CS425 HW 2

Hemanth Chiluka (hkc33)

March, 28 2019

1 Problem 1

1.1 Questions

1. Using a temporal encoding, the inputs would be represented as a spike trace. The trace specifies at what times spikes occurred, and each input is assigned a frequency that determines how often they occur. 0 will have a lower frequency than 1.

2. Using rate encoding, the inputs would be represented as spike count rate that's averaged over a time interval. This could directly be translated to the amount of current fed to the input neurons. 0 would have a lower rate than 1.

1.2 SNN Architecture

The SNN consists of an input, hidden, and output layer consisting of LIF neurons. The input layer is made of two neurons that receive the spike trains corresponding to the possible inputs [(0, 0), (1,0), (0,1), (1,1)]. The hidden layer is made of two neurons representing a NAND and OR gate. The output layer is made of one neuron representing an AND gate. The NAND neuron is inhibitory, meaning it provides a negative potential to prevent post-synaptic neurons from firing.

Each neuron sends a current to its post-synaptic neurons which is integrated over time to raise their action potential. When the potential of a neuron reaches a threshold, it fires, causing the current it outputs to increase. Both the action potential and current decay over time.

For each training session, the model iterates over a specified number of time steps propagating the input spikes forward. In each iteration, the model updates each synapse's weight using hebbian learning or STDP. For hebbian, oja's rule is used to change the weights. For STDP, each neuron that fires traces each spike train of its pre-synaptic neurons within a window. If the pre-synaptic neuron spikes within this window, the time difference and current weight are used to update the weight of their synapse.

Hyperparameters are stored and described in the params.py script

1.3 SNN Behavior

1.3.1 Hebbian

Input	0,0	0,1	1,0	1,1
Firing Rate	8	68	65	54

	(0, 0)	(1, 0)	(0, 1)	(1, 1)
Hidden Layer	0.1817	0.1817	0.4840	0.2565
Output Layer	8.8947e-5	0.0628		

1.3.2 STDP

Input	0,0	0,1	1,0	1,1
Firing Rate	44	117	117	98

	(0, 0)	(1, 0)	(0, 1)	(1, 1)
Hidden Layer	0.9206	0.9206	0.7416	0.7416
Output Layer	0.8911	0.6160		

With both learning methods, the model has the highest spiking rates (0, 1) and (1, 0).

1.4 Running

Run python snn.py (0=hebbian, 1=STDP)

2 Problem 2

2.1 SNN Architecture

The SNN is responsible for detecting a horizontal line of 1's on a 10X10 grid. On-center cells, which have a higher spiking rate for 1's, and off-center cells, which have a higher spiking rate for 0's, are used to make up the network.

The SNN has two layers. The first is made of 10 sets of 100 neurons, where each neuron responds to the binary value at its corresponding grid cell, and each set is responsible for detecting one row in the grid. In each set of neurons, on-center cells are placed on the row it is detecting, and off-center cells everywhere else.

The output layer is made of 10 neurons, one for each row. Each set of neurons in the first layer is connected to one neuron in the output layer.

2.2 SNN Behavior

The SNN is tested on 18 lines centered at the origin and rotated in increments of 20.

Degrees	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10
0	200.0	200.0	200.0	200.0	**266.0**	200.0	200.0	200.0	200.0	200.0
20	200.0	200.0	200.0	250.0	250.0	250.0	200.0	200.0	200.0	200.0
40	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0
60	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0
80	251.0	251.0	251.0	250.0	251.0	250.0	251.0	251.0	251.0	251.0
100	250.0	250.0	250.0	250.0	250.0	250.0	250.0	250.0	250.0	250.0
120	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0
140	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0
160	200.0	200.0	200.0	250.0	250.0	250.0	200.0	200.0	200.0	200.0
180	251.0	251.0	251.0	250.0	250.0	250.0	251.0	251.0	251.0	251.0
200	200.0	200.0	200.0	250.0	250.0	250.0	200.0	200.0	200.0	200.0
220	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0
240	200.0	200.0	200.0	201.0	200.0	201.0	200.0	200.0	200.0	200.0
260	250.0	250.0	250.0	250.0	251.0	250.0	250.0	250.0	250.0	250.0
280	250.0	250.0	250.0	250.0	250.0	250.0	250.0	250.0	250.0	250.0
300	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0
320	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0	200.0
340	201.0	201.0	201.0	250.0	250.0	250.0	201.0	201.0	201.0	200.0
360	200.0	200.0	200.0	200.0	**266.0**	200.0	200.0	200.0	200.0	200.0

In the horizontal line case, the output neuron that spikes the most is N5, which is expected since the line is centered at the origin.

2.3 Running

Run python detector.py