Run-Time Efficient RNN Compression for Inference on Edge Devices

Urmish Thakker
Senior Research Engineer
Arm ML Research
urmish.thakker@arm.com

Jesse Beu
Staff Research Engineer
Arm ML Research
jesse.beu@arm.com

Dibakar Gope
Senior Research Engineer
Arm ML Research
dibakar.gope@arm.com

Ganesh Dasika
Principal Research Engineer
Arm ML Research
ganesh.dasika@arm.com

Matthew Mattina
Senior Director, Machine Learning and AI Research
Arm ML Research
matthew.mattina@arm.com

Abstract

Recurrent neural networks can be large and compute-intensive, yet many applications that benefit from RNNs run on small devices with very limited compute and storage capabilities while still having run-time constraints. As a result, there is a need for compression techniques that can achieve significant compression without negatively impacting inference run-time and task accuracy. This paper explores a new compressed RNN cell implementation called Hybrid Matrix Decomposition (HMD) that achieves this dual objective. This scheme divides the weight matrix into two parts - an unconstrained upper half and a lower half composed of rank-1 blocks. This results in output features where the upper sub-vector has "richer" features while the lower-sub vector has "constrained" features". HMD can compress RNNs by a factor of 2-4× while having a faster run-time than pruning and retaining more model accuracy than matrix factorization. We evaluate this technique on 3 benchmarks.

Keywords RNN, Compression

ACM Reference Format:
Urmish Thakker, Jesse Beu, Dibakar Gope, Ganesh Dasika, and Matthew Mattina. 2019. Run-Time Efficient RNN Compression for Inference on Edge Devices. In *Proceedings of 2019 (EMC2 Workshop).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Recurrent neural networks have shown state-of-the-art results for a wide variety of applications. Though many of these applications run on mobile devices, they are typically enabled by querying a cloud-based system to do most of the computation. The energy, latency, and privacy implications associated with running a query on the cloud is changing where users run a neural network application. We should, therefore, expect an increase in the number of RNNs running on embedded devices. Due to the energy and power constraints of edge devices, embedded SoCs frequently use lower-bandwidth memory technologies and smaller caches compared to desktop and server processors. Thus, there is a need for good compression techniques to enable large RNN models to fit into an edge device or ensure that they run efficiently on devices with smaller caches [16]. Additionally, compressing models should not negatively impact the inference run-time as these tasks may have realtime deadlines to provide a good user experience.

In order to choose a compression scheme for a particular network, one needs to consider 3 different axes – the compression factor, the speedup over the baseline, and the accuracy. Ideally, a good compression algorithm should not sacrifice improvement along one axis for improvement along another. For example, network pruning [7] has shown to be an effective compression technique, but pruning creates a sparse matrix representation that is inefficient to execute on most modern CPUs. Our analysis shows that pruned networks can achieve a faster run-time than the baseline only for significantly high compression factors. Low-rank matrix factorization (LMF) is another popular compression technique that can achieve speedup proportional to the compression factor. However, LMF has had mixed results in maintaining model accuracy [1, 5, 10, 14, 15]. This is because LMF reduces the rank of a matrix significantly, reducing its expressibility. Lastly, structured matrices [3, 15] can also be used to compress neural networks. While these techniques show a significant reduction in computation, this reduction only translates to a realized run-time improvement for larger matrices [17] or while using specialized hardware [9, 15].

To overcome the problem of finding an alternative to pruning, when LMF leads to loss in accuracy, we introduce a new compression technique called Hybrid Matrix Decomposition (HMD) which can act as an effective compression technique for edge use cases. The results are very promising – HMD achieves iso-accuracy for

EMC2 Workshop, Arizona,
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Algorithm 1 Reconstructing A in D2

Input: Matrices A' of dimension $r \times n$, B of dimension $(m-r) \times 1$, C of dimension $1 \times (n/2)$, D of dimension $(m-r) \times 1$, E of dimension $1 \times (n/2)$
Output: Matrix A of dimension $m \times n$

1: $G \leftarrow B \times C$
2: $H \leftarrow D \times E$
3: $K = concatenate(G, H, column)$
4: $A = concatenate(A', K, row)$

a large compression factor (2× to 3×), improves the run-time over pruning by a factor of 2×, improves run-time over a structured matrix-based technique by a factor of 30× and achieves better model accuracy than LMF.

The key contributions of this paper are:

- Introduction of a new compression technique called Hybrid Matrix Decomposition that can regain most of the baseline accuracy at 2× to 3× compression factors.
- Comparison of the model accuracy, inference run-time and compression trade-offs of HMD with network pruning and matrix factorization

2 Related Work

The research in NN compression can be categorized under 4 topics - Pruning, structured matrix based techniques, quantization and tensor decomposition.

Pruning [7, 24] has been the most successful compression technique for all types of neural networks. Poor hardware characteristics of pruning has led to research in block based pruning technique [11]. However, block based pruning technique also requires certain amount of block sparsity to achieve faster run-time than baseline.

Structured matrices have shown significant potential for compression of NN [3, 15]. Block circular compression is an extension of structured matrix based compression technique, converting every block in a matrix into structured matrix. We show that HMD is a superior technique than structured matrix based compression methods.

Tensor decomposition (CP decomposition, Tucker decomposition etc) based methods have also shown significant reduction in parameters [18]. Matrix Factorization [8] can also be categorized under this topic.

Lastly, quantization is another popular technique for compression [20] and is orthogonal to the work presented in this paper.

3 HMD-Based RNN Compression

The output of a RNN layer is a vector. Each element of the vector is derived from multiple fully connected layers followed by a non linearity operation. Thus, every element of an output vector is connected to every element of the input and hidden vectors of a RNN layer. This leads to a large number of parameters. Generally, not all elements of the output vector need to be connected this way to derive useful information from the input and the hidden vector; heat-maps of LSTM layers in [4, 21] show that there are many output vector elements with sparse dependence on the input and hidden vectors. Pruning exploits these sparse connections in an unstructured manner. Additionally, most RNN networks are followed by a fully-connected softmax layer or another RNN layer. Even if the order of the elements in the output of a particular RNN layer changes, the weights in the subsequent fully connected or RNN layers can adjust to accommodate that. Thus, the order of the output vectors of RNN hidden layers is not strictly important.

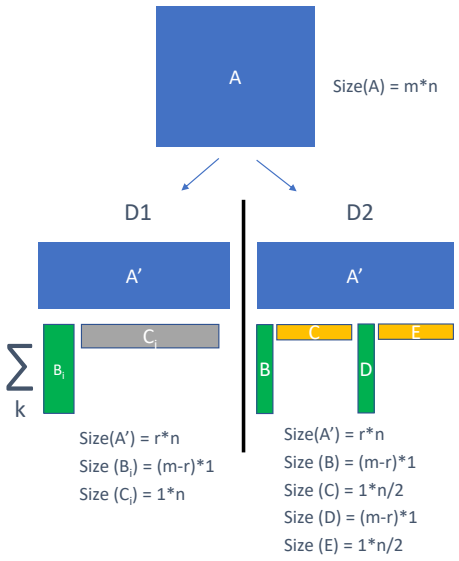

Figure 1. Representation of a matrix using hybrid decomposition

These two properties of a RNN layer can be used to create a more hardware-friendly compression scheme. This paper introduces one such scheme – Hybrid Matrix Decomposition. HMD splits the input and recurrent matrices in an RNN layer into two parts – a fully parameterized upper part and a lower part composed of rank-1 blocks. The upper part is used to generate elements of an output vector that need dense connectivity, while the lower part generates elements of the output vector that can generate useful information using sparse connectivity. There are multiple ways to constrain the lower part using rank-1 blocks. Figure 1 shows two such techniques (D1 and D2).

The D1 technique consists of an unconstrained upper half A' and a lower half that is composed of k rank-1 blocks. The lower half can be obtained by multiplying each B_i and C_i and adding all the corresponding matrices. If we decompose the weight matrix using D1 technique, the parameter reduction

HMD Technique	Reduction in model accuracy
D1 (k=1)	-2.03%
D1 (k=2)	-2.00%
D1 (k=5)	-1.20%
D2	**-0.15%**

Table 1. HMD compression's impact on the accuracy of an RNN KWS network for a compression factor of 2. Two variants of HMD are shown - D1 and D2. Negative values indicate a loss in accuracy due to compression, showing that D2 preserves significantly more accuracy than D1.

Algorithm 2 Matrix vector product when a matrix uses the HMD technique as shown in D2

Input 1: Matrices A' of dimension $r \times n$, B of dimension $(m-r) \times 1$, C of dimension $1 \times (n/2)$, D of dimension $(m-r) \times 1$, E of dimension $1 \times (n/2)$
Input 2: Vector I of dimension $n \times 1$
Output: Matrix O of dimension $m \times 1$

1: $O_{1:r} \leftarrow A' \times I$
2: $Temp1Scalar \leftarrow C \times I_{1:n/2}$
3: $Temp1 \leftarrow B \circ Temp1Scalar$
4: $Temp2Scalar \leftarrow E \times I_{1+n/2:n}$
5: $Temp2 \leftarrow D \circ Temp2Scalar$
6: $O_{r+1:m} \leftarrow Temp1 + Temp2$
7: $O = concatenate\{O_{1:r}, O_{r+1:m}\}$

is given by:

$$\frac{m \times n}{(r \times n) + k \times (m - r + n)} \tag{1}$$

The D2 technique is similar to the D1 technique in that it expresses the unconstrained upper half. However, it deviates from D1 in the way it constrains the lower half of the matrix. The lower half is composed of two rank-1 blocks. Algorithm 1 shows how to expand A',B,C,D, and E to get a matrix of size $m \times n$. If we decompose the weight matrix using D2 technique, the storage reduction is given by:

$$\frac{m \times n}{(r \times n) + 2 \times (m - r + n/2)} \tag{2}$$

We implemented these cells to empirically determine which technique (D1 or D2) works better. We used a keyword-spotting (KWS) benchmark from [23] as our baseline. The number of parameters was reduced by a factor of 2 using D2 and multiple configurations of D1. Table 1 shows the results of this experiment. Using D2, we only lose 0.15% of accuracy from the baseline. D1 leads to an accuracy loss of 1% or more for k values of 1,2 and 5. Based on these results, we use D2 as the preferred technique for HMD in the rest of this paper.

Apart from the storage reduction, HMD also leads to a reduction in the number of computations. Assuming a batch size of 1 during inference, Algorithm 2 shows how to calculate the matrix vector product when the matrix is represented using HMD. This algorithm avoids expanding the matrix A', B, C, D, and E into A as shown in Algorithm 1 and uses the associative property of matrix products to gain the computation speedup. For a matrix vector product between a matrix of size $m \times n$ and a vector of size $n \times 1$, the number of operations required to compute the product is $m \times n$ [19]. Referring to Algorithm 2, number of operations required to calculate $O_{1:r}$ is r×n. The Temp1 and Temp2 variables need $n/2 + m - r$ operations each. Temp1 and Temp2 are added together to calculate $O_{r+1:m}$. This requires an additional $m-r$ operations. Thus, the compression in number of operations when we use Algorithm 2 is:

$$\frac{m \times n}{r \times n + 2 \times (n/2 + m - r) + m - r} \tag{3}$$

As discussed previously, HMD divides the output into two stacked sub-vectors: One is a result of a fully-parameterized multiplication ($A' \times I$) and the other is the result of the low rank multiplication ($C \times R \times I_{1:n/2} + E \times F \times I_{1+n/2:n}$). Thus, the upper sub vector has "richer" features while the lower sub vector has "constrained" features.

4 Results

We do an extensive comparison of HMD with 2 other compression techniques – model pruning and matrix factorization. Additionally, we also compared HMD with a structured matrix-based compression technique called block circular decomposition (BCD) [2, 9]. BCD-compressed networks were able to recover the baseline accuracy for 2× - 4× compression. However, the run-time of the compressed network was 30× slower than baseline. As a result, we do not discuss the results using BCD compression in the rest of the paper.

Model pruning [24] induces sparsity in the matrices of a neural network, thereby reducing the number of non-zero valued parameters that need to be stored. Pruning creates sparse matrices which are stored in a specialized sparse data structure such as CSR. The overhead of traversing these data structures while performing the matrix-vector multiplication can lead to poorer inference run-time than when executing the baseline, non-sparse network.

Low Rank Matrix Factorization (LMF) [8] expresses a larger matrix A of dimension $m \times n$ as a product of two smaller matrices U and V of dimension $m \times d$ and $d \times n$, respectively. Parameter d controls the compression factor. Unlike pruning, Matrix Factorization is able to improve the run-time over the baseline for all compression factors.

4.1 Comparison of compression techniques across different ML tasks

The impact of compression on accuracy is compared for 3 benchmarks covering 2 different tasks – Human Activity Recognition and Language Modeling. These tasks are some

of the important applications that run on edge and embedded devices. We use the Tensorflow framework (v1.8) to train our models on a machine with two 1080 Ti GPUs. In order to compare the inference run-time of RNN cells compressed using the 3 techniques discussed above, we implemented these cells in C++ using the Eigen library. This paper focuses on inference on an edge device. As a result, we make the assumption that the batch size of the application will be 1 while measuring the run-time of an application. However, the observations regarding run-time should remain consistent for larger batch sizes as well. We ran our experiments on a single cortex-A73 core of the Hikey 960 board. The size of L3 cache is 2MB.

We compress the network using pruning, LMF, and HMD. Additionally, we train a smaller baseline with the number of parameters equal to that of the compressed baseline. This serves as a useful point of comparison to check if compression of a larger network leads to better performance or preserving the network architecture but reducing its dimensions.

4.1.1 Human Activity Recognition (HAR)

We train two different networks for human activity recognition. Both of these networks are trained on the Opportunity dataset [13]. However, they differ in the way they process the dataset and the body sensors they chose to train their networks on.

HAR1: The first HAR network is based on the work in [6]. Their network uses a bidirectional LSTM with hidden length of size 179 followed by a softmax layer to get an accuracy of 92.5%. Input is of dimension 77 and is fed over 81 time steps. The total number of parameters in this network are 374,468. Using their training infrastructure, the suggested hyperparameters in the paper got us an accuracy of 91.9%. Even after significant effort, we were not able get to the accuracy mentioned in the paper. Henceforth, we will use 91.9% as the baseline accuracy.

Figure 2 shows the result of compressing the LSTM layers in the baseline by 2×, 2.5×, 3.33× and 5×. As we increase the compression, the accuracy degradation becomes larger for all compression schemes. Thus, the best compression scheme for each compression factor is a function of task accuracy and speedup required to run the application. For 2× compression, HMD and pruning achieve better accuracy than the smaller baseline and LMF. Additionally, the HMD compressed network is 2× faster than the pruned network. Similar observations can be made for 2.5× and 3.33× compression. Thus, HMD can be used as the preferred compression scheme for these compression factors. At 5× compression, HMD is slightly more accurate than LMF while being 15% slower. The preferred choice for compression scheme depends on what criteria (accuracy or speed) one is willing to sacrifice. Finally, all three compression schemes have better accuracy than the smaller baseline.

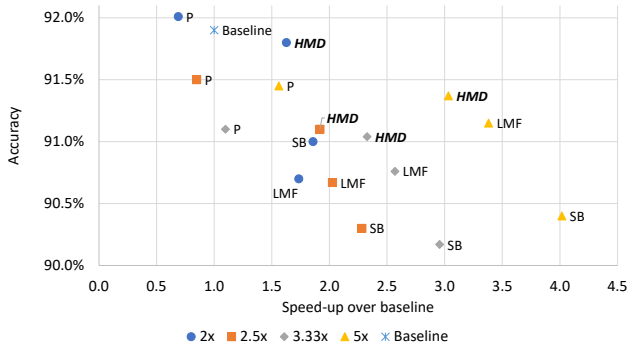

Figure 2. Accuracy vs speedup for the HAR1 network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values > 1 indicate a decrease in inference run-time and values < 1 indicate an increase in inference run-time. For each compression factor, the compression scheme that is most to the top-right is the ideal choice and is highlighted in bold italics. P = Pruning, LMF = Low rank matrix factorization, HMD = Hybrid matrix decomposition, SB = Smaller baseline.

HAR2: The second HAR network is based on the work in [12]. They use 113 sensors from the Opportunity dataset. The network has 4 convolutional layers followed by 2 LSTM layers and a softmax layer. The total number of parameters in the network are 3,964,754. The LSTM layers are of size 128 contributing to more than 95% of the total parameters.

Figure 3 shows the result of compressing the LSTM layers in baseline by 2×, 2.5×, 3.33× and 5×. As we increase the compression, the accuracy degradation becomes larger for all compression schemes. For 2× and 2.5× compression factors, HMD is the superior technique, achieving better run-time than pruning (2× faster) and better accuracy than LMF (improvement of 0.4%). Additionally, accuracy of the HMD compressed network is within 0.2% of baseline. For higher compression factors, LMF becomes an attractive option to compress the HAR2 application. For 3.33× compression, LMF, HMD and pruning achieve equivalent accuracy. However, LMF is slightly faster than HMD and more than 2× faster than pruning. Finally, all three compression schemes have better accuracy than the smaller baseline.

4.1.2 Language Modeling

We use the small model from [22] as our baseline. The baseline has 2 LSTM layers each with a hidden vector of size 200. Additionally, it uses 10,000 words from the English vocabulary. Together with the input and output word embeddings, the total size of the network is 4,171,000 parameters.

Figure 4 shows the results of compressing the LSTM layers in the baseline by 2×, 2.5×, 3.33× and 5×. In case of

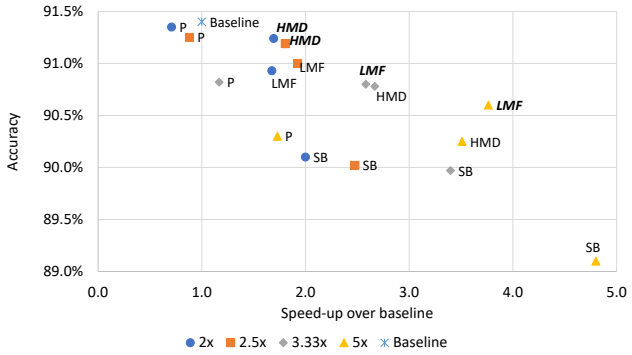

Figure 3. Accuracy vs speedup for HAR2 network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values > 1 indicate a decrease in inference run-time and values < 1 indicate an increase in inference run-time. For each compression factor, the compression scheme that is most to the top-right is the ideal choice and is highlighted in bold italics. P = Pruning, LMF = Low rank matrix factorization, HMD = Hybrid matrix decomposition, SB = Smaller baseline.

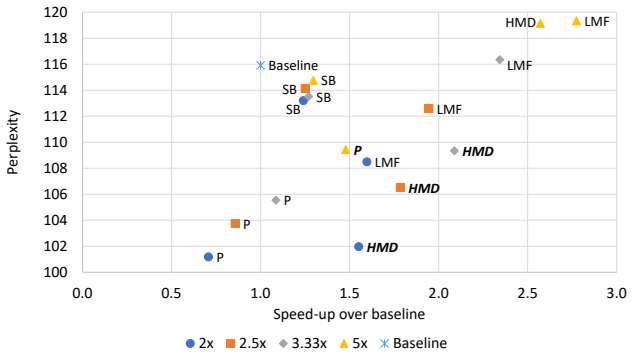

Figure 4. Perplexity vs speedup for PTB-LM network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values > 1 indicate a decrease in inference run-time and values < 1 indicate an increase in inference run-time. In case of perplexity, lower values are better. Thus, for each compression factor, the compression scheme that is most to the bottom-right is the ideal choice and is highlighted in bold italics. P = Pruning, LMF = Low rank matrix factorization, HMD = Hybrid matrix decomposition, SB = Smaller baseline.

LM, lower the perplexity, better the model. Pruning consistently achieves better accuracy than baseline and other compression techniques. However, pruning never achieves a better speedup than other compression techniques and starts getting a speedup over baseline for compression values of 3.33× or above. LMF achieves better perplexity than baseline for 2× and 2.5× compression and achieves speedup for all compression factors. But LMF, does not beat the perplexity values achieved by HMD. HMD simultaneously achieves better perplexity than baseline for most compression factor, better perplexity than LMF for all compression factors and faster inference run-time than baseline and pruned networks for all compression factors. HMD also achieves faster inference run-time than the smaller baseline, but has marginally less perplexity for 3.33× compression. Thus, HMD makes a strong case for being the preferred compression scheme for 2×, 2.5× and 3.33× compression factors. At 5× perplexity, pruning would be the preferred compression scheme.

5 Discussion of why HMD is a more effective compression technique

The results show that HMD is an exciting alternative to traditional compression techniques like pruning and LMF. HMD retains a dense representation of the matrices. As a result, it has better hardware characteristics than pruning and can deliver inference speed-up for all compression factors. For the same compression factor, HMD creates a higher rank matrix than LMF. For a matrix of size 256×256 and 2× compression, LMF creates a matrix of rank 64 while HMD creates a matrix of rank 128. Thus, HMD can lead to better accuracy than LMF. Infact, HMD is an extension of LMF. To understand this, let us revisit Algorithm 1, where the output A = [A'; BC, DE]. Suppose A'=RS, and let P=[B, D], Q=[C, 0; 0, E], then A becomes A=[R, 0; 0,P] * [S, 0; Q]=U'V', where both U' and V' have ranks at most (r+2). A standard LMF decomposition of A will also lead to a representation of the form UV, but this representation will have same parameters as HMD only if the rank of both U and V is at most (r+2)/2. Thus, HMD can be regarded as a (r+2) ranked LMF of A, with a sparsity forcing mask that reduces the number of parameters to express the (r+2) ranked matrix significantly. Finally, unlike the smaller baseline, HMD does not reduce the size of the output vector of a RNN layer. This allows more information to be captured after each RNN layer. Thus, HMD can lead to better accuracy than the smaller baseline.

6 Conclusion

Choosing the right compression technique requires looking at three criteria – compression factor, accuracy, and run-time. Pruning is an effective compression technique, but can sacrifice speedup over baseline for certain compression factors. LMF achieves better speedup than baseline for all compression factors, but can lead to accuracy degradation. This paper introduces a new compression scheme called HMD, which is extremely effective when compression using pruning does not lead to speedup over baseline and LMF leads to accuracy

degradation. Additionally, HMD compressed models can be further compressed using quantization.

References

[1] Ting Chen, Ji Lin, Tian Lin, Song Han, Chong Wang, and Denny Zhou. 2018. Adaptive Mixture of Low-Rank Factorizations for Compact Neural Modeling. *Advances in neural information processing systems (CDNNRIA workshop)* (2018). https://openreview.net/forum?id=B1eHgu-Fim

[2] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, Xiaolong Ma, Yipeng Zhang, Jian Tang, Qinru Qiu, Xue Lin, and Bo Yuan. 2017. CirCNN: Accelerating and Compressing Deep Neural Networks Using Block-circulant Weight Matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17)*. ACM, New York, NY, USA, 395–408. https://doi.org/10.1145/3123939.3124552

[3] Caiwen Ding, Ao Ren, Geng Yuan, Xiaolong Ma, Jiayu Li, Ning Liu, Bo Yuan, and Yanzhi Wang. 2018. Structured Weight Matrices-Based Hardware Accelerators in Deep Neural Networks: FPGAs and ASICs. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI (GLSVLSI '18)*. ACM, New York, NY, USA, 353–358. https://doi.org/10.1145/3194554.3194625

[4] Jamin Shin Elham J. Barezi Genta Indra Winata, Andrea Madotto. 2019. Low-Rank Matrix Factorization of LSTM as effective model compression. https://openreview.net/forum?id=BylahsR9tX

[5] Artem M. Grachev, Dmitry I. Ignatov, and Andrey V. Savchenko. 2017. Neural Networks Compression for Language Modeling. In *Pattern Recognition and Machine Intelligence*, B. Uma Shankar, Kuntal Ghosh, Deba Prasad Mandal, Shubhra Sankar Ray, David Zhang, and Sankar K. Pal (Eds.). Springer International Publishing, Cham, 351–357.

[6] Nils Y Hammerla, Shane Halloran, and Thomas Ploetz. 2016. Deep, convolutional, and recurrent models for human activity recognition using wearables. *IJCAI 2016* (2016).

[7] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* (2016).

[8] Oleksii Kuchaiev and Boris Ginsburg. 2017. Factorization tricks for LSTM networks. *CoRR* abs/1703.10722 (2017). arXiv:1703.10722 http://arxiv.org/abs/1703.10722

[9] Zhe Li, Shuo Wang, Caiwen Ding, Qinru Qiu, Yanzhi Wang, and Yun Liang. 2018. Efficient Recurrent Neural Networks using Structured Matrices in FPGAs. *CoRR* abs/1803.07661 (2018). arXiv:1803.07661 http://arxiv.org/abs/1803.07661

[10] Zhiyun Lu, Vikas Sindhwani, and Tara N. Sainath. 2016. Learning Compact Recurrent Neural Networks. *CoRR* abs/1604.02594 (2016). arXiv:1604.02594 http://arxiv.org/abs/1604.02594

[11] Sharan Narang, Eric Undersander, and Gregory F. Diamos. 2017. Block-Sparse Recurrent Neural Networks. *CoRR* abs/1711.02782 (2017). arXiv:1711.02782 http://arxiv.org/abs/1711.02782

[12] Francisco Javier OrdÃşÃśez and Daniel Roggen. 2016. Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. *Sensors* 16, 1 (2016). https://doi.org/10.3390/s16010115

[13] D. Roggen, A. Calatroni, M. Rossi, T. Holleczek, K. FÃűrster, G. TrÃűster, P. Lukowicz, D. Bannach, G. Pirkl, A. Ferscha, J. Doppler, C. Holzmann, M. Kurz, G. Holl, R. Chavarriaga, H. Sagha, H. Bayati, M. Creatura, and J. d. R. MillÃăn. 2010. Collecting complex activity datasets in highly rich networked sensor environments. In *2010 Seventh International Conference on Networked Sensing Systems (INSS)*. 233–240. https://doi.org/10.1109/INSS.2010.5573462

[14] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. 2013. Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 6655–6659. https://doi.org/10.1109/ICASSP.2013.6638949

[15] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. 2015. Structured Transforms for Small-Footprint Deep Learning. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3088–3096.

[16] Urmish Thakker, Ganesh Dasika, Jesse G. Beu, and Matthew Mattina. 2019. Measuring scheduling efficiency of RNNs for NLP applications. *CoRR* abs/1904.03302 (2019). arXiv:1904.03302 http://arxiv.org/abs/1904.03302

[17] Anna Thomas, Albert Gu, Tri Dao, Atri Rudra, and Christopher Ré. 2018. Learning Compressed Transforms with Low Displacement Rank. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 9066–9078. http://papers.nips.cc/paper/8119-learning-compressed-transforms-with-low-displacement-rank.pdf

[18] Andros Tjandra, Sakriani Sakti, and Satoshi Nakamura. 2017. Compressing recurrent neural network with tensor train. In *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 4451–4458.

[19] Lloyd Trefethen and David Bau. 1997. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics.

[20] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. 2011. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*.

[21] Wei Wen, Yiran Chen, Hai Li, Yuxiong He, Samyam Rajbhandari, Minjia Zhang, Wenhan Wang, Fang Liu, and Bin Hu. 2018. Learning Intrinsic Sparse Structures within Long Short-Term Memory. https://www.microsoft.com/en-us/research/publication/learning-intrinsic-sparse-structures-within-long-short-term-memory/

[22] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent Neural Network Regularization. *CoRR* abs/1409.2329 (2014). arXiv:1409.2329 http://arxiv.org/abs/1409.2329

[23] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. *CoRR* abs/1711.07128 (2017). arXiv:1711.07128 http://arxiv.org/abs/1711.07128

[24] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv e-prints*, Article arXiv:1710.01878 (Oct. 2017), arXiv:1710.01878 pages. arXiv:stat.ML/1710.01878