Final Report

Kaustubh Butte

BITS Pilani KK Birla Goa Campus

1 What is a Spiking Neural Network?

Spiking neural networks more closely mimic the behavior of biological neurons and are important in modelling real brains within the discipline of **Computational Neuroscience**.

A SNN is a type of neural network that deals with temporal data and produces a series of pulses as output. It is biologically more realistic than traditional ANNs. SNNs can change their topology during training via Hebbian Learning (meaning they can create and destroy connections between their neurons depending on the task).

Temporal data is just a continuous input of data with respect to time. Currently SNNs have application in brain simulations ,robotics as low powered AI control algorithms. Also for object detection in videos.

SNNs have strong generalization power of spiking neurons but they dont scale well with mainstream synchronous hardware but even this point is gradually becoming less significant after the introduction of Convolutional SNN.

2 What is Synaptic Plasticity?

Canadian psychologist Donald Hebb's postulate simply put- when two neurons fire at the same time the connections between them are strengthened and thus they become more likely to fire again in the future and when two neurons repeatedly fire in an uncoordinated manner the connections between them weaken and they are more likely to act independently in the future this can be simplified to the-cells that fire together wire together and cells that fire apart wire apart.

When an action potential reaches the end of a presynaptic cell it causes the release of neurotransmitters these cross the synaptic cleft and bind to receptors on the postsynaptic cell. These receptors can be metabolic receptors or ion channels. When a neurotransmitter stimulates a metabolic receptor it triggers a cascade of secondary messengers which may alter a number of proteins within the cell and when a neurotransmitter stimulates an ion channel it may cause it to open or close altering the flow of ions through it and causing a change in voltage in the post synaptic cell these changes can be excitatory or inhibitory.

The degree of voltage change in the postsynaptic neuron is what we mean by the strength of a connection. Strengthening of the synapse is known as long term potentiation the change in potential evoked by the presynaptic neuron will be greater and the weakening of synaptic strength is known as long term depression the change in potential will be smaller.

There is a critical window for synaptic plasticity with the peak time for changes to synaptic strength being in 20 milliseconds before and after an action potential If the presynaptic neuron fires before the postsynaptic neuron within the preceding 20 milliseconds long-term potentiation occurs and if the pre synaptic neuron fires after the postsynaptic neuron within the following 20 milliseconds long-term depression occurs this is known as spike timing-dependent plasticity and we can alter the initial hebbian hypothesis to include these new findings if the presynaptic neuron fires within a window of 20 milliseconds before the postsynaptic neuron the synapse will be strengthened however if the pre synaptic neuron fires within a window of 20 milliseconds after the postsynaptic neuron the synapse will be weakened.

Importantly these processes are localized to individual dendrites and each dendrite can be modulated individually.

3 Spiking Models

Biological neurons communicate by generating and propagating electrical pulses called action potentials or spikes. probability of firing (generating a spike) is increased by excitatory inputs and decreased by inhibitory inputs In mathematical terms a sequence of the firing times - a spike train - can be described as [4]

$$S(t) = \sum_f d(t - t^f) \tag{1}$$

Most common spiking neuron models are Integrate-and-Fire (IF) and Leaky-Integrate-and-Fire (LIF) units.

Dynamics of the LIF unit is described by the following formula-

$$C\frac{du(t)}{dt} = \frac{-1}{R}u(t) + (i_o(t) + \sum w_j i_j(t)) \tag{2}$$

The above formula describes the IF model. In both, IF and LIF models, a neuron is supposed to fire a spike at time tf , whenever the membrane potential u reaches a certain value called a firing threshold. Immediately after a spike the neuron state is reset to a new value ures¡. and hold at that level for the time interval representing the neural absolute refractory period. Arrival of a presynaptic spike at a synapse triggers an input signal i(t) into the postsynaptic neuron. This signal corresponds to the synaptic electric current flowing into the biological neuron.

Spiking neural Networks can be classified into three topologies-

1. Feedforward Neural Nertworks
2. Recurrent Neural Networks
3. Hybrid Neural Networks

Each millisecond, thousands of spikes emitted by sensory neurons are processed by the brain. Different Types of encodings-

1. Time to first spike
2. Rank order coding (ROC)
3. Latency Code
4. Resonant burst model
5. Coding by Synchrony
6. Phase coding

Of these only Rate based neural codes follow theoretically as well as experimentally. For Image classifications usually Rank Order Encoding is used.

Synaptic plasticity refers to the ability of synaptic connections to change their strength, which is thought to be the basic mechanism underlying learning and memory in biological neural networks.

4 Unsupervised Learning

By modifying synaptic strengths Hebbian processes lead to the reorganization of connections within a neural network and under certain conditions may result in the emergence of new functions, such as input clustering, pattern recognition, source separation, dimensionality reduction, formation of associative memories or formation of self-organizing maps

$\Delta w_{ji} \quad \alpha \quad v_i v_j$, where Δw_{ji} refers to the change of the strength of the synaptic coupling wji between the presynaptic neuron i and the postsynaptic cell j. The terms i,j in the Hebbs formula have traditionally been interpreted as neural firing rates.

5 Supervised Learning

Implementation of error backpropagation in spiking networks is difficult due to the complex and, in many models, discontinuous dynamics of spiking neurons. In this case indirect approaches or special simplifications must be assumed in order to estimate gradient of the error.

6 Convolutional STDP based Feature Learning in Spiking Neural Networks and their comparison with Conventional Fully connected SNNs

One important divide between humans and current machine learning systems is in the size of required training datasets. This remarkable learning capability of

the brain is a direct consequence of the hierarchical organization of the visual cortex and spike-based processing/learning scheme.[5]

Spiking Neural Networks (SNNs), popularly termed as the third generation of neural nets offer a promising solution for enabling on-chip intelligence.STDP enables the SNNs to learn the structure of input patterns without using labels in an unsupervised manner. Fully-connected architecture results in a large number of trainable parameters.

The Convolutional SNN trained on one set of classes (for instance, digits 6 and 7) is able to identify images from a completely different class (for instance, digits 9 and 1). This result shows that the proposed model can perform out-of-set generalization.Biologically inspired neural computing paradigms typically consist of a layer of input (pre) neurons connected by weighted synapses to the output (post) neurons.

At any given time-step, each input pre-neuronal spike is modulated by the corresponding synaptic weight to generate a resultant current into the post-neuron. This causes an increase in the neuronal membrane potential, which subsequently leaks in the absence of the input spikes. That is why the neuron model is called Leaky Integrate and Fire Neurons.

The potential is subsequently reset, and the neuron is restrained from spiking for a certain ensuing duration of time designated as the refractory period.

Weight update rule for STDP-

$$\Delta w_{STDP} = \eta \times \left[exp(\frac{t_{post} - t_{pre}}{\tau}) - offset \right] \times [w_{max} - w]^{\mu} \tag{3}$$

The non-linear (exponential) weight updates ensure gradual increase (decrease) in synaptic strength towards the maximum (minimum) value, which is desirable for efficient learning.

The fundamental limitation of learning in the fully connected topology is that it simply maps the given input image onto the synaptic weights and impedes the network from learning more generalized representations. To address this inadequacy, we explore a shared weight kernel feature extraction mechanism with a Convolutional SNN topology.

The input neurons are connected to every post-neuron in the excitatory layer by a unique weight kernel that is shared among the pre-neurons.That each excitatory neuron has a unique synaptic kernel, which is trained to acquire attributes that characterize a specific class of input patterns. The excitatory neurons are further connected to the inhibitory neurons in a one-to-one manner.

Every pixel in the image pattern constitutes an input pre-neuron, whose firing rate is proportional to the corresponding pixel intensity.

In the absence of a post-neuronal spiking event, the kernel is simply moved over the image by a pre-specified number of strides. However, in the event of a post-neuronal spike (time instant t_o+Δt the kernel weights are modified based on the temporal correlation between the post-spike and the corresponding pre-spikes as specified by the power-law weight-dependent STDP model.

Illustration of the convolutional STDP learning methodology for adapting the synaptic kernel connecting an input pattern to a specific excitatory post-neuron. At every time-step, the pre-neurons belonging to a distinct region in the input image are modulated by the kernel weights to generate a resultant current into the post-neuron. This increases the neuronal membrane potential m), which triggers an output spike if the potential exceeds a definite V_{mem} *threshold* θ. The kernel weights are modified at the instant of a post-neuronal spike in this example) based on the temporal correlation with the t_o + Δt corresponding pre-spikes as stipulated by the power-law weight-dependent plasticity model. The kernel is moved over (convolved with) the entire image by a pre-specified number of strides over multiple time-steps. This operation is repeated across the entire time duration for which the input pattern is presented.

The Convolutional SNN can potentially achieve comparable classification accuracy to that of a fully-connected network using fewer number of training examples. The Convolutional SNN exhibits rotational invariance in pattern recognition tasks. It is important to note that the recognition efficiency of the Convolutional SNN strongly depends both on the kernel size and the number of strides.

The recognition performance of the trained network is subsequently estimated on a separate testing dataset. Each test image is predicted to belong to the class represented by the group of neurons with the highest average spike-count throughout the simulation period.

Results indicate a drop in accuracy when stride value is increased thus, it is desirable to have minimal number of strides needed to cover the image pattern for a given size of the synaptic kernel.Decreasing the kernel size below 12 x 12 failed to predict many digits thus proving detrimental for the accuracy.

7 Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning

The multi-layer SNN is comprised of alternating convolutional and pooling layers and then the next layers being fully-connected layers, which are populated with leaky integrate-and-fire spiking neurons.[2]

STDP-based pre-training with gradient-based optimization provides better generalization, faster (2.5) training time and improved robustness. A spiking neuron transmits information in the form of electric event pulses (or spikes) through plastic synapses. Spike Timing Dependent Plasticity is a biologically plausible unsupervised learning mechanism that is simple and takes less training time to achieve a respectable but not state of that art accuracy.

STDP based pre-training with gradient based optimization has the following advantages-

1. Improved robustness
2. Faster training time
3. Better generalization

However, in the supervised learning in Spiking neural networks we face following drawbacks when using spike based Back propagation algorithm-

1. It is computation intensive
2. Requires large amount of data and effort. This impedes networks on-chip learning.
3. Procedure for computing derivatives of loss function with respect to weights is complicated.
4. Loss function does not have distinct global minima because of multi-dimensional non-convex loss function. Therefore it becomes hard to initialize the weights.

Both STDP and Spike based Back propagation algorithm have been demonstrated to capture hierarchical features in SNN.

$$\tau_m \frac{dV_{mem}}{dt} = -V_{mem} + w * \theta(t - t_k) \tag{4}$$

Where V_{mem} represents the membrane potential and τ_m denotes the rate of membrane leakage over time. Small value of τ_m means fast decay while large value of τ_m denotes slow decay. The incoming spike (Dirac-delta pulse) occurring at time instant t_k, denoted by $\theta(t - t_k)$, gets modulated by the synaptic weight (w) to produce resultant current that is integrated by the post neuron in its membrane potential. The input layer in Spiking convolutional network encodes image as Poisson distributed spike trains where the probability of spike generation is proportional to the pixel intensity.

Pooling operations offers following benefits-

1. It provides small amount of additional network invariance to input transformations
2. It increases the effective size of kernels for subsequent layers.

7.1 Unsupervised Learning

$$\Delta w_{STDP} = \eta \times \left(exp(\frac{t_{pre} - t_{post}}{\tau_{pre}}) - \chi_{offset} \right) \times (w_{max} - w) \times (w - w_{min}) \tag{5}$$

Δw is the change in w, η_{STDP} is the learning rate, w_{max} and w_{min} are the maximum and minimum values that w can take. The strength (weight) of synapse is potentiated if a pre-synaptic spike triggers the post-neuron within time window that is determined by a threshold(offset).

7.2 Supervised Fine tuning using Spike based back propagation

Unlike in ANNs, SNNs dont have differentiable activation function. Also, the post spike train is a non-differentiable function because of sudden discontinuity and also steep(large) slope. In spike-based backpropagation algorithm we low pass filter the post spike train to obtain a pseudo derivative by creating differentiable activation function.

$$net_j^{l+1}(t) = \sum_{i=1}^{n^l} w_{ij}^l * x_i(t), \quad where \quad x_i(t) = \sum_{k=1}^{t} \theta_i(t - t_k) \tag{6}$$

where net_j^{l+1} is the resultant current received by the j^{th} postneuron at $l + 1^{th}$ layer, n^l denotes the number of neurons in l^{th} layer, t_k represents the time instant at which pre-neuron spikes.

Forward Propagation

$$Activation \quad of \quad neuron, \quad a_j(t) = \sum_{k=1}^{t} exp(-\frac{t - t_k}{\tau_p}) \tag{7}$$

$$Final \quad output \quad error, \quad e_j = \frac{a_j^L}{max(a^L)} - label_j \tag{8}$$

$$Loss \quad function, \quad E = \frac{1}{2} \sum_{j=1}^{n^L} e_j^2 \tag{9}$$

Back-Propagation

$$\delta^L = e.a'(net^L) \tag{10}$$

$$a'(net^L) = a'(t) + 1 = \sum_{k=1}^{t} \left(-\frac{1}{\tau_p} e^{-\frac{t - t_k}{\tau_p}} \right) + 1 \tag{11}$$

$$\delta^h = ((w^h)^T * \delta^{h+1}).a'(net^h) \tag{12}$$

$$\Delta w^l = \frac{a^l}{max(a^l)} * \left(\delta^{l+1}\right)^T \tag{13}$$

$$w^l = w^l - \eta_{BP} \Delta w^l \tag{14}$$

8 Unsupervised learning of Digit recognition using Spike Timing dependent Plasticity

There are two main approaches of designing and training Spiking Neural Networks for Machine Learning applications[1]-

1. Learning weights of Spiking Neural Networks using different variants of Spike timing dependent Plasticity which is referred to as spike based learning
2. Training a rate based neural network with back propagation and using those weights for an Spiking Neural Networks

8.1 Neuron and Synapse model

$$\tau \frac{dV}{dt} = (E_{rest} - V) + g_e(E_{exc} - V) + g_i(E_{inh} - V) \tag{15}$$

$E_{rest} - V$= resting membrane potential
E_{exc}=equilibrium potential of excitatory synapses
E_{inh}=equilibrium potential of inhibitory synapses
g_e and g_i are conductances of those synapses

τ is larger for excitatory neurons than that for inhibitory neurons.
Synapses are modelled by conductance changes-

$$\tau_{ge} \frac{dg_e}{dt} = -g_e \tag{16}$$

$$\tau_{gi} \frac{dg_i}{dt} = -g_i \tag{17}$$

Real neurons' behaviour can be simulated between spikes **Inter Spike Interval** using Poisson process. To improve simulation speed the weights are computed using synaptic traces.

Homeostasis- To avoid a particular neuron from dominating the output pattern because of it's high firing rate we use adaptive membrane threshold control mechanism named as Homeostasis. In which the firing of a neuron is decided by the sum of threshold voltage and θ. The behaviour of θ is governed by the following equation-

$$\tau_\theta \frac{d\theta}{dt} = -\theta \tag{18}$$

Also, in the case when excitatory neurons fire less number of times, for eg: less than 5 spikes within 350ms, the maximum input firing rate is increased by a specific pre-decided value and the training set is presented till this issue of less firing rate gets solved.

In this paper the digit to be identified is predicted based on the responses of each neuron per class and then choosing the class with highest average firing rate. Also following important observations were made during the experiments -

- As number of neurons increased the acuuracy increased
- Exponential weight dependence STDP update rule was seen to give better accuracy than that with pre and post STDP update rule.

1. Exponential weight dependence STDP rule

$$\Delta w = \eta_{post}(x_{pre} - x_{post})exp(-\beta w) \tag{19}$$

β determines the strength of the weight dependence.
2. Pre and Post STDP update rule
   - For presynaptic spike

$$\Delta w = -\eta_{pre} x_{post} w^{\mu} \tag{20}$$

μ determines the weight dependence.
   - For post Synaptic spike

$$\Delta w = \eta_{post}\left(x_{pre} - x_{tar}\right)\left(w_{max} - w\right)^{\mu} \tag{21}$$

The second method of weight updating is computationally expensive that the first one. Therefore the first rule is used in this paper so as to decrease the power requirements.

9 Biologically plausible integrate and fire Dynamics

Choice on neuron model depends on the task for which we require it. It should be computationally less expensive and also should be biologically plausible. Most of the implementations in the research papers that I read had used the Leaky Integrate and Fire neuron model.[3]

Following are the values for the parameters of a LIF neuron which are usually observed to give good results and are also somewhat near to the real values that the neuro-cortical cells take.

Excitatory Cells

parameter	default	unit
V_{rest}	-65.0	mV
V_{reset}	-65.0	mV
V_{th}	-52.0	mV
τ_{m}	20.0	–
t_{ref}	2.0	ms
ρ_{AMPA}	0.5	–
τ_{AMPA}	1.5^{2}	ms
$\tau_{\text{NMDA,rise}}$	10.0^{2}	ms
τ_{NMDA}	100.0^{2}	ms
τ_{GABA}	5.5 or 11.0^{1}	ms

Inhibitory Cells

parameter	default	unit
V_{rest}	-60.0	mV
V_{reset}	-60 or -45^{1}	mV
V_{th}	-40.0	mV
τ_{m}	10.0	–
t_{ref}	1.0	ms
ρ_{AMPA}	1.0	–
τ_{AMPA}	1.5	ms
τ_{GABA}	5.5	ms

9.1 Program Code

Following are few MATLAB codes that I wrote for simulations and their results

Code to simulate Spikes of a neuron using the Izhikevich model

```
%Initialize parameters
dt=0.5;
d=8;
a=0.02;
c=-65;
b=0.2;
T=ceil(1000/dt);

%Reserve memory
v=zeros(T,1);
u=zeros(T,1);
v(1)=-70;
u(1)=-14;

%for loop over time
for t=1:T-1;
    if(t*dt>200 && t*dt<700)
        Iapp =7;
    else
        Iapp=0;
end

if v(t)<35
    %update ODE
    dv=(0.04*v(t)+5)*v(t)+140-u(t);
    v(t+1)=v(t)+(dv+Iapp)*dt;
    du=a*(b*v(t)-u(t));
    u(t+1)=u(t)+dt*du;
else
    %spike!
    v(t)=35;
    v(t+1)=c;
    u(t+1)=u(t)+d;
    end
end

%plot voltage trace
plot((0:T-1)*dt,v,'b');
xlabel('Time [ms]');
ylabel('Membrane voltage [mV]');
```

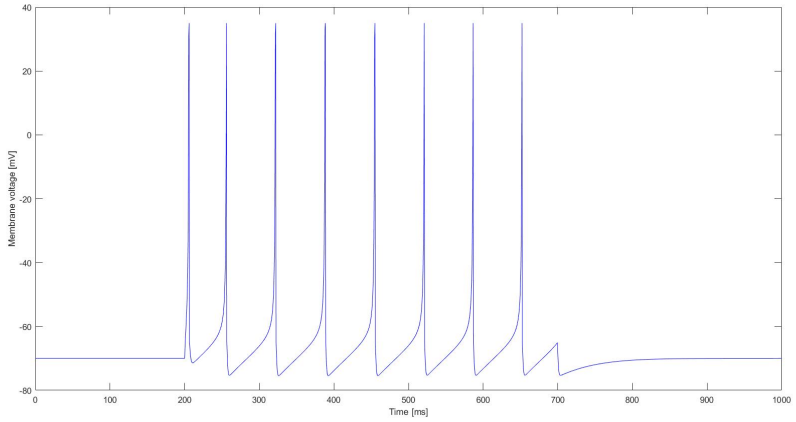

Fig. 1. Spikes

Code to generate poisson process based spikes and plot Raster plot based on the those spikes

Function to generate Spikes based on Poisson process

```
function [spikeMat, time_vector] = poissonSpikeGen(firing_rate, simulation_time, no_of_t
    dt = 1/1000; %1 ms
    no_of_bins = floor(simulation_time/dt);
    spikeMat = rand(no_of_trials, no_of_bins) < firing_rate*dt;
    time_vector = 0:dt:simulation_time-dt;
```

Function that generates Raster plot based on the the spikes generated by the above function

```
function [] = plotRaster(spikeMat, tVec)
    hold all;
    for trialCount = 1:size(spikeMat,1)
        spikePos = tVec(spikeMat(trialCount, :));
        for spikeCount = 1:length(spikePos)
            plot([spikePos(spikeCount) spikePos(spikeCount)], ...
                [trialCount-0.4 trialCount+0.4], 'k');
        end
    end
    ylim([0 size(spikeMat, 1)+1]);
```

Function to run and test above two functions

```
[spikeMat, tVec] = poissonSpikeGen(30, 1, 20);
plotRaster(spikeMat, tVec*1000);
xlabel('Time (ms)');
ylabel('Trial Number');
```

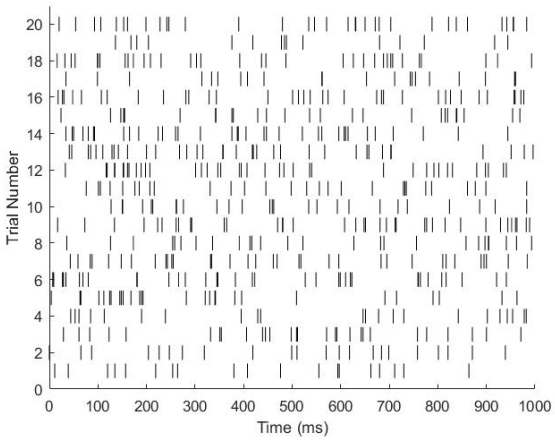

Fig. 2. Raster Plot

10 Setting up the environment

Brian is one of the popular libraries used for Spiking neural network simulation. Comparative studies between different simulation libraries can be found in Ruben A et. al's, paper titled Software for Brain Network Simulations [6]

Following are the steps for brian's correct installation-

1. Install pip utility to be able to install python packages

```
# On Linux/MacOsX:
pip install -U pip

# On Windows
python -m pip install -U pip
```

2. After installing pip you can install Brian library as follows

```
pip install brian2
```

3. Now we need to check if the library was installed correctly or not. To do that execute the following lines of code.

```
import brian2
brian2.test()
```

It should end with OK, showing a number of skipped tests but no errors or failures.If you are getting failed tests ,the test results do not mean you cannot use Brian! You can run Brian code in "numpy mode", the only disadvantage is that without the working C compiler it will not be as fast as it would be otherwise (typically by about a factor of 2). If weave/cython does not work, "numpy mode" will be automatically selected but you'll get a warning. Alternatively, you can also explicitly set prefs.codegen.target = 'numpy' and everything should work.

11 Code

```
#Importing all necessary libraries

    from brian2 import *
    import keras
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt
    import pandas as pd
    import os
    from keras.datasets import mnist
    import brian2tools
    from PIL import Image
    from brian2tools import *
    get_ipython().run_line_magic('matplotlib', 'inline')

############################################################

#Extracting Spike frequencies from images

def Make_pix_to_spike_dictionary():
    #array containing all pixel intensities from 0 to 255
    pix=range(0,256,1)
    #spike_freq list to store corresponding frequency
    #of the spike train for that pixel intensity value
    spike_freq=[]
    #Initialize frequency=5.1 for Pixel intensity of value 0
    init_spike_freq=5.1
```

```
    count=0
    for i in pix:
        if(count==5):
            #Count is used for increasing value of freq after
            #every group of 5 pixel intensities
            count=0
            init_spike_freq+=round(0.1,1)

        count+=1
        spike_freq.append(round(init_spike_freq,1))
    keys = pix
    values = spike_freq
    pix_to_spk_dict = dict(zip(keys, values))
    return pix_to_spk_dict

##########################################################
```

Following is the dictionary generated by the `Make_pix_to_spike_dictionary()` function-

0: 5.1, 1: 5.1, 2: 5.1, 3: 5.1, 4: 5.1, 5: 5.2, 6: 5.2, 7: 5.2, 8: 5.2, 9: 5.2, 10: 5.3, 11: 5.3, 12: 5.3, 13: 5.3, 14: 5.3, 15: 5.4, 16: 5.4, 17: 5.4, 18: 5.4, 19: 5.4, 20: 5.5, 21: 5.5, 22: 5.5, 23: 5.5, 24: 5.5, 25: 5.6, 26: 5.6, 27: 5.6, 28: 5.6, 29: 5.6, 30: 5.7, 31: 5.7, 32: 5.7, 33: 5.7, 34: 5.7, 35: 5.8, 36: 5.8, 37: 5.8, 38: 5.8, 39: 5.8, 40: 5.9, 41: 5.9, 42: 5.9, 43: 5.9, 44: 5.9, 45: 6.0, 46: 6.0, 47: 6.0, 48: 6.0, 49: 6.0, 50: 6.1, 51: 6.1, 52: 6.1, 53: 6.1, 54: 6.1, 55: 6.2, 56: 6.2, 57: 6.2, 58: 6.2, 59: 6.2, 60: 6.3, 61: 6.3, 62: 6.3, 63: 6.3, 64: 6.3, 65: 6.4, 66: 6.4, 67: 6.4, 68: 6.4, 69: 6.4, 70: 6.5, 71: 6.5, 72: 6.5, 73: 6.5, 74: 6.5, 75: 6.6, 76: 6.6, 77: 6.6, 78: 6.6, 79: 6.6, 80: 6.7, 81: 6.7, 82: 6.7, 83: 6.7, 84: 6.7, 85: 6.8, 86: 6.8, 87: 6.8, 88: 6.8, 89: 6.8, 90: 6.9, 91: 6.9, 92: 6.9, 93: 6.9, 94: 6.9, 95: 7.0, 96: 7.0, 97: 7.0, 98: 7.0, 99: 7.0, 100: 7.1, 101: 7.1, 102: 7.1, 103: 7.1, 104: 7.1, 105: 7.2, 106: 7.2, 107: 7.2, 108: 7.2, 109: 7.2, 110: 7.3, 111: 7.3, 112: 7.3, 113: 7.3, 114: 7.3, 115: 7.4, 116: 7.4, 117: 7.4, 118: 7.4, 119: 7.4, 120: 7.5, 121: 7.5, 122: 7.5, 123: 7.5, 124: 7.5, 125: 7.6, 126: 7.6, 127: 7.6, 128: 7.6, 129: 7.6, 130: 7.7, 131: 7.7, 132: 7.7, 133: 7.7, 134: 7.7, 135: 7.8, 136: 7.8, 137: 7.8, 138: 7.8, 139: 7.8, 140: 7.9, 141: 7.9, 142: 7.9, 143: 7.9, 144: 7.9, 145: 8.0, 146: 8.0, 147: 8.0, 148: 8.0, 149: 8.0, 150: 8.1, 151: 8.1, 152: 8.1, 153: 8.1, 154: 8.1, 155: 8.2, 156: 8.2, 157: 8.2, 158: 8.2, 159: 8.2, 160: 8.3, 161: 8.3, 162: 8.3, 163: 8.3, 164: 8.3, 165: 8.4, 166: 8.4, 167: 8.4, 168: 8.4, 169: 8.4, 170: 8.5, 171: 8.5, 172: 8.5, 173: 8.5, 174: 8.5, 175: 8.6, 176: 8.6, 177: 8.6, 178: 8.6, 179: 8.6, 180: 8.7, 181: 8.7, 182: 8.7, 183: 8.7, 184: 8.7, 185: 8.8, 186: 8.8, 187: 8.8, 188: 8.8, 189: 8.8, 190: 8.9, 191: 8.9, 192: 8.9, 193: 8.9, 194: 8.9, 195: 9.0, 196: 9.0, 197: 9.0, 198: 9.0, 199: 9.0, 200: 9.1, 201: 9.1, 202: 9.1, 203: 9.1, 204: 9.1, 205: 9.2, 206: 9.2, 207: 9.2, 208: 9.2, 209: 9.2, 210: 9.3, 211: 9.3, 212: 9.3, 213: 9.3, 214: 9.3, 215: 9.4, 216: 9.4, 217: 9.4, 218: 9.4, 219: 9.4, 220: 9.5, 221: 9.5, 222: 9.5, 223: 9.5, 224: 9.5, 225: 9.6, 226: 9.6, 227: 9.6, 228: 9.6, 229: 9.6, 230: 9.7, 231: 9.7, 232: 9.7, 233: 9.7, 234: 9.7, 235: 9.8, 236: 9.8, 237: 9.8, 238: 9.8, 239: 9.8, 240: 9.9, 241: 9.9, 242: 9.9, 243: 9.9, 244: 9.9, 245:

10.0, 246: 10.0, 247: 10.0, 248: 10.0, 249: 10.0, 250: 10.1, 251: 10.1, 252: 10.1, 253: 10.1, 254: 10.1, 255: 10.2

```
#Load the MNIST dataset

def load_mnist_dataset():
    (train_x, train_y) , (test_x, test_y) = mnist.load_data()
    return (train_x,train_y),((test_x, test_y))

#Extracts the pixel intensities from an Image

def get_image_pixels(index):
    load_mnist_dataset()
    img=test_x[index]
    test_img = img.reshape(1,784)
    return test_img,img

#########################################################

test_img,img=get_image_pixels(120)
mat = np.reshape(img,(28,28))

#########################################################


# Creates PIL image
img = Image.fromarray( mat , 'L')
plt.imshow(img)
plt.title('my picture')
plt.show()
```

The `get_image_pixels()` function returns the following matrix for the below sample image-

0 64 64 255 183 253 140 121 121 38 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 91 212 240 240 240 246 246 253 252 252 252 252 252 79 0 0 0 0 0 0 0 0 0 0 0 0 0 133 248 252 202 252 252 252 252 253 252 252 252 252 252 198 0 0 0 0 0 0 0 0 0 0 0 0 0 226 252 252 161 239 172 172 102 110 39 39 39 165 252 246 78 0 0 0 0 0 0 0 0 0 0 0 0 226 252 252 104 44 0 0 0 0 0 0 0 14 53 53 20 0 0 0 0 0 0 0 0 0 0 0 0 128 252 252 66 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 94 252 252 96 41 41 19 19 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 35 213 252 252 252 252 203 203 161 104 17 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 81 252 252 252 252 252 252 253 252 213 105 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 81 252 252 252 252 252 252 253 252 252 252 185 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 36 158 234 146 183 197 191 253 253 253 253 129 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 11 2 6 7 6 139 241 252 252 249 76 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 23 217 252 252 128 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 213 252 252 246 78 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 110 244 252 252 220 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 15 14 0 0 32 54 187 186 245 252 252 252 212 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 158 193 173 173 219 252 253 252 252 252 252 243 108 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 109 245 252 252 252 252 253 252 252 252 245 108 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 177 241 252 252 252 253 246 238 238 106 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 19 132 252 252 120 69 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

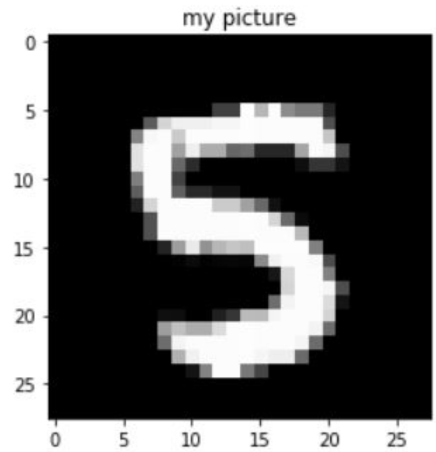

Fig. 3. Sample image

```
#Returns the spiking frequencies corresponding
#to all pixels in the image

def get_spike_freqs_from_image(index):
    arr,img=get_image_pixels(index)
    spike_freqs=[]
    pix_to_spk_dict=Make_pix_to_spike_dictionary()
    for i in arr[0]:
        if(pix_to_spk_dict.get(i)):
            spike_freqs.append(pix_to_spk_dict.get(i))
    return spike_freqs

########################################################
```

The `get_spike_freqs_from_image(index)` function returns the following matrix-

[5.1, 6.3, 6.3, 10.2, 8.7, 10.1, 7.9, 7.5, 7.5, 5.8, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 6.9, 9.3, 9.9, 9.9, 9.9, 10.0, 10.0, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 6.6, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 7.7, 10.0, 10.1, 9.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 9.0, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 9.6, 10.1, 10.1, 8.3, 9.8, 8.5, 8.5, 7.1, 7.3, 5.8, 5.8, 5.8, 8.4, 10.1, 10.0, 6.6, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 9.6, 10.1, 10.1, 7.1, 5.9, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.3, 6.1, 6.1, 5.5, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 7.6, 10.1, 10.1, 6.4, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 6.9, 10.1, 10.1, 7.0, 5.9, 5.9, 5.4, 5.4, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.8, 9.3, 10.1, 10.1, 10.1, 10.1, 9.1, 9.1, 8.3, 7.1, 5.4, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 6.7, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 9.3, 7.2, 5.3, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 6.7, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 8.8, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.8, 8.2, 9.7, 8.0, 8.7, 9.0, 8.9, 10.1, 10.1, 10.1, 10.1, 7.6, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.3, 5.1, 5.2, 5.2, 5.2, 7.8, 9.9, 10.1, 10.1, 10.0, 6.6, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.5, 9.4, 10.1, 10.1, 7.6, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 9.3, 10.1, 10.1, 10.0, 6.6, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 7.3, 9.9, 10.1, 10.1, 9.5, 5.5, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.4, 5.3, 5.1, 5.1, 5.7, 6.1, 8.8, 8.8, 10.0, 10.1, 10.1, 10.1, 9.3, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 8.2, 8.9, 8.5, 8.5, 9.4, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 9.9, 7.2, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 7.2, 10.0, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.1, 10.0, 7.2, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 8.6, 9.9, 10.1, 10.1, 10.1, 10.1, 10.0, 9.8, 9.8, 7.2, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.4, 7.7, 10.1, 10.1, 7.5, 6.4, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1]

```
#LIF neuron simulation

import numpy as np
import pandas as pd
import brian2 as b2
import os
import re
from brian2 import *


tau = 2 * msecond          # membrane time constant
Vt = -52 * mvolt           # spike threshold
Vr = -65 * mvolt           # reset value
El = -65 * mvolt           # resting potential (same as the
                               reset)

eqs = '''dv/dt = -(v-El)/tau : volt'''
group = NeuronGroup(28*28, eqs,
                    threshold='v > -50*mV',
                    reset='v = -70*mV')
M = StateMonitor(group, 'v', record=True)

#By default the time step is 0.1ms
run(1 * second)


plot(M.t/ms, M.v[0])
print(len(M.t/ms))
xlabel('Time (ms)')
ylabel('v')
```

```
#Function to plot the raster plot from spikemonitor

def Plot_raster_plot(spikemonitor):
    plot(spikemonitor.t/ms, spikemonitor.i, '.k')
    xlabel('Time (ms)')
    ylabel('Neuron index');

############################################################

#LIF neuron model 1 (excitatory neurons in second layer)
#Code to connect a single neuron to all the 784
#neurons in the input

start_scope()
a=spike_freqs_img_120*Hz
P = PoissonGroup(784,a)
W=5*siemens
```

```
eqs1 = '''dv/dt = -(v-El)/tau : volt'''

neuron=NeuronGroup(100,eqs1,threshold='v>Vt',reset='v=Vr',
                                    method='exact')
neuron.v=Vr
synapses=Synapses(P,neuron,'w: siemens')
synapses.connect()
synapses.w=W

M = SpikeMonitor(P)
N = SpikeMonitor(neuron)
run(1*second)
brian_plot(M)
brian_plot(N)
```

After doing this we have now converted image input to a 2D matrix of poisson spiking neurons with frequencies defined from the dictionary we had defined earlier. We can then visualize all those poisson inputs with the help of a raster plot as follows-

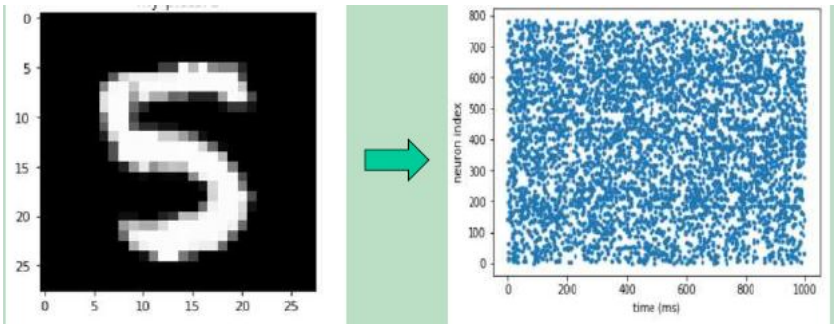

Fig. 4. Conversion of Image input to Poisson spike inputs

```
############################################################

#LIF neuron model 1 (excitatory neurons in second layer)
#Code to connect a single neuron to all the 784
#neurons in the input

start_scope()
a=spike_freqs_img_120*Hz
P = PoissonGroup(784,20*Hz)
refrac_e = 1. * ms
```

```
eqs = '''
dv/dt = ((v_rest_e - v) + (I_synE+I_synI) / nS) / (2*ms)  :
                                volt (unless refractory)
I_synE = ge * nS *          -v                            : amp
I_synI = gi * nS * (-70.*mV-v)                            : amp
dge/dt = -ge/(1.0*ms)                                     : 1
dgi/dt = -gi/(2.0*ms)                                     : 1
'''

neuron= NeuronGroup(1, eqs, threshold= 'v>Vt',refractory=
                                refrac_e, reset= 'v=Vr',
                                method='euler')

neuron.v=Vr
synapses=Synapses(P,neuron,'w: siemens')
synapses.connect(i=0,j=0)
synapses.w = 25*nS
M = SpikeMonitor(P)
N=SpikeMonitor(neuron)
run(5*second)
brian_plot(M)

############################################################

#Creating neurons of second and third layer

v_rest_e = -65. * mV
v_rest_i = -60. * mV
v_reset_e = -65. * mV
v_reset_i = -45. * mV
v_thresh_e = -52. * mV
v_thresh_i = -40. * mV
refrac_e = 5. * ms
refrac_i = 2. * ms

no_exci=400
no_inhi=no_exci

v_reset_e = 'v = v_reset_e; theta += theta_plus_e; timer = 0*
                                ms'

v_thresh_e_eqn = '(v>(theta - offset + v_thresh_e)) and (
                                timer>refrac_e)'
v_thresh_i_eqn = 'v>v_thresh_i'
v_reset_i_eqn = 'v=v_reset_i'


exc_neurons = '''
dv/dt = ((v_rest_e - v) + (I_synE+I_synI) / nS) / (100*ms)  :
                                  volt (unless refractory)
```

```
I_synE = ge * nS *          -v                              : amp
I_synI = gi * nS * (-100.*mV-v)                             : amp
dge/dt = -ge/(1.0*ms)                                       : 1
dgi/dt = -gi/(2.0*ms)                                       : 1
'''


inh_neurons = '''
dv/dt = ((v_rest_i - v) + (I_synE+I_synI) / nS) / (10*ms)  :
                                volt (unless refractory)
I_synE = ge * nS *          -v                              : amp
I_synI = gi * nS * (-85.*mV-v)                              : amp
dge/dt = -ge/(1.0*ms)                                       : 1
dgi/dt = -gi/(2.0*ms)                                       : 1
'''

neuron_groups['e'] = NeuronGroup(no_exci*len(population_names
                                ), exc_neurons, threshold=
                                v_thresh_e_eqn, refractory=
                                refrac_e, reset= v_reset_e,
                                method='euler')


neuron_groups['i'] = NeuronGroup(no_inhi*len(population_names
                                ), inh_neurons, threshold=
                                v_thresh_i_eqn, refractory=
                                refrac_i, reset= v_reset_i_eqn
                                , method='euler')

#############################################################

#Creating the one to one connections between excitatory
                                neurons
#of second layer with inhibitory neurons of third layer

pre = 'Apost += w'
model='''
w:1
dApre/dt=-Apre/taupre : 1 (event-driven)
dApost/dt=-Apost/taupost : 1 (event-driven)
'''

ei_connection=Synapses(neuron_groups['e'], neuron_groups['i']
                                , model=model, on_pre=pre)

for i in range(no_exci):
    ei_connection.connect(neuron_groups[i][0],
    neuron_groups[i][0])
```

References

1. Diehl, P. and Cook, M. (2019). Unsupervised learning of digit recognition using spike-timing-dependent plasticity.

2. Lee C, Panda P, Srinivasan G and Roy K (2018) Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning. Front. Neurosci. 12:435. doi: 10.3389/fnins.2018.00435

3. Jug, F. (2012). On Competition and Learning in Cortical Structures. Ph.D. thesis, Diss., Eidgenssische Technische Hochschule ETH Zrich, Nr. 20194, 2012

4. Ponulak, Filip Kasiski, Andrzej. (2011). Introduction to spiking neural networks: Information processing, learning and applications. Acta neurobiologiae experimentalis. 71. 409-33.

5. Panda, Priyadarshini Srinivasan, Gopalakrishnan Roy, Kaushik. (2017). Convolutional Spike Timing Dependent Plasticity based Feature Learning in Spiking Neural Networks.

6. Tikidji-Hamburyan R. A., Narayana V., Bozkus Z., El-Ghazawi T. A. (2017). Software for brain network simulations: a comparative study. Front. Neuroinformat. 11:46 10.3389/fninf.2017.00046