Approximate FPGA-based LSTMs under Computation Time Constraints

Michalis Rizakis, Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis

Dept. of Electrical and Electronic Engineering, Imperial College London
{michail.rizakis14, stylianos.venieris10, a.kouris16, christos-savvas.bouganis}@imperial.ac.uk

Abstract. Recurrent Neural Networks, with the prominence of Long Short-Term Memory (LSTM) networks, have demonstrated state-of-the-art accuracy in several emerging Artificial Intelligence tasks. Nevertheless, the highest performing LSTM models are becoming increasingly demanding in terms of computational and memory load. At the same time, emerging latency-sensitive applications including mobile robots and autonomous vehicles often operate under stringent computation time constraints. In this paper, we address the challenge of deploying computationally demanding LSTMs at a constrained time budget by introducing an approximate computing scheme that combines iterative low-rank compression and pruning, along with a novel FPGA-based LSTM architecture. Combined in an end-to-end framework, the approximation method parameters are optimised and the architecture is configured to address the problem of high-performance LSTM execution in time-constrained applications. Quantitative evaluation on a real-life image captioning application indicates that the proposed system required up to 6.5× less time to achieve the same application-level accuracy compared to a baseline method, while achieving an average of 25× higher accuracy under the same computation time constraints.

Keywords: LSTM, Low-Rank Approximation, Pruning, FPGAs

1 Introduction

Recurrent Neural Networks (RNNs) is a machine learning model which offers the capability of recognising long-range dependencies in sequential and temporal data. RNN models, with the prevalence of Long Short-Term Memory (LSTMs) networks, have demonstrated state-of-the-art performance in various AI applications including scene labelling [1] and image generation [2]. Moreover, LSTMs have been successfully employed for AI tasks in complex environments including human trajectory prediction [3] and ground classification [4] on mobile robots, with more recent systems combining language and image processing in tasks such as image captioning [5] and video understanding [6].

Despite the high predictive power of LSTMs, their computational and memory demands pose a challenge with respect to deployment in latency-sensitive and power-constrained environments. Modern intelligent systems such as mobile robots and drones that employ LSTMs to perceive their surroundings often operate under time-constrained, latency-critical settings. In such scenarios, retrieving the best possible output from an LSTM given a constraint in computation time may be necessary to ensure the timely operation of the system. Moreover, the requirements of such applications for low absolute power consumption, which would enable a longer battery life, prohibit the deployment of high-performance, but power-hungry platforms, such as multi-core CPUs and GPUs. In this context, FPGAs constitute a promising target device that can combine customisation and reconfigurability to achieve high performance at a low power envelope.

In this work, an approximate computing scheme along with a novel hardware architecture for LSTMs are proposed as an end-to-end framework to address the problem of high-performance LSTM deployment in time-constrained settings. Our approach comprises an iterative approximation method that applies simultaneously low-rank compression and pruning of the LSTM model with a tunable number of refinement iterations. This iterative process enables our framework to (i) exploit the resilience of the target application to approximations, (ii) explore the trade-off between computational and memory load and application-level accuracy and (iii) execute the LSTM under a time constraint with increasing accuracy as a function of computation time budget. At the hardware level, our system consists of a novel FPGA-based architecture which exploits the inherent parallelism of the LSTM, parametrised with respect to the level of compression and pruning. By optimising the parameters of the approximation method, the proposed framework generates a system tailored to the target application, the available FPGA resources and the computation time constraints. To the best of our knowledge, this is the first work in the literature to address the deployment of LSTMs under computation time constraints.

2 Background

2.1 LSTM Networks

A vanilla RNN typically processes an input and generates an output at each time step. Internally, the network has recurrent connections from the output at one time step to the hidden units at the next time step which enables it to capture sequential patterns. The LSTM model differs from vanilla RNNs in that it comprises control units named gates, instead of layers. A typical LSTM has four gates. The *input* gate (Eq. (1)), along with the *cell* gate (Eq. (4)) are responsible for determining how much of the current input will propagate to the output. The *forget* gate (Eq. (2)) is responsible for determining whether the previous state of the LSTM will be forgotten or not, while the *output* gate (Eq. (3)) determines how much of the current state will be allowed to propagate to the final output of the LSTM at the current time step. Computationally, the gates are matrix-vector multiplication blocks, followed by a nonlinear elementwise activation function. The equations for the LSTM model are shown below:

$$\boldsymbol{i}^{(t)} = \sigma(\boldsymbol{W}_{ix}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{ih}\boldsymbol{h}^{(t-1)}) \tag{1}$$

$$\boldsymbol{f}^{(t)} = \sigma(\boldsymbol{W}_{fx}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{fh}\boldsymbol{h}^{(t-1)}) \tag{2}$$

$$\boldsymbol{o}^{(t)} = \sigma(\boldsymbol{W}_{ox}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{oh}\boldsymbol{h}^{(t-1)}) \tag{3}$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \odot tanh(\boldsymbol{W}_{cx}\boldsymbol{x}^{(t)} + \boldsymbol{W}_{ch}\boldsymbol{h}^{(t-1)}) \tag{4}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{c}^{(t)} \odot \boldsymbol{o}^{(t)} \tag{5}$$

$\boldsymbol{i}^{(t)}$, $\boldsymbol{f}^{(t)}$ and $\boldsymbol{o}^{(t)}$ are the *input*, *forget* and *output* gates respectively, $\boldsymbol{c}^{(t)}$ is the current state of the LSTM, $\boldsymbol{h}^{(t-1)}$ is the previous output, $\boldsymbol{x}^{(t)}$ is the current input at time t and $\sigma(\cdot)$ represents the sigmoid function. Eq. (5) is frequently found in the literature as $\boldsymbol{h}^{(t)} = \boldsymbol{c}^{(t)} \odot tanh(\boldsymbol{o}^{(t)})$ with $tanh(\cdot)$ applied to the *output* gate. In this work, we follow the image captioning LSTM proposed in [5] which removes the $tanh(\cdot)$ from the *output* gate and therefore we end up with Eq. (5). Finally, all the $\boldsymbol{W}$ matrices denote the weight matrices that contain the trainable parameters of the model, which are assumed to be provided.

3 Related Work

The effectiveness of RNNs has attracted the attention of the architecture and reconfigurable computing communities. Li et al. [7] proposed an FPGA-based accelerator for the training of an RNN language model. In [8], the authors focus on the optimised deployment of the Gated Recurrent Unit (GRU) model [9] in data centres with server-grade FPGAs, ASICs, GPUs and CPUs and propose an algorithmic memoisation-based method to reduce the computational load at the expense of increased memory footprint. The authors of [10] present an empirical study of the effect of different architectural designs on the computational resources, on-chip memory capacity and off-chip memory bandwidth requirements of an LSTM model. Finally, Guan et al. [11] proposed an FPGA-based LSTM accelerator optimised for speech recognition on a Xilinx VC707 FPGA platform.

From an algorithmic perspective, recent works have followed a model-hardware co-design approach. Han et al. [12] proposed an FPGA-based speech recognition engine that employs a load-balance-aware compression scheme in order to compress the LSTM model size. Wang et al. [13] presented a method that addresses compression at several levels including the use of circulant matrices for three of the LSTM gates and the quantisation of the trained parameters, together with the corresponding ASIC-based hardware architecture. Zhang et al. [14] presented an FPGA-based accelerator for a Long-Term Recurrent Convolutional Network (LRCN) for video footage description that consists of a CNN followed by an LSTM. Their design focuses on balancing the resource allocation between the layers of the LRCN and pruning the fully-connected and LSTM layers to minimise the off-chip memory accesses. [12], [13] and [14] deviate from the faithful LSTM mapping of previous works but also require a retraining step in order to compensate for the introduced error of each proposed method. Finally, He et al. [15] focused on CNNs and investigated algorithmic strategies for model selection under computation time constraints for both training and testing.

Our work differs from the majority of existing efforts by proposing a hardware architecture together with an approximate computing method for LSTMs that is application-aware and tunable with respect to the required computation time and application-level error. Our framework follows the same spirit as [12][13][14] by proposing an approximation to the model, but in contrast to these methods does not require a retraining phase and assumes no access to the full training set. Instead, with a limited subset of labelled data, our scheme compensates for the induced error by means of iterative refinement, making it suitable for applications where the dataset is privacy-critical and the quality of the approximation improves as the time availability increases.

4 Methodology

In this section, the main components of the proposed framework are presented (Fig. 1). Given an LSTM model with its set of weight matrices and a small application evaluation set, the proposed system searches for an appropriate approximation scheme that meets the application's needs, by applying low-rank compression and pruning on the model. The design space is traversed by means of a roofline model to determine the highest performing configuration of the proposed architecture on the target FPGA. In this manner, the trade-off between computation time and application-level error is explored for different approximation schemes. The design point to be implemented on the device is selected based on user-specified requirements for the maximum computation time or application-level error tolerance.

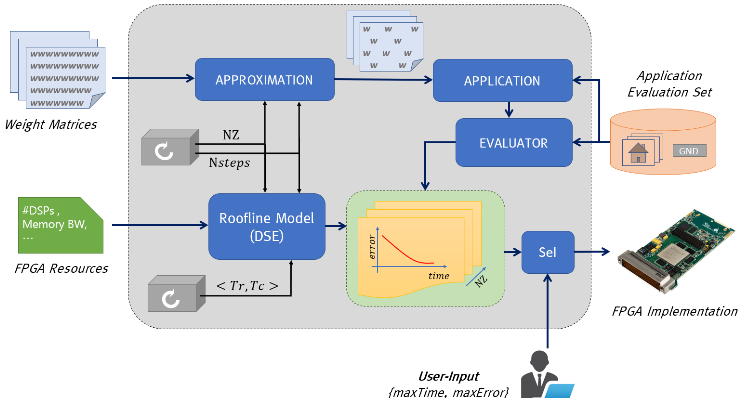

Fig. 1. Design flow of the proposed framework

4.1 Approximations for LSTMs

At the core of an LSTM's computational workload lie the matrix-vector multiplications in each of the four gates. Neural networks have been extensively studied to have redundancy in terms of their trained parameters [16]. To reduce the computational demands of the LSTM, we propose an approximate computing scheme that enables the tuning between computational cost and application-level accuracy. The proposed approach exploits the statistical redundancy of the LSTM by acting at two levels: (i) approximating the weight matrices with a low-rank, SVD-based decomposition and (ii) pruning the network by sparsifying the weight matrices based on an importance criterion of their elements.

Low-rank approximation. Based on the set of LSTM equations (1)-(4), each gate consists of two weight matrices corresponding to the current input and previous output vectors respectively. In our scheme, we construct an augmented matrix by concatenating the input and output weight matrices as shown in Eq. (7). Similarly, we concatenate the input and previous output vectors (Eq. (6)) and thus the overall gate computation is given by Eq. (8).

$$\tilde{\boldsymbol{x}}^{(t)} = \left[\boldsymbol{x}^{(t)T} \quad \boldsymbol{h}^{(t-1)T}\right]^{T} \tag{6}$$

$$\boldsymbol{W}_i = [\boldsymbol{W}_{ix} \;\; \boldsymbol{W}_{ih}]\,, \quad \forall i \in [1,4] \tag{7}$$

$$\boldsymbol{y}_i = nonlin(\boldsymbol{W}_i \tilde{\boldsymbol{x}}^{(t)}), \quad \forall i \in [1,4] \tag{8}$$

where $nonlin(\cdot)$ is either the sigmoid function $\sigma(\cdot)$ or $tanh(\cdot)$. In this way, a single weight matrix is formed for each gate, denoted by $\boldsymbol{W}_i \in \mathbb{R}^{R\times C}$ for the i_{th} gate. We perform a full SVD decomposition on the four augmented matrices independently as $\boldsymbol{W}_i = \boldsymbol{U}_i \boldsymbol{\Sigma}_i \boldsymbol{V}_i^T$, $\forall i \in [1,4]$, where $\boldsymbol{U}_i \in \mathbb{R}^{R\times R}$, $\boldsymbol{\Sigma}_i \in \mathbb{R}^{R\times C}$ and $\boldsymbol{V}_i \in \mathbb{R}^{C\times C}$ and employ a rank-1 approximation to obtain $\widetilde{\boldsymbol{W}}_i = \sigma_1^i \boldsymbol{u}_1^i \boldsymbol{v}_1^{iT}$ by keeping the singular vectors that correspond to the largest singular value.

Pruning by means of network sparsification. The second level of approximation on the LSTM comprises the structured pruning of the connectivity between neurons. With each neural connection being captured as an element of the weight matrices, we express network pruning as sparsification applied on the augmented weight matrices (Eq. (7)). To represent a sparse LSTM, we introduce four binary mask matrices $\boldsymbol{F}_i \in \{0,1\}^{R\times C}$, $\forall i \in [1,4]$, with each entry representing whether a connection is pruned or not. Overall, we employ the following notation for a (weight, mask) matrix pair $\{\boldsymbol{W}_i, \boldsymbol{F}_i \mid i \in [1,4]\}$.

In the proposed scheme, we explore sparsity with respect to the connections per output neuron and constrain each output to have the same number of inputs. We cast LSTM pruning as an optimisation problem of the following form.

$$\min_{\boldsymbol{F}_i} ||\boldsymbol{W}_i - \boldsymbol{F}_i \odot \boldsymbol{W}_i||_2^2, \quad \text{s.t.} \;\; ||\boldsymbol{f}_j^i||_0 = \text{NZ}, \;\; \forall i \in [1,4], \forall j \in [1,R] \tag{9}$$

where $\boldsymbol{f}_j^i$ is the j_{th} row of $\boldsymbol{F}_i$ and NZ is the number of non-zero elements on each row of $\boldsymbol{F}_i$. $||\cdot||_0$ is the l_0 pseudo-norm denoting the number of non-zero entries in a vector. The solution to the optimisation problem in Eq. (9) is given by keeping the NZ elements on each row of $\boldsymbol{W}_i$ with the highest absolute value and setting their indices to 1 in $\boldsymbol{F}_i$.

In contrast to the existing approaches, the proposed pruning method does not employ retraining and hence removes the computationally expensive step of retraining and the requirement for the training set, which is important for privacy-critical applications. Even though our sparsification method does not explicitly capture the impact of pruning on the application-level accuracy, our design space exploration, detailed in Section 5, searches over different levels of sparsity and as a result it explores the effect of pruning on the application.

Hybrid compression and pruning. By applying both low-rank approximation and pruning, we end up with the following weight matrix approximation:

$$\widetilde{\boldsymbol{W}}_i = \boldsymbol{F}_i \odot (\sigma_1^i \boldsymbol{u}_1^i \boldsymbol{v}_1^{iT}) \tag{10}$$

In this setting, for the i_{th} gate the ranking of the absolute values in each row of the rank-1 approximation $\sigma_1^i \boldsymbol{u}_1^i \boldsymbol{v}_1^{iT}$ depends only on $\boldsymbol{v}_1^i$, with each element of $\sigma_1^i \boldsymbol{u}_1^i$ operating as a shared scaling factor for all elements of a row. Therefore, for the i_{th} gate all the rows of $\boldsymbol{F}_i$ become identical and hence can be represented by a single mask vector $\boldsymbol{f}^i \in \{0,1\}^C$. This leads to a weight matrix with zeros along ($C-$NZ) of its columns, which is described by the following expression:

$$\widetilde{\boldsymbol{W}}_i = \sigma_1^i \boldsymbol{u}_1^i (\boldsymbol{f}^i \odot \boldsymbol{v}_1^i)^T \tag{11}$$

$$\tilde{\boldsymbol{y}}_i = \sum_{n=1}^{N_{steps}} \left\{ \sigma_1^{i(n)} \boldsymbol{u}_1^{i(n)} \left((\boldsymbol{f}^{i(n)} \odot \boldsymbol{v}_1^{i(n)})^T \tilde{\boldsymbol{x}}^{(t)} \right) \right\} \tag{12}$$

In order to obtain a refinement mechanism, we propose an iterative algorithm, presented in Algorithm 1, that employs both the low-rank approximation and pruning methods to progressively update the weight matrix. On lines 4-6 the first approximation of the weight matrix is constructed by obtaining the rank-1 approximation of the original matrix and applying pruning in order to have NZ non-zero elements on each row, as in Eq. (11). Next, the weight matrix is refined for N_{steps} iterations, by computing the error matrix $\boldsymbol{E}$ (line 10) and employing its pruned rank-1 approximation as an update (line 15).

Algorithm 1 Iterative LSTM Model Approximation

```
Inputs:
 1: Weight matrices $\boldsymbol{W}_i \in \mathbb{R}^{R \times C},\ \forall i \in [1, 4]$
 2: Number of non-zero elements, NZ
 3: Number of refinement iterations, $N_{steps}$
Steps:
 1: - - For all gates - -
 2: for $i = 1$ to 4 do
 3:     - - Initialise weight matrix approximation - -
 4:     $\left[\boldsymbol{u}_1^{i(0)}, \sigma_1^{i(0)}, \boldsymbol{v}_1^{i(0)}\right] = \mathrm{SVD}(\boldsymbol{W}_i)_1$
 5:     $\boldsymbol{f}^{i(0)} \leftarrow$ solution to Eq. (9) for vector $\boldsymbol{v}_1^{i(0)}$
 6:     $\widetilde{\boldsymbol{W}}_i^{(0)} = \sigma_1^{i(0)} \boldsymbol{u}_1^{i(0)} \left(\boldsymbol{f}^{i(0)} \odot \boldsymbol{v}_1^{i(0)}\right)^T$
 7:     - - Apply refinements - -
 8:     for $n = 1$ to $N_{steps}$ do
 9:         - - Compute error matrix - -
10:         $\boldsymbol{E} = \boldsymbol{W}_i - \widetilde{\boldsymbol{W}}_i^{(n-1)}$
11:         - - Compute refinement - -
12:         $\left[\boldsymbol{u}_1^{i(n)}, \sigma_1^{i(n)}, \boldsymbol{v}_1^{i(n)}\right] = \mathrm{SVD}(\boldsymbol{E})_1$
13:         $\boldsymbol{f}^{i(n)} \leftarrow$ solution to optimisation problem (9) for vector $\boldsymbol{v}_1^{i(n)}$
14:         - - Update weight matrix approximation - -
15:         $\widetilde{\boldsymbol{W}}_i^{(n)} = \widetilde{\boldsymbol{W}}_i^{(n-1)} + \sigma_1^{i(n)} \boldsymbol{u}_1^{i(n)} \left(\boldsymbol{f}^{i(n)} \odot \boldsymbol{v}_1^{i(n)}\right)^T$
16:     end for
17: end for
```

Notes: $\mathrm{SVD}(\boldsymbol{X})_1$ returns the rank-1 SVD-based approximation of $\boldsymbol{X}$.

Different combinations of levels of sparsity and refinement iterations correspond to different design points in the computation-accuracy space. In this respect, the number of non-zero elements in each binary mask vector and the number of iterations are exposed to the design space exploration as tunable parameters (NZ, N_{steps}) to explore the LSTM computation-accuracy trade-off.

4.2 Architecture

The proposed FPGA architecture for LSTMs is illustrated in Fig. 2. The main strategy of the architecture includes the exploitation of the coarse-grained parallelism between the four LSTM gates and is parametrised with respect to the fine-grained parallelism in the dot-product and elementwise operations of the LSTM, allowing for a compile-time tunable performance-resource trade-off.

SVD and Binary Masks Precomputation. In Algorithm 1, the number of refinement iterations (N_{steps}), the level of sparsity (NZ) and the trained weight matrices are data-independent and known at compile time. As such, the required SVD decompositions along with the corresponding binary masks are precomputed for all N_{steps} iterations at compile time. As a result, the singular values $\sigma_1^{i(n)}$, the vectors $\boldsymbol{u}_1^{i(n)}$ and only the non-zero elements of the sparse $\boldsymbol{f}^{i(n)} \odot \boldsymbol{v}_1^{i(n)}$ are stored in the off-chip memory, so that they can be looked-up at run time.

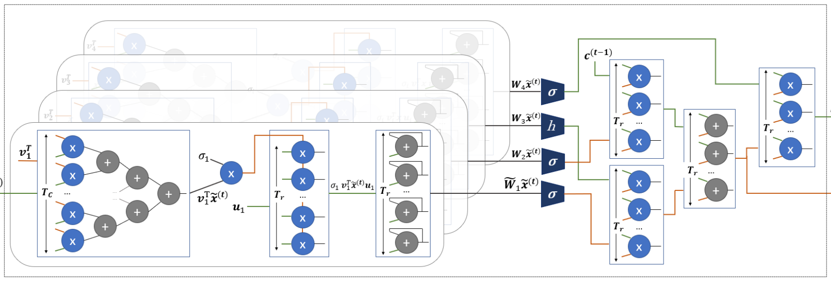

Fig. 2. Diagram of proposed hardware architecture

Inter-gate and Intra-gate Parallelism. In the proposed architecture, each gate is allocated a dedicated *hardware gate unit* with all gates operating in parallel. At each LSTM time-step t, a hardware gate unit computes its output by performing N_{steps} refinement iterations as in Eq. (12). At the beginning of the time-step, the current vector $\tilde{\boldsymbol{x}}^{(t)}$ is stored on-chip as it will be reused in each iteration by all four gates. The vectors $\boldsymbol{u}_1^{i(n)}$ and $\boldsymbol{v}_1^{i(n)}$ for each gate, along with their singular values $\sigma_1^{i(n)}$, are streamed in the architecture from the off-chip memory in a tiled manner. $\boldsymbol{u}_1^{i(n)}$ and $\boldsymbol{v}_1^{i(n)}$ are tiled with tile sizes of T_r and T_c respectively, leading to $\frac{R}{T_r}$ and $\frac{C}{T_c}$ tiles sequentially streamed in the architecture.

At each gate, a dot-product unit is responsible for computing the dot product of the current tile of $\boldsymbol{v}_1^{i(n)}$ with the corresponding elements of the input $\tilde{\boldsymbol{x}}^{(t)}$. The dot-product unit is unrolled by a factor of T_c in order to process one tile of $\boldsymbol{v}_1^{i(n)}$ per cycle. After accumulating the partial results of all the $\frac{C}{T_c}$ tiles, the result is produced and multiplied with the scalar $\sigma_1^{i(n)}$. The multiplication result is passed as a constant operand to a multiplier array, with $\boldsymbol{u}_1^{i(n)}$ as the other operand. The multiplier array has a size of T_r in order to match the tiling of $\boldsymbol{u}_1^{i(n)}$. As a final stage, an array of T_r accumulators performs the summation across the N_{steps} iterations as expressed in Eq. (12), to produce the final gate output.

The outputs from the *input*, *forget* and *output* gates are passed through a sigmoid unit while the output of the *cell* gate is passed through a *tanh* unit. After the nonlinearities stage, the produced outputs are multiplied element-by-element as dictated by the LSTM equations to produce the cell state $\boldsymbol{c}^{(t)}$ (Eq. (4)) and the current output vector $\boldsymbol{h}^{(t)}$ (Eq. (5)). The three multiplier arrays and the one adder array all have a size of T_r to match the tile size of the incoming vectors and exploit the available parallelism.

5 Design Space Exploration

Having parametrised the proposed approximation method over NZ and N_{steps} and its underlying architecture over NZ and tile sizes (T_r, T_c), corresponding metrics need to be employed for exploring the effects of each parameter on performance and accuracy. The approximation method parameters are studied based on an application-level evaluation metric (discussed in Section 5.2), that measures the impact of each applied approximation on the accuracy of the target application. In terms of the hardware architecture, roofline performance modelling is employed for exhaustively exploring the design space formed by all possible tile size combinations, to obtain the highest performing design point (discussed in Section 5.1). Based on those two metrics, the computation time-accuracy trade-off is explored.

5.1 Roofline Model

The design space of architectural configurations for all tile size combinations of T_r and T_c is explored exhaustively by performance modelling. The roofline model [17] is used to develop a performance model for the proposed architecture by relating the peak attainable performance (in terms of throughput), for each configuration on a particular FPGA device, with its operational intensity, which relates the ratio of computational load to off-chip memory traffic. Based on this model, each design point's performance can be bounded either by the peak platform throughput or by the maximum performance that the platform's memory system can support. In this context, roofline models are developed for predicting the maximum attainable performance for varying levels of pruning (NZ).

Given a tile size pair, the performance of the architecture is calculated as:

$$Perf(ops/s) = \frac{workload(ops/input)}{II(cycles/input)} clk = \frac{4N_{steps}(2NZ+2R+1)+37R}{max(N_{steps} max(\frac{R}{T_r}, \frac{NZ}{T_c}), 37\frac{R}{T_r})} clk \quad (13)$$

where each gate performs $2NZ+2R+1$ operations per iteration and $37R$ accounts for the rest of the operations to produce the final outputs. The initiation interval (II) is determined based on the slowest between the gate stage and the rest of the computations. Similarly, a gate's initiation interval depends on the slowest between the dot-product unit and the multiplier array (Fig. 2).

Respectively, the operational intensity of the architecture, also referred to in the literature as Computation-to-Communication ratio (CTC), is formulated as:

$$CTC(ops/byte) = \frac{operations(ops)}{mem\ access(bytes)} = \frac{4N_{steps}(2NZ + 2R + 1) + 37R}{4(4N_{steps}(NZ + R + 1) + 2R)} \quad (14)$$

where the memory transfers include the singular vectors and the singular value for each iteration of each gate and the write-back of the output and the cell state vectors to the off-chip memory. The augmented input vector $\tilde{\boldsymbol{x}}^{(t)}$ is stored on-chip in order to be reused across the N_{steps} iterations. All data are represented with a single-precision floating-point format and require four bytes.

The number of design points allows enumerating all possible tile size combinations for each number of non-zero elements and obtaining the performance and CTC values for the complete design space. Based on the target platform's peak performance, memory bandwidth and on-chip memory capacity, the subspace containing the platform-supported design points is determined. The proposed architecture is implemented by selecting the tile sizes (T_r, T_c) that correspond to the highest performing design point within that subspace.

5.2 Evaluating the Impact of Approximations on the Application

The proposed framework requires a metric that would enable measuring the impact of the applied approximations on the application-level accuracy for different (NZ, N_{steps}) pairs. In our methodology, the error induced by our approximation methods is measured by running the target application end-to-end over an evaluation set with both the approximated weight matrices given a selected (NZ, N_{steps}) pair and with the original pretrained LSTM, acting as a reference model. By treating the output of the reference model as the ground truth, an application-specific metric is employed that assesses the quality of the output that was generated by the approximate model, exploring in this way the relationship between the level of approximation and the application-level accuracy.

6 Evaluation

The image captioning system presented by Vinyals et al. [5] (winner of the 2015 MSCOCO challenge) is examined as a case study for evaluating the proposed framework. Input images are encoded by a CNN and fed to a trained LSTM model to predict corresponding captions. In the proposed LSTM, each gate consists of two $R \times R$ weight matrices, leading to a $(R \times C)$ augmented weight matrix per gate with $R = 512$ and $C = 2R$, for a total of 2.1 M parameters. To determine the most suitable approximation scheme, we use a subset of the validation set of the Common Objects in Context (COCO) dataset[1], consisting of 35 images. To obtain image captions that will act as ground truth for the evaluation of the proposed approximation method, the reference image captioning application is executed end-to-end over the evaluation set, using TensorFlow[2]. As a metric of the effect of low-rank approximation and pruning on the LSTM model, we select Bilingual Evaluation Understudy (BLEU) [18], which is commonly employed for the evaluation of machine translation's quality by measuring the number of matching words, or "blocks of words", between a reference and a candidate translation. Due to space limitations, more information about adopting BLEU as a quality metric for image captioning can be found in [5].

[1] http://cocodataset.org

[2] https://www.tensorflow.org

Experimental Setup. In our experiments, we target the Xilinx Zynq ZC706 board. All hardware designs were synthesised and placed-and-routed with Xilinx Vivado HLS and Vivado Design Suite (v17.1) with a clock frequency of 100 MHz. Single-precision floating-point representation was used in order to comply with the typical precision requirements of LSTMs as used by the deep learning community. Existing work [7][12] has studied precision optimisation in specific LSTM applications, which constitutes a complementary method to our framework as an additional tunable parameter for the performance-accuracy trade-off.

Baseline Architecture. A hardware architecture of a faithful implementation of the LSTM model is implemented to act as a baseline for the proposed system's evaluation. This baseline architecture consists of four gate units, implemented in parallel hardware, that perform matrix-vector multiplication in a tiled manner. Parametrisation with respect to the tiling along the rows (T_r) and columns (T_c) of the weight matrices is applied to this architecture and roofline modelling is used to obtain the highest performing configuration (T_r, T_c), similarly to the proposed system's architecture (Fig. 3). The maximum platform-supported attainable performance was obtained for $T_r = 2$ and $T_c = 1$, utilising 308 DSPs (34%), 69 kLUTs (31%), 437 kFFs (21%) and 26 18kbit BRAMs (2%). As Fig. 3 demonstrates, the designs are mainly memory bounded and as a result not all the FPGA resources are utilised. To obtain the application-level accuracy of the baseline design under time constrained scenarios, the BLEU of the intermediate LSTM output at each tile step of T_r is examined (Fig. 4).

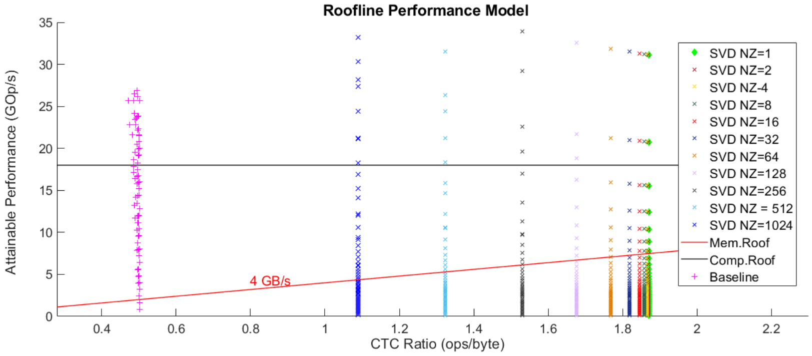

Fig. 3. Roofline model of the proposed and baseline architectures on the ZC706 board

6.1 Comparisons at Constrained Computation Time

This section presents the gains of using the proposed methodology compared to the baseline design under computation time constraints. This is investigated by exploring the design space, defined by (NZ, T_r, T_c), in terms of (i) performance (Fig. 3) and (ii) the relationship between accuracy and computation time (Fig. 4). As shown in Fig. 3, as the level of pruning increases and NZ becomes smaller, the computational and memory load per refinement iteration becomes smaller and the elementwise operations gradually dominate the computational intensity (Eq. (14)), with the corresponding designs moving to the right of the roofline graph. With respect to the architectural parameters, as the tiling parameters T_r and T_c increase, the hardware design becomes increasingly unrolled and moves towards the top of the roofline graph. In all cases, the proposed architecture demonstrates a higher performance compared to the baseline design reaching up to 3.72× for a single non-zero element with an average of 3.35× (3.31× geo. mean) across the sparsity levels shown in Fig. 3.

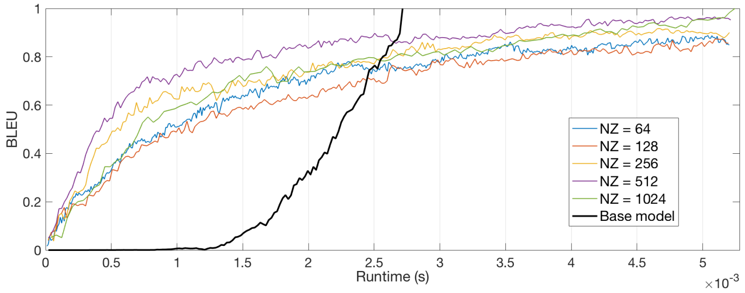

Fig. 4. BLEU scores over time for all methods

To evaluate our methodology in time-constrained scenarios, for each sparsity level the highest performing design of the roofline model is implemented. Fig. 4 shows the achieved BLEU score of each design over the evaluation set with respect to runtime, where higher runtime translates to higher number of refinements. In this context, for the target application the design with 512 non-zero elements (50% sparsity) achieves the best trade-off between performance per refinement iteration and additional information obtained at each iteration. The highest performing architecture with NZ of 512 has a tiling pair of (32, 1) and the implemented design consumes 862 DSPs (95%), 209 kLUTs (95%), 437 kFFs (40%) and 34 18kbit BRAMs (3%). In the BLEU range between 0.4 and 0.8, our proposed system reaches the corresponding BLEU decile up to 6.51× faster with an average speedup of 4.19× (3.78× geo. mean) across the deciles.

As demonstrated in Fig. 4, the highest performing design of the proposed method (NZ=512) constantly outperforms the baseline architecture in terms of BLEU score at every time instant up to 2.7 ms, at which a maximum BLEU value of 0.9 has been achieved by both methods. As a result, given a specific time budget below 2.7 ms, the proposed architecture achieves a 24.88× higher BLEU score (geo. mean) compared to the baseline. Moreover, the proposed method demonstrates significantly higher application accuracy during the first 1.5 ms of the computation, reaching up to 31232× higher BLEU. In this respect, our framework treats a BLEU of 0.9 and a time budget of 2.7 ms as switching points to select between the baseline and the architecture that employs the proposed approximation method and deploys the highest performing design for each case.

7 Conclusion

The high-performance deployment of LSTMs under stringent computation time constraints poses a challenge in several latency-critical applications. This paper presents a framework for mapping LSTMs on FPGAs in such scenarios. The proposed methodology applies an iterative approximate computing scheme in order to compress and prune the target network and explores the computation time-accuracy trade-off. A novel FPGA architecture is proposed that is tailored to the degree of approximation and optimised for the target device. This formulation enables the co-optimisation of the LSTM approximation and the architecture in order to satisfy the application-level computation time constraints. Future work includes the extension of the proposed methodology to scenarios where the training data are available to perform retraining, leading to even higher gains.

8 Acknowledgements

The support of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/L016796/1) is gratefully acknowledged. This work is also supported by EPSRC grant 1507723.

References

1. W. Byeon, T. M. Breuel, F. Raue, and M. Liwicki, "Scene Labeling with LSTM Recurrent Neural Networks," in *CVPR*, 2015, pp. 3547–3555.
2. K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, "DRAW: A Recurrent Neural Network For Image Generation," in *ICML*, 2015, pp. 1462–1471.
3. A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human Trajectory Prediction in Crowded Spaces," in *CVPR*, 2016.
4. S. Otte *et al.*, "Recurrent Neural Networks for Fast and Robust Vibration-based Ground Classification on Mobile Robots," in *ICRA*, 2016, pp. 5603–5608.
5. O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge," *TPAMI*, pp. 652–663, 2017.
6. J. Donahue *et al.*, "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," *TPAMI*, vol. 39, no. 4, pp. 677–691, 2017.
7. S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu, "FPGA Acceleration of Recurrent Neural Network Based Language Model," in *FCCM*, 2015, pp. 111–118.
8. E. Nurvitadhi *et al.*, "Accelerating Recurrent Neural Networks in Analytics Servers: Comparison of FPGA, CPU, GPU, and ASIC," in *FPL*, 2016, pp. 1–4.
9. J. Chung *et al.*, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," in *NIPS Workshop on Deep Learning*, 2014.
10. A. X. M. Chang and E. Culurciello, "Hardware Accelerators for Recurrent Neural Networks on FPGA," in *ISCAS*, 2017, pp. 1–4.
11. Y. Guan, Z. Yuan, G. Sun, and J. Cong, "FPGA-based Accelerator for Long Short-Term Memory Recurrent Neural Networks," in *ASP-DAC*, 2017, pp. 629–634.
12. S. Han *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *FPGA*, 2017, pp. 75–84.
13. Z. Wang, J. Lin, and Z. Wang, "Accelerating Recurrent Neural Networks: A Memory-Efficient Approach," *TVLSI*, vol. 25, no. 10, pp. 2763–2775, oct 2017.
14. X. Zhang *et al.*, "High-Performance Video Content Recognition with Long-Term Recurrent Convolutional Network for FPGA," in *FPL*, 2017, pp. 1–4.
15. K. He and J. Sun, "Convolutional Neural Networks at Constrained Time Cost," in *CVPR*, 2015.
16. M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas, "Predicting Parameters in Deep Learning," in *NIPS*, 2013, pp. 2148–2156.
17. S. Williams *et al.*, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.
18. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A Method for Automatic Evaluation of Machine Translation," in *ACL*, 2002, pp. 311–318.