Nested LSTMs

Joel Ruben Antony Moniz JRMONIZ@ANDREW.CMU.EDU
Carnegie Mellon University[*]

David Krueger DAVID.KRUEGER@UMONTREAL.CA
MILA, Université de Montréal

Editors: Yung-Kyun Noh and Min-Ling Zhang

Abstract

We propose *Nested LSTMs* (NLSTM), a novel RNN architecture with multiple levels of memory. Nested LSTMs add depth to LSTMs via nesting as opposed to stacking. The value of a memory cell in an NLSTM is computed by an LSTM cell, which has its own *inner* memory cell. Specifically, instead of computing the value of the (outer) memory cell as $c_t^{outer} = f_t \odot c_{t-1} + i_t \odot g_t$, NLSTM memory cells use the concatenation $(f_t \odot c_{t-1}, i_t \odot g_t)$ as input to an inner LSTM (or NLSTM) memory cell, and set c_t^{outer} = h_t^{inner}. Nested LSTMs outperform both stacked and single-layer LSTMs with similar numbers of parameters in our experiments on various character-level language modeling tasks, and the inner memories of an LSTM learn longer term dependencies compared with the higher-level units of a stacked LSTM.

Keywords: Nested LSTMs, LSTMs, Character-Level Language Modeling

1. Introduction

Learning long-term dependencies is a key challenge for current machine learning approaches to artificial intelligence. The ability of human beings to reconcile these long-term dependencies with the immediate context, i.e., to adapt and use knowledge that has been previously gained so as to be relevant to the current frame-of-reference, is indispensable. An important example of this ability, if on a much smaller scale, is the ability to predict characters and words in a sentence or document based on one's past experience (for example, in the form of commonly encountered constructions and phrases), the general subject dealt with in the document, and the precise wording of the specific sentence in question. Recurrent neural network based architectures have made significant progress towards having a machine mimic this ability.

Recurrent neural networks (RNNs) condition their present state on their entire history of inputs (or "observations" in reinforcement learning parlance), and so are a natural fit for learning temporally abstracted features. In theory, a simple RNN can represent arbitrary functions and thus have the capacity to solve tasks involving dependencies at arbitrary time-scales. In practice, more complex architectures have proven essential for solving many tasks. One reason for this is the vanishing gradient problem (Hochreiter, 1991; Bengio et al., 1994), which makes it difficult for simple RNNs to learn long-term dependencies. Successful RNN architectures, such as LSTMs (Hochreiter and Schmidhuber, 1997) typically incorporate memory mechanisms which ameliorate the problem of vanishing gradient.

[*] Work begun while the author was at MILA and École Polytechnique de Montréal

A more fundamental issue is that learning to detect long-term dependencies involves a fundamentally difficult credit assignment problem: in the absence of prior information, any past event may plausibly be responsible for current events. Architectural features such as memory mechanisms encode implicit priors which may help with the credit assignment problem. Memory mechanisms allow a model to remember past information over arbitrarily long time-scales, so that credit to be assigned to events in the distant past. We seek to encode an additional implicit prior of *temporal hierarchy* by the creation of a novel memory structure. In particular, we suggest selective memory access via nesting as an approach to constructing temporal hierarchies in memory.

While some prior work on hierarchical memory exists, LSTM (and variants) are still the most popular deep learning model for sequential tasks, such as in character-level language modeling. In particular, the default Stacked LSTM architecture uses a sequence of LSTMs stacked on top of each other to process the data, the input to a layer being the output of the previous layer. In this work, we propose and explore a novel *Nested* LSTM architecture (NLSTM), which we envision as a potential drop-in replacement for a stacked LSTM.

In NLSTMs, the LSTM memory cells have access to an *inner* memory, which they selectively read and write to using the standard LSTM gates. This key feature allows the model to implement a more effective temporal hierarchy than a conventional Stacked LSTM. In NLSTM, the (outer) memory cell are free to selectively read and write relevant long-term information to their inner cell. In contrast, in stacked LSTMs, the upper-level activations (analogous to the inner memories) are directly accessed to produce an output, and therefore must contain all the short-term information which is relevant to the current prediction. In other words, the primary difference between stacked LSTMs and Nested LSTMs is the idea of selective access to inner memories which the NLSTM implements. This frees the inner memories to remember and process events on longer time scales, even when these events are not relevant to the immediate present.

Our visualizations demonstrate that the inner memories of NLSTMs do in fact operate on longer time-scales than higher-level memories in a stacked LSTM. Our experiments also show that NLSTMs outperform Stacked LSTMs in a wide range of tasks.

2. Related Work

The problem of learning effective temporal hierarchies for dealing with long-term dependencies is well studied in the context of both RNNs and reinforcement learning. A comprehensive review of this topic is beyond our scope, we review some recent works and focus on the distinctive aspects of our approach.

Doing credit assignment over long time-scales is a central problem of reinforcement learning. The options framework in RL (Sutton et al., 1999) enables long-term planning over sequences of temporally abstracted actions called options. Selecting an option amounts to temporarily enacting a subpolicy which then selects the primitive actions at each time-step (or its own options). Although learning options has received some attention (Stolle and Precup, 2002; Brunskill and Li, 2014), including some recent gradient-based approaches (Arulkumaran et al., 2016; Bacon et al., 2016), most successful applications so far have used hand-crafted options.

2.1. Deep learning approaches to temporal abstraction

Currently, RNNs are frequently *stacked* creating a multi-layer feedforward network at each time-step. Hermans and Schrauwen (2013) argue that stacking may results in more abstract, long-term features; Zhang et al. (2016) argue that this may not be the case. Unlike stacking, nesting also increases recurrent depth. Although Zhang et al. (2016) argue that increased recurrent depth can worsen vanishing/exploding gradients, we believe the gated memory mechanism counter-acts this problem, allowing increased recurrent depth without deteriorating gradient flow. The authors also found empirically that increasing recurrent depth could result in better performance. Pascanu et al. (2013) add multi-layer input, output, or recurrent connections as an alternative to stacking; their deep recurrent connections increase recurrent depth, but are not commonly used. Multi-layer input connections have been used, however, for state-of-the-art speech-recognition (Hannun et al., 2014; Amodei et al., 2016) systems; these systems also incorporate stacked RNNs.

Our model is based on the popular Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) architecture. The hidden states of LSTMs include internal memory cells, which use identity connections to store long-term memories. The LSTM forget/remember[1] gate (Gers et al., 1999) allows memories to be forgotten with an (adaptive) multiplicative decay on these identity connections.

A wide variety of network architectures based on or inspired by LSTMs have been proposed (Graves et al., 2007; Cho et al., 2014; Chung et al., 2014; Kalchbrenner et al., 2015; Danihelka et al., 2016; Cheng et al., 2016). Perhaps the most popular and well know is the Gated Recurrent Unit (GRU) (Cho et al., 2014; Chung et al., 2014). GRUs function similarly to LSTMs, but they do not feature any internal memory; the entire hidden state is exposed to external computational units. This moves in the opposite direction of our work, which is focused on creating *more* internal memory. Some recent works with also apportion more of the total hidden-state into internal memories (Cheng et al., 2016; Rocki, 2016), but not in a way which involves nesting. Greff et al. (2015); Jozefowicz et al. (2015) evaluate architectural variants of LSTMs and GRUs; Greff et al. (2015) remove components of standard LSTMs, whereas Jozefowicz et al. (2015) use an evolutionary search procedure to search a wider space of possible models.

The LSTM remember gates allow the model to dynamically decay memories of different units at different rates, but do not explicitly encourage different units to model different levels of temporal dependency. Some other works attempt to encode the temporal hierarchy in the prior of a recurrent model. Temporal hierarchies among units can explicitly coded by hand, as in Clockwork RNNs (Koutnik et al., 2014) and hierarchical RNNs (Hihi and Bengio, 1996). This approach seems brittle; it would be preferable for the model to learn to operate at the appropriate time-scales. Chung et al. (2015) present a fully differentiable approach to this problem, based on adding additional gating mechanisms. A downside of this work is that the model size grows quadratically in the number of layers in the hierarchy. More recently, Chung et al. (2016) use the straight-through estimator (Hinton, 2012; Yoshua Bengio, 2013) to train a model which makes “crisp” binary decisions about when to update different recurrent units.

Recent work in Deep Learning considers augmenting RNN architectures with novel memory mechanisms inspired by computer memory architectures (Graves et al., 2014; Joulin and Mikolov, 2015; Grefenstette et al., 2015) and neural short-term memory mechanisms (Ba et al., 2016). Storing and accessing memories provide paths for gradient flow, but, just as in RNNs, using backpropagation through time becomes prohibitively computationally expensive when sequences become long. The standard solution to this problem is truncate gradient flow after some number of time-steps. Zaremba

[1] We support the efforts of Ba et al. (2016) to reverse this counter-intuitive naming convention.

and Sutskever (2015) attempt to use reinforcement learning (specifically, the REINFORCE algorithm (Williams, 1992)) to solve this problem in the context of training Neural Turing Machines (NTMs).

3. Nested LSTMs

The output gate in LSTMs encodes the intuition that memories which are not relevant at the present time-step may still be worth remembering. Nested LSTMs use this intuition to create a temporal hierarchy of memories. Access to the inner memories is gated in exactly the same way, so that longer-term information which is only situationally relevant can be accessed selectively.

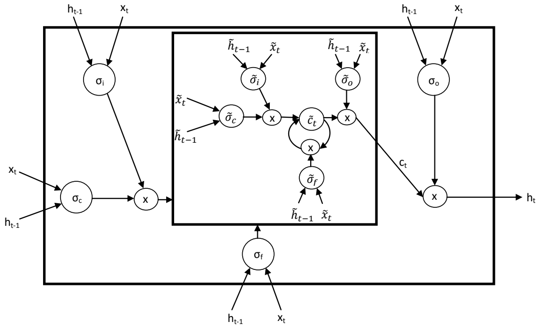


Figure 1: The Nested LSTM architecture

3.1. The architecture

In an LSTM, the equations updating the cell state and the gates are given by:

$$\boldsymbol{i}_t = \boldsymbol{\sigma}_i(\boldsymbol{x}_t \boldsymbol{W}_{xi} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i) \tag{1}$$

$$\boldsymbol{f}_t = \boldsymbol{\sigma}_f(\boldsymbol{x}_t \boldsymbol{W}_{xf} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f) \tag{2}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \tag{3}$$

$$+ \boldsymbol{i}_t \odot \boldsymbol{\sigma}_c(\boldsymbol{x}_t \boldsymbol{W}_{xc} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c) \tag{4}$$

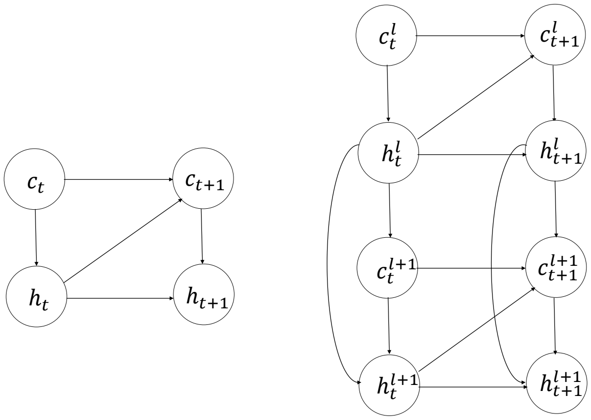
$$\boldsymbol{o}_t = \boldsymbol{\sigma}_o(\boldsymbol{x}_t \boldsymbol{W}_{xo} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o) \tag{5}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \boldsymbol{\sigma}_h(\boldsymbol{c}_t) \tag{6}$$

Note that these equations are similar to those defined in Graves (2013), but do not include peephole connections.

Nested LSTMs replace the addition operation used to compute c_t in LSTMs with a learned, stateful function, $c_t = m_t(f_t \odot c_{t-1}, i_t \odot g_t)$. We refer to the state of the function, m at time t as the *inner memory*, and calling the function to compute c_t also computes m_{t+1}. We chose to implement the memory function as another LSTM memory cell, producing a nested LSTM (see Figure 1 for an illustration). The memory function could instead be another *Nested* LSTM cell, permitting arbitrarily deep nesting.

Given these architecture choices, the input and the hidden states of the memory function in an NLSTM become:



(a) LSTM (b) Stacked LSTM (c) Nested LSTM

Figure 2: Computational graphs of the LSTM, Stacked LSTM and Nested LSTM. The hidden state, outer memory cell, and inner memory cell are represented by h, c, and d, respectively. While the current hidden state can influence the contents of the next inner memory cell directly, the inner memory influences the hidden state only via the outer memory.

$$\widetilde{\boldsymbol{h}}_{t-1} = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \tag{7}$$

$$\widetilde{\boldsymbol{x}}_t = \boldsymbol{i}_t \odot \boldsymbol{\sigma}_c(\boldsymbol{x}_t \boldsymbol{W}_{xc} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c) \tag{8}$$

In particular, note that if the memory function is addition, the entire system reduces to the classical LSTM, since the cell update becomes

$$\boldsymbol{c}_t = \widetilde{\boldsymbol{h}}_{t-1} + \widetilde{\boldsymbol{x}}_t \tag{9}$$

In the architectural variant of the Nested LSTM proposed here, an LSTM is used as the memory function, and the working of the inner LSTM is governed by:

$$\widetilde{\boldsymbol{i}}_t = \widetilde{\boldsymbol{\sigma}}_i(\widetilde{\boldsymbol{x}}_t \widetilde{\boldsymbol{W}}_{xi} + \widetilde{\boldsymbol{h}}_{t-1} \widetilde{\boldsymbol{W}}_{hi} + \widetilde{\boldsymbol{b}}_i) \tag{10}$$

$$\widetilde{\boldsymbol{f}}_t = \widetilde{\boldsymbol{\sigma}}_f(\widetilde{\boldsymbol{x}}_t \widetilde{\boldsymbol{W}}_{xf} + \widetilde{\boldsymbol{h}}_{t-1} \widetilde{\boldsymbol{W}}_{hf} + \widetilde{\boldsymbol{b}}_f) \tag{11}$$

$$\begin{aligned}\widetilde{\boldsymbol{c}}_t = {} & \widetilde{\boldsymbol{f}}_t \odot \widetilde{\boldsymbol{c}}_{t-1} \\ & + \widetilde{\boldsymbol{i}}_t \odot \widetilde{\boldsymbol{\sigma}}_c(\widetilde{\boldsymbol{x}}_t \widetilde{\boldsymbol{W}}_{xc} + \widetilde{\boldsymbol{h}}_{t-1} \widetilde{\boldsymbol{W}}_{hc} + \widetilde{\boldsymbol{b}}_c)\end{aligned} \tag{12}$$

$$\widetilde{\boldsymbol{o}}_t = \widetilde{\boldsymbol{\sigma}}_o(\widetilde{\boldsymbol{x}}_t \widetilde{\boldsymbol{W}}_{xo} + \widetilde{\boldsymbol{h}}_{t-1} \widetilde{\boldsymbol{W}}_{ho} + \widetilde{\boldsymbol{b}}_o) \tag{13}$$

$$\widetilde{\boldsymbol{h}}_t = \widetilde{\boldsymbol{o}}_t \odot \widetilde{\boldsymbol{\sigma}}_h(\widetilde{\boldsymbol{c}}_t) \tag{14}$$

The cell state update of the outer LSTM now becomes:

$$\boldsymbol{c}_t = \widetilde{\boldsymbol{h}}_t \tag{15}$$

4. Experiments

We evaluate Nested LSTMs on a wide variety of datasets and tasks: the PennTreebank Corpus and the larger Text8 dataset (both representing standard character-level language modeling, with Text8 being much larger than the PennTreebank Corpus), the Chinese Poem Generation dataset (which requires character-level language modeling on much smaller sequence with less temporal dependency than is common, but with a significantly larger number of characters than would typically be found in English), and the MNIST Glimpses task (which is a classification task, but one that contains temporal dependencies). We show that, in spite of these tasks representing diverse scenarios and objectives, nested LSTMs consistently improve performance over corresponding stacked LSTM baselines with a comparable number of parameters.

As is usual, we set the σ_i, σ_f and σ_o to the $sigmoid$ activation function, and σ_c and σ_h to $tanh$ in all our baseline LSTM runs. We set σ_i, σ_f, σ_o, $\widetilde{\sigma}_i$, $\widetilde{\sigma}_f$ and $\widetilde{\sigma}_o$ to the $sigmoid$ activation function, $\widetilde{\sigma}_c$, $\widetilde{\sigma}_h$ and σ_h to $tanh$, and σ_c to the $identity$ function in all our Nested LSTM runs

In all the following experiments, we initialize the hyperparameters of the Nested LSTM and the stacked LSTM baselines identically. Although we explicitly specify the hyperparameters, unless otherwise mentioned, the hyperparameters we use are identical to those used in Krueger et al. (2016); Cooijmans et al. (2016). We initialize the nested and stacked LSTMs' first input gates (which convert the input vector from having $vocabulary$ number of elements to $cell_size$ number of elements) using a Glorot initialization scheme Glorot and Bengio (2010), while all other gates in the LSTM (the other input, remember and output gates, the final output gate) are initialized using orthogonal initialization Saxe et al. (2013).

We also try to match the number of parameters of the different stacked LSTM baselines as closely as possible with our 2-layer Nested LSTM by adjusting the number of hidden units. While this is possible precisely with the 2-layer stacked baseline, this is slightly harder to achieve with a single-layered or 3-layered stacked LSTM. Correspondingly, we show the results of two single-layered LSTMs: one with the same number of parameters as in the reference paper, and one with a number of parameters *larger* than those used by our model. We also choose the number of hidden units of the 3-layered stacked LSTM to surpass the number of parameters used by our model. Thus, our model has an equal number of parameters as the 2-layered stacked LSTM, and is at a *disadvantage* to the larger single layered LSTM and the 3-layered stacked LSTM, but outperforms all these baselines.

4.1. Visualization

To analyze what the cell activations look like and how they depend on each other, we aim to visualize the changes in the cell activation states of both the inner and outer cell in the Nested LSTM as the sequence is fed into it. We do this on the model trained on the Penn Treebank dataset as described in Section 4.2, and then visualize the cell states as a sequence from the test set is fed into the Nested LSTM as in Karpathy et al. (2015).

We show our resulting visualization in Figure 3. The cells for which the visualizations have been shown are the first seven cells of the model. From the visualization, we see that the inner LSTM's cell activations tend to be relatively consistent across many time-steps, while the outer LSTM's cell activations fluctuate much more rapidly. This visualization demonstrates that the NLSTM hierarchy works as expected: outer memory operates at a shorter time-scale and uses inner memory to store longer-term information.

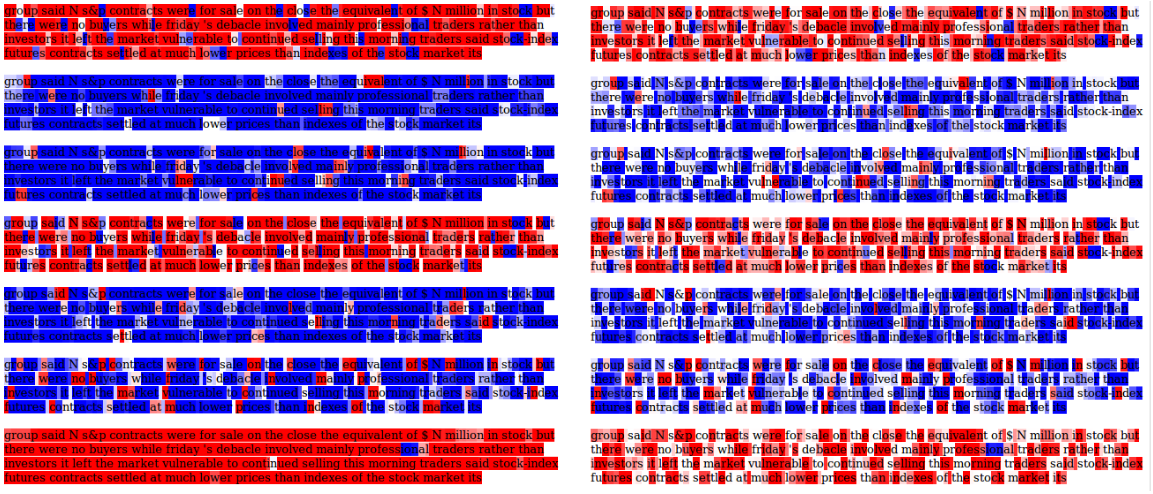


Figure 3: A visualization of the cell activations corresponding to the input character for the inner cell (left) and the outer cell (right). Red implies a negative cell state value, and blue a positive one. A darker shade implies a larger magnitude. In the case of the states of the inner LSTM, we visualize $tanh(\widetilde{\boldsymbol{c}}_t)$ (since $\widetilde{\boldsymbol{c}}_t$ is not constrained), while in the case of the outer LSTM, we directly visualize $\boldsymbol{c}_t$.

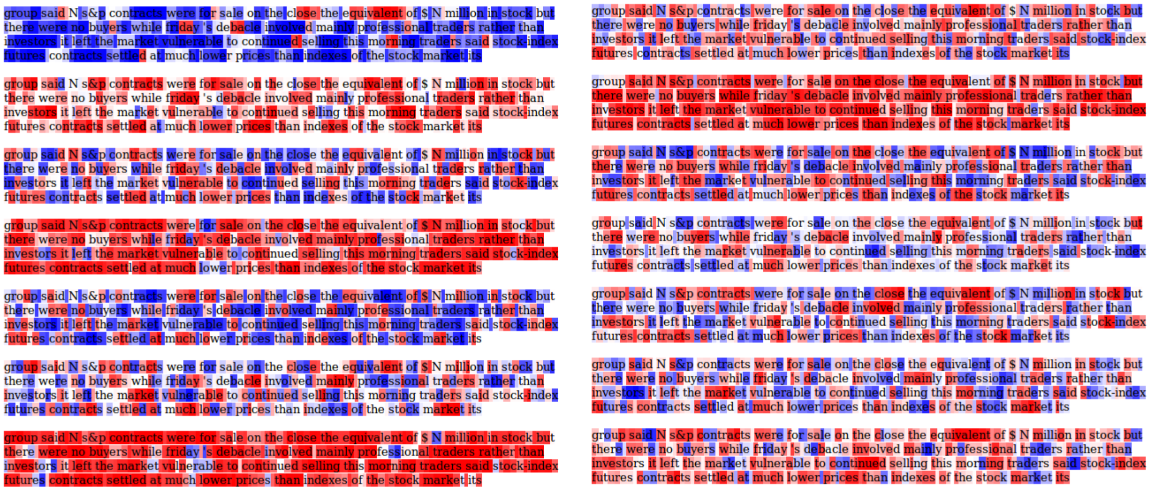


Figure 4: A visualization of $tanh(\boldsymbol{c}_t^n)$, representing the cell activations, corresponding to the input character for the first (right) and second (left) stacked layers. Red implies a negative cell state value, and blue a positive one. A darker shade implies a larger magnitude.

We contrast this with a similar visualization of a 2-layer stacked LSTM baseline, in Figure 4. While the higher layer memory (which is "further away" from the input) operates at a longer time-scale than the lower layer memory, it still fluctuates more rapidly than the inner cells of the NLSTM. This indicates that the NLSTM's ability to selectively process and remember information across multiple levels of nested memories frees the model to remember information over longer-periods, and supports our intuition that nested memory structures can form more effective temporal hierarchies.

4.2. PennTreebank Character-level Language Modeling

The Penn Treebank dataset contains around 1 million words, with a standard train:validation:test split. We train models on this dataset to perform character-level prediction, given an input sequence, and measure the negative log likelihood (NLL) loss and the bits per character (BPC, defined as the NLL divided by $ln2$).

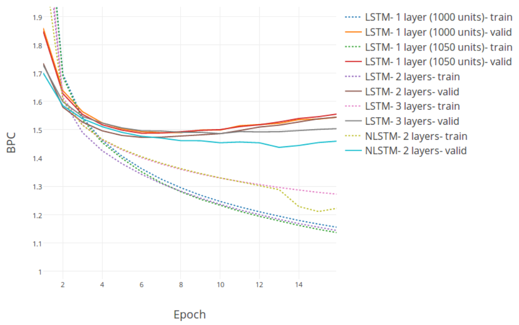


Figure 5: BPC as a function of epoch for character-level language modeling on PTB's test and validation sets

Model	n	cell size	Params	Valid	Test
LSTM	1	1000	4.25M	1.489	1.451
LSTM	1	1050	4.68M	1.487	1.448
LSTM	2	600	4.47M	1.473	1.434
LSTM	3	450	4.17M	1.486	1.448
NLSTM	2	600	4.47M	**1.437**	**1.399**

Table 1: BPC Losses for the Nested LSTM versus various baselines. The test BPC losses correspond to the respective model's loss at the epoch in which it had the minimum valid BPC (also shown).

For Penn Treebank, our first baseline is a single layer LSTM of 1000 hidden units, following prior works (Graves, 2013; Krueger and Memisevic, 2015; Krueger et al., 2016; Cooijmans et al., 2016). We compare this architecture with 2-layer and 3-layer stacked LSTMs and 2-layer nested LSTMs. The number of hidden units of each model is chosen to (approximately) balance the capacity at around 4 million parameters. We also choose a single layered LSTM with a larger number of parameters than the 2-layered LSTM and NLSTM models. We train using Adam (Kingma and Ba, 2014) with a learning rate of 0.002 in sequences of 100 and batches of 32, and clip gradients with a threshold of 1, as in the aforementioned papers. However, we train on non-overlapping sequences, and without any normalization (which we believe could further improve these results). We train each model for 35 epochs.

We find that nested LSTMs yield an improvement of .035 BPC over stacked LSTMs using the same number of hidden units and layers, which, in turn, outperforms other baseline models. Notably, both models and the 3-layered stack LSTM outperform the single-layer network, suggesting that the common use of single-layer nets for this task is sub-optimal. Learning curves are presented in Figure 5.

4.3. Chinese Poetry Generation

Model	n	cell size	Params	Valid	Test
LSTM	1	32	868k	680.15	669.99
LSTM	1	40	1.1M	674.82	670.27
LSTM	2	32	877k	810.54	771.88
LSTM	3	32	885k	944.93	925.47
NLSTM	2	32	877k	**629.71**	**625.19**

Table 2: Perplexity for the Nested LSTM versus various baselines on the Chinese Poetry Generation dataset

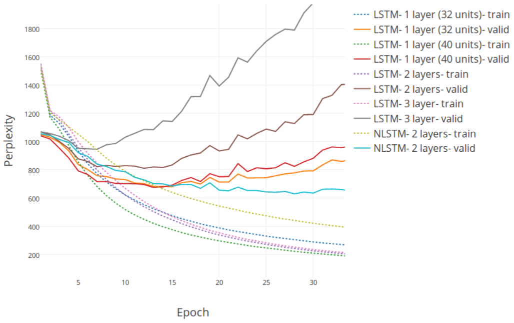


Figure 6: Perplexity as a function of epoch for character-level prediction on ChinesePG's train and validation sets

Here, we use the subset of the Chinese Poem Generation dataset Zhang and Lapata (2014) comprised of quatrains with 5-characters each, with the standard specified train:validation:test split. This task is significantly different from the PTB task: the sequence length (and as a result the length of the temporal dependency) is much shorter, but the number of characters (over 5000) is *two orders of magnitude* larger.

We follow the hyperparameter set used by the LSTM character-level prediction task used as the baseline in Yu et al. (2017) (using a learning rate of 0.002). Following suit, we keep one of our baselines as single-layered LSTM with a cell size of 32. Our other baselines are a single layered LSTM with cell size 40, a 2-layered stacked LSTM with cell size 32 and a 3-layered stacked LSTM with cell size 32. We compare these baselines against our Nested LSTM with a cell size of 32. Note that because of the small cell sizes, the number of parameters in all these models is roughly the same (around 850k) in spite of the different numbers of layers (except in the case of the single layered LSTM with a cell size of 40, which has around 1.1M parameters). All models that we compare against, however, have more or equal parameters when compared to the Nested LSTM (except for the 32-cell single layered LSTM, which is why we introduce the additional 40-cell single layered baseline). We measure the performance using perplexity (which is e^{NLL}), as in Che et al. (2017).

We find that the Nested LSTM outperforms the second best performing baseline by a perplexity of 45.11 on the valid set, and 44.80 on the test set. Surprisingly, we find that both the single-layered LSTM baselines outperform the corresponding stacked LSTM baselines, even the one-layered LSTM with slightly fewer parameters. This is possibly because of the relatively small cell size used in this experiment. However, we observe that the Nested LSTM outperforms the single-layered LSTM in this case as well, pointing to its robustness with respect to the model size (i.e., the number of cells, and by extension, parameters, used in the model).

4.4. MNIST Glimpses

Model	n	cell size	Params	Valid NLL	Test NLL	Valid Accuracy (%)	Test Accuracy (%)
LSTM	1	100	61.0k	0.1007	0.1229	97.85%	97.19%
LSTM	1	130	94.9k	0.1070	0.1242	97.89%	97.51%
LSTM	2	75	83.6k	0.1040	0.1149	98.15%	97.45%
LSTM	3	75	85.1k	0.1077	0.1242	98.07%	97.46%
NLSTM	2	75	83.6k	**0.0836**	**0.1136**	**98.23%**	**97.60%**

Table 3: NLL and percentage error for the Nested LSTM versus various baselines on the MNIST Glimpses task. The epoch at which the percentage error has been show corresponds to that at which each model had the lowest percentage error on the validation set. Similarly for NLL, the model's validation NLL has been used to determine the epoch at which the test NLL is examined.

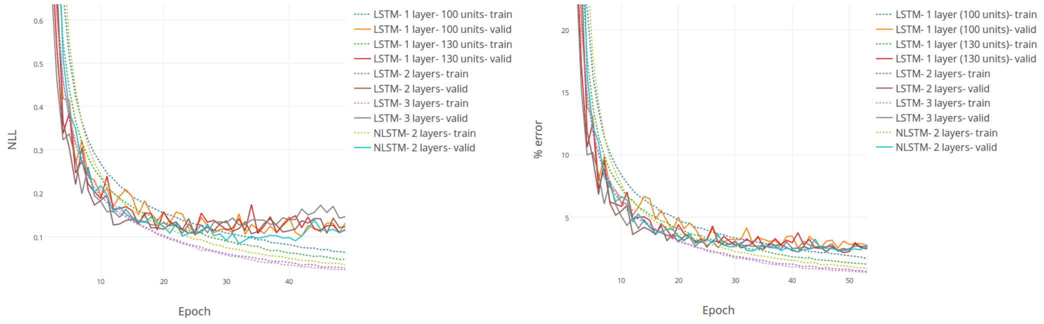


Figure 7: Plots of NLL (top) and percentage error (bottom) on the MNIST glimpses' train and validation sets (versus the epoch)

In the MNIST Glimpses task, introduced in Ba et al. (2016), each 28x28 image (with the pixel values normalized to the range [0, 1]) is split into 4 quadrants. Glimpses of each quadrant (in the form of alternate rows and columns), followed by the entire quadrant are then fed sequentially into the model (with 20 elements in the sequence, each element comprising of 49 pixels). The model then predicts which integer the input represented.

The hyperparameter set we use is similar to that chosen in the *pMNIST* task in Krueger et al. (2016): we train all models with an RMS Prop optimizer Tieleman and Hinton (2012) with a learning rate of 0.001 for 150 epochs (note that here, we use the more commonly used decay rate of 0.9 instead of 0.5 used there), and clip the gradients to a maximum norm of 1. As in the aforementioned *pMNIST* task, we use a 100 cell single layer LSTM baseline, along with 130 cell single layer, 75 cell two-layered stacked and 75 cell three-layered stacked LSTM baseline, and compare these baselines with a 75 cell Nested LSTM.

The Nested LSTM outperforms the (stacked) LSTM baselines in terms of both NLL and error percentage, both on the validation and test datasets. In particular, it reduces the validation error by

4.3% when compared to the next best performing model (to 1.77%, down from 1.85% for a 2-layered stacked LSTM), and the validation NLL by almost *17%* (down to 0.0836 from 0.1007 in the case of a single-layered LSTM).

4.5. text8

Model	n	cell size	Params	Valid	Test
LSTM	1	2000	16.28M	1.399	1.482
LSTM	1	2100	17.93M	1.396	1.480
LSTM	2	1200	17.45M	1.385	1.466
LSTM	3	950	18.19M	1.389	1.471
NLSTM	2	1200	17.45M	**1.363**	**1.445**

Table 4: BPC for the Nested LSTM versus various baselines on the text8 task

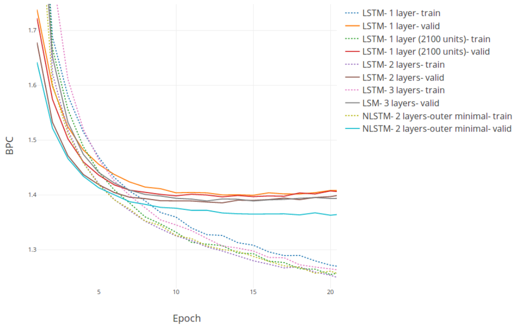


Figure 8: BPC vs epoch curves for character-level prediction on text8's train and validation sets

The text8 dataset comprises of the first 100MB of a cleaned-up version of enwik9 (which is comprised of text from Wikipedia) (Mahoney, 2011).

As in our earlier experiments, we keep our hyperparameters identical to Krueger et al. (2016); Cooijmans et al. (2016), except that we do not use any normalization, and that we train on non-overlapping sequences: we use a learning rate of 0.001, batch size of 128, sequence length of 180, a gradient clipping threshold of 1, with an adam optimizer (Kingma and Ba, 2014). Each model is trained for 40 epochs. Our baselines include 2000 and 2100 celled single layered LSTMs, a 1200 celled two-layered stacked LSTM and a 950 celled three-layered stacked LSTM baseline, pitted against a 1200 celled Nested LSTM.

Here too, we observe that our model outperforms the closest baseline (a 2-layered stacked LSTM) on both the valid (1.363 vs 1.385, respectively) and test (1.445 vs 1.466, respectively) sets. This indicates both that the proposed Nested LSTM is robust to different model sizes (as shown by the improvement it affords in the Chinese Poem generation task, where a relatively very small model was used), and that larger models trained on large datasets benefit from a nested architecture.

5. Conclusions

Nested LSTMs (NLSTM) are a simple extension of the LSTM model that add depth via nesting, as opposed to via stacking. The inner memory cells of an NLSTM form an internal memory, which is only accessible to other computational elements via the outer memory cells, implementing a form of temporal hierarchy. NLSTMs outperform stacked LSTMs with similar numbers of parameters in our experiments, and are more well defined in terms of the hierarchy their cell activations form vis-à-vis stacked LSTMs. Thus, NLSTMs represent an extremely promising alternative to stacked models.

ACKNOWLEDGMENTS

We thank Christopher Beckham for help with early experiments, and Tegan Maharaj and Nicolas Ballas for helpful discussions. We thank the developers of Theano (Theano Development Team, 2016), Fuel, and Blocks (van Merriënboer et al., 2015). We acknowledge the computing resources provided by ComputeCanada and CalculQuebec.

References

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

Kai Arulkumaran, Nat Dilokthanakul, Murray Shanahan, and Anil Anthony Bharath. Classifying options for deep reinforcement learning. *CoRR*, abs/1604.08153, 2016. URL http://arxiv.org/abs/1604.08153.

J. Ba, G. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using Fast Weights to Attend to the Recent Past. *ArXiv e-prints*, October 2016.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pages 4331–4339, 2016.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *CoRR*, abs/1609.05140, 2016. URL http://arxiv.org/abs/1609.05140.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

Emma Brunskill and Lihong Li. Pac-inspired option discovery in lifelong reinforcement learning. In *ICML*, pages 316–324, 2014.

Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983*, 2017.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078*, 2014.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Junyoung Chung, Caglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *CoRR, abs/1502.02367*, 2015.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *CoRR*, abs/1609.01704, 2016. URL http://arxiv.org/abs/1609.01704.

Tim Cooijmans, Nicolas Ballas, César Laurent, Caglar Gulcehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

Ivo Danihelka, Greg Wayne, Benigno Uria, Nal Kalchbrenner, and Alex Graves. Associative long short-term memory. *arXiv preprint arXiv:1602.03032*, 2016.

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 1999.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

A. Graves. Generating Sequences With Recurrent Neural Networks. *ArXiv e-prints*, August 2013.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. *Multi-dimensional Recurrent Neural Networks*, pages 549–558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74690-4. doi: 10.1007/978-3-540-74690-4_56. URL http://dx.doi.org/10.1007/978-3-540-74690-4_56.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *CoRR*, abs/1506.02516, 2015. URL http://arxiv.org/abs/1506.02516.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015. URL http://arxiv.org/abs/1503.04069.

Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

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

Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems*. 1996.

Geoffrey Hinton. Neural networks for machine learning coursera video lectures - geoffrey hinton. 2012. URL https://www.coursera.org/course/neuralnets.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *CoRR*, abs/1503.01007, 2015. URL http://arxiv.org/abs/1503.01007.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. *Journal of Machine Learning Research*, 2015.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. Grid long short-term memory. *CoRR*, abs/1507.01526, 2015. URL http://arxiv.org/abs/1507.01526.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

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

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

David Krueger and Roland Memisevic. Regularizing rnns by stabilizing activations. *arXiv preprint arXiv:1511.08400*, 2015.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron C. Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *CoRR*, abs/1606.01305, 2016. URL http://arxiv.org/abs/1606.01305.

Matt Mahoney. About the test data, 2011. URL http://mattmahoney.net/dc/textdata.

Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *CoRR*, abs/1312.6026, 2013. URL http://arxiv.org/abs/1312.6026.

Kamil Rocki. Recurrent memory array structures. *CoRR*, abs/1607.03085, 2016. URL http://arxiv.org/abs/1607.03085.

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

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 212–223. Springer, 2002.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4:2, 2012.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and fuel: Frameworks for deep learning. *CoRR*, abs/1506.00619, 2015.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL http://dx.doi.org/10.1007/BF00992696.

Aaron Courville Yoshua Bengio, Nicholas Léonard. Estimating or propagating gradients through stochastic neurons. *CoRR*, abs/1305.2982, 2013. URL http://arxiv.org/abs/1305.2982.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *CoRR*, abs/1505.00521, 2015. URL http://arxiv.org/abs/1505.00521.

Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. *arXiv preprint arXiv:1602.08210*, 2016.

Xingxing Zhang and Mirella Lapata. Chinese poetry generation with recurrent neural networks. In *EMNLP*, pages 670–680, 2014.