Run-Time Efficient RNN Compression for Inference on Edge Devices

Urmish Thakker [1]
Jesse Beu [1]
Dibakar Gope [1]
Ganesh Dasika [1]
Matthew Mattina [2]

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$\times$ while having a faster run-time than pruning (Zhu & Gupta, 2017) and retaining more model accuracy than matrix factorization (Grachev et al., 2017). We evaluate this technique on 5 benchmarks spanning 3 different applications, illustrating its generality in the domain of edge computing.

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 (Thakker et al., 2019b). Additionally, compressing models should not negatively impact the inference run-time as these tasks may have real-time 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 inference run-time 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 (Han et al., 2016) 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 (Grachev et al., 2017; Sainath et al., 2013; Chen et al., 2018; Sindhwani et al., 2015; Lu et al., 2016). This is because LMF reduces the rank of a matrix significantly, reducing its expressibility (Yang et al., 2018). Lastly, structured matrices (Sindhwani et al., 2015; Ding et al., 2018) 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 improve-

[1]Arm ML Research Lab, Austin, USA [2]Arm ML Research Lab, Boston, USA. Correspondence to: Urmish Thakker <urmish.thakker@arm.com>.

Preliminary work. Under review by the Systems and Machine Learning (SysML) Conference. Do not distribute.

ment for larger matrices (Thomas et al., 2018) or while using specialized hardware (Li et al., 2018; Sindhwani et al., 2015).

Given the good run-time characteristics, LMF can potentially act as an alternative to pruning. However, when LMF leads to an accuracy loss, we no longer have a viable alternative to pruning. **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). HMD can act as an effective compression technique for edge use cases on embedded CPUs.** The results are very promising – HMD achieves iso-accuracy for a large compression factor ($2\times$ to $4\times$), improves the CPU run-time over pruning by a factor of $2\times$, improves CPU run-time over a structured matrix-based technique by a factor of $30\times$ and achieves better model accuracy than LMF.

2 Related Work

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

Pruning (Han et al., 2016; Zhu & Gupta, 2017) 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 (Narang et al., 2017). 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 (Sindhwani et al., 2015; Ding et al., 2018; Wang et al., 2018a; Ding et al., 2017; Li et al., 2018; Cheng et al., 2015; Wang et al., 2018b). Block circular compression is an extension of structured matrix based compression technique, converting every block in a matrix into structured matrix. We will show in this paper that HMD is a superior technique than block circular decomposition.

Tensor decomposition (CP decomposition, Kronecker, Tucker decomposition etc) based methods have also shown significant reduction in parameters (Tjandra et al., 2017; Thakker et al., 2019c;a). Matrix Factorization (Kuchaiev & Ginsburg, 2017; Chen et al., 2018; Grachev et al., 2017) can be categorized under this topic. We will show in this paper that HMD can lead to better accuracy than LMF compressed RNNs.

Lastly, quantization is another popular technique for compression (Vanhoucke et al., 2011; Zhu et al., 2016; Lin et al., 2015; Hubara et al., 2017; 2016; Courbariaux & Bengio, 2016). We will show that HMD also benefits from any research in this domain.

3 Hybrid Matrix Decomposition Based RNN Compression

3.1 Why LMF can potentially lead to loss in accuracy

LMF (Kuchaiev & Ginsburg, 2017) 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 most compression factors. Unfortunately, compression via LMF can lead to loss in accuracy. We believe, this is because of two closely related reasons:

- **Rank-Loss**: The rank of a matrix is a measure of the expressibility of a matrix. A lower rank matrix can express lesser range of values, limiting its learning capacity. This can potentially lead to poorer accuracy. LMF compression leads to a lower ranked matrix. While before compression, the rank of matrix A is $min(m, n)$, after compression becomes $min(m, n, d)$. To achieve compression, d needs to be smaller than m and n. Thus, rank of compressed A is lower than the original matrix. Eg - for a 256×256 sized matrix, compression using LMF by a factor of 2 leads to U and V matrices of size 256×64 and 64×256. The resultant compressed matrix A (= $U * V$) is a 64 rank matrix. Thus, in order to compress the matrix by a factor of 2, LMF reduces the rank of a matrix by a factor of 4. **This loss in rank of the matrix expressed using LMF can lead to sub-par accuracy** (Yang et al., 2018).

- **Less expressive output features**: A closely related argument can be viewed when we extend the idea of low-rank matrix and its impact on the output features. Without loss of generality, an RNN calculates a matrix-vector product during inference. If we assume the parameters of a RNN are represented by a matrix A of dimension $m \times n$ and the input to the matrix is x of dimension $n \times 1$, then the output feature, $y = f(A * x)$, where f is a non-linear function and y is of dimension $m \times 1$. Thus, each element of y is a dot product of a row of A and the vector x followed by non-linearity. LMF expresses A in a lower dimensional space using the U and V matrix. If we rewrite the equation to calculate y, when A is expressed using LMF, we get -

$$y = f(U * V * x) \quad (1)$$

$$y = f(U * k) \ \ (assuming\ k = V * x) \quad (2)$$

where k has a dimension of $d \times 1$. Generally, for compression $d < m, n$. Thus, x of dimension $n \times 1$ is projected to a lower dimensional embedding of size $d \times 1$ and expanded again to $m \times 1$ to create y. Thus,

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: $A = concatenate(A', K, row)$
```

compressing A to a lower rank leads to output features calculated from a lower dimension embedding vector.

3.2 Hybrid Matrix Decomposition

This paper introduces a new compression technique that uses dense matrix representation to ensure fast run-time properties and avoids making the strong assumptions made by LMF. This technique is based on three assumptions -

- **A1:** Rank of a matrix is important to create a high-task accuracy RNN network (Yang et al., 2018)
- **A2:** LMF makes a strong assumption that all the elements of the output feature vector of a RNN can be expressed from a low-dimensional embedding vector (Equation 2). HMD is based on the assumption that a more relaxed constraint of having a hybrid output feature vector, where some elements are calculated from a lower dimensional embedding space and other's from a higher dimensional embedding space can lead to better accuracy.
- **A3:** 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 elements of the output vector of RNN layer is not strictly important.

These three intuitions of a RNN layer can be used to create a more hardware-friendly compression scheme. This paper introduces one such scheme – Hybrid Matrix Decomposition.

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. While various approaches can be taken to constrain the lower sub-matrix, two example strategies (Figure 1) are compared in the remainder of this section to demonstrate the design considerations when using this technique.

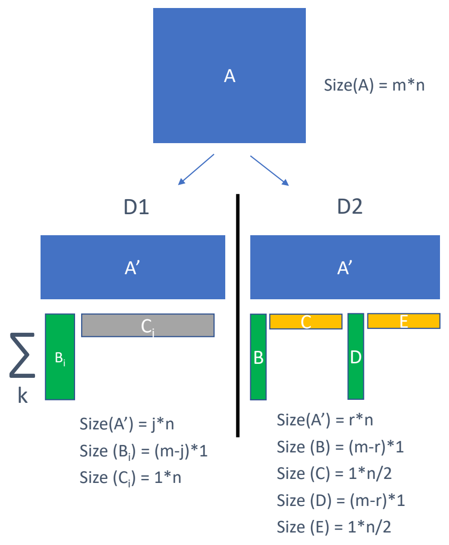

Figure 1. Representation of a matrix using hybrid decomposition

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 is given by:

$$\frac{m \times n}{(j \times n) + k \times (m - j + n)} \tag{3}$$

Thus, the maximum rank of the matrix becomes $j + k$. Different values of j and k can be used to control the amount of compression and the rank of the matrix.

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 four 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:

$$\frac{m \times n}{(r \times n) + 2 \times (m - r + n/2)} \tag{4}$$

Thus, the maximum rank of the matrix becomes $r + 2$. Different values of r can be used to control the amount of compression and the rank of the matrix.

Structuring a matrix as shown in D1 and D2 can lead to significant increase in maximum rank of a matrix. Table-1 shows the maximum possible rank of a 256×256 matrix compressed to the same number of parameters using the three compression techniques - LMF, HMD-D1 and HMD-D2. As shown, D1 and D2 can effectively double the rank of the matrix for the same number of parameters. To compress

Table 1. The maximum possible rank of a 256×256 sized matrix after it is compressed by 4 different factors using 3 different compression techniques. To compress a matrix by a given compression factor, D1 has 2 different parameters j and k to regulate the rank of the matrix. Hence, we see a range of rank values.

Matrix of Size (256,256)			
Compression Factor	LMF	D1	D2
1.25	102	103 - 204	205
1.67	76	78 - 153	153
2.50	51	52 - 101	102
5.00	25	26 - 50	50

Table 2. 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. **Based on these results, we use D2 as the preferred technique for HMD in the rest of this paper**

HMD Technique	Reduction in model accuracy
D1 (k=1)	-2.03%
D1 (k=2)	-2.00%
D1 (k=3)	-1.20%
D2	-0.15%

a matrix by a given compression factor, D1 has 2 different parameters j and k to regulate the rank of the matrix. Hence, we see a range of rank values.

We implemented these cells to empirically determine which technique (D1 or D2) works better. We used a keyword-spotting (KWS) benchmark from (Zhang et al., 2017) as our baseline. The number of parameters was reduced by a factor of 2 using D2 and multiple configurations of D1. Table 2 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 $m \times n$ (Trefethen & Bau, 1997). Referring to Algorithm 2, number of operations required to calculate $O_{1:r}$ is $r \times n$. The Temp1 and Temp2 variables need $n/2 + m - r$ operations each. Temp1 and Temp2 are added together to calculate $O_{r+1:m}$.

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: $O = concatenate\{O_{1:r}, O_{r+1:m}\}$
```

This requires an additional $m - r$ operations. Thus, the reduction in number of operations when we use Algorithm 2 is:

$$\frac{m \times n}{r \times n + 2 \times (n/2 + m - r) + m - r} \tag{5}$$

3.3 Impact on output feature vector

Algorithm 2 shows that, HMD divides the output into two stacked sub-vectors. One is a result of a fully-parameterized multiplication, $A' \times I$ (Line 1, Algorithm 2). The other is the result of the low rank multiplication : $C \times R \times I_{1:n/2} + E \times F \times I_{1+n/2:n}$ (Line 2-6, Algorithm 2). Thus, the upper sub-vector has "richer" features created from a higher dimensional embedding, while the lower sub vector has "constrained" features created from a lower dimensional embedding. By incorporating the HMD structure during training, we force an RNN to learn "richer" features in the upper sub-vector and the "constrained" features in the lower sub-vector. Because a RNN is followed by another RNN or a softmax layer, this restructuring should not impact the subsequent layers. **Thus HMD structure combines the assumptions A2 and A3 that were discussed previously**.

3.4 Relationship between HMD and LMF

HMD can also be viewed as 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; Q]=U'V', where both U' and V' can have a maximum rank of $r + 2$. 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. Neural networks seldom learn structured sparsity unless they are forced to (Narang et al., 2017), thus, an RNN trained with the LMF structure will rarely end up learning the same structure as HMD. Such a sparsity forcing mask also leads to creation of the decoupled output feature vectors as described in section 3.3.

4 Results

We compare HMD with LMF and 3 other compression techniques – model pruning, small baseline and a structured matrix based technique called block circular decomposition. These techniques and why they need to be considered are discussed below:

- BCD: In order to determine an alternative to pruning, when LMF leads to loss in accuracy, it is important to understand whether a structured matrix based compression technique can fill that void. BCD (Li et al., 2018; Ding et al., 2017) is one of the most popular compression method in this class. (Li et al., 2018; Ding et al., 2017) were able to recover the baseline accuracy for $2\times$ - $4\times$ compression factors. However, the runtime of this technique was evaluated on a specialized accelerator. *Our results indicate that the run-time of the BCD compressed networks on CPU was* $30\times$ *slower than baseline*. This is in line with the research in (Thomas et al., 2018), which states that structured matrix based compression techniques achieve speedup after compression for matrix sizes of 2048×2048 or more. The applications we evaluated in this paper don't have matrices that match those size requirements. As a result, we do not discuss the results using BCD compression in the rest of the paper.
- Pruning: Model pruning (Zhu & Gupta, 2017) 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. Thus, while pruning is an effective compression technique, its run-time performance on CPUs can make it a less appealing choice for compression. This paper tries to find an alternative to pruning, when LMF leads to loss in accuracy. To show that such an opportunity exists, we show results of an equivalently pruned RNN.
- Small Baseline: 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 because of two reasons. First, to check if compression of a larger network leads to better accuracy than compressing a network by reducing its dimensions (size of hidden layer or number of layers). This can help us verify if the network was originally over-parameterized. Second, to establish the hypothesis whether HMD's creation of a stacked output feature vector as described in section 3.3 adds any useful information in the network. Smaller baseline creates output feature vector that is created from a high-dimensional embedding only. Generally, for most compression points evaluated in this paper, the size of this output feature vector is slightly larger than the upper stack of HMD's output feature vector. HMD, additionally concatenates the output features created from lower dimensional embedding. Thus, comparing the accuracy of HMD with Smaller baseline helps evaluate the usefulness of the output features created using lower dimensional embedding.

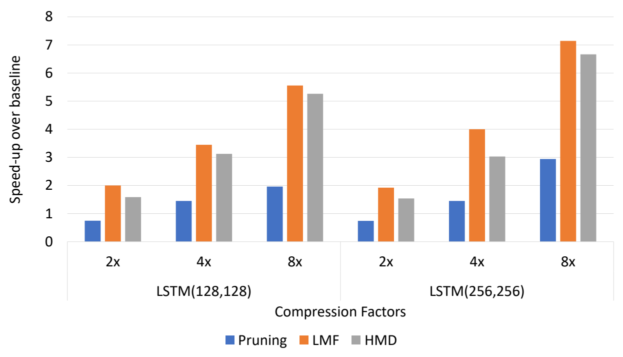

Figure 2. Inference speedup over baseline after applying various compression schemes (Pruning, LMF and HMD) by 3 different compression factors. Two different baselines are shown – one where the input and hidden vector length of the LSTM cell are 128 and other when the input and hidden vector length of the LSTM cell are 256. Values < 1 indicate a slowdown and values > 1 indicate a speedup over the baseline.

4.1 Effect of compression technique on inference run-time

In order to compare the inference run-time of RNN cells compressed using pruning, LMF and HMD, 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. For our experiments, we consider a single inference of an LSTM cell for different input and hidden lengths. We ran our experiments on a single cortex-A73 core of the Hikey 960 board. The size of L3 cache is 2MB.

Figure 2 shows the ratio of run-time of the compressed cell to the baseline. HMD can be $2\times$-$3\times$ faster than pruned LSTM cells. The run-time of LSTM cells compressed using LMF is also strictly better than that of the pruned LSTM cells. This is expected as LMF also avoids a costly sparse matrix multiplication. The difference in run-time between LMF and HMD LSTM cells decreases as the compression factor increases. At $2\times$ compression, LMF is 25% faster than HMD, while at $8\times$ compression, LMF is 7% faster than HMD. However, as shown in the next section, LMF gets this speed-up at the cost of accuracy.

4.2 Comparison of compression techniques across different ML tasks

The impact of compression on accuracy is compared for 5 benchmarks covering 3 different tasks – Key Word Spotting, Human Activity Recognition and Language Modeling. These tasks are some of the most 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.

4.2.1 Human Activity Recognition (HAR)

We train two different networks for human activity recognition. Both of these networks are trained on the Opportunity dataset (Roggen et al., 2010). 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 (Hammerla et al., 2016). 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 3 shows the result of compressing the LSTM layers in the baseline by $2\times$, $2.5\times$, $3.33\times$ and $5\times$. 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 $2\times$ compression, HMD and pruning achieve better accuracy than the smaller baseline and LMF. Additionally, the HMD compressed network is $2\times$ faster than the pruned network. Similar observations can be made for $2.5\times$ and $3.33\times$ compression. Thus, HMD can be used as the preferred compression scheme for these compression factors. At $5\times$ 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.

HAR2: The second HAR network is based on the work in (Ordez & Roggen, 2016). 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 4 shows the result of compressing the LSTM layers in baseline by $2\times$, $2.5\times$, $3.33\times$ and $5\times$. As we increase the compression, the accuracy degradation becomes larger for all compression schemes. For $2\times$ and $2.5\times$ compression factors, HMD is the superior technique, achieving better run-time than pruning ($2\times$ 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 $3.33\times$ compression, LMF, HMD and pruning achieve equivalent accuracy. However, LMF is slightly faster than HMD and more than $2\times$ faster than pruning. Finally, all three compression schemes have better accuracy than the smaller baseline.

4.2.2 Language Modeling

We use the small and medium model from (Zaremba et al., 2014) as our baselines. The PTB (Small) baseline has 2 LSTM layers each with a hidden vector of size 200. The PTB (Medium) baseline has 4 LSTM layers each with a hidden vector of size 650. Additionally, both use 10,000 words from the English vocabulary.

Figure 5 shows the results of compressing the LSTM layers in the PTB (Small) baseline by $2\times$, $2.5\times$, $3.33\times$ and $5\times$. In case of 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 $3.33\times$ or above. LMF achieves better perplexity than baseline for $2\times$ and $2.5\times$ 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 $3.33\times$ compression. Thus, HMD makes a strong case for being the preferred compression scheme for $2\times$, $2.5\times$ and $3.33\times$ compression factors. At $5\times$ perplexity, pruning would be the preferred

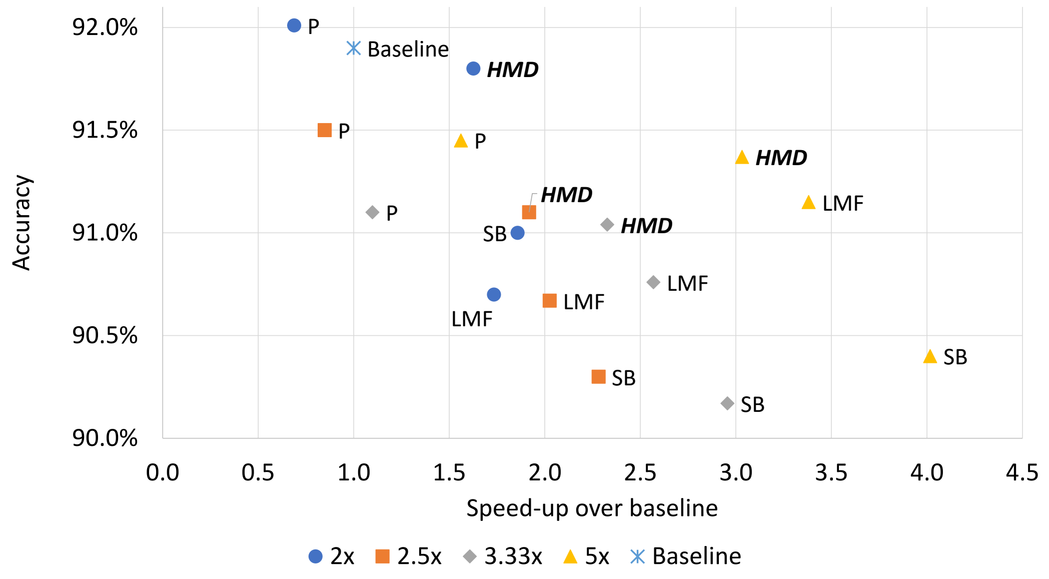

Figure 3. Accuracy vs speedup for the HAR1 network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values > 1 indicate a decrease in inference run-time and values < 1 indicate an increase in inference 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. The best way to view this figure is to focus on a compression point and see how the Pareto curve of accuracy vs speed-up changes as we add HMD. For all 4 compression points, HMD provides a viable alternative to both pruning and LMF.

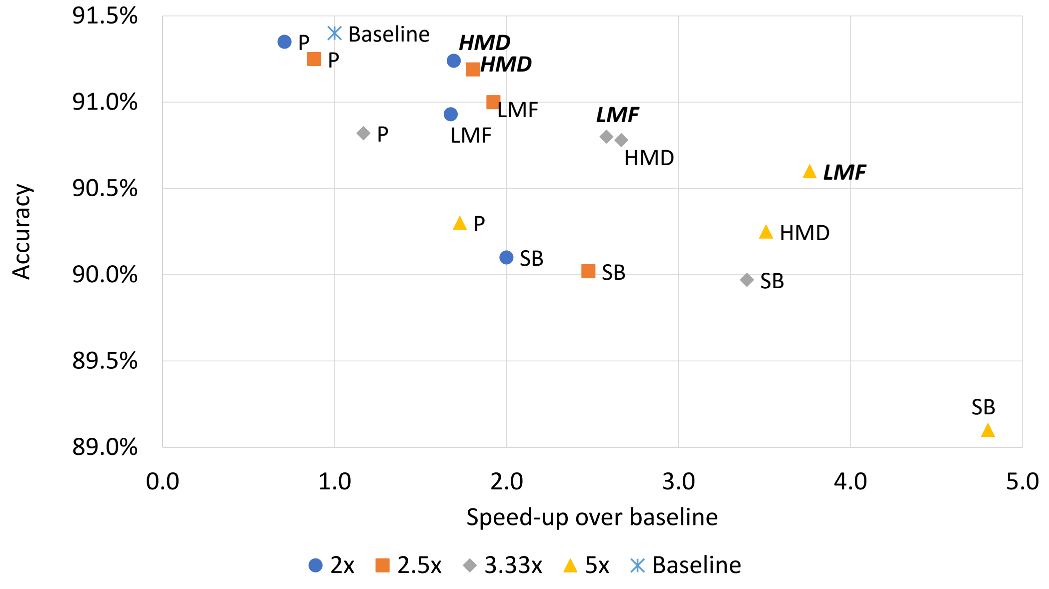

Figure 4. Accuracy vs speedup for HAR2 network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values > 1 indicate a decrease in inference run-time and values < 1 indicate an increase in inference 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. The best way to view this figure is to focus on a compression point and see how the Pareto curve of accuracy vs speed-up changes as we add HMD. HMD provides a viable alternative to both pruning and LMF at compression factors of 2x and 2.5x

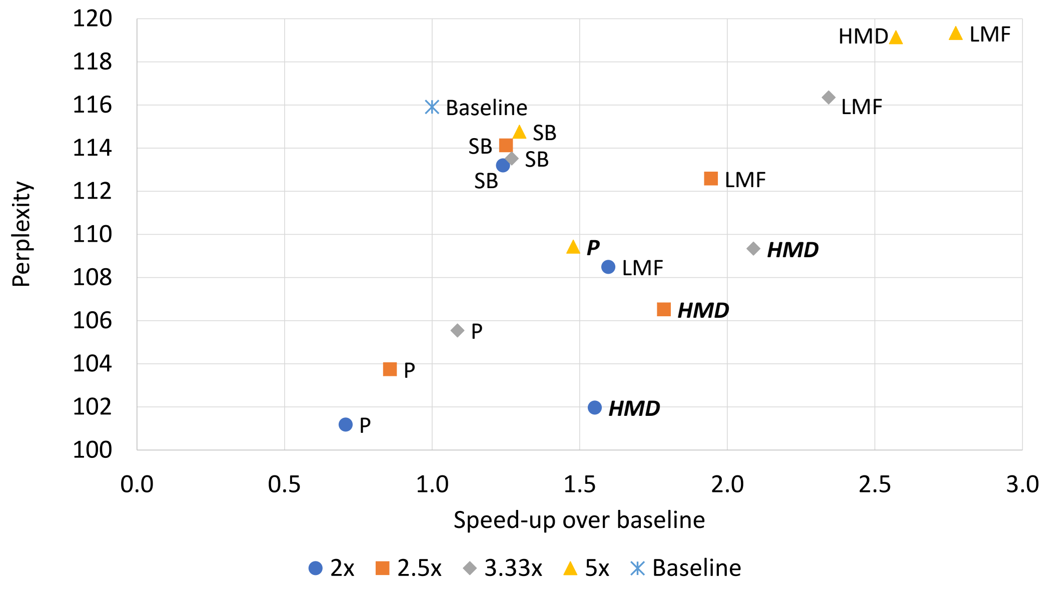

Figure 5. Perplexity vs speedup for PTB-LM (Small) network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values >1 indicate a decrease in inference run-time and values <1 indicate an increase in 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. The best way to view this figure is to focus on a compression point and see how the Pareto curve of perplexity vs speed-up changes as we add HMD. HMD provides a viable alternative to both pruning and LMF at compression factors of 2x, 2.5x and 3.33x.

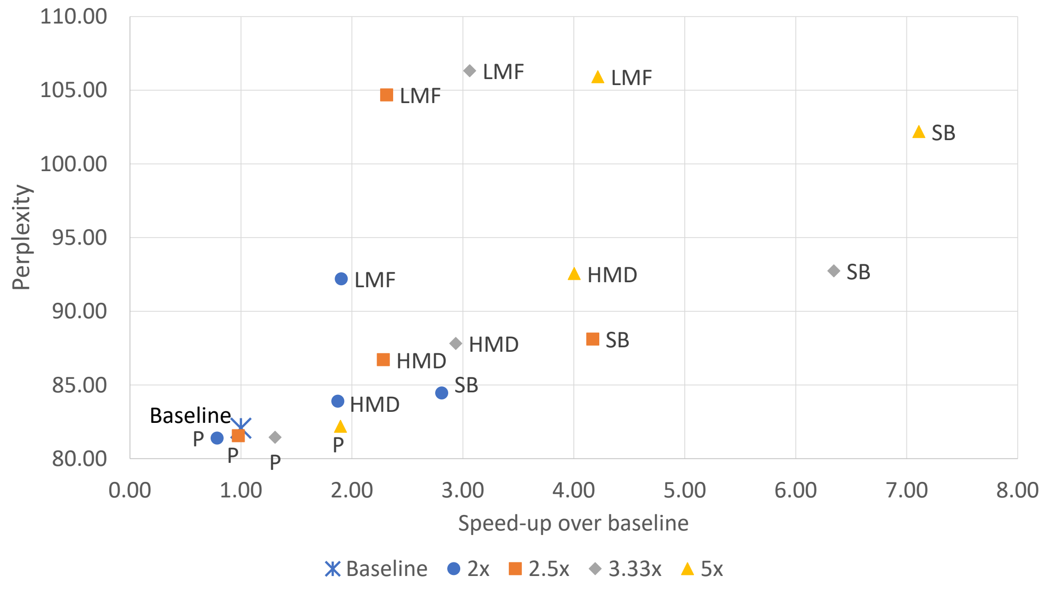

Figure 6. Perplexity vs speedup for PTB-LM (Medium) network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values >1 indicate a decrease in inference run-time and values <1 indicate an increase in 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. The best way to view this figure is to focus on a compression point and see how the Pareto curve of perplexity vs speed-up changes as we add HMD. HMD provides a viable alternative to both pruning and LMF at compression factors of 2x, 2.5x and 3.33x.

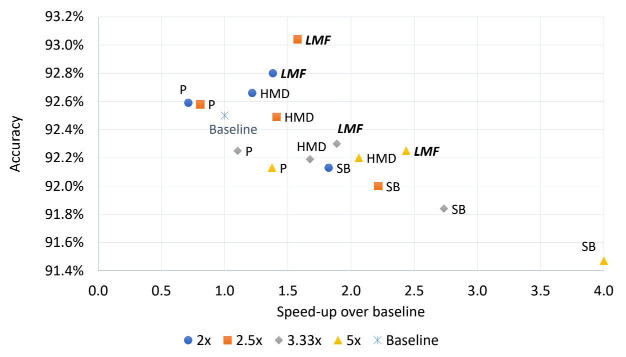

Figure 7. Accuracy vs speedup for a KWS-LSTM network comparing the baseline with a smaller baseline and the baseline compressed using different compression schemes at varying compression factors. Speed-up values > 1 indicate a decrease in inference run-time and values < 1 indicate an increase in inference 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. HMD was developed to act as an alternative to pruning when LMF leads to a loss in accuracy. However, LMF does not lead to loss in accuracy after compression for this benchmark.

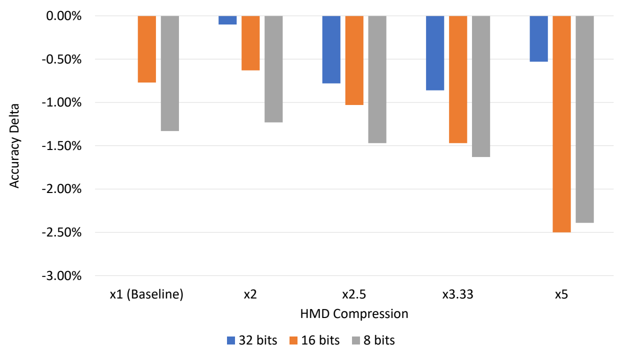

Figure 8. HAR1 accuracy with varying bit precision and compression amounts. Each of the bars show the impact on model accuracy relative to that of the baseline 32-bit accuracy of 91.90%. Precision is varied for different HMD compression factors. The data shows that lowering bit precision can be used in addition to HMD with minimal loss in accuracy up to a 3.33$\times$ compression ratio.

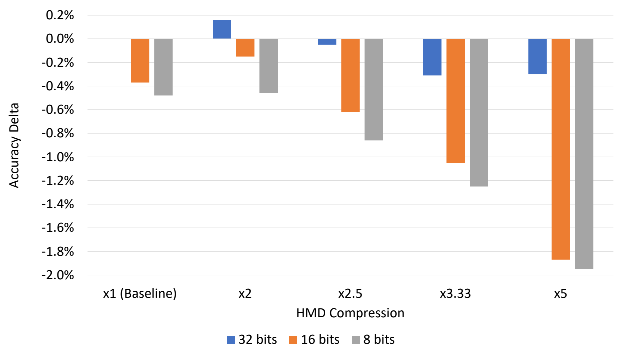

Figure 9. KWSLSTM accuracy with varying bit precision and compression amounts. Each of the bars show the impact on model accuracy relative to that of the baseline 32-bit accuracy of 92.50%. Precision is varied for different HMD compression factors. The data shows that lowering bit precision can be used in addition to HMD with minimal loss in accuracy up to a 3.33$\times$ compression ratio.

compression scheme.

Figure 6 shows the results of compressing the LSTM layers in the PTB (Medium) baseline by $2\times$, $2.5\times$, $3.33\times$ and $5\times$. As described in the previous paragraph, lower the perplexity, better the model. Pruning achieves the same (sometimes better) perplexity than baseline and other compression techniques. LMF leads to significant loss in perplexity for all compression factors while HMD achieves better perplexity than LMF and faster inference run-time than baseline and pruned networks for all compression factors. The preferred choice of compression scheme for different compression factors will depend on whether slight loss in perplexity can be accommodated for faster inference run-time or not. However, HMD still manages to serve as a more viable alternative to pruning than LMF for inference on edge CPUs.

4.2.3 *Key Word Spotting (KWS)*

We use the Google speech commands dataset(Warden, 2018) for KWS task. For our baseline network, we use the smallest LSTM model in (Zhang et al., 2017). The input to the network is 10 MFCC features fed over 25 time steps. The LSTM architecture uses a hidden length of size 118 and achieves an accuracy of 92.50%. The total number of parameters in the network are 62,316.

Figure 7 shows the result of compressing the LSTM layers in baseline by $2\times$, $2.5\times$, $3.33\times$ and $5\times$. Pruning and LMF perform equally well, sometimes performing better than the baseline. Accuracy of HMD compressed baseline is also equivalent to baseline compressed using pruning and LMF. However, the run-time of this network is $7 - 10\%$ slower than LMF compressed network. HMD is an effective substitute to LMF when LMF leads to accuracy degradation. For this network, LMF is able to regain all of the baseline accuracy. As a result, HMD while being an effective compression technique, will not replace LMF as the preferred scheme for compression. Finally, all three compressed networks have better accuracy than the smaller baselines.

5 Discussion

Effectively, HMD acts as an alternative to LMF whenever compression using pruning does not lead to the required run-time benefit and LMF leads to loss in accuracy. HMD has a better accuracy than LMF for most evaluation points, validating the assumption in the paper that rank of a matrix in a RNN is important for better task accuracy. Additionally, HMD has a better accuracy than smaller baseline for most evaluation points.

6 Impact of Quantization

Quantization (Vanhoucke et al., 2011; He et al., 2016) is another popular technique for compressing neural networks. It is orthogonal to the compression techniques discussed previously; prior work has shown that both pruning (Han et al., 2016) and BCD (Sindhwani et al., 2015) can benefit from quantization. To test whether HMD is compatible with quantization, we use the quantization flow provided by the authors of (He et al., 2016). We quantized the LSTM cells in the baseline and the HMD compressed networks to 16- and 8-bits floating point representations to test the robustness of HMD under reduced bit-precision. Figures 8 and 9 show that quantization works well with HMD. The HAR1 and KWS-LSTM networks compressed using HMD can be further compressed using quantization for most HMD compression factors. Additionally, we observe that networks compressed with HMD tend to have better accuracy than networks compressed with quantization. For example, the HAR1 baseline quantized to 16 bits has a lower accuracy than HAR1 network compressed by a factor of 2 using HMD. This might be due to the regularization effect of having rank-1 constraint in the lower half of the matrix. These results were achieved with minimal hyper-parameter exploration and without exploring other robust RNN quantization techniques (Xu et al., 2018), but are sufficient to conclude that HMD is generally compatible with quantization with relatively little accuracy loss.

7 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

Chen, T., Lin, J., Lin, T., Han, S., Wang, C., and Zhou, D. Adaptive mixture of low-rank factorizations for compact neural modeling. *Advances in neural information processing systems (CDNNRIA workshop)*, 2018. URL `https://openreview.net/forum?id=B1eHgu-Fim`.

Cheng, Y., Yu, F. X., Feris, R. S., Kumar, S., Choudhary, A., and Chang, S. An exploration of parameter redundancy in deep networks with circulant projections. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2857–2865, Dec 2015. doi: 10.1109/ICCV.2015.327.

Courbariaux, M. and Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL `http://arxiv.org/abs/1602.02830`.

Ding, C., Liao, S., Wang, Y., Li, Z., Liu, N., Zhuo, Y., Wang, C., Qian, X., Bai, Y., Yuan, G., Ma, X., Zhang, Y., Tang, J., Qiu, Q., Lin, X., and Yuan, B. 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, pp. 395–408, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4952-9. doi: 10.1145/3123939.3124552. URL `http://doi.acm.org/10.1145/3123939.3124552`.

Ding, C., Ren, A., Yuan, G., Ma, X., Li, J., Liu, N., Yuan, B., and Wang, Y. 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, pp. 353–358, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5724-1. doi: 10.1145/3194554.3194625. URL `http://doi.acm.org/10.1145/3194554.3194625`.

Grachev, A. M., Ignatov, D. I., and Savchenko, A. V. Neural networks compression for language modeling. In Shankar, B. U., Ghosh, K., Mandal, D. P., Ray, S. S., Zhang, D., and Pal, S. K. (eds.), *Pattern Recognition and Machine Intelligence*, pp. 351–357, Cham, 2017. Springer International Publishing. ISBN 978-3-319-69900-4.

Hammerla, N. Y., Halloran, S., and Ploetz, T. Deep, convolutional, and recurrent models for human activity recognition using wearables. *IJCAI 2016*, 2016.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016.

He, Q., Wen, H., Zhou, S., Wu, Y., Yao, C., Zhou, X., and Zou, Y. Effective quantization methods for recurrent neural networks. *CoRR*, abs/1611.10176, 2016. URL `http://arxiv.org/abs/1611.10176`.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016. URL http://arxiv.org/abs/1609.07061.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18(1):6869–6898, January 2017. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=3122009.3242044.

Kuchaiev, O. and Ginsburg, B. Factorization tricks for LSTM networks. *CoRR*, abs/1703.10722, 2017. URL http://arxiv.org/abs/1703.10722.

Li, Z., Wang, S., Ding, C., Qiu, Q., Wang, Y., and Liang, Y. Efficient recurrent neural networks using structured matrices in fpgas. *CoRR*, abs/1803.07661, 2018. URL http://arxiv.org/abs/1803.07661.

Lin, Z., Courbariaux, M., Memisevic, R., and Bengio, Y. Neural networks with few multiplications. *ArXiv e-prints*, abs/1510.03009, October 2015. URL http://arxiv.org/abs/1510.03009.

Lu, Z., Sindhwani, V., and Sainath, T. N. Learning compact recurrent neural networks. *CoRR*, abs/1604.02594, 2016. URL http://arxiv.org/abs/1604.02594.

Narang, S., Undersander, E., and Diamos, G. F. Block-sparse recurrent neural networks. *CoRR*, abs/1711.02782, 2017. URL http://arxiv.org/abs/1711.02782.

Ordez, F. J. and Roggen, D. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1), 2016. ISSN 1424-8220. doi: 10.3390/s16010115. URL http://www.mdpi.com/1424-8220/16/1/115.

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

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. 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*, pp. 6655–6659, May 2013. doi: 10.1109/ICASSP.2013.6638949.

Sindhwani, V., Sainath, T., and Kumar, S. Structured transforms for small-footprint deep learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 3088–3096. Curran Associates, Inc., 2015.

Thakker, U., Beu, J. G., Gope, D., Zhou, C., Fedorov, I., Dasika, G., and Mattina, M. Compressing rnns for iot devices by 15-38x using kronecker products. *CoRR*, abs/1906.02876, 2019a. URL http://arxiv.org/abs/1906.02876.

Thakker, U., Dasika, G., Beu, J. G., and Mattina, M. Measuring scheduling efficiency of rnns for NLP applications. *CoRR*, abs/1904.03302, 2019b. URL http://arxiv.org/abs/1904.03302.

Thakker, U., Fedorov, I., Beu, J. G., Gope, D., Zhou, C., Dasika, G., and Mattina, M. Pushing the limits of rnn compression. *ArXiv*, abs/1910.02558, 2019c.

Thomas, A., Gu, A., Dao, T., Rudra, A., and Ré, C. Learning compressed transforms with low displacement rank. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 9066–9078. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8119-learning-compressed-transforms-with-low-displacement-rank.pdf.

Tjandra, A., Sakti, S., and Nakamura, S. Compressing recurrent neural network with tensor train. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 4451–4458. IEEE, 2017.

Trefethen, L. and Bau, D. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1997.

Vanhoucke, V., Senior, A., and Mao, M. Z. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

Wang, S., Li, Z., Ding, C., Yuan, B., Qiu, Q., Wang, Y., and Liang, Y. C-lstm: Enabling efficient lstm using structured compression techniques on fpgas. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, pp. 11–20, New York, NY, USA, 2018a. ACM. ISBN 978-1-4503-5614-5. doi: 10.1145/3174243.3174253. URL http://doi.acm.org/10.1145/3174243.3174253.

Wang, Y., Ding, C., Li, Z., Yuan, G., Liao, S., Ma, X., Yuan, B., Qian, X., Tang, J., Qiu, Q., and Lin, X. Towards ultra-high performance and energy efficiency of deep learning systems: An algorithm-hardware co-optimization framework. *CoRR*, abs/1802.06402, 2018b. URL http://arxiv.org/abs/1802.06402.

Warden, P. Speech commands: A dataset for limited-vocabulary speech recognition. *CoRR*, abs/1804.03209, 2018. URL http://arxiv.org/abs/1804.03209.

Xu, C., Yao, J., Lin, Z., Ou, W., Cao, Y., Wang, Z., and Zha, H. Alternating multi-bit quantization for recurrent neural networks. *CoRR*, abs/1802.00150, 2018. URL http://arxiv.org/abs/1802.00150.

Yang, Z., Dai, Z., Salakhutdinov, R., and Cohen, W. W. Breaking the softmax bottleneck: A high-rank RNN language model. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkwZSG-CZ.

Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL http://arxiv.org/abs/1409.2329.

Zhang, Y., Suda, N., Lai, L., and Chandra, V. Hello edge: Keyword spotting on microcontrollers. *CoRR*, abs/1711.07128, 2017. URL http://arxiv.org/abs/1711.07128.

Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained ternary quantization. *CoRR*, abs/1612.01064, 2016. URL http://arxiv.org/abs/1612.01064.

Zhu, M. and Gupta, S. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv e-prints*, art. arXiv:1710.01878, October 2017.