Assignment 1 Report: Spiking Neural Network
ELL741 - Neuromorphic Engineering

November 3, 2019

This document is a report submitted, along with the relevant codes, as a part of the first homework of the course ELL741 - Neuromorphic Engneering.

Students involved:

• Lakshya Bhatnagar	2017EE10458
• Pradyumna Jalan	2017EE10469
• Janak Sharda	2017EE10454
• Ritvik Sharma	2017EE10485
• Siddharth Dangwal	2017EE10497
• Nilabjo Dey	2017EE10465

1 Introduction

In this part, we had to make a Spiking Neural Network[1], with the intricacies of the Model being left at our discretion. We have mostly followed the papers that were discussed during the lectures and provided as review material.

[1]We have not used any machine learning library in our submission other than for importing the data set and defining the test, train sets.

Contents

Part I
LIF: Partially Supervised Learning

2 Network

We have made a simple two layer network, where each output neuron is associated with one class of the target set. Number of output neurons is equal to the total number of distinct classes in the target set. Number of input neuron is a multiple of features, where each feature is processed in a manner that is mentioned in the next section. The input and output neurons are fully connected by excitatory synapses. The output neurons are themselves connected by lateral inhibitory synapses. Further, we have applied bias currents to the output layer as a measure to prevent output neurons corresponding to the wrong class from spiking.

3 Input Data

We have tried to keep the framework generalised to be implemented on any data-set.

3.1 Normalisation

The raw feature values are mapped on a scale of 0 to 1, given by the following equation. This is done for ease in transformation of the features.

$$x_i = \frac{z_i - min(z)}{max(z) - min(z)}$$

Where,

- z is the raw feature vector
- x is the normalised feature vector

3.2 Transformation

Upon taking the input data as it is, the weights are often optimised such that they become insensitive to the lower or middle range of the feature values. Of the multiple population schemes, we have implemented Gaussian transformation of the normalised features.

To count for this, we have allocated T neurons for each of the original feature of the data set.

3.2.1 Gaussian transformation

Our neurons take current as an input, so we have chosen to represent our transformed feature by a current.

$$I_j(x) = I_{max} \cdot exp\Big(\frac{x - \mu_j}{2\sigma^2}\Big)$$

Where,

- x is the normalised feature vector
- $I_{j,x}$ is the newer feature vector
- T is the number of features, each of the older feature gives rise to
- $\mu_j, j\epsilon\{0, 1, ...T-1\}, \sigma$ are the parameters of the Gaussian distribution
- $I_{max} = 4nA$, is a constant to be decided in accordance with the neuron model

4 Neurons

We have used the simple Leaky Integrate and Fire (LIF) model of neuron in our python code. The equation governing the functioning of a neuron is:

$$\frac{dV}{dt} = \frac{1}{C}\Big[-g(V - E_r) + I_{ip}\Big]$$

Where,

- V is the Membrane potential.
- C is the capacitance
- g is the conductance
- E_r is the resting potential
- I_{ip} is the input current to the neuron

Parameter	Value
E_r	-70mV
C	300 pF
g	30 nS

Table 1: Parameter values used for neuron modeling

For the input neurons, I_{ip} is simply the input feature current. For the output neurons, this is a sum of current inputs from (1) input neurons (excitatory), (2) other output neurons (inhibitory), (3) bias current (only when training).

Since we are coding, we don't really solve the differential equation, but take an approximation at every time step. Smaller the time step, the closer we get to the actual differential equation. After substituting the value, the equation for voltage update reduces to,

$$\Delta V = \big[-100(V + 0.07) + \frac{10}{3} I_{ip}\big]\Delta t$$

Where,

- ΔV is the voltage update
- Δt is the time step, value chosen by us
- I_{ip} is to be taken in nA
- V is in volts

We have considered the neuron to spike when its voltage value after the update, exceeds a threshold value. After spiking, the voltage value is reset to 0 (or to any reset potential). We have not considered any refractory period after each spike. Instead of having a refractory period, the reset potential can be set to a sufficiently negative value such that it increases the time requires for the neuron to spike again. Thus, inducing refractory period like effects.

5 Synapses

A synapse is a connection between two neurons. A synapse converts a voltage spike of input neuron into a corresponding current pulse for the output neuron. This synapse can be taken as a directed connection. We have taken two kinds of synapses, excitatory and inhibitory. Excitatory synapses connect input and output neurons. Inhibitory synapses connect the output neurons within themselves.

For our understanding, we have assumed that upon firing, a neuron produces an output current pulse(I^{out}) which upon travelling through the synapse gets multiplied by the weight of the synapse before entering the input of the other neuron.

The weights of the excitatory synapses are learned during training whereas the inhibtory synapses have fixed weights throughout.

The form of the current equation is given by,

$$I(t) = w \cdot I_o \cdot \Big[exp\big(-\frac{t - t_s}{\tau_m}\big) - exp\big(-\frac{t - t_s}{\tau_s}\big)\Big] = w \cdot I^{out}$$

Where,

- w is the weight of the synapse. $w > 0$ for excitatory and $w < 0$ for inhibitory synapse.
- t_s is the time instant for which the pre fires

- τ_m , τ_s are time constants with $\tau_m > \tau_s$
- I_o is the synaptic current constant. Vallue to be chosen such that the current pulse is scaled to required levels for the LIF neuron (treated as a hyperparameter)

Parameter	Value
τ_m (excitory)	10 mS
τ_m (inhibitory)	50 mS
τ_s	2.5 mS

Table 2: Parameters for synapse modeling

Lets say we have N input neurons and M output neurons. Then the input current to the kth output neuron is given as

$$I^{in}_{2,k}(t) = \sum_{i=0}^{N} w^{in}_{i,k} I^{out}_{1,i} + \sum_{j=0;j\neq k}^{M} w^{out}_{j,k} I^{out}_{2,j} + I_{bias}$$

First term is the excitatory current, second is the inhibitory current and the last is the target based bias current.

6 Learning Rule

The weight change is a function of the time difference between spiking of pre(t_{pre}) and post(t_{post}) neuron. ($\Delta t = t_{post} - t_{pre}$)

$$\Delta w = A_{up} \cdot \Big(1 - \frac{w}{w_{max}}\Big)^{\mu} \cdot exp\Big(\frac{-\Delta t}{\tau_{up}}\Big); \Delta t \geq 0$$

$$\Delta w = A_{down} \cdot \Big(\frac{w}{w_{max}}\Big)^{\mu} \cdot exp\Big(\frac{\Delta t}{\tau_{down}}\Big); \Delta t < 0$$

A_{up} and A_{down} have been chosen from the ranges mentioned in the reference paper. These are analogous to the learning rates in the conventional neural networks. Taking smaller values for there will result in slower learning, but reduces the extent of over-fitting for each data point. These are essentially hyper-parameters for us.

7 Training

While training, each sample is shown to the network input and a bias current is flown through the output neuron for a fixed time (we took 100 mS).

Even after taking the values as given in the paper, we were not getting good results. The key to choosing good parameter values is to keep in mind that the current values thus obtained should be in plausible ranges. That is, the current input to any neuron should not be orders above the critical current value, neither should the currents go to farther negative values. Though, there is no concrete proof to why this happens, it was observed that when keeping this analogy in considerations, and playing with the current values, the results improved.

While choosing the values of the synaptic current constants, I did not straight away went for maximising accuracy. Instead, I focused on keeping the values such that the increment current to the post neurons lie between -2 and 5 nA (2.7nA being out critical current).

Once, 100% training accuracy was obtained, we used a grid search to fine-tune the accuracy. The results were very sensitive to changes in values of any parameter.

7.1 Defining Accuracy

We are showing each sample for 100mS which is about twice the duration for Synaptic Pulses. For the purpose of calculating accuracy, we consider the later half of this time. Keeping track of the spiking of the output neurons for the duration, we consider the sum total of spikes for each of the neuron. We consider that the network has predicted the class to be the the one that corresponds to the neuron that has fired the most. In case of a tie, we consider it a wrong prediction. Prediction is considered right if the neuron that has spiked the most in this duration indeed corresponds to the correct label class.
We have calculated accuracy after each sample is given and reported the it after the training is over.

8 Testing

For purposes of testing, we have only taken data that is unseen to the network. The functioning of the network is same as that while training, just that there is no bias current given here.
After checking for different hyper-parameters at random, when we got a decent measure of accuracy, we performed a grid search[2] on similar values.
The code that we have enclosed gives a testing accuracy between 90% and 94% which varies by virtue of the random initialisation of the weights. We have observed that best training accuracy is observed when the network is trained only for one epoch. That too with a smaller training set. We observed that when we kept the train split to 20% of the whole iris data set, the results were best.

[2]The results are enclosed in the submission as a .csv file

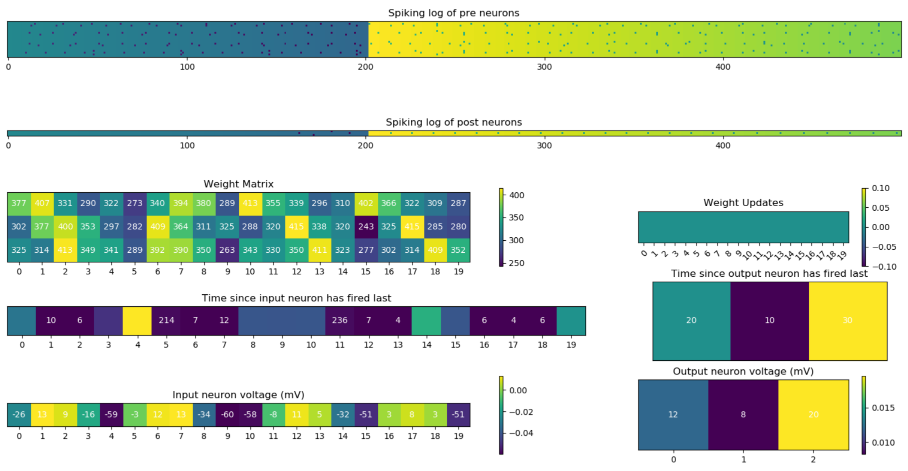

Figure 1: Sample Plot

9 Visualisation

We have included dynamically updated python plots[3] for better visualisation of the functioning of the network. See fig 1.

The values that are not explicitly mentioned on this plot (just depicted by a color) have little to no contribution at this moment in the simulation.

[3]Turning these off has no consequence on the correctness or working of the network

Part II
LIF: Unsupervised Learning

10 Comparison with Partially Supervised Model

The network is essentially the same as that in the previous section. Network, pre-processing of the input data, neuron characteristics, synapses are the same. There are minor changes in the learning rule and the way we interpret the output. These are explained in the subsequent sections. Same type of dynamically updated visuals are available for this part as well.

11 Learning Rule

Here, we simply drop the bias current term from the input to the post neuron. Then the input to the kth output neuron is is given as

$$I_{2,k}^{in}(t) = \sum_{i=0}^{N} w_{i,k}^{in} I_{1,i}^{out} + \sum_{j=0;j\neq k}^{M} w_{j,k}^{out} I_{2,j}^{out}$$

The weight update rule is kept the same. Now that, we are not giving a bias current based on the target output, we do not know for certain which output neuron corresponds to which class. For purposes of training, we need not know the association of output neurons with the classes.

12 Training and Testing

Apart from the difference in bias current there is not much difference from Part I. Here, we have increased the input neurons to an integral multiple of the number of labels.

12.1 Allocating labels

Once the training is done, we again show the training data to the network and map the output neurons to the class for which they spike the most. Then this mapping is used to calculate the testing accuracy.
Accuracy is determined in the same way as defined in Part I.

Part III
Diehl and Cook's method

13 Network

In this part, the network is one that was proposed by Peter Diehl and Mathew Cook in their paper, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity". The network is made up of three layers namely: Input layer(20 Neurons, 5 for each feature), Excitory Neuron Layer/ Output Layer (6 Neurons) and Inhibitory Neuron Layer (6 Neurons). Each neuron of output layer is connected to each neuron of output layer, whereas every neuron in inhibitory layer takes input from a single unique neuron in output layer and sends output spikes to all the neurons in output layer except the one to which it is connected.

14 Input Data

The input features were normalized and Gaussian transformation was used (Same as in Part I).

15 Neuron

The equation governing the functioning of a neuron is:

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

where:
E_{rest} = resting membrane potential
E_{exc} = equilibrium potential of excitatory synapses
E_{inh} = equilibrium potential of inhibitory synapses
g_e = conductance of excitatory synapses
g_e = conductance of inhibitory synapses
As observed in biology, we use a time constant τ, which is longer for excitatory neurons than for inhibitory neurons. When the neuron's membrane potential crosses its membrane threshold V_{thres}, the neuron fires and its membrane potential is reset to V_{reset}. Within the next few milliseconds after the reset, the neuron is in its refractory period and cannot spike again.

16 Synapses

Synapses are modeled by conductance changes, i.e., synapses increase their conductance instantaneously by the synaptic weight w when a presynaptic spike arrives at the synapse, otherwise the conductance is decaying exponentially. If the presynaptic neuron is excitatory, the dynamics of the conductance g_e are

$$\tau_{g_e} \frac{dg_e}{dt} = g_e$$

where τ_{g_e} is the time constant of an excitatory postsynaptic potential. Similarly, if the presynaptic neuron is inhibitory, a conductance g_i is updated using the same equation but with the time constant of the inhibitory postsynaptic potential τ_{g_i}.

17 Learning Rule

All synapses from input neurons to excitatory neurons are learned using STDP. To improve simulation speed, the weight dynamics are computed using synaptic traces. This means that, besides the synaptic weight, each synapse keeps track of another value, namely the presynaptic trace x_{pre} and postsynaptic trace x_{post}, which models the recent presynaptic spike and postsynaptic spike history, respectively. Every time a presynaptic spike arrives at the synapse, the trace is increased by 1, otherwise x_{pre} decays exponentially. When a postsynaptic spike arrives at the synapse the weight change Δw is calculated based on the presynaptic trace:

$$\Delta w = \eta_{post}(x_{pre} x_{tar})(w_{max} w)^{\mu}$$

where w_{max} = maximum weight
x_{tar} =target average value of the presynaptic trace at the moment of a postsynaptic spike
Increase of postsynaptic trace x_{post} is triggered by a postsynaptic spike. The weight change Δw for a presynaptic spike is:

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

where η_{pre} and η_{post} are the learning-rate for a presynaptic and postsynaptic spike and μ determines the weight dependence.

18 Training and Testing

To train the network, we present digits from the normalized and transformed Fischer Iris data set to the network. The data is split into 3:1 ratio train:test split. Before presenting a new data point, there is a 150 ms phase without any input to allow all variables of all neurons decay to their resting values (except for the adaptive threshold). After training is done, we set the learning rate to zero, fix each neuron's spiking threshold, and assign a class to each neuron, based on its highest response to the ten classes of digits over one presentation of the training set. This is the only step where labels are used, i.e., for the training of the synaptic weights we do not use labels.

The response of the class-assigned neurons is then used to measure the classification accuracy of the network on the test set. The predicted class is determined by the neuron which spikes the most.

19 Results

Though Diehl and Cook claim to have high accuracy for MNIST data set, but this network may not be generalized to other data sets. We were unable to achieve a decent testing accuracy for Fischer Iris data set in the given time. The following may be the reasons for this:

1. Dimensionality: The MNIST data set has a very high dimension, which may be one reason that the model works. In our case we tried to increse dimensionality of the data through gaussian transformation, but it seems this is not enough. We can try a few more kernels in future.

2. Parameters: For the assignment, the most parameters were taken form the paper and supplementary code. We tried to tweak some parameters but not all of them. We think an extensive parameter search is required to achieve some good results.

General Remarks

In out time playing with the parameters, we have found the outputs to be very sensitive the changes. If the parameters are not chosen correctly, the outputs may become unbounded and the analogy with the “actual neuron” starts to break.