LSTM NEURAL NETWORK IMPLEMENTATION USING MEMRISTIVE CROSSBAR CIRCUITS AND ITS VARIOUS TOPOLOGIES

Kazybek Adam, B.S.

Submitted in fulfilment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

School of Engineering
Department Electrical and Computer Engineering
Nazarbayev University

53 Kabanbay Batyr Avenue,
Astana, Kazakhstan, 010000

Supervisors: Alex James

December 13, 2018

DECLARATION

I hereby, declare that this manuscript, entitled “LSTM Neural Network Implementation Using Memristive Crossbar Circuits and Its Various Topologies*”,* is the result of my own work except for quotations and citations which have been duly acknowledged. I also declare that, to the best of my knowledge and belief, it has not been previously or concurrently submitted, in whole or in part, for any other degree or diploma at Nazarbayev University or any other national or international institution.

Name: Kazybek Adam

Date: December 13th, 2018

Abstract

Neural Network (NN) algorithms have existed for long time now. However, they started to reemerge only after computers had been invented, because computational resources are required to implement NN algorithms. In fact, computers themselves are not fast enough to train and run the NNs. It can take days to train some complex neural networks for certain applications. One of the complex NNs that became widely used is Long-Short Term Memory (LSTM) NN algorithm.

As a broader approach to increase the computation speed and decrease power consumption of neural network algorithms, hardware realizations of the neural networks have emerged. Mainly FPGA and analog hardware are used for these purposes. On this occasion, it happens to be only FPGA implementations of LSTM exist. Using this lack, this thesis work mainly aims to show that LSTM neural network is realizable and functional in analog hardware. In fact, analog hardware using memristive crossbars can be a potential solution to the speed bottleneck experienced in software implementations of LSTM and other complex neural networks in general.

This work mainly focuses on implementation of already trained LSTM neural networks in analog circuitry. Since training consists of both forward and backward pass computations through NNs, first, there should be focus on implementing the

circuitry that can run forward passes. This forward running circuit further can be extended to a complete circuit which would include training circuitry.

Additionally, there exists various LSTM topologies. Software analysis has been done to compare the performance of each LSTM architecture for time-series prediction and time-series classification applications. Each of the architectures can be implemented in analog circuitry without great difficulty using voltage-based LSTM circuit parts due its easiness to reconfigure. Fully functional implementation of the voltage-based memristive LSTM in SPICE circuit simulator is the main contribution of this thesis work. In comparison, current-based LSTM circuit parts may not be easily rearranged due to the difficulty of passing currents from one stage to the next without degradation in magnitude.

Acknowledgements

I would like to thank my supervisor, Professor James for providing me an opportunity to work on such an interesting research topic and helping on my research. I also would like to thank the research group members for their help. Particularly, Kamilya Smagulova and Olga Krestinskaya helped me in exploring my research topic for this thesis.

Table of Contents

List of Publications

1. Smagulova K, Adam K, Krestinskaya O, et al (2018) Design of CMOS-memristor Circuits for LSTM architecture. In: 2018 IEEE International Conference on Electron Devices and Solid State Circuits (EDSSC), Electron Devices and Solid State Circuits (EDSSC), 2018 IEEE International Conference on, p. 1. doi: 10.1109/EDSSC.2018.8487179
2. Adam K, Smagulova K, James AP (2018) Memristive LSTM network hardware architecture for time-series predictive modeling problem. arXiv preprint arXiv:1809.03119, URL http://arxiv.org/abs/1809.03119.
3. Adam K, Smagulova K, Krestinskaya O, et al (2018) Wafer Quality Inspection using Memristive LSTM, ANN, DNN and HTM. arXiv preprint http://arxiv.org/abs/1809.10438, URL http://arxiv.org/abs/1809.10438.

Chapter 1 – Introduction

1.1 General

Since the invention of transistors in 1947 by William Shockley, John Bardeen, and Walter Brattain the world has experienced technological boom. First flip-flop consisting two bipolar transistors was built by Jack Kilby of Texas Instruments [1]. Later transistors became the integral part of any electronic device. During the course of technological advancements, there was mainly a single change: transition from bipolar junction transistors (BJTs) to Metal Oxide Semiconductor Field Effect Transistors (MOSFETs) in digital IC design. Since then CMOS process technologies have been scaled down steadily by abiding the “Moore’s Law”. However, this transistor shrinking trend cannot continue when semiconductor industry reaches the point where further reductions in size will be intolerable resulting in unreliable operation of CMOS devices. Therefore, there is a need for alternative solution to continue manufacturing smaller, faster, and power-efficient electronic devices. Particularly, scaling down of CMOS memory devices (Flash, SRAM, etc.) is of utmost concern for semiconductor industry. In this regard, a new physical element called memristor has been proposed by scientists as a potential solution to replace transistor-based memory cells. In fact, memristors have found wide range of applications in electronics area [2]. They are used to realize neural networks in hardware: implementation of both neuromorphic computing systems [3] inspired by human brain; and their training algorithms such as back propagation [4] and Spike-Timing-Dependent-Plasticity [5-7]. Another application is using memristors in analog circuits as tunable resistors that can change the operation modes of the circuits [8]. They can be used in digital circuits as well replacing the transistors in implementing the logic gates [2]. These wide range of applications comes due to the properties of memristors: their metal-insulator-metal (MIM) structure, small footprint on a chip, memory in the form of resistance, low-switching time, high endurance, and low switching energy [2].

Therefore, memristors are believed to have bright future and become the next fundamental building block in both analog and digital IC design replacing the transistors where possible; and memristive systems with in-memory computing replacing the von Neumann architecture.

1.2 Research aims and Methods

The purpose of this thesis is to design fully functional analog circuit of LSTM using memristive crossbars and test it on solving machine learning problems.

The overall design was done on pen and paper. Then each part of the design was built and tested using circuit simulator program such as LTspice. After verifying each part works well, all the parts of the whole design were put together and tested again. The circuit-level testing of the circuit was compared to the software implementation results. The software implementation of LSTM was accomplished using Python programming language and Keras library. However, to dissect the algorithm and get the intermediary results of the algorithm, it has been implemented in Matlab from scratch as well without using LSTM library of Matlab. In addition, “recurrent.py” file from Keras library was extended to be able to build and run different topologies of LSTM in simple way.

Chapter 2 – Literature Review

2.1 Background Theory

2.1.1 Memory Resistor – Memristor

Leon Chua was the first person to notice that there was a missing fundamental circuit element and to publish a work in 1971 about the memristor [9]. He put forward that a memristor would complement the following list of fundamental circuit elements: resistor, capacitor, and inductor. In fact, it can be best shown by studying the relationship of fundamental circuit elements with the fundamental circuit variables as shown in Figure 2.1. In the figure, the diagonal line contains the fundamental circuit variables of charge, current, voltage, and flux. Each row contains the equations where corresponding diagonal variable is expressed in terms of the other circuit variables and the circuit elements. Whereas, each column contains the equations where corresponding diagonal variable is part of the expressions. It is not difficult to see the missing relationship between charge q and flux φ. Mathematically, following relationships of charge-controlled memristance with voltage and flux-controlled memristance with current were established by Chua [9]:

$$v(t) = M(q(t))i(t), \tag{2.1}$$

$$M\big(q(t)\big) = \frac{d\varphi(q)}{dq}. \tag{2.2}$$

Likewise, the current can be expressed as

$$i(t) = W\big(\varphi(t)\big)v(t), \tag{2.3}$$

$$W\big(\varphi(t)\big) = dq(\varphi)/d\varphi. \tag{2.4}$$

The memristor's unit is memristance (short for "memory resistance") $M\big(q(t)\big)$ and has units of resistance. Similarly, flux-controlled memristor's unit is memductance $W\big(\varphi(t)\big)$ which has units of conductance. Therefore, memristor is a passive two-terminal fundamental circuit component exhibiting memory property by changing its resistance depending on how much charge in total went through the memristor.

Chua and Kang, in 1976, extended the theory of memristors to memristive devices [10]. Memristive devices differ from memristors in the way how the change in their resistance occurs. Basically, a memristive device has an internal state x which influences its resistance. The internal state changes depending on how much and for how long voltage signal applied across or current passed through it. The internal state is not directly related to flux or charge as in the case of memristors. Mathematically, the same equations for memristors can be expressed as following [10]:

$$v(t) = M(x, i)i(t), \tag{2.5}$$

$$i(t) = W(x, v)v(t). \tag{2.6}$$

where $M(x,i)$ is the memristance of time-invariant current-controlled memristive device; and $W(x,v)$ is memductance of time-invariant voltage-controlled memristive device. Also, $\frac{dx}{dt} = f(x,i)$ is valid for the former case and $\frac{dx}{dt} = f(x,v)$ for the latter case.

Both memristors and memristive devices demonstrate hysteresis in their I-V curve as shown in Figure 2.2. The shapes may differ from device to device, but the hysteresis always goes through the origin. Memristive devices is a broad group that includes memristors which have $f(x,i) = i$.

Charge q	$q = \int i\, dt$	$q = Cv$	$q = \frac{\varphi}{M}$
$i = \frac{dq}{dt}$	*Current i*	$i = \frac{\mathrm{v}}{R}$	$i = \frac{\varphi}{L}$
$v = \frac{q}{C}$	$v = Ri$	*Voltage v*	$v = \frac{d\varphi}{dt}$
$\varphi = M\mathrm{q}$	$\varphi = \mathrm{Li}$	$\varphi = \int v\, dt$	*Magnetic flux* φ

Figure 2.1. Symmetry of relationships between the fundamental circuit variables and elements. Also, bottom left corner contains the symbol of memristor.

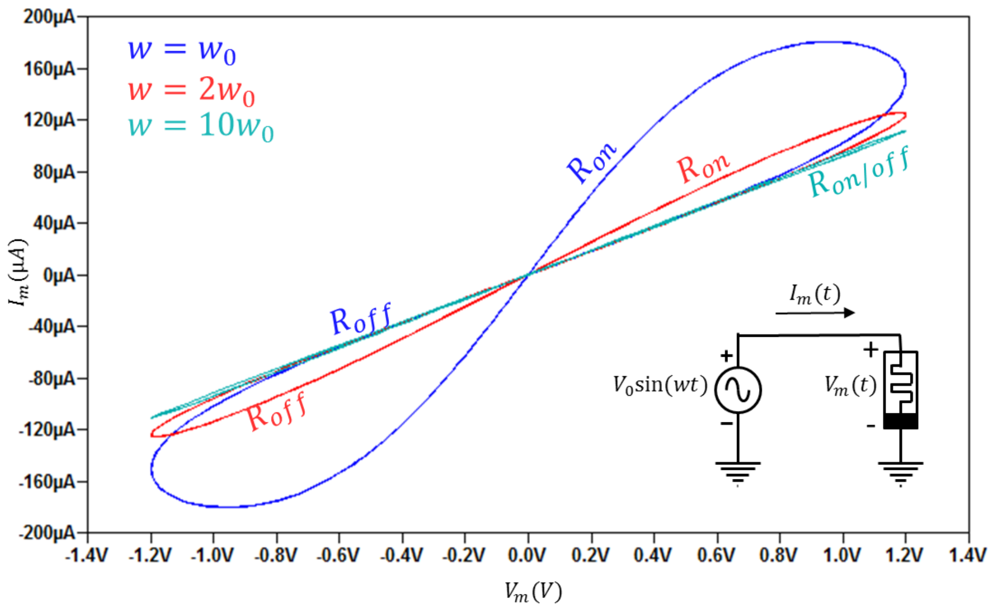

Figure 2.2. Example of I-V characteristic of a memristive device based on [11]. Sinusoidal voltage signal is applied across the memristive device for different frequencies.

Up until 2008, when Hewlett Packard proposed TiO_2 resistive switches as memristive devices [12], research on memristors and memristive devices was frozen. Hewlett Packard proposed the structure of the TiO_2 device as depicted in Figure 2.3 and the model of their TiO_2 device as following:

$$M(x,i) = \frac{R_{on}x(t)}{D} + R_{off}(1 - \frac{x(t)}{D}), \tag{2.7}$$

$$f(x,i) = \frac{\mu_V R_{on}}{D} i(t), \tag{2.8}$$

where R_{on} represents the lowest resistance which occurs at $x(t) = D$ and R_{off} is the highest resistance which occurs at $x(t) = 0$. D represents the total length of the device and μ_V is the dopant mobility.

However, the model proposed by Hewlett Packard does not fit well real devices. In fact, many other models have been developed that better represent real memristive devices [2], but still idealistic.

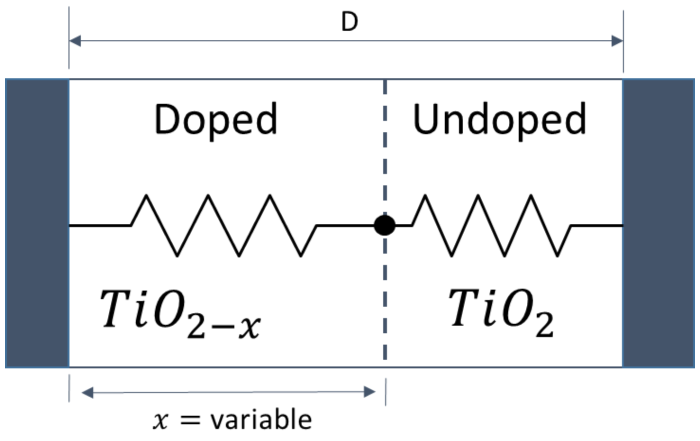

Figure 2.3. Memristor structure representation based on [12].

2.1.2 Memristive Crossbar Arrays

As already mentioned in the introduction, memristors can be used in many areas of electronics. Mostly they are used in crossbar configurations as shown in Figure 2.4 (and the idea of 2-dimensional resistive crossbar arrays was first proposed in 1961 by Steinbuch [13]). In the figure, particularly, implementation of neural network using memristive crossbar is illustrated. Its operation is simple. The inputs x_1, x_2, and x_3 are represented by voltage sources v_1, v_2, and v_3 in the circuit. The weights from w_{11} to w_{33} correspond to the memristor conducatnces from G_{11} to G_{33} in the same order. Finally, as expected, the outputs y_1, y_2, and y_3 correspond to currents i_1, i_2, and i_3. In practical implementations, the currents go to virtual grounds created using operational amplifiers. Note that here linear activation functions in the output layer was used for simplicity.

Basically, the crossbar executes vector-matrix multiplication (VMM) operation very fast and in parallel. Another advantage comes when the neural network size becomes large resulting in execution bottleneck (sequential execution of processing units) and data transfer bottleneck between processing unit and memory in computers while implementing the network training using software. In the case of the memristive crossbar there is no need to store weight values somewhere else – training of the weights happens in-place [14].

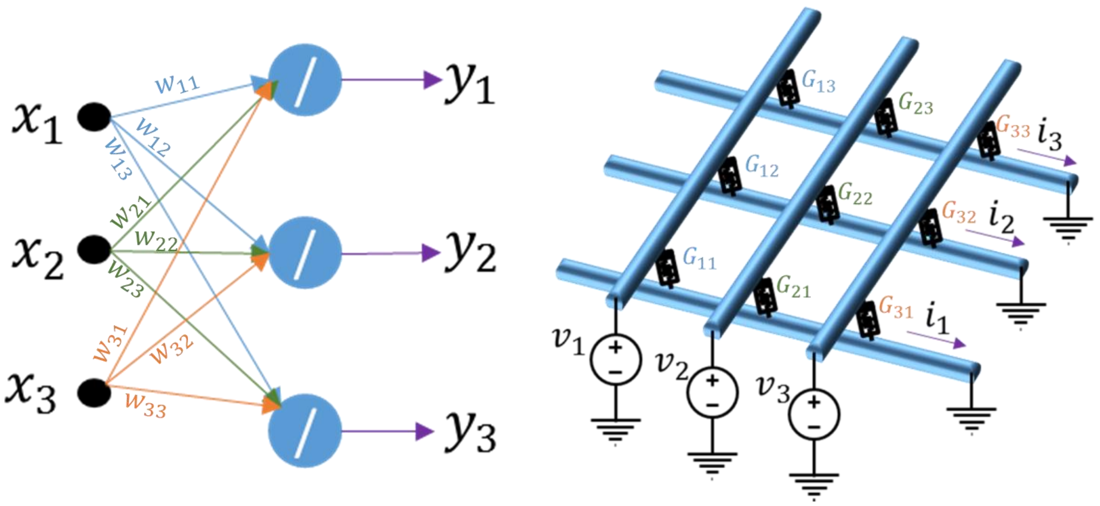

Figure 2.4. Simple neural network (left) and its crossbar implementation (right).

2.1.3 Long-Short Term Memory (LSTM)

LSTM, a type of Recurrent Neural Network (RNN), was invented by Hochreiter and Schmiduber in 1997 to solve the issue of vanishing and exploding gradients in RNNs [15]. The problem arises during the training of RNNs for patterns that are spread in long time steps. Diagram of RNN and its unrolled equivalent version with a vanishing gradient descent problem is visualized in Figure 2.5. In the figure, the RNN has a single input unit, a single recurrent hidden unit, and a single output unit. As neatly explained in [16], consider that the input at time t is non-zero and at the next time steps it is zero. Then, if we have recurrent weight value smaller than one, it means that the input at time t will contribute very little to the output at time t+3. In fact, as the difference of time steps between the input and the output increases the vanishing of the input's influence on the output happens exponentially fast. Therefore, the error derivative with respect to the input will vanish. Similarly, exploding gradient problem occurs when the recurrent edge has weight value higher than one. In addition, the type of activation function also contributes to the vanishing and exploding gradient problems. For instance, in the case of a sigmoid activation function, the vanishing gradient problem is more likely to happen since it will always output values smaller than one. Whereas, in the case of a rectified linear unit with its output equal to $\max(0, x)$, it tends to explode as its output value is not restricted to one.

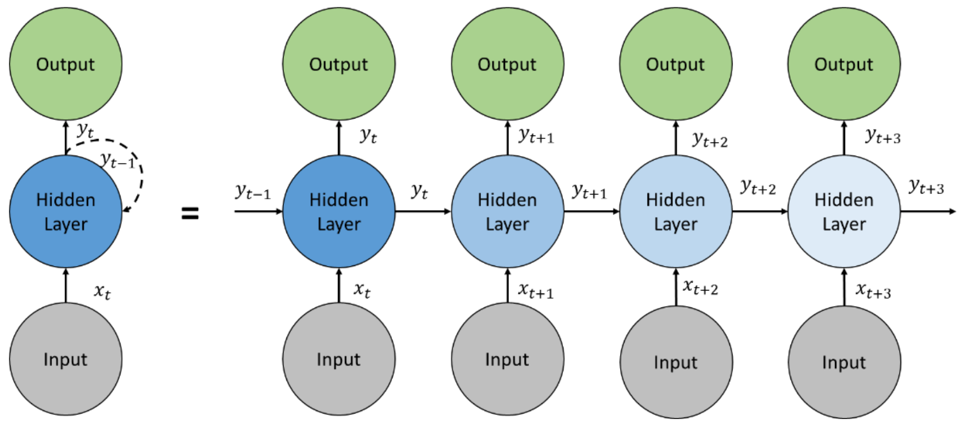

Figure 2.5. RNN representation on the left. Equivalent diagram on the right when RNN experiences a vanishing gradient problem.

So, there is no control over the information that is fed into and out of a simple RNN cell. Whereas, an LSTM RNN cell has control over what is forgotten, what is fed into, and what is outputted in a cell. This is achieved through gates: forget, input, and output gates which utilize sigmoid activation functions. Sigmoid function changes smoothly and ranges between zero and one. When a gate's output is one, it allows to pass all the current stage information to the next stage using hadamard multiplication operation. Similarly, when it is zero, the gate does not allow to pass any information to the next stage. From Figure 2.6, it can be seen that the structure of all three gates are the same: shared input vector which is the concatenation of network input value x_t at the current time step and the output value h_{t-1}, which is the output of the previous time-step LSTM cell; and the same activation functions and hadamard multiplication units. They differ only in different values of weights for the input vector ($[x_t, h_{t-1}, 1]$) as it can be seen below in mathematical form [17]:

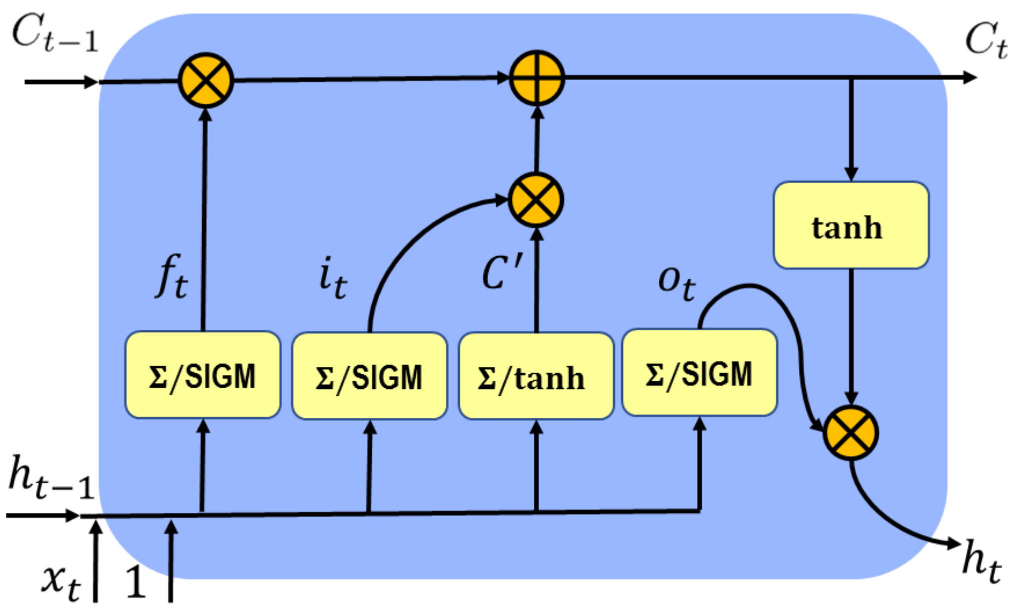

Figure 2.6. LSTM cell architecture based on [17].

$$f_t = \sigma(b^f 1 + w^f x_t + u^f h_{t-1}), \tag{2.9}$$

$$i_t = \sigma(b^i 1 + w^i x_t + u^i h_{t-1}), \tag{2.10}$$

$$o_t = \sigma(b^o 1 + w^o x_t + u^o h_{t-1}). \tag{2.11}$$

The cell state C_t of an LSTM cell is updated by forgetting some portion of the old cell state C_{t-1} and adding some portion of the candidate value C'_t, which has the same structure as the gates, except with different activation function:

$$C'_t = tanh(b^{C'} 1 + w^{C'} x_t + u^{C'} h_{t-1}), \tag{2.12}$$

$$C_t = f_t * C_{t-1} + i_t * C'_t \tag{2.13}$$

Finally, the current output of the LSTM cell is the some portion of the filtered cell state C_t:

$$h_t = \ o_t * \tanh(C_t). \tag{2.14}$$

Note that the equations (2.9)-(2.14) correspond to the case of a single LSTM hidden unit. In practice, LSTMs are used with larger weight matrices meaning that the size LSTM hidden layer is much larger. Also note that Keras library of Tensorflow uses the version of LSTM that was presented above [18]. Other architectures of LSTM will be presented in the methodology section of this thesis.

It is no wonder that LSTM became quite popular in recent times, since it is used widely in machine learning field. Particularly, application of LSTM can be found in natural language translation [19], image captioning [20-22], video captioning [23], speech recognition [24], and time-series prediction (part of this thesis).

2.2 Relevant Literature

As it has been already mentioned in the section 2.1.2, memristive crossbar arrays are more efficient in implementing neural networks than computers are with their software. Since memristors and memristive systems are still emerging as a technology, not many chip implementations of neural networks using memristive crossbar arrays exist. Instead, FPGA implementations of neural networks have become a general trend in literature works. Particularly, FPGA implementations of LSTM have been presented each year since 2015 by research community [25-29]. These works report efficiency of FPGA implementations of LSTM compared to the software implementations of LSTM. However, there are still a few works that are related to the implementation of LSTM in memristive crossbar arrays [30-32] and ultimately, it is believed that memristive crossbar implementations would outperform the digital implementations as memristor technology matures [31].

In [30], purely analog implementation of memristive LSTM in 0.18 μm CMOS technology is proposed. However, this work does not provide full circuit simulation of the whole system solving a particular machine learning problem. They propose only the separate analog building blocks of the whole system. Particularly, analog circuits for activation function and element-wise (hadamard) multiplication operation are proposed. In addition, an existing crossbar configuration is presented.

As opposed to the work in [30], authors of [31] solve a real-world problem (language modeling problem) and accomplish a system level simulation. Their simulations are done in their own built software tool (written in C++) rather than on circuit simulator such as SPICE. They use so-called non-linear function units (NLFs), which are digital blocks, to implement all the mathematical operations required to implement LSTM except for vector-matrix multiplication (since it is implemented using memristive crossbars). They set some constraints in the software to incorporate the non-idealities associated with the analog VMM operation. Their main finding is that memristors need to have symmetric change in conductance values when given positive and negative voltages across for good performance results. The slightest variation as low as 2% in the asymmetry can severely affect the performance results in large LSTM networks with the sizes of up to 512 hidden units compared to fully connected networks with smaller size and smaller training dataset.

Finally, the work by [32] presents a fabricated chip consisting of one-transistor one-memristor (1T1R) type crossbar array that implements the VMM operation part of the LSTM algorithm. However, the rest of the operations were implemented in software in their work. They were able to train their weight matrix of memristors *in-situ* and solve successfully time-series prediction and classification problems. The used memristors were from Ta/HfO_2.

Chapter 3 – Methodology

In this thesis, the problems that are solved using LSTM algorithm are framed as following: predict sample point at current time step t using previous sample points at time steps $t-1, \dots, t-d$. Selection parameter d is a look-back number or a prediction delay. Then, output of a single LSTM cell (neuron) can be expressed as following:

$$y(t) = f_{LSTM}(x\,(t-1), \dots,\; x\,(t-d)). \qquad 3.1$$

3.1 Selected Machine Learning Problems

3.1.1 Time-series prediction problem 1

As a first step in implementing LSTM neural network algorithm in analog hardware (in circuit simulator) using memristive crossbar arrays, a simple machine learning problem - prediction of international airline passenger count - was selected. In fact, a web-tutorial by Brownlee [33] was already solving the problem using LSTM in Keras library (which became the main tool for software simulations of different LSTM architectures in this thesis). This simple problem does not require large LSTM neural network to achieve satisfactory prediction results. Having smaller LSTM network means smaller weight matrices are used in the network resulting in smaller memristive crossbar arrays being used in the analog implementation of the algorithm. In addition, the dataset [34] of the problem is also small enough to run all the testing data in a circuit simulator. It contains 144 sample points which are monthly number of international airline passengers in thousands from 1949 to 1960. The plot of the dataset is shown in Figure 3.1 below. This dataset is used both in system-level and circuit level simulations involving LSTM algorithm.

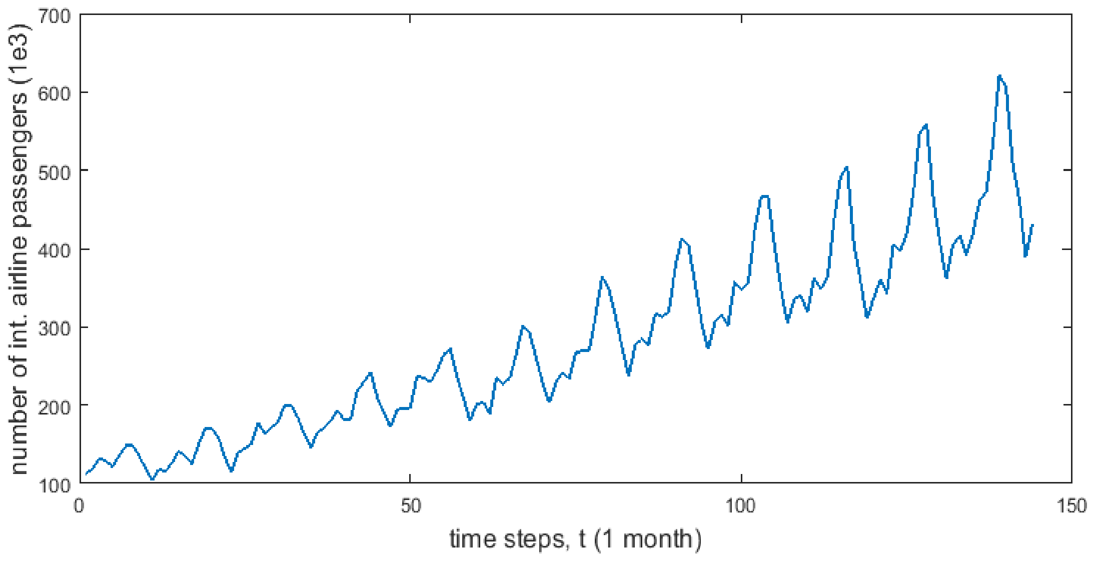

Figure 3.1: Plot of data samples of time-series prediction problem 1. Monthly data of international airline passenger count from 1949 to 1960 [34].

3.1.2 Time-series prediction problem 2

Another time-series prediction problem that was picked is the prediction of CO^2 emission volumes at volcano Mauna Loa. The dataset [35] of the problem contains 192 sample points which are monthly CO^2 emission levels starting from 1965 to 1980. The plot of the dataset is shown in Figure 3.2 above. However, the dataset of this problem was only used in system-level simulations for analysis of different LSTM architectures. It was chosen, because its plot has a little different shape and pattern than that of the problem 1 dataset.

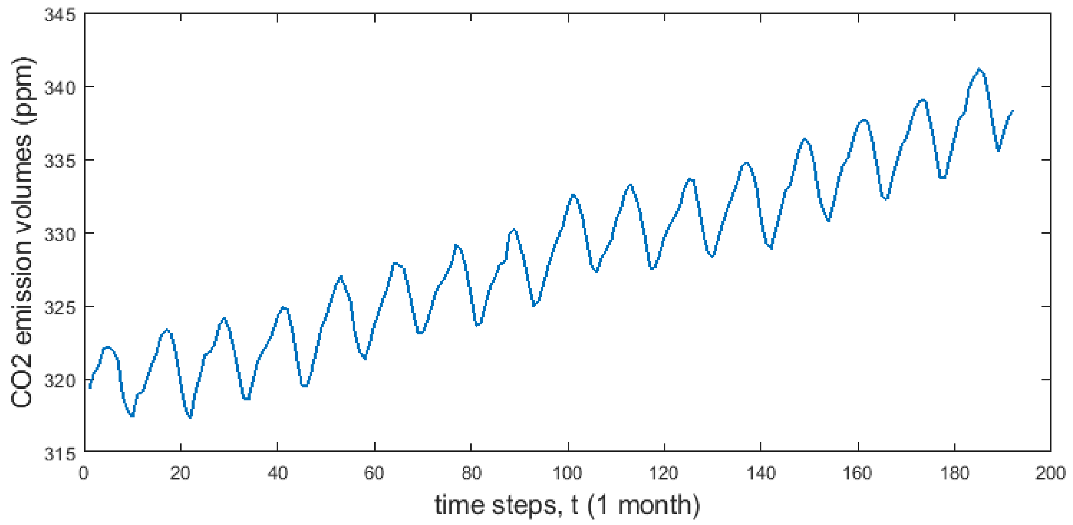

Figure 3.2: Plot of data samples of time-series prediction problem 2. Monthly data of CO*^{2} *emission volumes at volcano Mauna Loa from 1965 to 1980 [35].

3.1.3 Time-series prediction (classification) problem 3

The last problem is a wafer quality classification problem. It is a binary classification problem which is modeled as a time-series prediction problem where the predicted value represents a class rather than a more meaningful quantity as in the previous two problems. A wafer in this problem can be classified as either normal or abnormal. Therefore, here we are not highly concerned about the exact value of the prediction. As long as the predicted value is in one range for one class and in the other range for the other class, we can successfully classify a wafer. This problem contains large database consisting of 7164 datasets [36] each of them having 152 data samples and a single associated class label. These datasets are used both in system-level and circuit level simulations involving LSTM algorithm. Figure 3.3 shows plot of four different testing datasets from the wafer classification database.

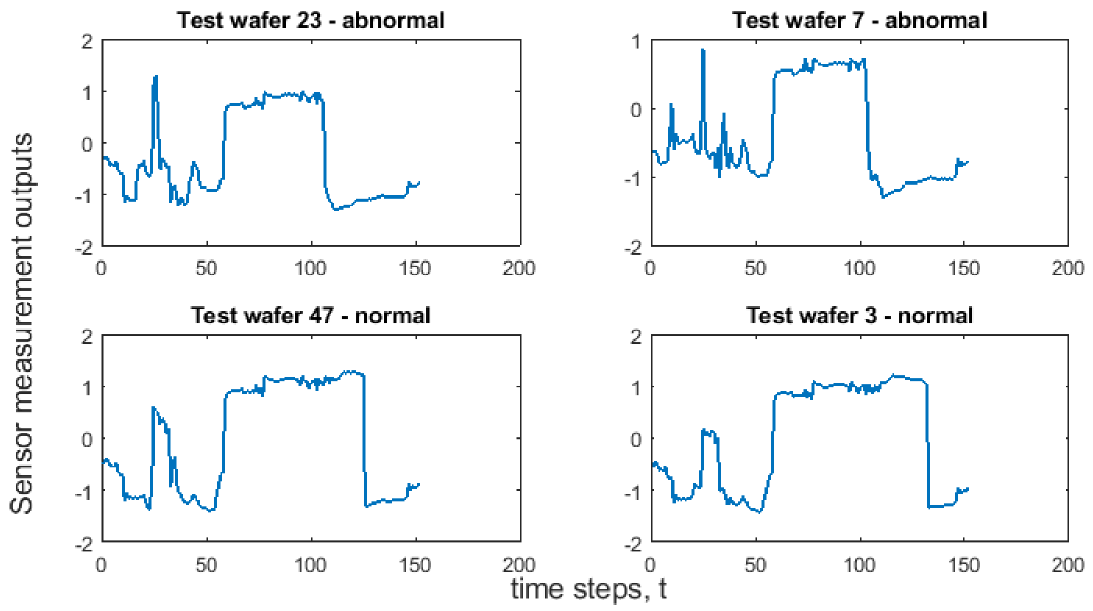

Figure 3.3: Plot of data samples of time-series classification problem for test wafers 23, 7, 47, and 3. A single inline sensor measures 152 times to obtain the samples [36].

3.1.4 Selected Models

Since the tutorial in [33] was already solving the problem, the model selection process was minimal for the first problem. To solve the problem a two layer neural network consisting of LSTM and Dense (fully-connected) layers was used as in the tutorial. The LSTM layer in the network contains four (a not too small and not too large value) hidden units and the dense layer contains a single unit which squashes the four outputs of the LSTM layer into a single output – prediction value. Another parameter is a look-back number or the number of recurrent operations before obtaining the last time step output and it was chosen to be equal to two, unlike the values in the tutorial. It was chosen empirically: for the same number of hidden units, it gave a better result for a look-back value of two than for values of one and three. One might question that if LSTM is good at learning long patterns of data through time then why it did not do well for a look-back value of three. It may be due to the lack of extra features or there is a bad pattern in the data. As for the training of the network, it was trained using mean square error loss function and adam optimizer [37] with default parameters [38]. However, instead of 100 epochs as in the tutorial, an epoch size of 500 was chosen while keeping a batch size of one. It was done to obtain approximately same performance results as in the former case while having weight constraints between ± 1 to keep input voltages into a memristive crossbar array small enough for a given range of memristances, [R_{on}, R_{off}]. Then trained weights are extracted for setting memristance values in the memristive crossbar arrays. The training to testing data ratio used is 2/1.

The same model was used for solving problem 2 since it is similar to the first problem. However, in system-level simulations for analysis of different LSTM architectures, the epoch size of 300 was used for both problems. This is a middle-ground value which gives enough learning opportunity to detect the performance of each LSTM architecture. It helps to accelerate system-level Monte Carlo simulations. The third problem, time-series classification problem, also used the same model as the first two except that a look-back value of 152, an epoch size of 40 with batch size of 1, training to testing ratio of roughly 1/7, and no weight constraints were used. Here a look-back value of 152 is a fixed value unlike in the first two problems. Epoch size was reduced, because this problem has large training data (6164 datasets). For the system-level analysis, however, the epoch size was reduced to 25 and a batch size of 15 was used for speeding up the Monte Carlo simulations. Weight constraints were lifted, otherwise classification accuracy reduces significantly. Interestingly, without the constraints only a few weights went up beyond the weight range of [-1, 1] and by small amounts. These weights were represented by two parallel memristors in the circuit implementation of the problem to keep memristances in the allowed range of [R_{on}, R_{off}].

The summary of selected models for each problem is shown in Table 3.1 below. 144 total sample points were converted to 142 datasets with each having 2 samples and a single target value. Similarly, 192 total sample points were converted to 190 datasets with each having 2 samples and a single target value. Data samples were scaled down linearly to be in the ranges of [0, 1] for the first two problems and [-0.5, 0.5] for the last problem. These are the ranges where LSTM performs well for the selected problems.

Table 3.1: Summary of the selected models for each problem.

	Problem 1	***Problem 2***	***Problem 3***
Network Config. [L1(units)+L2(units)]	*LSTM(4)+ Dense(1)*	*LSTM(4)+ Dense(1)*	*LSTM(4)+ Dense(1)*
Train/Test data ratio	*2/1*	*2/1*	*1/7*
Look-back number	*2*	*2*	*152*
Dataset size (features, samples)	*(1, 3)*	*(1, 3)*	*(1, 153)*
Total # of Datasets	*142*	*190*	*7164*
Epoch size (weight extraction/analysis)	*500/300*	*NA/300*	*40/25*
Batch size (weight extraction/analysis)	*1/1*	*NA/1*	*1/15*
Weight Constraints	*[-1, 1]*	*[-1, 1]*	*None*
Input range	*[0, 1]*	*[0, 1]*	*[-0.5, 0.5]*
Loss function	*MSE*	*MSE*	*MSE*
Optimizer	*adam*	*adam*	*adam*

3.2 Current-based LSTM

In this thesis, the first attempt to implement LSTM algorithm in analog hardware using memristive crossbar array resulted in a LSTM circuit design based on current-based activation function circuits [39]. These circuits implement sigmoid and hyperbolic tangent functions using currents as inputs. It forces to use other types of circuits in the overall design such as current-mirror [39] and voltage-to-current converter; and therefore the overall design can be called current-based LSTM.

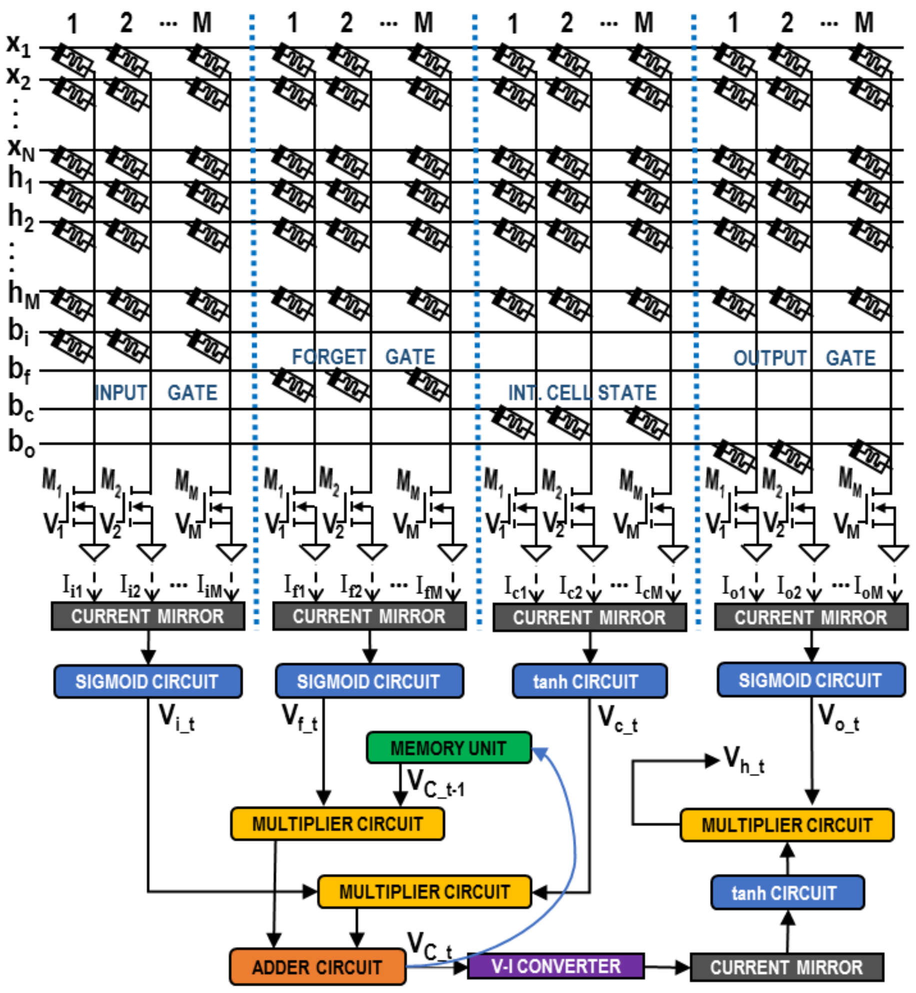

Figure 3.4: Circuit diagram of memristive current-based LSTM. Activation function, current mirror, and multiplier circuits were taken from [39]. Memristor R_{on}/R_{off} values are based on [40]. Circuit implementation was done using TSMC 0.18 um CMOS technology.

Figure 3.4 shows memristive crossbar implementation of a single LSTM layer with N inputs and M LSTM blocks (neurons). The features x_1 to x_N are input samples at time step t; and hidden unit values h_1 to h_M are the LSTM layer outputs of previous time step t-1. The biases for input, forget, and output gates are b_i, b_f, and b_o, respectively. The bias b_c is for the intermediary cell state c_t (also known as candidate cell state). The real cell state is C_t. In the figure, the four structures delimited by dashed blue lines compute the outputs of the gates: i_t, f_t, and o_t; and the intermediary cell state c_t. The transistors in those structures serve as switches that allow the execution of weighted summation once at a time per block. The resulting currents going to current mirrors are the summed values. The mirrored currents are then fed to corresponding activation function circuits: sigmoid and hyperbolic tangent circuits. The activation function circuits then output voltage values for i_t, f_t, c_t, and o_t. At this point, we have computed everything to obtain the cell state C_t except for C_{t-1}, cell state from previous time step. C_{t-1} is fetched from a memory unit and the computed C_t is stored in the same memory unit. Then voltage of C_t is converted to current before filtering it through hyperbolic tangent function. Finally, the current time-step output h_t of a current LSTM block is obtained as a voltage after multiplying the filtered C_t voltage and the voltage of o_t. However, M cycles of execution are required to obtain all hidden unit outputs h_t. This sequential operation of the circuit can save on-chip area in the expense of execution speed.

3.3 Voltage-based LSTM

In this section, voltage-based LSTM circuit design is proposed. Unlike current-based activation function circuits and current mirrors, voltage-based activation function circuits and voltage buffers provide us with higher accuracy and more predictable outputs. Current-based implementation could be used in solving a classification problem. This is because we are not interested in the analog output voltage – as long as it is high enough or low enough, we know that it is either digital 1 or digital 0. Whereas, in the case of time series prediction problem, we are interested in the analog output voltage to be much accurate rather than it being higher or lower of some threshold value. Therefore, high-accuracy sigmoid and hyperbolic tangent function circuits, which are voltage-based, were implemented. In addition, high accuracy four-quadrant multiplier circuit was adapted from [41]. They help to obtain accurate values at each stage to finally arrive to an accurate output value. Additionally, control circuit has been implemented to carry out the multiple time step feature of the LSTM RNN.

It is a fact that time series prediction will yield some error, for instance mean square error (MSE) or root mean square error (RMSE). If the RMSE of the circuit and the software implementations are close enough, then we can conclude that we successfully implemented the LSTM neural network in analog hardware.

The voltage-based memristive LSTM circuit diagram for solving the first time-series prediction problem is shown in Figure 3.5 below. In the figure, the ultimate sign of h_{t1} is consistent with its real sign in the LSTM algorithm equations. It is achieved by using activation function circuits that output inverted values.

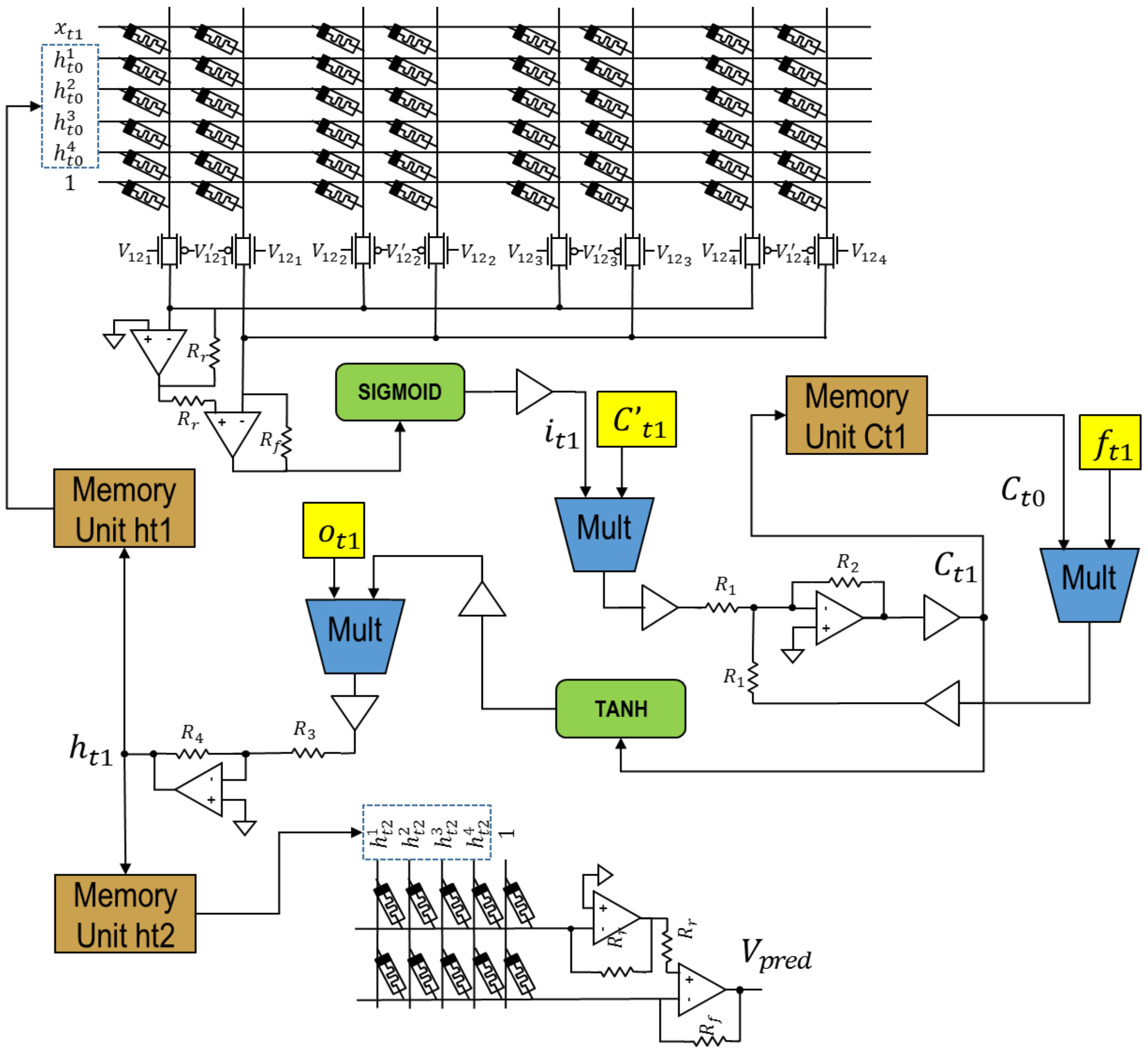

Figure 3.5: Full neural network circuit design consisting of LSTM and Dense layers which solves the time-series prediction problem 1. The network uses two previous time-steps (x_{t1}, x_{t2}) to predict x_{t3} which corresponds to V_{pred}. It means that there are two full cycles of operations inside the LSTM layer before its output is captured at memory unit ht2 for subsequent VMM operation in the dense layer. LSTM layer has 4 hidden units and dense layer has a single unit (no activation function). The design corresponds to the latest optimized version with RNIC-1 three-stage op-amps [42] and the multipliers based on symmetric complementary structure squarer circuits [43]. Ratios of $\frac{R_2}{R_1}$ and $\frac{R_4}{R_3}$ are equal to 10. The input voltage range of the network is [-0.1, 0.1]. The values in the yellow boxes are obtained in the same way as i_{t1}.

Similar circuit design is also used to solve the time-series classification problem. Their differences are only in memory unit circuits and control voltages. Memory unit circuits and control voltages for the first design are shown in Figure 3.6 and Figure 3.7, respectively. The memory units are used to store the previous time-step cell state values and the previous time-step outputs of LSTM hidden units. Since four hidden units are used in the network configuration, there are four sample and hold circuits in each memory unit. As for the control voltages, they ensure that each LSTM cell output is obtained in sequential manner, the same way as in current-based LSTM design. It takes 40 us to obtain all four LSTM hidden unit outputs at a current time step and total of 88 us to predict the target value of a single dataset.

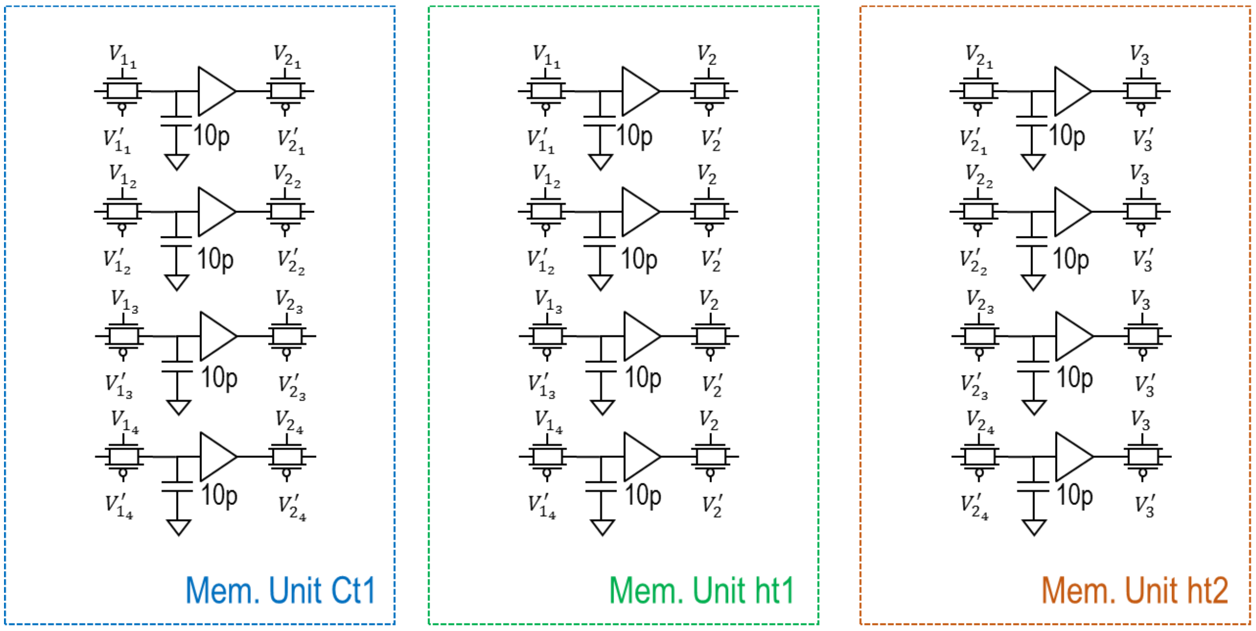

Figure 3.6: Memory units used in voltage-based LSTM (problem 1) consisting of sample and hold; and pass-transistor logic circuits. For the final design W/L ratio of 45um/0.18um is used for both NMOS and PMOS transistors in the pass logic switches.

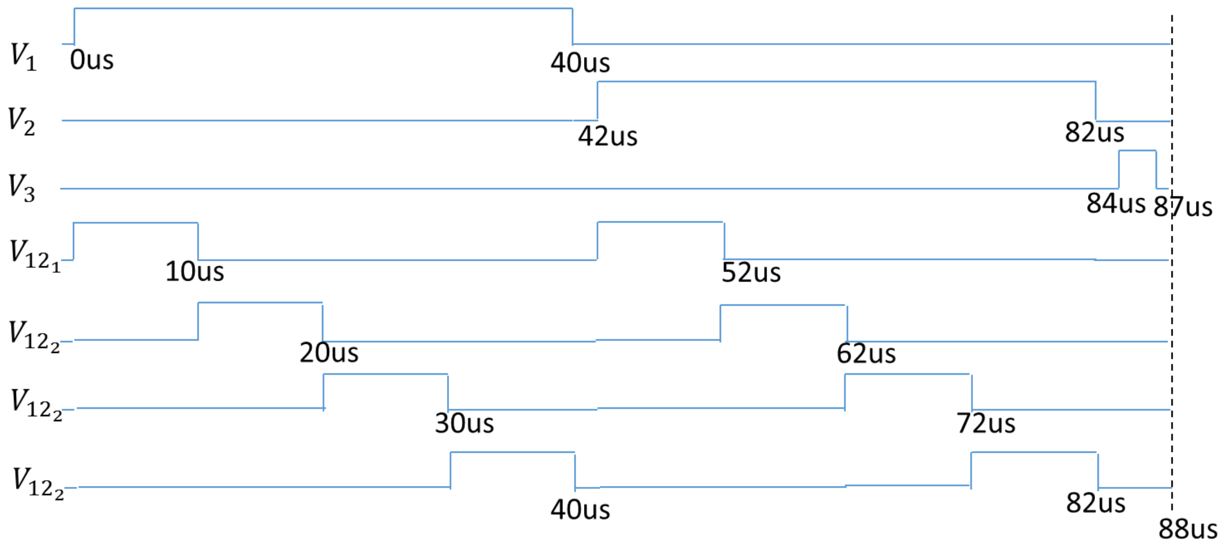

Figure 3.7: Control Voltage signals for voltage-based LSTM solving problem 1. Obtaining a single prediction value x_{t3} takes 88us. Amplitude of the pulses is 1.8V. The complements of the control voltages have the same pattern, but amplitude of -1.8V. Voltages V_{1_1} to V_{1_4}; and V_{2_1} to V_{2_4} are not shown in this figure, but they are first and second halves of signals $V_{12_1}$to V_{12_4}, respectively.

Corresponding memory units and control voltage signals are shown in Figure 3.8 and 3.9, respectively for the voltage-based LSTM design solving the time-series prediction problem 3. In Figure 3.9, control voltages V_1 to V_4 correspond to voltages V_{12_1} to V_{12_4} from the voltage-based LSTM solving problem 1 and they are periodic until 6.384 ms. Voltages V_h and V_{h1234} are periodic starting from 40us and ending at 6.384 ms. Unlike the memory units of the first design, the memory units of the second design use more sample and hold circuits and more capacitance values. These

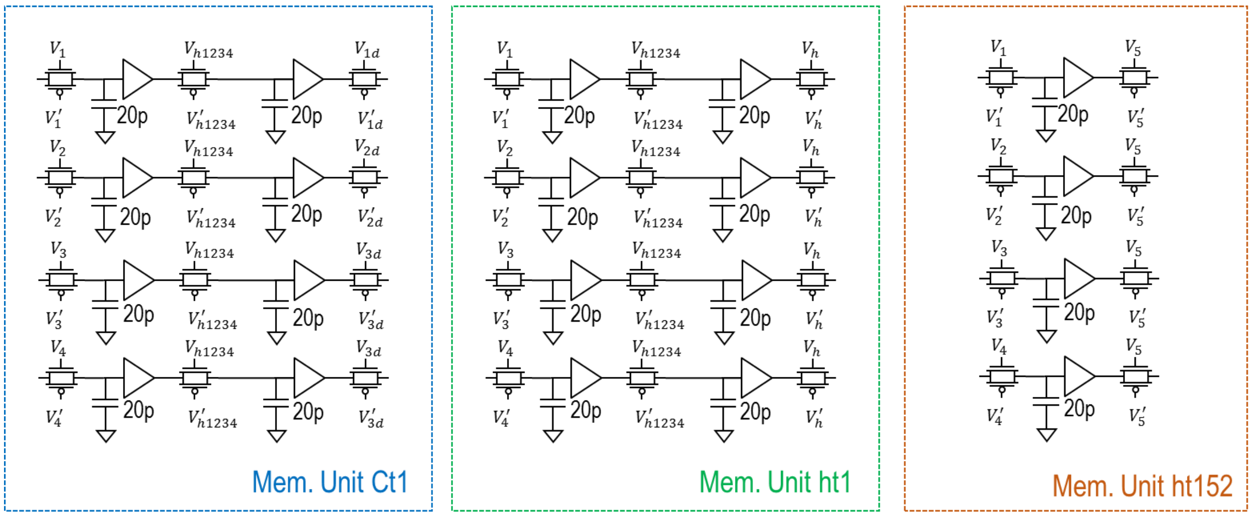

Figure 3.8: Memory units used in voltage-based LSTM (problem 3). Pass-transistor logics have W/L ratio of 18um/0.18um. . The input voltage range of the network is [-0.5, 0.5].

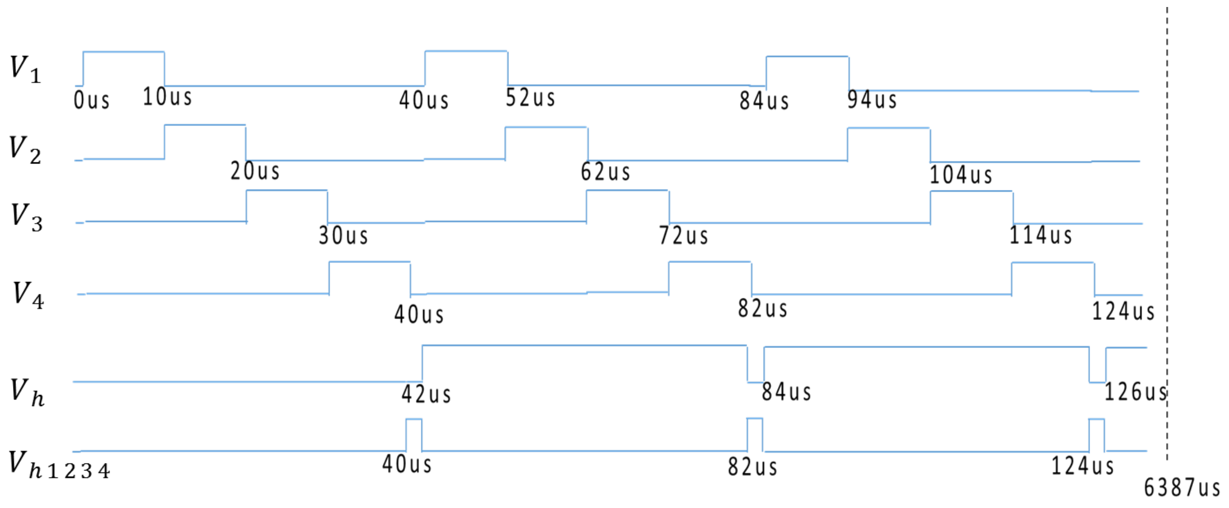

Figure 3.9: Control Voltage signals for voltage-based LSTM solving problem 3. Performing a single classification takes 6.387 ms. Voltages V_{1d} to V_{4d} are not shown in the figure, but they are the delayed versions of V_1 to V_4 with a delay of 40 us.

changes are due to the usage of a look-back number greater than two and the usage of two-stage op-amps [39], respectively. Initially, the first design also used two-stage op-amps, but later they were replaced with three-stage op-amps for greater prediction accuracy. Since we are classifying in the second design, we are not greatly concerned in the accuracy of the predicted analog values. Therefore, the second design used the older circuit parts: the two-stage op-amps and multiplier circuits based on Flipped Voltage Follower cells [41]. The input voltage range of the second design is [-1, 1].

3.4 Circuit Parts

3.4.1 Vector-Matrix-Multiplication (VMM) Circuit

The power of memristive crossbar circuits is the implementation of VMM operation in an efficient way. Op-amps are handy as always and provide virtual grounds for accumulating the currents in a crossbar column. In addition, they convert the accumulated current to voltage at its output. However, using single op-amp per column and single memristor per synapse restricts the range of implementable weights. In fact, that way we can only implement positive weights. The problem can be solved using two memristors per synapse and two op-amps per column in memristive crossbar [44]. The configuration of two op-amps in a crossbar is shown in Figure 3.10. The first column op-amp acts as inverting amplifier and the second column amplifier acts as summing amplifier. This configuration enables to subtract the current in the second column from the first one. The subtracted current is then multiplied to Rf and results in an accurate voltage at the output of the second op-amp. The voltage-based designs 1 and 2 use only single pair of op-amps per single sub-crossbar (total four of them). This is due to the use of sequential mode operation in the design.

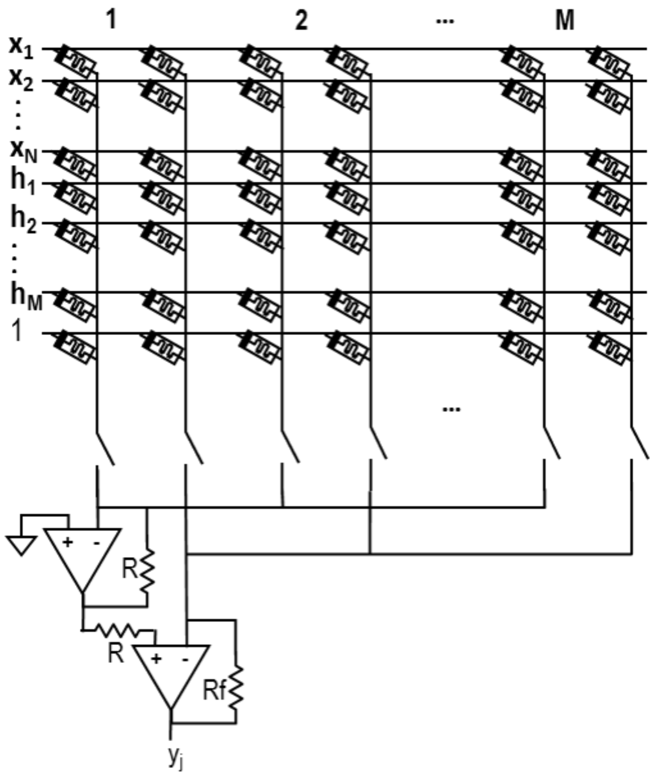

Figure 3.10: Memristive crossbar circuit with two memristors representing single synapse. Switches represent pass-transistor logic units. They are used for implementing VMM operation in sequential manner. The whole circuit of the voltage-based LSTM designs would have four of these crossbars. Output voltage is read at node y_j. R=1.25kΩ and Rf = 1kΩ/1.24kΩ for continuous/discrete memristance states.

The inference of problem 1 in the voltage-based LSTM circuit was simulated using both continuous and discrete memristance values. In the former case Ron and

Roff of memristors were chosen to be 10kΩ and 10MΩ, respectively based on the memristor device from [45]. In the latter case discrete memristance values between 1.1kΩ and 10kΩ were used [46]. The reason of using discrete weights is that in real memristive crossbar arrays, we cannot obtain an exact memristance states, rather we get some noisy states with Gaussian distribution [46]. According to the empirical data from [46], 68 memristance states were obtained between 1.1kΩ and 10kΩ range. The inference of problem 3 using the discrete weights is the part of future work.

In the previous section, it was mentioned that in the circuit designs of LSTM there were used two types of op-amps. The two-stage op-amp used in the designs is shown in Figure 3.11 and the three-stage op-amp is shown in Figure 3.12.

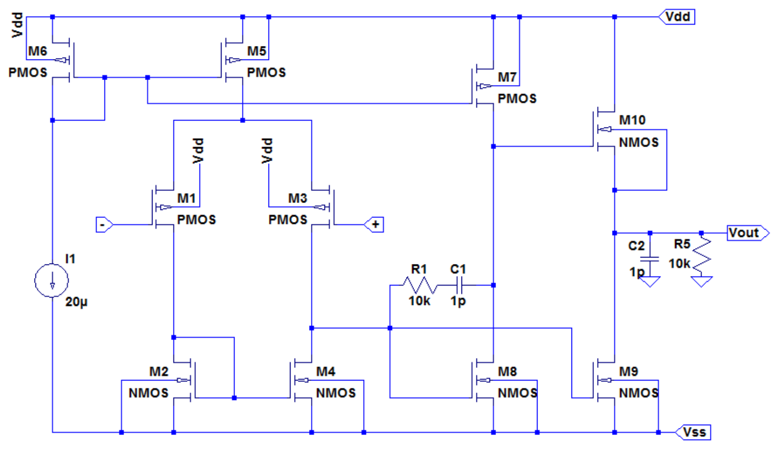

Figure 3.11: 2-stage op-amp used in the current-based and voltage-based LSTM designs [39].

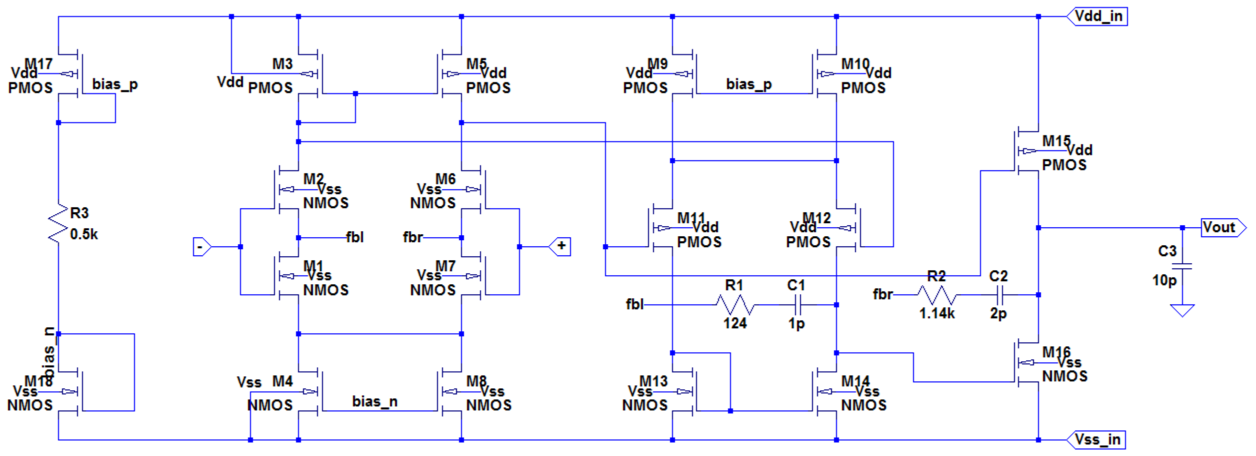

Figure 3.12: Three-stage op-amp – RNIC-1 op-amp based on [42].

3.4.2 Activation Function Circuit

Sigmoid and hyperbolic tangent functions can be obtained using circuit in Figure 3.13. It basically employs the property of differential amplifier – gradual and smooth increase of the output voltage when the differential input is swept between a desired range. The desired output range and form can be obtained by varying supply voltage V_{dd}, current I_1, and the sizes of NMOS transistors ($N1$ and $N2$). Voltage source values of V_1, V_2, and V_3 are used to shift the output values to match the graphs of the sigmoid and hyperbolic tangent functions. Since these two functions are different, the above mentioned parameters also change for each function. DC transfer characteristics for sigmoid and hyperbolic tangent function circuits are shown in Figure 3.14 and Figure 3.15, respectively. The graphs' input and output ranges are scaled down by -10 (negative part is canceled at later stages) to meet the operation range of the other circuit elements.

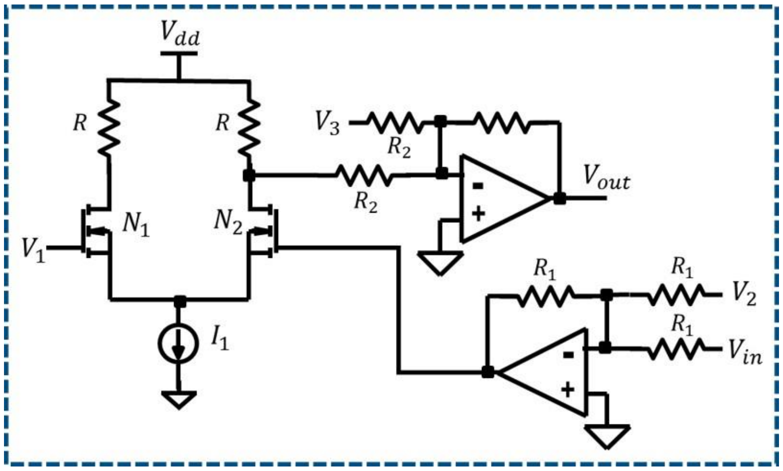

Figure 3.13: Proposed activation function circuit.

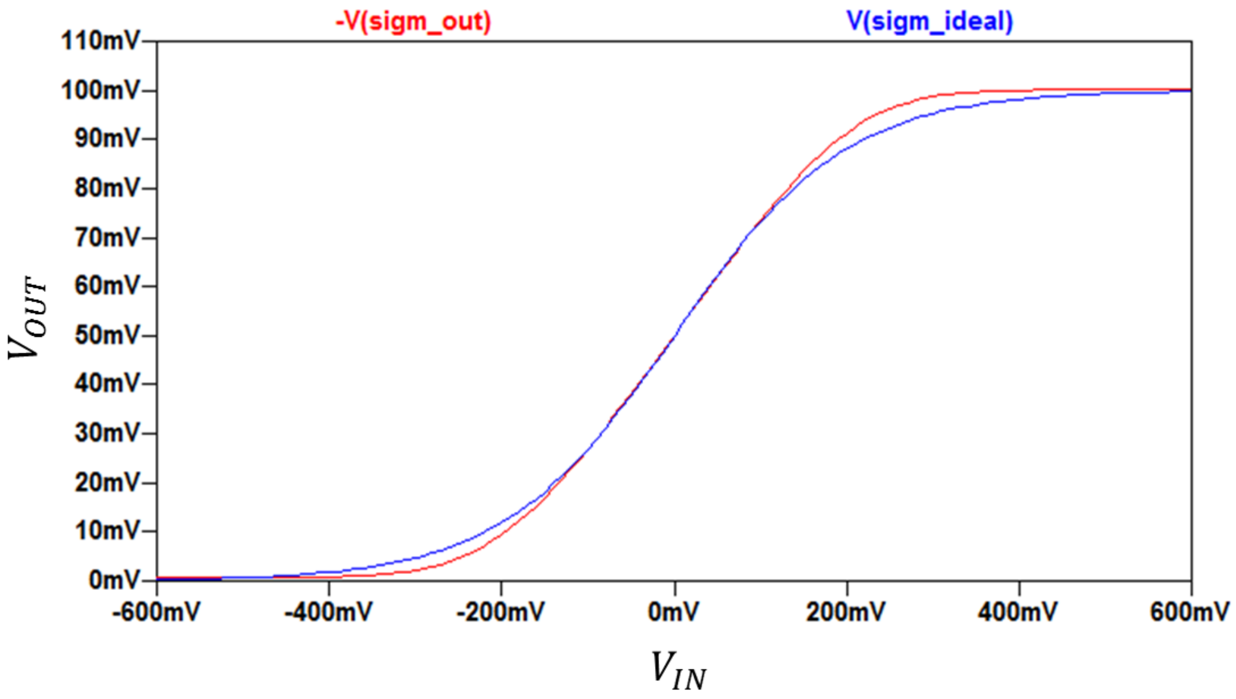

Figure 3.14: Comparison of sigmoid function implementations: red – circuit implementation, blue – software implementation. Inputs and outputs are scaled down by 10.

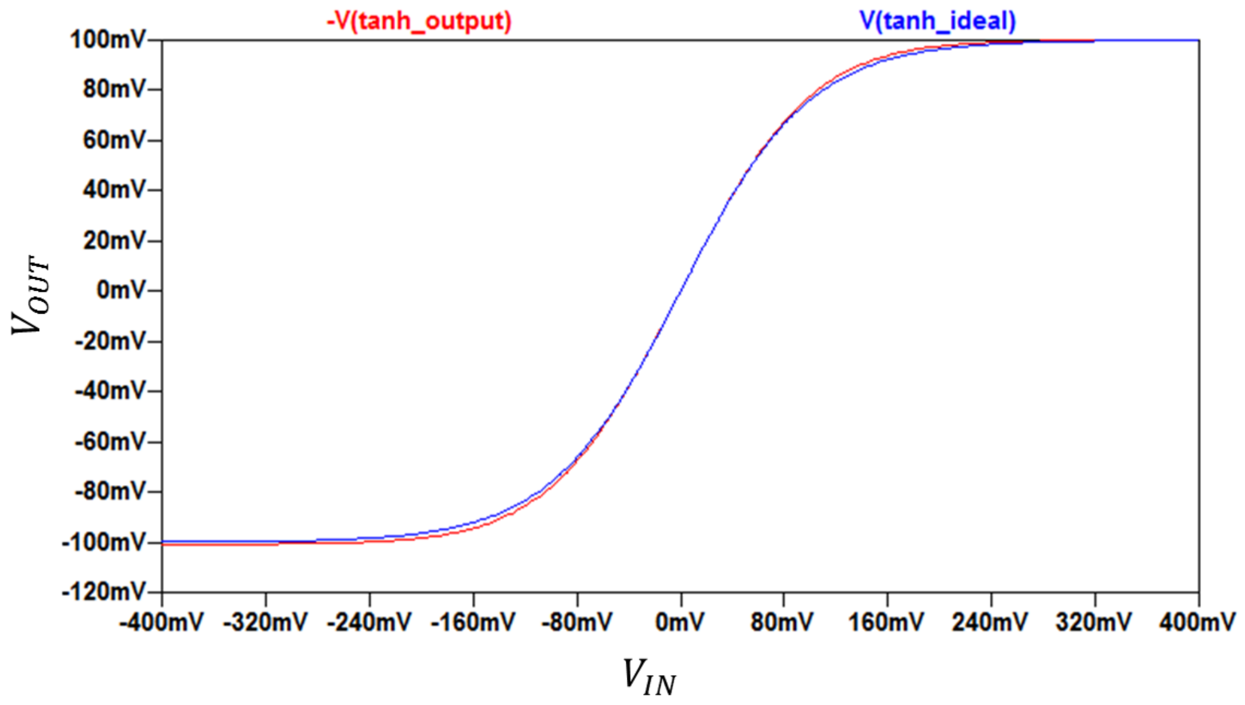

Figure 3.15: Comparison of hyperbolic tangent function implementations: red – circuit implementation, blue – software implementation. Inputs and outputs are scaled down by 10.

3.4.3 Analog Multiplier Circuits

Four-quadrant analog multiplier based on flipped voltage followers (FVFs) [41] was used in the implementation of hadamard multiplication operation. The circuit schematic and DC transfer characteristics of the FVF-based multiplier is shown in Figures 3.16 and 3.17, respectively. The core of the multiplier consists of NMOS transistors $M_1 - M_4$. Current source I_b and transistors M_a and M_b form a FVF cell. The difference between currents I_E and I_F results in output current $I_{out} = I_E - I_F = \mu_n C_{ox} W/L$. This expression holds true only if $V_E = V_F$, all the transistors are in triode mode, and if the sources driving nodes A and B have significantly low impedance. These conditions can be met using FVFs in feed-forward structure as current sensing elements and voltage buffers. The bottom FVFs create low impedance nodes E and F; and enable sensing the currents that pass through these nodes. Additionally, they are used to replicate these currents. The top FVFs implement very low impedance sources that drive nodes A and B. The resulting multiplier produces good linearity, operates at high frequencies, and has low supply (sub-volt) requirements in comparison with op-amp based multipliers.

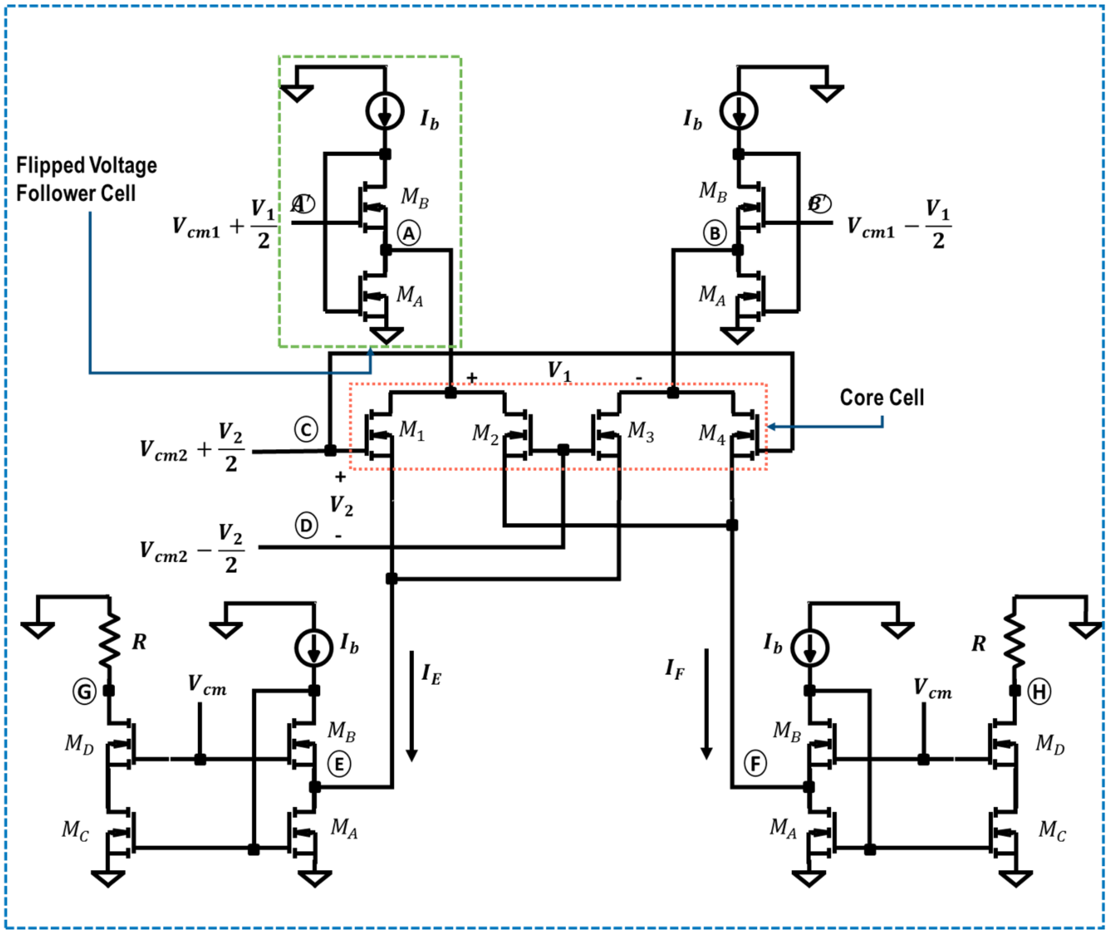
Flipped Voltage Follower Cell
Core Cell
I_b
M_B
M_A
A'
A
B'
B
$V_{cm1} + \frac{V_1}{2}$
$V_{cm1} - \frac{V_1}{2}$
V_1
M_1
M_2
M_3
M_4
C
D
$V_{cm2} + \frac{V_2}{2}$
$V_{cm2} - \frac{V_2}{2}$
V_2
R
G
H
E
F
V_{cm}
M_D
M_C
I_E
I_F

Figure 3.16: Four-quadrant analog Multiplier 1. All the transistors operate in linear region. Vcm =Vcm1 = Vcm2 = 1.4V; Ib = 600uA; R = 2kΩ; Input range is between ±0.4V [41].

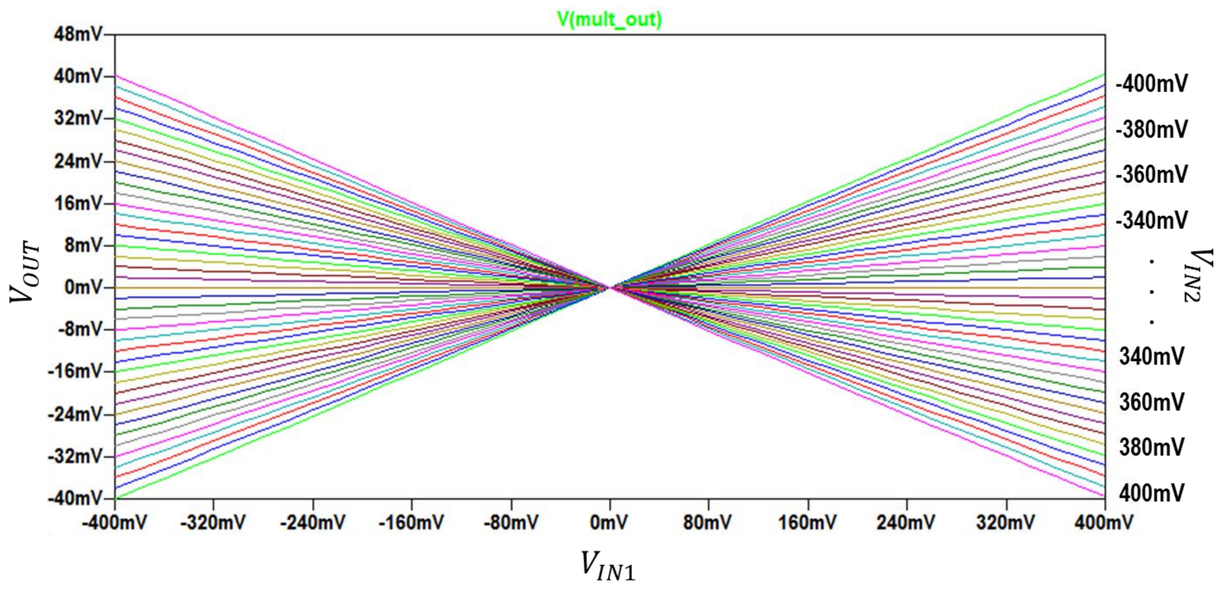

Figure 3.17: DC transfer characteristics of Multiplier 1 based on Flipped Voltage Follower cells. The multiplier circuit implements a scaled down (by -4) version of real multiplication.

In spite of its good linearity, the FVF-based multiplier was replaced in the final version of the voltage-based LSTM design by a more linear and more symmetric multiplier. This multiplier is based on symmetric complementary structure squarer circuits [43]. The circuit schematic and DC transfer characteristics of the latter multiplier is shown in Figures 3.18 and 3.19, respectively. In the former figure, the left-hand side transistors M_1 to M_6 constitute an analog voltage squarer with a “symmetric complementary push-pull source follower structure” [43]. Transistors M_5 and M_6 operate in triode region. The drain currents of M_5 and M_6 are expressed as following:

$$I_{D5} = \beta[(V_{GS5} - V_{Tn})V_{DS5} - \frac{1}{2}V_{DS5}^2], \tag{3.2}$$

$$I_{D6} = \beta[(V_{GS6} - V_{Tn})V_{DS6} - \tfrac{1}{2}V_{DS6}^2], \tag{3.3}$$

where $\beta = \mu_n C_{ox}\left(\frac{W}{L}\right)_n$. Then due to symmetry output currents in the circuit will be:

$$I_{o+} = I_{D5} + I_{D6} = -\beta(A+B)^2, \tag{3.4}$$

$$I_{o-} = I_{D11} + I_{D12} = -\beta(A-B)^2. \tag{3.5}$$

The difference of the above two currents will be:

$$I_o = I_{o+} - I_{o-} = -4\beta AB. \tag{3.6}$$

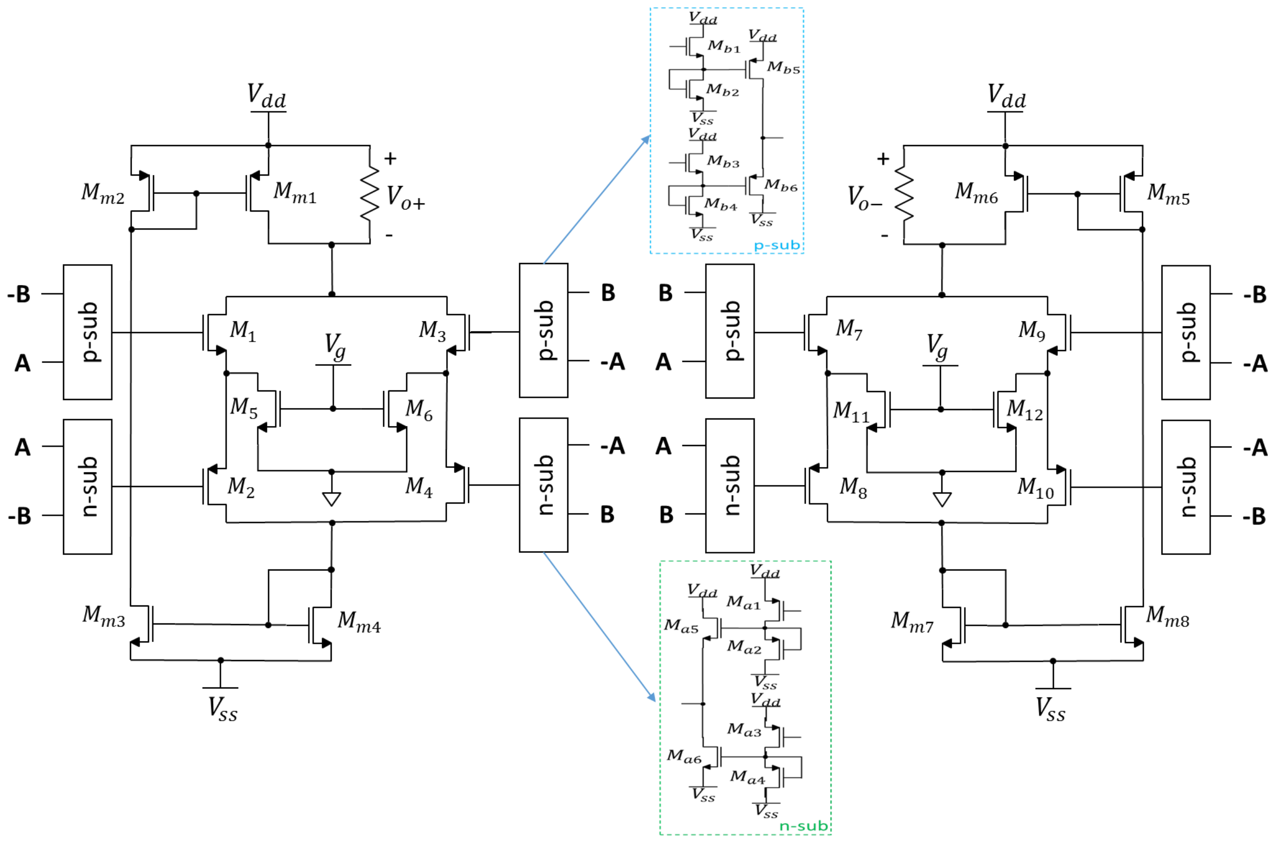

Figure 3.18: Four-quadrant analog Multiplier 2. Vg = 1.5V; R = 1kΩ. Input range is between ±0.5V, but ±0.1V is enough for implementation of LSTM [43].

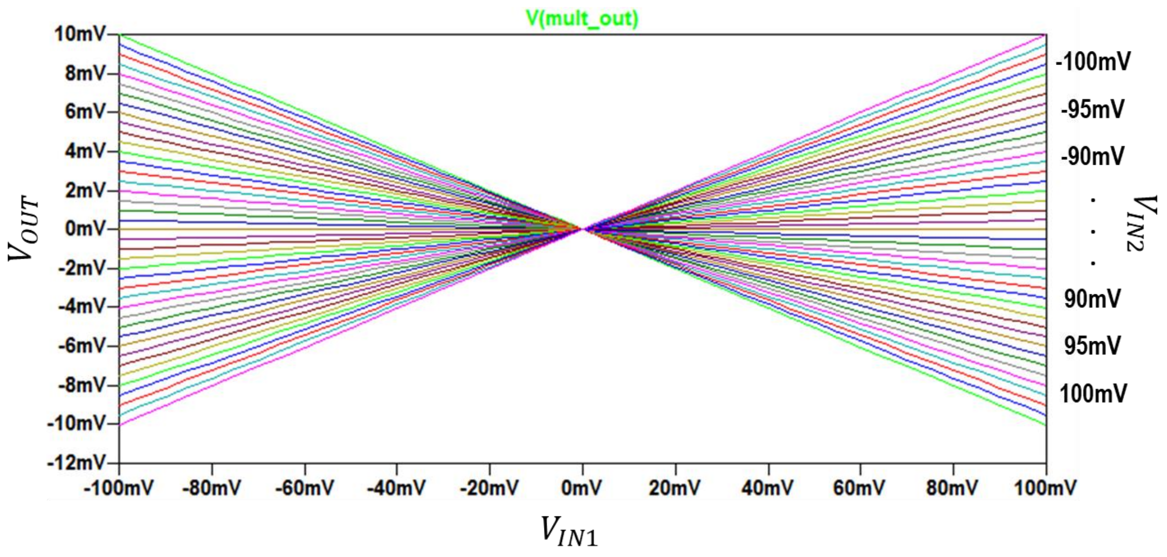

Figure 3.19: DC transfer characteristics of Multiplier 2 based on Symmetric Complementary Structure squarer circuits. The multiplier circuit implements a same-scale version of real multiplication.

3.5 LSTM Architectures

3.5.1 Vanilla LSTM

Hochreiter and Schmidhuber [15] invented the very first LSTM. Later the original LSTM evolved into the most common architecture [47] known as vanilla LSTM [48]. Figure 3.20 shows the detailed diagram of the vanilla LSTM layer. Vanilla LSTM differs from the standard LSTM used in Keras library [18] by additional weighted connections known as peepholes. Peephole weight vectors; and half-transparent dashed lines and math elements (multiplication and addition) are extra additions that separate the standard LSTM from the vanilla LSTM.

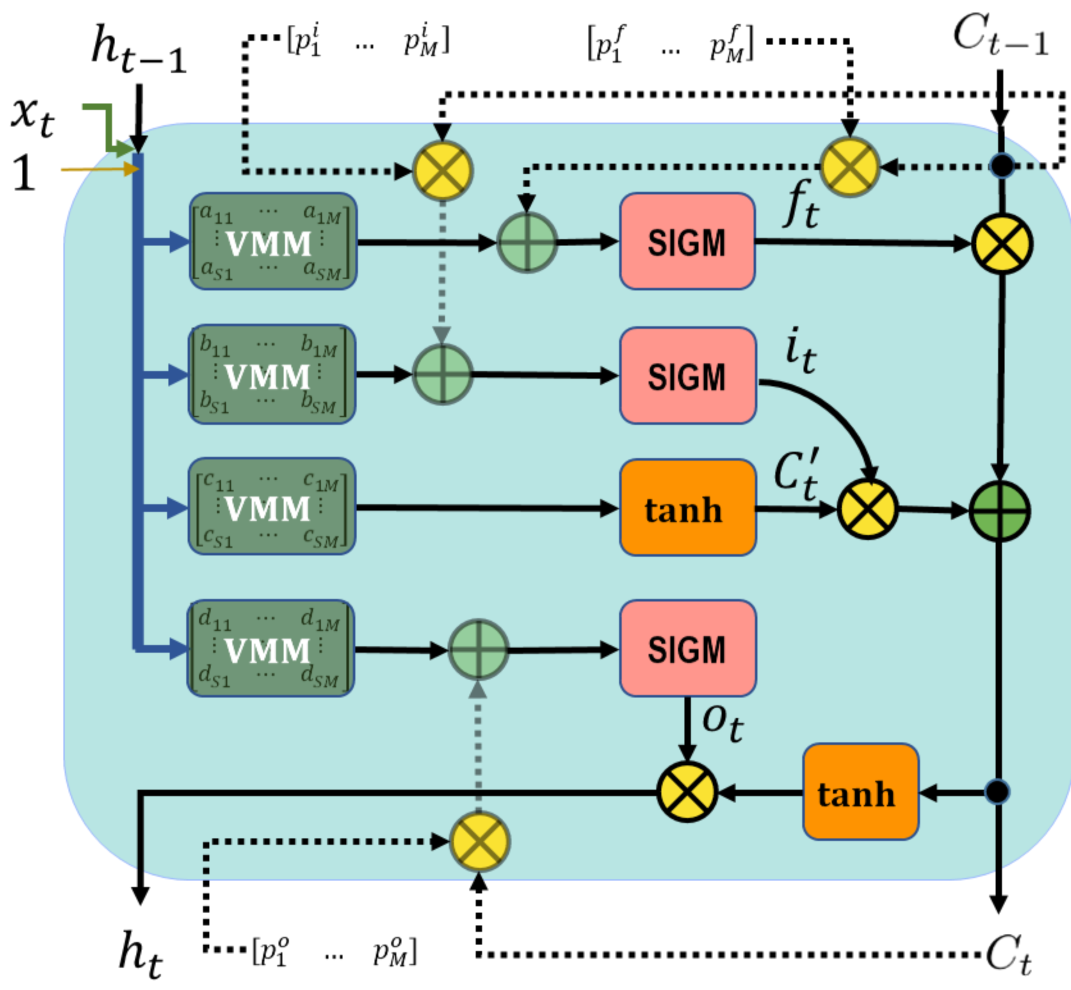
h_{t-1}
x_t
1
$[p_1^i \; \dots \; p_M^i]$
$[p_1^f \; \dots \; p_M^f]$
C_{t-1}
VMM
SIGM
tanh
f_t
i_t
C'_t
o_t
h_t
$[p_1^o \; \dots \; p_M^o]$
C_t

Figure 3.20: Detailed Vanilla LSTM layer diagram. Letters subscripted with t or t - 1 represent vectors of size M, except x_t which has size N. Bias is a scalar which is equal to unity.

All the lines in the diagram represent vectors. Blue line is the concatenation of following inputs along column axis: input vector (or feature vector) x_t of size 1-by-N, output vector from previous cell h_{t-1} of size 1-by-M, and a scalar bias value. Then the size of blue line is N+M+1 = S (denoted as S for short). The matrices inside Vector Matrix Multiplication (VMM) units represent weight matrices. They combine input, recurrent, and bias weights. The outputs of VMM units are row vectors of length M. In fact, all the black lines have size 1-by-M, where M is the number of LSTM hidden units or blocks. Therefore, ideally we should get M parallel operations and M outputs at each stage in the diagram. However, in hardware parallel operations may be implemented sequentially to save up chip area as in the proposed designs in this chapter. Then, as expected, multiplication elements perform element-wise (hadamard) multiplications; and addition elements perform vector additions.

3.5.2 Other Architectures

There are total of nine existing architectures of LSTM [48]. They can be classified into six categories: 1) Vanilla; 2) Standard (No Peepholes); 3) Full Gate Recurrence (FGR); 4) Coupled Input and Forget Gate (CIFG); 5) with a Linear Activation Function; and 6) with a Constant Gate of Unity.

The vanilla LSTM shown in Figure 3.20 can be described in concise mathematical forms below [48], where $\boldsymbol{Q}$ having size of S-by-M is the concatenation of input weight matrix $\boldsymbol{W}$, recurrent weight matrix $\boldsymbol{R}$, and bias weight vector $\boldsymbol{b}$ along row axis; and $\boldsymbol{q}$ is the input vector of size 1-by-S:

$$\boldsymbol{f_t} = \sigma(\boldsymbol{q_t Q_f} + \boldsymbol{C_{t-1}} * \boldsymbol{p_f}), \tag{3.6}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{q_t Q_i} + \boldsymbol{C_{t-1}} * \boldsymbol{p_i}), \tag{3.7}$$

$$\boldsymbol{C'}_{\boldsymbol{t}} = tanh(\boldsymbol{q_t Q_{C'}}), \tag{3.8}$$

$$\boldsymbol{C_t} = \boldsymbol{i_t} * \boldsymbol{C'_t} + \boldsymbol{f_t} * \boldsymbol{C_{t-1}}, \tag{3.9}$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{q_t Q_o} + \boldsymbol{C_t} * \boldsymbol{p_o}), \tag{3.10}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} * \tanh(\boldsymbol{C_t}), \tag{3.11}$$

where $\boldsymbol{p_f}$, $\boldsymbol{p_i}$, and $\boldsymbol{p_o}$ represent peephole weight vectors.

No peepholes version is self-explanatory. It has the same architecture as that of the vanilla LSTM except no peephole connections in its topology.

FGR is the most complex architecture of LSTM. As its name suggests, Full Gate Recurrence LSTM is featured by recurrent connections between its all gates. FGR is basically vanilla LSTM (except peephole weights of output gate are element-wise multiplied to previous cell state vector) plus the new recurrent connections among the gates. It adds to the weight matrix $\boldsymbol{Q}$ of each gate additional 3M-by-M recurrent weight matrix. Mathematically in detail, it can be described as following [48]:

$$\boldsymbol{f_t} = \sigma(\boldsymbol{q_t Q_f} + \boldsymbol{i_{t-1} R_{if}} + \boldsymbol{f_{t-1} R_{ff}} + \boldsymbol{o_{t-1} R_{of}} + \boldsymbol{C_{t-1}} * \boldsymbol{p_f}), \tag{3.12}$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{q_t Q_i} + \boldsymbol{i_{t-1} R_{ii}} + \boldsymbol{f_{t-1} R_{fi}} + \boldsymbol{o_{t-1} R_{oi}} + \boldsymbol{C_{t-1}} * \boldsymbol{p_i}), \tag{3.13}$$

$$\boldsymbol{o_t} = \sigma\big(\boldsymbol{q_t Q_o} + \boldsymbol{i_{t-1} R_{io}} + \boldsymbol{f_{t-1} R_{fo}} + \boldsymbol{o_{t-1} R_{oo}} + \boldsymbol{C_{t-1}} * \boldsymbol{p_o}\big). \tag{3.14}$$

CIFG architecture of LSTM is more known as GRU [49]. Again, its name (Coupled Input and Forget Gate) explains itself: $\boldsymbol{f_t} = \mathbf{1} - \boldsymbol{i_t}$. However, other than that, 1) there are also no peephole connections and no output activation function; 2) candidate cell state's recurrent inputs are filtered through output gate before being multiplied with recurrent weight matrix; and 3) cell state and cell output are combined together. The differences can be easier to see in mathematical equations:

$$\boldsymbol{C'_t} = tanh(\boldsymbol{x_t W_{C'}} + (\boldsymbol{h_{t-1}} * \boldsymbol{o_t})\boldsymbol{R_{C'}} + \boldsymbol{b_{C'}}), \tag{3.15}$$

$$\boldsymbol{h_t} = (\mathbf{1} - \boldsymbol{f_t}) * \boldsymbol{C'_t} + \boldsymbol{f_t} * \boldsymbol{h_{t-1}}. \tag{3.16}$$

The next category falls into LSTM architectures with a linear activation function either in a) candidate cell state generation stage or b) in output gate stage. From the vanilla LSTM equations only equations 3.8 and 3.11 change. In architecture a), known as No Input Activation Function (NIAF), equation 3.8 becomes $\boldsymbol{C'_t} = (\boldsymbol{q_t Q_{C'}})$. In architecture b), known as No Output Activation Function (NOAF), equation 3.11 becomes $\boldsymbol{h_t} = \boldsymbol{o_t} * \boldsymbol{C_t}$.

The last category includes LSTM architectures which have a constant gate of unity in one of its gates and everything else is the same as in the vanilla LSTM. Then there are three different such architectures: a) No Input Gate (NIG): $\boldsymbol{i_t} = \mathbf{1}$; b) No Forget Gate (NFG): $\boldsymbol{f_t} = \mathbf{1}$; and c) No Output Gate (NOG): $\boldsymbol{o_t} = \mathbf{1}$.

Chapter 4 – Results and Discussions

4.1 Current-based LSTM

Area of the proposed LSTM circuit is 83,493.5 μm^2 (without memory unit). The total power consumption of the single cell LSTM circuit is 105.9mW for input voltages between 0 and 1 Volts. Since this part does not provide a full circuit-level simulation solving the prediction problem, there are no results comparing circuit-level and system-level implementations of LSTM. The full circuit-level simulation was left out, because it was clear enough that voltage-based LSTM implementation would give higher accuracy of prediction. However, as a thorough comparison between the two implementations, the full circuit-level simulation of current-based LSTM used inside a neural network configuration can be considered as an open problem.

4.2 Voltage-based LSTM for time-series prediction problem

Figure 4.1 below shows the comparison of the results for the case with continuous weight values. The plots in this figure obtained while using 2-stage op-amps [39] and Flipped Voltage Follower [41] based multipliers. The numerical comparison of the plots is shown in Table 4.1.

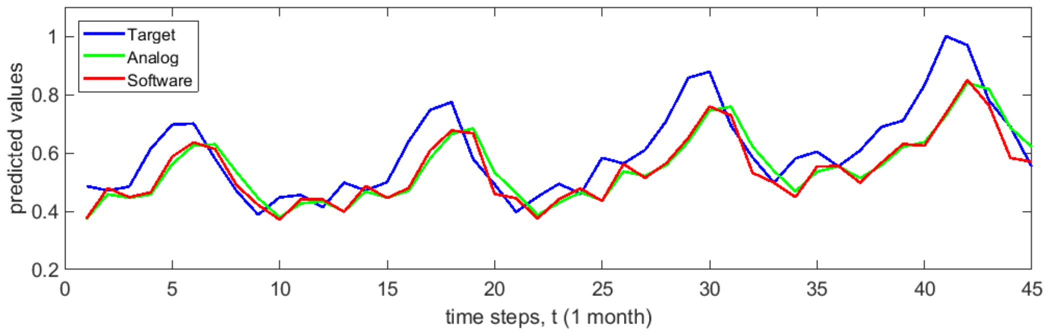

Figure 4.1: Visual Comparison of LSTM Software and Memristive (continuous weights) Analog implementation results against the Target values.

Table 4.1. Numerical Comparison Results for Figure 4.1 above.

Performance	Soft2Target	Analog2Target	Analog2Soft
'MSE'	0.010105	0.01108	0.00093553
'RSE'	0.59046	0.65461	0.068042
'MAE'	0.079965	0.086229	0.020943
'MAPE'	0.12425	0.13535	0.038815
'RMSE'	0.10052	0.10526	0.030586
'RRSE'	0.76841	0.80908	0.26085
'R2'	0.40954	0.34539	0.93196

Note that, in Table 4.1, performance results when estimating target values using software implementation of the algorithm are not good enough. An R2 score of only 0.41 means a not so good estimation. Probably, choosing more hidden units and more LSTM layers would result in higher scores. However, this was not the first priority in this work. The main goal was to demonstrate a successful implementation of the algorithm in analog hardware using memristive crossbar arrays. So far, this goal is achieved for some extent with an estimation score R2 of 0.932.

Figure 4.2 below shows the comparison of the results for both cases: with continuous and discrete weight values. In addition, the plots in this figure were obtained while using 3-stage RNIC-1 [42] op-amps and multiplier based on a symmetric complementary structure [43] for obtaining more accurate results. Again, the numerical comparison of plots is provided and shown in Table 4.2. From the table, it can be seen that new circuit units have helped to attain almost ideal circuit-level implementation of the whole neural network consisting of an LSTM layer and a Dense layer. R2 score jumps from 0.932 up to 0.995 when using the more accurate circuit components – op-amps and multipliers. With these new components even the implementation with discrete weights gives high enough R2 score – 0.975. This is a good score and promises that real memristors can give decent implementation scores. However, the last score was obtained when considering no noise in the memristors used in the circuit crossbar. Whereas, adding Gaussian noise to memristances of the memristors with the noise standard deviation equal to 5%, 10%, and 20% and running 30 Monte Carlo simulations give an expected degrading performance results as shown in Table 4.3. In addition, there is a need to simulate the real effect of wire resistances in the crossbar arrays similar to the case with memristor noises. Finally, combined effect of both non-idealities should be also tested in circuit simulations. This can be part of future work.

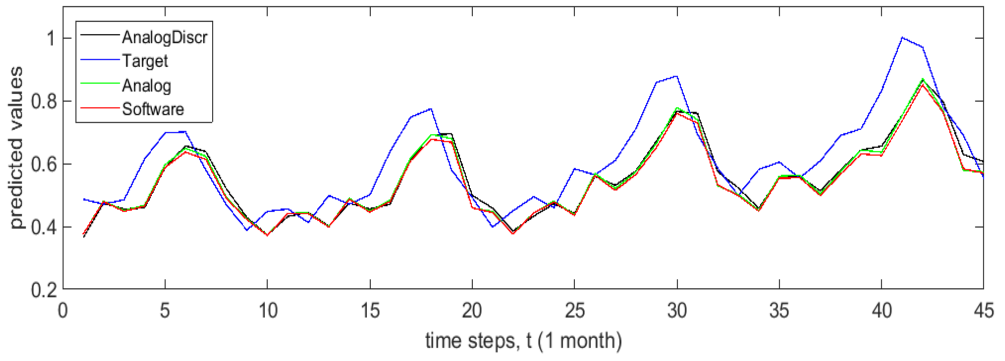

Figure 4.2: Visual Comparison of LSTM Software and Memristive (continuous and discrete states) Analog implementation results against the Target values using more accurate circuit parts. Total circuit simulation time take 3.96 ms to predict all 45 sample points.

Table 4.2: Numerical Comparison Results for Figure 4.2 above.

Performance	Soft2Target	Analog2Target	Analog2Soft	AnalogDiscr2Soft
'MSE'	0.010105	0.0091533	6.7539e-05	0.00037001
'RSE'	0.59046	0.52449	0.0048089	0.025086
'MAE'	0.079965	0.0773	0.0063793	0.015322
'MAPE'	0.12425	0.12156	0.010573	0.027883
'RMSE'	0.10052	0.095673	0.0082182	0.019236
'RRSE'	0.76841	0.72422	0.069346	0.15838
'R2'	0.40954	0.47551	0.99519	0.97491

Table 4.3: Average Performance results of 30 Monte Carlo simulations with Discrete Weights by adding to them Gaussian Noise with std. deviation equal to 5%, 10%, and 20% of Ron/Roff.

Performance Metrics	*AvgAnalogDiscr2Soft (5% weight noise)*	*AvgAnalogDiscr2Soft (10% weight noise)*	*AvgAnalogDiscr2Soft (20% weight noise)*
'MSE'	0.001163364	0.004764	0.014253
'RSE'	0.065070512	0.188028	0.332639
'MAE'	0.025969842	0.052764	0.092112
'MAPE'	0.047967865	0.097868	0.17319
'RMSE'	0.029740075	0.056086	0.095285
'RRSE'	0.231074325	0.385007	0.529074
'R2'	0.934929488	0.811972	0.667361

4.3 Voltage-based LSTM for time-series classification

The wafer quality classification can be solved in two ways: using sequential and parallel (window) methods. In the former case, a neural network consisting of an LSTM layer (with four hidden units) plus a dense layer (with a single unit) is used to predict the 153rd element (wafer quality) using 152 single sensor measurement values as a single feature spread in time. In the latter case a single LSTM layer with a single hidden unit and 152 features is used. That is those 152 single sensor measurement values are now fed in single time-step and therefore no time dimensionality exists among the sensor measurement values anymore. The output of the LSTM layer is a predicted wafer class in this latter method.

Each of them has different advantages and shortcomings. The case with sequential LSTM learns significantly slower than the one with parallel or single-time step predicting LSTM. This is due to the complexity of the former configuration

However, it steadily improves in performance as the number of epoch size is increased. It gives accuracies of 97.26%, 98.51%, and 98.86% for epoch sizes of 25, 40, and 55 respectively. The circuit of the sequential network consumes 255.8mW when the maximum input voltage values set to 0.5V. The on-chip area of the circuit is 257,503.20 um^2 which accounts the sizes for memory unit circuits and switches. On the other hand, the parallel LSTM has smaller on-chip area – 115,967.4 um^2. However, as expected, it consumes more power – 312.4 mW. The single-unit single-time step LSTM gives accuracy of 96.09% when using epoch size of 40 for training. Interestingly, the accuracy reaches up to 99.29% when epoch size is increased to 100. These kinds of high accuracy values are impressive considering the large imbalance in the data of wafer classification task: 10.7% of the training data and 12.1% of the testing data have abnormal labels. It may suggest that even the simple LSTM structure in fact complex enough to give such high accuracy percentages.

The circuit implementation of the sequential network configuration was proposed in the previous chapter. The results of the circuit simulations of the network are presented in figures below. In addition to the sequential operation of LSTM in the network, each hidden unit outputs and cell state values were obtained in sequential manner in the circuit. That is, four hidden units were not executed in parallel. It helps to save on chip-area, but sacrifices the overall execution speed. In Figure 4.3, we can see that total simulation time to obtain all 152*4 hidden unit values or to predict the last element/classify is 6.387ms. Whereas, parallel circuit implementation of the same network would take approximately 1.6ms. Extraction of outputs from the plot in Figure 4.3 results in separate plots that are illustrated in Figure 4.4 which also compares them with their corresponding software results. From the figure, it can be seen that in general the analog circuit results for the hidden units follow the plots obtained from the software implementation of the LSTM layer in the network. The spikes in the plot are results of the saturated outputs of the multiplier used in the circuit. The value of cell state C_t when accumulated through 152 time-steps, it exceeds the input range of the multiplier. However, we are more interested in the predicted value's sign: positive – normal wafer, negative – abnormal wafer. In fact, Table 4.4 shows that analog hardware testing results of 10 wafers, taken randomly from 1000 wafer datasets, match software testing results using LSTM algorithm.

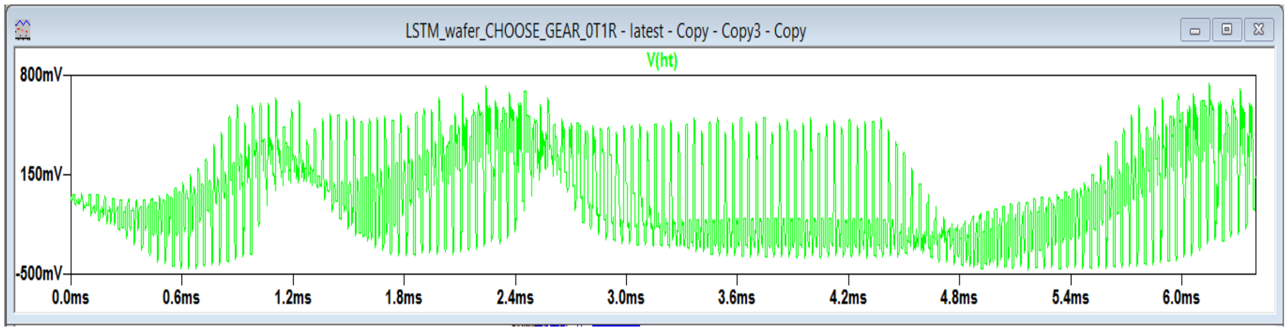

Figure 4.3: Plot of V(ht) voltage values from sequential operation of the LSTM circuit predicting the wafer quality class of test wafer 23. Total circuit simulation time is 6.387ms.

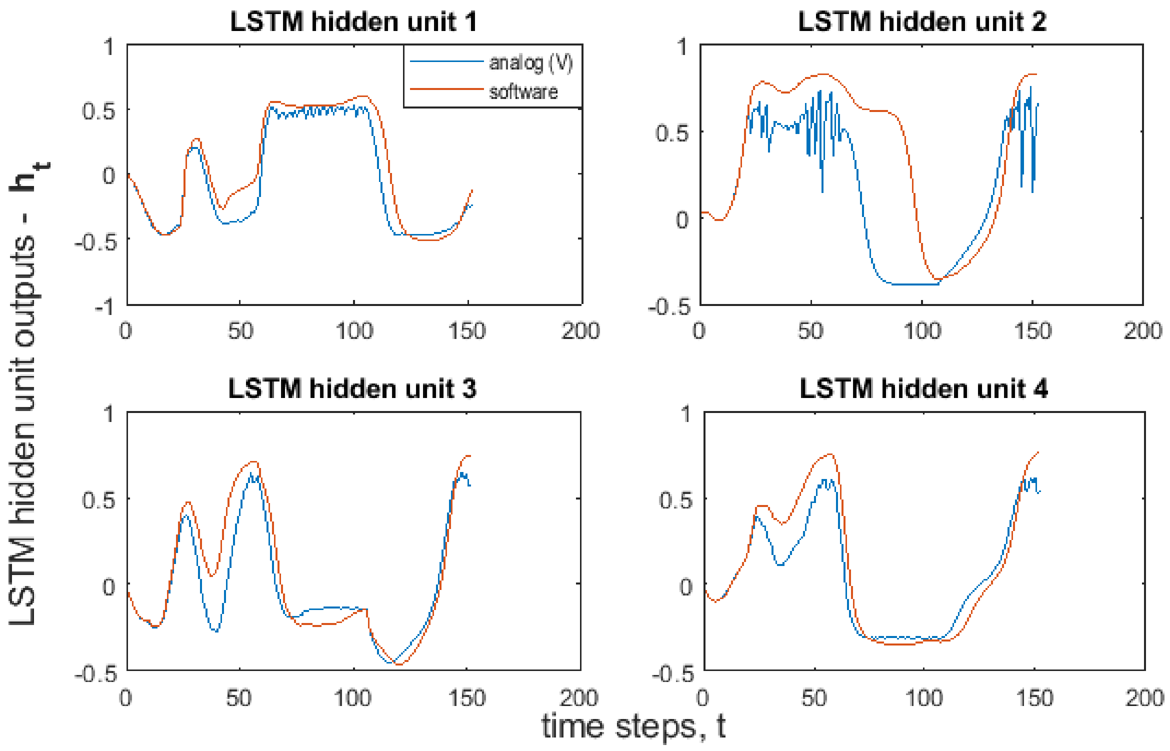

Figure 4.4: LSTM cell outputs for analog and software implementations for test wafer 23. The analog outputs were extracted from the plot in Figure 4.3.

Table 4.4: Comparison of Software and Hardware Implementation Results of LSTM.

Wafer test number	Predicted value (analog) (-mV)	Predicted value (software) (1e-3)	Class
23	-171.5	-423.7	-1
47	523.2	473.0	1
7	-247.4	-507.4	-1
3	470.7	511.6	1
3838	-485.5	-418.3	-1
193	501.84	497.0	1
6157	-247.3	-460.1	-1
411	531.9	489.8	1
1534	-437.2	-456.2	-1
4507	255.6	493.6	1

4.4 Power consumption and chip size

In Table 4.5, the first three implementations of the LSTM correspond to the network configuration designed for solving the time-series prediction problem 1 (prediction of international airline passenger count). The last two implementations correspond to the configuration solving the problem 3 (classification of wafer quality). The first and last implementations do not offer simulation results solving the problems 1 and 3, respectively. They can be a part of future works.

From the table, implementations 1 and 2 use two-stage operational amplifiers and the rest of the implementations use three-stage operational amplifiers. The difference between the two operational amplifiers is shown in Table 4.6. In addition, the third implementation, voltage-based 1b, uses different type of analog multiplier than the one used in the other voltage-based implementations. The difference of the two multiplier types in terms of power and area is shown in Table 4.7.

Table 4.5: Area and Power statistics from different LSTM circuit implementations.

#	***LSTM circuit implementation***	***Area (μm^2)***	***Power (mW)***	***Mem. Units/ Dense layer?***	***Input Range (V)***	***$R_{off}(\Omega)$/ $R_{on}(\Omega)$***
1	***Current-based***	*83,494*	*105.9*	*No/No*	*[0, 1]*	*2M/200k*
2	***Voltage-based 1a***	*117,075*	*225.67*	*Yes/Yes*	*[-1, 1]*	*10M/10k*
3	***Voltage-based 1b***	*126,062*	*228.11*	*Yes/Yes*	*[-0.1, 0.1]*	*10k/1.1k*
4	***Voltage-based 2 seq.***	*257,503*	*237.03*	*Yes/Yes*	*[-1, 1]*	*10M/10k*
5	***Voltage-based 2 par.***	*115,967*	*312.4*	*NA/NA*	*[-0.1, 0.1]*	*10M/10k*

Table 4.6: Area and Power statistics for the two types of op-amps in buffer configuration.

	Area (μm^2)	*Power (mW) at -0.1 V input*	*Power (mW) at 0.1 V input*	*Power (mW) at -1 V input*	*Power (mW) at 1 V input*
two-stage op-amp	*847.56*	*3.10*	*3.00*	*3.51*	*2.58*
three-stage op-amp	*1977.8*	*3.59*	*5.54*	*0.75*	*10.23*

In Table 4.6, the area difference is mainly due to the more usage of capacitance values in the three-stage op-amp for stability purposes than in the two-stage op-amp. In addition, the two-stage op-amp uses ideal current source for biasing, while the three-stage op-amp uses two MOSFETs and a resistor. The supply voltage rails used are (-1.8V, 1.8V) for the two-stage op-amp and (-1V, 1.8V) for the three-stage op-amp.

Three-stage amplifier has larger on-chip area and generally higher power consumption than that of the two-stage op-amp. However, accuracy-wise three-stage amplifier delivers good performance and works well with small-signal voltage values. This accuracy helps to obtain accurate intermediate voltage values in the circuits for LSTM and results in accurate final voltage values. Particularly, inverting, scaling up or down, summation, and multiplication of voltage signals decides the overall accuracy of a circuit-level LSTM compared to system-level LSTM.

In Table 4.7, the first multiplier is Flipped Voltage Follower cell based four-quadrant analog multiplier. The second multiplier is based on symmetric complementary structure squarer circuits. The first one uses the two-stage op-amps for its interface circuitry and for the extension of its outputs to make a single-ended output. Whereas, the second one uses the three-stage op-amps for the same purposes. In the table, for the comparison sake, the input voltages were set to the same values – (0.4, 0.4) Volts. However, multiplier 2 is used in the LSTM circuit implementation with input voltage range of [-0.1, 0.1] Volts (can work between [-0.5, 0.5] Volts, though). For maximum inputs of (0.1, 0.1) Volts, multiplier two consumes 29.05 mW power. Overall, multiplier 2 is smaller in size, consumes less power, and contributes to higher R2 score when used inside the complete circuit of the neural network consisting LSTM and dense layers. Therefore, it is a good candidate for future circuit designs, not only the circuit implementation of LSTM. Circuit designs Voltage-based 2 sequential and Voltage-based 2 parallel can be decreased in size by at least 45,000 um^2 each.

Table 4.7: Area, Power, and contributed accuracy statistics for the two types of multipliers.

	Area (μm^2)	***Power (mW) for max inputs of (0.4,0.4) Volts***	***R2 of LSTM + Dense circuit***
Multiplier 1	*24,430*	*35.36*	*0.932*
Multiplier 2	*9,139*	*30.12*	*0.995*

4.5 LSTM architectures

Average test and train performance metrics from 10 Monte Carlo simulations were obtained for the Simple RNN and the LSTM architectures from the previous chapter. In addition, for exploration purposes two more sets of simulations with peephole weight matrices having sizes of M-by-M and 0-by-0 (i.e. ne peephole connections) were performed. M is the number of hidden units. These are contrary to the usual peephole weight matrix size of 1-by-M. The simulations were trained and tested on three different datasets from the selected problems.

It is important to note from tables 4.9-4.14 that the least average test RMSE (highlighted in yellow) does not necessarily correspond to the least average train RMSE (highlighted in yellow, as well). Even though, there is a general trend of correspondence, one should not solely rely on train scores when comparing the different architecture performances of RNN. In the tables 4.15-4.17 the correspondence exists between testing and training performance metrics. It may be due to the large training data available in the wafer classification case which leaves no chance for mixed results. Tables 4.9-4.17 can be summarized in Table 4.8 below:

Table 4.8: Winner types of RNN for different problems and peephole weight matrix sizes. Left half columns correspond to the best performers on the testing data and the right half columns on the training data. The color intensity corresponds to the ranking of a performer inside a column. For example, NOAF LSTM 2 is the best in the first column.

Peephole weight matrix size	***Airflight passenger count prediction***		***CO^2 emission prediction***		***Time Ser. Classification of wafer quality***	
M-by-M	*NFG LSTM*	*FGR LSTM*	*NOG LSTM*	*FGR LSTM*	*FGR LSTM*	*FGR LSTM*
1-by-M	*NOAF LSTM*	*NOG LSTM*	*NOG LSTM*	*FGR LSTM*	*NOG LSTM*	*NOG LSTM*
0-by-0	*NOAF LSTM 2*	*GRU*	*GRU*	*FGR LSTM 2*	*NOG LSTM 2*	*NOG LSTM 2*

As it was already tested in other study [48], there are mixed performance results from different LSTM architectures for each task. The study was carried out on classification tasks, while in this work it is done on time-series prediction and time-series classification tasks. From Table 4.8, it can be concluded that there are mixed results with time-series prediction tasks as well. Note that wafer classification task is modeled as time-series prediction task where the last predicted point determines the class of the wafer under study. Positively predicted number indicates non-defective wafer and vice-versa.

However, some patterns can be observed from Table 4.8. The peephole connections do not improve the performance of LSTM architectures for the first two cases. That is, as the size of the peephole weight matrices grows, the peephole connections only degrade the performances of RNN types for the cases with small training data. The performances of the two cases for the training data give mixed results, but their difference is very small. However, both of the performances improve for the third case as the peephole weight matrix size grows. This is probably caused by the large training data and the large number of time-steps used for prediction in the third case. Also, note that the most complex architecture, FGR LSTM, beats the others in the third case. Therefore, it may be sound to conclude that in general as the training data becomes large, the more complex LSTM architectures outperform the rest of the architectures. Using the same logic, for small training data less complex LSTM architectures outperform the less complex ones.

Table 4.9: Performance comparison of LSTM architectures and Simple RNN for prediction of number of international air-flight passengers (a peephole weight matrix has M-by-M size).

#	*Type of RNN*	*Avg. Test RMSE*	*Avg. Train RMSE*
1	*Vanilla LSTM*	*0.107112*	*0.040618*
2	*No-output-gate LSTM*	*0.111376*	*0.040731*
3	*No-input-gate LSTM*	*0.113707*	*0.041158*
4	*No-forget-gate LSTM*	***0.102329***	*0.044298*
5	*No-input-activation-function LSTM*	*0.121440*	*0.040357*
6	*No-output-activation-function LSTM*	*0.105856*	*0.040538*
7	*Full-Gate-Recurrence LSTM*	*0.116826*	***0.040008***

Table 4.10: Performance comparison of LSTM architectures and Simple RNN for prediction of number of international air-flight passengers (a peephole weight matrix has 1-by-M size).

#	*Type of RNN*	*Avg. Test RMSE*	*Avg. Train RMSE*
1	*Vanilla LSTM*	*0.100513*	*0.042934*
2	*No-output-gate LSTM*	*0.105852*	***0.040247***
3	*No-input-gate LSTM*	*0.104337*	*0.040982*
4	*No-forget-gate LSTM*	*0.107426*	*0.046170*
5	*No-input-activation-function LSTM*	*0.105890*	*0.042321*
6	*No-output-activation-function LSTM*	***0.097092***	*0.042351*
7	*Full-Gate-Recurrence LSTM*	*0.109990*	*0.041353*

Table 4.11: Performance comparison of LSTM architectures for prediction of the number of international air-flight passengers (a peephole weight matrix has 0-by-0 size).

#	*Type of LSTM with no peepholes*	*Avg. Test RMSE*	*Avg. Train RMSE*
1	*Vanilla LSTM 2 (simple LSTM)*	*0.102161*	*0.043653*
2	*No-output-gate LSTM 2*	*0.105260*	*0.040340*
3	*No-input-gate LSTM 2*	*0.103408*	*0.041291*
4	*No-forget-gate LSTM 2*	*0.108595*	*0.046391*
5	*No-input-activation-function LSTM 2*	*0.106734*	*0.042972*
6	*No-output-activation-function LSTM 2*	***0.095907***	*0.043334*
7	*Full-Gate-Recurrence LSTM 2*	*0.109751*	*0.041412*
8	*Gated-Recurrent-Units*	*0.110889*	***0.040264***
9	*Simple RNN*	*0.113199*	*0.041263*

Table 4.12: Performance comparison of LSTM architectures and Simple RNN for prediction of CO^2 emission volumes at volcano Mauna Loa (a peephole weight matrix has M-by-M size).

#	*Type of RNN*	*Avg. Test RMSE*	*Avg. Train RMSE*
1	*Vanilla LSTM*	*0.048001*	*0.036825*
2	*No-output-gate LSTM*	***0.046437***	*0.033924*
3	*No-input-gate LSTM*	*0.048494*	*0.034320*
4	*No-forget-gate LSTM*	*0.049900*	*0.041443*
5	*No-input-activation-function LSTM*	*0.062435*	*0.036595*
6	*No-output-activation-function LSTM*	*0.046583*	*0.036172*
7	*Full-Gate-Recurrence LSTM*	*0.051111*	***0.033119***

Table 4.13: Performance comparison of LSTM architectures and Simple RNN for prediction of CO^2 emission volumes at volcano Mauna Loa (a peephole weight matrix has 1-by-M size).

#	*Type of RNN*	*Avg. Test RMSE*	*Avg. Train RMSE*
1	*Vanilla LSTM*	*0.045750*	*0.041151*
2	*No-output-gate LSTM*	***0.044268***	*0.034532*
3	*No-input-gate LSTM*	*0.044485*	*0.034321*
4	*No-forget-gate LSTM*	*0.055494*	*0.047465*
5	*No-input-activation-function LSTM*	*0.056436*	*0.041155*
6	*No-output-activation-function LSTM*	*0.045708*	*0.039960*
7	*Full-Gate-Recurrence LSTM*	*0.045578*	***0.033110***

Table 4.14. Performance comparison of LSTM architectures for prediction of CO^2 emission volumes at volcano Mauna Loa (a peephole weight matrix has 0-by-0 size).

#	*Type of LSTM with no peepholes*	*Avg. Test RMSE*	*Avg. Train RMSE*
1	*Vanilla LSTM 2 (simple LSTM)*	*0.046509*	*0.042726*
2	*No-output-gate LSTM 2*	*0.044248*	*0.034823*
3	*No-input-gate LSTM 2*	*0.044519*	*0.034716*
4	*No-forget-gate LSTM 2*	*0.057674*	*0.049057*
5	*No-input-activation-function LSTM 2*	*0.051584*	*0.042861*
6	*No-output-activation-function LSTM 2*	*0.044520*	*0.041993*
7	*Full-Gate-Recurrence LSTM 2*	*0.045378*	***0.033151***
8	*Gated-Recurrent-Units*	***0.043872***	*0.034458*
9	*Simple RNN*	*0.055096*	*0.033425*

Table 4.15: Performance comparison of LSTM architectures and Simple RNN for classification of wafer quality (a peephole weight matrix has M-by-M size).

#	*Type of RNN*	*Avg. Test Accuracy*	*Avg. Train Accuracy*
1	*Vanilla LSTM*	*0.938546*	*0.948100*
2	*No-output-gate LSTM*	*0.954932*	*0.961500*
3	*No-input-gate LSTM*	*0.897599*	*0.911800*
4	*No-forget-gate LSTM*	*0.896966*	*0.912300*
5	*No-input-activation-function LSTM*	*0.939195*	*0.948000*
6	*No-output-activation-function LSTM*	*0.934523*	*0.944200*
7	*Full-Gate-Recurrence LSTM*	***0.965688***	***0.967300***

Table 4.16: Performance comparison of LSTM architectures and Simple RNN for classification of wafer quality (a peephole weight matrix has 1-by-M size).

#	*Type of RNN*	*Avg. Test Accuracy*	*Avg. Train Accuracy*
1	*Vanilla LSTM*	*0.934523*	*0.942600*
2	*No-output-gate LSTM*	***0.950584***	***0.957900***
3	*No-input-gate LSTM*	*0.891467*	*0.907000*
4	*No-forget-gate LSTM*	*0.891467*	*0.907000*
5	*No-input-activation-function LSTM*	*0.936016*	*0.944600*
6	*No-output-activation-function LSTM*	*0.919825*	*0.932100*
7	*Full-Gate-Recurrence LSTM*	*0.935886*	*0.944600*

Table 4.17: Performance comparison of LSTM architectures with no peephole connections for classification of wafer quality (a peephole weight matrix has 0-by-0 size).

#	*Type of LSTM with no peepholes*	*Avg. Test Accuracy*	*Avg. Train Accuracy*
1	*Vanilla LSTM 2 (simple LSTM)*	0.925146	0.935600
2	*No-output-gate LSTM 2*	**0.942213**	**0.949100**
3	*No-input-gate LSTM 2*	0.891467	0.907000
4	*No-forget-gate LSTM 2*	0.891467	0.907000
5	*No-input-activation-function LSTM 2*	0.930565	0.940500
6	*No-output-activation-function LSTM 2*	0.930419	0.940100
7	*Full-Gate-Recurrence LSTM 2*	0.933274	0.942900
8	*Gated-Recurrent-Units*	*0.897972*	*0.912900*
9	*Simple RNN*	*0.891467*	*0.907100*

Chapter 5 – Conclusions

In this thesis, a functional circuit-level implementation of a neural network consisting of LSTM layer has been performed. The neural network was used to solve several time-series prediction problems. Therefore, custom circuits for each problem was built and tested in SPICE circuit simulator using TSMC 0.18 um process technology. In addition, extensive study of LSTM architectures was performed using Keras library, which was extended in this work.

5.1 Time-Series Prediction

Circuit simulation of problem 1 with discrete weight/conductance levels and discrete input voltages gives less accurate results as expected: R2 of 97.5 against 99.52; and RRSE of 0.158 against 0.0693. It is important to remember that training was performed in software and the extracted weights were discretized to 68 levels then used as stable conductance levels of memristors in the circuit. However, those 68 states themselves, in fact, are not clear-cut states. Each state will have some noise and we have seen that as the noise increases, the prediction accuracy in analog hardware degrades. If, however, training was performed on the circuit, it would be able keep running epochs until reaching some point where the states including their noises are contributing for the performance of the algorithm used in the circuit. In addition, time-series predictions are not 100% accurate and gives some room to tolerate the errors coming from the real circuits.

5.2 Wafer Classification

In the classification problem 3, using high accuracy circuit parts eliminates at the final stage additional circuitry such as circuit implementing softmax function required for classifying. Reading of the analog outputting voltage would already tell the predicted class. In addition, having less complex neural network due to omitting softmax function would make the network train faster both in hardware and software.

In addition, one can further simplify the neural network circuit by eliminating activation functions and use VMM outputs as approximate activated values [44]. This can be a close approximation if VMM outputs fall into linear part of sigmoid and hyperbolic tangent functions. This can be achieved by scaling feature input values going into the network. For example, scaling the feature values between -0.5 and 0.5 gives better results than scaling them between -1 and 1 in the wafer classification problem.

5.3 Flexibility of the Design

Since there is a trade-off between computation time, chip-area, and power consumption and depending on an application, the final design of the circuit will be different. In that sense, it would be convenient if the base circuit can be easily transformed to suit the requirements of a particular application. Voltage-based LSTM architecture comes to be handy on this occasion. In addition, adding M-by-M peephole weight matrices becomes effortless in hardware implementation, because VMM operation is very efficient in the hardware. However, sneak path problem will become more serious as the size of a crossbar becomes larger. It can be solved by transistors in series with each memristor as in [46].

5.4 Performance of LSTM architectures

Different LSTM architecture were used in a two-layer network for solving three different time-series prediction problems. It was found that complexity of an LSTM architecture becomes an advantage when using large training datasets. Whereas, less complex LSTM architectures such as NOAF and GRU outperform more complex FGR LSTM for problems with small training datasets. In addition, using more peephole connections increases the performance of FGR LSTM for large available training data.

REFERENCES

[1] Vora PH and Lad R. A Review Paper on CMOS, SOI and FinFET Technology. https://www.design-reuse.com/articles/41330/cmos-soi-finfet-technology-review-paper.html Accessed: 07-December-2018

[1] Watkins T. “The history of the Transistor.” Internet: San Jose State University. http://www.sjsu.edu/faculty/watkins/transist.htm Accessed: 07-December-2018

[2] Kvatinsky S. “Memristor-Based Circuits and Architectures,” Ph.D dissertation, Israel Institute of Technology, Haifa, 2014. [Online]. Available: http://www.hajim.rochester.edu/ece/users/friedman/Shahar_Kvatinsky_PhD.pdf

[3] Kim H-K, Gaba S, Wheeler D, Cruz-Albrecht JM, Hussain T, Srinivasa N and Lu W. "A Functional Hybrid Memristor Crossbar-Array/CMOS Systems for Data Storage and Neuromorphic Applications," *Nano Letters*, Vol. 12, No. 1, pp. 389-395, 2011.

[4] Soudry D, Di Castro D, Gal A, Kolodny A and Kvatinsky S. "Memristor-Based Multilayer Neural Networks With Online Gradient Descent Training," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 10, pp. 2408-2421, 2015. doi: 10.1109/TNNLS.2014.2383395

[5] Serrano-Gotarredona T, Masquelier T, Prodromakis T, Indiveri T and Linares-Barranco T. "STDP and STDP Variations with Memristors for Spiking Neuromorphic Learning Systems," *Frontiers in Neuroscience*, Vol. 7, No. 2, pp. 1-15, 2013.

[6] Afifi A, Ayatollahi A and Raissi F. "Implementation of Biologically Plausible Spiking Neural Network Models on the Memristor Crossbar-Based CMOS/Nano Circuits," *Proceedings of the European Conference on Circuit Theory and Design*, pp. 563-566, 2009.

[7] Jo SH, Chang T, Ebong I, Bhadviya BB, Mazumder P and Lu W. "Nanoscale Memristor Device as Synapse in Neuromorphic Systems," *Nano Letters*, Vol. 10, No. 4, pp. 1297-1301, 2010.

[8] Pershin YV and Di Ventra M. "Practical Approach to Programmable Analog Circuits with Memristors," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 57, No. 8, pp.1857-1864, 2010.

[9] Chua L. “Memristor-the missing circuit element,” *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, Sep 1971.

[10] Chua LO. and Kang SM. “Memristive Devices and Systems,” *Proceedings of the IEEE*, Vol. 64, No. 2, pp. 209- 223, February 1976.

[11] Joglekar YN and Wolf SJ. "The elusive memristor: properties of basic electrical circuits", European Journal of Physics, vol. 30, pp. 661-675, 2009.

[12] Strukov DB, Snider GS, Stewart DR, and Williams RS. "The Missing Memristor Found,” *Nature*, Vol. 453, pp. 80-83, May 2008.

[13] Steinbuch, K. Die lernmatrix. *Kybernetik* 1, 36–45. 1961.

[14] Li C, Belkin D, Li Y, et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks. *Nature Communications*. 9(1). doi:10.1038/s41467-018-04484-2.

[15] Hochreiter S and Schmidhuber J. Long short-term memory. Neural Computation, 9(8):1735-1780, 1997.

[16] Lipton ZC, Berkowitz J, Elkan C. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv preprint arXiv:1506.00019*, 2015.

[17] Zaremba W, Sutskever I, and Vinyals O. "Recurrent neural network regularization." *arXiv preprint arXiv:1409.2329*, 2014.

[18] Recurrent Layers - Keras Documentation. https://keras.io/layers/recurrent/#lstm. Accessed 09-December-2018

[19] Sutskever I, Vinyals O and Le QV. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104-3112, 2014.

[20] Vinyals O, Toshev A, Bengio S, and Erhan D. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156-3164, 2015.

[21] Karpathy A and Fei-Fei L. Deep visual-semantic alignments for generating image descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2017; (4):664.

[22] Mao J, Xu W, Yang Y, Wang J, and Yuille A. Deep captioning with multimodal recurrent neural networks (m-RNN). *arXiv preprint arXiv:1412.6632*, 2014.

[23] Venugopalan S, Rohrbach M, Donahue J, et al. Sequence to sequence-video to text. *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, 2015, pp. 4534-4542. doi: 10.1109/ICCV.2015.515

[24] Xiong W, Wu L, Alleva F, Droppo J, Huang X, Stolcke A. The Microsoft 2017 Conversational Speech Recognition System. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. 2018-April: 5934-5938. doi:10.1109/ICASSP.2018.8461870.

[25] Chang AXM, Martini B, Culurciello E. Recurrent Neural Networks Hardware Implementation on FPGA. *arXiv preprint arXiv:1511.05552v1*, 2015.

[26] Ferreira JC and Fonseca J. "An FPGA implementation of a long short-term memory neural network," *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Cancun, 2016, pp. 1-8. doi: 10.1109/ReConFig.2016.7857151

[27] Zhang Y, Wang C, Gong L, et al. "Implementation and Optimization of the Accelerator Based on FPGA Hardware for LSTM Network," *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, Guangzhou, 2017, pp. 614-621. doi: 10.1109/ISPA/IUCC.2017.00098

[28] Chen K, Huang L, Li M, Zeng X and Fan Y. "A Compact and Configurable Long Short-Term Memory Neural Network Hardware Architecture," *2018 25th IEEE International*

Conference on Image Processing (ICIP), Athens, Greece, 2018, pp. 4168-4172. doi: 10.1109/ICIP.2018.8451053

[29] Rizakis M, Venieris SI, Kouris A, and Bouganis C-S. Approximate FPGA-based LSTMs under Computation Time Constraints. *arXiv preprint arXiv:1801.02190*, 2018

[30] Smagulova K, Krestinskaya O, and James AP. A memristor-based long short term memory circuit. *Analog Integr Circ Sig Process*. 2018. https://doi.org/10.1007/s10470-018-1180-y

[31] Gokmen T, Rasch MJ and Haensch W. Training LSTM Networks With Resistive Cross-Point Devices *Front Neurosci*. 2018; 12: 745. doi: 10.3389/fnins.2018.00745

[32] Li C, Wang Z, Rao M, et al. Long short-term memory networks in memristor crossbars. 2018. doi:10.1038/s42256-018-0001-4.

[33] Brownlee J. "Time series prediction with lstm recurrent neural networks in python with keras," Available at: machinelearningmastery.com, 2016.

[34] Box and Jenkins. International airline passengers: monthly totals in thousands. https://datamarket.com/data/set/22u3/international-airline- passengers-monthly-totals-in-thousandsjan-49-dec-60. Accessed 01-December-2018, 1976.

[35] Hipel and McLeod. CO2 (ppm) mauna loa, 1965-1980. https://datamarket.com/data/set/22v1/co2-ppm-mauna-loa-1965-1980#!ds=22v1&display=line, 1994.

[36] Olszewski RT, "Generalized feature extraction for structural pattern recognition in time-series data," CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 2001.

[37] Kingma DP and Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014

[38] Optimizers - Keras Documentation. https://keras.io/optimizers/ Accessed 09-December-2018

[39] Krestinskaya O, Salama K, James AP. Learning in Memristive Neural Network Architectures using Analog Backpropagation Circuits. IEEE Transactions on Circuits and Systems I: Regular Papers. 2018. doi: 10.1109/TCSI.2018.2866510

[40] Xiao S, Xie X, Wen S, Zeng Z, et al. “Gstmemristor-based online learning neural networks,” *Neurocomputing*, 2017.

[41] Ramirez-Angulo J, Thoutam S, Lopez-Martin A, et. al. Low-voltage CMOS analog four quadrant multiplier based on flipped voltage followers. 2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512), Vancouver, BC, pp. I-681. 2014. doi:10.1109/ISCAS.2004.1328286.

[42] Saxena V, Baker RJ. Indirect compensation techniques for three-stage fully-differential op-amps. 53rd IEEE International Midwest Symposium on Circuits and Systems, Seattle,WA, pp. 588–591. 2010. doi: 10.1109/MWSCAS.2010.5548896.

[43] Li SC. "A symmetric complementary structure for RF CMOS analog squarer and four-quadrant analog multiplier." *Analog Integrated Circuits and Signal Processing* 23.2: 103-115. 2000

[44] Hasan R, Taha TM, Yakopcic C. On-chip training of memristor based deep neural networks. International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, pp. 3527-3534. 2017. doi:10.1109/IJCNN.2017.7966300

[45] Yu S, Wu Y, and Wong HSP. “Investigating the switching dynamics and multilevel capability of bipolar metal oxide resistive switching memory,” Applied Physics Letters, vol. 98, no. 103514, 2011.

[46] Li C, Hu M, Li Y, et al. Analogue signal and image processing with large memristor crossbars. *Nature Electronics*, *1*(1), 52. 2018.

[47] Graves A and Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5–6): 602–610, July 2005. ISSN 0893-6080. doi: 10.1016/j.neunet.2005.06.042.

[48] Greff K, Srivastava RK, Koutník J, Steunebrink BR and Schmidhuber J, "LSTM: A Search Space Odyssey," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222-2232, Oct. 2017. doi: 10.1109/TNNLS.2016.2582924

[49] Cho K, van Merrienboer B, Gulcehre C, et al (2014) Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv preprint arXiv:1406.1078, 2014. URL http://arxiv.org/abs/1406.1078.