FINN-L: Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs

Vladimir Rybalkin,
Muhammad Mohsin Ghaffar and Norbert Wehn
Microelectronic Systems Design Research Group
University of Kaiserslautern, Germany
{rybalkin, ghaffar, wehn}@eit.uni-kl.de

Alessandro Pappalardo,
Giulio Gambardella and Michaela Blott
Xilinx Research Labs
{alessand, giuliog, mblott}@xilinx.com

***Abstract*—It is well known that many types of artificial neural networks, including recurrent networks, can achieve a high classification accuracy even with low-precision weights and activations. The reduction in precision generally yields much more efficient hardware implementations in regards to hardware cost, memory requirements, energy, and achievable throughput. In this paper, we present the first systematic exploration of this design space as a function of precision for Bidirectional Long Short-Term Memory (BiLSTM) neural network. Specifically, we include an in-depth investigation of precision vs. accuracy using a fully hardware-aware training flow, where during training quantization of all aspects of the network including weights, input, output and in-memory cell activations are taken into consideration. In addition, hardware resource cost, power consumption and throughput scalability are explored as a function of precision for FPGA-based implementations of BiLSTM, and multiple approaches of parallelizing the hardware. We provide the first open source HLS library extension of FINN [1] for parameterizable hardware architectures of LSTM layers on FPGAs which offers full precision flexibility and allows for parameterizable performance scaling offering different levels of parallelism within the architecture. Based on this library, we present an FPGA-based accelerator for BiLSTM neural network designed for optical character recognition, along with numerous other experimental proof points for a Zynq UltraScale+ XCZU7EV MPSoC within the given design space.**

I. Introduction

Recurrent Neural Networks (RNNs) and in particular Long Short-Term Memory (LSTM) have achieved state-of-the-art classification accuracy in many applications such as language modeling [2], machine translation [3], speech recognition [4], and image caption generation [5]. However, high classification accuracy comes at high compute, storage, and memory bandwidth requirements, which makes their deployment particularly challenging, especially for energy-constrained platforms such as for portable devices. Furthermore, many applications have hard real-time constraints such as mobile robots, hearing aids and autonomous vehicles.

Compared to feed-forward Neural Networks (NNs), LSTM networks are especially challenging as they require state keeping in between processing steps. This has various adversary effects. First of all, extra processing is required of the recurrent connections along with the "feed-forward" input activations. Additionally, even though the required state memory is not particularly large, the state keeping creates data dependencies to previous steps, which forces sequentialization of the processing in parts and limits the degrees of parallelism that can be leveraged in customizable architectures and that makes multi-core general-purpose computing platforms inefficient. Many techniques have been proposed to alleviate the compute and storage challenges described above. Among the most common ones are pruning [6], [7], [8], and quantization (see Section II for details) or a combination thereof [9]. All of them are based on the typically inherent redundancy contained within the NNs, meaning that the number of parameters and precision of operations can be significantly reduced without affecting accuracy. Within this paper, we focus on the latter method, namely reducing compute cost and storage requirements through reduction of precision in the leveraged data types, whereby we leverage Field-Programmable Gate Arrays (FPGAs) as they are they only computing platform that allows for customization of pipelined compute data paths and memory subsystems at the level of data type precision, in a programmable manner. As such they can take maximum advantage of the proposed optimization techniques.

Within this paper, we extend a vast body of existing research on implementation of quantized NN accelerators for standard CNNs [10] to recurrent models. The novel contributions are:

- We conduct the first systematic exploration of the design space comprised of hardware computation cost, storage cost, power and throughput scalability as a function of precision for LSTM and Bidirectional LSTM (BiLSTM) in particular.
- We investigate the effects of quantization of weights, input, output, recurrent, and in-memory cell activations during training.
- We cross-correlate achievable performance with accuracy for a range of precisions to identify optimal performance-accuracy trade-offs with the design space.
- To the best of our knowledge, we present the first hardware implementation of binarized (weights and activations constrained to 1 and 0) LSTM and BiLSTM in particular.
- Last but not least, we provide the first open source HLS library extension of FINN [1] for parameterizable hardware architectures of LSTM layers on FPGAs, which provides full precision flexibility, allows for parameterizable performance scaling and supports a full training flow

978-1-5386-8517-4/18/$31.00 ©2018 IEEE
DOI 10.1109/FPL.2018.00024

that allows to train the network on new datasets.

The paper is structured as follows: in Section II we review existing research on quantized training and FPGA-based implementations of RNNs. In Section III we review the LSTM algorithm. We describe the training setup and procedure in Torch and PyTorch in Section IV. Section V provides the details of the hardware design. Finally, Section VI presents the results and Section VII concludes the paper.

II. Related work

A. Quantized Neural Networks

The most extreme quantization approach is binarization that results in Binarized Neural Networks (BNNs) with binarized weights only [11] or with both weights and activations quantized to 1-bit (usually {-1,1} during training and {0,1} at inference) that was firstly proposed in [12], [13]. Compared with the 32-bit full-precision counterpart, binarized weights drastically reduce memory size and accesses. At the same time, BNNs significantly reduce the complexity of hardware by replacing costly arithmetic operations between real-value weights and activations with cheap bit-wise XNOR and bit-count operations, which altogether leads to much acceleration and great increase in power efficiency. It has been shown that even 1-bit binarization can achieve reasonably good performance in some applications. In [14] they proposed to use a real-value scaling factor to compensate classification accuracy reduction due to binarization and achieved better performance than pure binarization in [12]. However, binarization of both weights and activations can lead to undesirable and non-acceptable accuracy reduction in some applications compared with the full-precision networks. To bridge this gap, recent works employ quantization with more bits like ternary {-1,0,1} [15], [16] and low bit-width [17], [18], [19] networks that achieve better performance bringing a useful trade-off between implementation cost and accuracy.

Among all existing works on quantization and compression, most of them focus on CNNs while less attention has been paid to RNNs. The quantization of the latter is even more challenging. [17], [19], [20] have shown high potential for networks with quantized multi-bit weights and activations. Recently, [21] showed that LSTM with binarized weights only can even surpass full-precision counterparts. [22] has addressed the problem by quantizing both weights and activations. They formulate the quantization as an optimization problem and solve it by the binary search tree. Using language models and image classification task, they have demonstrated that with 2-bit quantization of both weights and activations, the model suffers only a modest loss in the accuracy, while 3-bit quantization was on par with the original 32-bit floating point model.

Unlike other publications that mostly give qualitative and analytical estimations of advantages of using quantization with respect to general-purpose computing platforms only, we quantify the savings in resource utilization, gain in throughput and power savings using actual FPGA-based hardware implementations. Furthermore, we propose a new approach when the recurrent and output activations are quantized differently that keeps resource utilization of a hidden layer, while providing higher accuracy with respect to uniform approach. To the best of our knowledge, we are the first investigating influence of precision on classification accuracy on Optical Character Recognition (OCR) using fully hardware-aware training flow, where during training quantization of all aspects of the LSTM network including weights, input, output, recurrent, and in-memory cell activations are taken into consideration.

B. Recurrent Neural Networks on FPGA

1) With Quantization:

In [23], authors presented an FPGA-based hardware implementation of pre-trained LSTM for character level language modeling. They used 16-bit quantization for weights and input data, both of which were stored in off-chip memory. Intermediate results were also sent back to off-chip memory in order to be read for computing next layer or next timestep output that has been identified as a performance bottleneck. The following works tried to de-intensify off-chip memory communication by storing a model and intermediate results on chip. An LSTM network for a learning problem of adding two 8-bit numbers has been implemented in [24]. The design could achieve higher throughput than [23], while using 18-bit precision for activations and weights but stored in on-chip memory. [25] proposed a real-time, low-power FPGA-based speech-recognition system with higher energy efficiency than GPU. The FPGA-based implementation of LSTM used 8-bit activations and 6-bit weights stored in on-chip memory. They also experimented with 4-bit weights, however with significant accuracy degradation. [26] implemented an LSTM network for the speech recognition application. Although using higher precision 32-bit floating-point weights stored in off-chip memory, they could achieve higher throughput than previous works, because of smart memory organization with on-chip ping-pong buffers to overlap computations with data transfers. FP-DNN - an end-to-end framework proposed in [27] that takes TensorFlow-described network models as input, and automatically generates hardware implementations of various NNs including LSTM on FPGA using RTL-HLS hybrid templates. They used RTL for implementing matrix multiplication parts and to express fine-grained optimizations, while HLS to implement control logic. The FPGA implementation made use of off-chip memory for storing 16-bit fixed-point format weights and intermediate results, and ping-pong double buffers to overlap communication with computation.

The current work is based on existing architecture proposed in [28], where the authors presented the first hardware architecture designed for BiLSTM for OCR. The architecture was implemented with 5-bit fixed-point numbers for weights and activations that allowed to store all intermediate results and parameters in on-chip memory. The current research, differs from the previous works by combining both state-of-the-art quantization techniques in training and advanced hardware architecture. Non of the mentioned works targeting hardware implementation was addressing efficient hardware-aware training. The architecture is on-chip memory centric that avoids using off-chip memory for storing weights like in [23], [26]. Efficient quantization at training allows for efficient on-chip implementation even larger networks because of reduced precision without effecting accuracy rather than using off-chip bandwidth and higher precision like in [27]. Non of the previous works have demonstrated efficient support for a wide range of precisions. We provide an open source HLS library that supports a wide scope of precisions of weights and activations ranging from multi-bit to fully binarized version. To the best of our knowledge, we preset the first fully binarized implementation of LSTM NN in hardware. Finally, the library is modular and optimized, providing customized functions to the various layer types rather than only matrix multiplication functions. In contrast to [27], FINN hardware library is purely implemented using HLS rather than RTL-HLS hybrid modules.

2) *With Pruning and Quantization:*

For the sake of completeness, we also mention architectures targeting sparse models unlike our design that supports dense models. A design framework for a hybrid NN (CNN + LSTM) that uses configurable IPs together with a design space exploration engine was presented in [29]. The proposed HLS IPs implement MAC operations. They used pruning and 12-bit weights and 16-bit fixed-point activations. ESE [30] presented an efficient speech recognition engine implementing LSTM on FPGA that combined both pruning and 12-bit compact data types. The design can operate on both dense and sparse models. [31] proposed a compression technique for LSTM that reduces not only the model size, but also eliminates irregularities of compression and memory accesses. The datapath and activations were quantized to 16 bits. They also presented a framework called C-LSTM to automatically map LSTM onto FPGAs.

Our approach is based on retrained optimization based on quantization that avoids a use of redundant high precision calculations. In contrast to aforementioned works our architecture is on-chip memory oriented and designed to operate on dense models. Architecture presented in [31] is designed to operate on compressed weight matrices with block-circulant matrix based structured compression technique and cannot process not compressed models. Non of those works explored very low bit-width weights and activations.

III. LSTM Theory

For the sake of clarity, we review the basic LSTM algorithm, earlier presented in [32], [33].

$$
\begin{aligned}
\boldsymbol{I}^t &= l(\boldsymbol{W}_I\boldsymbol{x}^t + \boldsymbol{R}_I\boldsymbol{y}^{t-1} + \boldsymbol{b}_I)\\
\boldsymbol{i}^t &= \sigma(\boldsymbol{W}_i\boldsymbol{x}^t + \boldsymbol{R}_i\boldsymbol{y}^{t-1} + \boldsymbol{p}_i \odot \boldsymbol{c}^{t-1} + \boldsymbol{b}_i)\\
\boldsymbol{f}^t &= \sigma(\boldsymbol{W}_f\boldsymbol{x}^t + \boldsymbol{R}_f\boldsymbol{y}^{t-1} + \boldsymbol{p}_f \odot \boldsymbol{c}^{t-1} + \boldsymbol{b}_f)\\
\boldsymbol{c}^t &= \boldsymbol{i}^t \odot \boldsymbol{I}^t + \boldsymbol{f}^t \odot \boldsymbol{c}^{t-1}\\
\boldsymbol{o}^t &= \sigma(\boldsymbol{W}_o\boldsymbol{x}^t + \boldsymbol{R}_o\boldsymbol{y}^{t-1} + \boldsymbol{p}_o \odot \boldsymbol{c}^{t} + \boldsymbol{b}_o)\\
\boldsymbol{y}^t &= \boldsymbol{o}^t \odot h(\boldsymbol{c}^t)
\end{aligned}
\qquad (1)
$$

The LSTM architecture is composed of a number of recurrently connected "memory cells". Each memory cell is composed of three multiplicative gating connections, namely input, forget, and output gates ($\boldsymbol{i}$, $\boldsymbol{f}$, and $\boldsymbol{o}$, in Eq. 1); the function of each gate can be interpreted as write, reset, and read operations, with respect to the cell internal state ($\boldsymbol{c}$). The gate units in a memory cell facilitate the preservation and access of the cell internal state over long periods of time. The peephole connections ($\boldsymbol{p}$) are supposed to inform the gates of the cell about its internal state. However, they can be optional and according to some research even redundant [34]. There are recurrent connections from the cell output ($\boldsymbol{y}$) to the cell input ($\boldsymbol{I}$) and the three gates. Eq. 1 summarizes formulas for LSTM network forward pass. Rectangular input weight matrices are shown by $\boldsymbol{W}$, and square weight matrices by $\boldsymbol{R}$. $\boldsymbol{x}$ is the input vector, $\boldsymbol{b}$ refers to bias vectors, and t denotes the time (and so t-1 refers to the previous timestep). Activation functions are point-wise non-linear functions, that is *logistic sigmoid* ($\frac{1}{1+e^{-x}}$) for the gates (σ) and *hyperbolic tangent* for input to and output from the node (l and h). Point-wise vector multiplication is shown by $\odot$ (equations adapted from [35]).

Bidirectional RNNs were proposed to take into account impact from both the past and the future status of the input signal by presenting the input signal forward and backward to two separate hidden layers both of which are connected to a common output layer that improves overall accuracy [36].

Connectionist Temporal Classification (CTC) is the output layer used with LSTM networks designed for sequence labelling tasks. Using the CTC layer, LSTM networks can perform the transcription task without requiring pre-segmented input (see [37] for more details and related equations).

IV. Training

A. *Topology*

For our experiments, we implemented a quantized version of BiLSTM NN on Torch7 [38] and PyTorch [39] frameworks. For Torch, our Quantized BiLSTM (QBiLSTM) is based on RNN library [40] that provides full-precision implementation of various RNN models. For PyTorch, our quantized implementation, called *pytorch-quantization*, is based on official full-precision ATen [41] accelerated PyTorch BiLSTM implementation, on top of which we implemented an OCR training pipeline, called *pytorch-ocr*. Both *pytorch-quantization* [42] and *pytorch-ocr* [43] are open-source.

TABLE I: Training. General information.

Number of images in training set, $\boldsymbol{T}_{train}$	400
Number of images in test set, $\boldsymbol{T}_{test}$	100
Number of pixels per column (Input layer), $\boldsymbol{I}$	32
Max number of columns per image, $\boldsymbol{C}$	732
Number of LSTM memory cells in a hidden layer, $\boldsymbol{H}$	128
Number of units in the output layer, $\boldsymbol{K}$	82
Minibatch size, $\boldsymbol{B}$	32
Maximum number of epochs, $\boldsymbol{E}$	4000
Learning rate, $\boldsymbol{L}$	1e-4

We use OCR plain-text dataset of $\boldsymbol{T}_{train}$ images for training and $\boldsymbol{T}_{test}$ images for test. Each sample is a gray-scale image

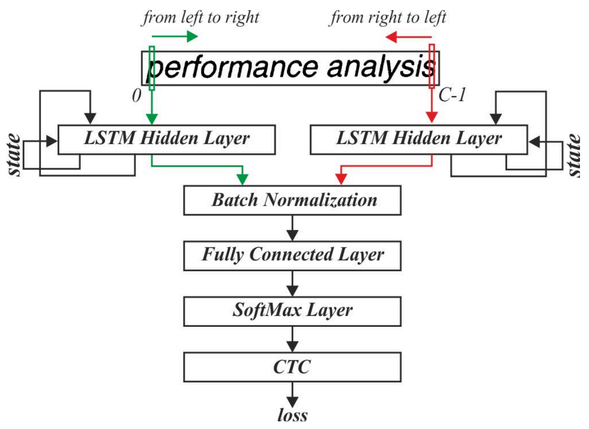

Fig. 1: Network topology used during training. Only forward path is shown.

of a text-line with a fixed height of $\boldsymbol{I}$ pixels and width up to $\boldsymbol{C}$ pixels. In LSTM terminology, height and width corresponds, respectively, to the input size of the LSTM cell and to the length of the input sequence.

In all experiments, we adopt the following topology:

- An input layer that feeds images column by column along the width dimension $\boldsymbol{C}$.
- A single BiLSTM layer, composed of two sub-layers, each comprising of a distinct set of $\boldsymbol{H}$ LSTM cells without peepholes. The two sub-layers go through the input image, respectively, from left-to-right and from right-to-left, each generating an output sub-sequence of length $\boldsymbol{C}$ and feature size $\boldsymbol{H}$. At each timestep, each gate of each memory cell operates on $1+\boldsymbol{I}+\boldsymbol{H}$ values, comprising of a single bias value, $\boldsymbol{I}$ pixels and the $\boldsymbol{H}$ output values computed from the previous column in the sequence, according to Eq. 1. The BiLSTM layer's output sequence is composed of the right-to-left output sub-sequence taken in reverse order and concatenated with the left-to-right output sub-sequence along the feature dimension.
- A batch normalization step with minibatch of size $\boldsymbol{B}$ on the output sequence of the BiLSTM layer, to speed up the training.
- A fully connected layer (with dropout 0.2 in Torch) mapping each column of the normalized output sequence to a vector of $\boldsymbol{K}$ features, one for each of the symbols in the dataset's alphabet including a blank.
- A softmax layer, to interpret the output of the fully connected layer as a vector of probabilities over the alphabet.
- A CTC layer as loss function, using a GPU-accelerated implementation from Baidu Research [44].

All training experiments were running on GPUs. After running a series of experiments, we take the best-effort validation accuracy over the multiple experiments and over the training frameworks (Torch and PyTorch) to evaluate the efficiency of the configurations. The classification accuracy is based on Character Error Rate (CER) computed as the Levenshtein distance [45] between a decoded sequence and a ground truth. In Section VI, we list accuracy of models trained at maximum over $\boldsymbol{E}$ epochs using a SGD learning rule with learning rate $\boldsymbol{L}$.

B. *Quantization*

Our quantization approaches are based on those presented in [14], [17], [18]. For multi-bit weights and activations, we adopt Eq. 2a as a quantization function, where $\boldsymbol{x}$ is a full-precision activation or weight, $\boldsymbol{x_q}$ is the quantized version, $\boldsymbol{k}$ is a chosen bit-width, and $\boldsymbol{f}$ is a number of fraction bits. For multi-bit weights, the choice of parameters is shown in Eq. 2b. For multi-bit activations, we assume that they have passed through a bounded activation function before the quantization step, which ensures that the input to the quantization function is either $\boldsymbol{x} \in [0,1)$ after passing through a *logistic sigmoid* function, or $\boldsymbol{x} \in [-1,1)$ after passing through a *tanh* function. As for the quantization parameters, we use Eq. 2b after the *tanh* and Eq. 2c after the *logistic sigmoid.*

$$\boldsymbol{x_q} = clip(round(\boldsymbol{x} \cdot 2^{\boldsymbol{f}}) \cdot 2^{-\boldsymbol{f}}, min, max), \text{where} \tag{2a}$$

$$\begin{aligned} \boldsymbol{f} &= \boldsymbol{k} - 1 \\ min &= -(2^{(\boldsymbol{k}-\boldsymbol{f}-1)}) = -1 \\ max &= -min - 2^{-\boldsymbol{f}} = 1 - 2^{-\boldsymbol{f}} \end{aligned} \tag{2b}$$

$$\begin{aligned} \boldsymbol{f} &= \boldsymbol{k} \\ min &= 0 \\ max &= 2^{\boldsymbol{k}-\boldsymbol{f}} - 2^{-\boldsymbol{f}} = 1 - 2^{-\boldsymbol{f}} \end{aligned} \tag{2c}$$

In the case of binarized activations $\boldsymbol{a_b}$, we apply a *sign* function as the quantization function [14], as shown in Eq. 3. In the case of binarized weights $\boldsymbol{w_b}$, on top of the sign function we scale the result by a constant factor, as shown in Eq. 4.

$$\boldsymbol{a_b} = sign(\boldsymbol{x}) \tag{3}$$

$$\begin{aligned} \boldsymbol{w_b} &= sign(\boldsymbol{x}) \cdot scaling\_factor \\ scaling\_factor &= \frac{1}{sqrt(\boldsymbol{H}+\boldsymbol{I})} \end{aligned} \tag{4}$$

Inspired by [14], we also experimented with using the mean of the absolute values of the full-precision weights as a layer-wise scaling factor for the binarized weights. However, it didn't show any improvements in classification accuracy with respect to the constant scaling factor described earlier.

In our experiments, we also considered different approaches for training quantized BiLSTM: In Torch, quantized model are trained from scratch. In PyTorch, quantized model are trained from scratch as well, but we leave out the quantization of the internal activations to a second short ($E = 40$) retraining step, during which we also fold the parameters of the batch-norm layer into the full-precision shadow version of the fully connected layer, forcing the quantized version to re-converge.

Avoiding the overhead of computing the quantized internal activations during the main training step grants some speed-up with no impact on accuracy. We run all experiments with all in-memory cell activations and weights of the fully connected output layer quantized to 8 bits, while varying quantization of weights, input, output, and recurrent activations.

V. Architecture for Inference

In the following, we used an existing architecture proposed in [28]. The design has been modified for parameterizable precision and parallelization, and optimized for higher frequency. The hardware accelerator is implemented on Xilinx Zynq UltraScale+ MPSoC ZCU104. The software part is running on Processing System (PS) and hardware part is implemented in Programmable Logic (PL). The software part plays an auxiliary role. It reads all images required for the experiment from SD card to DRAM and iteratively configures hardware blocks for all images. As soon as the last image has been processed, the software computes a Character Error Rate (CER) of the results based on Levenshtein distance [45] with respect to a reference.

The constructed network at inference differs from the one used during training. At inference, the batch normalization layer is merged with the fully connected layer into output layer. We avoid using a softmax activation layer and replace the CTC with a hardware implementation of the greedy decoder that maps each output vector of probabilities to a single symbol by taking a maximum over it.

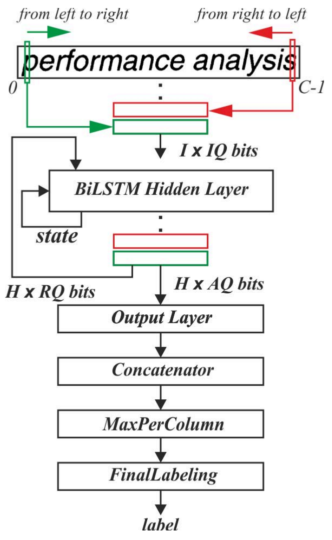

Fig. 2: Network topology used during inference.

The architecture of the hidden layer processes inputs corresponding to the left-to-right and right-to-left timesteps in an interleaved fashion, which avoids a duplication of the hidden layer in the case of bidirectional network. Additionally, processing independent inputs (from different directions) in an interleaved manner allows to keep the pipeline always busy, otherwise the pipeline will be part time idle because of the recurrent dependence. The same approach can be applied to uni-directional RNN that will allow for processing separate samples alike in a batch. Accordingly, the output layer is implemented to process the interleaved outputs corresponding to different directions. However, the $MaxPerColumn$ has to operate on outputs that correspond to the same timestep (column). For that the $Concatenator$ buffers the intermediate results until the first matching column that is a $C/2$ in an input sequence, and outputs the summed values that correspond to the same column but from different directions. The detailed explanation of the architecture can be found in [28].

A. Parameterizable Unrolling

The parameterizable architecture allows to apply coarse-grained parallelization on a level of LSTM cells, and fine-grained parallelization on a level of dot products, as shown in Fig. 3. The former, indicated as PE unrolling, allows the concurrent execution of different LSTM cells, while the latter, folds the execution of a single cell in multiple cycles. This flexibility allows for tailoring of parallelism according to the available resources on the target device. Increasing PE will increase parallelism, leading to higher hardware usage and decreased latency, while SIMD folding will decrease parallelism, thus scaling down hardware requirements at the expense of slower performance.

When PE unrolling is performed, the LSTM cell will be duplicated PE times, generating for each clock cycle PE output activations. When SIMD folding is applied, each gate will receive a portion of the input column each cycle, namely SIMD_INPUT and a portion of the recurrent state SIMD_RECURRENT. Each gate will generate the output of the dot product every F_s cycles, evaluated as $F\_s = \frac{I}{SIMD\_INPUT} = \frac{H}{SIMD\_RECURRENT}$.

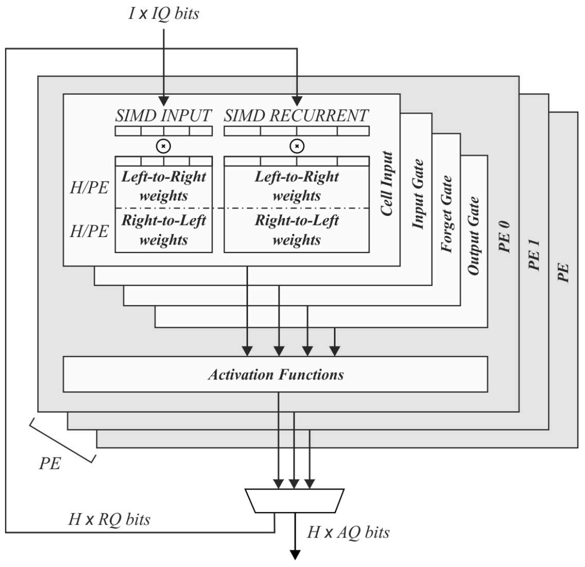

Fig. 3: LSTM Cell internal structure with SIMD and PE folding

VI. Results

The ZCU104 board features a XCZU7EV device, featuring 624 BRAMs, 96 URAMs, 1728 DSP48, 230 kLUTs and 460 kFFs. The HLS synthesis and place and route have been performed by Vivado HLS Design Suite 2017.4.

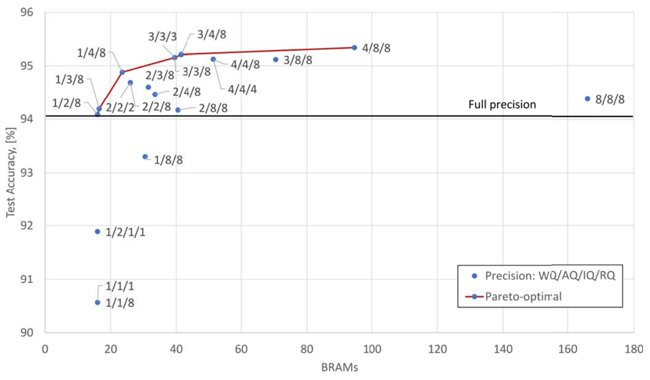

Fig. 4: Pareto-frontier for Accuracy against BRAMs

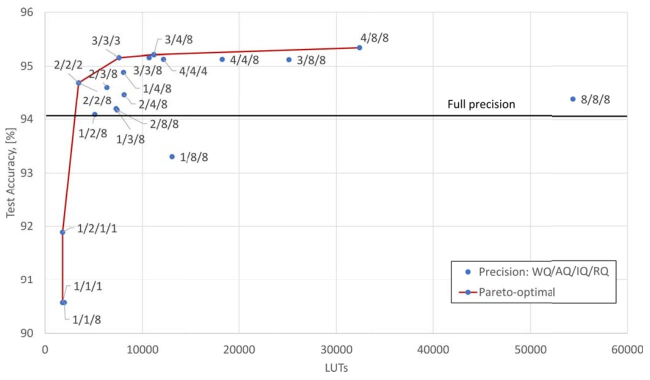

Fig. 5: Pareto-frontier for Accuracy against LUTs

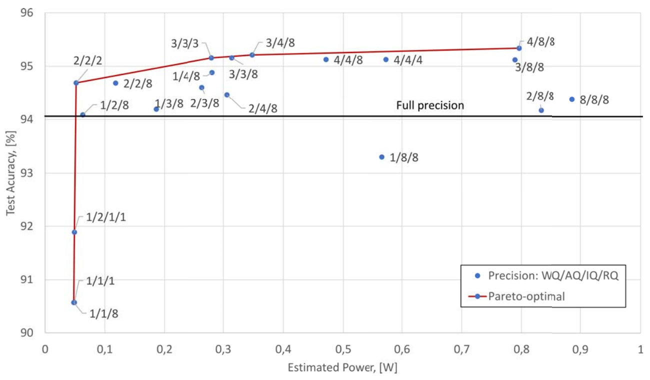

Fig. 6: Pareto-frontier for Accuracy against estimated Power consumption

A. Accuracy, hardware cost, and power vs. precision

The following experiments show the analysis of OCR accuracy depending on precision, and corresponding hardware costs and power consumption of a single instance of the accelerator. Each point represents a combination of number of bits used for representation of weights - WQ, output activations - AQ, input activations - IQ, and recurrent activation - RQ, if different from AQ. In the experiments, the parallelism of the hardware accelerator has been kept constant, with only single LSTM cell instantiated e.g., `PE`=1 and full SIMD width e.g., `SIMD_RECURRENT`=H, `SIMD_INPUT`=I, meaning all inputs and recurrent activations computed in parallel. Target frequency has been set to 200 MHz. All resources are given after post-placement and routing. Fig. 4 plots the achieved test accuracy over memory blocks for different precision. Memory blocks are counted as a sum of #BRAM36 used and 4 times #URAMs used (URAMs provide exactly 4x the capacity to BRAMs). Fig. 5 illustrates the achieved test accuracy over the LUT utilization, while Fig. 6 shows test accuracy over the vector-less activity estimated power consumption of the LSTM accelerator implemented in PL. [1]

B. Throughput scalability vs. precision

According to Eq. 1, a complexity of a single LSTM cell is $2 \times 4 \times (H + I) + 8$, where two comes from multiplication and addition counted as separate operations, four comes from number of gates including input to a memory cell, and eight is a number of point-wise operations. The computed number of operations has to be repeated for each cell and for each timestep of an input sequence. As the proposed architecture implements BiLSTM, the number of operations has to be doubled. The final formula is: $[2\times4\times(H+I)+8]\times H\times2\times C$. The same applies to the output layer: $[2\times(2\times H)+1]\times K\times C$, where $2\times H$ gives a number of concatenated outputs from the hidden layers to each unit of the output layer. We neglect a complexity of the rest of the network. Fig. 7 shows a measured on PYNQ-Z1 throughput for different precisions. The inference has been executed on a single image, 520 columns wide, and the accelerator running at 100 MHz [47]. Due to the limited available resources on PYNQ-Z1, the highest precision solution has been scaled down in parallelism with $F\_s = 8$, while the lowest precision features $PE = 1$, having 8 times parallelism w.r.t. the 4/4/8 configuration. Fig. 8 shows the estimated throughput on ZCU104 running at 266 MHz, achieved by instead implementing multiple accelerators, where each instance is a complete network depicted in Fig. 2. This type of scaling enables parallel execution over multiple images. All configurations have been successfully placed and routed with a realistic set of weights that for some configurations like 2/2/2 has high sparsity. It resulted that some arrays have been optimized out, consequently it allowed to place more instances than it would be possible in the worst case scenario when all arrays are fully implemented. In the case of some configurations like 4/4/4, a maximum number of instances is limited with a number of available BRAMs. In order to implement some arrays using URAM that is not initializable memory, the architecture has been extended with

[1]The power has been estimated resorting to the Vivado Power Analysis tool on the post-placed netlist, supposing a °25C ambient temperature, 12.5% of toggle rate and 0.5 of static probability in BRAMs.

TABLE II: Comparison of implementations for processing dense LSTM models on FPGA

	[23], 2015	[24], 2016	[25], 2016	[26], 2017	[46], 2017	[27], 2017	[28], 2017	[30], 2017	This work
Platform	Zynq XC7Z020	Virtex-7 XC7VX485T	Zynq XC7Z045	Virtex-7 XCVX485T	Zynq XC7Z020	Stratix-V GSMD5	Zynq XC7Z045	Kintex XCKU060	Zynq XCZU7EV
Model storage	off-chip	on-chip	on-chip	off-chip	on-chip	off-chip	on-chip	off-chip	on-chip
Precision[2] [bits]	16 fixed	18 fixed	4-6 fixed	32 float	- fixed	16 fixed	5 fixed	12 fixed	1-8 fixed
Frequency, [MHz]	142	141	100	150	200	150	142	200	266
Throughput[3] [GOP/S]	0.284	4.56	-	6.87	14.2	299	693.12	200/1789[4]	1833
Efficiency[5] [GOP/J]	0.15	-	-	0.37	-	12.63	55.88	4.88/61.46	-

[2] we indicate only precisions that have been demonstrated in the papers.
[3] throughput has been normalized with respect to 142 MHz, assuming that throughput depends linearly on frequency.
[4] throughput on a dense model / throughput on a sparse model.
[5] based on only measured power (not estimated).

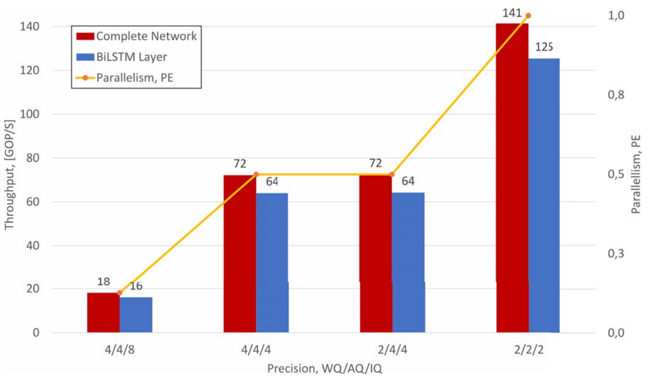

Fig. 7: Throughput scalability depending on precision on PYNQ-Z1

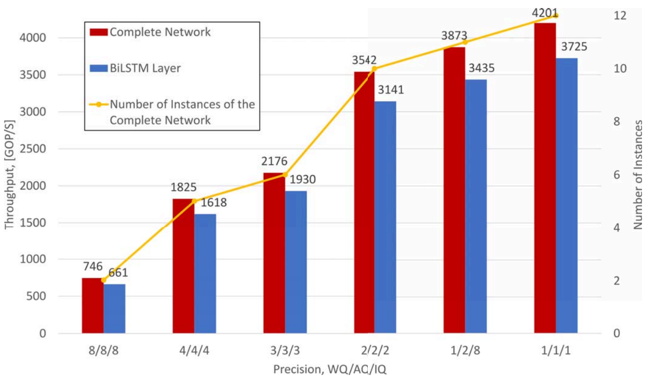

Fig. 8: Throughput scalability depending on precision on ZCU104

a block that configures arrays at the beginning. That allowed to place more instances by balancing BRAM and URAM usage.

C. Discussion of the results

We have shown that our quantized implementation can surpass a single-precision floating-point accuracy for a given dataset when using 1-bit precision for weights and multi-bit quantization for activations. The increase in precision beyond 3 bits is of negligible benefit and only results in increased hardware cost and power consumption. Our experiments confirm observations that have been done in previous works [21], [22] that low bit-width networks can outperform their high-precision counterparts because weight quantization plays a regularization role that prevents model overfitting.

It has been shown that our proposed approach of quantizing output and recurrent activations differently, namely 1/2/1/1, see Fig. 4-6, while being inferior to full-precision, outperforms 1/1/1 configuration by 1.32% in accuracy with no increase in complexity of the recurrent layer that is responsible for the most of complexity of the network in Fig. 2.

We have demonstrated parameterizable performance scaling for two different parallelization approaches, see Fig. 7 and Fig. 8. The results show flexible customizability of the architecture for different scenarios depending on hardware constrains and required accuracy.

For selected ZCU104 platform and 1/2/8 configuration without accuracy degradation with respect to the full-precision counterpart, the design could achieve a throughput of 3873 GOP/S for the complete network and 3435 GOP/S @ 266 MHz for the LSTM layer that is the highest with respect to state-of-the-art implementations on FPGAs operating on a dense LSTM model, see Table II. For throughput comparison we use GOP/S that has been used natively in all mentioned papers. Although application level throughput can be a fairer metric, it becomes complicated because of various applications that have been used for benchmarking the designs.

VII. Conclusion

This paper presents the first systematic exploration of hardware cost, power consumption, and throughput scalability as a function of precision for LSTM and BiLSTM in particular. We have conducted an in-depth investigation of precision vs. classification accuracy using a fully hardware-aware training flow, where during training quantization of all aspects of the network are taken into consideration.

We are providing the first open source HLS library extension of FINN [1] for parameterizable hardware architectures of LSTM layers on FPGAs which offers full precision flexibility and allows for parameterizable performance scaling.

Based on this library, we have presented an FPGA-based accelerator for BiLSTM NN designed for OCR that has achieved the highest throughput with respect to state-of-the-art implementations of RNNs on FPGAs operating on a dense LSTM model. We have also demonstrated the first binarized hardware implementation of LSTM network.

ACKNOWLEDGMENT

This work was partially supported by the InnoProm program of the state Rhineland-Palatinate, Germany. We also thank Insiders Technologies GmbH for consultancy in machine learning.

REFERENCES

[1] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[2] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[3] J. Chung, K. Cho, and Y. Bengio, "A character-level decoder without explicit segmentation for neural machine translation," *arXiv preprint arXiv:1603.06147*, 2016.

[4] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4960–4964.

[5] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*. IEEE, 2015, pp. 3156–3164.

[6] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 806–814.

[7] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[8] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[11] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[12] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[13] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[15] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[16] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv preprint arXiv:1609.07061*, 2016.

[18] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[19] S.-C. Zhou, Y.-Z. Wang, H. Wen, Q.-Y. He, and Y.-H. Zou, "Balanced quantization: An effective and efficient approach to quantized neural networks," *Journal of Computer Science and Technology*, vol. 32, no. 4, pp. 667–682, 2017.

[20] Q. He, H. Wen, S. Zhou, Y. Wu, C. Yao, X. Zhou, and Y. Zou, "Effective quantization methods for recurrent neural networks," *arXiv preprint arXiv:1611.10176*, 2016.

[21] L. Hou, Q. Yao, and J. T. Kwok, "Loss-aware binarization of deep networks," *arXiv preprint arXiv:1611.01600*, 2016.

[22] C. Xu, J. Yao, Z. Lin, W. Ou, Y. Cao, Z. Wang, and H. Zha, "Alternating multi-bit quantization for recurrent neural networks," *arXiv preprint arXiv:1802.00150*, 2018.

[23] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on fpga," *arXiv preprint arXiv:1511.05552*, 2015.

[24] J. C. Ferreira and J. Fonseca, "An fpga implementation of a long short-term memory neural network," in *ReConFigurable Computing and FPGAs (ReConFig), 2016 International Conference on*. IEEE, 2016, pp. 1–8.

[25] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin, and W. Sung, "Fpga-based low-power speech recognition with recurrent neural networks," in *Signal Processing Systems (SiPS), 2016 IEEE International Workshop on*. IEEE, 2016, pp. 230–235.

[26] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 629–634.

[27] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 2017, pp. 152–159.

[28] V. Rybalkin, N. Wehn, M. R. Yousefi, and D. Stricker, "Hardware architecture of bidirectional long short-term memory neural network for optical character recognition," in *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2017, pp. 1394–1399.

[29] X. Zhang, X. Liu, A. Ramachandran, C. Zhuge, S. Tang, P. Ouyang, Z. Cheng, K. Rupnow, and D. Chen, "High-performance video content recognition with long-term recurrent convolutional network for fpga," in *Field Programmable Logic and Applications (FPL), 2017 27th International Conference on*. IEEE, 2017, pp. 1–4.

[30] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition rngine with sparse lstm on fpga," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 75–84.

[31] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "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*. ACM, 2018, pp. 11–20.

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

[33] A. Graves, "Supervised sequence labelling," in *Supervised sequence labelling with recurrent neural networks*. Springer, 2012, pp. 5–13.

[34] T. M. Breuel, "Benchmarking of lstm networks," *arXiv preprint arXiv:1508.02774*, 2015.

[35] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[36] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[37] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.

[38] [Online]. Available: http://torch.ch/

[39] [Online]. Available: http://pytorch.org/

[40] N. Léonard, S. Waghmare, Y. Wang, and J.-H. Kim, "rnn: Recurrent library for torch," *arXiv preprint arXiv:1511.07889*, 2015.

[41] [Online]. Available: https://github.com/pytorch/pytorch/tree/master/aten

[42] [Online]. Available: https://github.com/Xilinx/pytorch-quantization

[43] [Online]. Available: https://github.com/Xilinx/pytorch-ocr

[44] [Online]. Available: https://github.com/baidu-research/warp-ctc

[45] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 168–173, 1974.

[46] Y. Hao and S. Quigley, "The implementation of a deep recurrent neural network language model on a xilinx fpga," *arXiv preprint arXiv:1710.10296*, 2017.

[47] [Online]. Available: https://github.com/Xilinx/LSTM-PYNQ/