Hybrid Analog-Spiking Long Short-Term Memory for Energy Efficient Computing on Edge Devices

Wachirawit Ponghiran
School of Electrical and Computer Engineering
Purdue University
West Lafayette, USA
wponghir@purdue.edu

Kaushik Roy
School of Electrical and Computer Engineering
Purdue University
West Lafayette, USA
kaushik@purdue.edu

***Abstract*—Recurrent neural networks such as Long Short-Term Memory (LSTM) have been used in many sequential learning tasks such as speech recognition and language translation. Running large-scale LSTMs for real-world applications is known to be compute-intensive and often relies on cloud execution. To enable LSTM operations on edge devices that receive inputs in real-time, there is a need to improve LSTM execution efficiency following the limited energy constraint of the mobile platforms. We propose a hybrid analog-spiking LSTM that combines the energy efficiency of spiking neural network (SNN) with the performance efficiency of analog (non-spiking) neural network (ANN). SNN, which processes and represents information as a sequence of sparse binary spikes or events, uses integrate and fire activation, hence consuming low power and energy for real-time inference (batch size of 1). The proposed Analog-Spiking LSTM is derived from a trained LSTM using a novel conversion method that transforms the fully-connected layers and the non-linearity function compatible for SNNs. We show that the default LSTM non-linearities are sources of output mismatch between the ANN and the SNN. We propose a set of replacement functions that lead to a minimal impact on the output quality of sequential learning problems. Our analyses on sequential image classification on MNIST dataset and sequence-to-sequence translation on the IWSLT14 dataset indicate $<1\%$ drop in average accuracy for row-wise and pixel-wise sequential image recognition and <1.5 drop in average BLEU score for the translation task. Implementation of the recognition system with the hybrid analog-spiking LSTM on Intel's spiking processor, Loihi, shows $55.9\times$ improvement in active energy per inference over the baseline system on Intel i7-6700. Based on our analysis, we estimate this benefit to be $3.38\times$ reduction in active energy per inference for the translation task.**

***Index Terms*—LSTM, SNN, Energy Efficiency, Edge Devices**

I. Introduction

The recent success stories of Recurrent Neural Networks (RNNs) include solving sequential learning problems such as speech recognition [1] and Neural Machine Translation (NMT) [2]. The feedback connections are the keys that allow information to retain temporally and enable RNNs to capture long-term dependencies over input sequences. The computation of RNNs is a sequential process over multiple time-steps (see Fig.1(a)). Portions of the input data, e.g. words in a sentence, are sequentially fed to the RNN unit. Although unrolling RNNs in time allows them to be trained using backpropagation algorithm (see Fig.1(b)), training RNNs was known to be difficult until Long Short-Term Memory (LSTM), a specially designed RNN unit, was introduced [3]. LSTMs then became common building blocks of language processing engines that deal with texts or speeches like Google Translate [4]. Nonetheless, such systems are often deployed on cloud servers because of their intensive computations. The quantized Google NMT from English to French, for instance, takes 1,322 seconds to run on two Intel Haswell CPUs [4].

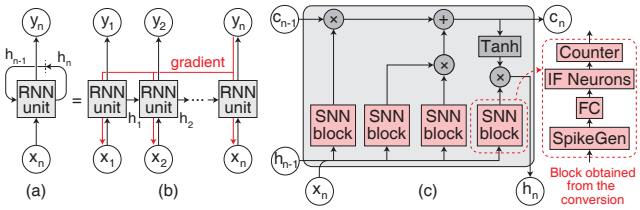

Fig. 1. (a) The computation of RNN at n^{th} step takes input x_n together with memory state h_{n-1} and produces h_n, which is again reused by the RNN unit. (b) Equivalent representation when the RNN operations are explicitly unrolled into n steps, allowing gradient (colored in red) to flow back from the output y_n during backpropagation. (c) Hybrid analog-spiking LSTM: FC layers and their activation functions of trained LSTM are converted to SNNs which run over multiple time-steps to match the output of the original ANNs.

With the emergence of the internet of things, computations are gradually shifting to mobile and edge devices. Hence, there is a need to improve the execution efficiency of real-time LSTM operation (batch size of 1) on power-constrained platforms. This differs from the motivations behind many prior efforts to improve compute efficiency of LSTM. For example, parallelizing LSTM workloads across CPU and GPU improves execution efficiency at the cost of extra hardware, leading to increased power consumption [4]–[6]. Implementation of LSTMs on FPGA is more efficient than GPUs; however, the power consumption on FPGA is still in the range of 10W [7]–[10]. State-of-the-art Application Specific Integrated Circuits (ASICs) like BrainWave and Google TPU are designed to achieve high throughput and do not target edge devices [11], [12]. Even the energy consumption of Google Edge TPU,

This work was supported in part by the Center for Brain Inspired Computing (C-BRIC), one of the six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, the National Science Foundation, the DoD Vannevar Bush Fellowship, the U.S. Army Research Laboratory, the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001, and Sandia National Labs.

an ASIC variant for edge inference (2W of power consumption [13]) may not be suitable for ultra low-power applications. Other techniques such as model compression and pruning insignificant LSTM states [14], [15] are orthogonal to the underlying implementation and can be applied independently to further increase compute efficiency.

A. Motivation and Contributions

In this paper, we propose hybrid analog-spiking LSTM that improves execution efficiency and addresses the limited power constraint of edge devices by running fully-connected layers, which are compute-intensive, as spiking neural networks (SNNs). SNNs are neural networks in which information is represented and processed as sequences of spikes or binary values. They are in contrast to the analog neural networks (ANNs), which communicate through real values. Because SNNs operate with sequences of spikes, computations can be simplified to addition as opposed to power consuming multiply operations. Spike sparsity can be exploited to avoid unnecessary computation in an event-driven hardware, i.e., only the presence of spike triggers a neuron update. Recent studies have shown that a functionally equivalent SNN can be constructed from a trained feed-forward ANN [16]–[18]. These observations motivated us to conduct experiments on SNN hardware. We found that active power consumption to execute 1-layer of a fully-connected SNN is 266× less than its ANN counterpart running on CPU.[1] Active energy per inference is also 2× smaller for SNNs.

Converting the fully-connected (FC) layers and their activation functions of LSTMs to SNNs is, however, not trivial. We cannot directly apply the existing conversion techniques [16]–[18] because LSTM uses activation functions that do not correspond to spiking neuron characteristics. We propose a set of training constraints that guarantee successful conversion to a hybrid analog-spiking LSTM. We show that the constraints have a minimal impact on the output quality. Note, the energy consumption of LSTMs is dominated by the FC layers.

Our contributions can be summarized as follows:

1) We propose hybrid analog-spiking LSTM derived from a trained LSTM through our proposed ANN-to-SNN conversion technique. The energy-hungry FC layers and the activation functions of the trained LSTM are converted to low-power and energy-efficient SNNs. Note, however, the sigmoid and the tanh activations of the LSTM gates require proper modification to preserve functionality and accuracy of the hybrid LSTM on complex datasets.
2) We show that the proposed network has minimal impact on the output quality of two popular sequential learning tasks, namely sequential image classification on the MNIST dataset and sequence-to-sequence translation from German to English on the IWSLT14 dataset.
3) We implemented the proposed hybrid analog-spiking LSTM on Intel's spiking processor, Loihi [19], and measured performance against Intel CPU for real-time operations (with batch size of 1). The improvement in active energy per inference is $55.9\times$ for the image recognition task with less than 1% drop in accuracy. Based on our measurements, $3.38\times$ improvement in the active energy per inference is estimated for the translation task with less than 1.5 drop in BLEU score.

[1] A single-layer fully-connected network with 512 input and 512 output neurons is used in this preliminary experiment. We measured performance on Intel Loihi and on Intel i7-6700 CPU, both fabricated in 14nm technology. Real-time operation with a batch size of 1 is assumed. SNN derived from ANN through conversion technique produces spike trains with output firing rates that approach activation values of ANN. We run the SNN for 128 time-steps to guarantee small output discrepancy between ANN and SNN (i.e. achieve mean squared error of 8.16×10^{-5} between ANN and SNN outputs).

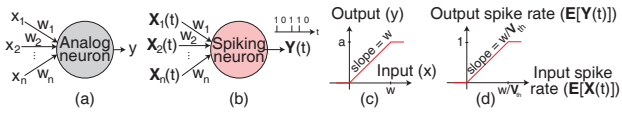

Fig. 2. (a) Analog neuron receives real numbers $x_1, x_2, ..., x_n$ as inputs and produces real number y as an output, (b) spiking neuron receives spike trains $\mathbb{X}_1(t), \mathbb{X}_2(t), ..., \mathbb{X}_n(t)$ as inputs and produces spike train $\mathbb{Y}(t)$ as an output. An output spike is produced when the neuron membrane potential, which is a dot product of the its inputs and weights, reaches a given spiking threshold. $w_1, w_2, ..., w_n$ denote weight of the connections to a neuron. (c) Transfer function of truncated ReLU and (d) transfer function of IF neuron.

The remainder of this paper is organized as follows. Section II provides a comparison between ANNs and SNNs, and ANN-to-SNN conversion techniques. Section III proposes our method to construct energy-efficient hybrid analog-spiking LSTM. Experimental results are presented in Section IV, where we discuss the impact of the proposed method on the output quality and present energy measurement on SNN hardware. The paper is concluded in Section V.

II. Background

A. Analog vs. Spiking Neural Networks (ANNs vs. SNNs)

ANNs and SNNs are two different classes of artificial neural networks and the difference lies in how their neurons operate and how the inputs are coded. ANNs represent and process information as real values; hence, an analog neuron receives real numbers as inputs (see Fig. 2(a)). The neuron then integrates the inputs after synaptic weight modulation and applies an activation function on top to produce an output. If a rectified linear unit (ReLU) is used as an activation function, the output of the neuron is computed as:

$$y = \begin{cases} \sum_{i=1}^{n} w_i x_i & \text{if } \sum_{i=1}^{n} w_i x_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where x_i represents an input through synaptic weight w_i.

SNNs work differently by transmitting information as sequence of spikes or binary values; thus, spiking neuron receives spike trains that indicate either the presence or absence of inputs over multiple time-steps (see Fig. 2(b)). To account for temporal information in inputs, the neuron has an internal state called membrane potential $\mathbb{V}(t)$. The neuron updates $\mathbb{V}(t)$ upon receipt of input spike associated with a synaptic weight, and generates an output spike once $\mathbb{V}(t)$ crosses a spiking threshold

$\mathbb{V}_{th}$, and the membrane potential $\mathbb{V}(t)$ is then reset. The output of an artificial spiking neuron can be written as:

$$\mathbb{Y}(t) = \begin{cases} 1 & \text{if } \mathbb{V}(t) > \mathbb{V}_{th} \\ 0 & \text{otherwise} \end{cases}, \tag{2}$$

$$\mathbb{V}(t) = \begin{cases} \mathbb{V}(t-1) + \sum_{i=1}^{n} w_i \mathbb{X}_i(t) - \mathbb{V}_{th} & \text{if } \mathbb{Y}(t-1) = 1 \\ \mathbb{V}(t-1) + \sum_{i=1}^{n} w_i \mathbb{X}_i(t) & \text{otherwise} \end{cases} \tag{3}$$

where $\mathbb{X}_i(t)$ represents a spike train through synaptic weight w_i. Spiking neuron which follows this behavior is commonly referred to as integrate-and-fire (IF) neuron. Because the IF neuron processes sequences of binary values, the computations of membrane potential are simplified to additions, and such additions are only performed upon the receipt of input spike(s).

B. ANN-to-SNN Conversion

The operations of an SNN are heavily dependent on how inputs to the SNN are coded. Rate coded inputs are widely used today for SNNs [16]–[18] although other coding techniques are being actively investigated [20]. Recent works have shown that under rate coding, IF neuron bears resemblance to activation function like truncated ReLU [16]–[18] (see Fig. 2(c)-(d)). Suppose an analog and a spiking neuron receive a real number input x and an input spike train $\mathbb{X}(t)$, respectively. The transfer function of the truncated ReLU can be mathematically expressed as:

$$y = \begin{cases} a & \text{if } w \cdot x > a \\ w \cdot x & \text{if } 0 < w \cdot x \leq a \\ 0 & \text{if } w \cdot x \leq 0 \end{cases} \tag{4}$$

where a is a threshold of the truncated ReLU, w is a synaptic weight, and y is the output of the truncated ReLU. There is a range in which IF neuron produces output spikes with a rate proportional to an input spike rate like the truncated ReLU. Outside that range, the output spike rate saturates or equals to zero. The transfer function of the IF neuron can be written as:

$$\mathbb{E}[\mathbb{Y}(t)] = \begin{cases} 1 & \text{if } \frac{w}{\mathbb{V}_{th}} \cdot \mathbb{E}[\mathbb{X}(t)] > 1 \\ \frac{w}{\mathbb{V}_{th}} \cdot \mathbb{E}[\mathbb{X}(t)] & \text{if } 0 < \frac{w}{\mathbb{V}_{th}} \cdot \mathbb{E}[\mathbb{X}(t)] \leq 1 \\ 0 & \text{if } \frac{w}{\mathbb{V}_{th}} \cdot \mathbb{E}[\mathbb{X}(t)] \leq 0 \end{cases} \tag{5}$$

where w is a synaptic weight and $\mathbb{Y}(t)$ denotes output spike train. $\mathbb{E}[\cdot]$ represents the spike rate or the expected value of the spike train. Given that $\mathbb{X}(t)$ is a spike train with firing rate x, it is obvious that $\mathbb{E}[\mathbb{Y}(t)] = y$ if both threshold of the truncated ReLU and the spiking neuron are 1. In general,

$$\mathbb{E}[\mathbb{Y}(t)] = y/a \tag{6}$$

holds for a proper choice of $\mathbb{V}_{th}$.

Because ReLU is a typical activation function for training feed-forward ANNs, we can substitute ReLUs on each layer with truncated ReLUs to enable ANN-to-SNN conversion. To avoid discrepancy at the network outputs, thresholds of the truncated ReLUs are set to be the maximum activation (i.e. maximum output) within a layer. Existing works show that computing the maximum activation based on a sample batch of training data provides a good estimate for the truncated ReLU threshold value [16]–[18]. Hence, any feed-forward ANN with ReLU activation function can be converted into a functionally equivalent SNN under the assumption of rate-coding.

In practice, the performance of SNNs also depends on the number time-steps (inference latency) required to infer a given test input. Accuracy for representing a real number with a sequence of spikes increases with the number of time-steps (or the latency). For instance, the mean squared error for representing 10^6 random numbers between 0 and 1 decreases from 1.18×10^{-3} (with 16 time-steps) to 3.03×10^{-4} (with 32 time-steps). Running SNN for a large number of time-steps guarantees that the outputs of SNN closely match the ANN counterpart. Note, however, the energy consumption increases with the number of time-steps required to represent the inputs. On the other hand, running SNN for fewer time-steps may affect the output quality.

III. LSTM to Hybrid Analog-Spiking LSTM Conversion

In an LSTM, the fully-connected layers and their activation functions are the most compute-intensive components. Assume LSTM has inputs and hidden states of dimension a and b, respectively. In such a situation, $4ab + 4b^2$ multiply-accumulate (MAC) operations are required for the FC layers while only $3b$ MAC operations are required for the rest of computations. We convert these power-hungry components to low-power SNNs and obtain hybrid analog-spiking LSTM as shown in Fig. 1(c). However, to preserve the functionality after the conversion, we impose two training constraints for the LSTM: (1) replacing activation functions with their piece-wise linear approximations and (2) normalizing inputs to have values between -1 and 1.

A. Constraint 1: Modified Activation Functions

Sigmoid and tanh are activation functions that follow the FC layers. These functions are transcendental, and have their transfer functions different from the transfer function of an IF neuron. Hence, we cannot apply existing ANN-to-SNN conversion techniques [16]–[18] directly. We could derive a spiking neuron that has a similar transfer function to sigmoid or tanh; however, this results in a spiking neuron with complex functionality. To address the issue, we replace sigmoid and tanh with their piece-wise linear approximations. We describe how to map each piece-wise linear (so-called hard) function to a spiking neuron in this subsection.

Hard sigmoid function follows equation:

$$y = \begin{cases} 1 & \text{if } x > 2 \\ \frac{x+2}{4} & \text{if } -2 < x \leq 2 \\ 0 & \text{if } x \leq -2 \end{cases} \tag{7}$$

where x and y are input and output of the function. Hard sigmoid outputs a value between 0 and 1 which is essential for gating information on the cell state. The function can be considered as a truncated ReLU with a bias, a unit threshold, and a scaled input. We achieve ANN-to-SNN conversion by replacing hard sigmoid with IF neuron that has $\mathbb{V}_{th} = 4$ and bias $= 2$. In the simplest case where $x = 0$, $y = 1/2$. $\mathbb{X}(t)$ is

an empty spike train generated according to x. The IF neuron receives no input spike and its $\mathbb{V}(t)$ increases by 2 from bias every time-step. Hence, the $\mathbb{V}(t)$ crosses $\mathbb{V}_{th}$ every other time, and the output firing rate is $1/2$.

Hard tanh function is computed as:

$$y = \begin{cases} 1 & \text{if } x > 2 \\ \frac{x}{2} & \text{if } -2 < x \leq 2 \\ -1 & \text{if } x \leq -2 \end{cases} \tag{8}$$

where x and y are input and output of the function. Hard tanh outputs a value between -1 and 1 which helps avoid gradients from exploding during training. However, mapping hard tanh to IF neuron is not straightforward. Hard tanh produces positive and negative values while the output firing rate of IF neuron is always positive. To get around the problem of a negative firing rate, we make IF neuron generate negative spikes. Whenever $\mathbb{V}(t)$ crosses $-\mathbb{V}_{th}$, IF neuron generates -1 and $\mathbb{V}(t)$ is reset after firing. Similar to (2) and (3), output of the IF neuron with dual spiking thresholds can be mathematically described by:

$$\mathbb{Y}(t) = \begin{cases} 1 & \text{if } \mathbb{V}(t) > \mathbb{V}_{th} \\ -1 & \text{if } \mathbb{V}(t) < -\mathbb{V}_{th} \\ 0 & \text{otherwise} \end{cases},$$

$$\mathbb{V}(t) = \begin{cases} \mathbb{V}(t-1) + \sum_{i=1}^{n} w_i \mathbb{X}_i(t) - \mathbb{V}_{th} & \text{if } \mathbb{Y}(t-1) = 1 \\ \mathbb{V}(t-1) + \sum_{i=1}^{n} w_i \mathbb{X}_i(t) + \mathbb{V}_{th} & \text{if } \mathbb{Y}(t-1) = -1 \\ \mathbb{V}(t-1) + \sum_{i=1}^{n} w_i \mathbb{X}_i(t) & \text{otherwise} \end{cases}$$

Equation (6) holds for the negative range because of symmetry in the transfer function of IF neuron with dual spiking thresholds. Hence, we achieve ANN-to-SNN conversion by replacing hard tanh with IF neuron that has dual spiking thresholds $\mathbb{V}_{th}$=2. We implement such neuron, that is not natively supported on Loihi [19], by using two IF neuron compartments: one for generating positive spikes and another for generating negative spikes. With proper synaptic connections, we make their $\mathbb{V}(t)$ to have the same magnitude but a different sign. Spike is then produced when $\mathbb{V}(t)$ crosses $\mathbb{V}_{th}$ or $-\mathbb{V}_{th}$.

B. Constraint 2: Input Normalization

The range of LSTM inputs varies for different learning tasks. Sequential image recognition typically employs a preprocessing step to normalize pixel intensity. The distribution of inputs then has zero mean and unit variance. Sequence-to-sequence translation, on the other hand, uses outputs of the embedded layer as inputs to the LSTM. The embedding layer works like a lookup table and outputs a vector of unbounded values. Hence, the unbounded nature of the inputs leads to two problems in generating input spike trains for the hybrid analog-spiking LSTM operation.

First, a negative number cannot be encoded into a spike train based on the rate coding scheme. To generate a spike train representing a negative number, we introduce a negative spike so that the input spike train becomes a sequence of ternary values: -1, 0, and 1. -1 and 1 indicate events that lead to depression and potentiation of $\mathbb{V}(t)$, respectively. Propagating negative spikes in SNN over multiple time-step thus resembles propagating negative real value in ANN.

Second, an input firing rate of ternary spike train is between -1 and 1 but unbounded input values may exceed that range. We address this problem by utilizing different input normalization scheme for each task. For sequential image recognition, input pixels are normalized to have zero mean and maximum absolute value of 1 instead of unit variance. For sequence-to-sequence translation, we apply vector normalization to the embedded layer. Vector normalization enforces the layer to output unit vectors. Based on our experiments, we found that these constraints do not impact the output quality of learning tasks as shown in Section IV.

The above constraints guarantee a minimal loss in generating a spike train from each input. Throughout this work, we generate input spike trains using a deterministic process. Given input value x, we use IF neuron with dual spiking thresholds $\mathbb{V}_{th}$=1. The neuron $\mathbb{V}(t)$ is incremented by x every time-step of SNN operation. The neuron generates output spike whenever $\mathbb{V}(t)$ crosses a positive or negative spiking threshold. This process ensures that the input firing rate is proportional to the input value and closely mimics the behavior of SNN hardware that we use to perform the energy measurement.

C. Conversion Algorithm and Hybrid LSTM Operation

The first step of our conversion method is to train a network with LSTM(s) for a particular task based on the constraints discussed previously. We then replace the activation function after each FC layer. Depending on the type of activation function that follows the FC layer, we substitute an analog neuron with an appropriate spiking neuron. We then insert additional blocks for SNN operations, namely spike generator and counter, to the trained network.

Once the hybrid LSTM receives vectors h_{n-1} and x_n, the two vectors are concatenated into one long vector (see Fig. 1(c)). The vector is then fed to a spike generator which generates spike trains over multiple time-steps. These time-steps of SNN operation are independent of the computational step of the hybrid LSTM. In other words, we run SNNs for a certain number of times-step (e.g. 128) to approximate outputs of the FC layers with activation functions. Outputs of SNNs are accumulated by spike counters. We then compute output firing rates before passing them to the rest of the hybrid LSTM. Operations to update c_n and generate h_n from this point onward are the same for the regular LSTM and the hybrid analog-spiking LSTM. Even though SNN is operated over multiple time-steps, we show in the following section that the benefits of running SNNs outweigh overhead in generating spike trains and computing output firing rates.

IV. Experimental Results

We implement the proposed method with PyTorch [21], a popular deep learning framework. The impact of the conversion method on output quality is evaluated on two sequential learning tasks, namely sequential image classification and sequence-to-sequence translation. Baseline architectures are vanilla LSTMs without any modification running on CPU.

TABLE I
HYBRID LSTM IMPLEMENTED ON LOIHI IS COMPARED AGAINST THE BASELINE LSTM ON CPU. COLUMN 2-4 SHOWS ACTIVE ENERGY PER INFERENCE OF THE RECOGNITION SYSTEMS BASED ON IMAGE ROWS. COLUMN 5-7 SHOWS LATENCY PER INFERENCE OF THE RECOGNITION SYSTEMS.

LSTM hidden states	Active energy (J) per inference			Latency (s) per inference		
	Hybrid LSTM	Baseline LSTM (batch size of 1)	Baseline LSTM (batch size 32)	Hybrid LSTM	Baseline LSTM (batch size of 1)	Baseline LSTM (batch size 32)
64	2.16×10^{-3}	2.35×10^{-1}	1.00×10^{-2}	6.31×10^{-2}	7.62×10^{-3}	9.69×10^{-3}
128	4.31×10^{-3}	2.41×10^{-1}	1.16×10^{-2}	1.22×10^{-1}	7.71×10^{-3}	1.10×10^{-2}
192	6.99×10^{-3}	2.44×10^{-1}	1.35×10^{-2}	1.06×10^{-1}	7.76×10^{-3}	1.24×10^{-2}

A. Impact on Accuracy of Sequential Image Classification

Sequential image classification on MNIST dataset is a popular task to evaluate variants of LSTMs in capturing long-term dependencies [22], [23]. Portions of an image are presented sequentially to the recognition system, and classification is performed after feeding all pixels. Each image in the MNIST dataset has a dimension 28×28, thus consisting of 784 pixels in total. LSTM has to learn long-term dependencies over the input sequence to achieve good classification accuracy. There are two variants of sequential image classification that we use in this experiment, namely recognition based on image pixels and image rows (i.e. pixels on the same row are fed to LSTM at a time).

When recognizing based image rows, hybrid LSTM runs iteratively for 28 times as there are 28 rows for each MNIST digit. Recognition system with the hybrid LSTM reaches 99% of the average baseline accuracy when 64 time-steps are used for SNN operation (see left graph in Fig. 3(a)). When recognizing based on image pixels, the recognition task becomes more difficult. The hybrid LSTM has to run 784 times. We expect error from computing with SNNs to accumulate over computation steps and can become critical. Results indicate that pixel-wise recognition system with the hybrid LSTM performs poorly when a small number of time-steps are used for SNN operation (see right graph in Fig. 3(a)). However, the prediction accuracy increases quickly and saturates after a certain number of time-steps. When 1024 time-steps are used for SNN operation, the pixel-wise recognition system with the hybrid LSTM reaches 99% of the average baseline accuracy. This asserts our assumption that recognition based on a short sequence is more error resilient than the recognition based on a long sequence. We found that accuracy slightly drops if the image pixels are permuted as previously reported in [22], [23]; however, the average impact is similar.

B. Impact on Sequence-to-Sequence Translation Performance

We are interested in the impact of the conversion method on a practical sequential learning task that uses LSTM(s) as building block(s). We focus on sequence-to-sequence language translation, a popular LSTM application. To evaluate the quality of translation, we use an automatic scoring system called the BLEU score. The BLEU score is computed by comparing the translation output with a reference sentence and quantitatively measuring how two sentences are matched. The higher BLEU score indicates better translation. We simulated the impact of the translation performance on the IWSLT14 German to English translation benchmarks [24].

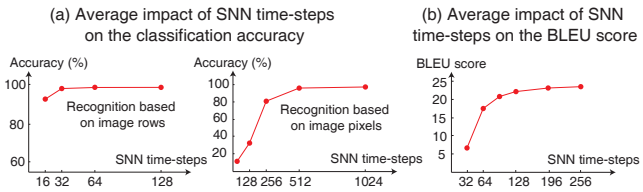

Fig. 3. (a) Average impact of time-step on the accuracy of sequential image recognition based on image rows and image pixels. (b) Average impact of time-step on the translation performance.

Our baseline BLEU score for German to English translation on the IWSLT14 dataset is 25.00 on average. Translation system with the hybrid LSTM achieves similar performance with BLEU score of 23.51 on average when 256 time-steps are used for SNN operation (see Fig. 3(b)). This is equivalent to less than 1.5 unit drop in BLEU score, resulting from the imposed constraints during training, the conversion method, and computation with SNN. We observe that an impact of time-steps on translation performance is similar to what we observed earlier for pixel-wise recognition task (see right graph in Fig. 3(a)).

C. Energy-Saving with Hybrid Analog-Spiking LSTM

In this section, we measure and estimate the energy of the hybrid LSTM realization on Intel Loihi [19], a hardware suitable for an event-driven SNN implementation. Due to the hardware constraint, we are only able to implement the recognition system in Intel Loihi hardware. We use 128 time-steps for SNN operation which guarantees less than 1% drop from baseline accuracy. The energy consumption of the system on Loihi is compared with measurement of the baseline system running on Intel i7-6700. Inference on the CPU is performed on PyTorch, which accounts for how each computation is executed. We utilize logging software to automatically record power consumption of the CPU for 15 minutes which is sufficient to perform inference over the test set multiple times. We report and focus on the active energy per inference as the architectural changes from LSTM to hybrid LSTM only affects the active power consumption. Minimizing idle energy, which is largely due to current leakage, is outside the scope of this study. Hence, we subtract a measured power reading with an average power consumption when no workload presents for CPU and Loihi.

Column 2-4 in Table I shows active energy per inference of the recognition systems with the baseline and hybrid LSTM. Performance of the system with the hybrid LSTM is obtained from Loihi. We run the recognition system with the baseline LSTM on the CPU at two different batch sizes. Recognition system with the proposed hybrid LSTM is 35-108$\times$ more energy efficient for real-time operation. We believe this energy saving is due to the benefit of running SNNs in an event-driven manner and the inefficiency of CPU to handle small workloads. Event-driven computation helps avoid unnecessary computations and enables the hybrid LSTM to exploit input sparsity. Alternatively, running the recognition system on CPU with a large batch size can increase the execution energy efficiency. Active energy per inference reduces approximately $20\times$ with batch size of 32 (see the fourth column of Table I). Batched inputs can better exploit the benefit of parallelism and vector multiplication. However, running workload in large batch may not always be feasible for mobile devices that receive inputs sequentially. Power consumption is also another important issue for the edge computing. Based on our measurements, we found that active power consumption of the recognition system with hybrid LSTM on Loihi is between 31-72 mW. Active power consumption of the system on CPU is around $30\mathrm{W}$ which is 2-3 orders of magnitude higher.

Drawback of the hybrid LSTM is an increase in the latency. Column 5-7 in Table I shows the latency of the recognition systems with the baseline and the hybrid LSTM. With batch size of 1, implementation on Loihi is 8-16$\times$ slower than the baseline on CPU. This large latency is a result of SNNs that have to run for 128 time-steps at every hybrid LSTM operation. The gap in inference speed, however, reduces slightly for the baseline on CPU with batched inputs (see the last column of Table I). Thus, an application of the conversion algorithm must be chosen carefully to meet the latency requirement.

To estimate performance of the translation system with the hybrid LSTM, we analyze the active energy consumption of the recognition system. Logic and SRAM contribute almost equally to the active energy consumption. The energy consumption increases proportionally with LSTM hidden states because the energy consumption is dominated by the number of spike events. When hidden states increase from 64 to 192, spike events and energy consumption increase roughly three times. The translation system uses two LSTMs with $1{,}024$ hidden states that run sequentially. Thus, we expect $16\times$ increase in the energy consumption from sequential recognition with 64 hidden states. Energy-saving with hybrid LSTM is approximately $6.75\times$ when 128 time-steps are used for SNN operation. If 256 time-steps are used to achieve <1.5 drop in BLEU score (as shown in Section IV-B), energy-savings with hybrid LSTM reduces to $3.38\times$.

V. Conclusion

LSTM is a popular recurrent neural network model used in many sequential learning tasks. Large LSTMs for real-world problems are compute-intensive and not suitable to be run on edge devices. We address this energy and power inefficiency by proposing hybrid analog-spiking LSTM derived from a trained LSTM using a novel conversion method. The proposed conversion method exploits existing frameworks to train LSTM with small modifications during training and is shown to have a small impact on output quality. Our realization of the recognition system with hybrid LSTM on Intel Loihi indicates $55.9\times$ active energy saving over the baseline on Intel i7-6700. Power consumption of the recognition system on Loihi is $32.4\ \mathrm{mW}$ which is 2 orders of magnitude less than the baseline on CPU. Based our analysis, we anticipate the benefit to translate to $3.38\times$ improvement in active energy per inference for the translation task.

References

[1] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP*. IEEE, 2013, pp. 6645–6649.

[2] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," in *Advances in NIPs*, 2014, pp. 3104–3112.

[3] S. Hochreiter, Y. Bengio, P. Frasconi *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[4] Y. Wu, M. Schuster, Z. Chen *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[5] B. Li, E. Zhou, B. Huang *et al.*, "Large scale recurrent neural network on GPU," in *IJCNN*. IEEE, 2014, pp. 4062–4069.

[6] J. Devlin, "Sharp models on dull hardware: Fast and accurate neural machine translation decoding on the cpu," in *Proceedings of the EMNLP*, 2017, pp. 2820–2825.

[7] S. Li, C. Wu, H. Li *et al.*, "FPGA acceleration of recurrent neural network based language model," in *FCCM*. IEEE, 2015, pp. 111–118.

[8] S. Han, J. Kang, H. Mao *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the FPGA*. ACM, 2017, pp. 75–84.

[9] Y. Guan, Z. Yuan, G. Sun *et al.*, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *ASP-DAC*. IEEE, 2017, pp. 629–634.

[10] S. Cao, C. Zhang, Z. Yao *et al.*, "Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity," in *Proceedings of the FPGA*, 2019, pp. 63–72.

[11] N. Jouppi, C. Young, N. Patil *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*. IEEE, 2017, pp. 1–12.

[12] J. Fowers, K. Ovtcharov, M. Papamichael *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *ISCA*. IEEE, 2018, pp. 1–14.

[13] "Coral Dev Board Datasheet version 1.3," https://coral.ai/static/files/Coral-Dev-Board-datasheet.pdf, accessed: 2020-05-08.

[14] A. See, M. Luong, and C. Manning, "Compression of neural machine translation models via pruning," in *SIGNLL CoNLL*, 2016, pp. 291–301.

[15] S. Sen and A. Raghunathan, "Approximate computing for LSTM neural networks," *IEEE TCAD*, vol. 37, no. 11, pp. 2266–2276, 2018.

[16] P. Diehl, D. Neil, J. Binas *et al.*, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *IJCNN*. IEEE, 2015, pp. 1–8.

[17] B. Rueckauer, I. Lungu, Y. Hu *et al.*, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[18] A. Sengupta, Y. Ye, R. Wang *et al.*, "Going deeper in spiking neural networks: VGG and residual architectures," *Frontiers in neuroscience*, vol. 13, 2019.

[19] M. Davies, N. Srinivasa, T. Lin *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[20] H. Mostafa, B. Pedroni, S. Sheik *et al.*, "Fast classification using sparsely active spiking networks," in *ISCAS*. IEEE, 2017, pp. 1–4.

[21] A. Paszke, S. Gross, F. Massa *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in NIPs*, 2019, pp. 8024–8035.

[22] Q. Le, N. Jaitly *et al.*, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[23] T. Cooijmans, N. Ballas, C. Laurent *et al.*, "Recurrent batch normalization," *arXiv preprint arXiv:1603.09025*, 2016.

[24] M. Cettolo, J. Niehues *et al.*, "Report on the 11th IWSLT evaluation campaign, IWSLT 2014," in *Proceedings of the IWSLT*, vol. 57, 2014.