Conference on Learning Representations*, 2015.

Yoshua Bengio, Patrice Simard, and Paolo Frasoni. Learning long-term dependencies with gradient descent is difficult. *Neural Networks*, 5:157–166, 1994.

Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, pp. 1137–1155, 2003.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. Bi-directional attention with agreement for dependency parsing. *In Proceedings of The Empirical Methods in Natural Language Processing (EMNLP 2016)*, 2016.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoderdecoder for statistical machine translation. *In Proceedings of The Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

Duchi. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, pp. 2121–2159, 2011.

Jeffrey Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

F. A. Gers, Jurgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, pp. 2451–2471, 2000.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

Michael Jordan. Serial order: A parallel distributed processing approach. *Advances in Psychology*, 121:471–495, 1997.

Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv:1312.6026*, 2014.

Pascanu Razvan, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *In Proceedings of The International Conference on Machine Learning (ICML 2013)*, pp. 1310–1318, 2013.

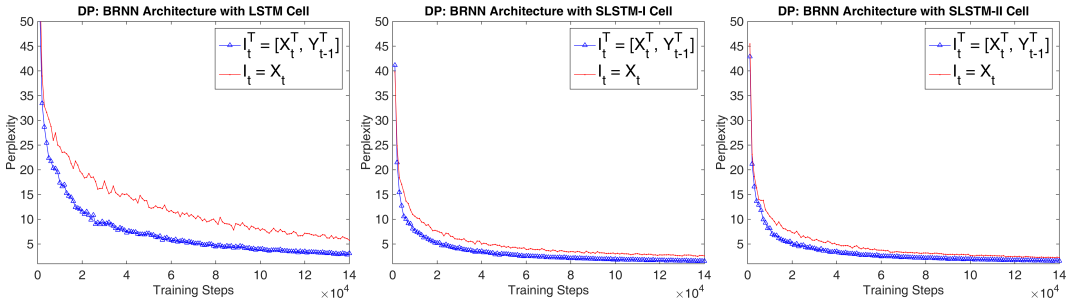
Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *Signal Processing*, 45:2673–2681, 1997.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. *Advances in Neural Information Processing Systems*, pp. 2773–2781, 2015.

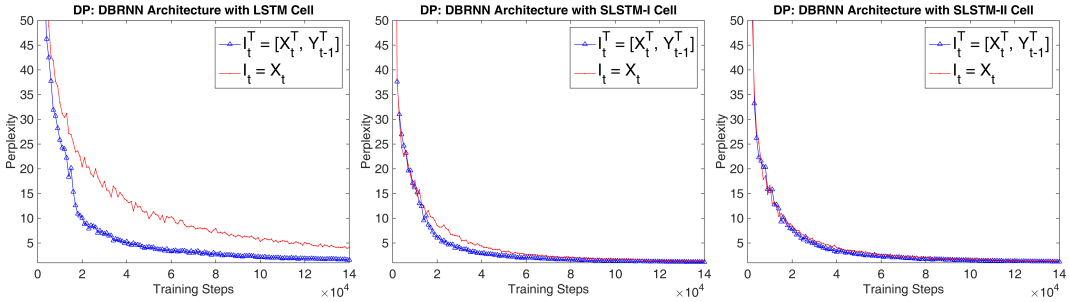
APPENDIX: MORE EXPERIMENTAL RESULTS

In the appendix, we provide more experimental results to shed light on the convergence performance in the training of various architectures with different cells for the POS tagging and the DP tasks. First, we compare the training perplexity between $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ for various architectures with the LSTM, the SLSTM-I and the SLSTM-II cells in Figs. 5 - 8. Then, we examine the training perplexity with $I_t^T = [X_t^T, Y_{t-1}^T]$ for various architectures with different cells in Figs. 9 - 12.



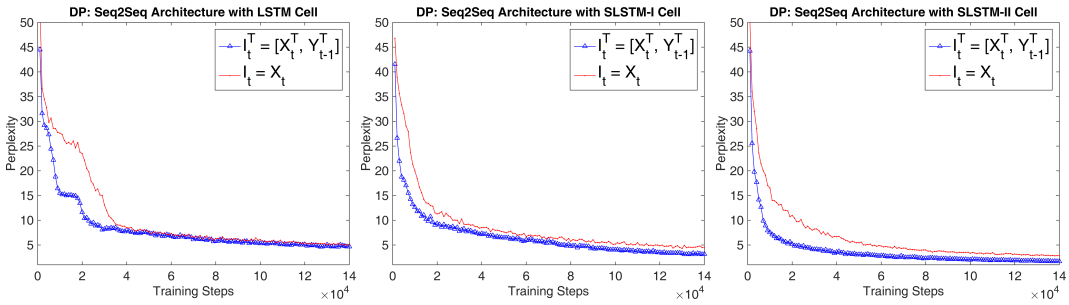
(a) (b) (c)

Figure 5: The training perplexity of the BRNN architecture with $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ for the DP task.



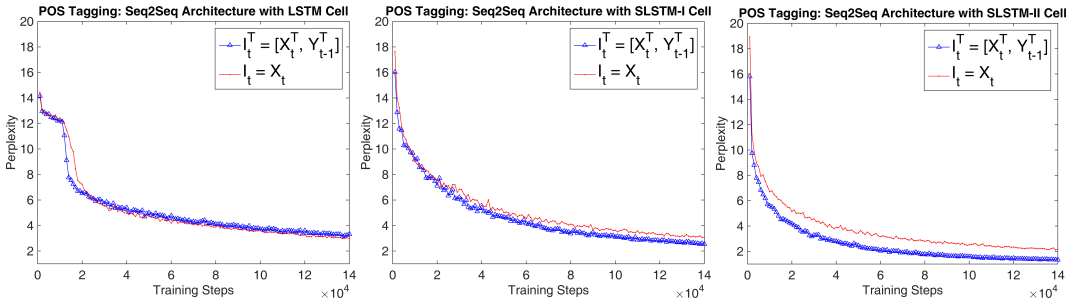
(a) (b) (c)

Figure 6: The training perplexity of the DBRNN architecture with $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ for the DP task.



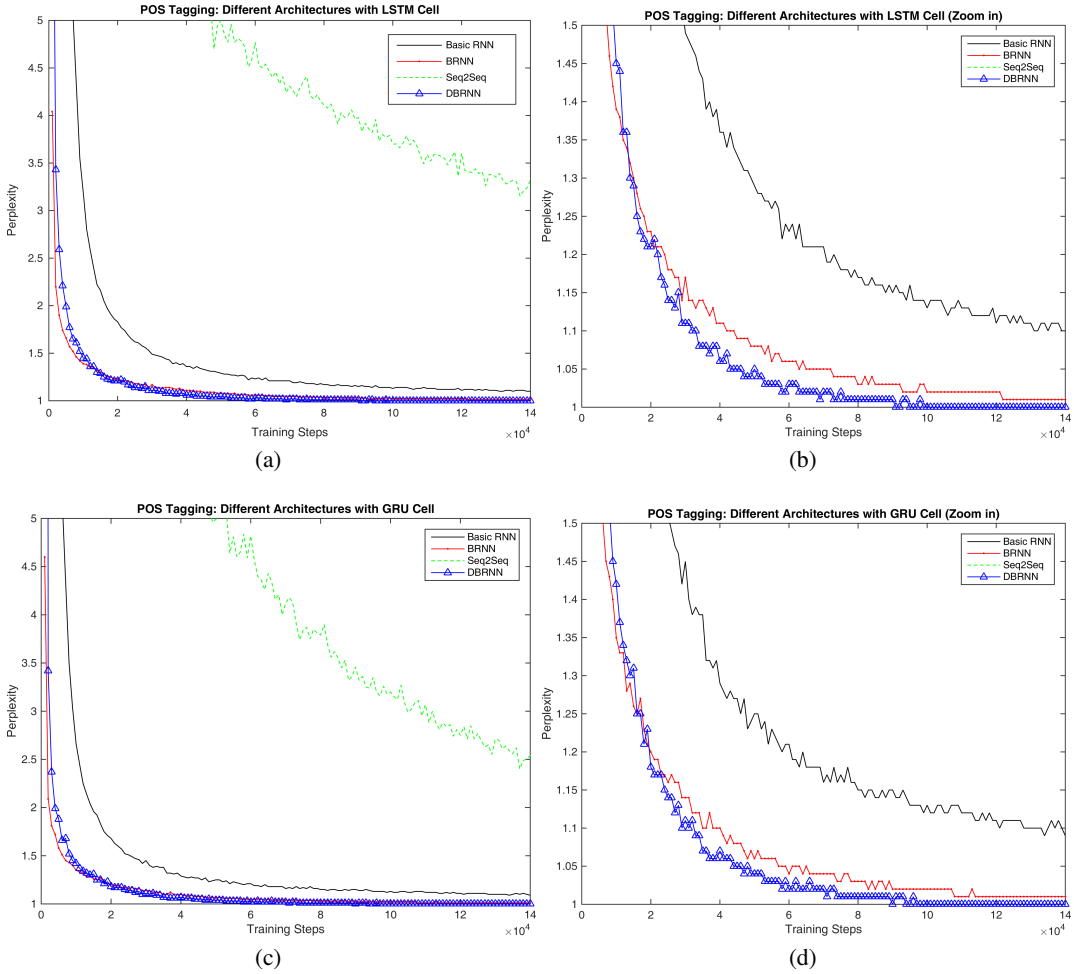
(a) (b) (c)

Figure 7: The training perplexity of the sequence-to-sequence architecture with $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ for the DP task.



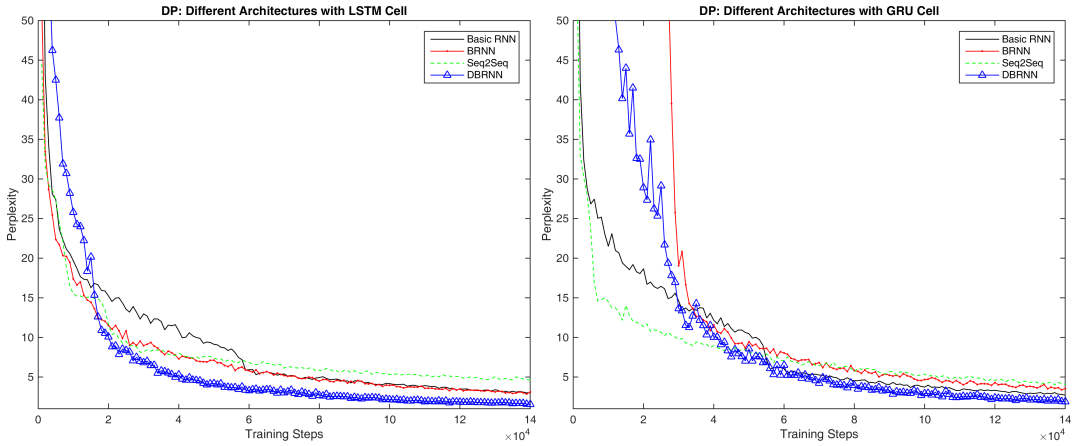
(a) (b) (c)

Figure 8: The training perplexity of the sequence-to-sequence architecture with $I_t = X_t$ and $I_t^T = [X_t^T, Y_{t-1}^T]$ for the POS tagging task.



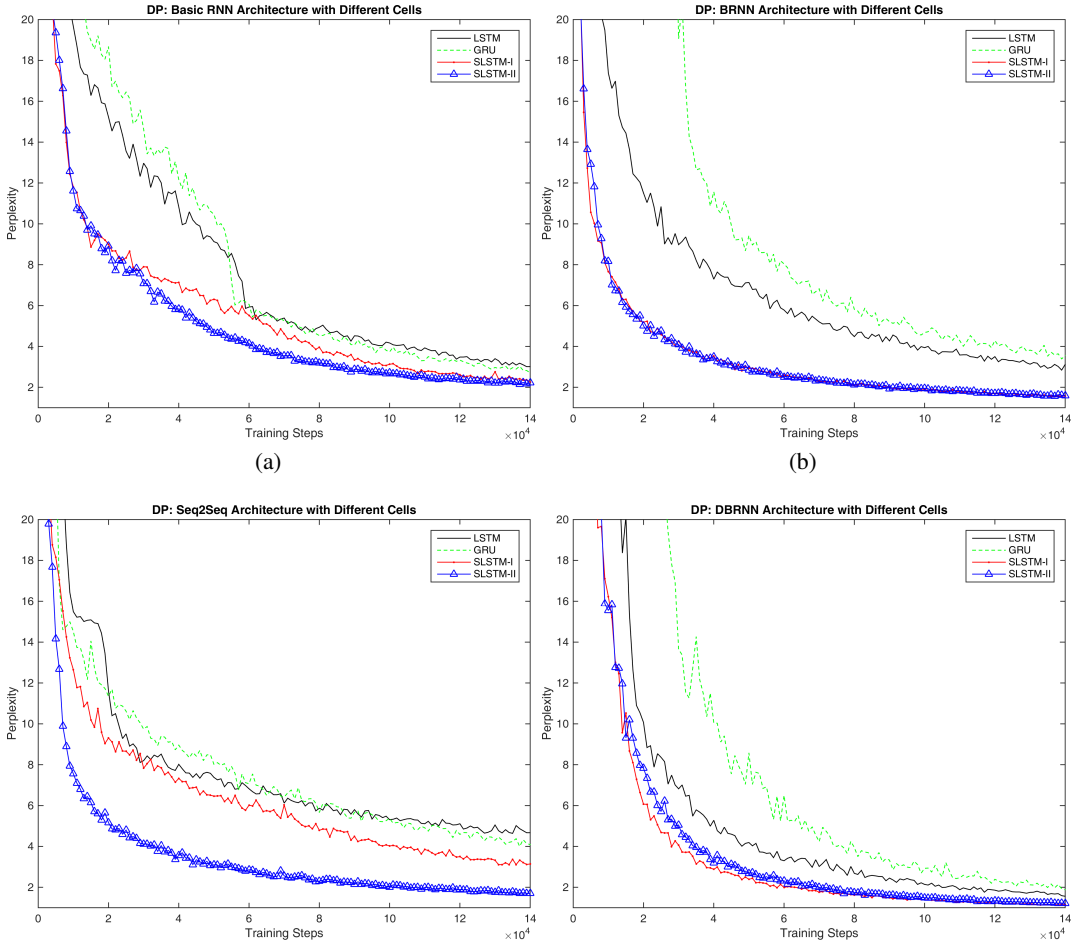
(a) (b) (c) (d)

Figure 9: The training perplexity of different architectures with the LSTM (top left), its zoom-in (top right) and the GRU (bottom left), its zoom-in (bottom right) for the POS tagging.



(a) (b)

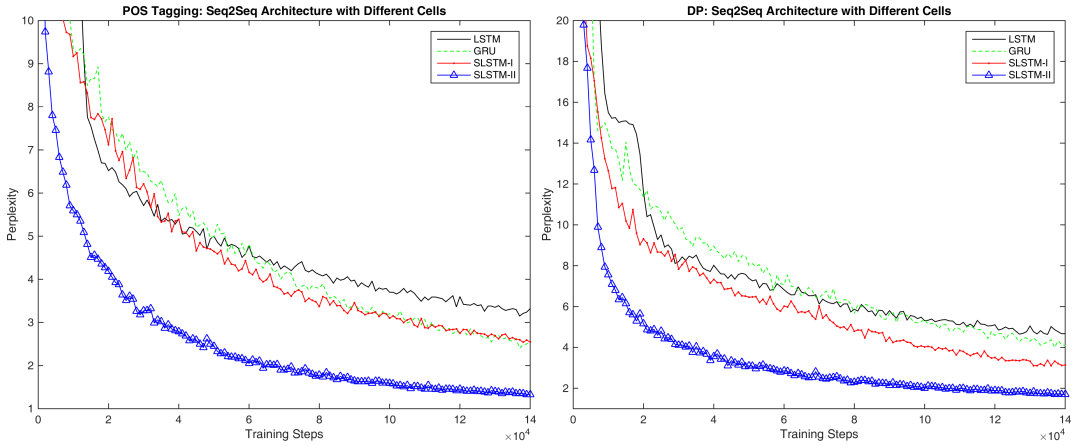
Figure 10: The training perplexity of different architectures with the LSTM (left) and the GRU (right) cells for the DP task.



(a) (b)

(c) (d)

Figure 11: The training perplexity for the basic RNN (top left), the BRNN (top right), the sequence-to-sequence (bottom left) and the DBRNN architectures (bottom right) for the DP task.



(a) (b)

Figure 12: The training perplexity of the sequence-to-sequence architecture with the SLSTM-II cell for the POS tagging (left) and for the DP (right) tasks.