Shihui Yin, Xiaoyu Sun, Shimeng Yu, Jae-sun Seo, and Chaitali Chakrabarti

School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, AZ 85281, USA
Email: syin11@asu.edu, xsun63@asu.edu, shimengy@asu.edu, jaesun.seo@asu.edu, chaitali@asu.edu

Abstract **– Recurrent neural networks (RNNs) provide excellent performance on applications with sequential data such as speech recognition. On-chip implementation of RNNs is difficult due to the significantly large number of parameters and computations. In this work, we first present a training method for LSTM model for language modeling on Penn Treebank dataset with binary weights and multi-bit activations and then map it onto a fully parallel RRAM array architecture ("XNOR-RRAM"). An energy-efficient XNOR-RRAM array based system for LSTM RNN is implemented and benchmarked on Penn Treebank dataset. Our results show that 4-bit activation precision can provide a near-optimal perplexity of 115.3 with an estimated energy-efficiency of ~27 TOPS/W.**

I. INTRODUCTION

Recurrent neural networks (RNNs) have shown remarkable performance on artificial intelligence tasks such as speech recognition [1], image caption [2], and language modeling [3]. Long-Short-Term Memory (LSTM) [4] and Gated Recurrent Units (GRU) [5] are the two most widely used RNNs. State-of-the-art RNNs are typically deployed on large-scale servers in data centers due to their high demand on memory storage size and computational power requirements. For example, the LSTM model in [6] for Penn Tree Bank (PTB) [7] language modeling task contains 66M parameters and requires 51M floating-point multiply-and-accumulate operations (MACs) per word prediction. Directly mapping this model onto a conventional processor would result in crippling energy consumption and is thus infeasible.

To facilitate efficient hardware implementation and reduce computation energy and latency, researchers have been studying approaches to reduce the precision of weights and hidden states of LSTMs without compromising their performance. Straight-through estimator (STE) has been used in [8] to train quantized LSTM models. Balanced quantization proposed in [9] to quantize the weights and activations to multi-bit values, has shown better performance on a 2-bit/3-bit precision for weights/activations compared to imbalanced quantization scheme. Xu et al. [10] recently proposed an optimization method that quantizes the weights and activations to multi-bit values by alternatively optimizing binary codes and real coefficients. This method achieved better performance than full precision baseline with 3-bit/3-bit precision for weights/activations for an LSTM model. Hou et al. [11] showed that binarized LSTM can achieve promising performance in character-level language modeling task. In addition to low-precision LSTM model training, model compression techniques have been recently explored to reduce the number of weights of LSTMs by pruning out unnecessary weights without hurting the accuracy of the models. Narang et al. [12] proposed a pruning-based method to reduce the number of weights in RNNs by ~90%, at the expense of reducing accuracy of the model by 10-20%. Wen et al. [13] proposed a method to learn compact structures in RNNs using group Lasso regularization.

In this work, we present a training method for an LSTM model for language modeling on PTB dataset with binary weights and multi-bit activations. Unlike rectified linear unit (ReLU) activation function widely used in modern deep convolutional neural networks (CNNs), the sigmoid and $tanh$ activation functions in LSTM models suffer from the gradient vanishing problem [14]. Thus, when we binarize the weights of LSTM models, careful selection of binary weight magnitude is required to stay away from the gradient-saturating region. This problem has not been considered by previous quantized LSTM training algorithms [9][10][11].

Exploiting the massive parallelism in the RNN models, hardware RNN accelerators have been proposed in recent years. Lee et al. [15] proposed an RNN accelerator for speech recognition on Xilinx XC7Z045 FPGA utilizing only on-chip memory with 6-bit quantized weights. Han et al. [16] presented an efficient FPGA accelerator for speech recognition with sparse LSTM on Xilinx XCKU060 FPGA. They quantized the weights to 12-bit without any accuracy loss, requiring two 4GB DRAMs as off-chip memory for data storage. Shin et al. [17] proposed a reconfigurable CMOS ASIC processor for CNN and RNN with quantization table-based matrix multiplication to reduce off-chip accesses.

In existing hardware accelerators of RNNs, SRAM is commonly utilized to store synaptic weights on the chip. However, a SRAM cell consumes more than 100 F^2 (F = technology feature size) in area, constraining the capacity of on-chip weight storage. To enhance on-chip storage, researchers have proposed using embedded non-volatile memories (eNVMs) with much less area (<10F^2) such as resistive random access memory (RRAM) [18] and phase change memory (PCM) [19] to implement "analog" synaptic weights. Long et al. [20] proposed an analog RRAM crossbar array based implementation of a conventional RNN for human activity detection. Even though analog RRAM hold great advantages on area-efficiency,

the non-ideal analog weight characteristics (e.g. weight update nonlinearity, limited dynamic range) introduce notable accuracy degradation [21]. Therefore, it is more practical to use technologically more mature binary RRAM that have been demonstrated at Gb chip-level by industry as a near-term solution [22]. Our prior work [23] demonstrated a one-transistor-one-resistor (1T1R) binary RRAM array based architecture, namely XNOR-RRAM, for implementing deep binary CNNs featuring binary inputs and binary weights. The proposed XNOR-RRAM features parallel read-out by activating all the wordlines (WLs) simultaneously and simulation results have shown ~33X improvement in energy- compared to traditional row-by-row read-out scheme with negligible accuracy degradation for MNIST and CIFAR-10 dataset.

Since a number of prior works [9][10] have reported that binarized activations (when used together with binarized weights) will significantly degrade the performance of large LSTM models, in this work, we present an XNOR-RRAM based architecture to support LSTM networks with binary weights and multi-bit activations. We benchmark the area, latency, and energy of designs with different activation precisions and analyze the tradeoffs. The contribution of this work includes:

- A new LSTM training method for language modeling is proposed featuring binary weights and multi-bit activations. The impact of binary weight magnitude when combined with different input and hidden state precisions on the language modeling performance is investigated. We found that careful selection of the binary weight magnitude helps to minimize the performance degradation.
- An XNOR-RRAM based architecture is proposed for implementing LSTM models. Simulation results show negligible accuracy degradation on PTB in spite of the 4-bit quantization of partial sums due to the proposed hardware.
- The area, latency, and energy of the proposed architecture is benchmarked with different activation precisions and the tradeoffs are analyzed. Benchmark results show that 4-bit activation could achieve a near-optimal accuracy with moderate area overhead and high energy-efficiency of ~27 TOPS/W.

The rest of this paper is organized as follows. Section II introduces the background of LSTM models and explains the training algorithm. Section III describes the proposed XNOR-RRAM based architecture for LSTM models. Section IV compares the performance on area, latency, and energy between designs with different precisions for input and hidden states. Section V concludes the paper.

II. Binary-Weight Multi-Bit-Activation LSTM

A. Long Short-Term Memory (LSTM)

In this paper, we consider LSTM model for word-level language modeling. The goal is to predict the next word given a sequence of words. The LSTM model starts with an embedding layer, which converts each word in a vocabulary to an embedding vector. Then multiple LSTM layers are stacked to extract the features of the input word based on context, followed by a softmax fully connected (FC) layer to predict the probability of each word in the vocabulary of the language.

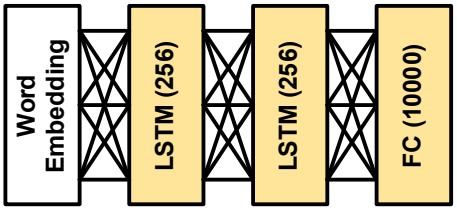

Fig. 1. LSTM model structure.

In an LSTM layer, cell states contain information from history, which are updated by current inputs and previous hidden states. Hidden states are then evaluated based on the updated cell states, current inputs and previous hidden states. The detailed model for an LSTM layer [4] is described in (1)-(6):

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad (1)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \quad (2)$$
$$u_t = \tanh(W_u \cdot [h_{t-1}, x_t] + b_c), \quad (3)$$
$$C_t = f_t * C_{t-1} + i_t * u_t, \quad (4)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \quad (5)$$
$$h_t = o_t * \tanh(C_t), \quad (6)$$

where f, i, u, and o represent the forget, input, transform and output gate, respectively. Four corresponding weight matrices W_f, W_i, W_u and W_o transform the concatenated vector of previous hidden states h_t and current input x_t to obtain the gate values. Sigmoid $\sigma(x)$ and hyperbolic tangent $\tanh(x)$ functions are the activation functions.

Let us denote the size of the LSTM layer as n, the vocabulary size as m, then the four matrices in each LSTM layer are all of size $n \times 2n$, the weight matrix in the softmax FC layer is of size $m \times n$, and the embedding matrix is of size $n \times m$.

B. Quantization of weights and inputs in LSTM layers

When the LSTM layer size n or vocabulary size m is large, it is impractical to store all the weights and inputs on the chip. For example, Fig. 1 shows an LSTM model with two LSTM layers for language modeling on a 10,000-word-vocabulary Penn Treebank (PTB) dataset, where the overall LSTM model contains 6.2M parameters. Therefore, quantizing them could significantly reduce the memory storage stress and computation complexity. In this work, we propose a method that binarizes the weights and quantizes inputs to multi-bit precisions in the LSTM model such that the LSTM model is suitable for an XNOR-RRAM array implementation. In particular, the binarizing function for the weights is:

$$bin(x) = sign(C(x)) * w_m, \quad (7)$$

where $sign(x)$ returns +1 if $x \geq 0$, otherwise returns -1. $C(x)$ is the clip function that clips x to the range of [-1, 1].

The k-bit input quantization function is:

TABLE I. 2-bit Input Quantization within [-1, 1]

Level	b_1 (2/3)	b_0 (1/3)	Value
0	$\bar{1}$	$\bar{1}$	-1
1	$\bar{1}$	1	-1/3
2	1	$\bar{1}$	+1/3
3	1	1	+1

$$q_{\mathrm{k}}(x) = R\left(C(x) \times \frac{2^{k}-1}{2}\right) \times \frac{2}{2^{k}-1} - 1, \quad (8)$$

where $R(x)$ is the round function that rounds x to its nearest integer number.

We can see that the 2^k levels are uniformly-spaced and zero-symmetric. For example, Table I shows the details of a 2-bit quantization scheme. The four quantization levels are {-1, -1/3, 1/3, 1}. A 2-bit input can be represented as $\overline{b_1 b_0}$, where b_1 is of weight 2/3, b_0 is of weight 1/3. b_1 and b_0 can be +1 or -1, represented as '1' and '$\bar{1}$', respectively. In general, a k-bit input can be represented as $\overline{b_{k-1} b_{k-2} \ldots b_0}$, where b_j can be +1 or -1, and has the weight of $\frac{2^j}{2^k-1}$, $j = 0, 1, \ldots, k-1$.

Therefore, the product of a k-bit input vector $\boldsymbol{x}$ and binary weight matrix $\boldsymbol{W}$ can be reduced to sum of k binary-input-binary-weight matrix-vector multiplications as shown in equations (9)-(11):

$$\boldsymbol{x} = \sum_{j=0}^{k-1} \boldsymbol{x}_{\mathrm{j}} \times \frac{2^j}{2^k-1}, \quad (9)$$

$$\boldsymbol{W} = w_m \boldsymbol{W}_s, \quad (10)$$

$$\boldsymbol{x}^{\mathrm{T}} \boldsymbol{W} = \sum_{j=0}^{k-1} \frac{w_m 2^j}{2^k-1} \times \boldsymbol{x}_j^T \boldsymbol{W}_s, \quad (11)$$

where $\boldsymbol{x}_{\mathrm{j}}$ is a binary (+1/-1) vector of $\boldsymbol{x}$'s j-th bits, w_m is the weight binary magnitude, and $\boldsymbol{W}_s$ is the sign matrix (+1/-1) of $\boldsymbol{W}$. XNOR-and-bitcounting operation is used to implement the dot product between a 1-bit input vector and a binary weight vector and k shift-add operations are required to accumulate the XNOR-and-bitcounting results to implement the dot-product between a k-bit input vector and a binary vector. This process is described in Algorithm 1.

C. Training Quantized LSTM

The weights and inputs of both LSTM layers and the softmax FC layer are binarized or quantized using functions defined in (7) and (8). However, the sign and round functions are not differentiable, hindering the back propagation of gradients. To overcome this problem, we use straight-through estimator (STE) [24][25] to approximate the gradient of sign and round function during back-propagation with constant one. Dropout [6] has been employed to avoid overfitting the LSTM model. In this work, for 1-bit input precision, dropout ratio is 0.05; for 2 to 5-bit input precisions, dropout ratio is 0.2; for floating-point input precision, dropout ratio is 0.4.

Batch normalization [26] has been used in quantized convolutional neural networks or fully-connected networks to speed up training and improve generalization performance. However, batch normalization in LSTM model is not straightforward and is hypothesized [27] to hurt training due to repeated rescaling, and is only limited to input-to-hidden transition. Proper initialization of batch normalization parameters [28] is suggested to make batch normalization work. Layer normalization [29] is another option for speeding up and improving generalization performance of LSTM models, which computes the layer normalization statistics overall the hidden units in the same layer. An additional benefit of batch normalization or layer normalization in quantized LSTM model

---

Algorithm 1: Dot product of binary weight matrix $\boldsymbol{W}$ and k-bit input vector $\boldsymbol{x}$.

1. Split $\boldsymbol{x}$ into k binary vectors $\{\boldsymbol{x}_{k-1}, \boldsymbol{x}_{k-2}, \ldots, \boldsymbol{x}_0\}$ such that equation (9) satisfies.
2. First compute multiplication between the MSB vector $\boldsymbol{x}_{k-1}^T$ and $\boldsymbol{W_S}$.
   $\boldsymbol{y} = \boldsymbol{x}_{k-1}^T \times \boldsymbol{W}_s$;
3. Keep left-shifting $\boldsymbol{y}$ and add with the product between each following binary vector and weight matrix.
   for ($j = k-2$; j ≥ 0; j--) {
   $\boldsymbol{y} = \boldsymbol{y} \ll 1$;
   $\boldsymbol{y} = \boldsymbol{y} + \boldsymbol{x}_j^T \times \boldsymbol{W}_s$;
   }
4. Scale $\boldsymbol{y}$ to get the final product.
   $\boldsymbol{y} = \frac{w_m}{2^k-1} \boldsymbol{y}$.

---

is rescaling the pre-activation values to reasonable input range for sigmoid or tanh functions. In this work, we applied a simple and easy-to-implement rescaling to the pre-activation values in both LSTM layers (12)-(15) and the FC layer (16):

$$f_t = \sigma\left(\gamma_f * (W_f \cdot [h_{t-1}, x_t]) + b_f\right), \quad (12)$$

$$i_t = \sigma(\gamma_i * (W_i \cdot [h_{t-1}, x_t]) + b_i), \quad (13)$$

$$u_t = \tanh\left(\gamma_g * (W_u \cdot [h_{t-1}, x_t]) + b_u\right), \quad (14)$$

$$o_t = \sigma(\gamma_o * (W_o \cdot [h_{t-1}, x_t]) + b_o), \quad (15)$$

$$y_t = softmax(\gamma_s * (W_s \cdot x_t) + b_s), \quad (16)$$

where W_f, W_i, W_u and W_o are binary weight matrices in an LSTM layer and W_s is binary weight matrix in the softmax FC layer; x_t is k-bit quantized input to each layer; h_t is k-bit quantized hidden states of each LSTM layer; γ_f, γ_i, γ_u, γ_{o} and γ_s are trainable scaling parameters, all initialized to 1 during training; $*$ means element-wise multiplication between two vectors.

We find that the weight magnitude w_m is crucial for binary-weight multi-bit-input LSTM model training. Since the inputs are quantized within the range of [-1, 1] for each layer, if w_m is too small, the pre-activation value will be too small, limiting the reachable range of activation values. If w_m is too large, the pre-activation values become too large, and may get stuck in small-gradient regions of sigmoid and hyperbolic tangent functions in the beginning of the training process. To see this more clearly, let's take an example of 1-bit input and LSTM layer of size n.

Let us assume that the input binary vector is almost random and uncorrelated, which means each entry is +1 with probability of 0.5. Similarly, we assume that the associated weight vector is also random and uncorrelated, and the input vector and weight vector are also uncorrelated with each other. Then, the product of each entry of the input and weight vector is $+w_m$ and $-w_m$ with probability of 0.5 and 0.5, respectively. The variance of each product is equal to w_m^2. The sum of n products has a variance of $n \times w_m^2$. The standard deviation of the weighted

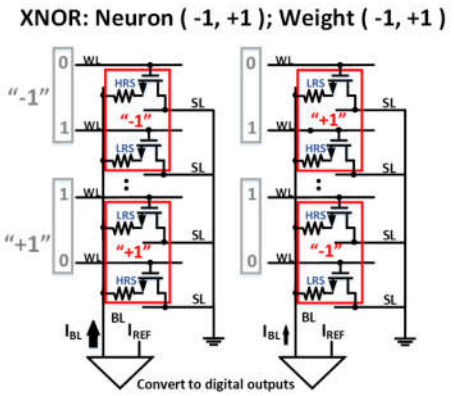

Fig. 2. The customized bit-cell design for implementing XNOR function in RRAM array [23]. Bit-counting results can be obtained by converting the total bitline current to digital outputs.

sum is $\sqrt{n} \times w_m$. When n is large, the weighted sum's binomial distribution is close to a normal distribution. If w_m is large, say, equal to 1, and n = 256, then the standard deviation of the weighted sum is 16, which means most weighted sums will fall in a region of sigmoid or hyperbolic tangent function where gradient is almost zero. Therefore, proper w_m should be selected properly based on the size of LSTM layer.

We train the LSTM model unrolled in 20 time steps, using stochastic gradient descent with momentum as the learning optimizer. The LSTM layer size n is equal to 256. The initial learning rate is 0.15 for the first 30 epochs, then decays by a factor of 0.85 till the end of 50 epochs. Batch size is 20. We sweep the w_m from 1/64 to 1/2 logarithmically for input precision from 1-bit to 5-bit. The performance of LSTM model for language modeling is measured by perplexity per word (PPW). Table II summarizes the perplexity on valid and test PTB dataset. Note that to reduce the performance gap between software and hardware fixed-point implementation, we also quantize intermediate variables such as cells states and pre-activation values to 8-bit fixed-point values. During LSTM training, STE is used for the quantization for these variables as well.

The baseline LSTM model with floating-point (FP) weights and floating-point inputs/activations achieves a test PPW of 103.0. After we binarize the weights, the test PPW degrades slightly to 106.3 when w_m is equal to 1/4. As we further decrease the precision for the inputs/activations, the test PPW degrades further. As we can see from Table II, 4-bit input/activation precision is required to avoid significant PPW degradation. For very low input/activation precisions such as 1-bit or 2-bit, w_m should be smaller to achieve good PPW values. This is because the input variance is generally larger for lower precision when we quantize input to the range of [-1, 1]. For instance, inputs quantized to 1-bit values can only take -1 and +1, while inputs quantized to 5-bit values can take values closer to 0 such as -1/31 and 1/31. Therefore, careful selection of binary weight magnitude is required when the input is quantized during training to minimize the performance degradation of an LSTM model compared to its full-precision counterpart.

TABLE II. PPW ON PTB VALID (TEST) DATASET

w_m	1/64	1/32	1/16	1/8	1/4	1/2	Baseline
FP	682.4 (639.1)	135.6 (131.0)	118.7 (113.4)	114.0 (109.4)	**111.1 (106.3)**	113.3 (107.8)	107.4 (103.0)
5	682.4 (639.1)	137.5 (132.4)	118.9 (114.6)	114.2 (109.6)	**112.3 (108.0)**	122.6 (114.8)	--
4	229.5 (221.2)	138.1 (133.1)	119.1 (114.5)	**115.9 (110.3)**	115.6 (111.0)	121.7 (115.6)	--
3	188.1 (181.3)	139.3 (133.4)	123.7 (118.1)	**120.1 (116.3)**	127.4 (121.3)	140.8 (133.4)	--
2	165.6 (158.1)	138.0 (132.8)	**132.4 (126.5)**	141.8 (134.9)	150.6 (143.1)	203.9 (192.9)	--
1	164.6 (157.0)	**152.5 (146.6)**	158.3 (152.1)	176.1 (165.9)	267.6 (254.2)	362.9 (345.7)	--

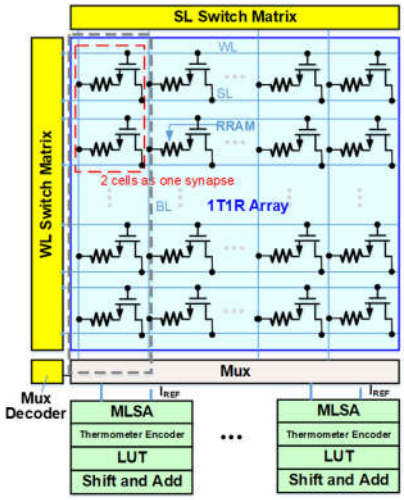

Fig. 3. The diagram of adapted XNOR-RRAM array structure. To enable the computation with multi-bit input, a shift-and-add module is added to the periphery.

III. XNOR-RRAM ARRAY ARCHITECTURE FOR LSTM

A. *Customized bit-cell for XNOR function*

As explained in Section II, the fundamental operations of the proposed algorithm for the LSTM model are XNOR-and-bitcounting operations. A customized RRAM bit-cell is utilized to efficiently implement XNOR function in the RRAM array. We only consider the pseudo-crossbar 1T1R array since the two-terminal threshold switch selectors for true crossbar array are currently not technologically mature for large scale integration. As shown in Fig. 2, every two RRAM cells in the same column are grouped as one synaptic binary weight. "+1" is represented by two cells where the top one is in low resistance state (LRS) and the bottom one is in high resistance state (HRS), "-1" is represented in the reversed pattern [23]. The input pattern is coded similarly with two complimentary wordlines (WLs). With this setup, when input data is "-1", the activated cell in "-1" weight cell is in LRS, leading to a large cell current, which can represent the bit-wise XNOR output of "+1"; for the cell of weight "+1", the activated cell is in HRS, leading to a small cell current, which can be regarded as XNOR output of "-1". As a result, XNOR function is successfully implemented in analog domain. Moreover, when multiple WLs are activated in parallel, the LRS-cells will dominate the contributions to the total bitline current (I_{BL}). Assuming the on/off ratio of RRAM device is

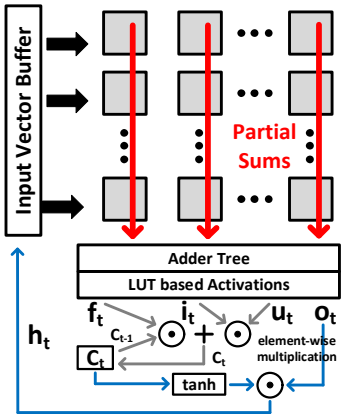

Fig. 4. The diagram of XNOR-RRAM array based architecture for implementing LSTM. Sub-array size is 64×64.

sufficiently large (e.g. >100), I_{BL} will be proportional to the XNOR-and-bit-counting result along the column. Therefore, the XNOR-and-bit-counting results can be obtained by converting the analog I_{BL} to digital outputs.

B. XNOR-RRAM architecture for LSTM

Fig. 3 shows the XNOR-RRAM array structure. Instead of the one-hot decoder for conventional memory array, a WL switch matrix is exploited to activate multiple WLs simultaneously to enable the fully parallel read operation. Multilevel sense amplifiers (MLSAs) are utilized to convert the analog I_{BL} to digital outputs of fixed-point precision, and a look-up table (LUT) based non-linear quantization method is used to generate the corresponding quantized bit-counting results as in [23]. Due to the layout pitch-match issue (the width of an MLSA block is much larger than an RRAM cell pitch), every 8 columns share one set of MLSA and LUT. In this work, a shift-and-add module is added to the peripheral circuitry to enable the computation with multibit input, i.e., k cycles will be consumed to generate the final result when the input is of k-bit precision.

Due to the intrinsic offset of sense amplifiers caused by process variation, the sensing pass rate (percentage of accurate sensing outputs) becomes worse when I_{BL} increases (as cell currents are summed up for a large array), leading to inaccurate bit-counting results, which may significantly degrade the accuracy. Therefore, the large weight matrices in LSTM are split into multiple small matrices and implemented with small sub-arrays. In this work, we conservatively use a relatively small sub-array size of 64 × 64 at the expense of larger number of sub-arrays to cover all the weight matrices. Fig. 4 shows the system level architecture, where each gray box represents a processing engine (PE), i.e., an adapted XNOR-RRAM array with peripheral circuits. The PE array is scalable depending on the desired weight matrix size. Input vector buffer temporally stores both the incoming input vector at the current time step and its own hidden state h_{t-1} from last time step. Activation functions (sigmoid and hyperbolic tangent) are implemented with LUTs.

A digital circuit block consisting of 8-bit multipliers and adders are placed after LUTs and used for updating C_t and h_t accordingly, as shown in Fig. 4. Cell state C_t is stored in a dedicated buffer. The computing process for one LSTM layer can be divided into 3 stages. In stage I, the partial sums, i.e., the bit-counting results, are read out from each sub-array (including shift-and-add) and then summed up through adder trees; the final sums go through the LUT based activation functions to generate four intermediate states f_t, i_t, u_t and o_t. The precision of four intermediate states is set to be 8-bit. In stage II, two groups of 8-bit multipliers are utilized to perform element-wise multiplication between f_t and C_{t-1}, i_t and u_t, respectively. Then the corresponding products are added up to update C_t in the buffer, which is also quantized to 8-bit after addition. In stage III, the new C_t is passed through hyperbolic tangent function and is then multiplied by o_t to generate h_t, which is then quantized to the desired precision and stored back to the input buffer for the computation in the next time step.

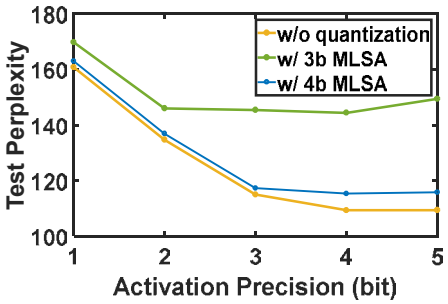

Fig. 5. Test perplexity with different activation precisions considering the effect of nonlinear quantization of partial sums by MLSAs.

To reduce the number of SAs required in MLSAs, we apply nonlinear quantization to the partial sums before we add them up across different sub-arrays. Quantization levels are determined based on the software partial sum distribution using Lloyd-Max algorithm [23][30]. We investigate the impact of the MLSA resolution on test perplexity to find the lowest resolution required to preserve good language modeling performance. As shown in Fig. 5, test perplexity is unacceptably high when the MLSA resolution is 3-bit. By increasing the resolution to 4-bit, the test perplexity can be significantly reduced to near-optimal level with degradation of less than 5 in PPW for all the cases. Therefore, 4-bit MLSAs are chosen for our implementation. With 4-bit activation precision and 4-bit MLSAs, the test PPW is equal to 115.3 with a slight degradation of 12.3 in PPW compared to the full-precision software test result.

IV. Benchmark on Area, Latency, and Energy

In this section, we estimate the area, latency, and energy of the proposed XNOR-RRAM array based architecture using NeuroSim [31], an open-source estimator for RRAM based hardware accelerators. The hierarchy of the estimator consists of different levels of abstraction from the memory cell parameters and transistor technology parameters, to the gate-level sub-circuit modules, and then to the array architecture.

We implemented one LSTM layer of size 256. The area, latency, and energy results for different activation precisions (1-bit to 5-bit) are summarized in Table III. Even though the main array structure remains the same for multi-bit activations, the area overhead increases with precision due to shift-and-add modules and larger buffers for activation data storage. However, the growth in area is relatively insignificant since the other peripheral circuits such as MLSAs, LUTs, and multipliers

TABLE III. COMPARISON BETWEEN DESIGNS WITH BINARY WEIGHTS AND DIFFERENT INPUT/ACTIVATION PRECISIONS

Precision	Perplexity	Area (mm^2)	Latency (ns)	TOPS/W
5-bit	115.8	0.66	185.21	22.06
4-bit	115.3	0.65	159.43	27.41
3-bit	117.3	0.64	133.64	35.60
2-bit	136.9	0.63	107.86	49.63
1-bit	163.0	0.60	82.07	79.00

dominate the area. As expected, the latency for one layer computation increases for a higher input precision as more rounds of array computation need to be performed, and the energy-efficiency of the system correspondingly decreases from ~79 TOPS/W (1-bit activation precision) to ~22 TOPS/W (5-bit activation precision). Considering both performance on perplexity and hardware overhead, the results suggest that 4-bit activation precision is a favorable option that provides near-optimal accuracy with moderate area overhead and a high energy-efficiency of ~27 TOPS/W, which represents a 2.3× improvement compared to an SRAM based ASIC CNN/RNN accelerator [17].

V. CONCLUSION

In this paper, we described an approach to train an LSTM model for language modeling on PTB dataset with binary weights and multi-bit activations to reduce the number of parameters by ~32X. Proper binary weight magnitude is selected to minimize the performance degradation. To efficiently implement the proposed LSTM in hardware, we presented a XNOR-RRAM array based architecture that can perform XNOR-and-bitcounting in the array. The PTB benchmark results suggest that binary-weight/4-bit-activation with 4-bit MLSAs can achieve a near-optimal perplexity with an estimated energy-efficiency of ~27 TOPS/W. Future works include investigating the impact of RRAM device variation on test perplexity and the related compensating techniques to pave the way for practical deployment.

ACKNOWLEDGMENT

This work is in part supported by NSF/SRC E2CDA program under NSF grant 1740225 and SRC contract 2018-NC-2762, NSF grant 1652866, and C-BRIC, one of six centers in JUMP, a SRC program sponsored by DARPA.

REFERENCES

[1] A. Graves *et al*., "Towards end-to-end speech recognition with recurrent neural networks," *International Conference on International Conference on Machine Learning (ICML)*, pp. II-1764-II-1772, 2014.

[2] J. Donahue *et al*., "Long-term recurrent convolutional networks for visual recognition and description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39(4), pp. 677-691, 2015.

[3] I. Sutskever *et al*., "Sequence to sequence learning with neural networks," *Neural Information Processing Systems*, pp. 3104-3112, 2014.

[4] S. Hochreiter *et al*., "Long short-term memory," *Neural Computation*, 9(8), pp. 1735-1780, 1997.

[5] K. Cho *et al*., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," arXiv:1406.1078, 2014.

[6] W. Zaremba *et al*., "Recurrent neural network regularization," arXiv: 1409.2329, 2014.

[7] M. P. Marcus *et al*., "Building a large annotated corpus of English: The Penn Treebank," *Computational linguistics*, 19(2), pp. 313-330, 1993.

[8] I. Hubara *et al*., "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," arXiv:1609.07061, 2016.

[9] S. Zhou *et al*., "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.

[10] C. Xu *et al*., "Alternating Multi-bit Quantization for Recurrent Neural Networks," *International Conference on Learning Representations (ICLR)*, 2018.

[11] L. Hou *et al*., "Loss-aware binarization of deep networks," *International Conference on Learning Representations (ICLR)*, 2017.

[12] S. Narang *et al*., "Exploring Sparsity in Recurrent Neural Networks," *International Conference on Learning Representations (ICLR)*, 2017.

[13] W. Wen *et al*., "Learning Intrinsic Sparse Structures within Long Short-Term Memory," *International Conference on Learning Representations (ICLR)*, 2018.

[14] Y. Bengio *et al*., "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Nerual Networks*, vol. 5, no. 2, pp. 157-166, 1994

[15] M. Lee *et al*., "FPGA-Based Low-Power Speech Recognition with Recurrent Neural Networks," *IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 230-235, 2016.

[16] S. Han *et al*., "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," *International Symposium on Filed-Programmable Gate Arrays (FPGA)*, pp. 75-84, 2017

[17] D. Shin *et al*., "DNPU: An 8.1TOPS/W Reconfigurable CNN-RNN Processor for General-Purpose Deep Neural Networks," *IEEE International Solid-State Circuits Conference (ISSCC),* pp. 240-241, 2017.

[18] S. Park *et al*., "Neuromorphic speech systems using advanced ReRAM based synapse," *IEEE International Electron Devices Meeting (IEDM)*, pp. 25.6.1-25.6.4, 2013.

[19] G. W. Burr *et al*., "Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element," *IEEE International Electron Devices Meeting (IEDM)*, pp. 3498-3507, 2014.

[20] Y. Long *et al*., "ReRAM crossbar based recurrent neural network for human activity detection," *International Joint Conference on Neural Networks (IJCNN)*, pp. 939-946, 2016.

[21] P.-Y. Chen *et al.*, "Mitigating effects of non-ideal synaptic device characteristics for on-chip learning," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 194-199, 2015.

[22] R. Fackenthal *et al*., "A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology," *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 338-339, 2014.

[23] X. Sun *et al*., "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1423-1428, 2018.

[24] G. Hinton, "Neural networks for machine learning," Coursera, video lectures, 2012.

[25] Y. Bengio, "Estimating or propagating gradients through stochastic neurons," arXiv:1305.2982, 2013.

[26] S. Ioffe *et al*., "Batch Normalization: Accelerating Deept Network Training by Reducing Internal Covariate Shift," arXiv:1502.03167, 2015.

[27] C. Laurent *et al*., "Batch normalized recurrent neural networks," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2657-2661, 2016.

[28] T. Cooijmans *et al*., "Recurrent Batch Normalization," *International Conference on Learning Representations (ICLR)*, 2017.

[29] J. L. Ba *et al*., "Layer Normalization," arXiv:1607.06450, 2016.

[30] J. Max *et al*., "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, 6(1), 7-12, 1960.

[31] P.-Y. Chen *et al*., "NeuroSim+: An integrated device-to-algorithm framework for benchmarking synaptic devices and array architectures," *IEEE International Electron Devices Meeting (IEDM)*, pp. 6.1.1-6.1.4, 2017.