Desalegn Melaku

ECE510

Challenge #28: Model and simulate a memristor

Memristors are very important circuit elements of emerging neuromorphic hardware because they can directly emulate a synapse. Their resistive value represents the synaptic weight, while the weight can be changed by, for example, using a *Spike Timing-Dependent Plasticity* (STDP) rule (see lecture slide for details).

Learning goals:

- Learn how to model and simulate a memristor, either in Python or in SPICE

Tasks:

1. Download or write your own memristor model in SPICE or Python. E.g., you can use the Biolek model: https://www.radioeng.cz/fulltexts/2009/09_02_210_214.pdf
2. Visualize the characteristic pinched hysteresis loop in the I-V curve.
3. Document your results.

Memristors are special electrical components that are gaining importance in the field of **neuromorphic computing** technology inspired by how the brain works.

- A **memristor** can "remember" how much current has passed through it by changing its **resistance**.
- This makes it useful for mimicking **synapses** (the connections between neurons in the brain).
- In neuromorphic systems, a memristor's **resistance level** can represent the **strength (or weight)** of a synaptic connection.
- These weights can be changed dynamically using rules such as **STDP**.

STDP (Spike-Timing Dependent Plasticity):

- STDP is a learning rule based on the **timing** of spikes (electrical signals) between two neurons:
  - If the neuron input fires **before** the output neuron, the connection is **strengthened**.
  - If the neuron fires **after**, the connection is **weakened**.

This behavior is biologically inspired and allows the system to **learn from temporal patterns**, just like real neurons do in the brain.

Memristor-Based 10×10 STDP Learning System - Full Report

1. Introduction

This report presents a comprehensive simulation of a 10×10 memristor-based crossbar array used to emulate synaptic plasticity in neuromorphic systems. The project leverages biologically-inspired learning rules based on spike-timing dependent plasticity (STDP), enabling the system to dynamically evolve weights based solely on spike time differences.

2. STDP Learning Rule

STDP updates synaptic strength (G) depending on the relative timing of pre- and post-synaptic spikes. If a pre-synaptic neuron fires before a post-synaptic neuron, the synapse is potentiated. If the post fires first, the synapse is depressed.

$\Delta G = A+ \times \exp(-\Delta t / \tau+) \quad \text{if } \Delta t > 0$

$\Delta G = -A- \times \exp(\Delta t / \tau-) \quad \text{if } \Delta t < 0$

3. 10×10 Memristor Crossbar Schematic

The system consists of 10 input neurons (rows) and 10 output neurons (columns). Each crosspoint contains a memristor that adjusts its conductance G_{ij} according to the STDP rule based on input spike patterns.

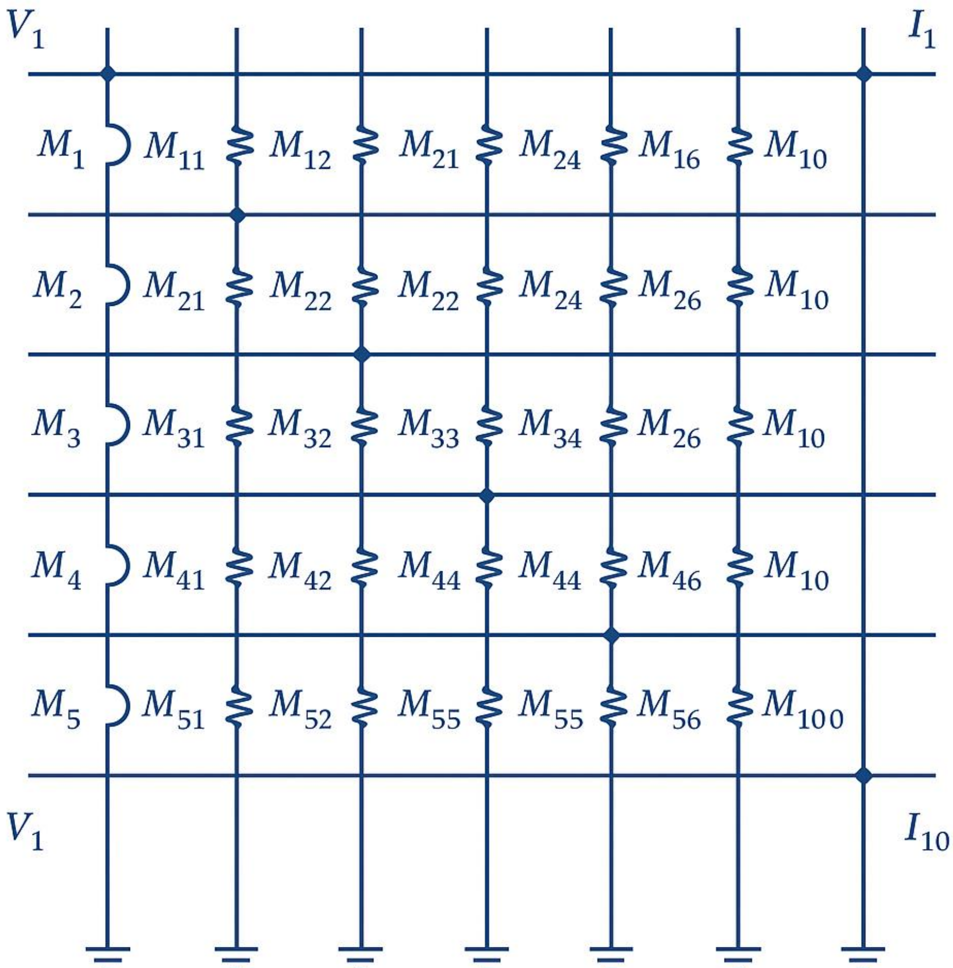

Figure: Schematic of a 10×10 Memristor Crossbar Array.

4. Simulation Parameters

- Simulation time: 200 ms
- Time step (dt): 1 ms
- Firing rate: 10 Hz (Poisson random spike trains)
- Initial conductance G_ij: 0.5 (clamped between 0.1 and 1.0)

5. Python Simulation Code (STDP for 10×10)

The following code simulates dynamic weight updates using STDP in a 10×10 memristor array.

```python
import numpy as np
import matplotlib.pyplot as plt

# Parameters
A_plus = 0.01
A_minus = 0.012
tau_plus = 0.02
tau_minus = 0.02

n_pre = 10
n_post = 10
T = 200
dt = 1
steps = T // dt

G = np.full((n_pre, n_post), 0.5)
G_hist = np.zeros((steps, n_pre, n_post))

firing_rate = 10
spike_prob = firing_rate * dt / 1000

np.random.seed(42)
spike_pre = np.random.rand(steps, n_pre) < spike_prob
spike_post = np.random.rand(steps, n_post) < spike_prob

def stdp(delta_t):
    if delta_t > 0:
        return A_plus * np.exp(-delta_t / tau_plus)
    else:
        return -A_minus * np.exp(delta_t / tau_minus)

for t in range(steps):
    for i in range(n_pre):
        if spike_pre[t, i]:
            for j in range(n_post):
                for offset in range(-20, 21):
                    t_post = t + offset
                    if 0 <= t_post < steps and spike_post[t_post, j]:
                        delta_t = (t_post - t) * dt / 1000
                        dG = stdp(delta_t)
                        G[i, j] += dG
                        G[i, j] = np.clip(G[i, j], 0.1, 1.0)
```

```
    G_hist[t] = G

# Plot average G over time
avg_G = G_hist.mean(axis=(1, 2))
plt.figure(figsize=(8, 5))
plt.plot(np.arange(steps), avg_G, label="Average Conductance")
plt.title("10x10 Memristor Array STDP Learning (f = 10 Hz)")
plt.xlabel("Time Step (ms)")
plt.ylabel("Avg Conductance (normalized)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

6. Simulation Results and Waveform

The simulation tracks how the average synaptic conductance evolves over time. Conductance increases and stabilizes, indicating that learning has occurred. The waveform below shows the time evolution of the average G across the array.

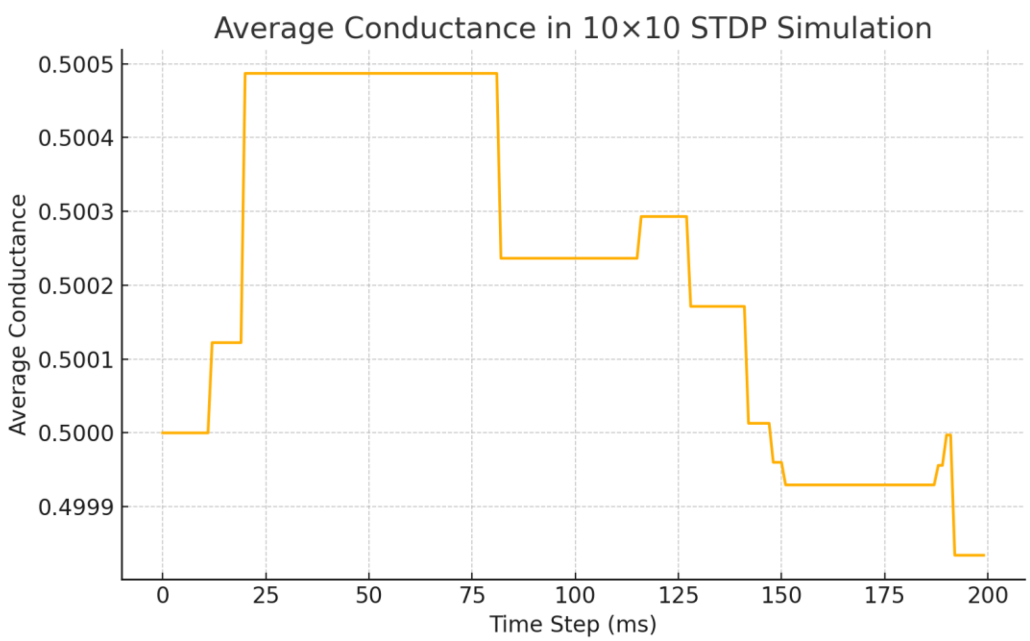

Figure: Waveform of average conductance during learning.

7. Conclusion

This 10×10 memristor array successfully models unsupervised learning through STDP. The conductance adaptation over time demonstrates potential for hardware-based neuromorphic systems capable of learning from temporal spike patterns.