Shahid Beheshti University

Anita Soroush

98222085

assignment 3 of Computational Neuroscience course

Abstract

In the human brain, most of the learning processes can be divided into two types: unsupervised or reinforcement learning (and in some cases a combination of these two). In this assignment, we try to implement two algorithms to mimic these two ways of learning.

Introduction

In this report, I will represent the results of the implementation of 2 algorithms including STDP (Spike-Timing Dependent Plasticity) and Reward-Based STDP. The first algorithm helps us with simulation of unsupervised learning processes in the brain and the second algorithm, which is an expansion of the first one, is an imitation of the reinforcement learning process.

In the implementation phase, I used the LIF excitatory neurons that I implemented for the first and second assignments.

Part 1
(STDP)

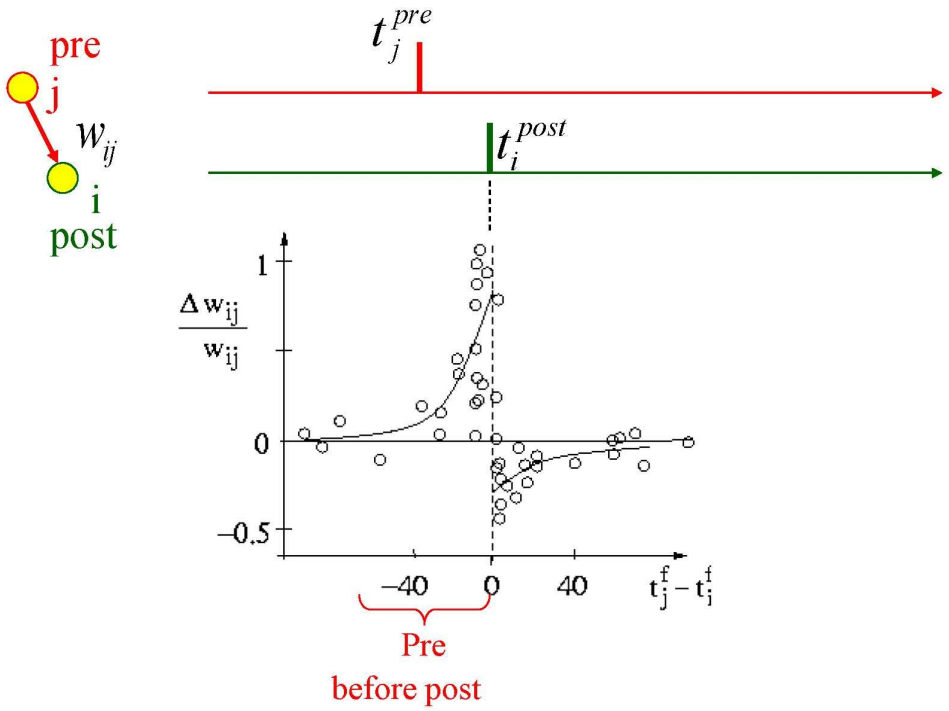
In this part, we were asked to make a fully connected neuron population made up of 3 neurons, each of which has a different input current and the synaptic weights between each two of them changes in time by the following rule:

$$\Delta w_{+} = A_{+}(w).exp(-|\Delta t|/\tau_{+}) \quad at\ t_{post}\ for\ t_{pre} < t_{post}$$

$$\Delta w_{-} = A_{-}(w).exp(-|\Delta t|/\tau_{-}) \quad at\ t_{pre}\ for\ t_{pre} > t_{post}$$

where $|\Delta t|$ is $|t_{post} - t_{pre}|$

This algorithm is based on some biological discoveries that are shown in the following plots.



After some trial and error I found the following values to match the best with other parameters in my LIF neurons that I had set before:

$A_+(w)$: +25

$A_-(w)$: -25

τ_+ : 5

τ_- : 5

- **Experiment 1:**

The input currents are as follows:

```
I1 = lambda x: 50
I2 = lambda x: 30
I3 = lambda x: 20
```

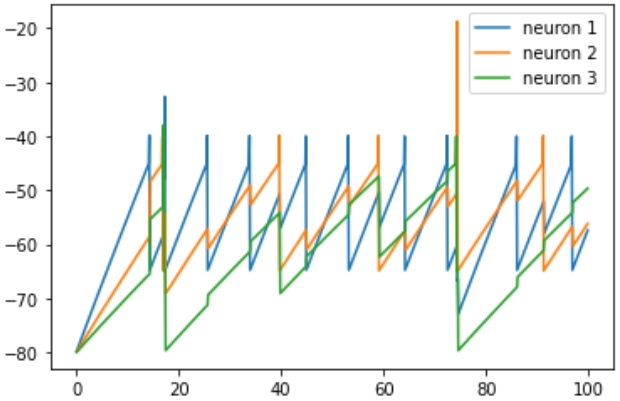
and the weights at the beginning are:

1 → 2 :10
1 → 3 :10
2 → 1 :10
2 → 3 :10
3 → 1 :10
3 → 2 :10

After running the STDP algorithm the weights have changed and the final weights are as follows:

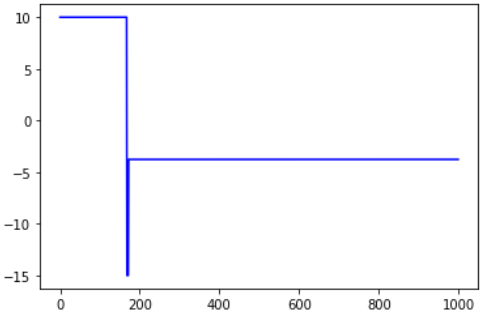
1 → 2: -3.764743276150938
1 → 3: 1.7580021872834368
2 → 1: -6.161019526329569
2 → 3: 1.7580011508910263
3 → 1: -5.982462533189049
3 → 2: 10.000000000001524

and the neurons' potentials have changed during the run time which can be seen in the following plot:

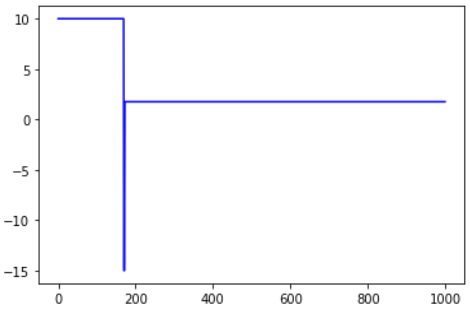


and the weights have updated like:

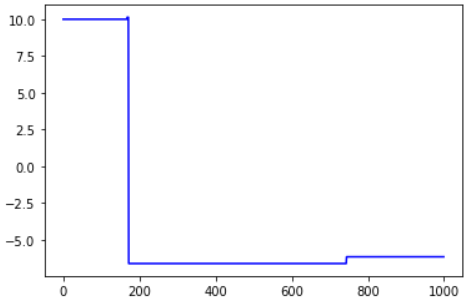
1→2:



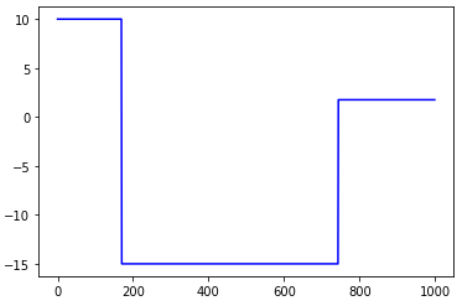
1→3:



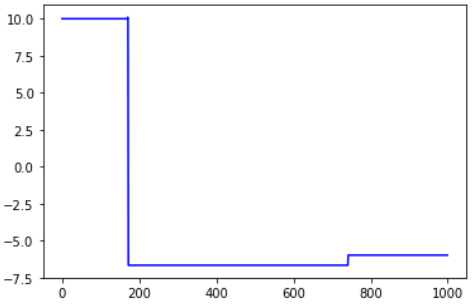
2 → 1:



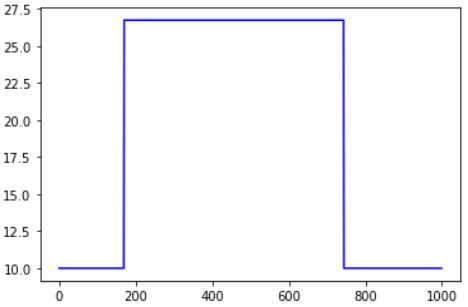
2 → 3:



3→1:



3 → 2:



- **Experiment 2:**

The input currents are as follows:

```
I1 = lambda x: 20* (math.sin(x/10) + 0.9)
I1 = lambda x: 50
I2 = lambda x: 80* (math.sin(x/10) + 0.9)
```

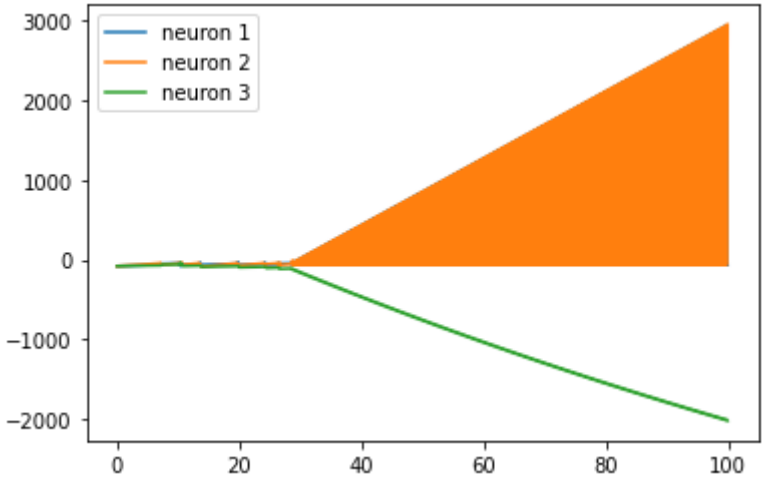
and the weights at the beginning are:

1 → 2 :10
1 → 3 :10
2 → 1 :10
2 → 3 :10
3 → 1 :10
3 → 2 :10

The weights at the end:

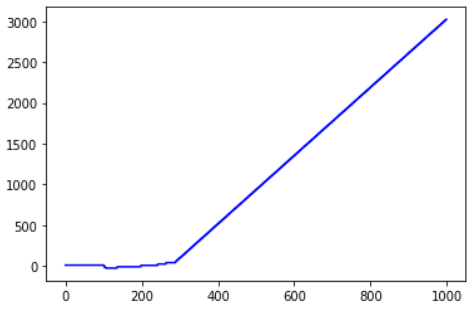
1 → 2 :3024.2822544003593
1 → 3 :-14.958460897062762
2 → 1 :3031.561598375236
2 → 3 :1.8199702105708213
3 → 1 :23.678751799413426
3 → 2 :9.938030940320163

And the neurons' potential alterations during the run time:

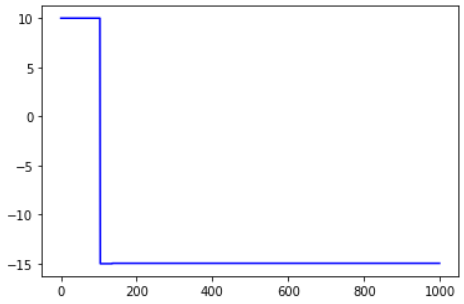


and the weights have updated like:

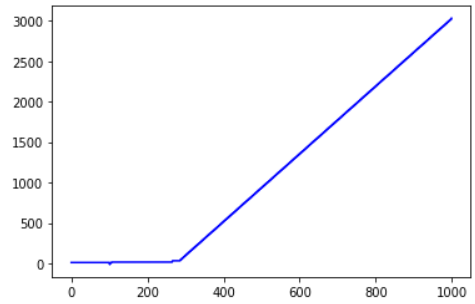
1→2:



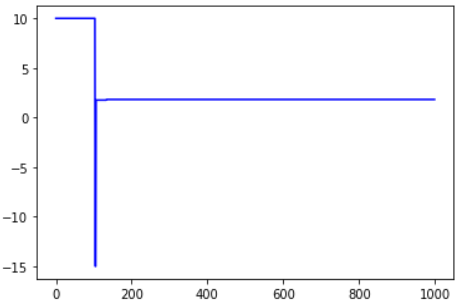
1→3:



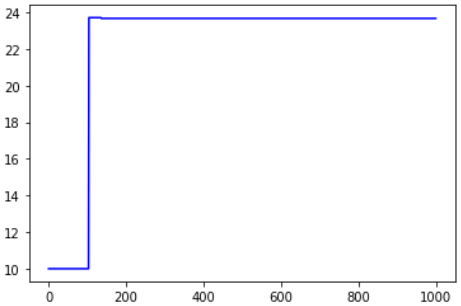
2 → 1:



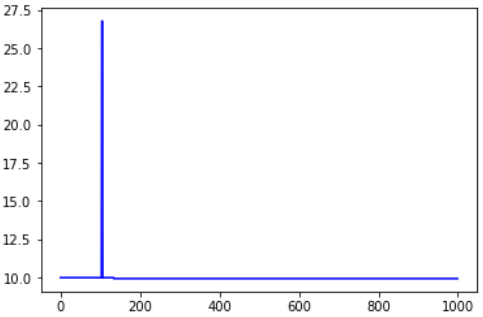
2 → 3:



3→1:



3 → 2:



Part 2

(Reward-Based STDP)

In this algorithm, in each step the synaptic weights are updated based on the amount of Dopamine which is released as a reward (if the spike is related to the correct neuron in the output layer) or as a punishment (otherwise).

To implement this spiking neural network, I simply used a neural population made up of 7 neurons. There is no connection between the first 5 and the last 2 neurons (which are actually mimicking the input and output layer) but each neuron in the input layer is connected to all of the neurons in the output layer.

I implemented a method named trainig_step and I called it 10 times (the same as the number of training data) and each time the weights were updated. All neurons receive a constant current of 0 unless they are in the input layer. The input layer neurons receive the currents which are given in the training data multiplied by 10.

Moving on to the testing stage, I made a SNN and used the weights which were proved to be the best by the end of the learning phase.

The result on the test data is as follows:

Test 1: after 1000 iterations no spike was observed in the output layer.
Test 2: True
Test 3: True
Test 4: True
Test 5: True
Test 6: after 1000 iterations no spike was observed in the output layer.
Test 7: False
Test 8: after 1000 iterations no spike was observed in the output layer.
Test 9: True
Test 10: False