Grow and Prune Compact, Fast, and Accurate LSTMs

Xiaoliang Dai[*]
Princeton University
xdai@princeton.edu

Hongxu Yin[*]
Princeton University
hongxuy@princeton.edu

Niraj K. Jha
Princeton University
jha@princeton.edu

Abstract

Long short-term memory (LSTM) has been widely used for sequential data modeling. Researchers have increased LSTM depth by stacking LSTM cells to improve performance. This incurs model redundancy, increases run-time delay, and makes the LSTMs more prone to overfitting. To address these problems, we propose a hidden-layer LSTM (H-LSTM) that adds hidden layers to LSTM's original one-level non-linear control gates. H-LSTM increases accuracy while employing fewer external stacked layers, thus reducing the number of parameters and run-time latency significantly. We employ grow-and-prune (GP) training to iteratively adjust the hidden layers through gradient-based growth and magnitude-based pruning of connections. This learns both the weights and the compact architecture of H-LSTM control gates. We have GP-trained H-LSTMs for image captioning and speech recognition applications. For the NeuralTalk architecture on the MSCOCO dataset, our three models reduce the number of parameters by 38.7× [floating-point operations (FLOPs) by 45.5×], run-time latency by 4.5×, and improve the CIDEr score by 2.6. For the DeepSpeech2 architecture on the AN4 dataset, our two models reduce the number of parameters by 19.4× (FLOPs by 23.5×), run-time latency by 15.7%, and the word error rate from 12.9% to 8.7%. Thus, GP-trained H-LSTMs can be seen to be compact, fast, and accurate.

1 Introduction

Recurrent neural networks (RNNs) have been ubiquitously employed for sequential data modeling because of their ability to carry information through recurrent cycles. Long short-term memory (LSTM) is a special type of RNN that uses control gates and cell states to alleviate the exploding and vanishing gradient problems [1]. LSTMs are adept at modeling long-term and short-term dependencies. They deliver state-of-the-art performance for a wide variety of applications, such as speech recognition [2], image captioning [3], and neural machine translation [4].

Researchers have kept increasing the model depth and size to improve the performance of LSTMs. For example, the DeepSpeech2 architecture [2], which has been used for speech recognition, is more than 2× deeper and 10× larger than the initial DeepSpeech architecture proposed in [5]. However, large LSTM models may suffer from three problems. First, deployment of a large LSTM model consumes substantial storage, memory bandwidth, and computational resources. Such demands may be too excessive for mobile phones and embedded devices. Second, large LSTMs are prone to overfitting but hard to regularize [6]. Employing standard regularization methods that are used for feed-forward neural networks (NNs), such as dropout, in an LSTM cell is challenging [7, 8]. Third, the increasingly stringent run-time latency constraints in real-time applications make large LSTMs, which incur high latency, inapplicable in these scenarios. All these problems pose a significant design challenge in obtaining compact, fast, and accurate LSTMs.

[*] indicates equal contribution. This work was supported by NSF Grant No. CNS-1617640.

In this work, we tackle these design challenges simultaneously by combining two novelties. We first propose a hidden-layer LSTM (H-LSTM) that introduces multi-level abstraction in the LSTM control gates by adding several hidden layers. Then, we describe a grow-and-prune (GP) training method that combines gradient-based growth [9] and magnitude-based pruning [10] techniques to learn the weights and derive compact architectures for the control gates. This yields inference models that outperform the baselines from all three targeted design perspectives.

2 Related work

Going deeper is a common strategy for improving LSTM performance. The conventional approach of stacking LSTM cells has shown significant performance improvements on a wide range of applications, such as speech recognition and machine translation [11–13]. The recently proposed skipped connection technique has made it possible to train very deeply stacked LSTMs. This leads to high-performance architectures, such as residual LSTM [14] and highway LSTM [15].

Stacking LSTMs improves accuracy but incurs substantial computation and storage costs. Numerous recent approaches try to shrink the size of large NNs. A popular direction is to simplify the matrix representation. Typical techniques include matrix factorization [16], low rank approximation [17], and basis filter set reduction [18]. Another direction focuses on efficient storage and representation of weights. Various techniques, such as weight sharing within Toeplitz matrices [19], weight tying through effective hashing [20], and appropriate weight quantization [21–23], can greatly reduce model size, in some cases at the expense of a slight performance degradation.

Network pruning has emerged as another popular approach for LSTM compression. Han et al. show that pruning can significantly cut down on the size of deep convolutional neural networks (CNNs) and LSTMs [10, 24, 25]. Moreover, Yu et al. show that post-pruning sparsity in weight matrices can even improve speech recognition accuracy [26]. Narang el al. incorporate pruning in the training process and compress the LSTM model size by approximately $10\times$ while reducing training time significantly [27].

Apart from size reduction, run-time latency reduction has also attracted an increasing amount of research attention. A recent work by Zen et al. uses unidirectional LSTMs with a recurrent output layer to reduce run-time latency [28]. Amodei et al. propose a more efficient beam search strategy to speed up inference with only a minor accuracy loss [2]. Narang et al. exploit hardware-driven cuSPARSE libraries on a TitanX Maxwell GPU to speed up post-pruning sparse LSTMs by $1.16\times$ to $6.80\times$ [27].

3 Methodology

In this section, we explain our LSTM synthesis methodology that is based on H-LSTM cells and GP training. We first describe the H-LSTM structure, after which we illustrate GP training in detail.

3.1 Hidden-layer LSTM

Recent years have witnessed the impact of increasing NN depth on its performance. A deep architecture allows an NN to capture low/mid/high-level features through a multi-level abstraction. However, since a conventional LSTM employs fixed single-layer non-linearity for gate controls, the current standard approach for increasing model depth is through stacking several LSTM cells externally. In this work, we argue for a different approach that increases depth within LSTM cells. We propose an H-LSTM whose control gates are enhanced by adding hidden layers. We show that stacking fewer H-LSTM cells can achieve higher accuracy with fewer parameters and smaller run-time latency than conventionally stacked LSTM cells.

We show the schematic diagram of an H-LSTM in Fig. 1. The internal computation flow is governed by Eq. (1), where $\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$, $\mathbf{g}_t$, $\mathbf{x}_t$, $\mathbf{h}_t$, and $\mathbf{c}_t$ refer to the forget gate, input gate, output gate, vector for cell updates, input, hidden state, and cell state at step t, respectively; $\mathbf{h}_{t-1}$ and $\mathbf{c}_{t-1}$ refer to the hidden and cell states at step $t-1$; DNN, H, $\mathbf{W}$, $\mathbf{b}$, σ, and $\otimes$ refer to the deep neural network (DNN) gates, hidden layers (each performs a linear transformation followed by the activation function), weight matrix, bias, $sigmoid$ function, and element-wise multiplication, respectively; * indicates zero or more H layers for the DNN gate.

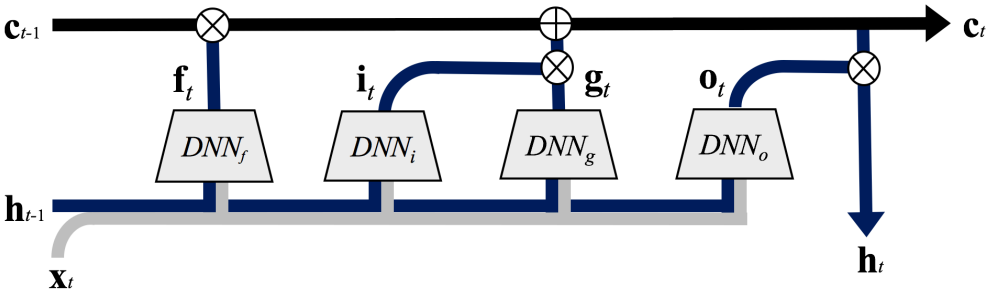

Figure 1: Schematic diagram of H-LSTM.

$$\begin{pmatrix} \mathbf{f}_t \\ \mathbf{i}_t \\ \mathbf{o}_t \\ \mathbf{g}_t \end{pmatrix} = \begin{pmatrix} DNN_f([\mathbf{x}_t, \mathbf{h}_{t-1}]) \\ DNN_i([\mathbf{x}_t, \mathbf{h}_{t-1}]) \\ DNN_o([\mathbf{x}_t, \mathbf{h}_{t-1}]) \\ DNN_g([\mathbf{x}_t, \mathbf{h}_{t-1}]) \end{pmatrix} = \begin{pmatrix} \sigma(\mathbf{W}_f H^*([\mathbf{x}_t, \mathbf{h}_{t-1}]) + \mathbf{b}_f) \\ \sigma(\mathbf{W}_i H^*([\mathbf{x}_t, \mathbf{h}_{t-1}]) + \mathbf{b}_i) \\ \sigma(\mathbf{W}_o H^*([\mathbf{x}_t, \mathbf{h}_{t-1}]) + \mathbf{b}_o) \\ tanh(\mathbf{W}_g H^*([\mathbf{x}_t, \mathbf{h}_{t-1}]) + \mathbf{b}_g) \end{pmatrix} \tag{1}$$

$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \mathbf{g}_t$$

$$\mathbf{h}_t = \mathbf{o}_t \otimes tanh(\mathbf{c}_t)$$

The introduction of DNN gates provides three major benefits to an H-LSTM:

1. **Strengthened control**: Hidden layers in DNN gates enhance gate control through multi-level abstraction. This makes an H-LSTM more capable and intelligent, and alleviates its reliance on external stacking. Consequently, an H-LSTM can achieve comparable or even improved accuracy with fewer external stacked layers relative to a conventional LSTM, leading to higher compactness.
2. **Easy regularization**: The conventional approach only uses dropout in the input/output layers and recurrent connections in the LSTMs. In our case, it becomes possible to apply dropout even to all control gates within an LSTM cell. This reduces overfitting and leads to better generalization.
3. **Flexible gates**: Unlike the fixed but specially-crafted gate control functions in LSTMs, DNN gates in an H-LSTM offer a wide range of choices for internal activation functions, such as the ReLU. This may provide additional benefits to the model. For example, networks typically learn faster with ReLUs [7]. They can also take advantage of ReLU's zero outputs for FLOPs reduction.

3.2 Grow-and-Prune Training

Conventional training based on back propagation on fully-connected NNs yields over-parameterized models. Han et al. have successfully implemented pruning to drastically reduce the size of large CNNs and LSTMs [10, 24]. The pruning phase is complemented with a brain-inspired growth phase for large CNNs in [9]. The network growth phase allows a CNN to grow neurons, connections, and feature maps, as necessary, during training. Thus, it enables automated search in the architecture space. It has been shown that a sequential combination of growth and pruning can yield additional compression on CNNs relative to pruning-only methods (e.g., 1.7× for AlexNet and 2.0× for VGG-16 on top of the pruning-only methods) [9]. In this work, we extend GP training to LSTMs.

We illustrate the GP training flow in Fig. 2. It starts from a randomly initialized sparse seed architecture. The seed architecture contains a very limited fraction of connections to facilitate initial gradient back-propagation. The remaining connections in the matrices are dormant and masked to zero. The flow ensures that all neurons in the network are connected. During training, it first grows connections based on the gradient information. Then, it prunes away redundant connections for compactness, based on their magnitudes. Finally, GP training rests at an accurate, yet compact, inference model. We explain the details of each phase next.

GP training adopts the following policies:

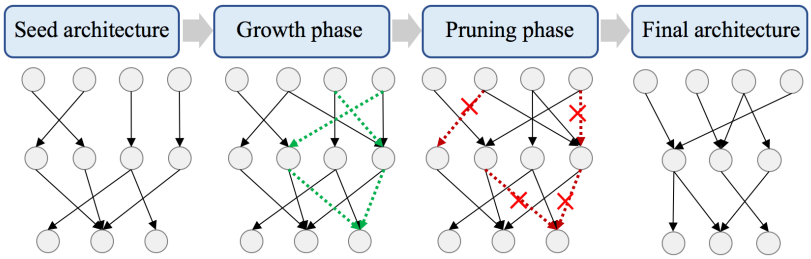

Figure 2: An illustration of GP training.

Growth policy: Activate a dormant w in $\mathbf{W}$ iff $|w.grad|$ is larger than the $(100\alpha)^{th}$ percentile of all elements in $|\mathbf{W}.grad|$.

Pruning policy: Remove a w iff $|w|$ is smaller than the $(100\beta)^{th}$ percentile of all elements in $|\mathbf{W}|$.

w, $\mathbf{W}$, $.grad$, α, and β refer to the weight of a single connection, weights of all connections within one layer, operation to extract the gradient, growth ratio, and pruning ratio, respectively.

In the growth phase, the main objective is to locate the most effective dormant connections to reduce the value of loss function L. We first evaluate $\partial L/\partial w$ for each dormant connection w based on its average gradient over the entire training set. Then, we activate each dormant connection whose gradient magnitude $|w.grad| = |\partial L/\partial w|$ surpasses the $(100\alpha)^{th}$ percentile of the gradient magnitudes of its corresponding weight matrix. This rule caters to dormant connections iff they provide most efficiency in L reduction. Growth can also help avoid local minima, as observed by Han et al. in their dense-sparse-dense training algorithm to improve accuracy [29].

We prune the network after the growth phase. Pruning of insignificant weights is an iterative process. In each iteration, we first prune away insignificant weights whose magnitudes are smaller than the $(100\beta)^{th}$ percentile within their respective layers. We prune a neuron if all its input (or output) connections are pruned away. We then retrain the NN after weight pruning to recover its performance before starting the next pruning iteration. The pruning phase terminates when retraining cannot achieve a pre-defined accuracy threshold. GP training finalizes a model based on the last complete iteration.

4 Experimental Results

We present our experimental results for image captioning and speech recognition benchmarks next. We implement our experiments using PyTorch [30] on Nvidia GTX 1060 and Tesla P100 GPUs.

4.1 NeuralTalk for Image Captioning

We first show the effectiveness of our proposed methodology on image captioning.

Architecture: We experiment with the NeuralTalk architecture [3, 31] that uses the last hidden layer of a pre-trained CNN image encoder as an input to a recurrent decoder for sentence generation. The recurrent decoder applies a beam search technique for sentence generation. A beam size of k indicates that at step t, the decoder considers the set of k best sentences obtained so far as candidates to generate sentences in step $t+1$, and keeps the best k results [3, 31, 32]. In our experiments, we use VGG-16 [33] as the CNN encoder, same as in [3, 31]. We then use H-LSTM and LSTM cells with the same width of 512 for the recurrent decoder and compare their performance. We use $Beam = 2$ as the default beam size.

Dataset: We report results on the MSCOCO dataset [34]. It contains 123287 images of size 256×256×3, along with five reference sentences per image. We use the publicly available split [31], which has 113287, 5000, and 5000 images in the training, validation, and test sets, respectively.

Training: We use the Adam optimizer [35] for this experiment. We use a batch size of 64 for training. We initialize the learning rate at 3×10^{-4}. In the first 90 epochs, we fix the weights of the CNN and train the LSTM decoder only. We decay the learning rate by 0.8 every six epochs in this phase. After

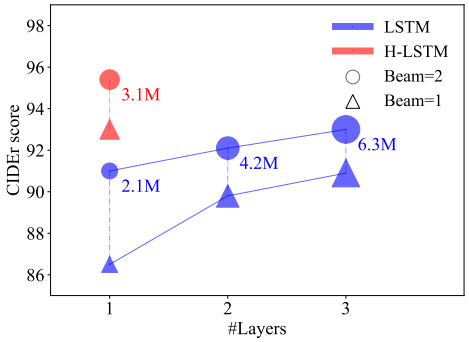

Figure 3: Comparison of NeuralTalk CIDEr for the LSTM and H-LSTM cells. Number and area indicate size.

Table 1: Cell comparison for the NeuralTalk architecture on the MSCOCO dataset.

Model	CIDEr	#Param	Latency
Single LSTM	91.0	2.1M	21ms
Stacked 2-layer LSTMs	92.1	4.2M	29ms
Stacked 3-layer LSTMs	92.8	6.3M	36ms
H-LSTM (Beam=2)	**95.4**	**3.1M**	**24ms**
H-LSTM (Beam=1)	**93.0**	**3.1M**	**8ms**

90 epochs, we start to fine-tune both the CNN and LSTM at a fixed 1×10^{-6} learning rate. We use a dropout ratio of 0.2 for the hidden layers in the H-LSTM. We also use a dropout ratio of 0.5 for the input and output layers of the LSTM, same as in [6]. We use CIDEr score [36] as our evaluation criterion.

4.1.1 Cell Comparison

We first compare the performance of a fully-connected H-LSTM with a fully-connected LSTM to show the benefits emanating from using the H-LSTM cell alone.

The NeuralTalk architecture with a single LSTM achieves a 91.0 CIDEr score [3]. We also experiment with stacked 2-layer and 3-layer LSTMs, which achieve 92.1 and 92.8 CIDEr scores, respectively. We next train a single H-LSTM and compare the results in Fig. 3 and Table 1. Our single H-LSTM achieves a CIDEr score of 95.4, which is 4.4, 3.3, 2.6 higher than the single LSTM, stacked 2-layer LSTM, and stacked 3-layer LSTM, respectively.

H-LSTM can also reduce run-time latency. Even with $Beam = 1$, a single H-LSTM achieves a higher accuracy than the three LSTM baselines. Reducing the beam size leads to run-time latency reduction. H-LSTM is 4.5×, 3.6×, 2.6× faster than the stacked 3-layer LSTM, stacked 2-layer LSTM, and single LSTM, respectively, while providing higher accuracy.

4.1.2 Training Comparison

Next, we implement both network pruning and our GP training to synthesize compact inference models for an H-LSTM ($Beam = 2$). The seed architecture for GP training has a sparsity of 50%. In the growth phase, we use a 0.8 growth ratio in the first five epochs. We summarize the results in Table 2, where CR refer to the compression ratio relative to a fully-connected model. GP training provides an additional 1.40× improvement on CR compared with network pruning.

Table 2: Training algorithm comparison

Method	Cell	#Layers	CIDEr	#Param	CR	Improvement
Network pruning	H-LSTM	1	95.4	550K	5.7×	-
GP training	**H-LSTM**	**1**	**95.4**	**394K**	**8.0×**	**1.40×**

4.1.3 Inference Model Comparison

We list our GP-trained H-LSTM models in Table 3. Note that the accurate and fast models are the same network with different beam sizes. The compact model is obtained through further pruning of the accurate model. We choose the stacked 3-layer LSTM as our baseline due to its high accuracy. Our accurate, fast, and compact models demonstrate improvements in all aspects (accuracy, speed,

Table 3: Different inference models for the MSCOCO dataset

Model	#Layers	Cell	Beam	CIDEr	#Param	FLOPs	Latency
Single LSTM	1	LSTM	2	91.0	2.1M	4.2M	21ms
Stacked LSTMs	2	LSTM	2	92.1	4.2M	8.4M	29ms
Stacked LSTMs	3	LSTM	2	92.8	6.3M	12.6M	36ms
Ours: accurate	1	H-LSTM	2	95.4	394K	670K	24ms
Ours: fast	1	H-LSTM	1	93.0	394K	670K	8ms
Ours: compact	1	H-LSTM	2	93.3	163K	277K	24ms

and compactness), with a 2.6 higher CIDEr score, 4.5× speedup, and 38.7× fewer parameters, respectively.

4.2 DeepSpeech2 for Speech Recognition

We now consider another well-known application: speech recognition.

Architecture: We implement a bidirectional DeepSpeech2 architecture that employs stacked recurrent layers following convolutional layers for speech recognition [2]. We use Mel-frequency cepstral coefficients (MFCCs) as network inputs, extracted from raw speech data at a 16KHz sampling rate and 20ms feature extraction window. There are two CNN layers prior to the recurrent layers and one connectionist temporal classification layer for decoding [37] after the recurrent layers. The width of the hidden and cell states is 800, same as in [38, 39]. We also set the width of H-LSTM hidden layers to 800.

Dataset: We use the AN4 dataset [40] to evaluate the performance of our DeepSpeech2 architecture. It contains 948 training utterances and 130 testing utterances.

Training: We utilize a Nesterov SGD optimizer in our experiment. We initialize the learning rate to 3×10^{-4}, decayed per epoch by 0.99. We use a batch size of 16 for training. We use a dropout ratio of 0.2 for the hidden layers in the H-LSTM. We apply batch normalization between recurrent layers. We apply L2 regularization during training with a weight decay of 1×10^{-4}. We use word error rate (WER) as our evaluation criterion, same as in [38, 39, 41].

4.2.1 Cell Comparison

We first compare the performance of the fully-connected H-LSTM against the fully-connected LSTM and gate recurrent unit (GRU) to demonstrate the benefits provided by the H-LSTM cell alone. GRU uses reset and update gates for memory control and has fewer parameters than LSTM [42].

For the baseline, we train various DeepSpeech2 models containing a different number of stacked layers based on GRU and LSTM cells. The stacked 4-layer and 5-layer GRUs achieve a WER of 14.35% and 11.60%, respectively. The stacked 4-layer and 5-layer LSTMs achieve a WER of 13.99% and 10.56%, respectively.

We next train an H-LSTM to make a comparison. Since an H-LSTM is intrinsically deeper, we aim to achieve a similar accuracy with a smaller stack. We reach a WER of 12.44% and 8.92% with stacked 2-layer and 3-layer H-LSTMs, respectively.

We summarize the cell comparison results in Fig. 4 and Table 4, where all the sizes are normalized to the size of a single LSTM. We can see that H-LSTM can reduce WER by more than 1.5% with two fewer layers relative to LSTMs and GRUs, thus satisfying our initial design goal to stack fewer cells that are individually deeper. H-LSTM models contain fewer parameters for a given target WER, and can achieve lower WER for a given number of parameters.

4.2.2 Training Comparison

We next implement GP training to show its additional benefits on top of just performing network pruning. We select the stacked 3-layer H-LSTMs for this experiment due to its highest accuracy. For GP training, we initialize the seed architecture with a connection sparsity of 50%. We grow the networks for three epochs using a 0.9 growth ratio.

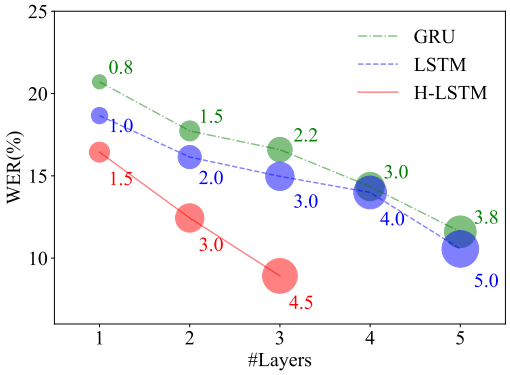

Figure 4: Comparison of DeepSpeech2 WERs for the GRU, LSTM, and H-LSTM cells. Number and area indicate relative size to one LSTM.

Table 4: Cell comparison for the DeepSpeech2 architecture on the AN4 dataset

Cell Type	#Layers	Size	WER
GRU	4	3.0	14.35%
LSTM	4	4.0	13.99%
H-LSTM	**2**	**3.0**	**12.44%**
GRU	5	3.8	11.64%
LSTM	5	5.0	10.56%
H-LSTM	**3**	**4.5**	**8.92%**

Table 5: Training algorithm comparison

Method	Cell	#Layers	WER	#Param	CR	Improvement
Network pruning	H-LSTM	3	10.49%	6.4M	6.8×	-
GP training	**H-LSTM**	**3**	**10.37%**	**2.6M**	**17.2×**	**2.44×**

For best compactness, we set an accuracy threshold for both GP training and the pruning-only process at 10.52% (lowest WER from relevant work [38, 39, 41]). We compare these two approaches in Table 5. Compared to network pruning, GP training can further boost the CR by 2.44× while improving the accuracy slightly. This is consistent with prior observations that pruning large CNNs potentially inherits certain redundancies from the original fully-connected model that the growth phase can alleviate [9].

4.2.3 Inference Model Comparison

We obtain two GP-trained models by varying the WER constraint during the pruning phase: an accurate model aimed at a higher accuracy (9.00% WER constraint), and a compact model aimed at extreme compactness (10.52% WER constraint).

We compare our results against prior work from the literature in Table 6. We select a stacked 5-layer LSTM [38] as our baseline. On top of the substantial parameter and FLOPs reductions, both the accurate and compact models also reduce the average run-time latency per instance from 691.4ms to 583.0ms (15.7% reduction) even without any sparse matrix library support.

Table 6: Different inference models for the AN4 dataset

Model	RNN Type	WER(%)	ΔWER(%)	#Param(M)	FLOPs(M)
Lin et al. [38]	LSTM	12.90	-	50.4 (1.0×)	100.8 (1.0×)
Alistarh et al. [41]	LSTM	18.85	+5.95	13.0 (3.9×)	26.0 (3.9×)
Sharen et al. [39]	GRU	10.52	-2.38	37.8 (1.3×)	75.7 (1.3×)
Ours: accurate	**H-LSTM**	**8.73**	**-4.17**	**22.5 (2.3×)**	**37.1 (2.9×)**
Ours: compact	**H-LSTM**	**10.37**	**-2.53**	**2.6 (19.4×)**	**4.3 (23.5×)**

The introduction of the ReLU activation function in DNN gates provides additional FLOPs reduction for the H-LSTM. This effect does not apply to LSTMs and GRUs that only use $tanh$ and $sigmoid$ gate control functions. At inference time, the average activation percentage of the ReLU outputs is 48.3% for forward-direction LSTMs, and 48.1% for backward-direction LSTMs. This further reduces the overall run-time FLOPs by 14.5%.

Table 7: GP-trained compact 3-layer H-LSTM DeepSpeech2 model at 10.37% WER

	Sparsity	Sparsity	Sparsity
Layers	Seed	Post-Growth	Post-Pruning
H-LSTM layer1	50.00%	38.35%	94.26%
H-LSTM layer2	50.00%	37.68%	94.20%
H-LSTM layer3	50.00%	37.86%	94.21%
Total	50.00%	37.96%	94.22%

The details of the final inference models are summarized in Table 7. The final sparsity of the compact model is as high as 94.22% due to the compounding effect of growth and pruning.

5 Discussions

We observe the importance of regularization in H-LSTM on its final performance. We summarize the comparison between fully-connected models with and without dropout for both applications in Table 8. By appropriately regularizing DNN gates, we improve the CIDEr score by 2.0 on NeuralTalk and reduce the WER from 9.88% to 8.92% on DeepSpeech2.

Table 8: Impact of dropout on H-LSTM

Architecture	Dropout	CIDEr	Architecture	Dropout	WER
NeuralTalk	N	93.4	DeepSpeech2	N	9.88%
NeuralTalk	Y	95.4	DeepSpeech2	Y	8.92%

Some real-time applications may emphasize stringent memory and delay constraints instead of accuracy. In this case, the deployment of stacked LSTMs may be infeasible. The extra parameters used in H-LSTM's hidden layers may also seem disadvantageous in this scenario. However, we next show that the extra parameters can be easily compensated by a reduced hidden layer width. We compare several models for image captioning in Table 9. If we reduce the width of the hidden layers and cell states in the H-LSTM from 512 to 320, we can easily arrive at a single-layer H-LSTM that dominates the conventional LSTM from all three design perspectives. Our observation coincides with prior experience with neural network training where slimmer but deeper NNs (in this case H-LSTM) normally exhibit better performance than shallower but wider NNs (in this case LSTM).

Table 9: H-LSTM with reduced width for further speedup and compactness

Cell	#Layers	Width	#Param	CIDEr	Latency	CIDEr	Latency
				Beam=2		Beam=1	
LSTM	1	512	2.1M	91.0	21ms	86.5	7ms
H-LSTM	1	512	3.1M	95.4	24ms	93.0	8ms
H-LSTM	1	320	1.5M	92.2	18ms	88.1	5ms

We also employ an activation function shift trick in our experiment. In the growth phase, we adopt a leaky ReLU (reverse slope of 0.01) as the activation function for H^* in Eq. (1). Leaky ReLU effectively alleviates the 'dying ReLU' phenomenon, in which a zero output of the ReLU neuron blocks it from any future gradient update. Then, we change all the activation functions from Leaky ReLU to ReLU while keeping the weights unchanged, retrain the network to recover performance, and continue to the pruning phase.

6 Conclusions

In this work, we combined H-LSTM and GP training to learn compact, fast, and accurate LSTMs. An H-LSTM adds hidden layers to control gates as opposed to conventional architectures that just employ a one-level nonlinearity. GP training combines gradient-based growth and magnitude-based pruning to ensure H-LSTM compactness. We GP-trained H-LSTMs for the image captioning and speech recognition applications. For the NeuralTalk architecture on the MSCOCO dataset, our models

reduced the number of parameters by 38.7× (FLOPs by 45.5×) and run-time latency by 4.5×, and improved the CIDEr score by 2.6. For the DeepSpeech2 architecture on the AN4 dataset, our models reduced the number of parameters by 19.4× (FLOPs by 23.5×), run-time latency by 15.7%, and WER from 12.9% to 8.7%.

References

[1] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen *et al.*, "Deep Speech 2 : End-to-End speech recognition in English and Mandarin," in *Proc. Int. Conf. Machine Learning*, vol. 48, 2016, pp. 173–182.

[3] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2015, pp. 3128–3137.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[5] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep Speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[6] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[8] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. Courville, and C. Pal, "Zoneout: Regularizing RNNs by randomly preserving hidden activations," *arXiv preprint arXiv:1606.01305*, 2016.

[9] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," *arXiv preprint arXiv:1711.02017*, 2017.

[10] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2017, pp. 75–84.

[11] A. Graves, N. Jaitly, and A.-R. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Proc. IEEE Workshop Automatic Speech Recognition and Understanding*, 2013, pp. 273–278.

[12] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, 2013, pp. 6645–6649.

[13] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," *arXiv preprint arXiv:1312.6026*, 2013.

[14] J. Kim, M. El-Khamy, and J. Lee, "Residual LSTM: Design of a deep recurrent architecture for distant speech recognition," *arXiv preprint arXiv:1701.03360*, 2017.

[15] Y. Zhang, G. Chen, D. Yu, K. Yaco, S. Khudanpur, and J. Glass, "Highway long short-term memory RNNs for distant speech recognition," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2016, pp. 5755–5759.

[16] M. Denil, B. Shakibi, L. Dinh, N. De Freitas *et al.*, "Predicting parameters in deep learning," in *Proc. Advances in Neural Information Processing Systems*, 2013, pp. 2148–2156.

[17] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Advances in Neural Information Processing Systems*, 2014, pp. 1269–1277.

[18] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv preprint arXiv:1405.3866*, 2014.

[19] Z. Lu, V. Sindhwani, and T. N. Sainath, "Learning compact recurrent neural networks," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 2016, pp. 5960–5964.

[20] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Machine Learning*, 2015, pp. 2285–2294.

[21] H. Zen, Y. Agiomyrgiannakis, N. Egberts, F. Henderson, and P. Szczepaniak, "Fast, compact, and high quality LSTM-RNN based statistical parametric speech synthesizers for mobile devices," *arXiv preprint arXiv:1606.06061*, 2016.

[22] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. NIPS Workshop Deep Learning and Unsupervised Feature Learning*, vol. 1, 2011, p. 4.

[23] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[24] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.

[25] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," *arXiv preprint arXiv:1804.03294*, 2018.

[26] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2012, pp. 4409–4412.

[27] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv preprint arXiv:1704.05119*, 2017.

[28] H. Zen and H. Sak, "Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2015, pp. 4470–4474.

[29] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, and W. J. Dally, "DSD: Regularizing deep neural networks with dense-sparse-dense training flow," *arXiv preprint arXiv:1607.04381*, 2016.

[30] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," *NIPS Workshop Autodiff*, 2017.

[31] A. Karpathy, "Image captioning in Torch," https://github.com/karpathy/neuraltalk2, 2016.

[32] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2015, pp. 3156–3164.

[33] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[34] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. European Conf. Computer Vision*, 2014, pp. 740–755.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[36] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, "CIDEr: Consensus-based image description evaluation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2015, pp. 4566–4575.

[37] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proc. Int. Conf. Machine Learning*, 2006, pp. 369–376.

[38] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," *arXiv preprint arXiv:1712.01887*, 2017.

[39] S. Naren, "Speech recognition using DeepSpeech2," https://github.com/SeanNaren/deepspeech.pytorch/releases, 2018.

[40] A. Acero, "Acoustical and environmental robustness in automatic speech recognition," in *Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 1990.

[41] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. Advances in Neural Information Processing Systems*, 2017, pp. 1709–1720.

[42] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.