EITP25 Group Project Instructions

Mattias Borg 2020-04-27

1. Introduction

In this project the aim 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. You will be using 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 you started, you can download the template Python code from Canvas. This code as it is written right now will load the MNIST data, set up a network with a given set of parameters, as well as train the network and save the synapse weights to disk. What is left to implement is to determine the classes that each neuron in the network map to after training, and to implement evaluation of the accuracy of the network on the test dataset.

In the following the implementation of the network will be detailed, as well explaining the tasks for the group project.

2. Network Implementation

The SNN implementation here follows that of Diehl and Cook (Front. Comp. Neurosci. 9, 99 (2015)), with some modifications. In the initial implementation it consists of 28x28 input neurons 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.. You are free to vary N_{out}, but to simplify plotting of synapse weights you should make sure that $\sqrt{N_{out}}$ is an integer. 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`. See Figure 1 for a graphical representation of the network structure.

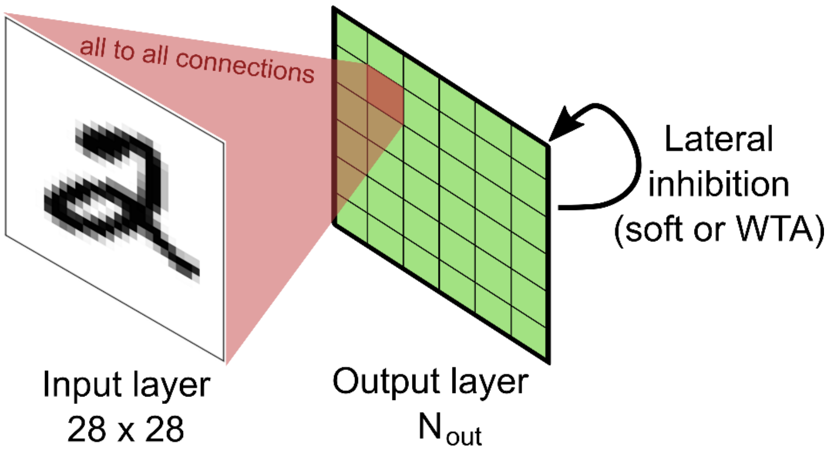

Figure 1. Illustration of the Spiking neural network structure to be used in the project.

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']`.

3. Code walkthrough

“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:

Table 1. General Parameters

Parameter name	**Description**
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.

3.1. 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 image could be named “network_N100_WTA_2000”. The numbers at the end are added automatically as the network is saved during training. If you are clever you 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.

3.2. 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 post-neuron is assumed to have capacitance `C_post`, giving the change of postneuron potential for each spike as $\Delta V = \frac{Q_{max}}{C_{post}} * g/g_{max}$.

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 of 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).

3.3. Neuron implementation

The output neurons are modelled by a potential *v*, which is increased by each incoming presynaptic spike and decays otherwise towards `v_0` with the time constant `tau_v`. If v exceeds the threshold *T* the neuron spikes and v is reset to `v_0` and *T* is increased by `T_plus`. This thus implements an adaptive threshold. *T* decays towards it's equilibrium threshold `T_0` with the time constant `tau_T`, 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).

4. Class determination

After having trained a network with a given set of parameters it is time 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 2). 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. You can therefore "label" each neuron as belonging to a specific class, which allows you to later test the predictions of the network on the testing data set.

To count the spikes in the network, you may use a SpikeMonitor object. Please refer to the BRIAN2 documentation for information on how to use this type of object.

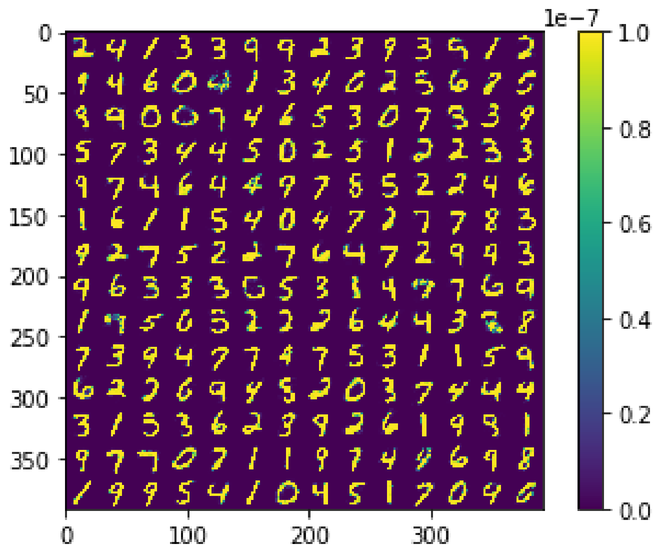

Figure 2. Example of a Synapse map after 60000 images trained. The graph plots the receptive field of each of the 14x14 output neurons next to eachother. Thus each pixel represents one synaptic connection.

You may of course implement the determination of neuron class in whatever way you feel is best. However, one option is described in pseudo-code below:

```
Set learning rate = 0 and fix thresholds
For a random subset (~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
```

5. Testing

Once you have labelled each neuron as belonging to a certain class, test the predictions of the network by running through the testing data set, similarly as is done for the training data set. Make sure to keep learning_rate = 0 and thresholds fixed. For each image check which output neurons generate pulses and by some algorithm determine which image class the network predicts it to be. Again you are free to come up with any smart way of doing this, below is just one example:

```
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
```

6. Training Tasks

In the current implementation training on 1 image takes 1.3s on a laptop from 2016, which gives 1 epoch of training in roughly 22 h. Take this into account when you plan your training tasks. i.e. **prepare well before setting out to train your network**. Discuss carefully which parameter should be varied between training session? Do you need to train fully or can we draw conclusions from the synapse maps (Figure 2) already after a few 1000 images? Monitoring spikes using the `monitorSpikes` option makes the simulation run somewhat slower, but is useful to check that the network is responding at a reasonable rate to the input.

Below follows a description of the various training tasks to do. Please choose at least three for the project:

6.1. Type of lateral inhibition

The code implements two types of lateral inhibition, soft lateral inhibition and winner takes all. Vary the type and see how it changes the dynamics of the network. Also try to turn lateral inhibition off completely. How does this change the synapse maps? How does lateral inhibition affect the learning accuracy?

6.2. Network size

Test varying the number of output neurons by the variable `N_out`. How does this change the overall spiking activity? How does it change the time it takes for clear patterns in the synapse map to evolve? Is a large network needed to achieve high classification accuracy? Why do you think that is?

How fast does it learn at different sizes? You can test the network performance at different stages to see this.

6.3. Adaptive threshold

What is the actual effect of having an adaptive threshold of the output neurons? It can be especially useful in case where there is device to device variations in neuron threshold. Investigate this by setting a random distribution of thresholds around the usual value on line 156, such as:

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

Which gives uniformly distributed thresholds between 40 and 60 mV.

Compare the classification performance of this network with and without adaptive threshold. i.e. you can adjust `res` function so as to not change the threshold when spiking.

6.4. Non-linear memristor response

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. See Figure 3.

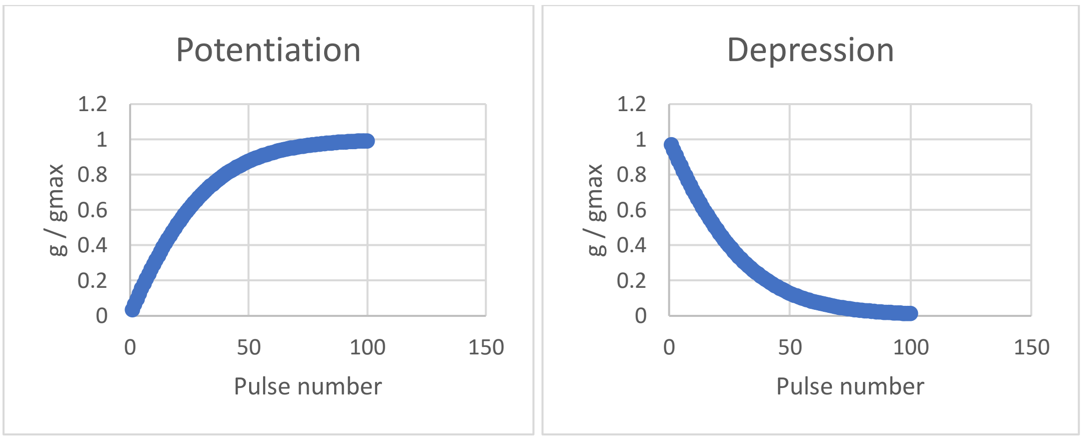

Figure 3. Non-linear potentiation and depression behaviour similar to what is often observed in real memristors.

Explore the effect of such a behaviour by altering the parameters the `dgp` and `dgd` to:

$$dgp = learning\_rate * gmax * \left(1 - \exp\left(\frac{g - gmax}{gmax}\right)\right)$$

$$dgd = learning\_rate * gmax * \left(1 - \exp\left(\frac{g}{gmax}\right)\right)$$

What is the impact on the classification accuracy? Why do you think that is?

6.5. Fashion dataset

As a final task after having optimized your network, change to the Fashion data set and train the network on this instead with the same parameters. MNIST Fashion is a compilation of images of clothes from the Zalando data base. Compared to numbers, how does the network fare on Fashion? If not as good, why do you think that it is a harder data set?

7. Reporting

As a group please write a report that describes your experiments. Please be sure to explain what parameters you have changed and any observations you make on the effect of changing them. As is usual for a scientific report, put most focus on the discussion of your results. The description of the tasks in Section 6 give you some hints about what questions to look for answers to. Figures and Tables should have captions and be referred to in the text.

Excluding figures, a length of about 5 pages seems appropriate for the report. Please submit the report through Canvas as .docx or pdf format. Also include your source code files in a zipped file, and in the report describe how to understand which source file does what.