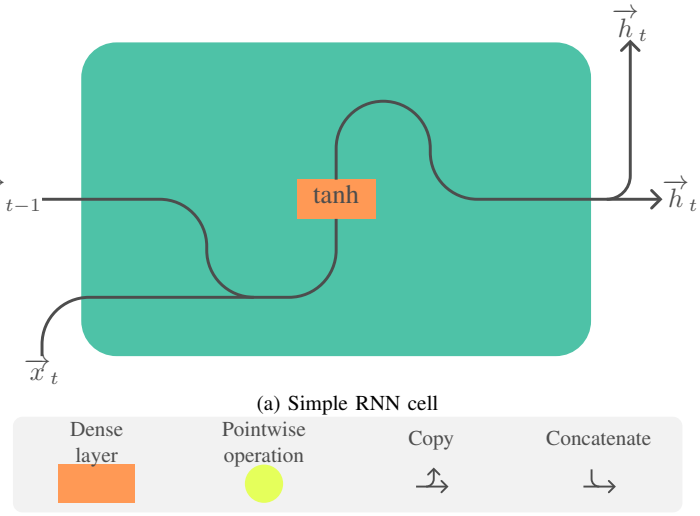
Memristor-based recurrent modules for neural computing

Valentin BARBAZA
Lisbon, Portugal
valentin@barbaza.org

***Abstract*—In this work we propose an analog structure for memristor based recurrent modules targeting neural computing. The system is fully analog and implements a working LSTM circuit block and a work in progress GRU circuit block. Both of those blocks contain memristors to be used as weights in a analog VMM capable circuit. These circuit blocks allow to run very fast computation of RNNs of any size, in a relatively small integrated circuit. As part of the LSTM and GRU blocks, an analog activation function circuit was designed. This specific circuit is capable of reproducing sigmoid and tanh like functions, with similar shapes and the same output ranges. The work also include the implementation of a memory cell used to store an analog value for a short period of time. The LSTM block can be serialized or not with the ability to choose the level of serialization. Serializing the system allows to save onChip area at the cost of execution time. To the author's knwoledge this is the first analog implementation of the behavior of C. elegans using the LSTM block. Such an analog system provides ground for real time implementation of nervous systems.**

***Index Terms*—Memristors, Crossbar array, VMM, Machine Learning, RNN, LSTM, GRU, Analog computing, Analog circuits, Analog activation functions, Tensorflow, Keras, Embedded systems, Neural computing**



(a) Simple RNN cell

(b) Legend

Fig. 1

I. Introduction

AIs (Artificial Intelligences) are one of the main research domain in computer sciences and is used in other scientific domains. One great application of AI, is in embedded systems. Lots of small electronic devices around us could benefit us. **Google** and **Apple** are pushing AI in phones with their in house designed ARM processors having, very advertised, tensor cores. Being able to run low powered AIs is thus one of the biggest computer objective of the deceny. Analog computers are known to be a low powered technology that most importantly give almost instant results. An analog ASIC (Application Specific Integrated Circuit) could then be fabricated for specific embedded applications. This would definitely improve digital technologies in terms of time, and maybe in power consumption as well, these being two prerequisite for embedded system use. For this reason this work aims at creating a functioning software simulation of analog circuit capable of running AI. The thesis will focus on the software simulation, the first step of the work in order to have a fully working chip able to run AI algorithms.

II. State of the art

A. Recurrent Neural Networks (RNN)

RNNs are a family of neural networks that differentiate themselves by having feedback connections. It is often used with sequences of data [1], sometimes, with varying amount of input data. RNNs are used for handwriting recognition and language translation [2].

The feedback connection is caracterized by the hidden state vector ($\overrightarrow{h}_t$) which serves as the output and as part of the input for the next time step, a RNN is defined as (1).

$$\overrightarrow{h}_t = f(\overrightarrow{x}_t, \overrightarrow{h}_{t-1}) \tag{1}$$

The simple version of the RNN is defined by (2).

$$\overrightarrow{h}_t = tanh([\overrightarrow{x}_t, \overrightarrow{h}_{t-1}] \cdot W + \overrightarrow{b}) \tag{2}$$

Where (W , $\overrightarrow{b}$) are the pair of weights matrix and bias vectors. $\overrightarrow{x}_t$ is the input vector and $\overrightarrow{h}_t$ is the hidden state vector.

The graphical representation of the simple RNN cell is shown in Fig. 1a.

B. Long Short Term Memory (LSTM)

LSTMs are a type of RNN used to solve the vanishing gradiant problem [3]. It was improved a few times before becoming the modern LSTM [4]. The LSTM differs from a simple RNN because of its second feedback variable being the cell state ($\overrightarrow{c}_t$).

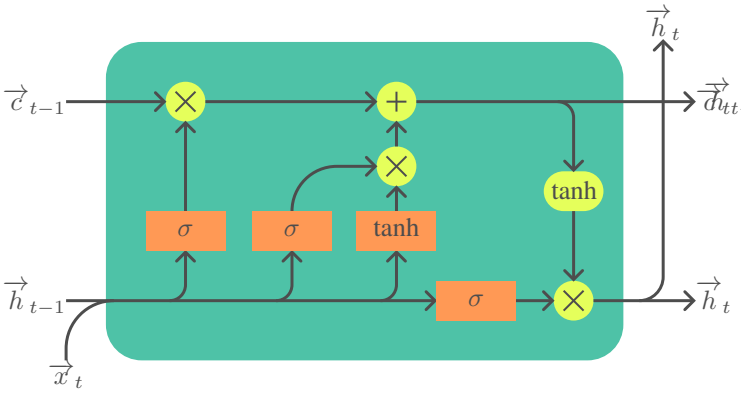


Fig. 2: LSTM cell, adapted from [5]

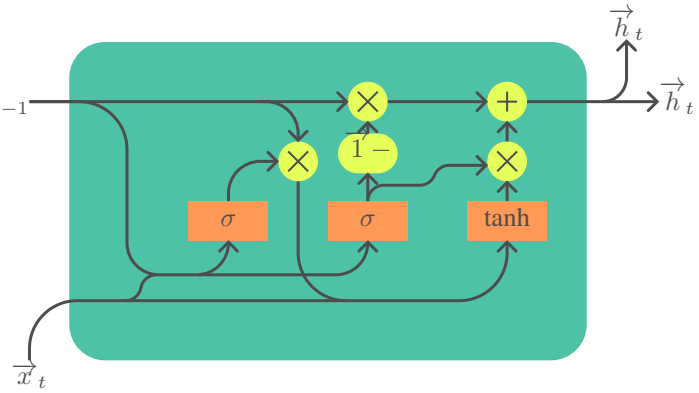


Fig. 3: Encoder GRU cell, legend in Fig. 1b

The LSTM contains four activated gates, that each serve its own purpose. The input gate (3) controls whether the cell state is updated, the forget gate (4) that determines how the current cell state is affected by the old cell state, the output gate (5) that controls how much the hidden state is affected by the cell state. The candidate cell state gate (6) computes the change in the future cell state.

$$\overrightarrow{i}_t = \sigma([\overrightarrow{x}_{t_1}, \overrightarrow{h}_{t-1}] \cdot W_i + \overrightarrow{b}_i) \tag{3}$$

$$\overrightarrow{f}_t = \sigma([\overrightarrow{x}_{t_1}, \overrightarrow{h}_{t-1}] \cdot W_f + \overrightarrow{b}_f) \tag{4}$$

$$\overrightarrow{o}_t = \sigma([\overrightarrow{x}_{t_1}, \overrightarrow{h}_{t-1}] \cdot W_o + \overrightarrow{b}_o) \tag{5}$$

$$\overrightarrow{cc}_t = tanh([\overrightarrow{x}_{t_1}, \overrightarrow{h}_{t-1}] \cdot W_c + \overrightarrow{b}_c) \tag{6}$$

The next part of the LSTM consist of computing the new cell state in the following equation :

$$\overrightarrow{c}_t = \overrightarrow{f}_t \odot \overrightarrow{c}_{t-1} + \overrightarrow{i}_t \odot \overrightarrow{cc}_t \tag{7}$$

The hidden state is then determined using the current cell state (8).

$$\overrightarrow{h}_t = \overrightarrow{o}_t \odot tanh(\overrightarrow{c}_t) \tag{8}$$

Where $(W_i, \overrightarrow{b}_i)$, $(W_f, \overrightarrow{b}_f)$, $(W_o, \overrightarrow{b}_o)$ and $(W_c, \overrightarrow{b}_c)$ are the pair of weights matrixes and bias vectors for the input, forget, output and candidate cell gates respectively. $\overrightarrow{x}_t$ is the input vector and $\overrightarrow{h}_t$ is the hidden state vector.

The graphical representation of an LSTM cell is found in Fig. 2.

C. Gated Recurrent Units (GRU)

GRUs are another type of RNN. It is also known to reduce the effect of the vanishing gradient problem. It was first introduced to improve translation techniques [2].

The GRU is very often compared to the LSTM, being sometimes assimilated as a type of LSTM [6]. Their performance was found to be very similar in most situations [7], making those two types of RNN coexistant in the modern machine learning world.

There are two versions of the GRU, both are found on the internet, they are known as the encoder and decoder version [2]. They were originally designed to encode the message to translate and then decode in the translation. PyTorch only supports the decoder version [8], while the Keras library supports both [9] chosen by changing an argument. This work only uses the encoder version, for that reason, it will be the only one described.

It contains an update gate (9), a reset gate (10), a candidate activation gate (11). The hidden state is then computed (12) using the previous results.

$$\overrightarrow{z}_t = \sigma([\overrightarrow{x}_t, \overrightarrow{h}_{t-1}] \cdot W_z + \overrightarrow{b}_z) \tag{9}$$

$$\overrightarrow{r}_t = \sigma([\overrightarrow{x}_t, \overrightarrow{h}_{t-1}] \cdot W_r + \overrightarrow{b}_r) \tag{10}$$

$$\overrightarrow{ch}_t = tanh(\overrightarrow{x}_t \cdot W_{hx} + (\overrightarrow{r}_t \odot \overrightarrow{h}_{t-1}) \cdot W_{hh} + \overrightarrow{b}_h) \tag{11}$$

$$\overrightarrow{h}_t = (\overrightarrow{1} - \overrightarrow{z}_t) \odot \overrightarrow{h}_{t-1} + \overrightarrow{z}_t \odot \overrightarrow{ch}_t \tag{12}$$

Where $(W_z, \overrightarrow{b}_z)$, $(W_r, \overrightarrow{b}_r)$,$(W_{hx}, W_{hh}, \overrightarrow{b}_h)$ are the weights matrixes and bias vectors for the update, reset and candidate activation gates respectively.

A visual representation of the encoder GRU cell is available in Fig. 3.

D. Memristors

Memristors are the lesser known fourth fundamental passive component of electronics, among resistors, capacitors and inductor. It was first theorized in 1971 as a missing fundamental component in [10]. The name comes from the blend of *memory* and *resistance*. The missing component linking the four fundamental circuit variables, voltage (v), charge (q), current (i) and flux (ϕ). Fig. 4 shows the four fundamental variables are on each side of the square, with the ones on opposite sides being linked by the following equations :

$$d\phi = v \cdot dt \tag{13}$$

$$dq = i \cdot dt \tag{14}$$

Resistors, capacitors and inductors were already very established and well known components, so it was theorized that a fourth device should then exist to physically link flux (ϕ)

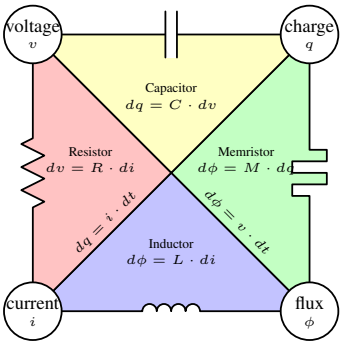


Fig. 4: Fundamental passive components, adapted from [13]

and charge (q). The flux in this case is not a magnetic flux and is defined as such : $d\phi = v \cdot dt \Rightarrow \phi = \int v\, dt$.

The component stayed theoretical until 2008 when it was implemented in a physical device for the first time [11]. It took 37 years to have an actual working device.

An extension to the memristor, reffered to as the memristive device, was theorized in 1976 [12]. The difference between the memristor and the memristive devices is its internal behavior. Memristive devices are commonly referred to as memristors as well.

A memristor is used for its ability to change its internal resistance based on the current that flowed through it.

E. Memristors Crossbar Arrays

Setting memristors in a crossbar array allows to perform analog Vector Matrix Multiplication (VMM), also called Multiply and Accumulate. Fig. 5 shows what a typical crossbar array looks like.

The circuit uses physical properties of electrical systems to perform analog computation. The following part will focus only on the circuit node in Fig. 6.

A voltage is applied on the k^{th} line, and because every column is virtually grounded, the voltage applied to the memristor, with resistance R_i, is V_i. By applying Ohm's law, we know that the current flowing into the memristor (i_k) is bound by the following equation :

$$v_k = R \cdot i_k \Rightarrow i_k = v_k \cdot (\frac{1}{R_k}) = v_k \cdot G_k \tag{15}$$

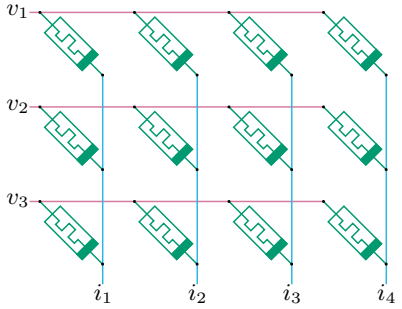


Fig. 5: Crossbar array schematics, inspired from [14]

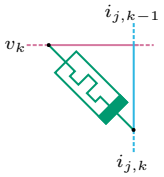


Fig. 6: Memristor crossbar node of the k^{th} line and j^{th} column

With G_k being the conductance of memristor, defined as $G_k = \frac{1}{R_k}$.

This line then joins the column where a current of $i_{j,k-1}$ is flowing, then according to Kirchhoff's current law the resulting current is :

$$i_{j,k} = i_{j,k-1} + i_k = i_{j,k-1} + v_k \cdot G_k \tag{16}$$

Unfolding the equations will give the current at the bottom of the column, for example, the current at the bottom of the first column in Fig. 5 is :

$$i_1 = G_1 \cdot v_1 + G_2 \cdot v_2 + G_3 \cdot v_3 \tag{17}$$

With G_1, G_2 and G_3 being the conductance of the 3 memristors in the first column.

III. Circuits

The system will be working with a v_{dd} of $1.8V$. Such a value was chosen because this is a low power system.

The way values are encoded in the analog system will be descibed here as it serves for the entire thesis. In order for the system to support negative numbers we're going to use a v_{cm} set to $\frac{v_{dd}}{2}$. That means that $v_{cm} = \frac{v_{dd}}{2} = 0.9V$. This v_{cm} will then describe a zero. A step of one was chosen to be $0.1V$ in the analog circuit. TABLE I shows the conversions from a real number to its voltage equivalent.

Since the system cannot reach voltage outside of the operating range with the intended behavior, the voltage is then restricted to $v \in [0, 1.8]$. This means that the range of real value that the systems can handle is $x \in [-9, 9]$.

A. Activation functions

Producing analog activation functions is quite important as using hard sigmoid or hard hyperbolic tangent (tanh) functions impacts the results [15], [16].

The analog activation circuit thus plays an important role in the final result's quality.

The circuit used is the same as the one in [17], and is shown in Fig. 7. The circuit's technology used being different, all the parameters had to be determined empirically to best fit a sigmoid shape. The parameters can be found in TABLE II.

TABLE I: Real/Voltage Conversion Table.

Real value	**Voltage**
0	$0.9V$
1	$1.0V$
x	$v(x) = \frac{x}{10} + v_{cm}$
$real(v) = (v - v_{cm}) \cdot 10$	v

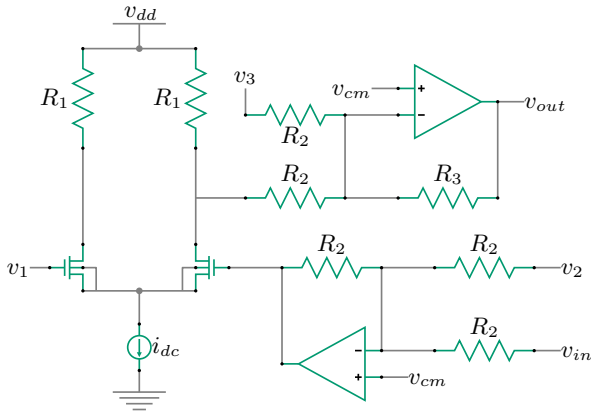


Fig. 7: Activation functions circuit

TABLE II: Circuits parameters

Parameter	Sigmoid	tanh
v_1	$1.1V$	
v_2	$635mV$	
v_3	$0.8V$	$550mV$
i_{dc}	$150uA$	
w	$900nm$	
l	$60nm$	
R_1	$5k\Omega$	
R_2	$10k\Omega$	
R_3	$2k\Omega$	$4k\Omega$

Due to the nature of the functions we want to generate, we will use the same circuit for both a sigmoid and a tanh like functions. The two different functions are generated by changing two parameters. The functions generated are the same shape and only differ by their output range.

Here, w and l are, respectively, the width and length of the two NMOS of the circuit.

The outputed voltage depends on the input passed on. The results obtained by are rather convincing and can be found in Fig. 8. Fig. 8 also shows the Root Mean Square Error (RMSE) of the analog results with the ideal result.

The symbols for the sigmoid and the tanh are separated for better understanding and are available in Fig. 9.

These functions are still a bit different from the original functions (especially for the tanh). However that does not matter too much as the trainning will be happening with the extracted analog functions, all weights will be set in the circuit. This is the reason why such a difference does not matter. As long as the curves have the similar shape, the result will not be drastically affected.

B. Memory cell

The memory cell is a circuit that is able to store an analog value for a limited time. It works using capacitors that have the ability to store a voltage for a short period of time.

The circuit is shown in Fig. 10. The voltage is stored in the capacitor and is kept using CMOS switches.

The CMOS switches found in the circuit have width of $w = 200nm$ and a length of $l = 60nm$.

The cirucit has a two CMOS switches design to avoid voltage leakage through the swicthes. The system has voltage leakage when only one CMOS swicth, and thus leads to a large memory leak. This is due to the high voltage difference between the two sides of the CMOS swicth. Using two CMOS switches allows for this difference to be mitigated (Fig. 11).

The symbol, shown in Fig. 12, for this circuit is designed to show a capacitor because it is its memory mechanism.

C. Voltage-driven crossbar circuit

The crossbar circuit theory has already been explained in the section II. The actual implementation in the circuit is described here. The circuit is the one in Fig. 13. The circuit depends on three parameters :

- n_i : The number of input for our crossbar array (not including bias for a more general circuit).
- n_o : The number of parallel output for our crossbar array.
- n_s : The serial size of our crossbar system.

In this work, the choice was made to use a two memristor per synapse architecture. Using two memristor per synapse doubles the area but doubles the weight range [18] and allows to easily use negavite weights. The output voltage will be centered around v_{cm} and be compliant with the standard set in TABLE I.

This design is the one that is used in [18]. Let's assume that a given memristor has a resistance range of $R \in [R_{min}, R_{max}]$, that means its conductance range is $G \in [G_{min}, G_{max}]$ (with $G_{min} = \frac{1}{R_{max}}$ and $G_{max} = \frac{1}{R_{min}}$). This design works using two opAmp connected to v_{cm} with the positive pin and the negative pin to the output of the crossbar array. Equations (18) to (20) are describing how this architecture works. A simplified version of the double memristors per synapse circuit is also available in Fig. 14.

Using v_k as the voltage for the input line k. The highest opAmp is identified as $opAmp_0$ and the lowest $opAmp_1$.

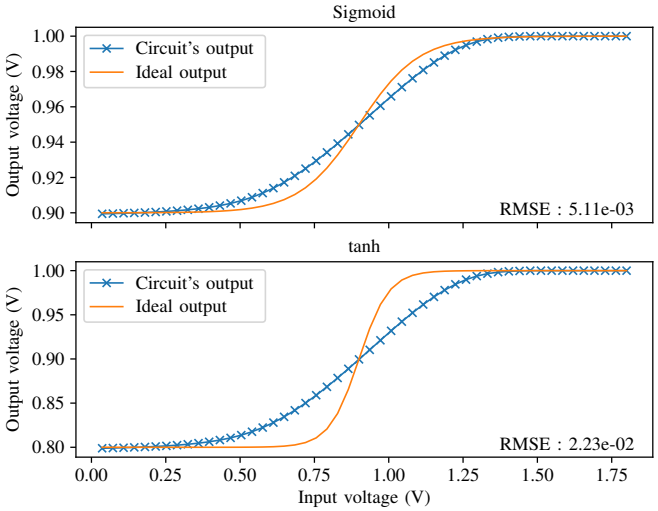


Fig. 8: Input/Output graph of the activation function circuit for both sigmoid and tanh functions

(a) Sigmoid symbol (b) tanh symbol

Fig. 9: Activation functions symbols with the input and output pins on either side depending on the flow of the current for better readability

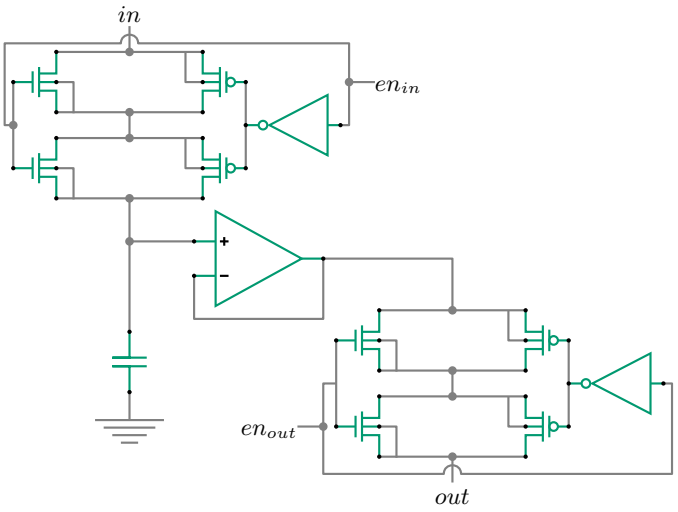


Fig. 10: Memory cell circuit

For the sake of simplicity, the following equations considers the ground to be v_{cm}.

$$v_{opAmp_0} = -R_r \cdot i_+ \tag{18}$$

$$i_{R_f} = i_- + \frac{v_{opAmp_0}}{R_r} = i_- - i_+ \tag{19}$$

$$\begin{aligned} v_{opAmp_1} = v_{out} &= -R_f \cdot (i_- - i_+) = R_f \cdot (i_+ - i_-) \\ &= R_f \cdot \textstyle\sum_{k=0}^{n} (G_{k+} - G_{k-}) \cdot v_k \end{aligned} \tag{20}$$

With $i_+ = \sum_{k=0}^{n} G_{k+} \cdot v_k$ and $i_- = \sum_{k=0}^{n} G_{k-} \cdot v_k$.

This circuit has the option to be serialized with varying degrees. The idea of serializing the circuit came from [17]. Serializing the circuit reduces the number of components required and thus reduces the final onChip area. Serializing the system increases the time it takes to compute the output.

Serializing the system means not computing all values of the output vector at the same time, but instead computing group by group, the groups' size are n_o. The first output group is computed during e_0 and the i^{th} group is computed during e_i. The timing of when the outputs are available is found in Fig. 15. Those flag control the CMOS swicthes present in Fig. 13, the switches control which output group is outputed.

The CMOS switches are here to open the necessary input gates when the output is required, the CMOS switches are controlled as in Fig. 15.

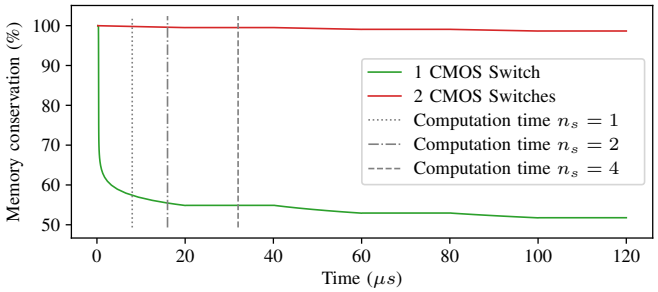


Fig. 11: Memory conservation in a memory cell with 1 CMOS switch vs 2 CMOS swicthes

Fig. 12: Memory cell symbol with the input enable pin (top) and the output enable pin (bottom). The left and right pins are interchangebly the input and ouput for the sake of readability

When the system is used fully in parallel, the CMOS switches are not required can then be removed to lower the final onChip area.

In this work, the analog system will be simulated for the inference of the NN, thus the weights will not have to change during the simulation. Because the weights are represented in the analog circuit by the internal resistances of the memristors, the memristors can be replaced by resistors with a set resistance for the simulation.

The symbol (Fig. 16) defined for the voltage based memristor crossbar array used in this thesis is more compact and helps the readability of the circuits that require a crossbar array. It

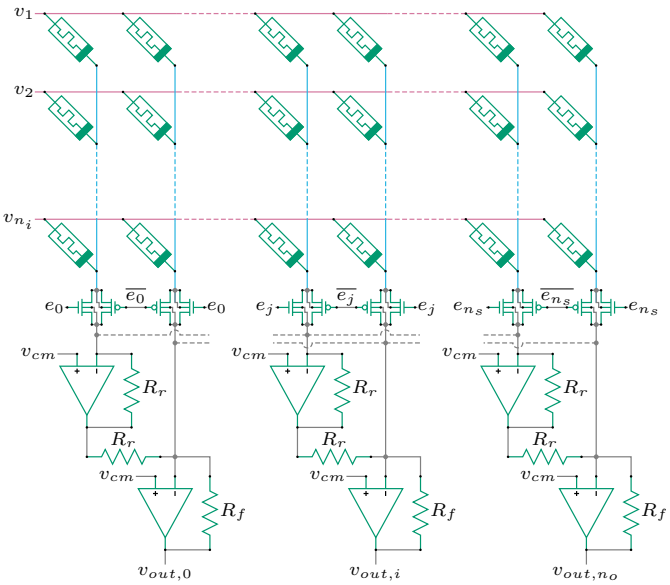


Fig. 13: Circuit of the crossbar array used in the final system (n_i, n_o, n_s)

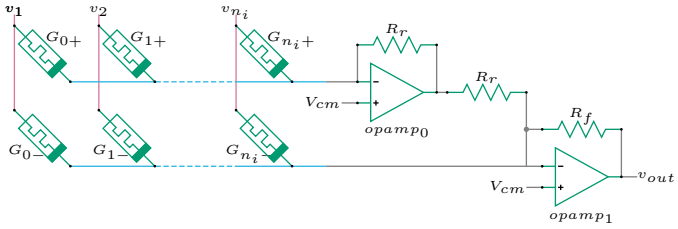


Fig. 14: Simplified circuit of a double memristor per synapse architecture

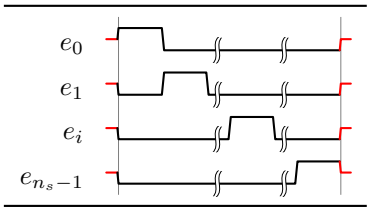


Fig. 15: Enable flags timing for any value of n_s in a single time step

depends on several parameters, the number of inputs (n_i), the number of outputs (n_o) and the serial size (n_s).

D. Verilog-A models

This work uses Verilog-A components to replace component that could not be designed due to lack of time. Those components are the operational amplifier (opAmp) and a voltage multiplier.

1) Operational amplifier: This component is the very famous opAmp. This specific component was not designed for the thesis, it required a current range that did not allow to use ones already made by members of the research group. The choice was then to use a verilog-A model, and then design one if time allows it.

$$V_{out} = \mu \cdot (V_+ - V_-) \tag{21}$$

Equation (21) is the equation used in the verilog-A model. It makes it so it behaves like an ideal opAmp. For this thesis μ has been set to $\mu = 10^5$.

2) Voltage multiplier: This component while far less popular than the latter, is just as useful for our specific use. It allows us to multiply, as its name implies, two voltages. It is used to compute the pointwise multiplications of the LSTM or GRU (Figs. 2 and 3).

In oder for the circuit to be compliant with the real value to voltage conversion (TABLE I), a multiplication needs to be as in (22).

$$v_{out} = 10 \cdot (v_{in_1} - v_{cm}) \cdot (v_{in_2} - v_{cm}) + v_{cm} \tag{22}$$

This is taken care of using in reality two parts, the actual voltage multiplier ((23)) and a non inverting amplifier ((24)).

$$v_{voltMult} = -(v_{in_1} - v_{cm}) \cdot (v_{in_2} - v_{cm}) + v_{cm} \tag{23}$$

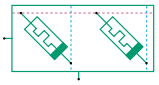
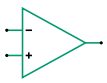


Fig. 16: Symbol used for the crossbar array, the input pin is a bus of size n_i and the output pin is a bus of size n_o



(a) opAmp's symbol

(b) Voltage multiplier's symbol

Fig. 17: Symbols used for the verilog-A components

$$v_{out} = -(v_{voltMult} - v_{cm}) \cdot 10 + v_{cm} \tag{24}$$

Where v_{out} is the output voltage the inverting amplifier, $v_{voltMult}$ is the out voltage of the voltage multiplier itself and v_{in_1} and v_{in_2} are the input voltages.

The (23) is assumed possible because of the actual voltage multiplier's datasheet available at [19].

The symbols for those two components are the ones in Fig. 17.

The opAmp uses its IEEE symbol (Fig. 17a) while the voltage multiplier uses a custom symbol (Fig. 17b).

E. LSTM analog implementation

This section describes the circuit of an LSTM with an input vector of size n_i, a n_h hidden states, a serial size of n_s and n_{ts} time steps. $n_o = n_h/n_s$ is going to be used for future references in this section. In order for the crossbar array to be used n_o must be an integer, in other words, n_s must divide n_h. The circuit used for the LSTM is shown in Fig. 18.

The system is built using the crossbar array from section III-C with ($n_i + n_h + 1$,n_o, n_s) as parameters.

- $\overrightarrow{z_t}$ is the input of the crossbar but not the input of the LSTM. It is defined by $\overrightarrow{z_t} = (\overrightarrow{x_t}, \overrightarrow{h_{t-1}}, \overrightarrow{b})$.
- $e_{j,0}$ and $e_{j,1}$ are two enable flags that respectively represent the first and second half of e_j.
- e_{in} and e_{out} are the flags used to enable the hidden state values to go to the input (feedback connection) or to the output of the circuit.
- e_{next} is the enable flag on in between two time steps.
- R_{amp0} and R_{amp1} are the two resistances used to amplify the output voltage of the voltage multipliers. Their value is set so that $\frac{R_{amp1}}{R_{amp0}} = 10$.

The wires coming into the crossbar are a bus of size $n_i + n_h + 1$ and the output of the crossbar is a bus of size n_o (Fig. 16). This is why everything in the system apart from the crossbar arrays is only shown once in Fig. 18 but in reality those components are present n_o times. Those extra components are needed in order for the parallel channels to work.

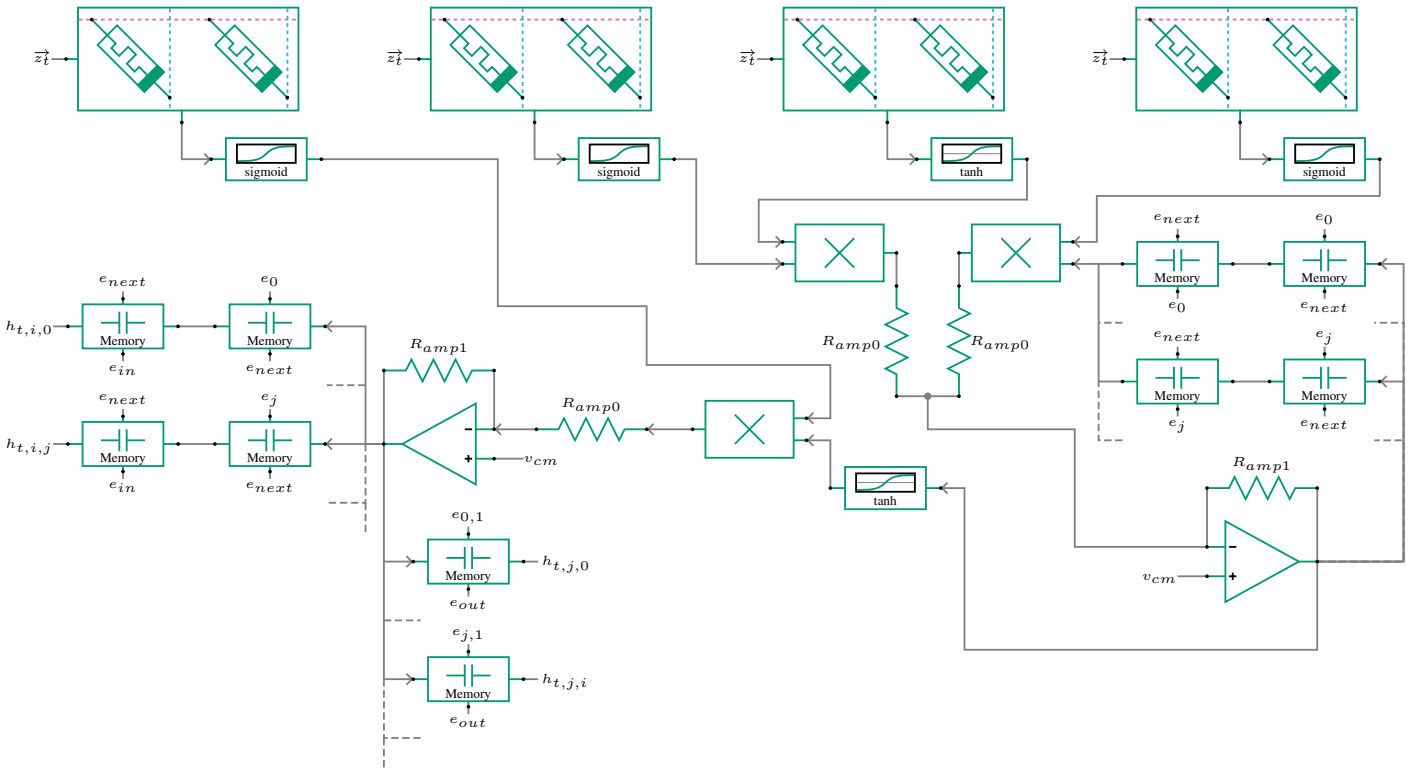


Fig. 18: LSTM circuit

The circuit uses two memory cells for its feedback connections not to overwrite the value of the current stored value by the value of the next time step.

The LSTM circuit can be serialized allowing to reduce the number of pointwise components in the circuit by a factor n_s. However serializing the circuit increases the times it takes to compute the outputs by a factor of n_s.

The symbol of the LSTM circuit can be found in Fig. 19. It depends on several parameters, the number of inputs (n_i), the number of hidden states (n_h), the serial size of the system (n_s) and the number of time steps (n_{ts}).

F. GRU analog implementation

This section describes the circuit of an encoder GRU with an input vector of size n_i, a n_h hidden states and n_{ts} time steps. The decoder GRU could also be implemented in an analog circuit, but the choice was made to focus on the encoder GRU. The GRU is by its nature very similar to the LSTM. For that reason it has a similar circuit to LSTM circuit (Fig. 18). The system is built, once again, using crossbar array with ($n_i + n_h + 1$,n_h, 1) as parameters.

Fig. 19: Symbol used for the LSTM circuit

The GRU equations requires to compute the function defined in (25).

$$f(x) = 1 - x \tag{25}$$

The solution found to do compute the operation consist of using an opAmp as an inverter around V_{inv}, the formula is available in (26).

$$f_v(v) = -(v - v_{inv}) + v_{inv} = 2 \cdot v_{inv} - v \tag{26}$$

With $v_{inv} = v_{cm} + 0.05$.

The GRU circuit cannot be serialized. This is because the candidate hidden state requires a fully computed reset vector to be computed.

IV. Python tools

A. Weights generation

The weigths were trained in python [20] using the tensorflow library [21].

The weights need to be trained. The training is performed using the analog activation functions generated by the circuits in section III-A. The circuit is thus trained as if it were trained directly on the chip, circuit's inaccuracies aside.

The weights are constrained to make sure the voltage is within the active voltage range of the circuit. If the voltage gets out of range, the weights will not be correct as the voltage cannot get out of range in the physical circuit.

Once the weights are trained, they are saved in a file, that will later serve as the basis to generate the netlist.

The code used to generate the weights is available on my github repo [22].

B. Netlist generation

In order to be able to run the simulation with different kinds of NN architecture of varying sizes. The point of the part is thus to explain how the netlist generator tool works.

The netlist is generated using a python script to generate a SPICE netlist. It can generate the netlist for different kinds of circuits. It can generate both LSTM and GRU circuits, with any size of input, time steps or even number of hidden states. When it is possible the serial size can also be selected.

The weights are stored in a single file. This file contains a brief description of the architecture being used. The first index stores this desciption. The following indexes are for the weights of each layer, in the order given in the description. The weights are stored in a python list and are distributed among the different resistances.

All the code used to generate the netlist can be found on my github page [23].

V. Datasets

A. Airline passengers

The first dataset contains a time series of international airline passengers from January 1949 to December 1960, the data is recorded monthly and is in thousands. The dataset then contains twelve years of monthly data so 144 sample points. Although the results of this problem are not necessarely tailored to the system, in the sense that it is not time sensitive information, it is nonetheless a problem that can, in a straightforward manner, demonstrate the predictive capacity of the proposed model. The dataset is graphed in Fig. 20.

This is a regression problem, the role of the LSTM is to predict the number of passenger flying the next month being given previous months' passenger count.

The 144 sample points have been transformed into 142 data points for training. The data is grouped by three, two for the input and one the target output. Two third of the dataset is being used for training and the other third is used for validation.

B. C. elegans

This dataset is far more interesting and complex than the latter. This data set aims to use LSTM to mimic the behavior of real neurons. As explained in [24], Caenorhabditis elegans (C. elegans) are simple organisms that are getting very popular for whole brain organization studies. The point of this problem is to reproduce the nervous system of the C. elegans. This is done using recorded data of the input of 4 neurons and the output of 4 other neurons.

The dataset is great for our study because :

- It comes from a very recent paper from the research group in which this work is also being developed.
- It aims at reproducing the behavior of the brain of a simple organism (C. elegans). In the (very) long term, a full parts of the human brain could be replaced by a very low powered chip.

An example of an input/output sequence is shown in Fig. 23 along with the digital and analog predictions.

The dataset contains a set of fourty set of input/output sequences. Each set of input and output set contains $500ms$ of data with a time step of $0.5ms$, making a thousand points for each neurons. This data is recorded after applying current to the input neurons (PLML2, PLMR, AVBL, AVBR), and monitoring the output of four neurons (DB1, LUAL, PVR, VB1) that are known to have strong activity in response to the four inputs neurons. The simulation is done using a known model of C. elegans connectome (complete overview of the brains connections) [24].

VI. Results

The target for our circuit is to reproduce, as closely as possible the digital prediction.

A. Airline passengers

The problem is solved using an LSTM or GRU with four hidden state connected to a Dense layer with an output size of one. The weights were trained for three hundreds epochs.

The analog results obtained in Fig. 21 are very close to the digital predictions. This is confirmed by the values in

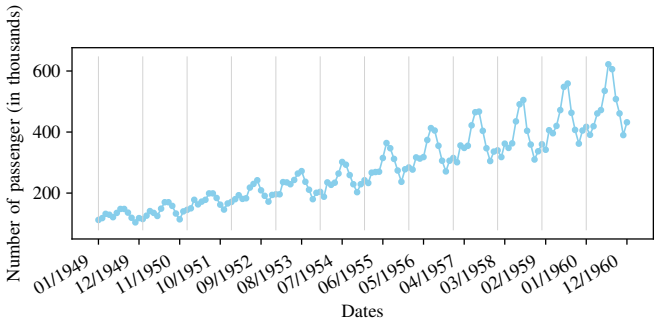


Fig. 20: The airline dataset. The vertical lines represent a full year.

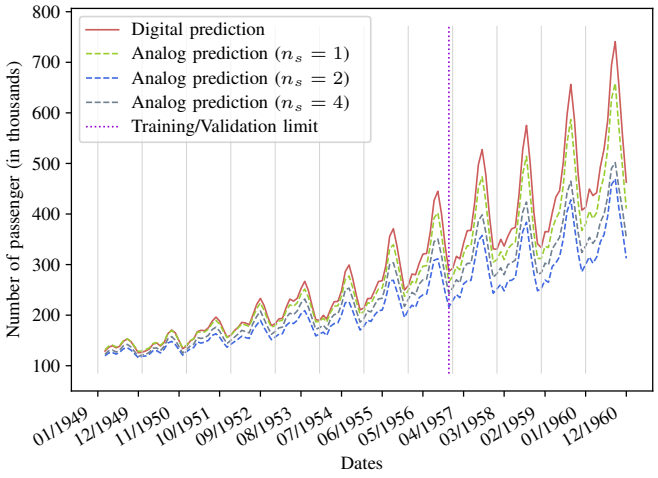


Fig. 21: Analog predictions trained with the analog activation functions.

TABLE III: RMSE of each analog prediction to their associated digital prediction

	Analog prediction			Target
Serial size	$n_s = 1$	$n_s = 2$	$n_s = 4$	**data**
Digital prediction	28.4	92.6	68.5	52.3

TABLE III. The results the closest to the target are the ones generated using a serial size of one ($n_s = 1$). Indeed, its error (28.4) is lower than the error from the digital results (52.3) to the original target values, almost half of it. The other ones are still quite close to the targeted curve, but are still lower. The errors are higher when running the system in serial mode. This higher inaccuracy is probably due to the data staying for longer in the memory cells as the time steps get longer.

The GRU circuit not being fully working produces very off predictions. For this reason, the GRU results are not included.

B. C. elegans

The problem is solved using an LSTM or GRU with eights hidden state connected to a Dense layer with an output size of four. The weights were trained for one thousand epochs.

Fig. 22 combines the digital and analog of all four output neurons. However the analog predictions are quite unstable. The origin of this issue being unknown, the output are thus artificially smoothed out using software. The graph with the smoothed out curves is Fig. 23.

The output neurons predictions are quite good, expect for the unstable analog predictions. The results are very promising.

The other sequences of inputs were ran but connot all be shown here. More details on the C . elegans analog predictions are available in the thesis this paper goes with.

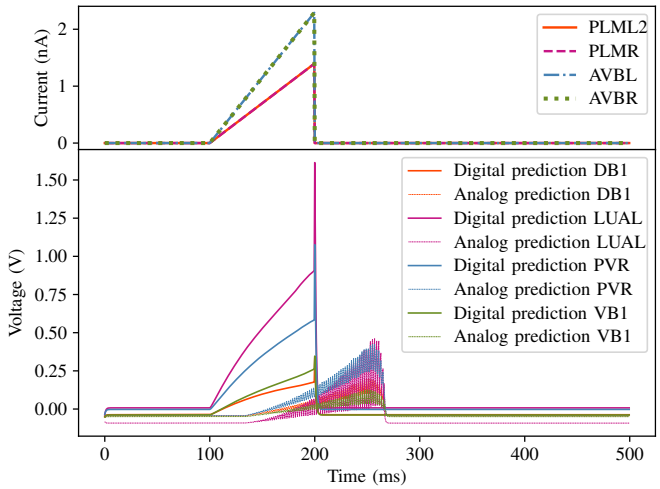


Fig. 22: C. elegans analog responses with their digital counterparts

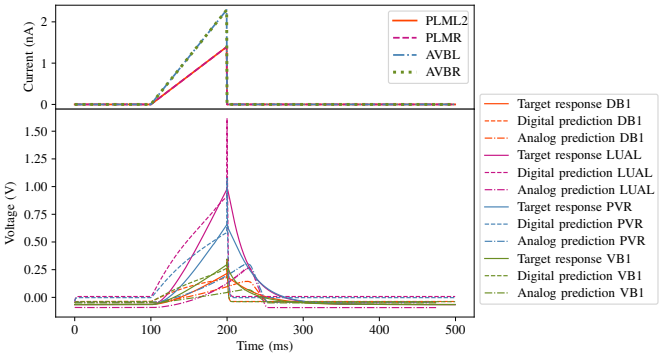


Fig. 23: C. elegans analog responses with their digital counterparts

VII. Conclusion

A. LSTM

The circuit's performance has been evaluated in the previous chapter. The results highly depend on the parameters at play. For example the complexity of the dataset used, out of the two used in the thesis, the airline dataset is the simple dataset while the C. elegans dataset is the more complicated datset.

The best performance is the with the airline dataset, using the circuit with a serial size of one ($n_s = 1$) (TABLE III). While the results are not quite right, it is assumed that this issue will be elevated once inSitu training is implemented.

More complex datsets like C. elegans, the issue is getting more present. Indeed, the dataset feeds to the circuit four inputs across a thousand time steps and outputs just as much data. Any inaccuracy is scaled up.

With a simple input sequence, such as sequence 5 (Fig. 23), the resulting predictions are quite good and on par with what is expected. The predictions are a bit late and it is not clear what is causing this specific issue. A theory is that the memory cells deteriorate its stored value by a very small amount every time step, thus the stored value is affected a lot after a large amount of time steps.

More complex input sequences produce even more inaccuracies. Indeed, when working with sequence 15, not shown in this paper, the inaccuracies explode and gives out a barely usable prediction. The curve generated does not fit its digital counterpart anymore.

The results obtained demonstrate that the LSTM circuit block can be run with low error when dealing with simple inputs. The circuit is ready for a full simulation, meaning also simulating training with the analog circuit.

B. GRU

The GRU circuit, despite not having result in this paper, is still a work in progress. The outputed predictions are scaled down for a still unknown reason, but show the right output shape. Once those issues are dealt with and have been fixed, the GRU circuit will be ready for an analog training as well.

C. Execution time

The execution time is only interresting when dealing with time sensistive data, like with the C. elegans problem. The problem gets input every $500\mu s$, however, the circuit takes at most $65\mu s$ to compute when using eight hidden states and a serial size of eight. This execution time would even allow to take in more frequent inputs, like the $100\mu s$ sequences dealt with in [24].

REFERENCES

[1] W. R. Moskolaï, W. Abdou, A. Dipanda, and Kolyang, "Application of deep learning architectures for satellite image time series prediction: A review," *Remote Sensing*, vol. 13, no. 23, p. 4822, Nov. 2021. [Online]. Available: https://doi.org/10.3390/rs13234822

[2] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014. [Online]. Available: https://arxiv.org/abs/1406.1078

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[4] F. Gers, "Learning to forget: continual prediction with LSTM," in *9th International Conference on Artificial Neural Networks: ICANN '99*. IEE, 1999. [Online]. Available: https://doi.org/10.1049/cp:19991218

[5] Jul 2023. [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory

[6] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017. [Online]. Available: https://doi.org/10.1109/tnnls.2016.2582924

[7] N. Gruber and A. Jockisch, "Are GRU cells more specific and LSTM cells more sensitive in motive classification of text?" *Frontiers in Artificial Intelligence*, vol. 3, Jun. 2020. [Online]. Available: https://doi.org/10.3389/frai.2020.00040

[8] [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.GRU

[9] [Online]. Available: https://keras.io/api/layers/recurrent_layers/gru/

[10] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971. [Online]. Available: https://doi.org/10.1109/tct.1971.1083337

[11] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, May 2008. [Online]. Available: https://doi.org/10.1038/nature06932

[12] L. Chua and S. M. Kang, "Memristive devices and systems," *Proceedings of the IEEE*, vol. 64, no. 2, pp. 209–223, 1976.

[13] Oct 2023. [Online]. Available: https://en.wikipedia.org/wiki/Memristor

[14] D. Joksas and A. Mehonic, "badcrossbar: A python tool for computing and plotting currents and voltages in passive crossbar arrays," *SoftwareX*, vol. 12, p. 100617, Jul. 2020. [Online]. Available: https://doi.org/10.1016/j.softx.2020.100617

[15] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," in *2018 Chinese Control And Decision Conference (CCDC)*. IEEE, Jun. 2018. [Online]. Available: https://doi.org/10.1109/ccdc.2018.8407425

[16] R. Maksutov, "Deep study of a not very deep neural network. part 2: Activation functions," Jul 2018. [Online]. Available: https://towardsdatascience.com/deep-study-of-a-not-very-deep-neural-network-part-2-activation-functions-fd9bd8d406fc

[17] K. Adam, K. Smagulova, and A. James, "Generalised analog LSTMs recurrent modules for neural computing," *Frontiers in Computational Neuroscience*, vol. 15, Sep. 2021. [Online]. Available: https://doi.org/10.3389/fncom.2021.705050

[18] R. Hasan, T. M. Taha, and C. Yakopcic, "On-chip training of memristor crossbar based multi-layer neural networks," *Microelectronics Journal*, vol. 66, pp. 31–40, Aug. 2017. [Online]. Available: https://doi.org/10.1016/j.mejo.2017.05.005

[19] Feb 2011. [Online]. Available: https://www.analog.com/en/products/ad734.html

[20] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[21] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[22] V. Barbaza, "LSTM weights generator." [Online]. Available: https://github.com/bicheTortue/LSTM-weights-generator

[23] ——, "Spice LSTM netlist generator code." [Online]. Available: https://github.com/bicheTortue/memristor-LSTM-generator

[24] R. Barbulescu, G. Mestre, A. L. Oliveira, and L. M. Silveira, "Learning the dynamics of realistic models of c. elegans nervous system with recurrent neural networks," *Scientific Reports*, vol. 13, no. 1, Jan. 2023. [Online]. Available: https://doi.org/10.1038/s41598-022-25421-w