Assignment 2

CS 425/525: Brain Inspired Computing
Instructor: Konstantinos Michmizos, Fall 2020
Rutgers University

Seth Karten (sak295), Madhu Sivaraj (ms2407), Alex Goodkind (amg540)
Due Date: Dec 8, 2020

Contributions

Everyone contributed towards the theoretical questions during discussion. Madhu and Alex took notes and wrote down the solutions agreed upon. Seth was the leader on the programming. Madhu helped visualize the data. All three put the report together in LaTeX.

Our code is in an attached jupyter python notebook file.

Exercises

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

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

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

In temporal encoding, we could represent the inputs by applying Time-to-first-spike encoding. Here, we would assign importance to the precise timing of the very first spikes after a neural network is stimulated. The strength of stimulation is coded into the time-to-first-spike, triggered by an external stimulus. The stronger a stimulus, the earlier the spike. In an idealized network each neuron only fires once, after that enters a refractory period until the next stimulus of the network occurs.[1]

Essentially, this SNN model would have a dynamic threshold in order to reduce the number of spikes and latency of inference. For example, once a neuron spikes, it would not spike again so there would be a noticeable different between spike timing for the two neurons. After considering

the refractory period, the firing rate will be the same for both neurons. The output neuron would also be represented using a time-to-first-spike encoding. The time to spike can inversely be encoded in current, that is, the current will be higher for earlier spikes and lower for spikes that took a longer time. Thus, a 1 (or True) input corresponds to a low time-to-fire, creating a large input current and a 0 (or False) input will have a high time-to-fire, creating a low input current. When decoding, a high current will represent a 1 and a low current will represent a 0 since they correspond to a low and high time-to-fire respectively.

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

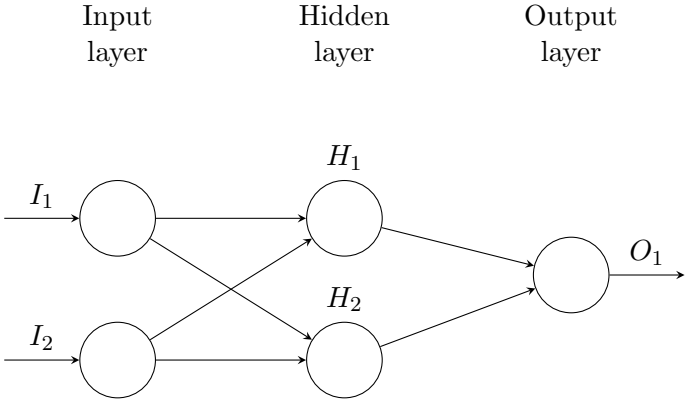
As we know, rate coding assumes the spike count rate as basis of coding in neural systems, and that the rate a neuron fires spikes carries all the information. The spike count rate is found by counting the spikes in a time interval Δt and dividing it by Δt, what means calculating the temporal average. The length of the time interval depends on the type of neuron and the stimulus (typical one hundred or a few hundred milliseconds). [1]

In our implementation, we represent the inputs x and y with the rate encoding described [1] by applying a spike count rate averaged over time.

For example, the high spike count rate would output 1 (true) and low spike count rate would output 0 (false).

1.2.3 Build a single layer SNN that can learn the AND function.

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



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

We implemented our single layer SNN using a rate encoding (refer to question 1.2.2). We encode both high and low firing rates and if the input value is True, it is a high (or 1) firing rate, and if False, it is a 0 (or low) firing rate. For decoding, we translate a high firing rate to a True boolean value and a 0 (or low) firing rate to a False boolean value. This enables us to only have 1 output neuron.

(c) Describe the learning rule you used.

We applied a rate-based Hebbian learning with Oja's rule, decay, and postsynaptic threshold rule. In Hebbian learning, a synapse between two neurons is strengthened when the neurons on either side of the synapse (input and output) have highly correlated outputs.

In essence, when an input neuron fires, if it frequently leads to the firing of the output neuron, the synapse is strengthened. Following the analogy to an artificial system, the tap weight is increased with a high correlation between two sequential neurons.[2]

Oja's rule provides weight regularization similar to an L2 Norm. We also use Oja's rule which converges asymptotically to synaptic weights that are normalized to. The sum implies competition between the synapses that make connections to the same postsynaptic neuron. So, for example, if some weights grow, others must decrease. The decay will provide a decrease in the weight between the pre and post synaptic neurons when they do no both fire. The postsynaptic threshold rule decreases the weight between the pre and post syanptic neurons when the presynaptic neuron fires but the postsynaptic neuron does not fire. Since weights can become arbitrarily large, there is no mechanism for weights to decrease. This keeps the weight matrix from growing without bound. We also apply Hebb with postsynaptic LTP/LTD threshold. This decreases the weights when the presynaptic fires but the postsynaptic does not fire.

(d) Describe the training process you used.

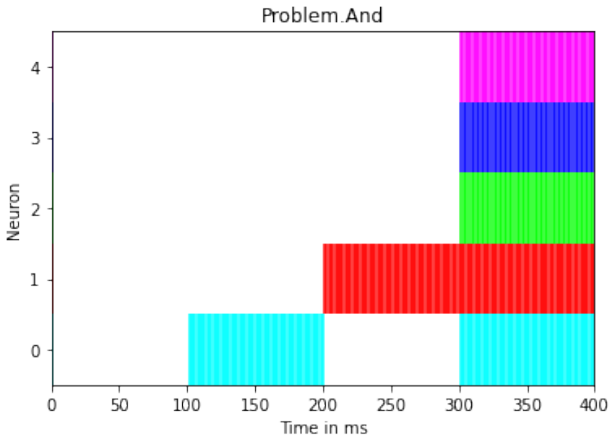
The training process we used for our network starts with a forward computation of the entire network. When we know that a neuron fires, we multiply its weight by a base current to determine the input current to the postsynaptic neuron. We use the LIF equation to calculate our spike count rate, below. If the current is above a threshold current $I_{th} = \frac{V_m}{R_m}$, we know that it will have a high firing rate.

$$f(I) = \begin{cases} 0, & I \leq I_{\text{th}} \\ [t_{\text{ref}} - R_{\text{m}} C_{\text{m}} \log(1 - \frac{V_{\text{th}}}{I R_{\text{m}}})]^{-1}, & I > I_{\text{th}} \end{cases}$$ [3]

Next, we train layer by layer, first with the hidden layer. We ensure that the outputs for the hidden feature space match the expected outputs during learning by using a learning neuron for each neuron in the hidden layer. The same training is then performed similarly for the output network. We remove the learning neurons when we perform validation and test accuracy. We additionally decay the learning rate over time to converge to the optimal weights.

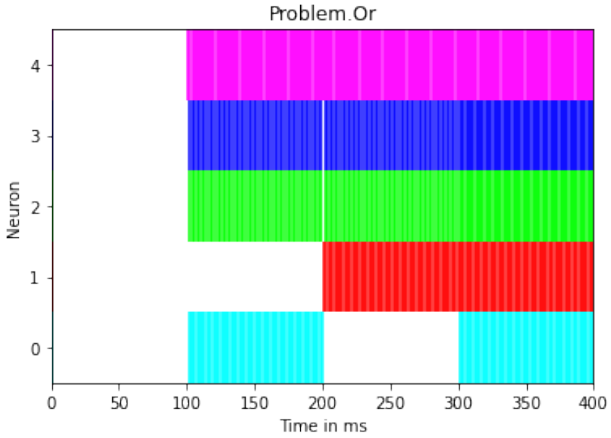
(e) Demonstrate that your SNN learned the AND function by visualizing network activity in the form of a raster plot with each neuron labeled for the decoded input/output.

Yes, our single-layer SNN can learn the AND function. Below is a visualization of our network activity in the form of a raster plot. The y axis contains the ith neuron such that the ith neuron corresponds to the index of [input 1, input 2, hidden 1, hidden 2, output 1], respectively.



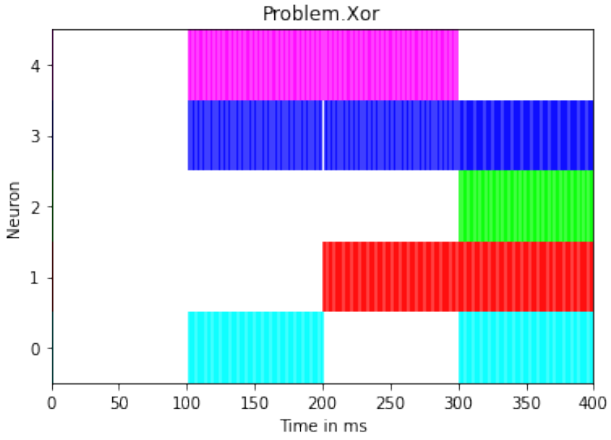
1.2.4 Can your single-layer SNN learn the OR function?

Yes, our single-layer SNN can learn the OR function. Below is a visualization of our network activity in the form of a raster plot. The y axis contains the ith neuron such that the ith neuron corresponds to the index of [input 1, input 2, hidden 1, hidden 2, output 1], respectively.



1.2.5 Can your single-layer SNN learn the XOR function?

Yes, our single-layer SNN can learn the XOR function. Below is a visualization of our network activity in the form of a raster plot. The y axis contains the ith neuron such that the ith neuron corresponds to the index of [input 1, input 2, hidden 1, hidden 2, output 1], respectively.



2.1.2 Training Methodology Partitions

We split our training data into 3 sets: 20% test, 16% validation, and 64% training. The test accuracy is found from the epoch during which our weights were tuned to predict the labels best on the validation set.

2.2.1 Modify your previously built single layer SNN to accommodate the digit data set. Explain your modifications as follows:

(a) How many input neurons does your network now have? Why?

Our network's number of inputs is equivalent to the number of pixels in image based on our rate based encoding. Each input neuron has a firing rate associated with each pixel in the image. We use an 8x8 image so we have 8x8 = 64 pixels = 64 inputs. This is intuitive because we need to process the data for every pixel.

(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".

$$\frac{1}{num\_outputs} * 100$$

In this problem, using random chance, it would be 10% because we have 10 outputs because we have one output for each digit.

(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 to classify all 10 digits?

When training to only classify three digits (1,2,5), we will have 3 output neurons based on our rate based decoding. The neuron with the highest firing rate is corresponds to our predicted label. This is because the number of output neurons is always equivalent to the number of

classes, per our network decoding process. If we train the network to classify all 10 digits, we would use 10 output neurons.

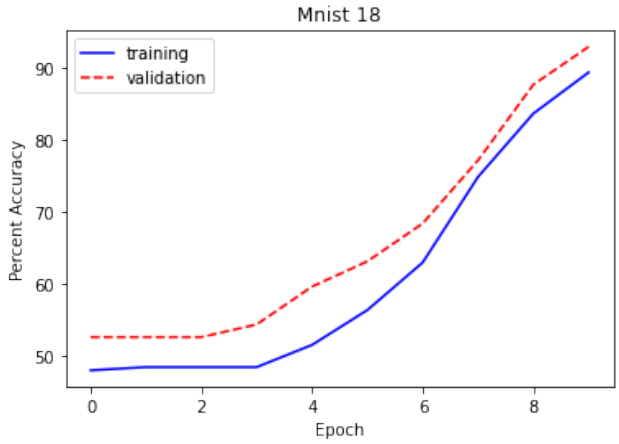
(d) If you changed your input data encoding method, describe the new encoding method you are using.

In our network, it will have a high firing rate if the pixel intensity greater than our defined threshold; otherwise it will have a low firing rate. The threshold was found quantitatively based on performance (best accuracy). We found that a threshold of 6 found the best results with our network features.

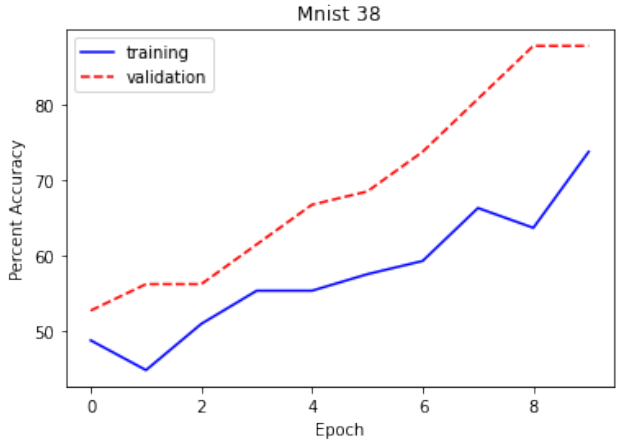
Additionally, it is worth noting that our network architecture is trained to determine horizontal and vertical lines in the hidden layer's feature space. Imagine the pixels in the 8x8 image compressed into 2 3x3 grids, one which describes vertical lines and one which describes horizontal lines. We used 18 hidden neurons to describe 9 vertical lines and 9 horizontal lines that can be found.

2.2.2 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?

Training curve for the classification of digits 1,8 is below. The test accuracy was 78%.



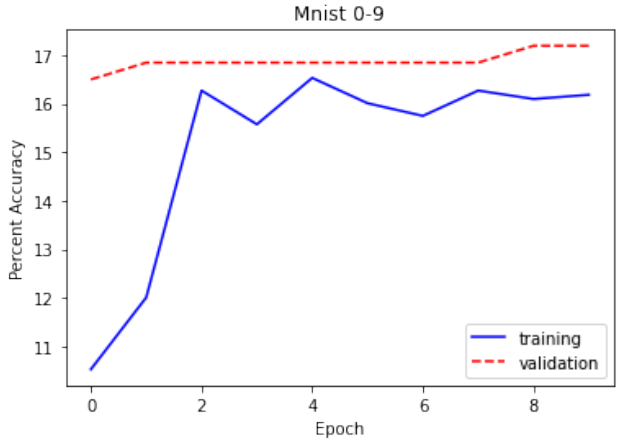
Training curve for the classification of digits 3,8 is below. The test accuracy was 73%.



The test accuracy for our SNN was higher for the 1,8 problem because it is easier to distinguish between 1,8 than 3,8. Since the features between 3 and 8 are more similar than 1 and 8, it is harder to distinguish between 3 and 8.

2.2.3 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"?

Training curve for the classification of digits 0 to 9 is below. Our test accuracy was 16%. For other seeds, we have found test accuracies as high as 20%. The training for our full mnist is very dependent on the seed since we randomly select the order in which the training data is used by the network each epoch.



Since random chance would have, in expectation, 10% accuracy, our model does better than chance.

References

[1] M. Kiselev. "Rate coding vs. temporal coding - is optimum between?" In: *2016 International Joint Conference on Neural Networks (IJCNN)*. July 2016, pp. 1355–1359. DOI: 10.1109/IJCNN.2016.7727355.

[2] Tomasz Szandała. "Comparison of Different Learning Algorithms for Pattern Recognition with Hopfield's Neural Network". In: *Procedia Computer Science* 71 (2015). 6th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2015, 6-8 November Lyon, France, pp. 68–75. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2015.12.205. URL: http://www.sciencedirect.com/science/article/pii/S1877050915036662.

[3] Wulfram Gerstner and W Kistler. "Spiking Neuron Models". In: Jan. 2002. ISBN: 0521890799. DOI: 10.1017/CBO9780511815706.002.