# CS425/525 Assignment 2

**Due December 1, 2020**

# 1 Problem 1: A basic SNN performing basic logic operations

## 1.1 Overview

The simplest spiking neuronal network (SNN) is one that has a single layer of neurons. This is essentially an output layer and directly receives connections from input neurons. Input neurons act as encoders of input data. Output is decoded from the output neuron activity. Network training adjusts connection weights between input neurons and output neurons, so that specific inputs are mapped to specific outputs. Figure 1 shows a visualization of a single layer SNN with 2 output neurons.

### 1.1.1 Encoding Network Input

To introduce real-world data to a SNN, we need to encode input using a spike code. There are two primary methods of encoding real valued data into spikes: 1) rate coding, and 2) temporal coding.

### 1.1.2 Network Computation

Once the encoding and decoding schemes are established, a network output can be computed for a specific input. In order to compute an SNN's output, one must integrate all the neurons in the network over time with input neurons receiving input. During integration, neurons generate spikes, which are sent through connections and integrated by other neurons in the network. Eventually, spikes propagate from the input neurons to the output neurons, resulting in network output.

### 1.1.3 Decoding Network Output

Since output neurons emit spikes, their spiking activity has to be decoded into real-world values. Normally, decoding is performed using the same or similar methods as encoding. For example, we count the number of spikes per a specific time window, and we use that number (rate) to represent an output value.

### 1.1.4 Training

To train the network to produce the right output given a certain input, we need to change the weights of the network connections. A variety of learning rules exist for SNNs ranging from biologically inspired Hebbian plasticity rules, such as spike timing dependent plasticity (STDP), to purely engineered rules such as pseudo-gradient descent error minimization [2, 3].

## 1.2 Exercises

Consider the AND, OR and XOR logic functions that are described in the table below:

| $\mathbf{x}$ | $\mathbf{y}$ | $\mathbf{x} \wedge \mathbf{y}$ | $\mathbf{x} \vee \mathbf{y}$ | $\mathbf{x} \veebar \mathbf{y}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

1. How would you represent the inputs $x$, $y$ using a **temporal encoding**? Provide a specific example.

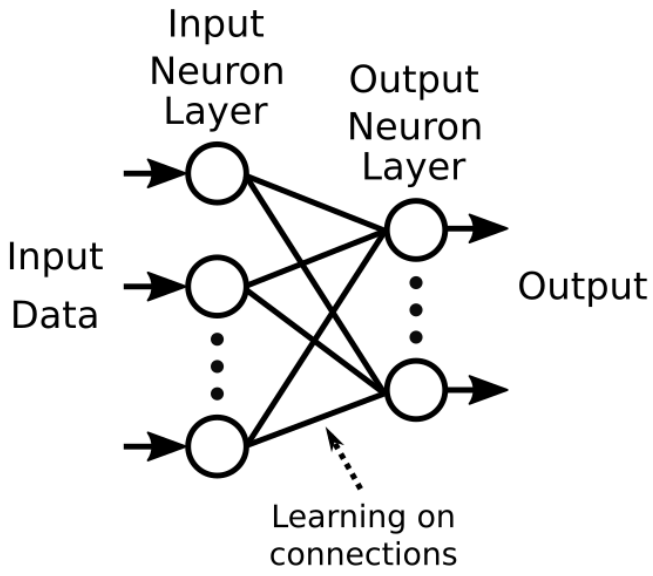


Figure 1: **One Layer Spiking Neuronal Network**

2. How would you represent the inputs $x$, $y$ using a **rate encoding**? Provide a specific example.

3. Build a single layer SNN that can learn the AND function. You can use either LIF or Izhikevich neurons that you have developed in Assignment 1. The network should have atleast 2 input neurons (depending on your encoding method); one for input $x$ and one for input $y$. The network should have atleast 1 output neuron (depending on your decoding method). You can use any encoding/decoding and learning rules but make sure that you adequately explain and cite the methods you choose. Specifically, please provide the following about your solution:

   - A graphic of the network architecture, similar to Figure 1
   - Describe the encoding and decoding scheme you used.
   - Describe the learning rule you used.
   - Describe the training process you used.
   - Demonstrate that your SNN learned the AND function by visualizing network activity in the form of a raster plot with each neuron labeled for the decoded input/output.

4. Can your single-layer SNN learn the OR function? If not, explain why not and provide references to back up your claims. If yes, demonstrate that your network learned the OR function by visualizing network activity in the form of a raster plot with each neuron labeled for the decoded input/output, as before.

5. Can your single-layer SNN learn the XOR function? If not, explain why not and provide references to back up your claims. If yes, demonstrate that the network learned the XOR function by visualizing network activity in the form of a raster plot with each neuron labeled for the decoded input/output, as before.

# 2 Problem 2: An SNN tackling a real-world classification problem

## 2.1 Overview

Here you will redesign your SNN to apply it to a real world classification problem with real valued input dated.

#### 2.1.1 Dataset

You will be training your SNN to classify handwritten digits, based on the following dataset:
`https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits`
You can use the *scikit-sklearn* library to download and read the dataset:
`https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html`
The data consists of 1797 images sized $8x8$ pixels. Images can be found under the dictionary key label *images*. Labels for each corresponding image are located under dictionary key label *target*.

#### 2.1.2 Training Methodology

Proper training methodology requires that a dataset be partitioned into 3 sets:

1. **Training Set**: This is the set of data that will be used to train your network. It should be the biggest partition, to expose your network to the widest possible representation of the input space.
2. **Validation Set**: This is the set of data used to evaluate your network's performance during training. When the network's validation performance has peaked, training is complete. This partition is usually smaller or the same size as the test set and its goal is to test whether the network can generalize well, i.e. whether the network can predict an input-output relationship for data that has never seen before.
3. **Test Set**: This is the data your network never sees during training. Once training is complete, the SNN's final performance is evaluated on the test set. This partition acts as the unseen data likely to be encountered in the real world. The size of the test set can range from $100 - 10\%$ the size of the training set.

### 2.2 Exercises

1. Modify your previously built single layer SNN to accommodate the digit dataset. Explain your modifications as follows:
   (a) How many input neurons does your network now have? Why?
   (b) If your network does not predict anything, but rather randomly picks one output value, what would be its accuracy? Note that the reported accuracy is called "chance level".
   (c) Suppose you are training your network to only classify digits $1, 2, 5$. How many output neurons will you have? Why? How many output neurons would you use if you train the network classify all 10 digits?
   (d) If you changed your input data encoding method, describe the new encoding method you are using.
2. Train your network to classify digits $1, 8$. Provide a figure showing change in validation data accuracy as training proceeds. Also, state the accuracy achieved on test data. Now train and show results for digits $3, 8$. How do the test accuracies compare? Why do you think you see this relationship?
3. Train your SNN on all 10 classes. Provide a figure showing validation accuracy as training progresses. State the accuracy achieved on the test data. Is your SNN's accuracy better than "chance level"?

Note: You do not need to chase the highest possible accuracy. However, you do need to show that your network performs better than random chance.

## References


[1] Sen Song, Kenneth D. Miller, and Larry F. Abbott. Competitive Hebbian learning through spike-time-dependent synaptic plasticity. *Nature neuronscience*, 3(9):919-926, 2000.

[2] Andrzej Kasinski and Filip Ponulak. Comparison of supervised learning methods for spike time coding in spiking neural networks. *International Journal of Applied Mathematics and Computer Science*, 16:101-113, 2006

[3] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. *Spatio-temporal backpropagation for training high-performance spiking neural networks*, Frontiers in neuroscience 12 (2018): 331.