Space Station Power Forecasting with LSTMs on FPGAs

by

Joseph Richard Kocik

B.S. Computer Engineering, University of Pittsburgh, 2019

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

Master of Science in Electrical and Computer Engineering

University of Pittsburgh

2021

Space Station Power Forecasting with LSTMs on FPGAs

Joseph Richard Kocik, M.S.

University of Pittsburgh, 2021

Autonomous operations of space systems is an important and difficult task that will become even more imperative as space missions become increasingly remote. Accurate prediction of telemetry data can improve system monitoring and facilitate fault detection. This thesis presents a methodology for the acceleration of short-term forecasting of power data on an embedded platform designed for space. Initially, a long short-term memory (LSTM) network is trained to forecast voltage and current values from the International Space Station. This LSTM forecasts voltage and current minutes into the future while maintaining a low error rate. This LSTM network's weights and biases are then used to create a new accelerated network which can be deployed on the FPGA of a Zynq-7045 system on a chip (SoC). The Zynq-7045 was selected because it is the same SoC used on the SHREC Space Processor, a space computer targeted for this study. A number of networks of varying sizes and history lengths are realized in hardware and evaluated against a software baseline. These networks were designed to be deployed on the resource-constrained FPGA fabric of the Zynq-7045 while maintaining the LSTM network architecture. The best performing LSTM networks were able to achieve over 3$\times$ speedup against a software baseline with minimal increase in forecasting error.

Table of Contents

List of Tables

List of Figures

Preface

This work was supported by SHREC industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783.

I would like to thank my mom and dad for their support, advice, and encouragement throughout my educational career. I would also like to thank Toby for her encouragement, love, and for always being there when things got tough. I could not have done this without you guys.

1.0 Introduction

Missions pushing farther into deep space will define modern space exploration. The proposed Gateway space station's lunar orbit will be the first of its kind, placing a human-occupied station farther away from Earth than ever before. Gateway will have higher communication latency and be more difficult to command than the International Space Station (ISS) due to its remote location. Gateway is also planned to be crewed intermittently and will be required to handle extended periods of autonomous operation. Therefore, strong onboard monitoring is necessary for autonomous station control. The ability to accurately forecast the future state of the station can further facilitate control operations. Significant strides have been made using machine-learning (ML) methods to enhance forecasting and fault prediction for terrestrial power systems [1]. These same techniques can be applied to systems in space, including crewed vehicles such as Gateway or the ISS.

To forecast spacecraft telemetry, a ML structure amenable to time-series prediction is useful. Recurrent neural networks (RNNs) are a type of ML network that are well-suited for use with time-series datasets [2]. Long short-term memory (LSTM) networks are a type of RNN that allow structured forgetting of information, improving their performance over baseline RNN models. LSTMs maintain the strength of RNNs in time-series applications and were selected for this research because of their forecasting abilities and potential for acceleration. They have been shown to be effective for forecasting both short-term power system states and spacecraft telemetry [1, 3].

LSTMs, however, can be difficult to deploy on embedded platforms for space. LSTM do contain parallelizable structures that allow acceleration [4]. However, LSTM networks contain fully-connected layers and require the associated weights to be stored. This can make LSTMs difficult to implement due to the large amount of memory and device resources required to instantiate an accelerated network. Modern hybrid space computers, which combine reliable radiation-hardened components with commercial-off-the-shelf (COTS) components, can alleviate some of these issues by supporting the use of high-performance computers in space. Even so, careful attention must be given to ensure that LSTM network implementations are small enough to be successfully deployed.

This thesis focuses on exploiting the strengths of ML methods to produce a start-to-finish methodology for LSTM creation and acceleration. Using time-series data collected from the ISS, an application was created showcasing the potential of LSTMs for critical telemetry forecasting on data from space stations. LSTM networks were trained to forecast future voltage and current levels on the order of minutes into the future. The training of these LSTM networks was performed using TensorFlow. LSTM network parameters, weights, and biases were then exported and used to create FPGA-accelerated networks. Networks were then deployed on a ZC706 (Zynq-7045) embedded system, which served as an emulation platform for the SHREC Space Processor (SSP), an embedded system designed for space [5]. The accelerated LSTM networks were then profiled on their resource utilization and speedup in comparison to a software-only baseline.

2.0 Related Work

Numerous methods for accelerating LSTMs exist, but not all are amenable to low-resource platforms like the SSP. One example of LSTM acceleration using FPGAs was outlined by Guan et al. in their creation of an accelerated speech recognition model [6]. Their process includes storing network weights and biases in on-chip block random-access memory (BRAM) instead of DDR to improve performance through closer data proximity. They also used parallelized LSTM operations and buffered input and output to accelerate their network. This design did not employ any quantization, as the authors were concerned about the effects of fixed-point error. This approach was able to achieve over 20× speedup on a Xilinx VC707 board with a Virtex7-485t FPGA system on a chip (SoC) when compared to a software-only application deployed on Intel Xeon CPU. The design was profiled on a relatively large FPGA and CPU and used a large proportion of the VC707's resources. The design used more BRAM and digital signal processing (DSP) slices than are even available on smaller platforms like the SSP. However, the data storage methodology and parallel optimizations were successful and applicable to smaller devices, albeit in a more modest fashion, to reduce resource utilization.

Another example of accelerated LSTMs was summarized by Sun et al. in their project which accelerated forecasting of aircraft flight data using FPGAs [4]. Their methodology included running matrix multiplications in parallel and using a piecewise linear approximation for sigmoid activation functions. They also outlined a technique for using piecewise linear approximation to compute hyperbolic tangents as well, but this was not implemented due to its impact on anomaly detection being considered unacceptable by the authors. The FPGA-accelerated implementation, on a Xilinx VC707, yielded a 28.8× speedup over a software-only version. However, this network, similarly to the research from [6], uses the large Virtex7-485t SoC. Sun et al.'s network once again uses more DSPs than are available on the SSP [4]. In addition, this design used a large amount of lookup tables (LUTs), again more than are available on the SSP, a device more feasible for space applications. Accelerating via parallelized matrix multiplication, however, was very applicable to the SSP, although dot products were used instead due to network dimensions. Segmented approximation of activation functions was also explored, but precomputed memory banks were favored over piecewise linear functions to reduce DSP usage. Additionally, Sun et al.'s use case, aircraft telemetry prediction, is rather similar to the goal of spacecraft power telemetry prediction.

A more complex scheme for acceleration of an LSTM network was outlined by Rybalkin et al. [7]. Their research focuses on the hardware acceleration of a Bidirectional LSTM (BiLSTM), an LSTM that presents its sequence both forward and backward to separate hidden layers. This network relied heavily on quantization and hardware-aware training to reduce network size. Using hardware-aware training, the authors achieved high-accuracy optical character recognition with low power and low resource utilization for their BiLSTM. This methodology created exceptionally small networks that could fit on even small embedded devices. However, it required specialized training techniques to avoid a negative impact on accuracy due to network quantization. Because of these rigid constraints on network design and training, our work instead focused on creating more generalized accelerator. Instead this thesis focused on preserving the network architecture of the LSTM while creating an FPGA-accelerated design that could be deployed on an embedded system for space, such as the SSP.

In addition to LSTM acceleration research, their has been research specifically on methods to effectively accelerate activation functions used in ML networks. Meher et al. and Ngah et al. are examples of such research, focused on sigmoid and hyperbolic tangent, respectively [8][9]. In [8], different methods for approximating sigmoids were investigated ranging from a simple LUT design, PL approximation, piecewise nonlinear approximation, and a new methodology which combines second-order nonlinear function approximation and a differential LUT with stored deviations. The new methodology provided greater accuracy and reduced storage but at greater computational complexity than simpler methods such as a pure LUT design. [9] focuses on optimizing a LUT design by using combinational logic and leveraging the symmetry of hyperbolic tangent, as well as using piecewise linear approximation for small values. These works outline different methodologies for implementing activation functions in ways amenable to hardware systems. These techniques can be modified and applied to FPGA systems to aid in accelerating the activation functions used in

LSTM networks. This research focused on the simple LUT design outlined in [8], favoring its lack of complexity and resource-light implementation.

LSTM's potential to excel in power system forecasting has also been shown. Tan et al. applied LSTMs for ultra-short-term forecasting (on the order of minutes) of power demand [1]. Their design employed a two-layer LSTM network to capture patterns in time-series sequences of power demand data. The authors also proposed an enhanced training scheme for their LSTMs. LSTMs trained in this way were compared to an array of state-of-the-art prediction algorithms in terms of their error in predicting power system state, with target lengths of one to five minutes into the future. Their networks were more accurate for all target lengths. A standard LSTM model and LSTM-based methodology called sequence-to-sequence, outlined by Marino et al. in [10], closely followed the method proposed here. Their more accurate network is of particular relevance to this research. In this power prediction task, LSTM models, even without augmentation, outperformed other ML networks and achieved consistently lower error than deep belief networks or networks using support-vector regression.

LSTMs have not only shown promise for terrestrial power forecasting, but have also demonstrated their capabilities in space. LSTMs were one of a variety of telemetry forecasting methods applied to data from the Egyptsat-1 mission in [3]. Three parameters from the satellite's power system were forecasted: power bus voltage, load current, and battery temperature. In terms of error rate, LSTMs outperformed multilayer perceptrons, basic RNNs, and gated recurrent units. The classical time-series-forecasting model, autoregressive integrated moving average, was the only method to beat LSTMs. However, increasing dataset sizes has been shown to improve performance of LSTM networks in the the research of Fischer and Krauss in [11] which trained LSTM networks using 15 years of financial data. The authors of [3] believed using a dataset larger than Egyptsat-1's three years of telemetry data could improve these results. Ibrahim et al.'s usage of LSTMs for forecasting power system data was indicative of their potential for power forecasting on space stations such as ISS or Gateway. Additionally, while the ISS dataset used in this research is limited in duration, its focus on short-term power prediction gives us significantly more data points.

Even with only weeks of data, this study obtained over three times as many time steps as the Egyptsat-1 dataset.

3.0 Background

In this section, RNNs and specifically LSTMs will be outlined. Additionally, the ISS power system dataset will be explored. Finally, the target embedded platform used in this research is considered.

3.1 RNNs

RNNs are a type of neural network that retain state information from previous inferences and use this information to generate future predictions [12]. RNNs receive sequential data and store past information as a recurrently passed hidden state. Each cell of an RNN feeds a weighted summation of the hidden states and input into an activation function, oftentimes the hyperbolic tangent function. The recurrent nature of RNNs makes them well suited for complex sequence-detection applications.

3.2 LSTMs

LSTM networks, a variant of RNNs, remedy the pitfalls of RNNs by addressing the vanishing-gradient problem that occurs in vanilla RNNs [2, 13]. LSTMs inherit the same general structure as an RNN, with a recurrently passed hidden state value. However, unlike RNNs, LSTM units use arithmetic combinations of a forget gate (f), input gate (i), output gate(o), and candidate ($\tilde{c}$), as shown in Figure 1 [6]. The candidate values and gates receive weighted sequential inputs (x_t) and weighted previous hidden states (h_{t-1}). LSTM units contain two types of activation functions with gates typically using a sigmoid function and the candidate value using a hyperbolic tangent function. LSTM units then use arithmetic combinations of the gates and candidate outputs as well as previous cell state (c_{t-1}) to generate cell state (c_t) and hidden state (h_t). Hidden state and cell state are then recurrently

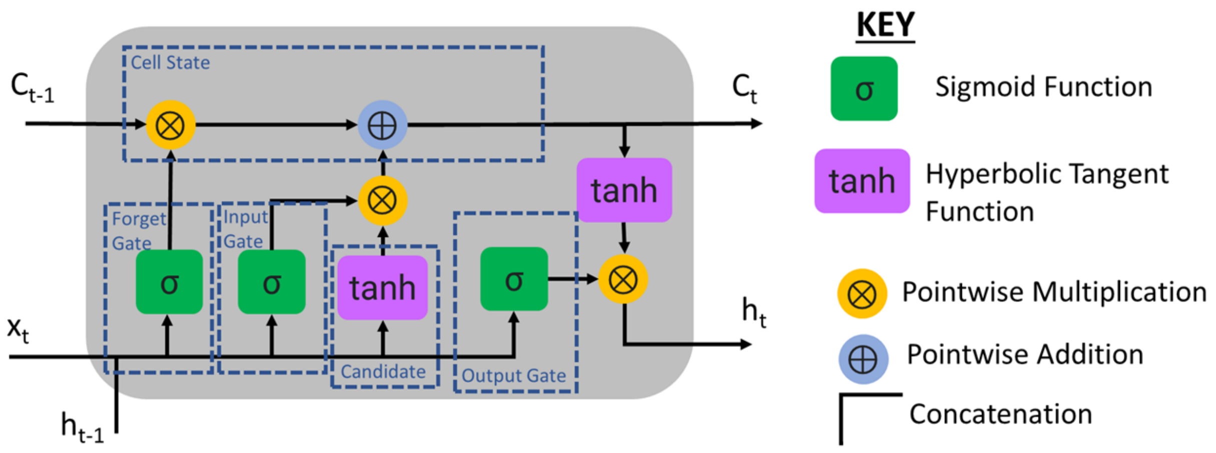

Figure 1: LSTM unit structure displaying data flow and gates.

passed and the process repeats until an output is needed. The hidden state is then used to compute the output.

Single LSTM units are combined to create a higher-dimensional network. By receiving and using a vector of previous hidden states from all units' outputs, more complex sequence detection is enabled. LSTM networks can be used in a few ways. This project focuses on using them to create short-term forecasts based on a brief history. Oftentimes, a dense layer is placed after an LSTM layer to compute the final predicted output. In this research, a single dense layer is used to compute a weighted summation of the hidden-state outputs from all units in an LSTM network. This dense layer is trained in concert with the weights of the LSTM units and is treated as part of the overall LSTM forecasting network.

3.3 ISS Power System

Operation of the ISS is dependent on many interconnected systems, including the power system. The ISS is located in low Earth orbit, where it collects solar energy as its means of power generation. Eight solar arrays are used to generate power for the space station [14].

The ISS orbits the Earth every 92 minutes [15]. Each array outputs power at 160-V DC to primary networks. The power generated by the solar array is then converted to the 120-V DC and 28-V DC for American and Russian secondary networks, respectively.

For this research, a dataset with power system information from the ISS was used. Electrical power system information was collected in real-time from the ISS Live public server [14]. Both voltage and current levels were collected from the ISS's solar arrays before they were distributed to primary networks. This research focused specifically on the voltage and current levels of a single solar array, 1A. Data was collected with a frequency of once per minute. This frequency was chosen as it was high-enough granularity for short-term forecasting on the order of minutes.

3.4 CHREC Space Processor

The CHREC Space Processor (CSP) is a hybrid space computer developed at the National Science Foundation (NSF) Center for High-Performance and Reconfigurable Computing (CHREC) [16]. The CSP contains a Zynq-7020, a 7-Series System on a Chip (SoC), with both dual ARM Cortex-A9 fixed logic and Artix-7 programmable logic fabric. This combination of CPU and FPGA allows for mixed computation and hardware acceleration of applications. A hybrid system architecture also allows the combination of commercial-of-the-shelf (COTS) components with radiation-hardened (rad-hard) components. This allows for lower cost designs using COTS components without an excessive sacrifice of reliability.

This combination of hybrid processor for acceleration and a hybrid-system architecture has allowed the CSP to be successfully deployed on the ISS for two of the Department of Defense's Space Test Program (STP) missions. In the STP-H5/ISEM (STP-Houston 5/ISS SpaceCube Experiment Mini) payload, a dual-CSPv1 flight-box was deployed as a submodule. The STP-H5/ISEM mission's CSP proved its reliability operating from its initial launch in February 2017 [17]. CSP received further flight heritage on the Spacecraft Supercomputing for Image and Video Processing (SSIVP) experiment aboard the ISS as part of the

STP-H6 pallet [18]. SSIVP expanded the work of its predecessor by including five CSPv1 cards, which were used as compute nodes.

3.5 SHREC Space Processor

The SHREC Space Processor (SSP) is a hybrid space computer developed by the Center for Space, High-Performance, and Resilient Computing (SHREC) [5]. The SSP builds upon the techniques and design philosophy of the CSP [16]. The SSP contains a Zynq-7045 SoC, which includes a dual-core ARM Cortex-A9 CPU and an Kintex-7 FPGA. The FPGA fabric area available in the Zynq-7045 on the SSP is substantially larger than that of the Zynq-7020 on the CSP, containing significantly more computing resources. This allows for more flexibility in the type and scale of applications that can be accelerated. The SSP also incorporates the hybrid reliability architecture combining low cost COTS components with radiation-hardened monitoring and management components.

The SSP is a state-of-the-art space computer, and as such does not have flight heritage like its predecessor, the CSP [17, 18]. The SSP is, however, slated for an upcoming mission onboard the ISS as part of STP-H7-CASPR [5]. STP-H7-CASPR will feature a CSP computer as a central node as well as two SSPs as processing nodes for space app execution. This mission will serve as a flight test for the SSP and allow for observation of the SSP's reliability and potential for future missions.

4.0 Approach

In this section, the approach for this research is outlined. First, LSTM networks for short-term voltage and current prediction are designed and trained on a PC. These networks are then exported and accelerated using high-level synthesis (HLS). Finally, the accelerated networks are executed and evaluated on the ZC706 embedded platform.

4.1 LSTM Network Creation

The initial prototyping of LSTM networking design was performed using Python and the TensorFlow ML library with the Keras deep-learning API [19, 20]. Networks were trained using the dataset outlined in Section 3.3. From this dataset, over 1,100 hours of data were used for training, and the remaining approximately 260 hours of data were used for validation. Networks were trained separately for both voltage and current prediction. Current and voltage data from solar array 1A were used. Each network was trained for 50 epochs with 1000 evaluations per epoch. The Adam optimizer, with a learning rate of 0.001, was employed for training, and mean absolute error (MAE) was used as the loss function. Networks were designed with an input layer, an LSTM layer, and a dense output layer. LSTM units were all created using the conventional LSTM structure with sigmoid and hyperbolic tangent as activation functions.

4.2 Network Acceleration in Vitis

After networks had been created on a PC, their weights and biases were exported. These weights and biases were then used to create an accelerated LSTM network using the Xilinx Vitis platform [21]. The accelerated LSTM network was instantiated on the Zynq-7045's programmable logic (PL), with weights, biases, network size, and dimensions included as a part of compilation. The processing system served as a controller for network execution and manged data flow.

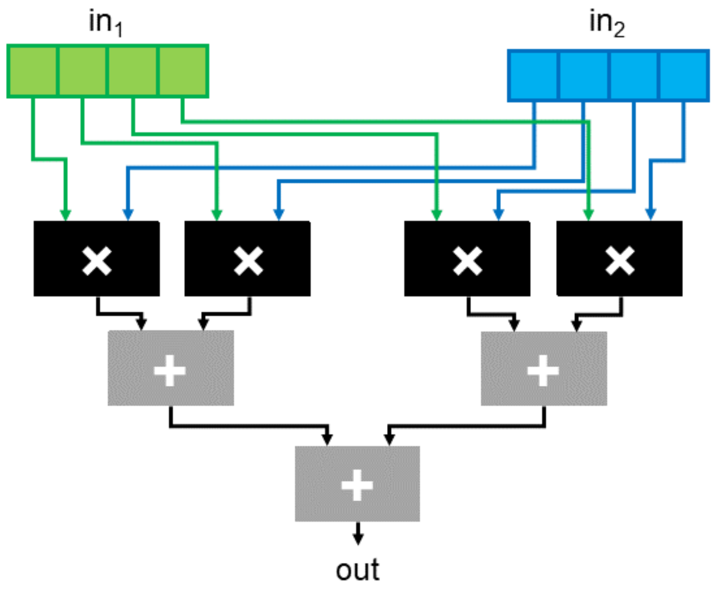

Figure 2: Dot product of two vectors (length four) using an adder tree of width four

Acceleration was primarily performed on the level of the LSTM unit. Individual LSTM units were accelerated in a number of ways. The forget gate, input gate, output gate, and candidate can all execute in parallel. In order to run the candidate and gates in parallel, data is duplicated and received simultaneously. The gates and candidate contain dot products between hidden states and weights with the result added to the weighted input and bias. Dot products were computed using Xilinx's implementation of the basic linear algebra subroutines (BLAS). The FPGA-accelerated dot product uses parallel multiplication and trees of addition operations to compute the final dot product sum as shown in Figure 2. The forget, input, and output gates all feed their sums into sigmoid functions while the candidate uses a hyperbolic tangent. These operations were parallelized so that the four dot products run simultaneously followed by the activation functions, which are also calculated in parallel. An LSTM unit with these parallelized operations is shown in Figure 3.

For the activation functions, sigmoid and hyperbolic tangent, precomputed tables were used to reduce the amount of costly floating-point operations. These tables omitted the

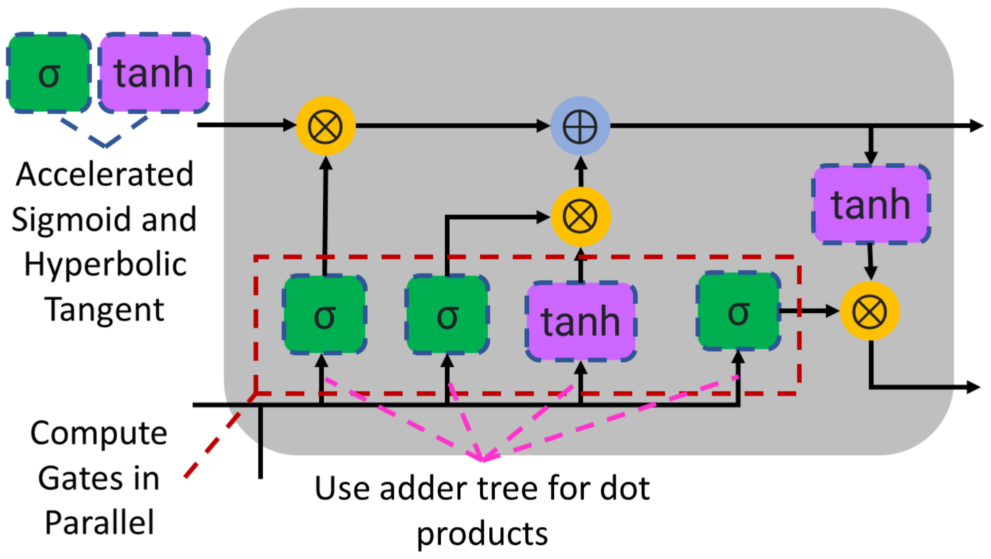
σ
tanh
Accelerated Sigmoid and Hyperbolic Tangent
σ
σ
tanh
σ
tanh
Compute Gates in Parallel
Use adder tree for dot products

Figure 3: Accelerated LSTM unit structure displaying acceleration optimizations

complex exponential math and instead simply use the input to address a precomputed memory bank. Sigmoid tables covered ranges from -6 to 6, and hyperbolic tangent tables ranged -3 to 3. Inputs outside of these ranges were clamped to their asymptotic limits. These ranges were chosen as they were able to capture the values within the activation functions effectively. The sizes of the tables were variable to allow for the analysis of the trade-off between accuracy and execution time. Figure 4 displays the sigmoid and hyperbolic tangent functions as approximated using a 32-entry table.

4.3 LSTM Implementation using HLS

The parallelized LSTM network that was defined above was then implemented using Xilinx's Vitis unified software platform. The ZC706 embedded platform was targeted as it contains the same Zynq-7045 SoC as the SSP. Vitis was used to compile both the software programs and accelerated hardware kernels. The host CPU code was cross-compiled for the ARM Cortex-A9 processor. The accelerated LSTM network is treated as a hardware kernel

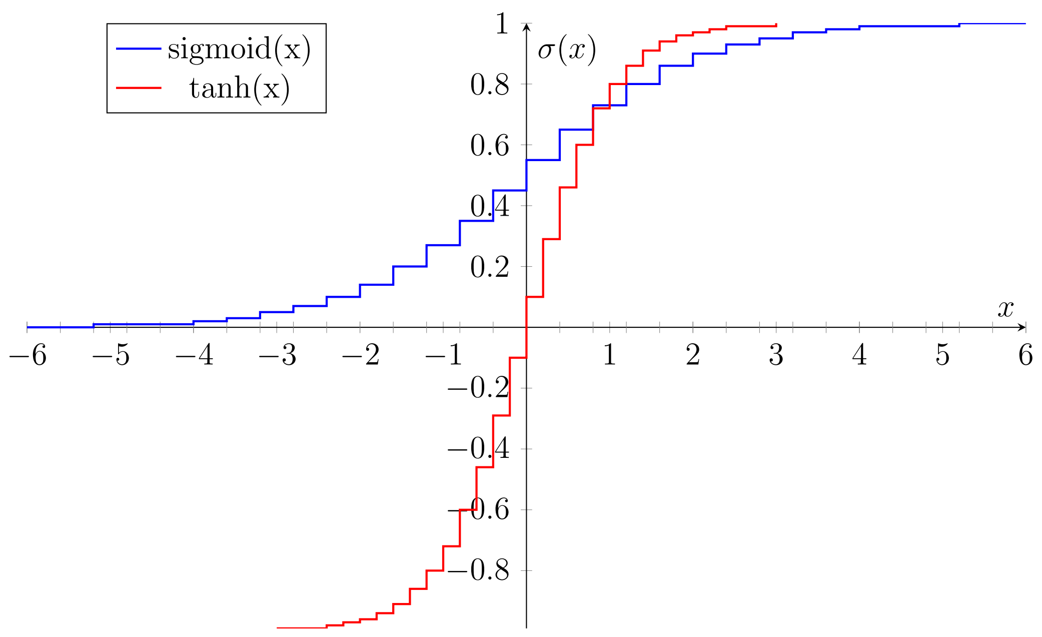

Figure 4: Sigmoid and hyperbolic tangent functions approximated with 32-entry tables

and compiled using Vitis HLS for the Kintex-7. System loops were pipelined and parallel sections were enumerated, as outlined in the previous section, using HLS. Vitis HLS then compiled the network to use the resources of the Zynq-7045. The compiled LSTM network design was then incorporated with a base system design created using Xilinx's Vivado. This yielded a complete design, with the CPU and LSTM hardware kernel connected via an Advanced eXtensible Interface (AXI) interconnect.

4.4 Network Executables and Runtime Environment

During LSTM network compilation by Vitis, a bitstream for running on an FPGA as well as the boot files necessary for running Petalinux were also created. Petalinux is an embedded Linux operating system (OS) provided by Xilinx and built using Yocto. It is designed to interface with Vivado and Vitis projects, and was included within the Vitis build. By combining a Petalinux project with a base Vivado design and an accelerator design in Vitis HLS, Vitis was used to build the OS in addition to the hardware bitstream and software executables.

The files for the first-stage bootloader, U-Boot primary bootloader, Petalinux Image, and compiled bitstream all were constructed by the combined Petalinux and Vitis tools. These files, as well as a baseline software-only LSTM network executable and an accelerated LSTM network executable, were loaded onto an SD card. This SD card was connected to a ZC706 board, where the bitstream was loaded into programmable logic and the Petalinux OS was booted. The SD card was also partitioned to include a full Linux file system. This file system was created with Petalinux and compiled to include all Xilinx libraries necessary for the accelerated LSTM application.

4.5 LSTM Network Testing

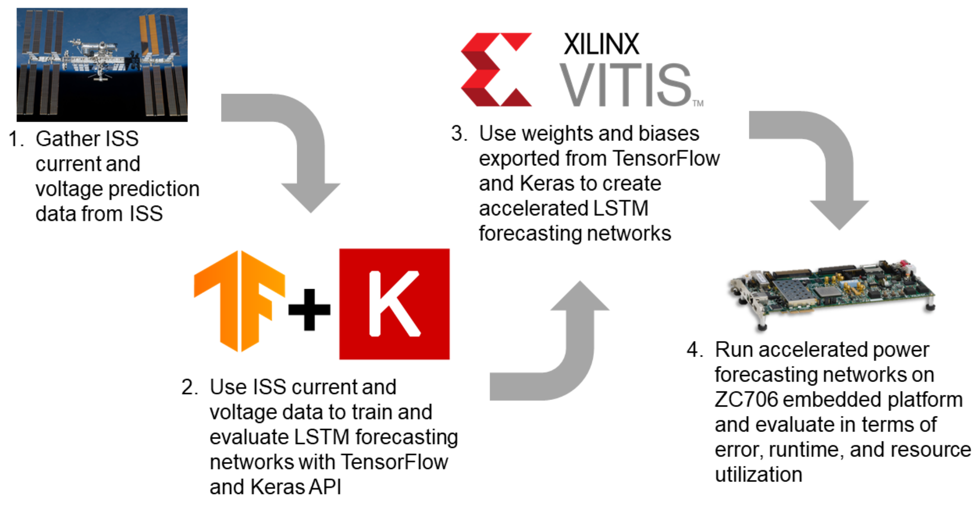
1. Gather ISS current and voltage prediction data from ISS
2. Use ISS current and voltage data to train and evaluate LSTM forecasting networks with TensorFlow and Keras API
XILINX VITIS
3. Use weights and biases exported from TensorFlow and Keras to create accelerated LSTM forecasting networks
4. Run accelerated power forecasting networks on ZC706 embedded platform and evaluate in terms of error, runtime, and resource utilization

Figure 5: Toolchain for creation and deployment of LSTM forecasting networks.

Using the previously outlined techniques, the full network creation and acceleration process was completed. The full process of LSTM acceleration is outlined in Figure 5. Various networks were designed and tested in order to explore optimal LSTM size and parameters for current and voltage prediction. First history length (number of input timesteps) and target length (number of output timesteps) were selected. Then the number of LSTM units was selected. The process of compiling a network followed as enumerated previously. The network was trained using Python and TensorFlow with the Keras API. This process also allows a preliminary analysis of forecasting error as performance is also evaluated during training.

Once the performance of the network was sufficient, its weights and biases were exported. The weights and biases were incorporated into the accelerated LSTM program created with Vitis. The history length, target length, and number of units in the LSTM network were also exported and instantiated as parameters in the accelerated network. With this information, a fully accelerated network was compiled with the Vitis tools. This created the executables and environment required for testing the network on the ZC706 embedded platform.

Tested networks included a variety of history and target lengths as well as network sizes. History lengths of 15, 25, and 35 minutes were all tested. Networks were designed for short-term forecasting, so target lengths of one, five, and ten minutes were used. Different network sizes were also explored. 32-unit and 64-unit networks were used across all history and target lengths. Different activation function table sizes were also investigated with lengths of 8, 16, 24, and 32 entries. Each of these networks had to be compiled individually in addition to a corresponding software-only baseline.

Once these networks were compiled, their files were placed on an SD card and loaded onto a ZC706 evaluation board. The bitstream was loaded into the programmable logic of the Zynq-7045 SoC and execution was performed after booting Petalinux. The software baselines were run as a typical executable using only the ARM CPU. The execution process of accelerated networks was similar, with the additional component of a hardware binary. This hardware binary allows the host executable to access the parallelized LSTM kernel which had been loaded into the FPGA fabric from a bitstream.

At execution time, metrics were recorded to profile the performance of the accelerated networks and the software baselines. The runtime of each LSTM network was recorded. In addition, forecasting error was calculated in terms of root-mean-square error (RMSE) and

MAE. Results were collected for accelerated networks and the software baselines to allow calculation of speedup as well as the effect of acceleration on forecasting error. Additionally the resource utilization of networks was obtained from Vivado at compile time.

5.0 Experimental Results

In this section, the experimental results for current and voltage forecasting LSTMs are presented. Accelerated networks and software baselines were run on the ZC706 and compared. Networks were profiled in terms of speedup, error rate, and resource utilization.

5.1 Network Resource Utilization

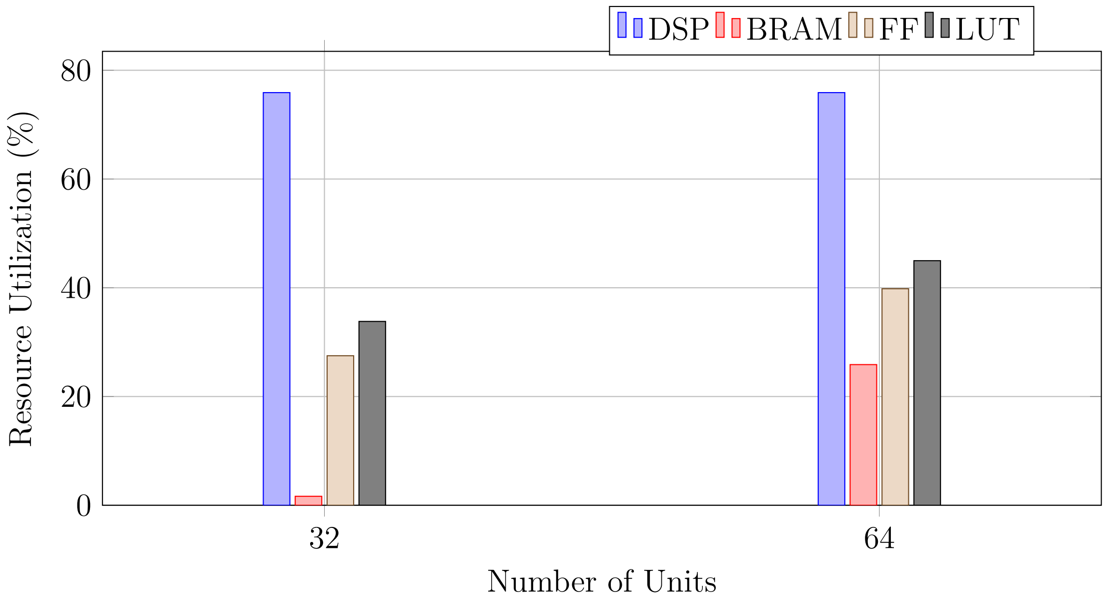

Figure 6: Resource utilization for maximum size voltage and current prediction networks.

The first obstacle in creating a network was ensuring it could be deployed on Zynq-7045 FPGA in the ZC706 SoC. Figure 6 displays the resource utilization of the largest networks for each LSTM size that could be instantiated on the ZC706. This applies to both voltage and current forecasting networks. The maximum amount of parallelization was applied to the dot products. Structurally, adder tree sizes are limited to powers of two. For 32-unit networks, fully parallelized dot products could be implemented, with all 32 inputs processed in parallel. For 64-unit networks, fully parallelized dot products required too many DSPs, and instead dot products were processed 32 inputs at a time. BRAM, flip-flop (FF), and LUT utilization was directly tied to network size, and thus as the number of units increased, so did utilization.

5.2 Power Forecasting Speedup

After determining which networks could be realized, their performance was evaluated. All tested networks had at least some degree of speedup over their software baseline. The speedup and execution time of all networks, in comparison with their software baselines, is shown in Table 1. Execution time is given on a per-inference basis to adjust for differences in number of predictions due to varied history and target lengths. As expected, execution time was faster for the smaller 32-unit networks across both the software baseline and accelerated networks. Accelerated 64-unit networks typically took 2.0× as long as their 32-unit counterparts, whereas the software baseline took over 3.7× longer. This resulted in the speedup of 3.0× for 64-unit networks and 1.6× for 32-unit networks. Execution time increased as history length grew but remained relatively constant with target length. This makes sense, as histories are input sequentially, while outputs are computed by the dense output layer. Thus, the impact of increasing output size on execution time is minimal for both software baselines and accelerated networks.

Generally, for all networks, regardless of network size, execution time increased and speedup decreased with activation function table size. Table 2 showcases the effect of activation function table size on speedup and execution time for 32- and 64-unit networks with a 15-minute history and one-minute target, but the trend was applicable across networks of varying history and target lengths. LSTM networks with eight-entry activation function tables obtained the highest speedup of over 2.0× for 32-unit networks and approaching 4.0× in 64-unit networks. The lowest speedup was obtained in networks with the largest number of entries in their activation function tables, with networks using 32-entry tables resulting in 1.6× and 3.0× speedup for 32-unit and 64-unit networks, respectively.

Table 1: Average execution time (per-inference) and speedup of voltage and current forecasting networks

Network Size (units)	**History Len. (min)**	**Target Len. (min)**	**Base Execution Time (ms)**	**Accel. Execution Time (ms)**	**Speedup**
32	15	1	1.09	0.68	1.61
	25		1.83	1.13	1.62
	35		2.54	1.58	1.60
	15	5	1.10	0.68	1.62
	25		1.82	1.13	1.60
	35		2.55	1.58	1.61
	15	10	1.11	0.68	1.63
	25		1.84	1.14	1.62
	35		2.57	1.59	1.62
64	15	1	4.13	1.35	3.05
	25		6.92	2.26	3.06
	35		9.53	3.16	3.02
	15	5	4.17	1.36	3.07
	25		6.91	2.26	3.06
	35		9.63	3.16	3.05
	15	10	4.14	1.36	3.04
	25		6.93	2.26	3.06
	35		9.58	3.17	3.03

While resource utilization was used to determine the maximum size of a network that could be instantiated on the target platform, it was still important to evaluate the benefits of using maximum size networks. LSTMs networks were tested with dot products that could process inputs of sizes ranging from two to the maximum realizable size, as defined

Table 2: Effect of activation function table size on speedup and average execution time (per-inference)

Network Size (units)	**Activation Func. Table Size**	**Base Execution Time (ms)**	**Accel. Execution Time (ms)**	**Speedup**
32	8	1.09	0.52	2.08
	16		0.58	1.87
	24		0.59	1.84
	32		0.68	1.61
64	8	4.13	1.05	3.94
	16		1.16	3.55
	24		1.18	3.49
	32		1.35	3.05

in Section 5.1, in parallel. The results of this testing are shown in Figure 7. Across networks, the maximum level of parallelism typically did not yield the highest performance. In 64-unit networks, a width of eight performed best. Both widths of four and eight performed equivalently for 32-unit networks. This performance reduction, in spite of increased parallelization, was mostly likely caused by the computation overhead diminishing performance. However, across both networks sizes, speedup was fairly equivalent regardless of dot product parallelization.

5.3 Network Prediction Error

For forecasting error, the network size as well as history and target lengths had a significant impact on accuracy in terms of both RMSE and MAE. All combinations of networks outlined in Section 4.5 were evaluated for both current and voltage prediction. For all of these networks, 32-entry tables were used for computing activation functions. For each his-

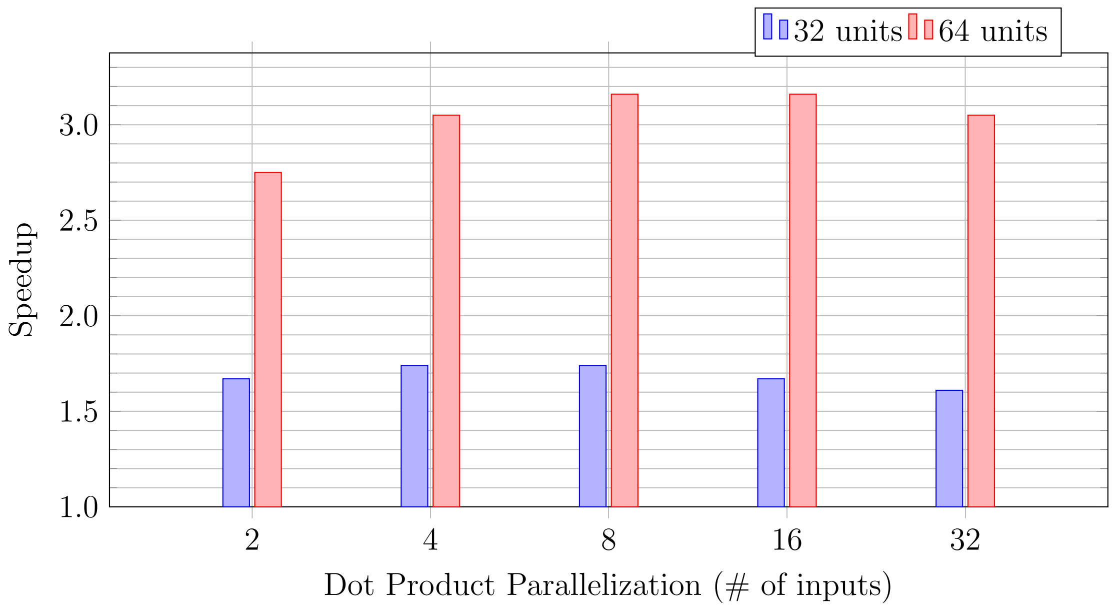

Figure 7: Effect of dot product parallelization on speedup

tory length, target length, and network size, the RMSE and MAE are given in Table 3 and Table 4 for voltage and current, respectively. Values of RMSE and MAE which were lowest for a given network size and target length are shown in bold.

For the best performing networks of each target length, the RMSE was below 1.5 A and 2.5 V and MAE below 1 A and 1.5 V. This was adequate for a system where the range in voltage exceeds 10 V and the range in current exceeds 65 A. While the range of voltage was smaller, error was greater for voltage forecasting. This greater error is due to voltage values changing less gradually and therefore being more difficult to forecast. Generally, current forecasting networks performed better than voltage forecasting networks. Additionally, the increase in error that was introduced by network acceleration was below 0.05 V or 0.07 A for both RMSE or MAE.

These results also show that the 64-unit networks outperformed 32-unit networks consistently. For every target length except one, 64-unit networks were more accurate in terms of both MAE and RMSE. Only a single 32-unit network, for the five-minute target voltage, was able to outperform its 64-unit counterpart.

Table 3: Error of current forecasting LSTM networks

Network Size (units)	**History Len. (min)**	**Target Len. (min)**	**Base RMSE (A)**	**Accel. RMSE (A)**	**Base MAE (A)**	**Accel. MAE (A)**
32	15	1	0.83	1.11	0.50	0.80
	25		0.82	1.14	0.49	0.83
	35		0.84	**1.00**	0.51	**0.69**
	15	5	1.18	**1.37**	0.75	**0.96**
	25		1.18	1.59	0.74	1.20
	35		1.20	1.42	0.76	1.02
	15	10	1.44	1.63	0.93	1.15
	25		1.45	1.59	0.94	1.11
	35		1.46	**1.58**	0.96	**1.10**
64	15	1	0.80	**0.86**	0.47	**0.55**
	25		0.81	0.87	0.48	0.57
	35		0.81	0.89	0.48	0.58
	15	5	1.18	1.22	0.74	0.79
	25		1.17	**1.21**	0.74	**0.78**
	35		1.19	1.22	0.75	0.80
	15	10	1.45	**1.47**	0.94	0.97
	25		1.46	1.48	0.94	**0.96**
	35		1.45	1.48	0.95	1.00

As expected, the forecasting error was lower for networks with lower target lengths, both in terms of RMSE and MAE. History length did not consistently impact forecasting error. Networks with 15-minute and 25-minute histories often outperformed their 35-minute counterparts, particularly for current forecasting.

Table 4: Error of voltage forecasting LSTM networks

Network Size (units)	**History Len. (min)**	**Target Len. (min)**	**Base RMSE (V)**	**Accel. RMSE (V)**	**Base MAE (V)**	**Accel. MAE (V)**
32	15	1	1.24	1.56	0.43	1.02
	25		1.21	**1.52**	0.41	**0.95**
	35		1.34	1.68	0.68	1.23
	15	5	2.11	2.14	1.08	**1.09**
	25		2.11	**2.12**	1.07	1.25
	35		2.18	2.20	1.17	1.19
	15	10	2.79	2.76	1.78	1.82
	25		2.76	**2.74**	1.83	**1.74**
	35		2.87	2.91	1.86	1.89
64	15	1	1.22	1.26	0.41	0.54
	25		1.22	**1.22**	0.46	0.45
	35		1.23	1.26	0.42	**0.44**
	15	5	2.03	2.27	0.97	1.22
	25		1.96	**2.00**	1.17	**1.20**
	35		1.99	2.15	1.12	1.24
	15	10	2.76	2.82	1.74	1.70
	25		2.58	2.95	1.63	2.07
	35		2.35	**2.31**	1.40	**1.38**

The sizes of the precomputed tables used to estimate activation function values were also evaluated in terms of error. The impact of table size on networks with 15-minute histories and one-minute targets is demonstrated in Table 5 for current and Table 6 for voltage. These results are for the same networks whose performance was evaluated in Table 2, and once again represent the reality across other network sizes. The smallest tables, of length

Table 5: Impact of activation function table size on error for current forecasting

History Length (min)	**Target Length (min)**	**Network Size (units)**	**Activation Func. Table Size**	**RMSE (A)**	**MAE (A)**
15	1	32	8	4.15	3.29
			16	1.84	1.43
			24	1.31	0.99
			32	1.11	0.80
		64	8	1.55	1.18
			16	0.98	0.68
			24	0.89	0.59
			32	0.86	0.55

Table 6: Impact of activation function table size on error for voltage forecasting

History Length (min)	**Target Length (min)**	**Network Size (units)**	**Activation Func. Table Size**	**RMSE (V)**	**MAE (V)**
15	1	32	8	3.34	3.04
			16	1.66	1.13
			24	1.53	1.16
			32	1.56	1.02
		64	8	1.92	1.20
			16	1.41	0.74
			24	1.25	0.47
			32	1.26	0.54

eight, were consistently the least accurate in terms of MAE and RMSE for both current and voltage. Networks with 16-entry tables fared better but still faced reduced accuracy. Forecasting error rate was more competitive between 24-entry and 32-entry tables as they often generated very close results. The low error of 32-entry activation function tables was why they were used for the previous evaluations of network size and performance as they were accurate yet and provided a conservative evaluation of performance. Larger tables with 48 and 64 entries were also initially tested but not comprehensively as they were less accurate than 32-entry tables.

6.0 Discussion

As shown in the results above, LSTM networks for voltage and current forecasting on FPGAs outperform their software baselines. Networks are able to be deployed within the resource-constrained FPGA fabric of the Zynq-7045 indicating their viability on space platforms such as the SSP embedded computer. However, the limited resources of the Zynq-7045 did restrict network size. The parallelization of dot products was limited in 64-unit LSTMs by DSP utilization as shown in Figure 6. Reduced dot parallelization, however, was not detrimental to performance because, as shown in Figure 7, the impact on speedup within LSTM networks of this size was fairly minimal. In fact, dot products with moderate levels of parallelization performed better than their larger counterparts. The overhead of executing fully parallelized dot products was too great to be offset by parallel execution. These results demonstrate that partial parallelization of dot products is preferable in terms of performance.

In general, LSTM networks accelerated via the proposed methodology were able to achieve some degree of speedup while maintaining the same network structure as the floating-point software baselines. Achieving this performance improvement without modifying the network architecture is promising as it demonstrates this methodology's ability to rapidly convert networks prototyped in TensorFlow to accelerated networks. This work also gives an analysis of various network types and indicates what types of networks perform best using this acceleration scheme.

64-unit networks performed the best in terms of speedup, with networks consistently achieving 3.0× speedup over a software baseline. Speedup across 32-unit and 64-unit networks remained consistent regardless of history and target length as shown in Table 1. This result is likely due to the easily pipelined architecture of LSTM networks, which allows computation to continue even while longer data streams are received as inputs. 32-unit networks did result in faster execution times in comparison to 64-unit networks as expected due to their smaller size. However, 32-unit networks yielded more modest speedups of 1.6× as they performed worse than 64-unit networks when compared against their software baselines. The superior performance of 64-unit networks in terms of speedup was due to the networks larger size allowing the parallelism of FPGA acceleration to be exploited more effectively.

Across many of the tested networks, voltage-forecasting and, particularly, current-forecasting LSTMs were able to achieve good accuracy. Moreover, this research focused on the process of general LSTM network acceleration and was less concerned in creating the most accurate networks through state-of-the-art training optimizations. As such, error was primarily evaluated in comparison with a software baseline of the same network. Comparison allows for the impact of acceleration on forecasting error to be evaluated. Additionally, because the methodology of accelerating LSTMs is generalized, the same methodology is applicable for improving performance in other, more rigorously optimized, LSTM networks.

Accuracy was typically best for 64-unit networks, as shown in Table 3 and Table 4, which were able to keep both RMSE and MAE low for both current and voltage. The highest accuracy networks were able to achieve a RMSE below 1.5 A and 2.5 V and a MAE below 1 A and 1.5 V. This accuracy was achieved in a system where voltages ranged 10 V and current ranged 65 A. Current prediction networks were typically more accurate even with currents larger range. This is due to the fact that voltage was more difficult to predict, with current varying more gradually than voltage. Even so, both voltage and current prediction networks were successful in demonstrating the potential of LSTM for power system forecasting.

Additionally, the acceleration optimizations used for the FPGA implementation yielded minimum increases in error in comparison with a software baseline. In fact, the increase in short-term forecasting error for LSTM networks was always below a single volt or amp. In the highest accuracy networks, the error was below 0.2 V or 0.1 A. This demonstrates the low impact of network acceleration on forecasting accuracy.

Increasing speedup by reducing the number of entries in the tables used to compute activation function was possible, as seen in Table 2. However, by looking at the resulting error rate in Table 5 and Table 6, this optimization is considered unsatisfactory as it results in increased error rates.

In general, networks were created that maintained low error rates for short-term voltage and current forecasting while also achieving speedup through FPGA acceleration within the resource constraints of an embedded system. This research provides a broad overview of the exploration of different design and optimization methods as well as the accuracy and performance of networks with various sizes as well as history and target lengths. This comparison provides valuable information on the utility of these networks as well as insights that are applicable for future forecasting tasks that employ LSTMs and the acceleration methodology proposed in this study.

7.0 Conclusions

In this research, a methodology for accelerating LSTM networks for an embedded space platform is outlined. Specifically, the process for converting a prototype short-term voltage- or current-forecasting network to a network accelerated with an FPGA is explained. This technique demonstrated acceleration across all tested networks. Furthermore, 64-unit networks were shown to achieve the highest level of acceleration with over 3.0× speedup. These 64-unit networks also performed best in terms of reducing the RMSE and MAE for both short-term voltage and current forecasting. This showcases 64-unit networks abilities to achieve high accuracy while executing substantially faster then their software baseline. Additionally, a variety of history and target lengths were tested for current and voltage prediction networks. The impact of history and target length on error was varied, requiring testing to determine the most accurate network. However, speedup was consistent across history and target lengths, so regardless of which network had the highest accuracy, performance was consistently improved. These results were achieved within the resource-constrained FPGA fabric of the Zynq-7045 that is contained within the ZC706 embedded platform. This further showcases the potential of these networks to be applied in the environment of space, as the ZC706 serves as a facsimile of the SSP, a device conceived for space applications.

While this research focuses on applying optimized LSTMs to short-term forecasting for power data from the ISS, the process in generalized and can be applied to other LSTM networks created using TensorFlow. The LSTM architecture and floating-point operations were intentionally preserved so that accelerated networks were directly equivalent to their source networks. This supports the acceleration of other networks, which could be more rigorously optimized with state-of-the-art training methods. Additionally the process was designed to be device-agnostic. By using HLS tools and Xilinx's Vitis software development platform, the design is flexible and can be applied to other devices. Similar speedups and minimal impact on error rate can be expected regardless of LSTM network implementation details. In conclusion, this research's methodology facilitates the start-to-finish design and acceleration of LSTM networks for resource-constrained platforms.

8.0 Future Work

This research focused primarily on maintaining network structure by using floating point arithmetic. Future research into automatically quantizing network weights and values to support fixed-point arithmetic could yield substantial improvements in terms of speedup and resource utilization.

Bibliography

[1] M. Tan, S. Yuan, S. Li, Y. Su, H. Li, and F. He. Ultra-Short-Term Industrial Power Demand Forecasting Using LSTM Based Hybrid Ensemble Learning. *IEEE Transactions on Power Systems*, 35(4):2937–2948, 2020.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] S. K. Ibrahim, A. Ahmed, M. A. E. Zeidan, and I. E. Ziedan. Machine Learning Methods for Spacecraft Telemetry Mining. *IEEE Transactions on Aerospace and Electronic Systems*, 55(4):1816–1827, 2019.

[4] Z. Sun, Y. Zhu, Y. Zheng, H. Wu, Z. Cao, P. Xiong, J. Hou, T. Huang, and Z. Que. FPGA Acceleration of LSTM Based on Data for Test Flight. In *2018 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 1–6, 2018.

[5] S. Roffé, T. E. Schwarz, T. B. Cook, N. Perryman, J. Goodwill, E. Gretok, A. Phillips, M. Moran, T. Garrett, and A. George. CASPR: Autonomous Sensor Processing Experiment for STP-H7. In *Proceedings of the AIAA/USU Small Satellite Conference*, 2020.

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

[7] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott. FINN-L: Library Extensions and Design Trade-Off Analysis for Variable Precision LSTM Networks on FPGAs. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 89–897, 2018.

[8] P. Kumar Meher. An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks. In *2010 18th IEEE/IFIP International Conference on VLSI and System-on-Chip*, pages 91–95, 2010.

[9] S. Ngah and R. A. Bakar. Sigmoid function implementation using the unequal segmentation of differential lookup table and second order nonlinear function. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, 9(2-8):103–108, 2017.

[10] D. L. Marino, K. Amarasinghe, and M. Manic. Building energy load forecasting using deep neural networks. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 7046–7051, 2016.

[11] T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[12] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[13] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[14] Tietronix Software, Inc. ISSLive! Bringing the International Space Station to every generation. `http://isslive.com/`, 2015.

[15] E. W. Gholdston, K. Karimi, F. C. Lee, J. Rajagopalan, Y. Panov, and B. Manners. Stability of large dc power systems using switching converters, with application to the international space station. In *IECEC 96. Proceedings of the 31st Intersociety Energy Conversion Engineering Conference*, volume 1, pages 166–171 vol.1, 1996.

[16] C. Wilson, J. Stewart, P. Gauvin, J. MacKinnon, et al. CSP hybrid space computing for STP-H5/ISEM on ISS. *Proc. of the 29th Annu. AIAA/USU Conf. on Small Satellites*, 2015.

[17] A. D. George and C. M. Wilson. Onboard processing with hybrid and reconfigurable computing on small satellites. *Proceedings of the IEEE*, 106(3):458–470, 2018.

[18] S. Sabogal, P. Gauvin, B. Shea, D. Sabogal, A. Gillette, C. Wilson, et al. SSIVP: Spacecraft supercomputing experiment for STP-H6. *Proceedings of the 31st Annual AIAA/USU Conference on Small Satellites*, pages 1–12, 2017.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden,

M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[20] Francois Chollet et al. Keras, 2015.

[21] V. Kathail. Xilinx vitis unified software platform. In *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '20, page 173–174, New York, NY, USA, 2020. Association for Computing Machinery.