Cell-aware Stacked LSTMs for Modeling Sentences

Jihun Choi **Taeuk Kim** **Sang-goo Lee**
Department of Computer Science and Engineering
Seoul National University, Seoul, Korea
{jhchoi,taeuk,sglee}@europa.snu.ac.kr

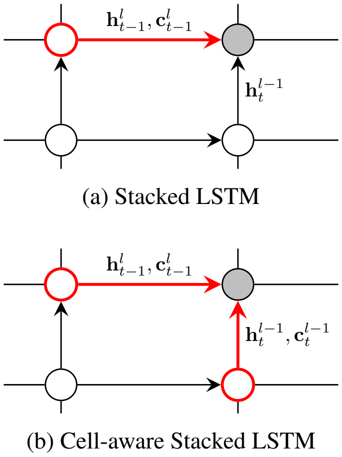
Abstract

We propose a method of stacking multiple long short-term memory (LSTM) layers for modeling sentences. In contrast to the conventional stacked LSTMs where only hidden states are fed as input to the next layer, our architecture accepts both hidden and memory cell states of the preceding layer and fuses information from the left and the lower context using the soft gating mechanism of LSTMs. Thus the proposed stacked LSTM architecture modulates the amount of information to be delivered not only in horizontal recurrence but also in vertical connections, from which useful features extracted from lower layers are effectively conveyed to upper layers. We dub this architecture Cell-aware Stacked LSTM (CAS-LSTM) and show from experiments that our models achieve state-of-the-art results on benchmark datasets for natural language inference, paraphrase detection, and sentiment classification.

1 Introduction

In the field of natural language processing (NLP), the most prevalent neural approach to obtaining sentence representations is to use recurrent neural networks (RNNs), where words in a sentence are processed in a sequential and recurrent manner. Along with their intuitive design, RNNs have shown outstanding performance across various NLP tasks e.g. language modeling (Mikolov et al. 2010; Graves 2013), machine translation (Cho et al. 2014; Sutskever, Vinyals, and Le 2014; Bahdanau, Cho, and Bengio 2015), text classification (Zhou et al. 2015; Tang, Qin, and Liu 2015), and parsing (Kiperwasser and Goldberg 2016; Dyer et al. 2016).

Among several variants of the original RNN (Elman 1990), gated recurrent architectures such as long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) and gated recurrent unit (GRU) (Cho et al. 2014) have been accepted as de-facto standard choices for RNNs due to their capability of addressing the vanishing and exploding gradient problem and considering long-term dependencies. Gated RNNs achieve these properties by introducing additional gating units that learn to control the amount of information to be transferred or forgotten (Goodfellow, Bengio, and Courville 2016), and are proven to work well without relying on complex optimization algorithms or careful initialization (Sutskever 2013).


$\mathbf{h}^l_{t-1}, \mathbf{c}^l_{t-1}$
$\mathbf{h}^{l-1}_t$

(a) Stacked LSTM

$\mathbf{h}^l_{t-1}, \mathbf{c}^l_{t-1}$
$\mathbf{h}^{l-1}_t, \mathbf{c}^{l-1}_t$

(b) Cell-aware Stacked LSTM

Figure 1: Visualization of (a) plain stacked LSTM and (b) CAS-LSTM. The red nodes indicate the blocks whose cell states directly affect the cell state $\mathbf{c}^l_t$.

Meanwhile, the common practice for further enhancing the expressiveness of RNNs is to stack multiple RNN layers, each of which has distinct parameter sets (stacked RNN) (Schmidhuber 1992; El Hihi and Bengio 1996). In stacked RNNs, the hidden states of a layer are fed as input to the subsequent layer, and they are shown to work well due to increased depth (Pascanu et al. 2014) or their ability to capture hierarchical time series (Hermans and Schrauwen 2013) which are inherent to the nature of the problem being modeled.

However this setting of stacking RNNs might hinder the possibility of more sophisticated recurrence-based structures since the information from lower layers is simply treated as input to the next layer, rather than as another class of state that participates in core RNN computations. Especially for gated RNNs such as LSTMs and GRUs, this means that layer-to-layer connections cannot fully benefit from the carefully constructed gating mechanism used in temporal transitions. Some recent work on stacking RNNs suggests alternative methods that encourage direct and effective interaction between RNN layers by adding residual connections (Kim, El-Khamy, and Lee 2017; Nie and Bansal 2017), by shortcut connections (Nie and Bansal 2017; Chen, Ling, and Zhu 2018), or by using cell states of LSTMs (Zhang et al.

2016; Kalchbrenner, Danihelka, and Graves 2016).

In this paper, we propose a method of constructing multi-layer LSTMs where cell states are used in controlling the vertical information flow. This system utilizes states from the left and the lower context equally in computation of the new state, thus the information from lower layers is elaborately filtered and reflected through a soft gating mechanism. Our method is easy-to-implement, effective, and can replace conventional stacked LSTMs without much modification of the overall architecture.

We call the proposed architecture Cell-aware Stacked LSTM, or CAS-LSTM, and evaluate our method on multiple benchmark datasets: SNLI (Bowman et al. 2015), MultiNLI (Williams, Nangia, and Bowman 2018), Quora Question Pairs (Wang, Hamza, and Florian 2017), and SST (Socher et al. 2013). From experiments we show that the CAS-LSTMs consistently outperform typical stacked LSTMs, opening the possibility of performance improvement of architectures that use stacked LSTMs.

Our contribution is summarized as follows.

- We bring the idea of utilizing states coming from multiple directions to construction of stacked LSTM and apply the idea to the research of sentence representation learning. There is some prior work addressing the idea of incorporating more than one type of state (Graves, Fernández, and Schmidhuber 2007; Graves and Schmidhuber 2009; Kalchbrenner, Danihelka, and Graves 2016; Zhang et al. 2016; Yao et al. 2015), however to the best of our knowledge there is little work on applying the idea to sentence encoding and text classification.
- We conduct extensive evaluation of the proposed method and empirically prove its effectiveness on encoding sentences. Our models achieve new state-of-the-art results on SNLI and Quora Question Pairs datasets, and are on par with the best performing models on MultiNLI and SST datasets.

This paper is organized as follows. We give a detailed description about the proposed method in §2. Experimental results are given in §3. We study prior work related to our objective in §4 and conclude in §5.

2 Model Description

In this section, we give a detailed formulation of the architectures used in experiments.

2.1 Notation

Throughout this paper, we denote matrices as boldface capital letters ($\mathbf{A}$), vectors as boldface lowercase letters ($\mathbf{b}$), and scalars as normal italic letters (c). For LSTM states, we denote a hidden state as $\mathbf{h}$ and a cell state as $\mathbf{c}$. Also, a layer index of $\mathbf{h}$ or $\mathbf{c}$ is denoted by superscript and a time index is denoted by a subscript, i.e. $\mathbf{h}^l_t$ indicates the hidden state at time t and layer l. $\mathbf{a} \odot \mathbf{b}$ means the element-wise multiplication between two vectors. We write i-th component of vector $\mathbf{b}$ as b_i. All vectors are assumed to be column vectors.

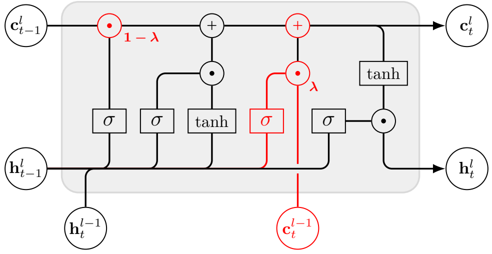


Figure 2: Schematic diagram of a CAS-LSTM block.

2.2 Stacked LSTMs

While there exist various versions of LSTM formulation, in this work we use the following, one of the most common versions:

$$\mathbf{i}^l_t = \sigma(\mathbf{W}^l_i \mathbf{h}^{l-1}_t + \mathbf{U}^l_i \mathbf{h}^l_{t-1} + \mathbf{b}^l_i) \tag{1}$$

$$\mathbf{f}^l_t = \sigma(\mathbf{W}^l_f \mathbf{h}^{l-1}_t + \mathbf{U}^l_f \mathbf{h}^l_{t-1} + \mathbf{b}^l_f) \tag{2}$$

$$\tilde{\mathbf{c}}^l_t = \tanh(\mathbf{W}^l_c \mathbf{h}^{l-1}_t + \mathbf{U}^l_c \mathbf{h}^l_{t-1} + \mathbf{b}^l_c) \tag{3}$$

$$\mathbf{o}^l_t = \sigma(\mathbf{W}^l_o \mathbf{h}^{l-1}_t + \mathbf{U}^l_o \mathbf{h}^l_{t-1} + \mathbf{b}^l_o) \tag{4}$$

$$\mathbf{c}^l_t = \mathbf{i}^l_t \odot \tilde{\mathbf{c}}^l_t + \mathbf{f}^l_t \odot \mathbf{c}^l_{t-1} \tag{5}$$

$$\mathbf{h}^l_t = \mathbf{o}^l_t \odot \tanh(\mathbf{c}^l_t), \tag{6}$$

where $t \in \{1, \cdots, T\}$, $l \in \{1, \cdots, L\}$, and $\mathbf{W}^l_\cdot \in \mathbb{R}^{d_l \times d_{l-1}}$, $\mathbf{U}^l_\cdot \in \mathbb{R}^{d_l \times d_l}$, $\mathbf{b}^l_\cdot \in \mathbb{R}^{d_l}$ are trainable parameters. $\sigma(\cdot)$ and $\tanh(\cdot)$ are the sigmoid activation and the hyperbolic tangent activation function respectively. Also we assume that $\mathbf{h}^0_t = \mathbf{x}_t \in \mathbb{R}^{d_0}$ where $\mathbf{x}_t$ is the t-th input to the network.

The input gate $\mathbf{i}^l_t$ and the forget gate $\mathbf{f}^l_t$ control the amount of information transmitted from $\tilde{\mathbf{c}}^l_t$ and $\mathbf{c}^l_{t-1}$, the candidate cell state and the previous cell state, to the new cell state $\mathbf{c}^l_t$. Similarly the output gate $\mathbf{o}^l_t$ soft-selects which portion of the cell state $\mathbf{c}^l_t$ is to be used in the final hidden state.

We can clearly see that cell states ($\mathbf{c}^l_{t-1}$, $\tilde{\mathbf{c}}^l_t$, $\mathbf{c}^l_t$) play a crucial role in forming horizontal recurrence. However the current formulation does not consider $\mathbf{c}^{l-1}_t$, the cell state from $(l-1)$-th layer, in computation and thus the lower context is reflected only through the rudimentary way, hindering the possibility of controlling vertical information flow.

2.3 Cell-aware Stacked LSTMs

Now we extend the stacked LSTM formulation defined above to address the problem noted in the previous subsection. To enhance the interaction between layers in a way similar to how LSTMs keep and forget the information from the previous time step, we introduce the *additional forget gate* $\mathbf{g}^l_t$ that determines whether to accept or ignore the signals coming from the previous layer. Therefore the proposed Cell-aware Stacked LSTM is formulated as follows:

$$\mathbf{i}^l_t = \sigma(\mathbf{W}^l_i \mathbf{h}^{l-1}_t + \mathbf{U}^l_i \mathbf{h}^l_{t-1} + \mathbf{b}^l_i) \tag{7}$$

$$\mathbf{f}^l_t = \sigma(\mathbf{W}^l_f \mathbf{h}^{l-1}_t + \mathbf{U}^l_f \mathbf{h}^l_{t-1} + \mathbf{b}^l_f) \tag{8}$$

$$\mathbf{g}^l_t = \sigma(\mathbf{W}^l_g \mathbf{h}^{l-1}_t + \mathbf{U}^l_g \mathbf{h}^l_{t-1} + \mathbf{b}^l_g), \tag{9}$$

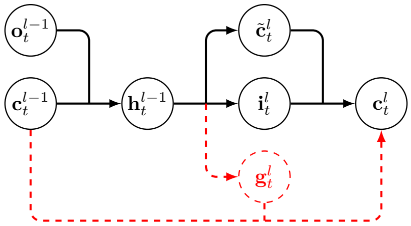


Figure 3: Visualization of paths between $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$. In CAS-LSTM, the direct connection between $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$ exists (denoted as red dashed lines).

$$\tilde{\mathbf{c}}_t^l = \tanh(\mathbf{W}_c^l \mathbf{h}_t^{l-1} + \mathbf{U}_c^l \mathbf{h}_{t-1}^l + \mathbf{b}_c^l) \tag{10}$$

$$\mathbf{o}_t^l = \sigma(\mathbf{W}_o^l \mathbf{h}_t^{l-1} + \mathbf{U}_o^l \mathbf{h}_{t-1}^l + \mathbf{b}_o^l) \tag{11}$$

$$\mathbf{c}_t^l = \mathbf{i}_t^l \odot \tilde{\mathbf{c}}_t^l + (\mathbf{1} - \boldsymbol{\lambda}) \odot \mathbf{f}_t^l \odot \mathbf{c}_{t-1}^l + \boldsymbol{\lambda} \odot \mathbf{g}_t^l \odot \mathbf{c}_t^{l-1} \tag{12}$$

$$\mathbf{h}_t^l = \mathbf{o}_t^l \odot \tanh(\mathbf{c}_t^l), \tag{13}$$

where $l > 1$ and $d_l = d_{l-1}$. $\boldsymbol{\lambda}$ can either be a vector of constants or parameters. When $l = 1$, the equations defined in the previous subsection are used. Therefore, it can be said that each non-bottom layer of CAS-LSTM accepts two sets of hidden and cell states—one from the left context and the other from the below context. The left and the below context participate in computation with the equivalent procedure so that the information from lower layers can be efficiently propagated. Fig. 1 compares CAS-LSTM to the conventional stacked LSTM architecture, and Fig. 2 depicts the computation flow of the CAS-LSTM.

We argue that considering $\mathbf{c}_t^{l-1}$ in computation is beneficial for the following reasons. First, $\mathbf{c}_t^{l-1}$ contains additional information compared to $\mathbf{h}_t^{l-1}$ since it is not filtered by $\mathbf{o}_t^{l-1}$. Thus a model that directly uses $\mathbf{c}_t^{l-1}$ does not rely solely on $\mathbf{o}_t^{l-1}$ for extracting information, due to the fact that it has access to the raw information $\mathbf{c}_t^{l-1}$, as in temporal connections. In other words, $\mathbf{o}_t^{l-1}$ no longer has to take all responsibility for selecting useful features for both horizontal and vertical transitions, and the burden of selecting information is shared with $\mathbf{g}_t^l$.

Another advantage of using the $\mathbf{c}_t^{l-1}$ lies in the fact that it directly connects $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$. This direct connection helps and stabilizes training, since the terminal error signals can be easily backpropagated to model parameters. Fig. 3 illustrates paths between the two cell states.

We find experimentally that there is little difference between letting $\boldsymbol{\lambda}$ be constant and letting it be trainable parameters, thus we set $\lambda_i = 0.5$ in all experiments. We also experimented with the architecture without $\boldsymbol{\lambda}$ i.e. two cell states are combined by unweighted summation similar to multidimensional RNNs (Graves and Schmidhuber 2009), and found that it leads to performance degradation and unstable convergence, likely due to mismatch in the range of cell state values between layers ($[-2, 2]$ for the first layer and $[-3, 3]$ for the others). Experimental results on various $\boldsymbol{\lambda}$ are presented in §3.

Connection to tree-structured RNNs. The idea of having multiple states is also related to tree-structured RNNs (Goller and Kuchler 1996; Socher et al. 2011). Among them, tree-structured LSTMs (Tree-LSTMs) (Tai, Socher, and Manning 2015; Zhu, Sobihani, and Guo 2015; Le and Zuidema 2015) are similar to ours in that they use both hidden and cell states from children nodes. In Tree-LSTMs, states for all children nodes are regarded as input, and they participate in the computation equally through weight-shared (in Child-Sum Tree-LSTMs) or weight-unshared (in N-ary Tree-LSTMs) projection. From this perspective, each CAS-LSTM layer (where $l > 1$) can be seen as a binary Tree-LSTM where the structures it operates on are fixed to right-branching trees. The use of cell state in computation could be one reason that Tree-LSTMs perform better than sequential LSTMs even when trivial trees (strictly left- or right-branching) are given (Williams, Drozdov, and Bowman 2018).

Connection to multidimensional RNNs. Multidimensional RNNs (MDRNN) are an extension of 1D sequential RNNs that can accept multidimensional input e.g. images, and have been successfully applied to image segmentation (Graves, Fernández, and Schmidhuber 2007) and handwriting recognition (Graves and Schmidhuber 2009). Notably multidimensional LSTMs (MDLSTM) (Graves and Schmidhuber 2009) have an analogous formulation to ours except the $\boldsymbol{\lambda}$ term and the fact that we use distinct weights per column (or 'layer' in our case). From this view, CAS-LSTM can be seen as a certain kind of MDLSTM that accepts a 2D input $\{\mathbf{h}_t^l\}_{t=1,l=0}^{T,L}$. Grid LSTMs (Kalchbrenner, Danihelka, and Graves 2016) also take n inputs but emit n outputs, which is different from our case where a single set of hidden and cell states is produced.

2.4 Sentence Encoders

The sentence encoder network we use in our experiments takes T words (assumed to be one-hot vectors) as input. The words are projected to corresponding word representations: $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ where $\mathbf{x}_t \in \mathbb{R}^{d_0}$. Then $\mathbf{X}$ is fed to a L-layer CAS-LSTM model, resulting in the representations $\mathbf{H} = (\mathbf{h}_1^L, \cdots, \mathbf{h}_T^L) \in \mathbb{R}^{T \times d_L}$. The sentence representation, $\mathbf{s} \in \mathbb{R}^{d_L}$, is computed by max-pooling $\mathbf{H}$ over time as in the work of Conneau et al. (2017). Similar to their results, from preliminary experiments we found that the max-pooling performs consistently better than mean- and last-pooling.

To make models more expressive, a bidirectional CAS-LSTM network may also be used. In the bidirectional case, the forward representations $\mathbf{H} = (\mathbf{h}_1^L, \cdots, \mathbf{h}_T^L) \in \mathbb{R}^{T \times d_L}$ and the backward representations $\widehat{\mathbf{H}} = (\widehat{\mathbf{h}}_1^L, \cdots, \widehat{\mathbf{h}}_T^L) \in \mathbb{R}^{T \times d_L}$ are concatenated and max-pooled to yield the sentence representation $\mathbf{s} \in \mathbb{R}^{2d_L}$. We call this bidirectional architecture Bi-CAS-LSTM in experiments.

2.5 Top-layer Classifiers

For the natural language inference experiments, we use the following heuristic function proposed by Mou et al. (2016)

in feature extraction:

$$\phi(\mathbf{s}_1, \mathbf{s}_2) = \mathbf{s}_1 \oplus \mathbf{s}_2 \oplus |\mathbf{s}_1 - \mathbf{s}_2| \oplus (\mathbf{s}_1 \odot \mathbf{s}_2), \tag{14}$$

where $\oplus$ means vector concatenation, and $| \cdot |$ and $-$ are applied element-wise.

And we use the following function in paraphrase identification experiments:

$$\phi(\mathbf{s}_1, \mathbf{s}_2) = |\mathbf{s}_1 - \mathbf{s}_2| \oplus (\mathbf{s}_1 \odot \mathbf{s}_2), \tag{15}$$

as in the work of Ji and Eisenstein (2013).

For sentiment classification, we use the sentence representation itself.

$$\phi(\mathbf{s}) = \mathbf{s} \tag{16}$$

We feed the feature extracted from ϕ as input to the MLP classifier with ReLU activation followed by the fully-connected softmax layer to predict the label distribution:

$$P(\mathbf{y}|\mathbf{X}) = \text{softmax}(\mathbf{W}_c \text{MLP}(\phi(\cdot))), \tag{17}$$

where $\mathbf{W}_c \in \mathbb{R}^{|L| \times d_h}$, $|L|$ is the number of label classes, and d_h the dimension of the MLP output,

3 Experiments

We evaluate our method on natural language inference (NLI), paraphrase identification (PI), and sentiment classification. We also conduct analysis on gate values and experiments on model variants. For detailed experimental settings, we refer readers to the supplemental material.

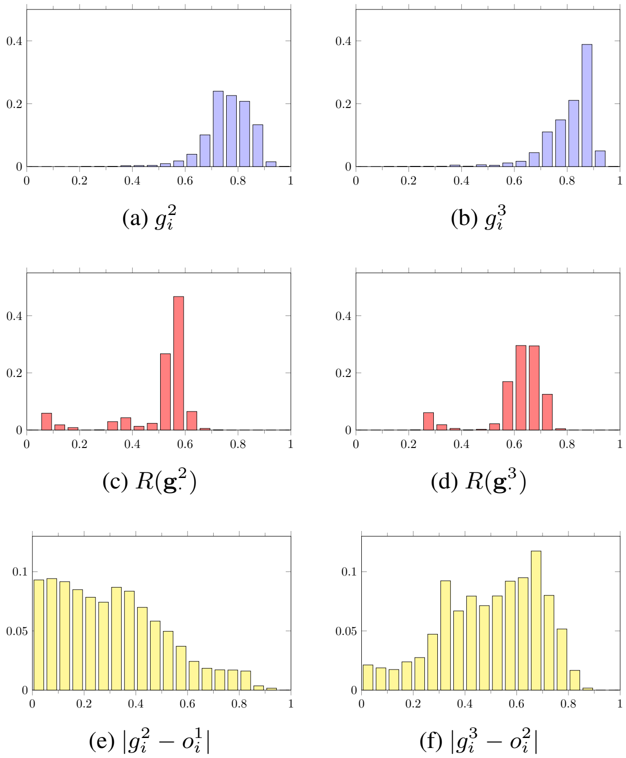
For the NLI and PI tasks, there exists recent work specializing in sentence pair classification. However in this work we confine our model to the architecture that encodes each sentence using a shared encoder without any inter-sentence interaction, in order to focus on the effectiveness of the models in extracting semantics. But note that the applicability of CAS-LSTM is not limited to sentence encoding based approaches.

3.1 Natural Language Inference

For the evaluation of performance of the proposed method on the NLI task, SNLI (Bowman et al. 2015) and MultiNLI (Williams, Nangia, and Bowman 2018) datasets are used. The objective of both datasets is to predict the relationship between a premise and a hypothesis sentence: *entailment*, *contradiction*, and *neutral*. SNLI and MultiNLI datasets are composed of about 570k and 430k premise-hypothesis pairs respectively.

GloVe pretrained word embeddings[1] (Pennington, Socher, and Manning 2014) are used and remain fixed during training. The dimension of encoder states (d_l) is set to 300 and a 1024D MLP with one or two hidden layers is used. We apply dropout (Srivastava et al. 2014) to the word embeddings and the MLP layers. The features used as input to the MLP classifier are extracted following Eq. 14.

Table 1 and 2 contain results of the models on SNLI and MultiNLI datasets. In SNLI, our best model achieves the new state-of-the-art accuracy of 87.0% with relatively fewer parameters. Similarly in MultiNLI, our models match the accuracy of state-of-the-art models in both in-domain (matched) and cross-domain (mismatched) test sets. Note that only the GloVe word vectors are used as word representations, as opposed to some models that introduce character-level features. It is also notable that our proposed architecture does not restrict the selection of pooling method; the performance could further be improved by replacing max-pooling with other advanced algorithms e.g. intra-sentence attention (Liu et al. 2016) and generalized pooling (Chen, Ling, and Zhu 2018).



(a) g_i^2 (b) g_i^3

(c) $R(\mathbf{g}_{\cdot}^2)$ (d) $R(\mathbf{g}_{\cdot}^3)$

(e) $|g_i^2 - o_i^1|$ (f) $|g_i^3 - o_i^2|$

Figure 4: (a), (b): Histograms of vertical forget gate values. (c), (d): Histograms of the ranges of vertical forget gate per time step. (e), (f): Histograms of the absolute difference between the previous output gate and the current vertical forget gate values.

3.2 Paraphrase Identification

We use Quora Question Pairs dataset (Wang, Hamza, and Florian 2017) in evaluating the performance of our method on the PI task. The dataset consists of over 400k question pairs, and each pair is annotated with whether the two sentences are paraphrase of each other or not.

Similar to the NLI experiments, GloVe pretrained vectors, 300D encoders, and 1024D MLP are used. The number of CAS-LSTM layers is fixed to 2 in PI experiments. Two sentence vectors are aggregated using Eq. 15 and fed as input to the MLP. The results on the Quora Question Pairs dataset are summarized in Table 3. Again we can see that our models outperform other models by large margin, achieving the new state of the art.

[1] https://nlp.stanford.edu/projects/glove/

Model	Acc. (%)	# Params
300D LSTM (Bowman et al. 2016)	80.6	3.0M
300D TBCNN (Mou et al. 2016)	82.1	3.5M
300D SPINN-PI (Bowman et al. 2016)	83.2	3.7M
600D BiLSTM with intra-attention (Liu et al. 2016)	84.2	2.8M
4096D BiLSTM with max-pooling (Conneau et al. 2017)	84.5	40M
300D BiLSTM with gated pooling (Chen et al. 2017)	85.5	12M
300D Gumbel Tree-LSTM (Choi, Yoo, and Lee 2018)	85.6	2.9M
600D Shortcut stacked BiLSTM (Nie and Bansal 2017)	86.1	140M
300D Reinforced self-attention network (Shen et al. 2018)	86.3	3.1M
600D BiLSTM with generalized pooling (Chen, Ling, and Zhu 2018)	86.6	65M
300D 2-layer CAS-LSTM (ours)	86.4	2.9M
300D 2-layer Bi-CAS-LSTM (ours)	86.8	6.8M
300D 3-layer CAS-LSTM (ours)	86.4	4.8M
300D 3-layer Bi-CAS-LSTM (ours)	**87.0**	8.6M

Table 1: Results of the models on the SNLI dataset.

Model	In Acc. (%)	Cross Acc. (%)	# Params
CBOW (Williams, Nangia, and Bowman 2018)	64.8	64.5	-
BiLSTM (Williams, Nangia, and Bowman 2018)	66.9	66.9	-
Shortcut stacked BiLSTM (Nie and Bansal 2017)*	**74.6**	73.6	140M
BiLSTM with gated pooling (Chen et al. 2017)	73.5	73.6	12M
BiLSTM with generalized pooling (Chen, Ling, and Zhu 2018)	73.8	**74.0**	18M**
2-layer CAS-LSTM (ours)	74.0	73.3	2.9M
2-layer Bi-CAS-LSTM (ours)	**74.6**	73.7	6.8M
3-layer CAS-LSTM (ours)	73.8	73.1	4.8M
3-layer Bi-CAS-LSTM (ours)	74.2	73.4	8.6M

Table 2: Results of the models on the MultiNLI dataset. ‘In Acc.’ and ‘Cross Acc.’ represent accuracy calculated from the matched and mismatched test set respectively. *: SNLI dataset is used as additional training data. **: computed from hyperparameters provided by the authors.

3.3 Sentiment Classification

In evaluating sentiment classification performance, the Stanford Sentiment Treebank (SST) (Socher et al. 2013) is used. It consists of about 12,000 binary-parsed sentences where constituents (phrases) of each parse tree are annotated with a sentiment label (*very positive*, *positive*, *neutral*, *negative*, *very negative*). Following the convention of prior work, all phrases and their labels are used in training but only the sentence-level data are used in evaluation.

In evaluation we consider two settings, namely SST-2 and SST-5, the two differing only in their level of granularity with regard to labels. In SST-2, data samples annotated with ‘neutral’ are ignored from training and evaluation. The two positive labels (very positive, positive) are considered as the same label, and similarly for the two negative labels. As a result 98,794/872/1,821 data samples are used in training/validation/test, and the task is considered as a binary classification problem. In SST-5, data are used as-is and thus the task is a 5-class classification problem. All 318,582/1,101/2,210 data samples for training/validation/test are used in the SST-5 setting.

We use 300D GloVe vectors, 2-layer 150D or 300D encoders, and a 300D MLP classifier for the models, however unlike previous experiments we tune the word embeddings during training. The results on SST are listed in Table 4. Our models achieve the new state-of-the-art accuracy on SST-2 and competitive accuracy on SST-5, without utilizing parse tree information.

3.4 Forget Gate Analysis

To inspect the effect of the additional forget gate, we investigate how the values of vertical forget gates are distributed. We sample 1,000 random sentences from the development set of the SNLI dataset, and use the 3-layer CAS-LSTM model trained on the SNLI dataset to compute gate values.

If all values from a vertical forget gate $\mathbf{g}_t^l$ were to be 0, this would mean that the introduction of the additional forget gate is meaningless and the model would reduce to a plain stacked LSTM. On the contrary if all values were 1, meaning that the vertical forget gates were always *open*, it would be impossible to say that the information is modulated effectively.

Fig. 4a and 4b represent histograms of the vertical forget gate values from the second and the third layer. From the figures we can validate that the trained model does not fall into the degenerate case where vertical forget gates are ignored. Also the figures show that the values are right-skewed, which we conjecture to be a result of focusing more on a strong interaction between adjacent layers.

To further verify that the gate values are diverse enough within each time step, we compute the distribution of the range of values per time step, $R(\mathbf{g}_t^l) = \max_i g_{t,i}^l - \min_i g_{t,i}^l$, where $\mathbf{g}_t^l = [g_{t,1}^l, \cdots, g_{t,d_l}^l]^\top$. We plot the histograms in Fig. 4c and 4d. From the figure we see that a vertical for-

Model	Acc. (%)
CNN (Wang, Hamza, and Florian 2017)	79.6
LSTM (Wang, Hamza, and Florian 2017)	82.6
Multi-Perspective LSTM (Wang, Hamza, and Florian 2017)	83.2
LSTM + ElBiS (Choi, Kim, and Lee 2018)	87.3
REGMAPR (BASE+REG) (Brahma 2018)	88.0
CAS-LSTM (ours)	88.4
Bi-CAS-LSTM (ours)	**88.6**

Table 3: Results of the models on the Quora Question Pairs dataset.

Model	SST-2 Acc. (%)	SST-5 Acc. (%)
Recursive Neural Tensor Network (Socher et al. 2013)	85.4	45.7
Constituency Tree-LSTM (Tai, Socher, and Manning 2015)	88.0	51.0
Neural Semantic Encoder (Munkhdalai and Yu 2017)	89.7	52.8
Constituency Tree-LSTM with recurrent dropout (Looks et al. 2017)	89.4	52.3
byte mLSTM (Radford, Jozefowicz, and Sutskever 2017)*	91.8	52.9
Gumbel Tree-LSTM (Choi, Yoo, and Lee 2018)	90.7	**53.7**
BCN + Char + ELMo (Peters et al. 2018)*	-	54.7
CAS-LSTM (ours)	91.1	53.0
Bi-CAS-LSTM (ours)	**91.3**	53.6

Table 4: Results of the models on the SST dataset. *: models pretrained on large external corpora are used.

Model	Acc. (%)	Δ
Bi-CAS-LSTM (*baseline*)	87.0	
(i) Plain stacked BiLSTM	86.0	-1.0
(ii) Diverse $\boldsymbol{\lambda}$		
(a) $\lambda_i = 0.25$	86.8	-0.2
(b) $\lambda_i = 0.75$	86.8	-0.2
(c) Trainable $\boldsymbol{\lambda}$	86.9	-0.1
(iii) No $\boldsymbol{\lambda}$	86.6	-0.4
(iv) Integration through peepholes	86.5	-0.5

Table 5: Results of model variants.

get gate controls the amount of information flow effectively, making the decision of retaining or discarding signals.

Finally, to investigate the argument presented in §2 that the additional forget gate helps the previous output gate with reducing the burden of extracting all needed information, we inspect the distribution of the values from $|\mathbf{g}_t^l - \mathbf{o}_t^{l-1}|$. This distribution indicates how differently the vertical forget gate and the previous output gate select information from $\mathbf{c}_t^{l-1}$. From Fig. 4e and 4f we can see that the two gates make fairly different decisions, from which we demonstrate that the direct path between $\mathbf{c}_t^{l-1}$ and $\mathbf{c}_t^l$ enables a model to utilize signals overlooked by $\mathbf{o}_t^{l-1}$.

3.5 Model Variations

In this subsection, we see the influence of each component of a model on performance by removing or replacing its components. the SNLI dataset is used for experiments, and the best performing configuration is used as a baseline for modifications. We consider the following variants: (i) models that use plain stacked LSTMs, (ii) models with different $\boldsymbol{\lambda}$, (iii) models without $\boldsymbol{\lambda}$, and (iv) models that integrate lower contexts via peephole connections.

Variant (iv) integrates lower contexts via the following equations:

$$\mathbf{i}_t^l = \sigma(\mathbf{W}_i^l \mathbf{h}_t^{l-1} + \mathbf{U}_i^l \mathbf{h}_{t-1}^l + \mathbf{p}_i^l \odot \mathbf{c}_{t-1}^l + \mathbf{b}_i^l) \tag{18}$$

$$\mathbf{f}_t^l = \sigma(\mathbf{W}_f^l \mathbf{h}_t^{l-1} + \mathbf{U}_f^l \mathbf{h}_{t-1}^l + \mathbf{p}_f^l \odot \mathbf{c}_{t-1}^l + \mathbf{b}_f^l) \tag{19}$$

$$\mathbf{g}_t^l = \sigma(\mathbf{W}_g^l \mathbf{h}_t^{l-1} + \mathbf{p}_{g_1}^l \odot \mathbf{c}_{t-1}^l + \mathbf{p}_{g_2}^l \odot \mathbf{c}_t^{l-1} + \mathbf{b}_g^l), \tag{20}$$

$$\tilde{\mathbf{c}}_t^l = \tanh(\mathbf{W}_c^l \mathbf{h}_t^{l-1} + \mathbf{U}_c^l \mathbf{h}_{t-1}^l + \mathbf{b}_c^l) \tag{21}$$

$$\mathbf{o}_t^l = \sigma(\mathbf{W}_o^l \mathbf{h}_t^{l-1} + \mathbf{U}_o^l \mathbf{h}_{t-1}^l + \mathbf{p}_o^l \odot \mathbf{c}_{t-1}^l + \mathbf{b}_o^l) \tag{22}$$

$$\mathbf{c}_t^l = \mathbf{i}_t^l \odot \tilde{\mathbf{c}}_t^l + \mathbf{f}_t^l \odot \mathbf{c}_{t-1}^l + \mathbf{g}_t^l \odot \mathbf{c}_t^{l-1} \tag{23}$$

$$\mathbf{h}_t^l = \mathbf{o}_t^l \odot \tanh(\mathbf{c}_t^l), \tag{24}$$

where $\mathbf{p}_\cdot^l \in \mathbb{R}^{d_l}$ represent peephole weights that take cell states into account. Among the above equations, those that use the lower cell state $\mathbf{c}_t^{l-1}$ are Eq. 20 and 23. We can see that $\mathbf{c}_t^{l-1}$ affects the value of $\mathbf{g}_t^l$ only via peephole connections, which makes $\mathbf{g}_t^l$ independent of $\mathbf{h}_{t-1}^l$.

Table 5 summarizes the results of model variants. We can again see that the use of cell states clearly improves sentence modeling performance (*baseline vs. (i)* and *(iv) vs. (i)*). Also from the results of *baseline* and *(ii)*, we validate that the selection of $\boldsymbol{\lambda}$ does not significantly affect performance but introducing $\boldsymbol{\lambda}$ is beneficial (*baseline vs. (iii)*) possibly due to its effect on normalizing information from multiple sources, as mentioned in §2. Finally, from the comparison between *baseline* and *(iv)*, we show that the proposed way of combining the left and the lower contexts leads to better modeling of sentence representations than that of Zhang et al. (2016) in encoding sentences.

4 Related Work

Stacked recurrent neural networks. There is some prior work on methods of stacking RNNs beyond the plain stacked RNNs (Schmidhuber 1992; El Hihi and Bengio

1996). Residual LSTMs (Kim, El-Khamy, and Lee 2017) add residual connections between the hidden states computed at each LSTM layer, and shortcut-stacked LSTMs (Nie and Bansal 2017) concatenate hidden states from all previous layers to make the backpropagation path short. In our method, the lower context is aggregated via a gating mechanism, and we believe it modulates the amount of information to be transmitted in a more efficient and effective way than vector addition or concatenation. Also, compared to concatenation, our method does not significantly increase the number of parameters.[2]

Highway LSTMs (Zhang et al. 2016) and depth-gated LSTMs (Yao et al. 2015) are similar to our proposed models in that they use cell states from the previous layer, and they are successfully applied to the field of automatic speech recognition and language modeling. However in contrast to CAS-LSTM, where the additional forget gate aggregates the previous layer states, and thus contexts from the left and below participate in computation equitably, in Highway LSTMs and depth-gated LSTMs the previous layer states are considered only through peephole connections (Gers, Schraudolph, and Schmidhuber 2002). The comparison of our models and this architecture is presented in §3.

Multidimensional recurrent neural networks. There is another line of research that aims to extend RNNs to operate on multidimensional inputs. Grid LSTMs (Kalchbrenner, Danihelka, and Graves 2016) are a general n-dimensional LSTM architecture that accepts n sets of hidden and cell states as input and yields n sets of states as output, in contrast to our architecture, which emits a single set of states. 2D and 3D Grid LSTMs bring a performance gain on character-level language modeling and machine translation respectively. Multidimensional RNNs (Graves, Fernández, and Schmidhuber 2007; Graves and Schmidhuber 2009) have a similar formulation as ours, except that they do not normalize cell states and weights for all columns (layers) are tied. However they are often employed to model multidimensional data such as images of handwritten text with RNNs, rather than stacking RNN layers for modeling sequential data.

Deep recurrent transitions. Rather than stacking recurrent layers, some work focuses on increasing the depth of horizontal recurrence. Pascanu et al. (2014) have investigated various architectures to increase the depth of RNNs, inter alia Deep Transition RNNs address the problem of deep hidden-to-hidden transitions. Graves (2016) proposed an adaptive computation time algorithm that learns how many micro time steps to take between receiving an input and emitting an output. Fast-Slow RNNs (Mujika, Meier, and Steger 2017) process data on different timescales by letting a fast cell iterate for a fixed number of time steps before a slow cell receives the next input. Multiscale RNNs e.g. Clockwork RNNs (Koutnik et al. 2014) and Hierarchical Multiscale RNNs (Chung, Ahn, and Bengio 2017) can be also regarded as architectures with increased recurrence depth. However as noted by Zilly et al. (2017), increase in recurrent depth results in a longer maximum path than stacking recurrent layers and makes training difficult without careful initialization or architectural choice.

5 Conclusion

In this paper, we proposed a method of stacking multiple LSTM layers for modeling sentences, dubbed CAS-LSTM. It uses not only hidden states but also cell states from the previous layer, for the purpose of controlling the vertical information flow in a more elaborate way. We evaluated the proposed method on various benchmark tasks: natural language inference, paraphrase identification, and sentiment classification. Our models achieve the new state-of-the-art accuracy on SNLI and Quora Question Pairs datasets and obtain comparable results on MultiNLI and SST datasets. The proposed architecture can replace any stacked LSTM under one weak restriction—the size of states should be identical across all layers.

For future work we plan to apply the CAS-LSTM architecture beyond sentence modeling tasks. Various problems e.g. sequence labeling, sequence generation, and language modeling might benefit from sophisticated modulation on context integration. Aggregating diverse contexts from sequential data, e.g. those from forward and backward reading of text, could also be an intriguing research direction.

Acknowledgments

We thank Dan Edmiston for the review of the manuscript.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*, 632–642.

Bowman, S. R.; Gauthier, J.; Rastogi, A.; Gupta, R.; Manning, C. D.; and Potts, C. 2016. A fast unified model for parsing and sentence understanding. In *ACL*, 1466–1477.

Brahma, S. 2018. REGMAPR - A recipe for textual matching. *CoRR, cs.CL/1808.04343v1*.

Chen, Q.; Zhu, X.; Ling, Z.-H.; Wei, S.; Jiang, H.; and Inkpen, D. 2017. Recurrent neural network-based sentence encoder with gated attention for natural language inference. In *RepEval*, 36–40.

Chen, Q.; Ling, Z.-H.; and Zhu, X. 2018. Enhancing sentence embedding with generalized pooling. In *COLING*, 1815–1826.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMNLP*, 1724–1734.

Choi, J.; Kim, T.; and Lee, S.-g. 2018. Element-wise bilinear interaction for sentence matching. In **SEM*, 107–112.

Choi, J.; Yoo, K. M.; and Lee, S.-g. 2018. Learning to compose task-specific tree structures. In *AAAI*, 5094–5101.

[2] The l-th layer of a typical stacked LSTM requires $(d_{l-1}+d_l+1)\times 4d_l$ parameters, and the l-th layer of a shortcut-stacked LSTM requires $(\sum_{k=0}^{l-1} d_k + d_l + 1)\times 4d_l$ parameters. CAS-LSTM uses $(d_{l-1}+d_l+1)\times 5d_l$ parameters at the l-th ($l>1$) layer.

Chung, J.; Ahn, S.; and Bengio, Y. 2017. Hierarchical multiscale recurrent neural networks. In *ICLR*.

Conneau, A.; Kiela, D.; Schwenk, H.; Barrault, L.; and Bordes, A. 2017. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*, 670–680.

Dyer, C.; Kuncoro, A.; Ballesteros, M.; and Smith, N. A. 2016. Recurrent neural network grammars. In *NAACL-HLT*, 199–209.

El Hihi, S., and Bengio, Y. 1996. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, 493–499.

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

Gers, F. A.; Schraudolph, N. N.; and Schmidhuber, J. 2002. Learning precise timing with lstm recurrent networks. *JMLR* 3(Aug):115–143.

Goller, C., and Kuchler, A. 1996. Learning task-dependent distributed representations by backpropagation through structure. *Neural Networks* 1:347–352.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Graves, A., and Schmidhuber, J. 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In *NIPS*, 545–552.

Graves, A.; Fernández, S.; and Schmidhuber, J. 2007. Multi-dimensional recurrent neural networks. In *ICANN*, 549–558.

Graves, A. 2013. Generating sequences with recurrent neural networks. *CoRR, cs.NE/1308.0850v5*.

Graves, A. 2016. Adaptive computation time for recurrent neural networks. *CoRR, cs.NE/1603.08983v6*.

Hermans, M., and Schrauwen, B. 2013. Training and analysing deep recurrent neural networks. In *NIPS*, 190–198.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Ji, Y., and Eisenstein, J. 2013. Discriminative improvements to distributional sentence similarity. In *EMNLP*, 891–896.

Kalchbrenner, N.; Danihelka, I.; and Graves, A. 2016. Grid long short-term memory. In *ICLR*.

Kim, J.; El-Khamy, M.; and Lee, J. 2017. Residual LSTM: Design of a deep recurrent architecture for distant speech recognition. In *INTERSPEECH*, 1591–1595.

Kiperwasser, E., and Goldberg, Y. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *TACL* 4:313–327.

Koutnik, J.; Greff, K.; Gomez, F.; and Schmidhuber, J. 2014. A clockwork rnn. In *ICML*, 1863–1871.

Le, P., and Zuidema, W. 2015. Compositional distributional semantics with long short term memory. In **SEM*, 10–19.

Liu, Y.; Sun, C.; Lin, L.; and Wang, X. 2016. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR, cs.CL/1605.09090v1*.

Looks, M.; Herreshoff, M.; Hutchins, D.; and Norvig, P. 2017. Deep learning with dynamic computation graphs. In *ICLR*.

Mikolov, T.; Karafiát, M.; Burget, L.; Černockỳ, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *INTERSPEECH*, 1045–1048.

Mou, L.; Men, R.; Li, G.; Xu, Y.; Zhang, L.; Yan, R.; and Jin, Z. 2016. Natural language inference by tree-based convolution and heuristic matching. In *ACL*, 130–136.

Mujika, A.; Meier, F.; and Steger, A. 2017. Fast-slow recurrent neural networks. In *NIPS*, 5915–5924.

Munkhdalai, T., and Yu, H. 2017. Neural semantic encoders. In *EACL*, 397–407.

Nie, Y., and Bansal, M. 2017. Shortcut-stacked sentence encoders for multi-domain inference. In *RepEval*, 41–45.

Pascanu, R.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. How to construct deep recurrent neural networks. In *ICLR*.

Pennington, J.; Socher, R.; and Manning, C. 2014. GloVe: Global vectors for word representation. In *EMNLP*, 1532–1543.

Peters, M.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *NAACL-HLT*, 2227–2237.

Radford, A.; Jozefowicz, R.; and Sutskever, I. 2017. Learning to generate reviews and discovering sentiment. *CoRR, cs.LG/1704.01444v2*.

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

Shen, T.; Zhou, T.; Long, G.; Jiang, J.; Wang, S.; and Zhang, C. 2018. Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling. In *IJCAI*, 4345–4352.

Socher, R.; Lin, C. C.; Manning, C. D.; and Ng, A. Y. 2011. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 129–136.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 1631–1642.

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

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*, 3104–3112.

Sutskever, I. 2013. *Training recurrent neural networks*. Ph.D. Dissertation, University of Toronto.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*, 1556–1566.

Tang, D.; Qin, B.; and Liu, T. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, 1422–1432.

Wang, Z.; Hamza, W.; and Florian, R. 2017. Bilateral multi-perspective matching for natural language sentences. In *IJCAI*, 4144–4150.

Williams, A.; Drozdov, A.; and Bowman, S. R. 2018. Do latent tree learning models identify meaningful structure in sentences? *TACL* 6:253–267.

Williams, A.; Nangia, N.; and Bowman, S. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT*, 1112–1122.

Yao, K.; Cohn, T.; Vylomova, K.; Duh, K.; and Dyer, C. 2015. Depth-gated recurrent neural networks. *CoRR, cs.NE/1508.03790v4*.

Zhang, Y.; Chen, G.; Yu, D.; Yaco, K.; Khudanpur, S.; and Glass, J. 2016. Highway long short-term memory RNNs for distant speech recognition. In *ICASSP*, 5755–5759.

Zhou, C.; Sun, C.; Liu, Z.; and Lau, F. 2015. A C-LSTM neural network for text classification. *CoRR, cs.CL/1511.08630v2*.

Zhu, X.; Sobihani, P.; and Guo, H. 2015. Long short-term memory over recursive structures. In *ICML*, 1604–1612.

Zilly, J. G.; Srivastava, R. K.; Koutník, J.; and Schmidhuber, J. 2017. Recurrent highway networks. In *ICML*, 4189–4198.