LEARNING COMPACT RECURRENT NEURAL NETWORKS

Zhiyun Lu[*1], *Vikas Sindhwani*[2], *Tara N. Sainath*[2]

[1] University of Southern California, Los Angeles, CA, USA
[2] Google, Inc. New York, NY, USA
zhiyunlu@usc.edu, {sindhwani,tsainath}@google.com

ABSTRACT

Recurrent neural networks (RNNs), including long short-term memory (LSTM) RNNs, have produced state-of-the-art results on a variety of speech recognition tasks. However, these models are often too large in size for deployment on mobile devices with memory and latency constraints. In this work, we study mechanisms for learning compact RNNs and LSTMs via low-rank factorizations and parameter sharing schemes. Our goal is to investigate redundancies in recurrent architectures where compression can be admitted without losing performance. A hybrid strategy of using structured matrices in the bottom layers and shared low-rank factors on the top layers is found to be particularly effective, reducing the parameters of a standard LSTM by 75%, at a small cost of 0.3% increase in WER, on a 2,000-hr English Voice Search task.

1. INTRODUCTION

Recurrent neural network architectures have become very popular for automatic speech recognition (ASR) tasks in the past few years. Architectures such as recurrent neural networks (RNNs), long short-term memory networks (LSTMs) and convolutional, long-short term memory networks (CLDNNs) have produced state of the art results for many large vocabulary continuous speech recognition (LVCSR) tasks [1, 2, 3].

In order to fully exploit thousands of hours of training data for LVCSR tasks, the best performing neural network architectures are typically very large in size. Consequently, they require long training time and consume a significant number of floating point operations per prediction once deployed. These characteristics are further exacerbated in large, deep RNN architectures, which unroll the network for a sequential number of time frames (i.e., 10-20), and must compute the output from one time step before feeding into the next time step. This situation is at odds with the need to enable high-performance on-device speech recognition on storage and power constrained mobile phones for which compact, small-sized models are strongly preferred.

Numerous approaches have recently been proposed in the model compression literature to build compact neural network models with fast training and prediction speed. A popular technique is low-rank matrix factorization [2, 4, 5], which attempts to compress neural network layers by representing them as matrices with low-rank. This was shown to reduce parameters by 30-50% for DNNs with no loss in accuracy for LVCSR tasks. Other techniques include inducing zeros in the parameter matrices via sparsity regularizers [6]; storing weights in low fixed-precision formats [7, 8]; using specific param-

---

*The author performed the work as a summer intern at Google, NYC.

eter sharing mechanisms [9, 10], or training smaller models on soft outputs of larger models [11].

The dominant attention in this literature has so far been on reducing the size of fully connected and convolutional architectures (DNNs and CNNs). Given the importance of recurrent architectures in the speech community, the goal of this work is to explore compact architectures for deep RNNs and LSTMs. Several open questions immediately arise in this context: where precisely is the redundancy in recurrent architectures for speech recognition tasks? Which compact architectural variations retain the most performance? How should size constraints vary across layers, between recurrent and feedforward weights, and between different gates in a recurrent model? Our contributions in this paper are as follows:

- We are the first to undertake a systematic study of various new compact architectures for RNNs and LSTMs. Specifically, we compare the effectiveness of low-rank models [2, 4] and various parameter sharing schemes implemented using hashing [9] and structured matrices [10]. Note that sparsity promoting regularizers and the use of low-precision storage formats are complementary to our study and could yield even more compact models.

- Our investigation reveals the following: (i) for aggressive reduction of parameters in the bottom layers, Toeplitz-like structured matrices [10] outperform Hashing based schemes and low-rank factorizations, (ii) shared low-rank factors [2] are very effective for parameter reduction across the network, (iii) a particularly effective hybrid strategy for building compact LSTMs - using Toeplitz-like structured matrices [10] in bottom layers and projection layers involving shared low-rank factors in the upper layers – can save 75% parameters with 0.3% increase in Word Error Rate (WER), compared to a full LSTM, (iv) LSTMs are relatively insensitive to whether compression is applied to recurrent or non-recurrent weights, and similarly so for input/output/forget gates; on the other hand, the cell state is critical to preserve for better performance.

2. RECURRENT NEURAL NETS

We start by setting some notations and by giving an overview of the basic RNN and LSTM architectures.

RNN: An RNN maps an input sequence $x = (x_1, \ldots, x_T)$ to output sequence $y = (y_1, \ldots, y_T)$. At each time step $t \in T$, the RNN is modeled with the following equations for the hidden unit h_t^l in each layer $l \in \{1, \ldots, L\}$ and the output y_t, where $h_t^0 = x_t$.

$$h_t^l = \sigma[W^l h_t^{l-1} + U^l h_{t-1}^l + b^l] \qquad l = 1, \ldots, L$$
$$y_t = \text{softmax}[W^{L+1} h_t^L + b^{L+1}] \tag{1}$$

We will refer to Equation 1 as RNN full model. We call U^l as recurrent weights, and W^l as feedforward or non-recurrent weights.

LSTM: A LSTM is an alternative model that has been used to address vanishing/exploding gradient issues with RNN, and to model longer-term temporal dependencies [12]. For each time step $t \in T$ and layer $l \in \{1, \ldots, L\}$, the LSTM sequence to sequence mapping, which includes the forget gate [13] and peephole connection [14], is described as follows (as above $h_t^0 = x_t$).

$$i_t^l = \sigma[W_i^l h_t^{l-1} + U_i^l h_{t-1}^l + b_i^l + D(v_i^l)c_{t-1}^l] \qquad \text{input gate} \quad (2)$$

$$f_t^l = \sigma[W_f^l h_t^{l-1} + U_f^l h_{t-1}^l + b_f^l + D(v_f^l)c_{t-1}^l] \qquad \text{forget gate} \quad (3)$$

$$c_t^l = f_t^l \cdot c_{t-1}^l + i_t^l \tanh[W_c^l h_t^{l-1} + U_c^l h_{t-1}^l + b_c^l] \quad \text{cell state} \quad (4)$$

$$o_t^l = \sigma[W_o^l h_t^{l-1} + U_o^l h_{t-1}^l + b_o^l + D(v_o^l)c_t^l] \qquad \text{output gate} \quad (5)$$

$$h_t^l = o_t^l \cdot \tanh[c_t^l] \qquad\qquad\qquad\qquad\qquad \text{cell output} \quad (6)$$

The output of the network is given by,

$$y_t = \text{softmax}[W^{L+1} h_t^L + b^{L+1}].$$

The peephole connection $D(v_i^l)$ is a diagonal matrix with v_i^l as its diagonal. We will refer to Equations 2, 3, 4, 5 as input gate, forget gate, cell state and output gate respectively.

3. PARAMETER REDUCTION SCHEMES

We now describe various parameter reduction methods explored in this work, to make the recurrent architectures described in Section 2 more compact. For sake of simplicity, we will refer to the generic matrix we want to compress as W, which is of size $m \times n$. The goal of the compression schemes is to reduce the number of parameters of W, namely mn.

3.1. Low Rank Factorization

In a low-rank compression scheme, we assume that matrix W has rank r. Then there exists [15] a factorization $W = W_a \times W_b$ where W_a is a full-rank matrix of size $m \times r$ and W_b is a full-rank matrix of size $r \times n$. Thus, we replace matrix W by matrices W_a and W_b which is equivalent to replacing a fully connected layer with a linear bottleneck layer. Notice there is no non-linearity (i.e. sigmoid) between matrices W_a and W_b. A low-rank decomposition of Equation 1 for recurrent and feedforward matrices U^l and W^l, for layer $l \in \{1, \ldots, L\}$, is shown in Equation 7.

$$h_t^l = \sigma[W_a^l W_b^l h_t^{l-1} + U_a^l U_b^l h_{t-1}^l + b^l] \qquad (7)$$

We can reduce the number of parameters of the system so long as the number of parameters in W_a (i.e., mr) and W_b (i.e., rn) is less than W (i.e., mn). Low-rank matrix factorization was first explored for speech in [4], where it was found DNN models could be reduced by 30-50% with no loss in accuracy. Furthermore, in the context of DNNs, [5] explored computing the singular-value-decomposition (SVD) of a full matrix to learn the two smaller low-rank matrices.

3.2. Sharing Low-rank across Layers: Projection Model

More recently, [2] shared the low-rank factor across layers of the recurrent architecture. To be more specific, we require $W_b^l = U_b^{l-1}$ when we apply low rank to both U^{l-1} and W^l as in Equation 7. We can rewrite Equation 1 for RNNs as

$$h_t^l = \sigma[W_a^l m_t^{l-1} + U_a^l m_{t-1}^l + b^l]$$

$$m_t^l = U_b^l h_t^l \qquad\qquad\qquad l = 1, \ldots, L \quad (8)$$

$$y_t = \text{softmax}[W^{L+1} m_t^L + b^{L+1}] \qquad\qquad (9)$$

where $m_t^0 = x_t$. m_t^l, the low rank output in Equation 8, can be seen as a linear projection of the original hidden layer, which is shared across layers. We refer to this as a projection compression scheme.

The projection model is compact with weight sharing. Besides, since the projection weight U_b^{l-1} is shared between U^{l-1} and W^l, its gradient also receives error signals from both factors. The error component of U_b^{l-1} coming from W^l is closer to the output, compared to that of U^{l-1}, which makes the learning of recurrent connection easier in the projection model (Equation 9) compared to the full model (Equation 1). Therefore, the projection model regularizes the full model with fewer number of parameters and facilitates learning through weight sharing.

Similarly for LSTM, we can modify Equation 6 to be

$$h_t^l = P^l o_t^l \cdot \tanh[c_t^l] \qquad\qquad \text{projection node} \quad (10)$$

which projects the cell c_t^l down to h_t^l of lower dimensionality, with P^l having a similar interpretation to U_b^l in Equation 8. Then the projection node h_t^l will feed forward to next layer and recurrently to the next time step of the same layer, for all gates and cell activations in Equation 2-5. We refer the reader to [2] for detailed equations regarding the LSTM projection model.

3.3. HashedNets

The HashedNets scheme was recently proposed in [9] to reduce memory consumption of DNN layers for computer vision tasks. Here, we assume that the matrix W has only k unique parameter values, instead of mn. The connections in W are randomly grouped together and hashed into one bucket of length-k parameter vector v, $W_{ij} = v_{h(i,j)}$, where $h(i,j) : \mathbb{N} \times \mathbb{N} \to \{1, \ldots, k\}$ is a predefined hashing function. v_m is shared among all entries of W_{ij} where $h(i,j) = m$ in both feedforward and back-propagation [9].

To be consistent with other methods, we use pseudo rank r for HashedNets to refer to $k = 2nr$ number of parameters. We will apply this hashing trick to the U^l and W^l matrices for RNNs.

3.4. Toeplitz-like Structured Matrices

Recently, [10] proposed a new family of parameter sharing schemes for small-footprint deep learning based on structured matrices characterized by the notion of *displacement operators* [16]. Unlike HashedNets where weights are randomly grouped, parameter sharing mechanisms in structured matrices are highly specific and deterministic. The structure can be exploited for fast matrix-vector multiplication (forward passes) and also gradient computations during back-propagation typically using Fast Fourier Transform like operations. To get a flavor of this approach, consider Toeplitz matrices where parameters are tied along diagonals.

$$\begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-(n-1)} \\ t_1 & t_0 & \cdots & \vdots \\ \vdots & \vdots & \vdots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{bmatrix}$$

It is known that $n \times n$ Toeplitz matrices admit $O(n \log n)$ time to compute matrix-vector products, which are essentially equivalent to performing linear convolutions. Toeplitz matrices also have the property that via certain shift and scale operations as implemented by specific displacement operators, they can be linearly transformed into matrices of rank less than or equal to 2. Thus, the so called *displacement rank* of all Toeplitz matrices is up to 2. [10] propose learning parameter matrices that are generalizations of the Toeplitz structure by allowing the displacement rank r to be higher. This class

of matrices are called *Toeplitz-like* and they include products and inverses of Toeplitz matrices, and their linear combinations, which can be interpreted as composition of convolutions and deconvolutions. These matrices can be parameterized via a sum over products of r Circulant and Skew-circulant matrices. The displacement rank r serves as a knob on modeling capacity. High displacement rank matrices are increasingly unstructured. [10] show that on mobile speech recognition problems, such transforms are highly effective for learning compact Toeplitz-like layers compared to fully connected DNNs.

With displacement rank r, there are $2nr$ free parameters in the Toeplitz-like structured matrix. We will apply the Toeplitz-like transform to U^l, W^l in RNNs, and $W_i^l, W_f^l, W_c^l, W_o^l, U_i^l, U_f^l, U_c^l, U_o^l$ to LSTMs in our experiment.

Summary: Table 1 gives a summary of different methods with its number of parameters as a function of an appropriate notion of rank. For simplicity we assume $m = n$. For projection, the number of parameters is averaged across layers where low-rank factors are shared.

method	knob	# of params
Low-rank	rank r	$2nr$
Projection	rank r	$\frac{3}{2}nr$
HashedNets	pseudo-rank r	$2nr$
Toeplitz-like	displacement rank r	$2nr$

Table 1: comparison of compression methods

4. EXPERIMENTAL DETAILS

We report two sets of experiments: with RNNs on a medium-sized noisy training set of 300 thousand English-spoken utterances (300 hours), and with LSTMs on a larger training set of 3 million utterances (2,000 hours). These data sets are created by artificially corrupting clean utterances using a room simulator, adding varying degrees of noise and reverberation such that the overall SNR is between 5dB and 30dB. The noise sources are from YouTube and daily life noisy environmental recordings. All training sets are anonymized and hand-transcribed, and are representative of Google's voice search traffic. The training sets is randomly split into 90% for model training, and 10% for heldout used to evaluate frame accuracy. WER is reported on a noisy test set containing 30,000 utterances (over 20 hours).

The input feature for all models are 40-dimensional log-mel filterbank features, computed every 10ms. All recurrent layers are initialized with uniform random weights between -0.02 to 0.02. The RNNs and LSTMs are unrolled for 20 time steps for training with truncated backpropagation through time (BPTT). In addition, the output state label is delayed by 5 frames, similar to [3].

All neural networks are trained with the cross-entropy criterion, using the asynchronous stochastic gradient descent (ASGD) optimization strategy described in [17]. All networks have 42 phone output targets such that the output layer would have few parameters and we could focus our attention on compressing other layers of the network. We use a exponentially decaying learning rate, which starts at 0.004 and has a decay rate of 0.1 over 15 billion frames. We apply gradient clipping from $\{1, 10, 100\}$ in RNN training, and cell clipping at 50 for LSTM training.

5. RESULTS

5.1. Learning Compact RNNs

5.1.1. Parameter reduction in bottom layers

We benchmark a full, standard RNN, which has 3 hidden layers with 600 cells per layer, and studied how we can reduce parameters relative to it with various compact architectures.

We compared different methods introduced in Section 3, with a focus on how bottom layers can be heavily condensed. To be more specific, we compress weight matrices U^1, W^2, U^2 of the first two layers, down to rank $r = 5$ for low rank, HashedNets and Toeplitz-like matrices, with the definition of rank shown in Table 1. Note that $r = 5$ is very limited compared to the original 600 dimensionality. 1.85 million parameters of the baseline model is cut down to $\sim$790k, roughly 40% of its original size.

model	compression	# of params	frame accuracy	WER
full	-	1.85m	73.26	43.5
low rank	$5, 5, 600$	790k	68.08	54.6
HashedNets	$5, 5, 600$	790k	70.09	49.2
Toeplitz-like	$5, 5, 600$	790k	70.79	48.4

Table 2: Learning different compact models for RNN

From Table 2, the Toeplitz-like transform is the most efficient to compress bottom layers, which attains the lowest WER under similar number of parameters. Given a fixed budget on model size, different compression schemes make different assumptions while compressing. The low rank assumption performs the poorest because rank 5 is too constrained. HashedNets imposes a somewhat weaker structure on the parameters via random grouping, and also performs moderately. On the other hand, a Toeplitz-like structured matrix with rank 5 can be interpreted as composition of convolutions and deconvolutions, and performs the best to reduce parameters in the bottom layers.

5.1.2. Parameter reduction across all layers with projection

We also benchmarked a low-rank model with shared factors [2] as described in 3.2. Here, we use 100 projection nodes for m_1, m_2 and 200 nodes for m_3. With just around 635k parameters, this model achieves an frame accuracy of 73.72% and word error rate of 43.5% matching the full RNN model with one-third the number of parameters. This projection model shares weights and thus gradients across all the layers and appears to be a more effective than the aggressive compression of bottom layers alone using HashedNets, Toeplitz-like matrices or untied low-rank models. One hypothesis we have is that perhaps Toeplitz-like compression, with its convolution interpretation, is better for lower layers, while shared low-rank factorizations are more effective for higher layers. This experiment was found to be difficult to run with RNNs, since in our experiments increasing the number of RNN layers exacerbated the vanishing/exploding gradient problem. Hence, we reserve this set of experiments for the next section, where we compress a deep LSTM with 5 layers, which exhibited more stable optimization behavior for this setting.

5.1.3. Displacement rank of Toeplitz-like transfrom

Next we explore the behavior of Toeplitz-like matrix by changing the displacement rank r. In Table 3, the WER improves when we compress U^1, W^2, U^2 to higher ranks, at the cost of increasing parameters and training time. From the column of seconds per ASGD optimization step, the training time is proportional to the displacement rank of Toeplitz-like matrix. As rank 5 gives us a reasonable

tradeoff between performance and training time, we will use it for further structured matrix experiments.

rank	# of params	sec. per step	frame accuracy	WER
2	779k	0.13	70.41	49.6
5	790k	0.28	70.79	48.4
10	808k	0.50	70.99	47.9

Table 3: Toeplitz-like matrices with different rank

5.2. Learning Compact LSTMs

In light of the observations from the RNN experiments, we mainly focus on LSTM with projection layer (Equation 10) and Toeplitz-like transforms in bottom layers for compact LSTM experiments. The full LSTM model has 5 hidden layer, with 500 hidden unit per layer. For projection model, we introduce projection $P^1 - P^5$, as in Equation 10, to all hidden layers, and the two numbers in column "compression" of Table 4 indicate the dimensions of h^1 to h^4, and h^5 respectively. For Toeplitz-like transform, we start from a projection model with 100 nodes in h^1 to h^4, and 200 nodes in h^5, and replace weight matrices with Toeplitz-like matrices progressively. 'U_x^j' in column "compression" stands for that all four weights $U_i^j, U_f^j, U_c^j, U_o^j$ in recurrent connections of layer j are compressed. We reduce all gates and cell state equally. Note that when we compress U_x^{l-1} and W_x^l at the same time, projection layer P^{l-1} will be removed. Column "projection" details the projection layers in the model.

model	proj.	compression $h^1 - h^4, h^5$	# of params	WER
full	-	-	9.12m	33.1
proj.	$P^1 - P^5$	100, 200	2.41m	33.6
proj.	$P^1 - P^5$	90, 200	2.23m	33.8
proj.	$P^1 - P^5$	80, 200	2.05m	34.2
Toep.	$P^1 - P^5$	U_x^1	2.23m	33.4
Toep.	$P^2 - P^5$	U_x^1, W_x^2	2.00m	33.5
Toep.	$P^2 - P^5$	U_x^1, W_x^2, U_x^2	1.82m	33.9
Toep.	$P^3 - P^5$	$U_x^1, W_x^2, U_x^2, W_x^3$	1.59m	35.4

Table 4: Learning different compact models for LSTM

For both projection model and Toeplitz-like transform, the performance drops as we reduce the number of parameters. However, we see that given a fixed model size, it is more effective to compress lower layers with Toeplitz-like matrices of low displacement rank, which has more of a convolutional interpretation, compared to projection compression on all layers with moderate rank. We see this is quite different than the behavior of compression on a shallow RNN (Section 5.1.2), since we can afford to have a deeper network with LSTM. Overall, with the combination of Toeplitz-like transform and projection, we are able to compress the LSTM model down by 75% to 2.2m parameters, with only a 0.3% increase in WER to 33.4.

Next, we try to answer the question of where in the LSTM we should apply the compression. We compare the effect of compressing recurrent weight U^l or non-recurrent weight W^l in Table 5. "compression" lists all Toeplitz-like matrices, where all 4 gates are compressed equally, and "projection" specifies projection layers in the model. As we can see, it makes no significant difference whether we compress recurrent or non-recurrent weight as long as the number of parameter matches.

Thus far we have compressed all gates in the same way. However, different gates have different levels of importance in LSTM [18] and perhaps less important gates can be compressed

compression	projection	# of params	WER
U_x^1, W_x^2	$P^1 - P^5$	2.05m	33.8
U_x^1, U_x^2	$P^1 - P^5$	2.05m	33.5
$U_x^1, W_x^2, U_x^2, W_x^3$	$P^2 - P^5$	1.64m	34.8
$U_x^1, W_x^2, U_x^2, U_x^3$	$P^2 - P^5$	1.64m	34.7

Table 5: Compression of feedforward or recurrent weights in LSTM

more. Thus, we investigate how the WER is affected by reducing different gates differently with Toeplitz-like matrices.

We take a LSTM projection model, which has 100 projection nodes in h^2 to h^4, and 200 projection nodes in h^5. All gates and cell states of U^1, W^2, U^2 are Toeplitz-like matrices. We vary the compression on cell state and gates of W^3 with Toeplitz-like transforms, and record the change in WER. These models lie in between the models of last two rows in Table 4, which has a huge jump in WER from 33.9 to 35.4.

In Table 6, gates column indicates if input gate, forget gate, cell state or output gate of W^3, is being compressed. Compressing the cell state (Equation 4) makes a major difference, 0.4% in WER, for LSTM performance. Compressing forget gate or not alone does not show much impact with 0.1% increase in WER. But reducing input gate, output gate and forget gate altogether, with 0.14m fewer parameters, would make 0.5% worse in WER. Both [18, 19] notice that removing forget gate would significantly hurt the performance, and [19] finds that output gate is of the least importance. We do not identify significant difference in compressing different gates in our experiment, probably because we only change the gates for the third layer input weights in a five-layer LSTM, and that rank 5 Toeplitz-like matrices are sufficient to retain enough information for most gates.

proj.	compression	compressed gates	# of params	WER
$P^2 - P^5$	U_x^1, W_x^2, U_x^2	$W_i^3, W_f^3, W_c^3, W_o^3$	1.64m	34.8
$P^2 - P^5$	U_x^1, W_x^2, U_x^2	W_i^3, W_f^3, W_o^3	1.68m	34.4
$P^2 - P^5$	U_x^1, W_x^2, U_x^2	W_i^3, W_o^3	1.73m	34.3
$P^2 - P^5$	U_x^1, W_x^2, U_x^2	$-$	1.82m	33.9

Table 6: Compression of different gates in LSTM

6. CONCLUSIONS

In this work, we studied how to build compact recurrent neural networks for LVCSR tasks. In our RNN experiments, we noted that Toeplitz-like structured matrices outperform HashedNets and Low-rank bottleneck layers for aggressive parameter reduction in the bottom layers. For LSTM parameter reduction, architecting upper layers with projection nodes to moderate rank, and bottom layers with Toeplitz-like transforms was found to be a particularly effective strategy. With this strategy, we are able to build a compact model with 75% fewer parameters than a standard LSTM model, while only incurring 0.3% increase in WER. Compressing recurrent or non-recurrent weight does not make significant difference. We find that LSTM performance is sensitive to cell state compression, making a noticeable change in WER.

7. ACKNOWLEDGEMENTS

8. REFERENCES

[1] G. Saon, H. Soltau, A. Emami, and M. Picheny, "Unfolded Recurrent Neural Networks for Speech Recognition," in *Interspeech*, 2014.

[2] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," in *Proc. Interspeech*, 2014.

[3] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks," in *Proc. ICASSP*, 2015.

[4] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-Rank Matrix Factorization for Deep Belief Network Training," in *Proc. ICASSP*, 2013.

[5] Jian Xue, Jinyu Li, and Yifan Gong, "Restructuring of Deep Neural Network Acoustic Models with Singular Value Decomposition," in *INTERSPEECH*, 2013, pp. 2365–2369.

[6] M. Collins and P. Kohli, "Memory-bounded Deep Convolutional Neural Networks," in *Proc. ICASSP*, 2013.

[7] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan, "Deep Learning with Limited Numerical Precision," in *Proc. ICML*, 2015.

[8] M. Courbariaux, J.-P. David, and Y. Bengio, "Low-precision Storage for Deep Learning," in *Proc. ICLR*, 2015.

[9] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen, "Compressing Neural Networks with the Hashing Trick," *arXiv preprint arXiv:1504.04788*, 2015.

[10] V. Sindhwani, T. N. Sainath, and S. Kumar, "Structured Transforms for Small-footprint Deep Learning," in *Proc. NIPS*, 2015.

[11] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," in *Deep Learning and Representation Learning Workshop, NIPS*, 2014.

[12] Sepp Hochreiter and Jürgen Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins, "Learning to Forget: Continual Prediction with LSTM," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[14] F. Gers, J. Schmidhuber, et al., "Recurrent Nets that Time and Count," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*. IEEE, 2000, vol. 3, pp. 189–194.

[15] G. Strang, *Introduction to Linear Algebra*, Wellesley-Cambridge Press, 4th edition, 2009.

[16] V. Pan, *Structured Matrices and Polynomials: Unified Superfast Algorithms*, Springer, 2001.

[17] J. Dean, G.S. Corrado, R. Monga, K. Chen, M. Devin, Q.V. Le, M.Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A.Y. Ng, "Large Scale Distributed Deep Networks," in *Proc. NIPS*, 2012.

[18] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber, "LSTM: A Search Space Odyssey," *arXiv preprint arXiv:1503.04069*, 2015.

[19] Wojciech Zaremba, "An Empirical Exploration of Recurrent Network Architectures," 2015.