Memory Tchnologies for Machine Learning - EITP25
Project Group 8 - Spring 2020

Training Spiking Neural Networks using STDP learning mechanism

Christian Clemedtson (tfy14ccl)
Mikael Kristensson (tfy14mkr)
Hicham Mohamad (hi8826mo-s)

June 14, 2020

1 Introduction

In this project, we are going to work on practical implementations that cover image recognition tasks using the unsupervised learning approach simulated in **Spiking Neural Networks** (SNN). At the end, we will get familiar with the dynamics of the **STDP** learning mechanism, a biologically plausible rule that allows efficient learning, and practice on writing a simulation for SNN in a Python software package called **Brian2** [15]. In particular, we will Train a SNN for a 10-class classification problem, that involves the `MNIST` image dataset of handwritten digits, consisting of data-target pairs with images $\mathbf{X}_i \in [0,1]^{28\times 28}$ and targets $t_i \in \{0, 1, \cdots, 9\}$.

The resulting plots and values in this report are obtained by running the implemented Jupyter notebook `EITP25_project_spiking_MNIST_G8_opt`.

This report does not assume prior knowledge about spiking neurons, and it contains an extensive list of references to relevant literatures on spiking neuron networks and neurobiology. In section 2, we review the most relevant theory behind STDP learning mechanism and neuron activity. In section 3 we explain the structure of our SNN, and in section 4 we review the useful parameters in our Brian2 simulation, file saving and then neuron and synapse implementations. Section 5 and 6 contain the simulation tasks.

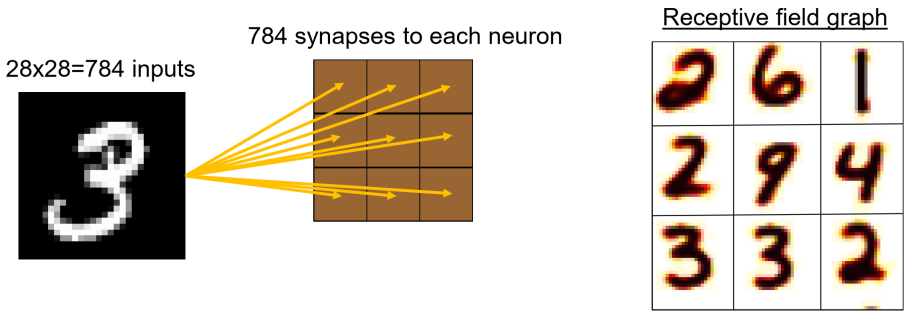


Figure 1: Illustration of the Spiking neural network structure to be used in the project. We have $28 \times 28 = 784$ *synapses to each output neuron.*

Objectives

The objectives of this project work are to:

1. assign a class to each output neuron, based on its highest response to the ten classes of digits over one presentation of a random subset (1000 examples) of the MNIST training dataset.

2. measure the classification accuracy of the network on the test dataset.
3. Then we need to investigate at least three of the following **training tasks**:
   (a) How varying the setting of the lateral inhibition affects the synapse map and the accuracy.
   (b) The effect of varying the network size.
   (c) The effect of having an adaptive threshold.
   (d) Explore the effect of the nonlinear memristor response.
   (e) Retrain the network, with the same parameters, on the MNIST Fashion dataset.

2 Background

2.1 Neuro-inspired computing using resistive synaptic devices

The biological brain is a highly energy efficient system. There are approximately 10^{11} **neurons** and 10^{15} **synapses** in the brain. Neurons produce **spikes** with a frequency of 10 Hz, and only 1 % of the neurons are active at the same time. Total power consumption of human brain is $\sim$ 20 W [3]. Therefore, we can figure out how the popular artificial neural networks (ANN) are inspired by features found in neuroscience.

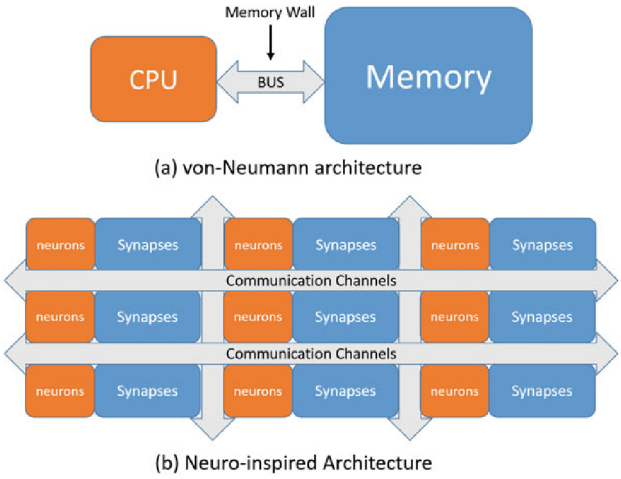


Figure 2: A revolutional shift of the computing paradigm from the computation-centric (von Neumann architecture) to the data-centric (neuro-inspired architecture).

However, the neuron-inspired learning algorithms in artificial neural networks (ANN) require huge computational resources to train the weights in the network. In addition, the well-known "**memory wall**" problem that the data movement between the microprocessor and off-chip memory/storage has became the bottleneck of the entire system which is based on Von-Neumann architecture.

In terms of energy efficiency, spiking neural networks (SNN) perform better and are more biologically realistic than ANNs [3]. This promising solution is based on the **neuro-inspired architecture** where the **neurons** are simple computing units, and the **synapses** are local memories that are massively connected via the communication channels [3], as illustrated in Figure 2.

2.2 Biological neuron activity

In order to understand how the **STDP learning mechanism** is performing, we need to have a basic understanding of the biological neuron activity. In this section, we introduce the basic components of the biological neural networks and their primary mechanisms.

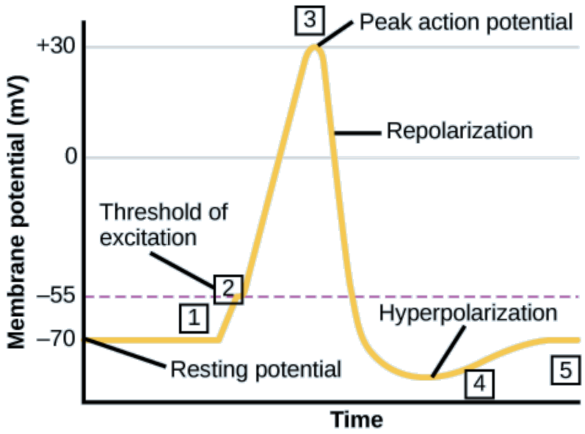
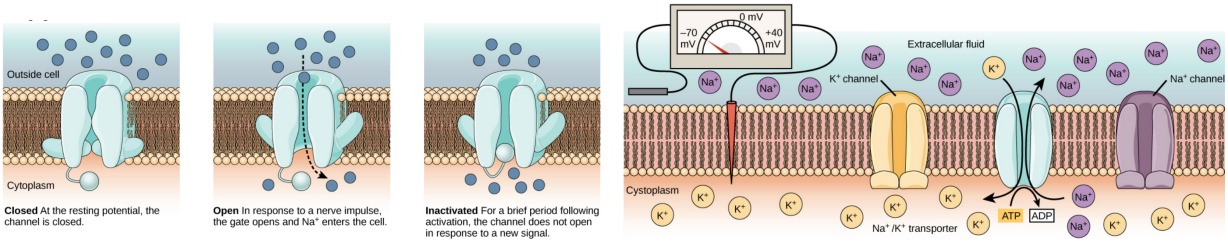


*Figure 3: Formation of an action potential can be divided into five steps: (1) A **stimulus** from a sensory cell or another neuron causes the target cell to depolarize toward the threshold potential. (2) If the **threshold of excitation** is reached (-55 mV), all Na+ channels open and **the membrane depolarizes**, i.e. having a decrease in the difference in voltage between the inside and outside of the neuron. (3) At the **peak** action potential (+40 mV), K^+ channels open and K^+ begins to leave the cell. At the same time, Na^+ channels close. Once depolarization is complete, the cell must now "reset" its membrane voltage back to the resting potential. (4) The membrane becomes **hyperpolarized** as K^+ ions continue to leave the cell (diffusion), in that the membrane potential becomes more negative than the cell' s normal resting potential. The hyperpolarized membrane is in a **refractory period** and cannot fire and produce another action potential because its sodium channels will not open. (5) The K^+ channels close and the Na^+/K^+ transporter restores the **resting potential**. At this point, the sodium channels will return to their resting state, meaning they are ready to open again if the membrane potential again exceeds the threshold potential.*

The cell membrane of a neuron encloses **cytoplasm** with various ions dissolved in it. The neuron itself is immersed in a salt solution, the extracellular fluid. The ions of the cytoplasm consist mainly of positively charged **potassium ions** K^+ and large negatively charged organic molecules, such as proteins. Outside the cell, the extracellular fluid contains mostly positively charged **sodium ions** Na^+ and negatively charged **chloride ions** Cl^-. In the following, we explore some of the basics of the **neuron activity**.

Resting potential

Unstimulated, neurons maintain a constant electrical **difference**, or potential, across their cell membranes. This potential, called **resting potential**, is always negative inside the cell. It ranges from -40 to -90 mV. If a neuron is stimulated, the negative potential inside the neuron can be made either more or less negative, depending on the stimulus.



(a) (b)

*Figure 4: a) Voltage-gated ion channels open in response to changes in membrane voltage. After activation, they become inactivated for a brief period and will no longer open in response to a signal. b) At the **resting potential**, all voltage-gated Na^+ channels and most voltage-gated K^+ channels are closed. The Na^+/K^+ transporter pumps K^+ ions into the cell and Na^+ ions out.*

Action potential

If potential is made sufficiently less negative, it reaches a level called **threshold** (-55 mV), and **action potential** is triggered. During the action potential, the neuron suddenly becomes 20 to 50 mV positive

inside. Action potentials last a few milliseconds before the cell restores its negative resting potential.

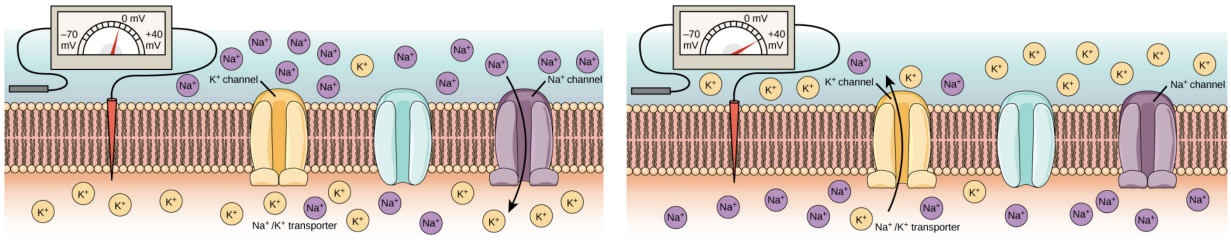


Figure 5: A nerve impulse causes Na^+ to enter the cell, resulting in (a) depolarization. At the peak action potential, K^+ channels open and the cell becomes (b) hyperpolarized.

2.3 Hodgkin-Huxley Model

Hodgkin and Huxley performed experiments on the giant **axon of the squid** and found three different types of **ion current**, sodium, potassium, and a leak current that consists mainly of Cl^- ions. This Hodgkin-Huxley model can be understood with the help of Figure 6.

The semipermeable **cell membrane** separates the interior of the cell from the extracellular liquid and acts as a **capacitor**. If an input current $I(t)$ is injected into the cell, it may add further charge on the capacitor, or leak through the **channels** in the cell membrane. Each channel type is represented by a resistor. The unspecific channel has a leak resistance R, the sodium channel a resistance R_{Na} and the potassium channel a resistance R_K. In this way, we get a **biologically plausible** leaky-integrate-and-fire (LIF) model based composed of three parallel channels Na, K and leakage

$$C_n \frac{dv(t)}{dt} = I_{input}(t) - I_L - I_{Na} - I_K \tag{1}$$

These three currents are defined by the following equations, where m, h, $n \in f(v,t)$ are activation functions

$$\begin{aligned} I_{Na} &= g_{Na}\ m^3\ h(v(t) - V_{Na}) \\ I_K &= g_K\ n^4(v(t) - V_K) \\ I_L &= \frac{1}{R}(v(t) - V_L) \end{aligned} \tag{2}$$

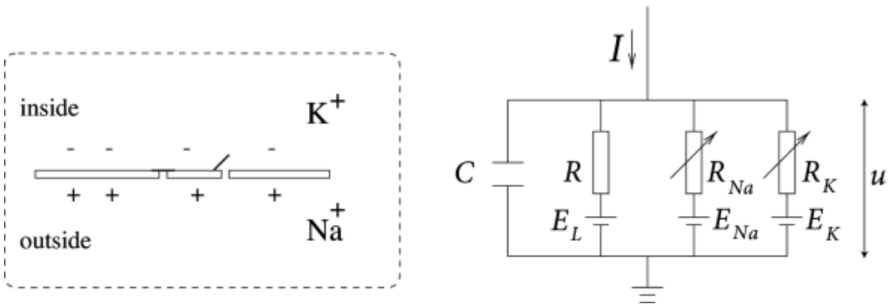


Figure 6: Schematic diagram for the Hodgkin-Huxley model.

2.4 Synapses - STDP

For a biological synapse, in Spike Timing Dependent Plasticity (STDP), the **difference in time** between the firing of pre- and post-synaptic neurons determines whether the "strength" of the connection between those neurons increases or decreases.

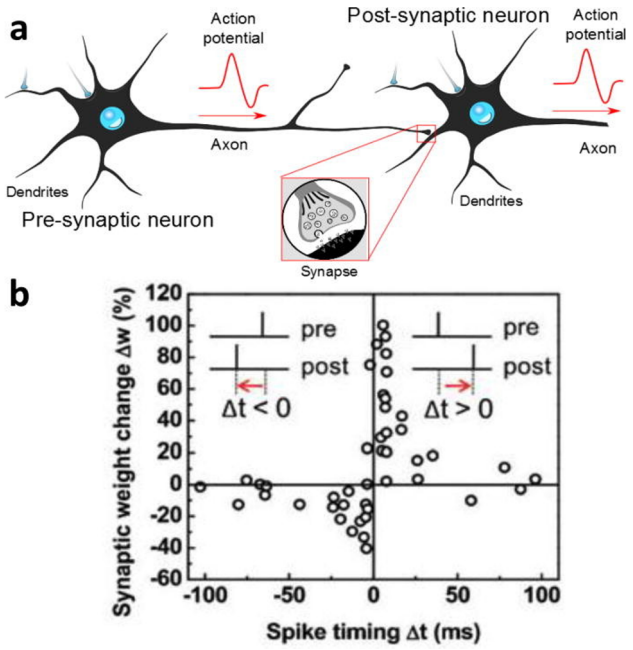


Figure 7: (a) Schematic illustration of a ***synaptic connection*** *and the corresponding pre- and post-synaptic neurons. The synaptic connection strengthens or weakens based on the spike activity of these neurons; a process referred to as* ***synaptic plasticity****. (b) A well-known plasticity mechanism is* ***spike time-dependent plasticity*** *(STDP), leading to* ***weight changes*** *that depend on the relative timing between the pre- and post-synaptic neuronal spike activities.*

In other words, when the pre-synaptic spike arrives before the post-synaptic spike ($\Delta t > 0$), the synapse exhibited increased synaptic weight ($\Delta w > 0$) and when the pre-synaptic spike arrives after the post-synaptic spike ($\Delta t < 0$) that the synapse exhibited decreased synaptic weight ($\Delta w < 0$).

2.5 Synapse with excitatory and inhibitory plasticity

In neurobiology, **lateral inhibition** is the capacity of an excited neuron to reduce the activity of its neighbors. Lateral inhibition disables the spreading of action potentials from excited neurons to neighboring neurons in the lateral direction.

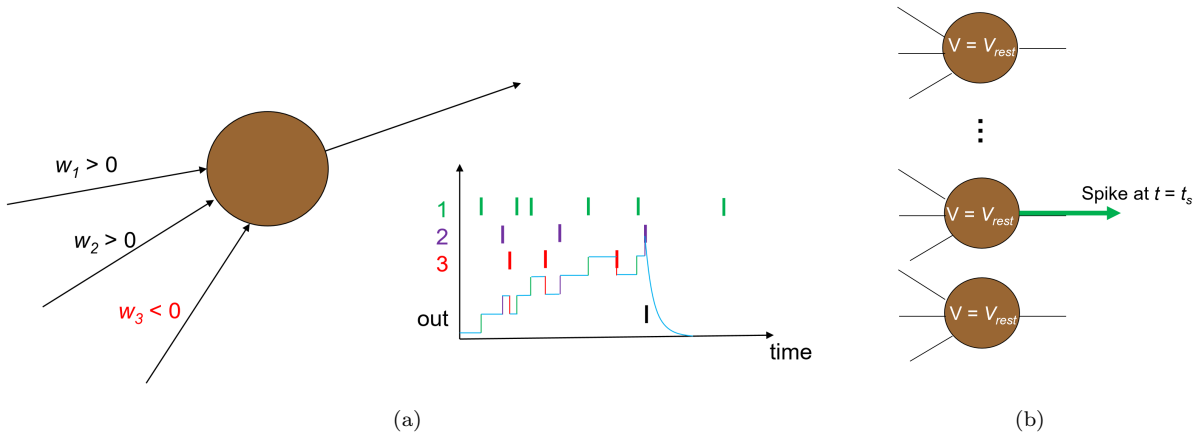


Figure 8: a) ***Excitatory/Inhibitory*** *synapses: Inhibition can be modeled by synapse with negative weight. b)* ***Winner-takes-all*** *(WTA): all neurons in a layer can completely inhibit all others in the layer, i.e. the first to fire will be the only one to fire.*

Not all synapses shift the neuron potential towards threshold (**excitation**), but some reduce the neuron potential (**inhibition**). Inhibition can be modeled by synapse with negative weight, and can be a crucial feature for learning in SNNs.

With **Winner-takes-all** (WTA), all neurons in a layer can completely inhibit all others in the layer, i.e. the first to fire will be the only one to fire. It is typically implemented as inhibition for a certain time, matching **refractive period**, i.e.

$$V = V_{rest} \quad \text{on all until} \quad t = t_s + t_{refr}. \tag{3}$$

3 Network Implementation

In this project, the SNN implementation follows that in *Diehl and Cook* [5] , with some modifications.

The network consists of two layers. The first layer is the **input layer**, containing 28×28 neurons (one neuron per image pixel), and the second layer is the **output layer**, containing a variable number of excitatory neurons and as many inhibitory neurons (processing layer).

Each input is a **Poisson spike-train**, which is fed to the excitatory neurons of the second layer. The rates of each neuron are proportional to the intensity of the corresponding pixel in the example image.

The **excitatory** neurons of the second layer are connected in a **one-to-one fashion** to inhibitory neurons, i.e., each spike in an excitatory neuron will trigger a spike in its corresponding inhibitory neuron.

Each of the **inhibitory** neurons is connected to all excitatory ones, except for the one from which it receives a connection. This connectivity provides lateral inhibition and leads to competition among excitatory neurons.

3.1 Input layer implementation

In the initial implementation, the network consists of 28×28 input neurons (one neuron per image pixel) as **Poisson sources** with spike frequency between 0-50 Hz depending on the pixel value of the image, i.e. white = 50 Hz. Each **input neuron** connects with synapses to each of the N_{out} neurons in the **output layer**. It is possible to vary N_{out}, but to simplify plotting of synapse weights we should make sure that $\sqrt{N_{out}}$ is an integer.

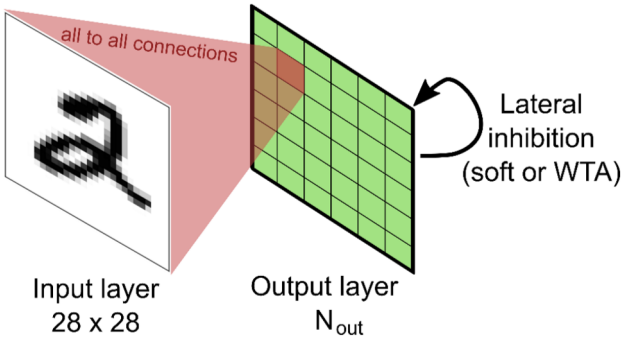


Figure 9: Illustration of the Spiking neural network structure to be used in the project. We have $28 \times 28 = 784$ *synapses to each output neuron.*

3.2 Output layer implementation

The output neurons implement a **variable threshold**, such that strong spiking activity will be offset by a higher and higher threshold. The **output layer** also implements **lateral inhibition**, which can be either *disabled*, *soft* or *winner-takes-all*, depending on the value of the parameter `latInh`. A graphical representation of the network structure is illustrated in Figure 9.

3.3 Image data

The image data can be accessed through the variables `training_data` (for the 60 000 training examples) and `testing_data` (10 000 examples for testing). These are **dictionaries** in which the **image data** can be retrieved by the label 'x', for example:

"`for image in training_data['x']`:" which would iterate through all images.

The corresponding **labels** are accessed through `training_data['y']`, such that

`int(training_data['y'][2])` would correspond to the label for the image at position 2 in `training_data['x']`.

4 Code walk-through

4.1 Simulation parameters

"`EITP25_project_template.py`" contains the main code which should be run. "`plotting_tools.py`" contains some functions that can be used to plot data during or after training. In the main program there are some parameters which values control the flow of the program:

- `debug` If set to false you will compile the code into C++ code which is slightly faster, but then you can't plot or save stuff during training. I suggest to leave this to True.
- `useFashion` If set to true you will use the MNIST Fashion data set instead of MNIST numbers. Fashion is a harder challenge to be tested at the end of the project.
- `train` If true the program will train the network on the chosen dataset.
- `test` If `train = False`, this determines whether to do determine the classes (if false) or test the network on the test data set (if true).
- `restart_sim` Defaults to false. If true then start over training despite there being previous versions of the network.
- `plotEvery` During training the receptive fields (synapse strengths) of the output layer will be plotted every `<plotEvery>` trained images.
- `saveEvery` During training the synapse strengths will be saved to disk every `<saveEvery>` trained images. The filename will be `<network_file>+<nbr of images trained>`. Thus one can save multiple states of the network during training which can then be used during evaluation if one wants to see how the network has evolved.
- `monitorSpikes` If true will include a `SpikeMonitor` in the network to monitor the spiking of the output layer. It will also plot a map of the spiking activity at each `<plotEvery>` interval. The use of SpikeMonitors slows down the training somewhat.
- `N_out` The number of neurons in the output layer
- `latInh` If =2 use *winner-takes-all* as lateral inhibition, if =1 uses *soft lateral* inhibition, if < 1 use no lateral inhibition at all!
- `rate_max` The maximum input spiking rate (corresponding to a completely white pixel). This may be adjusted to avoid having too high overall spiking activity.
- `presentation_time` The time which each image will be presented to the network during training.
- `rest_time` The time in between image presentations, in which all dynamic variables are allowed to go back to their resting states.
- `learning_rate` Controls how large fraction of the maximum weight/conductance value the synapse weight will change with each STDP weight update. This should be set to zero for class evaluation and testing.
- `network_file` The filename (prefix) to be used for saving the network data during training.
- `epochs` The number of epochs (60000 images) to train for. I have only tested the code for epochs = 1, so expect problems if set higher.

4.2 File saving

The files that are saved contain only the **parameter values of the network**, the network objects still need to be recreated in the same way for it to be reloaded properly. Thus make sure you keep track of all the parameters of the network during your experiments, so that you can reload any saved data for the **testing phase**. The names of the saved files will be `<network_file> + <nbrOfTrainedImages>`.

For example a file corresponding to a network with 100 neurons and implementing Winner-takes-all which have been trained on 2000 images could be named "`network_N100_WTA_2000`". The numbers at the end are added automatically as the network is saved during training.

Thus, it is clever to choose `network_file` in a way that allows you to determine what **important parameters** were used for the network. The actual file will have the number of images trained added to the end. Upon reload the last number in the filename will be used to determine how many images have already been trained.

4.3 Neuron and Synapse Model

Synapse implementation

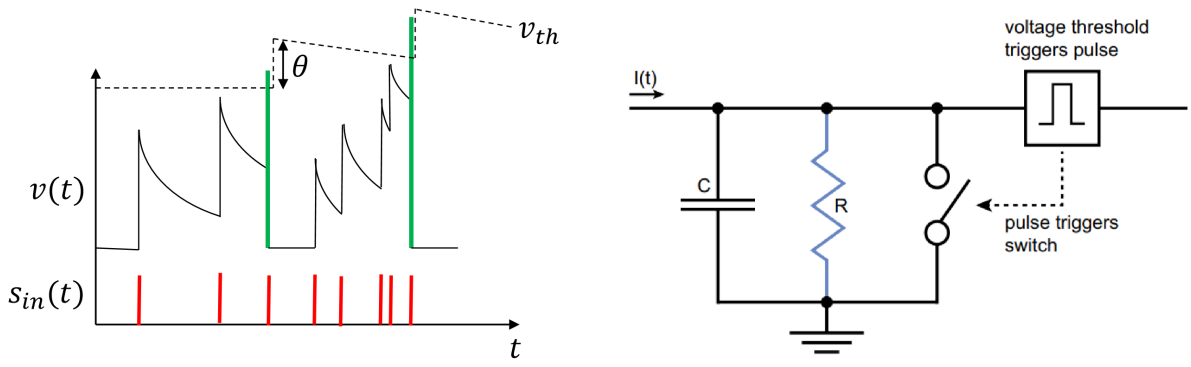
The synapses between input and output are **conductance synapses** that could represent any memristor technology discussed during this course. The weight is thus represented by a **conductance** g, that may take a value between 0 and `gmax`.

For simplicity each **spike** through the synapse is set to transfer the **same charge** `Q_max` (in the case of maximum weight), and the **postneuron** is assumed to have capacitance `C_post`, giving the change of postneuron potential for each spike as

$$\delta V = \frac{Qmax}{Cpost} \cdot g/gmax. \tag{4}$$

STDP is implemented in a simplified fashion to optimize the **computation time**:

- At each **pre-neuron spike** a timer is started, represented by the variable `tpre`.
- At a **post-neuron spike**,
  - the synapse is potentiated by `learning_rate*gmax` if `tpre` $<$ `pot_win`.
  - If not then the synapse is depressed by `learning_rate*gmax*depression_domination`.
- `pot_win` defines a **time window** in which to potentiate, outside of which depression occurs.
- `depression_domination` is a parameter that allows for modifying whether potentiation or depression should be strongest.
- The **synaptic behaviour** is defined in the equations `stdp_eq`, `on_pre_eq` (run on each pre-neuron spike), and `on_post_eq` (run on each post-neuron spike).



(a) (b)

*Figure 10: (a) Adaptive threshold: For each spiking event we increase the threshold is increased, then it decays to "resting value". In this way, it is hard to have high spiking rate. This keeps overall spiking rate **uniform**. (b) The leaky integrate-and-fire (LIF) model.*

Neuron implementation

The output neurons are modelled by a differential equation of the **membrane potential** v , which describes the **leakiness** of the neuron. This is increased by each incoming pre-synaptic spike and decays otherwise towards `v_0` with the **time constant** `tau_v`.

$$\frac{dv}{dt} = -\frac{(v - v_0)}{\tau_v} \tag{5}$$

If v exceeds the threshold T the neuron spikes and v is reset to `v_0` and T is increased by `T_plus`, i.e θ as illustrated in Figure 10a. In this way, we implement an **adaptive threshold**. T decays towards it's **equilibrium** threshold `T_0` with the time constant `tau_T`

$$\frac{dv_{th}}{dt} = -\frac{(v_{th} - v_{th,0})}{\tau_{th}} \quad \text{on spike:} \quad v_{th} = v_{th} + \theta \tag{6}$$

thus the threshold will be restored after a period of **inactivity**. The **neuronal behaviour** is defined in the equations `eqs`, `thres` and `res` (run at each post-neuron spike).

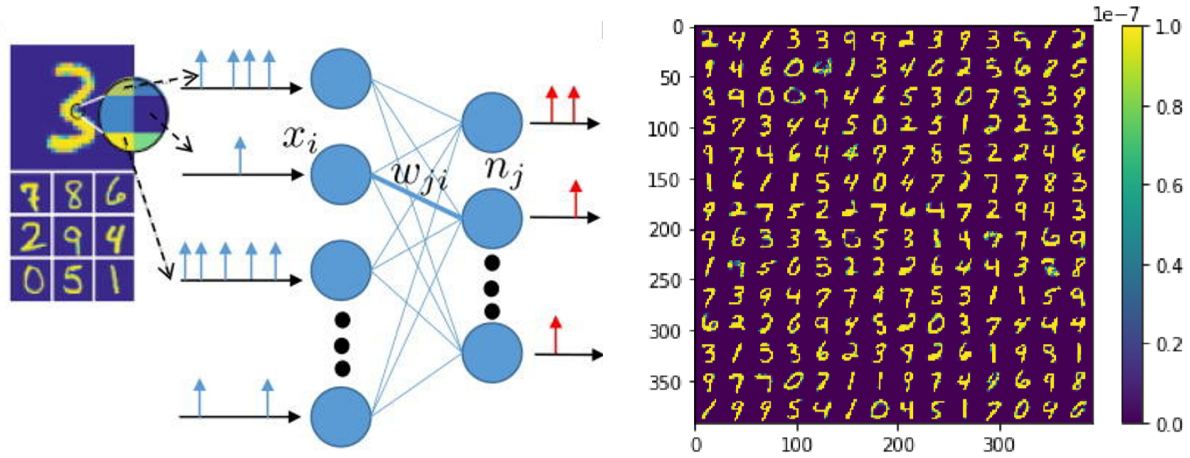
5 Tasks Part I - Classification and Evaluation

In this project, the goal is to gain deeper understanding of the workings of Spiking Neural Networks and the parameters of STDP for the successful learning of image recognition tasks. A Spiking Neural Network (SNN) represents a third generation neural network model, being a significant improvement in terms of biological realism over its predecessors [7].

We use BRIAN2 and Python to implement unsupervised learning in a simple **one-layer SNN** for the MNIST datasets. The aim is not to achieve the highest possible learning but to explore how the network depends on the various **network parameters**.

To get started, it is provided a skeleton Python code that can be used to

1. load the MNIST data.
2. set up a **baseline network** with a given set of parameters.
3. train the baseline network.
4. save the synapse weights to disk.



(a) (b)

*Figure 11: (a) A **spiking neural network** (SNN) architecture for unsupervised learning of handwritten digits. (b) The synaptic **weight map** corresponding to the* 14×14 *output neurons after training 60000 images. The graph plots the receptive field of each of the* 14×14 *output neurons next to each other. Thus each pixel represents one synaptic connection.*

5.1 Classification of the output neurons

Depending on the pixel value, the information from each input neuron (image) can be encoded into the rate of neuron spiking, i.e. conversion of the intensity images to spike-trains. In this case, the class is predicted from a neuron that spikes with highest rate.

After having trained the network with a given **set of parameters**, here we need to determine which class of images each neuron in the output layer has specialized on.

Typically one can see this by eye in the synapse map (Figure 11b). Here it is clear that each neuron has specialized on a particular class of images (for example the number "3"). It is likely that this means that the **spiking frequency** of a particular neuron will thus be highest for a certain class of images.

One can therefore "label" each neuron as belonging to a specific class, which allows us to later test the predictions of the network on the **testing** data set.

To count the spikes in the network, we may use a `SpikeMonitor` object in Brian2. For instruction on how to use this type of object can be read in BRIAN2 documentation [15].

The determination of **neuron class** is solved by simulating the SNN using Python and the Brian2 simulator, by following the steps described in the pseudo-code below:

```
Set learning rate = 0 and fix thresholds
For a random subset (almost 1000) of the training data set:
        Present image for 250 ms
        For each output neuron, count the number of spikes:
                Save result in 10x1 array (one slot for each class)
For each output neuron:
        determine class as the one with the most spikes
save all neuron class belongings in array
```

In other words, what need to implement is the following:

1. **count the number of spikes for each neuron** during each of the images we use.
2. **put the number of spikes for each neuron in the corresponding slot** each time we test with an image. For each neuron, we then have **one slot in an array per label**, i.e. 10 labels for the numbers 0,1,2,3...,9.
3. **see for which label each neuron has spiked the most**, this will then be the "label" of the neuron.

```
---- Classification ----

Loaded network from file: network_N=196_latInh=2_60000
[[2. 6. 3. 2. 7. 1. 1. 9. 9. 1. 3. 4. 3. 0.]
 [4. 5. 4. 2. 4. 5. 4. 9. 6. 5. 5. 4. 5. 4.]
 [8. 6. 4. 7. 1. 9. 1. 6. 7. 2. 3. 3. 4. 7.]
 [2. 1. 9. 6. 7. 1. 4. 7. 1. 1. 1. 7. 1. 5.]
 [6. 1. 6. 2. 3. 6. 8. 4. 0. 9. 4. 7. 1. 0.]
 [5. 9. 5. 0. 6. 0. 4. 3. 2. 3. 4. 1. 3. 4.]
 [9. 1. 6. 3. 3. 7. 3. 6. 5. 9. 4. 3. 7. 7.]
 [7. 5. 9. 8. 9. 5. 0. 7. 1. 0. 8. 7. 4. 2.]
 [9. 2. 7. 7. 6. 3. 3. 5. 3. 8. 2. 6. 0. 4.]
 [7. 7. 2. 5. 7. 4. 4. 4. 4. 4. 7. 4. 6. 2.]
 [7. 4. 4. 3. 9. 1. 3. 0. 6. 9. 2. 7. 1. 4.]
 [1. 7. 5. 5. 6. 8. 3. 8. 1. 2. 7. 7. 9. 2.]
 [1. 3. 5. 9. 9. 4. 0. 2. 3. 7. 9. 2. 2. 7.]
 [4. 1. 7. 2. 3. 1. 6. 2. 7. 8. 2. 9. 5. 7.]]
 ---- Classification IMPLEMENTED ! ----
```

5.2 Accuracy measurement - Testing

Once we have labelled each neuron as belonging to a certain class, we need to test the **predictions** of the network by running through the **testing dataset**, similarly as it is done for the training data set.

Here also we need to keep `learning_rate = 0` and thresholds fixed. Thus, for each image in the testing dataset, we need to:

1. run the image through the network.
2. see which neurons spike the most.
3. check which "label" these correspond to.
4. draw a conclusion on which type of image it is.

This is implemented in Python/Brian2 as in the following steps:

```
For each image in test_data:
        Present_image for 250 ms.
        Create empty 10x1 array -> P
        For each neuron:
                Check number of spikes and save into P[this_neuron_class]
        Prediction = index of maximum nbr of spikes in P
        If prediction == true label of image:
                        Add one to nbrOfCorrect
Accuracy = nbrOfCorrect/10000
```

After implementing the algorithm in Python and being simulated with Brian2, we obtain that the number of correct classifications is 7871 and consequently the test accuracy is: 0.7871.

6 Tasks Part II - Tuning and Training Tasks

In this section, we need to choose and solve at least three of the various training tasks described below. In this way, we can optimize our baseline network implemented above in Part I and hopefully can improve the classification rate.

In the current implementation, training on 1 image takes 1.3 s on a laptop from 2016, which makes that 1 epoch of training spends roughly 22 h. Thus, we should take this issue into account when we plan the training tasks, and study carefully which parameter should be varied between training session. Do we need to train fully or can we draw conclusions from the synapse maps (Figure 11b) already after a few 1000 images?

Monitoring spikes using the `monitorSpikes` option makes the simulation run somewhat slower, but it is useful to check that the network is responding at a reasonable rate to the input.

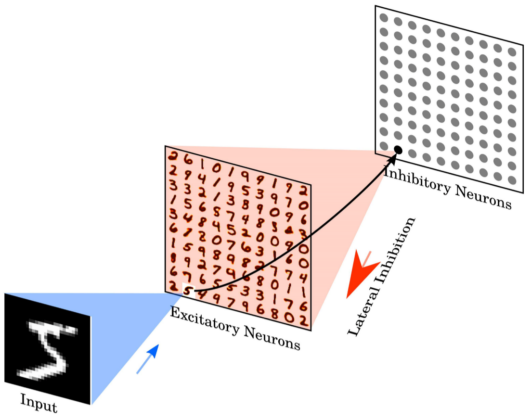


*Figure 12: Illustration of the Spiking neural network structure to be used with output neurons' **lateral inhibition**. When an output neuron spikes, it sends inhibitory signals to the other output neurons of the layer that prevent them from spiking during the inhibition time and resets their potential to zero.*

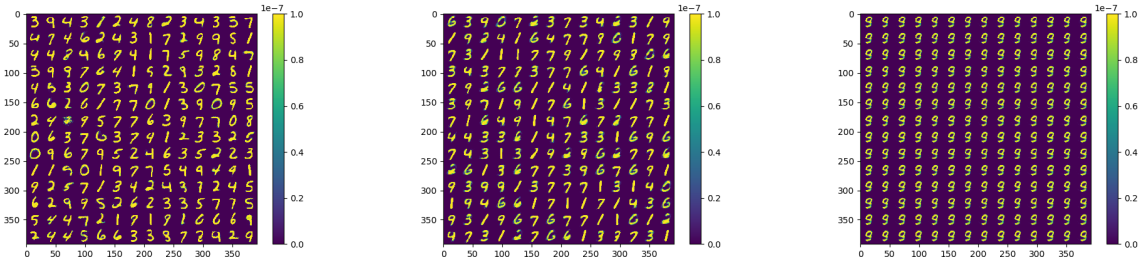
6.1 Type of lateral inhibition

In simulation, **Excitatory cells** receive input and the neuron receiving the most input activation is the first to reach its spiking threshold. Spiking excites the **inhibitory cell**, which in turn prevents other cells from responding. The code implements two types of lateral inhibition, **soft lateral inhibition** and **winner takes all**. Lateral inhibition promotes neuronal learning of **specific features** in the data, since only a few neurons can spike and adjust their **receptive field** towards a specific prototype. Lateral inhibition also prevents **overfitting** as the competition between neurons forces them to learn different features/prototypes.

In this task, we need to Vary the type `latInh` and see how it changes the dynamics of the network. At the end, we try to turn lateral inhibition off completely. For studying how lateral inhibition affects the learning accuracy and the synapse maps, three networks are trained with different kinds of lateral inhibition, as shown in Table 1.

N_out neurons	Lateral inhibition	N_input images	Accuracy
196	winner-takes-all	60000	76.64 %
196	soft	60000	52.40 %
196	disabled	4000	-

Table 1: Classification performance of the trained networks with different kinds of lateral inhibition.



(a) (b) (c)

Figure 13: Synaptic field for a) winner-takes-all lateral inhibition, b) soft lateral inhibition, c) no lateral inhibition

```
[[951   0   3   0   0   5  21   0   0   0]
 [210 861  13   5   0   1   7   1   7  30]
 [ 59   5 910   3   7   2  12  19  14   1]
 [ 40   1  33 812   4  79   2   6  21  12]
 [ 16   2   6   0 655   7  95   1   4 196]
 [141   0   0  61  14 596  31   1  36  12]
 [121   1   0   0   2  14 820   0   0   0]
 [ 49  10  47   5  13   1   4 828   4  67]
 [ 80   0  53  78   8  90  46  25 563  31]
 [ 44   1   4   8 213  11  13  36  11 668]]
```

(a)

```
[[876   0   3  81   2   0   8   8   1   1]
 [214 861   7  33   2   0  11   1   2   4]
 [132  14 466 294   3   0  97  17   2   7]
 [ 74   0   4 907   0   0   5  10   1   9]
 [ 41   3   4   8 427   0 139  16   1 343]
 [237   3   4 454  14   0  47  16  17 100]
 [253   1  36  10 113   0 544   0   0   1]
 [ 69  16  22  21  29   0   7 800   7  57]
 [ 32   2  36 690   8   0  66  30  25  85]
 [ 37   3   4  55 235   0  21 113   7 534]]
```

(b)

Figure 14: Confusion matrices for a) winner-takes-all lateral inhibition, b) soft lateral inhibition

The synaptic fields for the trained networks are shown in Figure 13. Training with winner-takes-all lateral inhibition means that only one neuron fires for every image seen. It takes more images to make the neurons start to sufficiently detect the classes but it results in an even split of the classes for each neuron, to some extent. In the confusion matrix in Figure 14a this is seen, where every class is being correctly predicted for a majority of the test images. Although, '5':s and '8':s are slightly harder for the network to classify. A higher accuracy might be reachable if tested on a network where more neurons were specializing on those classes, since the classes of the neurons are switched throughout training.

On the other hand, soft lateral inhibition results in neurons that sufficiently detects classes after much fewer images are seen. The problem is that many of the neuron detects the same classes and more images are needed to be seen to make the neurons detect all classes. It is still a problem after training with 60000 images. Not a single neuron is labelled as a '5' and only 6 neurons are labelled as an '8'. This is

seen in the confusion matrix in Figure 14b where most '8':s are being classified as '3':s. Also, since there are no neurons specializing in '5':s there are no predictions on that class. This problem might be fixed with more training examples or a larger number of output neurons.

Disabling lateral inhibition results in all neurons detecting the same class since there is nothing from stopping them from all spiking. This is a totally worthless classifier and would result in an accuracy of about 10%.

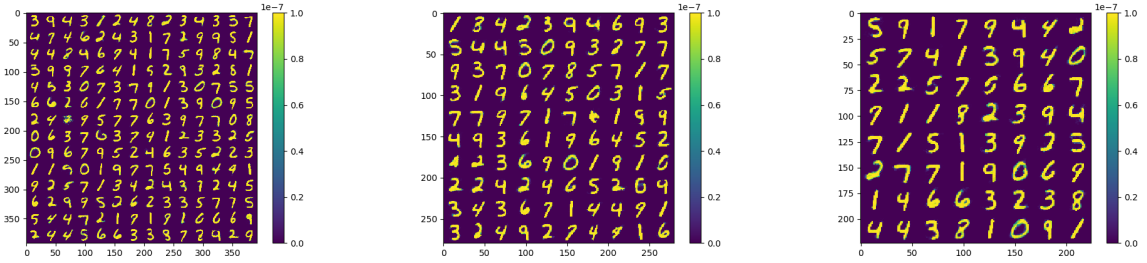
6.2 Network size

In this task, we need to try varying the number of **output neurons** by the variable `N_out` and to study how this affects the overall spiking activity. In this way, we investigate the effect of having a **large network** on the classification accuracy and learning speed.

Three networks with different amount of output neurons were trained according to Table 2. The networks were evaluated fully trained (60 000 images seen) and not fully trained (15 000 images seen).

N_out neurons	lateral Inhibition	N_input images	Accuracy
196	winner-takes-all (2)	60000	76.64 %
100	winner-takes-all (2)	60000	72.88 %
64	winner-takes-all (2)	60000	72.27 %
196	winner-takes-all (2)	15000	69.7 %
100	winner-takes-all (2)	15000	68.2 %
64	winner-takes-all (2)	15000	65.0 %

Table 2: Training results: Comparing the classification performance of the network by exploring the effect of varying the number of output neurons, i.e. by altering the parameters `N_out`. The first 3 were tested on 10000 images. The latter 3 were only tested on 1000 images.



(a) (b) (c)

Figure 15: Synaptic field with 60 000 trained images for a) 196 output neurons, b) 100 output neurons, c) 64 output neurons

The synaptic fields of the three networks trained are shown in Figure 15. All networks seem fully trained. As seen in Table 2, the accuracy is decreasing for smaller amount of output neurons. Using 64 and 100 output neurons resulted in very similar test accuracies. Using 64 output neurons might be acceptable for this simple dataset but if the dataset consisted of more classes or more inner class variations it would not. The network needs to have enough neurons to represent each class and the different variations each class can possess. Generally a larger amount of output neurons will result in higher accuracies.

The downside of using more output neurons is that it takes longer to train using winner-takes-all lateral inhibition. Because only one neuron spikes for every image, more input images are required for each neuron to spike enough to reach a good accuracy. After only 15000 images seen, the networks were not fully trained. The largest network had many neurons that had not clearly started specializing in a class yet and the smallest network had only a few. Still the larger network outperformed the smaller networks but with a smaller margin compared to the networks trained on more images. This indicates that larger networks are better but if amount of training images are very limited it might be worth to consider a smaller one.

6.3 Adaptive threshold

Here in this task, we need to investigate the actual effect of having an adaptive threshold on the output neurons. It can be especially useful in case where there is **device-to-device variation** in neuron threshold.

We can obtain **uniformly distributed** thresholds between 40 and 60 mV, by setting a **random distribution** of thresholds around the usual value of equilibrium threshold (on line 156), i.e.

```
O.T_0 = 'v_0 + 50.0*mV' # equilibrium threshold
```

as following

```
O.T_0 = 'v_0 + (40.0+rand()*10.0)*mV'
```

N_out neurons	adaptive threshold	N_input images	Accuracy
196	with	60000	79 %
196	without	60000	83 %

Table 3: Training results: Comparing the classification performance of the network with and without adaptive threshold.

To compare the classification performance of this network with and without adaptive threshold, we adjust `res` function such that not to change the threshold when spiking, i.e. `res = '''v=v_0'''`

```
1 ------- Adaptive Testing -------
2  number of correct:  7888
3 Test accuracy is:  0.7888
4 ------- Adaptive Testing IMPLEMENTED ! ------
```

After training the "adaptive" network and simulating it in Python/Brian2, we get the obtained labels of the output neurons in the following:

```
1 ------ Adaptive Classification ------
2 Loaded network from file: network_N=196_latInh=2_60000
3
4 [[3. 6. 8. 5. 1. 5. 6. 9. 0. 8. 4. 9. 1. 0.]
5  [3. 5. 9. 6. 2. 8. 3. 9. 1. 1. 2. 4. 3. 2.]
6  [1. 2. 8. 5. 5. 1. 0. 9. 4. 9. 5. 9. 7. 8.]
7  [6. 4. 2. 4. 5. 6. 1. 3. 6. 1. 8. 5. 5. 7.]
8  [7. 5. 4. 2. 6. 3. 8. 4. 7. 9. 4. 1. 3. 7.]
9  [4. 1. 7. 5. 2. 1. 9. 2. 0. 4. 4. 9. 3. 7.]
10 [6. 7. 0. 8. 4. 6. 6. 4. 9. 4. 5. 8. 6. 4.]
11 [2. 0. 5. 1. 4. 7. 7. 6. 5. 1. 1. 1. 8. 3.]
12 [7. 4. 1. 3. 5. 1. 1. 3. 4. 2. 3. 4. 4. 2.]
13 [9. 5. 6. 3. 3. 8. 0. 3. 4. 4. 4. 5. 0. 3.]
14 [9. 5. 7. 9. 0. 5. 1. 7. 5. 2. 2. 2. 9. 2.]
15 [9. 6. 1. 2. 7. 0. 7. 8. 7. 1. 4. 6. 6. 2.]
16 [5. 0. 7. 8. 3. 9. 3. 7. 3. 7. 6. 4. 2. 4.]
17 [0. 4. 2. 2. 0. 7. 2. 7. 5. 8. 7. 4. 3. 0.]]
18
19  ------ Adaptive Classification IMPLEMENTED ! ------
```

```
1 ------- Without Adaptive Testing -------
2 number of correct:  8259
3 Test accuracy is:  0.8259
4 ------- Without Adaptive Testing IMPLEMENTED ! ------
```

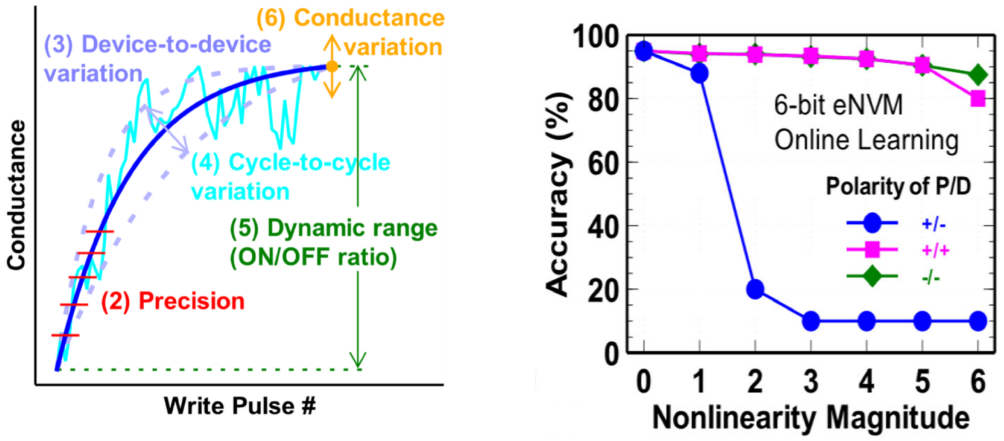
```
1  ------ Without Adaptive Classification ------
2
3 Loaded network from file: network_N=196_latInh=2_60000
4 [[2. 8. 4. 5. 9. 4. 4. 0. 8. 3. 9. 1. 1. 1.]
5  [0. 7. 5. 7. 8. 8. 1. 3. 4. 2. 1. 8. 0. 1.]
6  [7. 2. 6. 2. 0. 7. 8. 9. 2. 2. 5. 5. 8. 6.]
```

```
[8. 4. 3. 0. 2. 3. 4. 3. 7. 1. 8. 5. 0. 8.]
[8. 8. 9. 7. 2. 9. 0. 1. 4. 4. 9. 8. 3. 0.]
[4. 2. 9. 2. 1. 6. 6. 3. 9. 3. 0. 4. 4. 5.]
[3. 9. 7. 0. 9. 0. 5. 2. 3. 3. 0. 4. 2. 2.]
[4. 7. 3. 8. 6. 6. 0. 5. 6. 7. 2. 5. 9. 1.]
[8. 7. 4. 0. 5. 5. 8. 2. 8. 6. 2. 8. 1. 1.]
[4. 3. 5. 6. 5. 4. 2. 2. 5. 8. 6. 9. 3. 6.]
[0. 1. 5. 6. 8. 6. 3. 8. 0. 7. 7. 5. 6. 1.]
[4. 1. 7. 2. 9. 9. 1. 4. 8. 3. 4. 4. 2. 0.]
[4. 4. 2. 6. 0. 1. 6. 2. 3. 0. 5. 5. 0. 9.]
[3. 0. 0. 2. 8. 7. 4. 8. 5. 2. 0. 1. 1. 9.]]
——— WITHOUT Adaptive Classification IMPLEMENTED ! ———
```

6.4 Impact of nonlinear memristor response

As reported in the literatures, *the linearity in weight update refers to the linearity of the curve between the device conductance and the number of identical programming pulses.* Ideally, this should be a linear relationship for the **direct mapping** of the weights in the algorithms to the conductance in the devices. However, the resistive synaptic devices generally have the nonlinearity in weight update.

This nonlinearity is undesired because the **weight change** (Δw) depends on the current weight (w), or in other words, the weight update has a **history dependence**. Recent results have shown that this nonlinearity has caused the learning accuracy loss in the neural networks [3].



(a) (b)

Figure 16: (a) Non-ideal synaptic device properties: (2) weight precision, (3) device-to-device weight update variation, (4) cycle-to-cycle weight update variation, (5) dynamic range (conductance ON/OFF ratio) and (6) conductance variation. (b) Non-linear weight update impact on the accuracy. [9]

Training task

In the default implementation, the synapse is linearly potentiated/depressed. A real synaptic device often does not respond linearly but rather exponentially to an **STDP update**, as shown in Figure 16a.

In this task, we need to explore the effect of such a behaviour by altering the parameters `dgp` and `dgd` to:

$$\texttt{dgp} = \texttt{learning\_rate} \cdot \texttt{gmax} \cdot \left(1 - \exp\left(\frac{g - \texttt{gmax}}{\texttt{gmax}}\right)\right) \tag{7}$$

$$\texttt{dgd} = \texttt{learning\_rate} \cdot \texttt{gmax} \cdot \left(1 - \exp\left(\frac{g}{\texttt{gmax}}\right)\right) \tag{8}$$

To solve this task we redefine these parameters in the `on_post_eq` string in the weight update as shown below

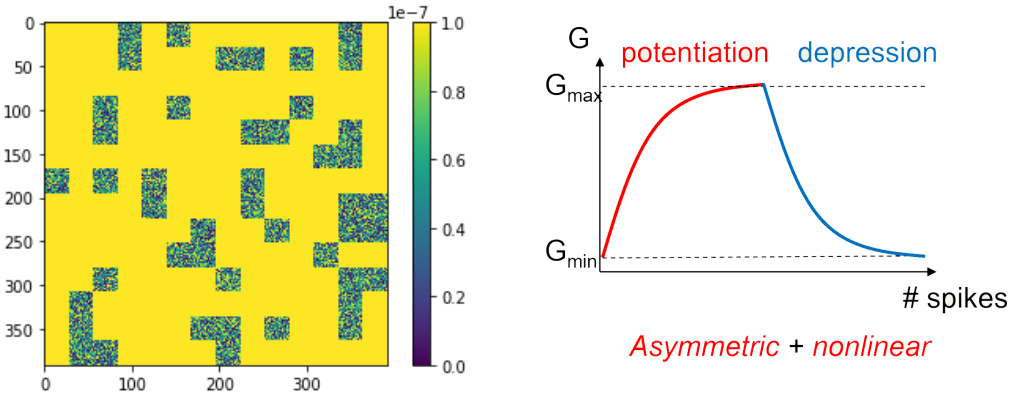
```
on_post_eq = '''
    dgp = learning_rate*gmax*(1-exp((g-gmax)/gmax))
```

```
3       dgd = learning_rate*gmax*(1-exp(g/gmax))
4       g = clip(g + dgp*(tpre <= pot_win)-dgd*(tpre>pot_win),0*nS, gmax)
5       '''
```

Results: what is the impact on the classification accuracy?

As expected the weight update non-linearity affect both the final training results and training convergence speed. In fact, this training was slower and two examples of resulting weight map is shown in Figure 17a. Then after evaluating the network the result shows that nonlinear weight update has crucial impact and degrade the learning accuracy to zero, if our implementation using Brian2 is right.

As illustrated in Figure 16b, produced in [9], the result shows that the **asymmetry** (positive potentiation P and negative depression D) is the key factor that degrades the accuracy, and high non-linearity can be tolerated if P/D have the **same polarity**. However, for common situations where P/D is positive/negative, the impact of nonlinearity on the online learning accuracy is very critical. High accuracy can only be achieved with small nonlinearity (< 1).



(a) (b)

Figure 17: (a) Impact of Non-linear weight update on training during the simulation. The resulting weight map at the end of the training. (b) Non-linear potentiation/depression behaviour and asymmetry similar to what is often observed in real memristors: the trajectory of the long-term-potentiation (LTP) process that increases the conductance differs from that of the long-term-depression (LTD) process that decreases the conductance. The weight tends to saturate at the end of LTP or LTD processes.

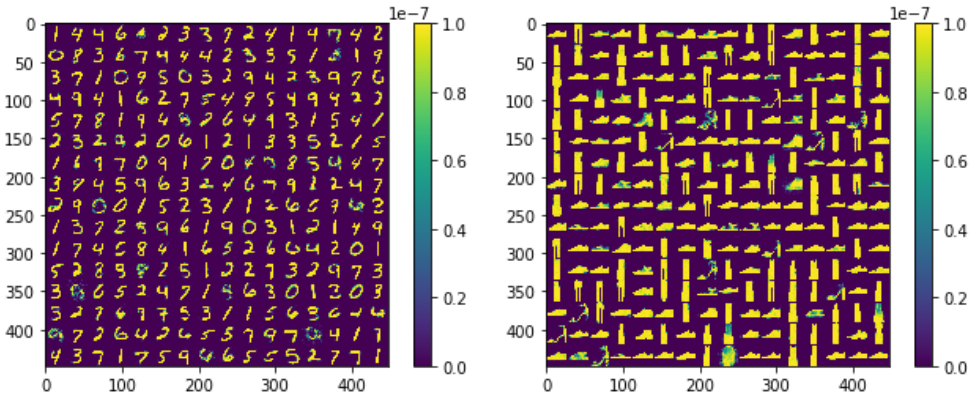
To remedy the effect of non-linearity, it is important to fix the asymmetry issue of the memristive memory device whereas we can live with the other non-idealities shown in Figure 16a. As summarized in [9], a symmetric and close to linear weight update with sufficient ON/OFF ratio is critical, while a reasonable amount of deviceto-device, cycle-to-cycle variations could be tolerated.

6.5 MNIST Fashion dataset

At the end, after optimizing our network, in this task we need to train the network, with the same parameters, on the MNIST Fashion dataset. MNIST Fashion is a compilation of images of clothes from the Zalando database.

Compared to MNIST numbers, we need to investigate how the network fare on Fashion. First it is worthy to remember that it is much harder to get good classification results on this dataset than MNIST digits, mainly due to to small **inter-class variation**. A good image classification model must retain sensitivity to the **inter-class variations**. After training the network with 16×16 output neurons and winner-takes-all lateral inhibition we obtain accuracy of 37%, as shown in the output of the simulation below.

```
1 ------- FASHION Testing -------
2 Loaded network from file: network_N=256_latInh=2_60000
3 number of correct:  3736
4 Test accuracy is:  0.3736
5 ------- FASHION Testing IMPLEMENTED ! -------
```



(a) (b)

Figure 18: Comparing between the weight map from MNIST digits (a) and that from Fashion MNIST in (b).

```
------ FASHION Classification ------

Loaded network from file: network_N=256_latInh=2_60000
[[7. 7. 0. 7. 3. 5. 7. 5. 7. 3. 7. 7. 3. 7. 1. 7.]
 [1. 3. 7. 3. 7. 3. 3. 5. 7. 9. 1. 7. 7. 7. 3. 0.]
 [3. 7. 5. 3. 7. 7. 7. 1. 7. 7. 0. 3. 5. 7. 9. 3.]
 [3. 7. 0. 5. 3. 7. 7. 1. 0. 0. 5. 0. 3. 0. 3. 7.]
 [3. 3. 7. 5. 5. 3. 7. 5. 3. 3. 7. 0. 3. 7. 5. 3.]
 [3. 7. 5. 7. 3. 5. 9. 7. 7. 7. 7. 9. 5. 3. 1. 7.]
 [3. 3. 5. 3. 7. 3. 7. 3. 7. 3. 3. 3. 0. 7. 7. 3.]
 [7. 1. 3. 0. 7. 7. 1. 7. 3. 7. 1. 7. 3. 3. 7. 5.]
 [3. 7. 0. 7. 7. 5. 7. 1. 7. 7. 7. 7. 3. 7. 7. 7.]
 [7. 5. 7. 1. 5. 5. 7. 7. 5. 5. 7. 7. 7. 3. 7. 7.]
 [1. 7. 0. 9. 7. 1. 7. 3. 7. 7. 5. 0. 7. 0. 3. 7.]
 [3. 5. 7. 3. 5. 1. 3. 5. 3. 0. 3. 3. 3. 3. 7. 7.]
 [1. 1. 1. 1. 7. 1. 3. 5. 7. 3. 1. 3. 3. 5. 7. 3.]
 [7. 5. 4. 7. 7. 7. 3. 7. 3. 7. 9. 3. 0. 3. 3. 0.]
 [5. 7. 7. 3. 9. 3. 7. 3. 0. 7. 7. 3. 3. 3. 7. 3.]
 [7. 0. 5. 5. 7. 7. 7. 3. 9. 7. 7. 3. 3. 7. 7. 7.]]
------ FASHION Classification IMPLEMENTED ! ------
```

7 Discussion and conclusions

After working on this project we can notice the difficulty in simulating and training SNNs in terms of time consuming. Perhaps we can say that slow training speed is one of the disadvantages of on-chip training using neuromorphic systems.

In this project, we have studied and practised on training and testing a Spiking Neural Network (SNN) employing a Non Volatile Memory (NVM), memristor, as artificial synapses interconnections represented by a conductance g. In this way, we could also gain better understanding about the STDP learning rule and signal accumulation performed by leaky Integrate-and-Fire (LIF) neuron model.

In the first part of the project in section 5 the SNN required a training phase simulated using Brian2 in order to calculate the classification parameters for images from MNIST dataset. Then, in a testing phase we run the SNN, with the calculated parameters, on the unseen images in the test dataset and obtain an accuracy of 79 %.

In the second part of this work in section 6, we explored how the network performance depends on tuning various parameters of the implemented SNN. The performance can be improved by increasing the size of the output `N_out` and having lateral inhibition of type Winner-takes-all. Specifically, with 16×16 ouput layer and increasing the time constant `tau_v` to 40 ms the accuracy is improved to 80 %, as shown in the simulation output below.

```
1 ------- Optimized Testing -------
2 Loaded network from file: network_N=256_latInh=2_60000
3  number of correct:  8040
4 Test accuracy is:  0.804
5 ------- Optimized Testing IMPLEMENTED ! -------
```

However, as reported in [10], the classification performance of the neural network was found to be degraded since the conductance weights became distorted due to the **degradation characteristics** of the synapse device, i.e. it is difficult to minimize the variability and maximize the conductance ratio, in addition to the core requirements of symmetric and linear conductance changes.

8 Appendix

References

[1] Mattias Borg, Memory Technologies For Machine Learning - EITP25, Lectures notes: `https://canvas.education.lu.se/courses/2101`

[2] Chen, Hutchby, Zhirnov, Bourianoff, *Emerging Nanoelectronic Devices*, Wiley 2015.

[3] Yu Shimeng, *Neuro-inspired Computing Using Resistive Synaptic Devices*. Springer, 2017, ISBN 978-3-319-54312-3, ISBN 978-3-319-54313-0 (eBook).

[4] Manan Suri, *Advances in Neuromorphic Hardware Exploiting Emerging Nanoscale Devices*. Springer, 2017, ISBN: 978-81-322-3701-3.

[5] P. Diehl, M. Cook, Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Frontiers in Computational Neuroscience, Institute of Neuroinformatics, ETH Zurich and University Zurich, Zurich, Switzerland: `https://www.frontiersin.org/articles/10.3389/fncom.2015.00099/full`.

[6] G. Burr, R. Shelby, A. Sebastian, et al. Neuromorphic computing using non-volatile memory. ADVANCES IN PHYSICS:X,2017.

[7] Wolfgang Maass, Networks of Spiking Neurons: The Third Generation of Neural Network models, 1996.

[8] Abu Sebastian, Manuel Le Gallo, et al. Tutorial: Brain-inspired computing using phase-change memory devices. JOURNAL OF APPLIED PHYSICS 124, 111101 (2018).

[9] Raisul Islam et al., Device and materials requirements for neuromorphic computing. Journal of Physics D: Applied Physics, 2019.

[10] K. Moon, S. Lim, J. Park, et al. RRAM-based synapse devices for neuromorphic system., Faraday Discussions, 2019.

[11] Andy Thomas, Memristor-based neural networks, 2013 J. Phys. D: Appl. Phys. 46 093001.

[12] Action Potential, Lumen Module 14: The Nervous System. `https://courses.lumenlearning.com/wm-biology2/chapter/action-potential/`

[13] VanderPlas, A Whirlwind Tour of Python. O'Reilly 2016. Online reading: `https://jakevdp.github.io/WhirlwindTourOfPython/`

[14] Geron, Jupyter notebook on linear algebra from Machine Learning and Deep Learning in Python using Scikit-Learn, Keras and TensorFlow: `https://github.com/ageron/handson-ml2`

[15] Brian2 Documentation: `https://brian2.readthedocs.io/en/stable/index.html`.

[16] Lumen Leanrning, Resting Membrane Potential: `https://courses.lumenlearning.com/wm-biology2/chapter/resting-membrane-potential/`.

[17] Neuronal Dynamics, Online book. The Hodgkin-Huxley Model: `https://neuronaldynamics.epfl.ch/online/Ch2.S2.html`.

[18] Harvard Extention School, Action Potential in the Neuron: https://youtu.be/oa6rvUJlg7o.

[19] Ren Hartung, Neuron Action Potential for Beginners. Lecture on Youtube: https://www.youtube.com/watch?v=8g1RBj03Syo&feature=emb_rel_end.

[20] Brian Hays. The Memristor: https://www.americanscientist.org/article/the-memristor, American Scientist.

[21] L. F. Abbott and Sacha B. Nelson, Synaptic plasticity: taming the beast. 2000 Nature America Inc.

[22] D. Querlioz, et al, Bioinspired networks with nanoscale memristive devices that combine the unsupervised and supervised learning approaches, IEEE/ACM International Symposium on Nanoscale Architectures, 203–210 (2012).

[23] Spiking neural networks: https://towardsdatascience.com/spiking-neural-networks-558dc4479903.