MNIST Digit Classificaton using SNN

Arisht Daiya

July 2, 2025

1 Data Preprocessing

I first downloaded the MNSIT dataset and accessed it using the pandas dataframe. Then I normalized the pixel values and implemented **Poisson Rate Decoding** to convert these into spike trains. For the poisson rate decoding, I took 100 time steps and generated a random integer for each one and compared it to the normalized pixel value, if it is less than the pixel value, it returned 1 which meant spike else it returned 0.

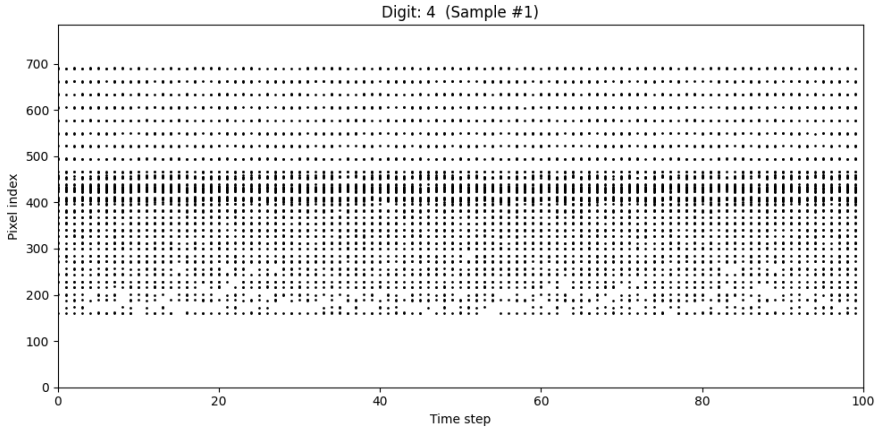

Figure 1: SpikeMap for Digit 4

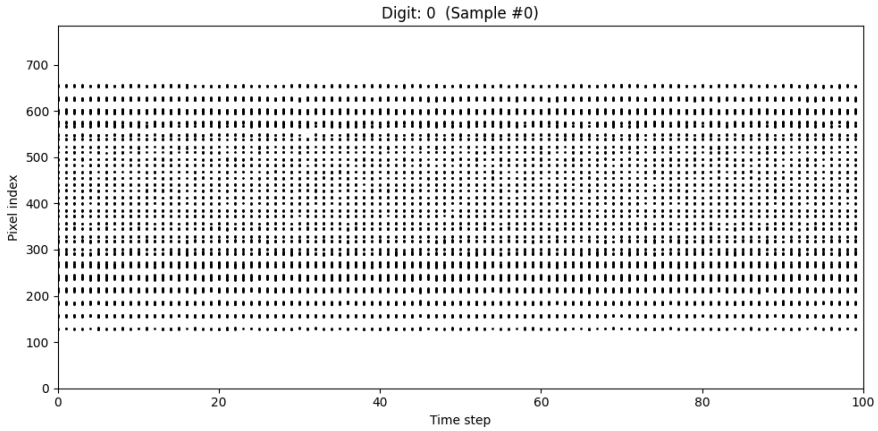

Figure 2: SpikeMap for Digit 0

2 SNN Model Implementation

For the model architecture, I went with an input layer of 784 neurons(flattened 28x28 pixels), a hidden SNN layer with LIF model, and an output layer with 10 neurons(one for each digit). The LIF layer consisted of 256 neurons. Custom LIF layer modules simulate membrane potential dynamics, with threshold-triggered spike generation which leak over time and reset once a spike is triggered. Since spikes are binary and non differentiable, we used a surrogate gradient approach. The Surrogate Spike class defines a fake gradient using the derivative of the sigmoid function to enable gradient-based training. We used on the fly Poisson rate encoding as it is computationally less expensive while giving almost the same results.

3 Training and Evaluation

For the training, we took batches with batch size 64. We took 100 time steps for each image which were properly preprocessed and encoded them batch wise to save memory and time. The model used Adam optimizer with learning rate e-3 to minimize the cross entropy loss. We trained the model for 5 epochs, measuring the training accuracy in each case. The results can be seen below:

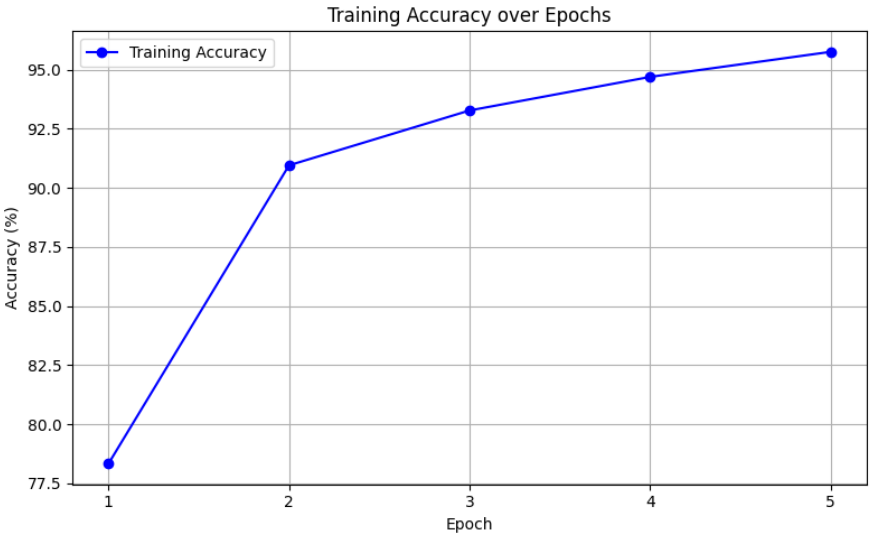

Figure 3: Accuracies for different training episodes

The cross entropy loss thus decreased with consecutive epochs.

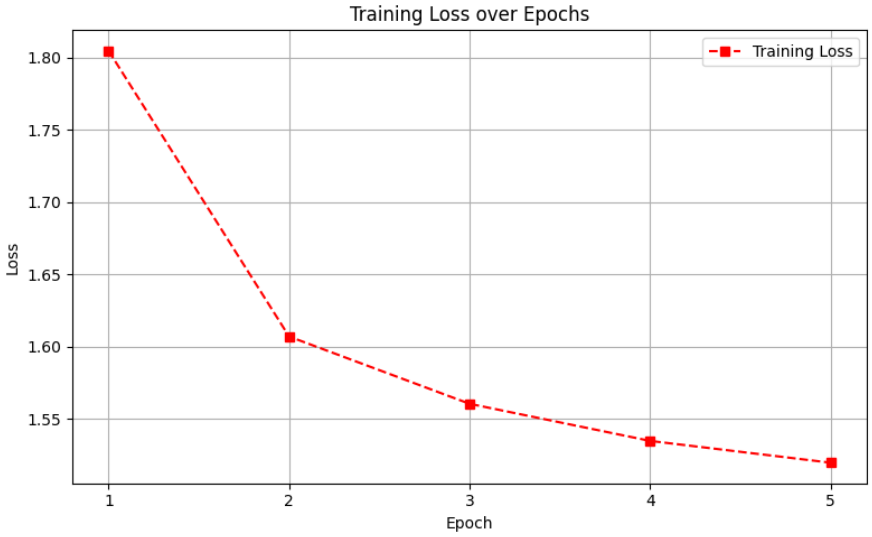

Figure 4: Cross entropy losses for different epochs

4 Challenges Faced and their Solutions

4.1 The problem

Encoding all spike trains at once drastically increases memory requirements. A single MNIST image (28×28 pixels) contains 784 values, but after Poisson encoding with, say, 100 time steps, each image expands to a (784 × 100) array — or 78,400 binary values. For the full MNIST training set of 60,000 images, this amounts to nearly 4.7 billion values, consuming gigabytes of RAM even when stored efficiently. This makes the method unsuitable for systems with limited memory or for larger datasets.

4.2 The solution

On the fly Encoding : To address the limitations of precomputing spike trains in spiking neural networks (SNNs), we implemented a solution based on on-the-fly Poisson encoding. This approach dynamically generates spike trains for each image during data loading, just before feeding them into the network. By generating spike trains only when needed, the system avoids storing large spike arrays for all 60,000 images at once. This enables the use of long time windows (e.g., 100 steps) and large datasets without exhausting RAM or GPU memory.

5 SNN vs ANN : Observations and Differences

Training Time and Speed SNNs typically learn more slowly than ANNs. Surrogate gradients are an approximation; they don't propagate true gradient signals like ReLU or tanh in ANNs. LIF neurons introduce memory (leaky integration), but this temporal dynamic makes training less direct. This can be seen clearly as the initial training accuracy was pretty low(78.33%) which increased over consecutive epochs. Also it took a decent amount of time to train the model.

Efficiency SNNs use sparse, event-driven computation. Neurons only "fire" when membrane potential exceeds a threshold. This leads to a lot of zeros in spike tensors → potential for hardware efficiency. Even though our SNN has a large input size over time (784 × 100), only a small fraction of inputs are spikes. This sparsity can be exploited on neuromorphic hardware (e.g., Intel Loihi, SpiNNaker) for massive energy savings — something not possible with dense ANN activations.

Accuracy With good training, we achieved an accuracy of 96.34% which is pretty decent and comparable to a standard ANN. This can be further increased on using specialised hardware discussed in the previous section.

References

- Y. LeCun, C. Cortes, and C. J. Burges, "MNIST Handwritten Digit Database," https://www.kaggle.com/datasets/hojjatk/mnist-dataset
- Poisson rate encoding," https://medium.com/@baxterbarlow/poisson-spike-generators-stochastic-theory-to-python-code-a76f8cc7cc32
-