Sparse Hebbian Networks for Higher Training Accuracy with Limited-Size Training Datasets [*]

Phillip Allen Lane
phillip.a.lane@vanderbilt.edu

November 30, 2022

1 Introduction

In this term project, I investigate Hebbian networks with limited training data. I demonstrate that increasing sparsity and dropping "dead" neurons which do not influence the result can significantly increase training accuracy (up to 29%) when tested on a subset of the MNIST dataset, from LeCun, Cortes, and Burges (1998), with only 500 training patterns. My tests show a traditional dense neural network with Hebbian learning and Oja's rule can only reach about 45% accuracy at the maximum, but usually falls closer to 35%. However, a sparse network can reach 55% accuracy or higher.

2 Background

Dense Hebbian learning, from Polyn (2022), is traditionally done by initializing a weights matrix w with random numbers under a normal distribution. After initialization, training a network is performed by picking a small λ (sometimes known as the *learning rate*), usually around 0.01, and updating each w_{ij} according to the following rule:

$$\Delta w_{ij} = \lambda i_i o_j.$$

After all of the neurons have been updated for a given input in the training dataset, *Oja's rule* is then applied, which normalizes the matrix such that each output neuron's input vector has magnitude 1, i.e, $||w_{*j}|| = 1$

3 Constructing a Sparse Hebbian Network

Original to this work, the only difference between my dense Hebbian network (DHN) and my sparse Hebbian network (SHN) is in the initialization of w. Instead of filling w with random numbers, I connect the first input neuron to *only* the first output neuron, the second input neuron to *only* the second output neuron, etc., until the 11^{th} input neuron wraps around to being connected again to the first output neuron. This is easily visualized by a "stack" of identity matrices with a height equal to the number of input neurons. The

[*]Term project report for NSC 6270 at Vanderbilt University, Dr. Sean Polyn

number of input neurons is equal to $28 \times 28 = 784$, which is the dimensions of the MNIST training dataset.

My SHN is trained identically to my DHN, meaning that my sparse matrix starts out with a sparsity of 90% and slowly drops as it incurs *fill-in*. This fill-in is expected and desired, because now I am only setting relevant neurons for classification, which is tested to significantly improve training accuracy on small, limited datasets. In a biological context, this can be visualized as starting with a loosely-connected network of neurons. As knowledge is obtained, only relevant connections are made between neurons, leaving unrelated neurons unconnected.

4 Test Methodology and Results

I construct a DHN and a SHN and train both networks on a random 500 entries in the MNIST dataset to simulate a severely limited training set. The random 500 entries are chosen by randomly permuting the full MNIST dataset, then choosing 50 of each type of digit. This is done to ensure that there is no severe underrepresentation or overrepresentation of certain patterns. For testing, I test on all $10,000$ entries in the test dataset. Both the DHN

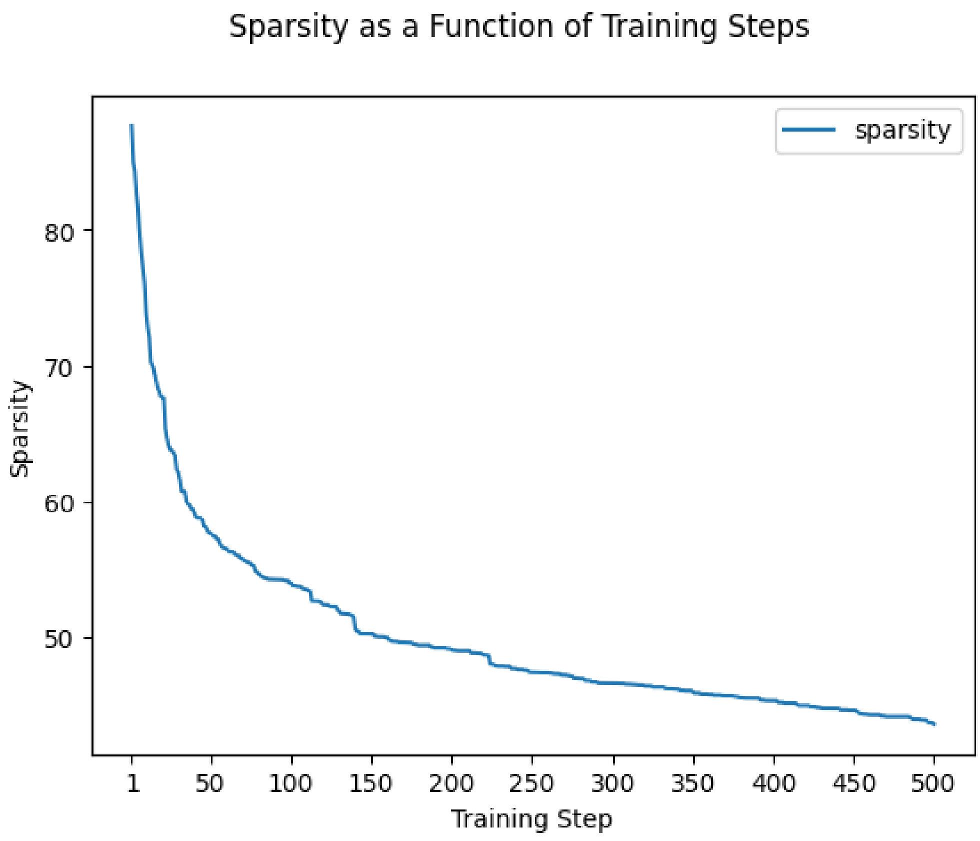

Figure 1: Sparsity as a function of training steps for the SHN. Each entry among the x axis specifies the sparsity after a particular datum in the data set is processed. The y axis represents sparsity as a percentage.

and SHN use the same training kernel. Again, the only difference is in the initialization. λ is chosen as 0.01.

I find that my sparse network incurs substantial fill-in, dropping from 90% sparsity to about 43% sparsity. The fill-in is visualized in Figure 1. However, this still means that only 57% of possible connections are made or needed.

As for performance, I find that my SHN reaches an accuracy of around 55% or higher, reaching as high as 59% in testing. On the other hand, the DHN usually gets stuck at 45% or worse, dropping as low as 29% in testing. The substantial fill-in means that using a dedicated sparse matrix storage format like compressed sparse row (CSR) does not save on storage space. In a sense, w is transformed into a dense matrix through training. However, I note the substantially higher training accuracy as the selling point of this term project.

5 Discussion and Other Remarks

When I trained both networks using the first $5,000$ entries in the MNIST dataset, both networks trained to about the same accuracy (around 70% accuracy). This demonstrates that sparsity can't *hurt* performance. Because I am training in software, I opt not to test on the entire $60,000$-entry dataset, because training would then take several hours. I leave this up to the reader to try, but would recommend implementing the networks in a compiled language like C before attempting. I also tried initializing w to a zero matrix and only performing Oja's rule if $||w_{*j}|| > 0$ (to avoid divide-by-zero errors) but accuracy was profoundly poor, only slightly better than random guessing.

Finally, I tried randomizing the connections in the SHN in two ways. The first way was done by randomly assigning each input neuron to one output neuron, and the training accuracy was about the same as the DHN. The second way was done by randomly assigning each input neuron to one output neuron with the guarantee that each output neuron would have the same number of incoming connections. The training accuracy comes out better than the DHN, but not as well as the construction described in Section 3, topping out at about 50% accuracy.

References

LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). *The mnist database of handwritten digits.* Retrieved from `http://yann.lecun.com/exdb/mnist/`

Polyn, S. (2022, November 23). *Week 4b slides.*