Project A: Neural network simulation over Y–Flash memristor devices

Authors: Sahar Carmel 305554453¸Amir Saad 305393464
Superviser: Loai Danial

December 24, 2019

Abstract

In this project we present a python module based on pytorch ML module that presents a neural network architecture based on Y-Flash memristive device physical features. The comparison we present is based on the differences between state of the art neural networks over traditional benchmarks databases vs memristor based neural networks.

Contents

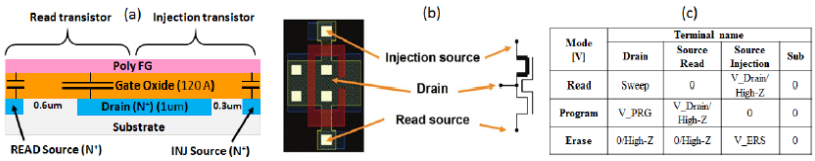
List of Figures

1 Background and motivation

Artificial neural networks (ANN) became a common solution for a wide variety of problems in many fields, such as control and pattern recognition. Many ANN solutions reached a hardware implementation phase, either commercial or with prototypes, aiming to accelerate its performance. Recent work has shown that hardware implementation, utilizing nanoscale devices, may increase the network performance dramatically, leaving far behind their digital and biological counterparts, and approaching the energy efficiency of the human brain. The background of these advantages is the fact that in analog circuits, the vector-matrix multiplication, the key operation of any neuromorphic network, is implemented on the physical level. The key component of such mixed-signal neuromorphic networks is a device with tunable conductance, essentially an analog nonvolatile memory cell, mimicking the biological synapse. There have been significant recent advances in the development of alternative nanoscale nonvolatile memory devices, such as phase change, ferroelectric, and magnetic memories. In particular, these emerging devices have already been used to demonstrate small neuromorphic networks. However, their fabrication technology is still in much need for improvement and not ready yet for the large-scale integration, which is necessary for practically valuable neuromorphic networks. This project investigates a network prototype based on mature technology of nonvolatile floating-gate memory cells.

2 Goals and project requirements

1. Simulating the non-volatile memristive device with virtousuo.
2. Dicing and Packaging at Towerjazz facilities.
3. Simulating state of the art neural networks on MNIST data base for comparison.
4. Using the last network now using Manhattan rule weights update algorithm and comparing to the state of the art architecture.
5. Using the last network now dividing the weights of the network to positive and negative weights. Comparing the results to the SOTA performance.
6. Using the last network now limiting weights to physical constraints . Comparing the results to the SOTA performance.
7. Using the last network now using only program operations without clear. Comparing the results to the SOTA performance.
8. Simulating a CNN network using the FF layers with the physical features developed above.



Mode [V]	Terminal name			
	Drain	Source Read	Source Injection	Sub
Read	Sweep	0	V_Drain/ High-Z	0
Program	V_PRG	V_Drain/ High-Z	0	0
Erase	0/High-Z	0/High-Z	V_ERS	0

Figure 1: Y-Flash NVM device

3 Alternative solutions

Today there are many solutions to engineering problems involving neural networks. Most networks are programmatically implemented, which means that artificial networks are made on non-dedicated hardware. Many companies in the market, including Google, Nvidia and Intel, realize that there is a limit to the performance of artificial networks and are looking for dedicated solutions. In recent years, Nvidia and Google have been developing and marketing dedicated hardware components for learning and implementing networks by creating optimal components for these problems. Nevertheless, these hardware components are still limited in performance since the transfer of information between the various components, and their memory storage is still artificially implemented. Therefore, a solution as we propose: a purely analog neuronal network is expected to provide significantly higher capabilities than artificial networks.

4 Algorithm description

4.1 Y-Flash Memristor

The Y-Flash memristor device is comprised of two NMOS transistors with a common Floating gate (FG). The control of the FG is via the capacitance between the FG to the common drain. As we can see in Fig-1

When a positive voltage is applied on the drain, part of it is applied on the FG due to Miller capacitance. If the voltage goes above V_{th} the cell starts to saturate. When higher voltage is applied on the drain, hot electrons are created at the drain junction, part of those are injected into the FG which results in raise of the threshold voltage. The mentioned process is the programming process and this kind of action changes the characteristics of the memristor. In order to erase the programming, high voltage is applied on the source while the drain and the bulk are connected to ground, with this setup there are no current inside the device. Hot holes are created in the junction, the hot holes are injected into the FG which lowers the threshold voltage of the device. The program and erase sequence demands a numerous iterations in order to achieve desired threshold voltages as we can see in Fig-2

As we can see in order to achieve certain resistance numerous iterations needed to be applied.

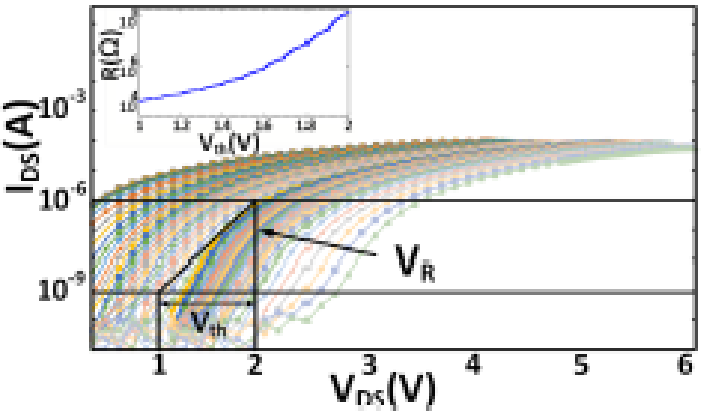


Figure 2: V_{th} vs program iterations

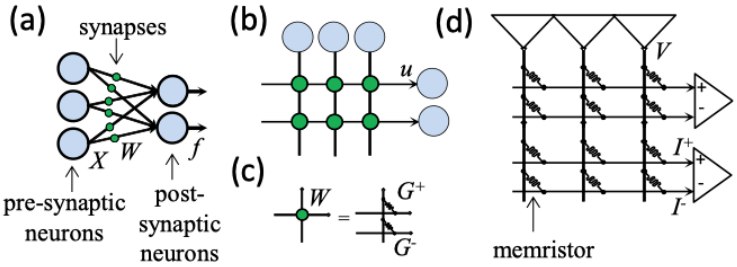


Fig. 1. Feedforward artificial neural network: (a) Abstracted graph representation of one layer with three input and two output neurons, (b) its crossbar circuit mapping, and (c, d) memristor-based crossbar implementation.

Figure 3: Analog neural network architecture

4.2 Manhattan Rule

Neuron networks consist of layered perceptron networks as seen in Fig-3 a.

As can be seen, each neuron is represented by the sum of synapses attached to it and transmits the sum through a nonlinear function defined when used. The transition to analog use in neuronal servers is depicted in the Fig-3 b. Each node indicates a synapse and weights are represented by resistors and outputs sum the currents into an op-amp followed by a non-linear function. Because the weights of synapses can be negative, the analog network size is multiplied by the amount of synapses since there is one that represents a positive weight and one that represents a negative weight. And so the weight of each synapse is defined by

$$W_{ij} = G_{ij}^{+} - G_{ij}^{-}$$

And like so we can create artificially negative weights. Fig- 3c shows the complete analog neuron network architecture where the voltages V are the network inputs and the op-amp outputs the network classification.
As part of the network learning process, the weight of synapses are modified in order to improve the network's performance. In order to create a convenient weight change platform, we will use Meristors on Y-Flash NVM Memristor technology. Because many changes are required in each learning iteration of the network and each component is individually required to change the resistor resistance, this rule proposes to change the resistors in only two directions in one intensity at a time.

$$\Delta W_{ij}^{M} = sgn\left[\Delta W_{ij}\right]$$

In other words, the change in the weight of the synapse is related only to the gradient sign. We will investigate the impact of this rule on the process of training.

4.3 Y-Flash Memristor array

In this section, we will describe the network working method described in the previous section. First, we note that because we do not work with resistors but with meristors, it is necessary to work under a working voltage greater than 0 since meristors are transistors at their base which one can modify his gate voltage and thus change the current transferred through them. And so if we work with voltages that are smaller than the threshold voltage we will not receive any current. This limitation dictates a way in which the input voltages vary around the working voltage of the transistors. Therefore, a training round and network inference test will look like this:

1. Apply working voltage to the network and store network bias output.
2. Convert the network inputs to voltages around the working voltage and apply to net.
3. Subtract the bias output from the network output, this the output of the network.
4. Modify the network's weights by Manhattan rule.
5. Repeat 1-4 up until convergence.

After the learning process has completed in order to use the network in it's inference mode, steps 1-3 are needed in order to get the network's outputs.
Another characteristics of out device is her physical dimension. Since the network is fabricated using memristor devices on silicon, the network dimension is dictated from the technology constraints on the number of devices that can be fabricated on a single chip. A sample of the chip we worked with can be seen in Fig-4

Figure 4: The device e after packing and dicing

We can see that the device has 20 legs that are connected to the read and write lines of the device. This device has 8*12 grid of memristors which translates into a 4*6 grid neural network. A scheme of the device can be seen in Fig-5

5 Software implementation

6 Results

7 Problems and solutions

During our work, we encountered complex difficulties, due in part to the fact that the project involves physical characteristics along with programming characteristics. The main challenge we are faced with is simulating as close as possible to the reality of the Memristor device described above and casting its physical properties into the neural networks. On this issue, we were required to spend most of our time because the open source community hardly touche these areas and the work done on the simulation of physical devices is inaccessible to use. Also, because we used neural network libraries (pytorch) that are accustomed to a standard network structure, the integration of changes into the basic components of the network was an obstacle to the task accomplishment. Among other things, we needed to modify and adapt basic elements such as neurons, synapses, activation functions, and behavior in order to accomplish true simulation.
Another difficulty rised when we started training the network on databases. Because the network we were asked to train was composed of physical components in the form of meristors, we were required to work on device sets with a limited amount of components. As we approached the problem with a low-value array, we found that the basket of problems that could be solved given that array was diluted and no significant conclusions could be drawn about the use of memory-based networks. Also, unlike neural networks where each neuron can be accessed individually, we had to deal with a memory system programming and deletion, which means that we can erase on rows only and not individually. Also, a

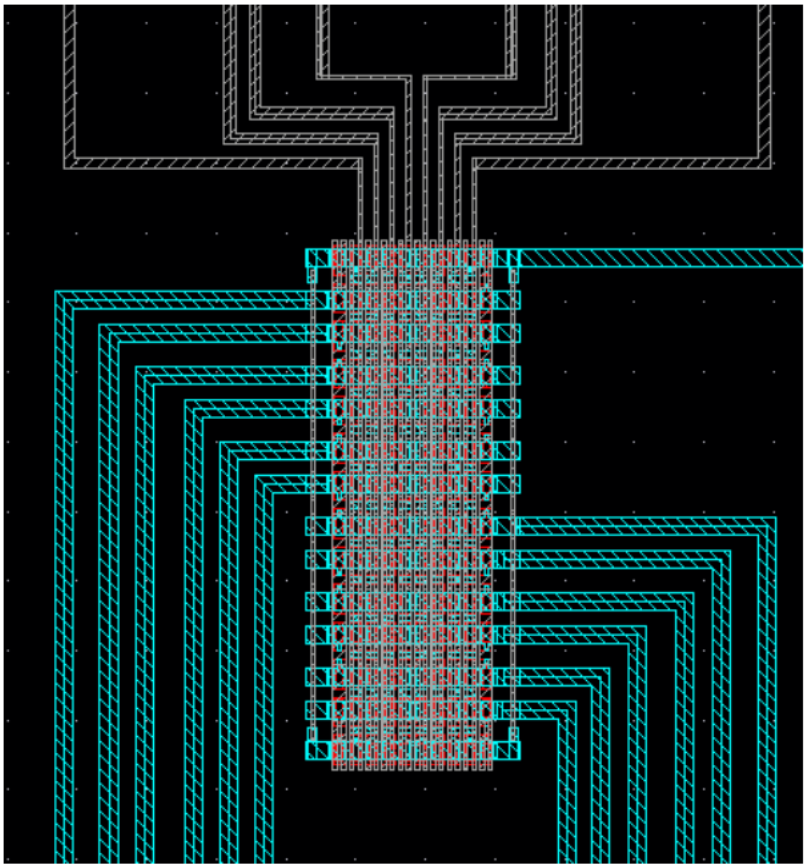


Figure 5: Memristor array scheme. In blue are the read lines, brown are program lines.

problem that arises from working with memory components is the volatility of the information stored therein, naturally because the information stored in the memory components is based on the principle of an electron floating gate and is not hermetically stored and thus the memory component voltage.
Regarding the physical nature of the device, as opposed to neuronal networks in which the synapse weight value is changed by changing the numeric value of the memory stored weight, the physical component programming is required to do voltage pulses on the device. This creates a situation in which the backpropagation and weighting stage requires additional algorithms to improve network performance.

8 Summary and Conclusions

To summrize, we have shown that using flash-based memory devices to fabricate and train neural networks is feasible, when adding the physical restrictions we found that network performance decreases when tested against networks without memory devices. For the complete simulation in which we developed a layer of meristor and quantization of the input image, we obtained that the convergence time of the grid increased 5-fold relative to networks without the properties of the meristor. We should note that the optimization of the networks in the Python certainly does not take into account the fact that for the same network we received a double number of parameters, Since our net weights are represented by positive and negative weights as explained earlier. In addition, we estimate that the increase in training time is apparently a programming problem since in the final realization of the system we intend to use physical devices and not the programmatic realization of the devices. Also, if we want to improve network performance, we can further optimize the network's software implementation by taking advantage of the fact that the positive and negative weights of the network can be represented as two separate networks and by parallel optimization of the network implementation to improve its performance.
Another thing to emphasize is the low network's ability to determine accuracy over time. As you can see in the graph- [fig: compare-success-percentages] The accuracy of the net is having difficult to converge because, unlike the classical net where the step size is getting smaller with each epoch, the memristor net is getting small step size every epoch, but due to the use of the Manhattan rule the gradient step size is constant. We assume that if we are able to physically reduce the size of the pulse we can have a better convergence of the network. Alternatively, one can start the network training with a sequence of several pulses and decrease the number of pulses in the same way that the lr decreases.

9 Appendix

10 Bibliography