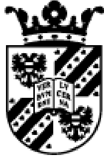

university of groningen / faculty of science and engineering

Simulating Prescribed Error Sensitivity learning using memristors.

Bachelor's Project Thesis

Arseniy Nikonov, a.nikonov@student.rug.nl,
Supervisors: Dr. Jelmer Borst & Prof. Dr. Niels Taatgen

Abstract: Memristors are devices that are part of an electrical circuit and that act as resistors and can control the flow of current through the system. By controlling the polarity and magnitude of a voltage we can set the device into multiple resistance states. It was theorized that these devices can be used for neuromorphic computations. Artificial neural networks are collection of connected nodes. This connections represent synapses of biological brains. Memristor properties can be used to emulate synapses. In this research, the simulation of prescribed error sensitivity(PES) learning was built using memristive devices as connectors between the neurons. Using memristors two different networks were built and compared to the PES network without the use of memristors. In the first network, memristors were initialized in the highest resistance state while in the second the initial resistances were randomized between certain values. Both networks were able to learn the identity function but were slower than the non-memristive simulation. For the second experiment, we investigated which values of the initial resistance the simulation can work successfully and reached the conclusion that the minimal starting resistance should be above 10^6 Ohm. From this experiment we conclude that memristors can be used effectively to emulate synapses.

1 Introduction

Most of the modern computers are based on the von Neumann's (Von Neumann, 1981) architecture and even though it is extremely reliable for some task it consumes orders of magnitude more space and energy than some biological systems like human brain. Growth in the computational power was largely fueled by continued shrinking of transistors controlling the flow of information in the hardware. Nonetheless this trend is challenged by technological limits. On top of the technological limitation contemporary Von Neumann's paradigm faces difficulties in scalability and power consumption for many intelligent task such as processing non-structured data like images. New computing paradigm such as neuromorphic computing, the concept of designing a system inspired by biological systems(Mead, 1990). One of the feature of this approach is storing and processing information at the same location(Gamrat, 2010). The key challenges of neuromorphic systems are matching the abilities and energy consumption of the human brain.
The focus of the paper is the process of learning in the human brain. This process relies on changing the strength of the connections between neurons, known as synaptic weights. This behavior can be emulated by memristors. Memristors are devices that were first theorized in 1973 by Leon Chua(Chua, 1971). It was built for the first time in 2008 by HP labs (Strukov et al., 2008). Memristor devices were defined by Leon Chua(Chua, 2019) as:"Any 2-terminal electronic device devoid of internal power source and which is capable of switching between two resistances upon application of an appropriate voltage or current signal, and whose resistance state at any instant of time can be sensed by applying a relatively much smaller sensing signal, is a memristor, defined either by the ideal memristor equation, or by one of its unfolded siblings". According to Cai and Chua memristors are almost ideal electronic analogue of the synapses (Cai and Tetzlaff, 2014; Chua, 2013). There are at least two way to use memristors for neuromorphic comput-

ing: use them as a weights storage which allows us to use conventional techniques such as back-propagation rule or use them to directly implement learning rules for example Hebbian learning that would only depend on firing time and pre and post synaptic activity (Sheridan and Lu, 2014). In the current research the former approach was used since learning based on the error feedback was implemented.

Memristors are part of the circuit elements and they are similar to resistors in a manner that when they are added to the circuit the control the flow of the current thorough the system. In this research we used devices that changed their resistance based on the voltage passed through them.(Goossens et al., 2018) By applying negative voltage pulses device can be set to a higher resistance state while applying positive voltage pulses would set it into a lower resistance state. One of the main features of these devices are their energy efficiency, non-volatile memory and the ability to set them into multiple resistance state. All of this properties fit the paradigm of neuromorphic computing and theoretically should allow for be beneficial for building efficient biologically similar networks.

One of the conventional methods of emulating biological system is artificial neural networks(ANN). Normally ANN is a collection of connected units(nodes). Nodes represent the neurons in biological brain while connections are used to transmit information and represent synapses. Commonly connection represent their strength and weight and nodes transform input into output using one of the functions.

In 2003 the Neural Engineering Framework (NEF) was proposed by Eliasmith and Anderson(Eliasmith and Anderson, 2003). This approach allows us to build neural networks based on the single neuron models. The neural engineering framework proposes three principles that enables the construction of a model: representation, transformation and dynamics. We will discuss them in more details later.

The NEF typically learns its decoders offline but there are a number of biologically plausible rules to learn connection weights online. One of these rules is the Prescribed Error Sensitivity (Bekolay et al., 2013) which learns function by minimizing external error.

$$\Delta d = ke\mathbf{a} \tag{1.1}$$

where $k>0$ is the learning rate, $\mathbf{d}$ is the decoding vector, e is an error between network output and intended output and $\mathbf{a}$ is the rate of activity of each neuron.

The goal of this research project was to simulate a learning network that incorporates memristive devices, to estimate if it is plausible to construct such a network with real devices and to develop efficient methods of training the network. On top of that we can compare how well the network based on memristive devices can work compared to the use of PES rule on the conventional architecture.

As mentioned earlier for this research we used properties of memristive device described by (Goossens et al., 2018). This memristive device can be set to multiple resistance levels by applying positive or negative voltages. Positive voltage pulse results in lower resistance states while negative pulse result in higher resistance states. We can use this properties to simulate PES learning based on the properties of this memristive device.

1.1 NEF

NEF depends on three principles that i have mentioned before. In this section i would like to go into details of these principles.

1.2 Representation

Population of neurons used to encode the information using vectors of real numbers. By injecting specific amounts of current into single neuron we can cause it to spike. How likely is this spiking depends on the neurons tuning curve: function that describes how likely the neuron going to spike in respond to a given input. Examples of tuning curves can be seen in the Figure 1.1 Important property of tuning curves is the fact that it can be determined for any type of neuron and therefore encoding is not dependent on any particular neuronal model(Bekolay et al., 2014). In the decoding process spikes are filtered, then summed together with weights and result in the output. Decoders themselves are precomputed and most of the time do not depends on the input. In this research we would like to replace this precomputed decoders first with the variant of PES learning and later with the variant

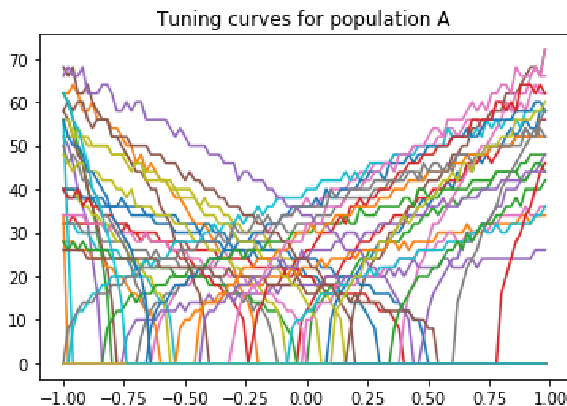

Figure 1.1: A graph of a neural response as a function of stimuli

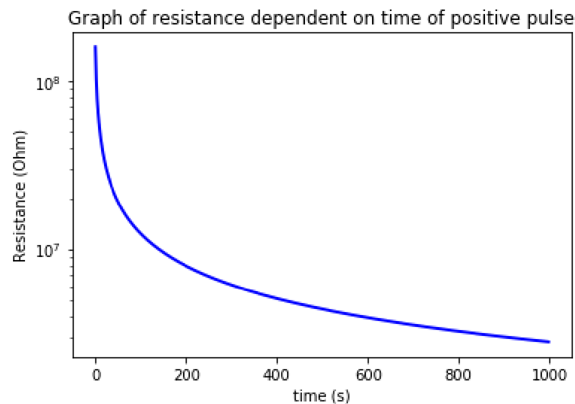

Figure 2.1: Graph of the resistance based on the time of positive pulses applied to the device

of PES learning that would use properties of memristors.

1.3 Transformation

In biological systems neurons communicate using synapses. When neuron fires it causes some amount of current to be transported to the post-synaptic neuron using release of neurotransmitters. This connection is simulated as connection weights between neurons that represents the strength of the connection.In order to compute transformation function between two population of neuron we use the weight matrix. This connection weights are defined by the product of the decoding weights in the first population and the encoding weights in the second population. This decoding weights are the weights that we would like to learn using PES learning.

1.4 Dynamics

Dynamics part of NEF are not that relevant for this research but nonetheless this part of NEF is used when simple feed-forward activity is not enough and recurrent connections are needed. According to the Bekolay equations for such a model can be analysed and designed using the methods of control theory.

2 Methods

2.1 Simulating a memristor

We used the same data for the device B from Investigation of Interface Memristive Devices on Nb doped SrTiO3 for Hardware Based Neural Networks For the first experiment all memirstors were set into the highest possible resistance state - $2.5 * 10^8 \Omega$. For this research we only used positive bias voltage pulses that could only lower resistance of the device. Following equation was used to model the positive bias voltage pulses.

$$R(n, V) = R_0 + R_1 n^{(a+bV)} \quad (2.1)$$

where $a = -0.128, b = -0.522, R_1 = 2.5*10^8, R_0 = 100, n$-the pulse number and V - the input voltage. This equation can be solved with respect to n,

$$n(R, V) = (\frac{R - R_0}{R_1})^{\frac{1}{a+bV}} \quad (2.2)$$

Using equations 2.1 and 2.2 we can compute pulse number n based on the current resistance and update resistance to (n+1) state. Graph of the resistance depending on the number of 1 second pulses can be seen in Figure 2.1
The negative bias voltage pulses were ignored in this research partially due to the lack of data about memristor behavior and due to the simulation network design for which they were not required.

2.2 Description of the network

Network that was used for this project consisted of two ensembles of neurons with 50 neurons in the first ensemble and 40 neuron in the second ensemble. As an input sin(x) was used. Each neuron from ensemble A is connected to an ensemble B and there connection can be characterized be weight matrix of 50 by 40. The goal of the network is to learn some connection function between ensembles A and B. For the first experiment we used the simplest possible function - identity function x=x so in a sense we were trying to learn simple communication channel that would pass value x from one ensemble to another. This network was used first for the implementation of PES learning and later for the implementation of PES learning using memristors properties. For the latter memristors acted as a connections and nodes receive incoming current and computed connection weights based on the current. As an input sin(x) function was used for all the experiments. Later we tried to apply various other functions to see if memristive network would still produce adequate results.
The models are available for download at
`https://github.com/ArseniyNikonov/Memristor_PES`

2.3 PES learning

For the first part we wanted to learn decoders value using PES learning rule instead of the predefined formula that was originally used in NEF. To do that we used the equation 1.1. After each epoch we computed the error and updated decoders using formula 1.1. This is shown in detail in algorithm 2.1.
Simulation was run for 30 sec with an update step of 0.001s. Data from this simulation was used as a control group to estimate the quality of memristor simulation.

2.4 Memristor PES learning

Each decoder was simulated as two memristive device, one of each was responsible for positive value while the other was used for the negative values. Decoders value was inversely proportional to the resistance of the device and was computed by the following formula:

$$d = (\frac{R_0}{R_{1+}} - \frac{R_0}{R_{1-}}) * m \tag{2.3}$$

Where $R_0 = 100$ is minimal resistance, R_{1+} is the resistance of "positive" memristor and R_{1-} is the resistance of "negative" memristor, $m = 100000$ is the value that decoder takes if memristor is in its minimal resistance state.
The weight matrix of the connection was calculated as a product of the decoders of ensemble A(simulated by memristors) and encoders of ensemble B.
During this simulation we used slightly different approach compared to PES learning. Just as in PES learning every epoch we computed the overall error but know instead of directly updating decoders value we change the value of memristor corresponding to the decoder. This value was updated in the following way: first the error was calculated, if the error was positive we applied 1s positive pulse to the "positive" memristor of the neurons that fired during that epoch and by doing that we lowered the resistance of the device and subsequently raised the decoder weight. If the value was negative we applied the same procedure but for the "negative" memristor. This is shown in detail in Algorithm 2.2

Algorithm 2.1 Training algorithm of PES learning network

```
Initialize encoders_A at 0.0
weights ⇐ encoders_A · decoders_B
while t < t_0 do
    input_A ⇐ getInput()
    spikes_A ⇐ run_neurons(input_A)
    for i in spikes_A do
        input_B[i] ⇐ calculateInput(weights)
    end for
    spikes_B ⇐ run_neurons(input_B)
    output ⇐ calculateOutput(spikes_B)
    error ⇐ input -output
    weights ⇐ updateWeights(weights, error)
end while
```

2.5 Memristor PES learning with random initialization

During this experiment we kept the previous design with one difference: instead of initializing the resis-

Algorithm 2.2 Training algorithm for memristor PES learning network

```
Initialize mem_plus and mem_minus
Initialize encoders_A at 0.0
weights ⇐ encoders_A · decoders_B
while t < t_0 do
    input_A ⇐ getInput()
    spikes_A ⇐ run_neurons(input_A)
    for i in spikes_A do
        input_B[i] ⇐ calculateInput(weights)
    end for
    spikes_B ⇐ run_neurons(input_B)
    output ⇐ calculateOutput(spikes_B)
    error ⇐ input -output
    if Error > 0 then
        mem_plus ⇐ onePulse(mem_plus)
        weights ⇐ memToWeights(mem_plus − mem_minus)
    else
        mem_minus ⇐ onePulse(mem_minus)
        weights ⇐ memToWeights(mem_plus − mem_minus)
    end if
end while
```

tance to its highest state we initialized it randomly between certain values and investigated if this approach is possible and if there are certain constrains on possible values of starting resistance.

2.6 Learning $f(x) = x^2$ using memristor PES learning with random initialization

During this experiment one small adjustment was made on top of the previous model: instead of learning the identity function we tried to learn simple math function - square function. It was done to test if the network would still work if we tried to compute something more complex.

3 Results

3.1 PES learning

As expected with PES learning we were able to teach current network to pass the input value from ensemble A to ensemble B quite easily. Output of

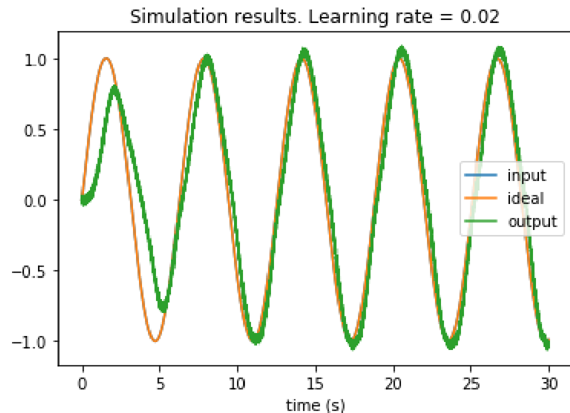

Figure 3.1: Simulation results of PES learning with learning rate of 0.02

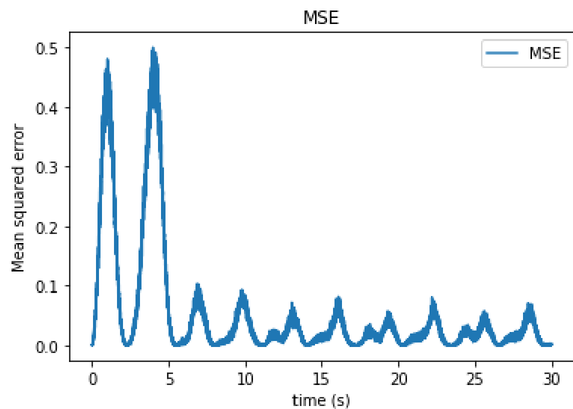

Figure 3.2: Mean squared error as a function of time for our network.

the simulation and the graph of the mean squared error can be seen in Figure 3.1 and Figure 3.2

Depending on the learning rate we can get different simulation results: lower learning rate will end up in slower learning and larger MSE in the beginning of the simulation while higher learning rate will results in oscillatory behavior around the ideal output and higher mean squared error. Learning rate of 0.02 seemed suitable for this task because with this learning rate the learning process would not be too slow or too fast and hence was used to have a "control" group of the experiment.

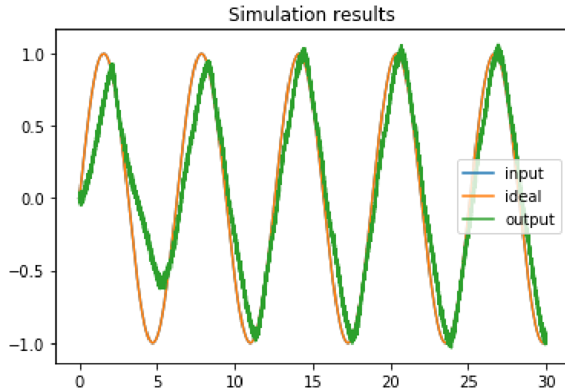

Figure 3.3: Simulation results of PES learning using memristors as connections of the network.

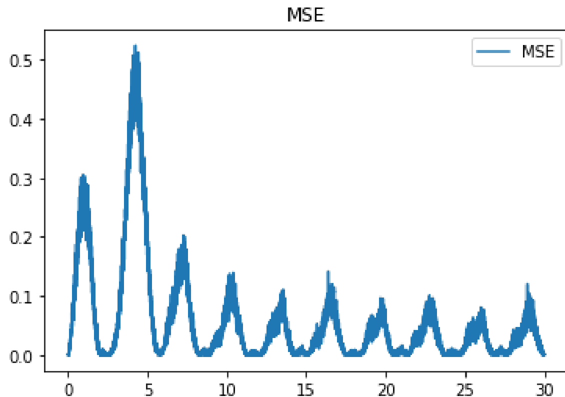

Figure 3.4: Mean squared error as a function of time of the memristors simulation.

3.2 Memristor PES learning

After running the simulating with memristors acting as connections we got the following graphs of the output and MSE that can be seen in Figure 3.3 and figure 3.4 As we can see from the figures memristor simulation of PES learning showed adequate results. Compared to the original PES learning it takes longer to reduce the initial error due to memristors properties. Since in memristor experiment we always apply the same voltage pulse regardless of the size of the error it takes longer to reduce the mean squared error. On top of that for the same reason it is more difficult to reduce the minimal error. As we can see from the Graph 3.2 even during the original PES learning mean squared error oscillated between 0 and 0.04. This oscillation became bigger in memristor case due to the properties of memristors. Once again since we always use one second pulses we can't control that precisely the value of the decoders. As a results mean squared error in memristors case oscillates between 0 and 0.1. The average value of positive memristors equaled to $1.02{*}10^7$ Ω and the average value of negative memristors equaled to $8.45{*}10^7$ Ω. The minimum value of a positive memristor equaled to 3.510^6 Ω while the minimum value of the positive one was $3.53{*}10^6$ Ω. At the same time the maximum values were $8.78{*}10^7$ Ω and $3.35{*}10^7$ Ω for positive and negative memristor. As we can see from those values every memristor was quite active during the learning process since the resistances reduced quite substantially from theirs initial values of $2.5{*}10^8$ Ω. There is a good explanation for that: since we never stop learning during those 30 seconds and eventually most of the "neurons" will most likely fire at some point we would lower the resistance of one of the memristors and because we only use positive voltage pulses we can't rise up the resistance back if we lowered it too much during the previous steps. To compensate for that we would need to lower the resistance of an opposite sign memristors and as a result eventually we will lower both of the respective resistances resulting in this outcome.

3.3 Memristor PES learning with random initialization

In this network we decided to investigate if the network would still work if we initialize memristors randomly in some range and for which ranges that would work. Several ranges of initial values were investigated and the results can be seen in Table 3.1. As we can see from that table and Figure 3.5 simulations works successfully if resistance is randomly initialized in the range between 10^6 and $2.5 * 10^8$ Ω but the time it takes to reach low error is now de-

Table 3.1: Possible ranges of initial resistances

Range of resistances (Ω)	Results
$10^7 - 2.5 * 10^8$	Successful
$10^6 - 10^7$	Successful
$10^5 - 10^6$	Unsuccessful
Ranges below 10^5	Unsuccessful

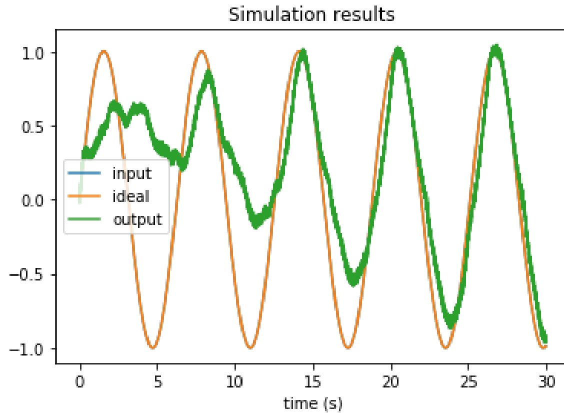

Figure 3.5: Simulation of PES learning while memristors randomly initialized between $10^6 - 10^7\Omega$ range

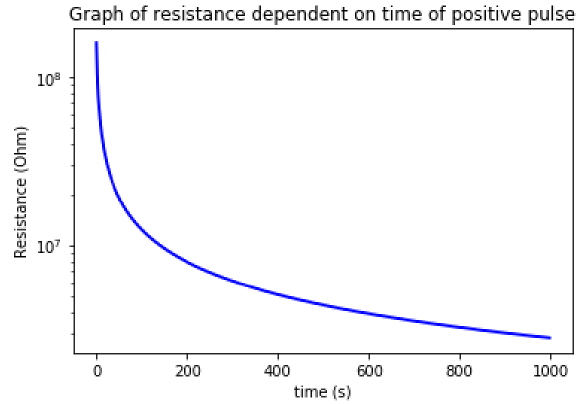

Figure 3.6: Graph of the resistance based on the time of positive pulses applied to the device

pendent on random initialization. From Figure 3.5 we can see that for that run it took almost 25 sec to get close to the ideal while in the condition with all memristors initialized to their highest resistance it took 10 sec. There are at least two factors that can explain this difference: first, of course, since the initialization is random we can get some unfavorable starting resistance;second, since they are initialized within $10^6 - 10^7$ range it is more difficult to change the resistance using 1 second pulses. As we can see from the Figure 3.6 - the lower the resistance, the lower the change after of resistance after 1 sec pulse. Because of that when the resistance is initialized in that range it takes longer to change it to the values we need. That is also one of the reason for why the lower initial range values no longer lead to a working simulations. If the resistances initialized bellow $10^6\ \Omega$ value it simply takes too long to try and change them. The other reason is the value of the parameter m that was mentioned in 2.4 section. Since i chose the value of m=10^5 for the resistance of $10^6 - 10^7$ translation from resistance to weight would result in a decoder weight between 1-10 and even higher for lower resistance which on top of the difficulties of changing resistance results in a not working network.
The average resistance of "plus" memristors were $7.8 * 10^6\ \Omega$ and the negative once $7.1 * 10^6\ \Omega$. The minimal value of the positive memristors was $3.36 * 10^6\ \Omega$ and the value of the negative was $3.35 * 10^6\ \Omega$. At the same time the maximal value for the positive one was $2.06 * 10^7\ \Omega$ and the maximal value of the negative once was $2.54 * 10^7\ \Omega$ Even though the starting values for the initial resistance for this run were significantly lower($10^6 - 10^7$) than for the previous experiment the end values of the memristors were quite close to the original experiment, especially the mean value of the resistances. This correspond well with the power law that we used for simulating memristors(Formula 2.1). Since during the first steps resistance falls much quicker than during consecutive steps it makes sense that starting at a lower power didn't result in a huge difference between end resistance of two different runs, since with lower resistance the lower would be the next resistance changed after applying 1s voltage pulse.

3.4 Learning $f(x) = x^2$ using memristor PES learning with random initialization

In this section we used the same network as we used in previous example with resistances randomly initialized between 10^7 and $2.5 * 10^8\ \Omega$. As an additional test of our network we tried to calculate square function. After running the simulation we got the following graphs for the output and mean squared error that can be seen in Figure 3.7 and 3.8. As we can see from the graphs the network successfully works towards learning the function but compared to the identity function it takes longer to learn it and

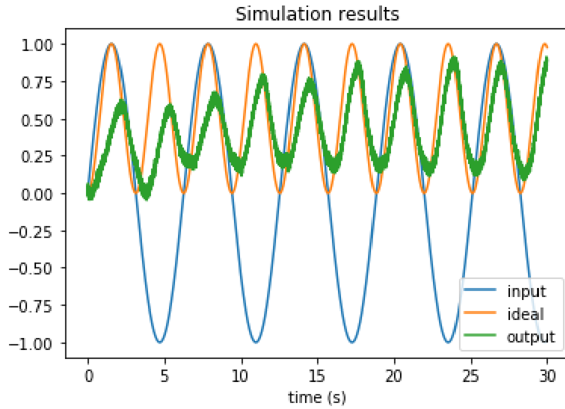

Figure 3.7: Simulation of PES learning of x^2 while memristors are randomly initialized between $10^7 and 2.5 * 10^8\ \Omega$

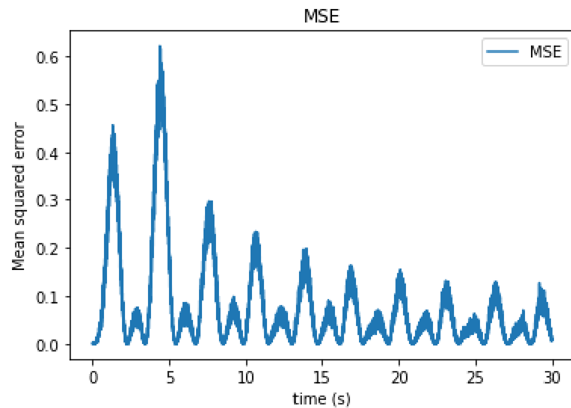

Figure 3.8: Mean squared error as a function of time of the memristor simulation for learning x^2

4 Discussion

In this research project we managed to build a learning network using memristors as a connectors between neurons. The model had some limitations but it was sufficient to simulate PES learning of the identity function. We tested two network: first with initializing memristors at their high resistance state and second with random initialization between certain intervals. Both networks were able to learn the identity function but the one with random initialization took longer to reach lower error due to the reasons descried in the results.

There were a few crude assumptions in this research. First is the power law that was used to simulate memristive devices. We have no data to confirm that the resistance would indeed act close to his law after similar number of voltage pulses due to the lack of an experimental data. Second is the way memristors were translated into simulation. In out simulation we used memristors instead of decoders values and based on this decoders the weight matrix was computed. That allowed us to have 50 memristors in the simulation. But if we were to build that network in real life we would need to use memristors for each connection and it would result in the use of 5000(2500 for positive and 2500 for negative connections) memristors. It didn't create any difference during the simulation since weight matrix is computed by matrix multiplication of 2 vectors and the end result would have been the same in both cases, because during the simulation every memristor acts exactly the same. But that would not be the case in real life. In real world we expect them to have similar but not the same properties and would involve some degree of randomness. PES learning is robust and it is plausible that the simulation would still work in the real world since we act based on the error and it would just results in different end values for the memristors.

On top of that there were some limitations to our research. In this research we only tried learning the identity function and the x squared function. PES learning rule is widely used in many models using Neural Engineering Framework. If we could implement memristors PES learning into one of the larger models it could give us some valuable information on plausibility of using memristors for this kind of learning. Another possible improvement is

using pulses with several different timings based on the size of the error instead of just 1 second pulses. Potentially this could lead to a smaller error but it also comes with some problems. Since the change of the pulse differs based on the current resistance it is not the trivial to figure out the preferable time of the pulse and since in the real world version of this network we would not know the current resistance without applying reading pulse(which creates some overhead cost) it would be quite difficult to figure out the new time of the pulse. Also using negative pulses for the updating step should potentially improve the work of the network but to verify that more information on memristors behavior is needed.
In the current research to calculate the weights of the decoders we used formulae 2.3 and in that formula we've used somewhat arbitrary value of m. It is possible that there are better value of the parameter m that would lead to a more efficient network, but on top of that it is possible that there is a better method of transforming resistance into the weight altogether. Possibly some other method would be more fitting for memristive devices.
For the possible physical implementation we might try to use system based on memristor crossbar arrays (Hassan et al., 2019). According to Hassan it was already used to implement similar systems in which voltage and current were used to represent an actual output and input data. To implement that and additional hardware was needed to allow for constant weight updating. In Figure 4.1 we can see a schematic implementation for 4x2 neural network. In our case we would need to scale this up to the size of our network. The biggest downside of this kind of system is the need of an additional hardware that is needed for weight updating based on the error.
To sum up we think that this research project is the first towards understanding the possible range of possible application of said devices. The main problem is that our research if fairly abstract and simplistic. If we were able to apply PES learning to one of the existing larger networks it would have given us better understanding of possible problems and limitations.

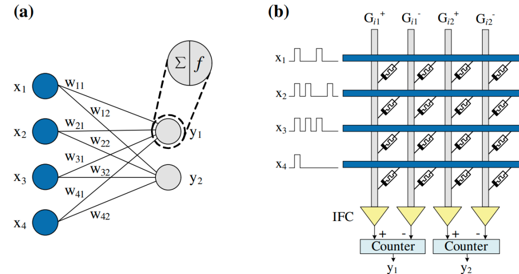

Figure 4.1: Neural network implementation using memristor crossbar arrays. a Shows a conventional diagram for 4x2 neural network while b shows the crossbar implementation of that network. Reprinted from (Hassan et al., 2019)

5 Acknowledgements

We would like to thank A. Goosens, N. Taatgen and T. Banerjee for providing experimental data on the behavior of their memristive device.

References

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T., Rasmussen, D., Choo, X., Voelker, A., and Eliasmith, C. (2014). Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7:48.

Bekolay, T., Kolbeck, C., and Eliasmith, C. (2013). Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks. pages 169–174.

Cai, W. and Tetzlaff, R. (2014). *Synapse as a Memristor*, pages 113–128. Springer International Publishing, Cham.

Chua, L. (1971). Memristor – the missing circuit element. *IEEE TRANS. ON CIRCUIT THEORY*, 18(5):507–519.

Chua, L. (2013). Memristor, hodgkin–huxley, and edge of chaos. *Nanotechnology*, 24(38):383001.

Chua, L. (2019). *Resistance Switching Memories are Memristors*, pages 197–230. Springer International Publishing, Cham.

Eliasmith, C. and Anderson, C. (2003). *Neural Engineering: Computation, Representation and Dynamics in Neurobiological Systems.*

Gamrat, C. (2010). Challenges and perspectives of computer architecture at the nano scale. pages 8–10.

Goossens, A., Das, A., and Banerjee, T. (2018). Electric field driven memristive behavior at the schottky interface of nb doped srtio3.

Hassan, A. M., Liu, C., Yang, C., (Helen) Li, H., and Chen, Y. (2019). *Designing Neuromorphic Computing Systems with Memristor Devices*, pages 469–494. Springer International Publishing, Cham.

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636.

Sheridan, P. and Lu, W. (2014). *Memristors and Memristive Devices for Neuromorphic Computing*, pages 129–149. Springer International Publishing, Cham.

Strukov, D. B., Snider, G. S., Stewart, D. R., and Williams, R. S. (2008). The missing memristor found. *Nature*, 453(7191):80–83.

Von Neumann, J. (1981). The principles of large-scale computing machines. *IEEE Ann. Hist. Comput.*, 3(3):263–273.