Brain Inspired Computing: Assignment 2

Konstantinos P. Michmizos
TA: Vladimir Ivanov

Alfonso Buono (ajb393)
Andrea Cho (aac223)
Shyam Patel (spp128)
Kathleen Wilcox (krw93)

Due: December 8th, 2020

1 Division of Labor

All members of the team contributed in discussions and peer reviews of each others work throughout the project effort. All member participated in the implementation and research of the various models. Upon request, we can grant the grader or the professor access to the private git repository where our source code and commit log can be accessed. The python and notebook files are provided alongside this report.

2 Problem 1

1. How would you represent the inputs x, y using temporal encoding?

   We will first denote the following values: T is the total simulation time. The input neurons could simply be encoded as a vector spike times along the spike train which is of length $\frac{T}{timestep}$ and contains values of 0's and 1's to denote nonspiking and spiking behavior, respectively.

2. How would you represent the inputs x, y using a rate encoding?

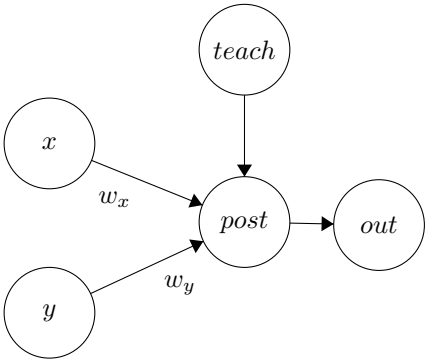
   We will first denote the following values: T is the total simulation time, dt length of the time frame between two arbitrary points t_1 and t_2, $r_{neuron}n$ is the fire rate of either input neuron over time T. To determine whether either input neuron is spiking we use the probability defined by $p = r_{neuron} * dt$. Therefore, for each time step we generate a random variable within the range [0, 1) and compare it to p to determine if a given

input neuron spikes.

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) A graphic of the network architecture, similar to Figure 1

   Note: The teach neuron is not in the network. Instead, it lives outside of the network. However, we included it to explain other parts of this question. Out is just our decoded output and not a neuron.



   (b) Describe the encoding and decoding scheme you used.

   The input neurons x and y are to be encoded as a firing rates. Given inputs of 0's or 1's we will represent them as 1Hz and 4Hz firing rates, respectively. For decoding the output, we are effectively using a binary step activation function for our firing rate. We will check the firing rate of the post neuron to see if it is at least 4hz. If the post neuron's firing rate is equal to that of the encoded fire rate for an input of 1, then the output is true; otherwise, the output is false.

(c) Describe the learning rule you used.

We opted to use the Oja rule for rate-based Hebbian learning because we will be decoding the output based on the post neuron's fire rate, which is induced by the fire rate of the input neurons, x and y.

(d) Describe the training process you used.

Our training process is fairly simple–we have a set of inputs, along with their respective output label in the form of an array of tuples of length three. The first two values in the tuple are the inputs' values of either 0 or 1 for our x and y neurons. The third value is the respective output label, based on the two inputs. We then run a simulation for each of the inputs in which we train the network using Oja's rule. To ensure Oja's rule is adjusting for the weights accordingly, we employ an external teacher neuron that makes sure the post-synaptic neuron fires when it should. We run this for 5 epochs.

(e) 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.

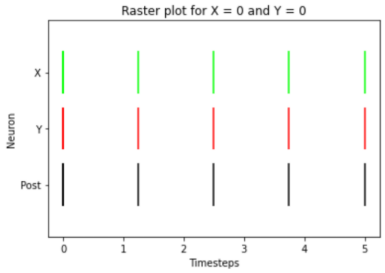


Figure 1: AND function with input X=0, Y=0 decoded to output = 0.

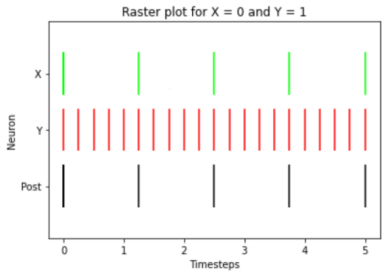


Figure 2: AND function with input X=0, Y=1 decoded to output = 0.

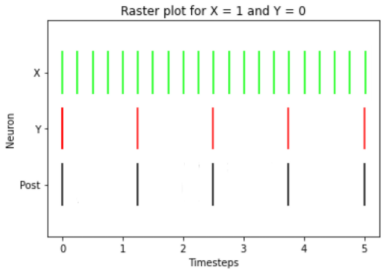


Figure 3: AND function with input X=1, Y=0 decoded to output = 0.

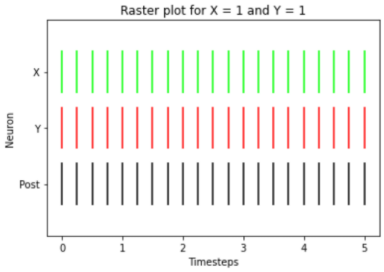


Figure 4: AND function with input X=1, Y=1 decoded to output = 1.

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.

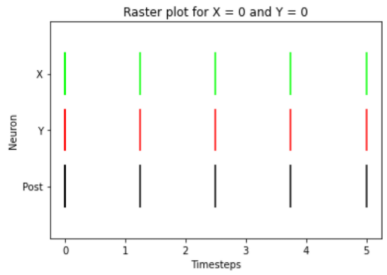


Figure 5: OR function with input X=0, Y=0 decoded to output = 0.

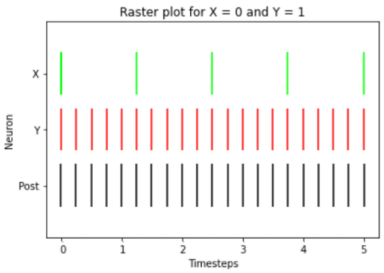


Figure 6: OR function with input X=0, Y=1 decoded to output = 1.

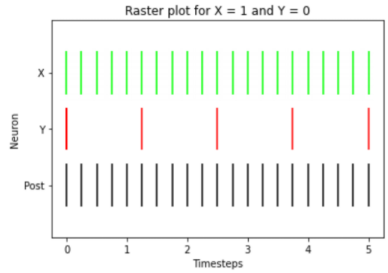


Figure 7: OR function with input X=1, Y=0 decoded to output = 1.

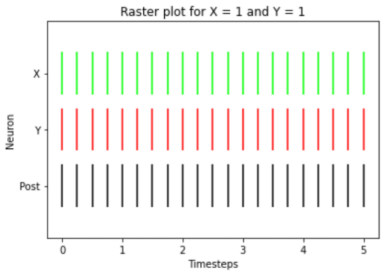


Figure 8: OR function with input X=1, Y=1 decoded to output = 1.

The network should be able to learn the OR function since we are decoding our output via a binary step activation on the post neuron's firing rate. The only change that would need to be made would be to lower the firing rate threshold for activation.

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.

   Our single layer network cannot learn an exclusive or operation because XOR is not linearly separable and cannot be learned on a simple perception [2, 3, 4], like the one that we built. As such, it would require at least 2 layers.

3 Problem 2

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?

   The network will contain 64 input neurons because each of the hand drawn digit images are 8 pixels wide and 8 pixels tall. Thus, we will need an input neuron for each pixel in the image.

   (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".

The expected accuracy would be around 10% because there are 10 labels in the digit dataset.

(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?

If we were training the network to only classify 1, 2 and 5, then we would only require 3 output neurons where each neuron represents one of the three digits. If we wanted to train the network to classify all 10 digits, we would need 10 output neurons as per the aforementioned logic for classifying three digits.

(d) If you changed your input data encoding method, describe the new encoding method you are using.

We used the Poisson process to create a neuron simulation that has behavior that is closer to the behavior of a real neuron.

Input: An image array of size 64. Each of the 64 entries represented one pixel. A pixel can have a value ranging from 1 to 16, 16 being the color white and 1 being the color black. Each pixel has a corresponding input neuron. A neuron will typically fire when the corresponding pixel has a higher value.

Algorithm: For Poisson Process, we decide if a neuron will "fire" by multiplying the probability that a neuron will fire, which is:

$$fr * dt$$

where fr is the firing rate and dt is the time step.
To get the firing rate, we normalize the image array by taking the largest value in the image array and dividing each entry by the max value. Then we continue by using the Poisson Process [1]. We generate an array of random numbers that is the same size as the image array. The random values range from 0 to 1. We compare each entry in the random array to product of the corresponding firing rate of the current neuron and the time step. If the random number generated is less than the probability, then the neuron will fire. We then return an array of size 64 in which each entry has a value of 0 or 1 depending if the corresponding neuron fired according to this algorithm.

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?

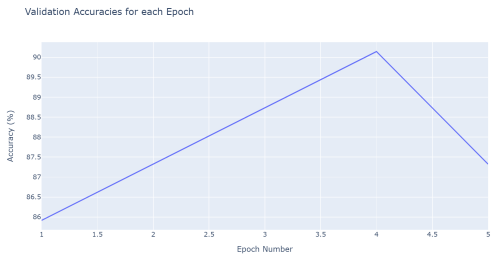


Figure 9: Change in validation data accuracy as training progresses.

Our accuracy for the test set was 87.38%.

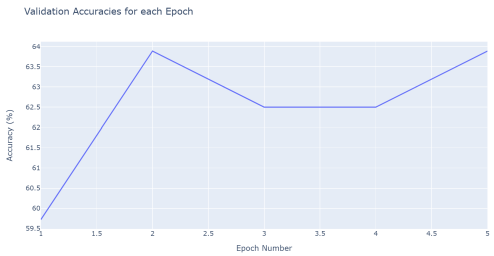


Figure 10: Change in validation data accuracy as training progresses.

Our final testing accuracy reached 51.39% accuracy.

The accuracy for training and testing on 1 and 8 is significantly better than the accuracy of training and testing on 3 and 8. The reason why we believe this to be the case is because a 3 has a similar shape to an 8 and as such the network can confuse the two. However, a 1 is pretty distinct to an 8 and can thus easily distinguish between the two.

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"?

   As our network trained more and more the validation accuracy fluctuated between different values but ended up higher than the initial value.

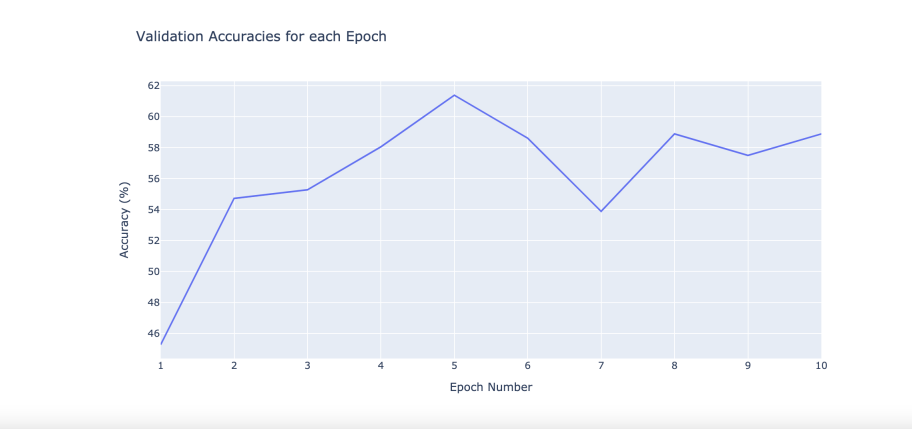


Figure 11: Change in validation accuracy as training progresses.

Our SNN's final test accuracy reached 58.89% accuracy. This is well above "chance level", which would be 10% (the probability of selecting 1 correct digit out of all 10 possible digits 0-9).

References

[1] Devienne Fatahi Shahsavari Ahmadi, Ahmadi. evt_mnist: A spike based version of traditional mnist. *1st International Conference on New Research Achievements in Electrical and Computer Engineering*, 2016.

[2] M. Arbib. Review of 'perceptrons: An introduction to computational geometry' (minsky, m., and papert, s.; 1969). *IEEE Transactions on Information Theory*, 15(6):738–739, 1969.

[3] André Grüning and Ioana Sporea. Supervised learning of logical operations in layered spiking neural networks with spike train encoding. *Neural Processing Letters*, 36, 12 2011.

[4] M. Reljan-Delaney and J. Wall. Solving the linearly inseparable xor problem with spiking neural networks. In *2017 Computing Conference*, pages 701–705, 2017.