EE 533 – Final Project

Note: Undergraduate students will work in pairs and complete Tasks 1-9. Graduate students will also work in pairs and complete Tasks 1-10.

Overview

You will design, train, and evaluate spiking neural networks (SNNs) for multi-class classification on the MNIST dataset. You'll explore how network size, data size, training method, and weight precision affect performance, and propose an optimal SNN that balances accuracy with hardware efficiency, and implement a single neuron of the network using an analog neuron (graduates only).

Tasks

1. Build a Spiking Neural Network (SNN). Use the Python package SpikingJelly based on PyTorch [1-2] to create a multi-layer, fully connected SNN.
    a. The output layer must have 10 neurons (one for each MNIST digit: 0–9).
    b. It is up to you how many layers and how many neurons per layer the network has.
2. Resize MNIST Images. The original MNIST images are 28×28 pixels. Try using resized versions of the images at:
    a. 4x4
    b. 7×7
    c. 14x14
    d. 28×28 (i.e., the original sizes)
3. Encode Images into Spikes. Use Poisson spike encoding to turn the images into spike trains. Test using different numbers of time steps of
    a. 5
    b. 10
    c. 20
4. Train the Network with Two Methods. Try both of these training methods:
    a. Surrogate gradient descent (*supervised* learning)
    b. Spike-Time Dependent Plasticity (STDP) (*unsupervised* learning)
5. Quantize the Weights. After training, reduce the precision of the weights by using:
    a. 2-bit
    b. 3-bit
    c. 4-bit
    d. 5-bit quantization
6. Plot Spike Activity. Create spike raster plots to visualize the spiking behavior of the network.
    a. The point is you want to visually verify the network operates properly.
    b. Do this for the input encoding, hidden layers, and output layer.
    c. Use a few example digits to test the plots seem reasonable (especially the input encoding and the output layer).
7. Measure Accuracy. Compare the classification accuracy for different:
    a. Image sizes
    b. Time steps
    c. Training methods
    d. Weight quantization levels

8. Make a Smaller Network. Try reducing the number of layers and the number of neurons in each layer.
    a. Repeat steps 1–7, this time using a smaller number of layers and number of neurons per layer.
    b. Target an accuracy of >75%.
9. Write a Report. Create a report that presents and compares different networks, image sizes, training methods, and weight quantization levels in terms of the accuracies and latencies (i.e., time steps). See the "Report Requirements" section below.
10. **(Graduate students only)** Simulate one output neuron using hardware and compare it with the Python model. Use Cadence Virtuoso to implement a single output neuron of the most accurate network/input/training configuration. Use the Integrate-and-Fire neuron circuit shown in class. Modify the circuit to take spikes from the previous layer while weighted by 3-bit weights. Use actual spike data from Python (raster plots) as inputs to your circuit and the 3-bit synaptic weight matrix from Python as weights in your circuit.
    a. Output comparison. Simulate the neuron's spike output in Cadence and compare it to the corresponding Python model.
        i. Perform this comparison using three sample images for each digit (0–9).
        ii. Quantify how closely the hardware output matches the Python simulation.
    b. Power analysis. Simulate the power consumption of your hardware output neuron circuit.
        i. Estimate the total power that would be required if the full network were implemented in hardware.

Report Requirements (20 points)

(Report submission deadline: Wednesday May 7th, 11:59PM.)

The final report must be clear with plots and tables in the report, and codes in the appendix. Follow the following report structure.

1. Introduction
    a. A brief overview of the project
    b. A brief summary of the objectives
2. Methodology
    a. Network Design
        i. Details of your SNN architecture: number of layers, neurons per layer with a diagram of the network.
    b. Image Preprocessing
        i. Describe how you resized MNIST images to: 4x4, 7×7, 14×14, and 28×28.
    c. Spike Encoding
        i. Briefly explain Poisson spike encoding and discuss the anticipated impact of time steps: 5, 10, and 20.
    d. Training Methods
        i. Briefly describe the operation of surrogate gradient descent (supervised learning) and STDP (unsupervised learning)

    e. Weight Quantization
        i. Briefly describe how weights were quantized to: 2-bit, 3-bit, 4-bit, and 5-bit levels and explain the motivation for quantization (e.g., hardware friendliness).
    f. Accuracy & Performance Evaluation
        i. Present and compare results for different image sizes, time steps, training methods, and weight quantization levels
        ii. Include both accuracy and latency (time step count).
    g. Network Optimization
        i. Describe how you reduced network size (fewer layers/neurons). Show results for the "best performing" small network. Justify your final proposed architecture in terms of accuracy vs. complexity. Comment on whether and why the target accuracy of >75% is achieved or not.
        ii. For the "best performing" small network, provide a histogram of the synaptic weights for all layers.
    h. Spike Activity Visualization
        i. For the best performing small network, provide spike raster plots for input encoding, hidden layers, and the output layer. Use sample inputs (e.g., digits 6, 8, 0) and provide observations (e.g., which neuron in what layer is the most or least active, what that might mean)
3. <u>Summarize your findings:</u>
    a. What network/input/training configurations worked the best/worst?
    b. How did accuracy change with the different parameters?
    c. Which training method was more effective?
    d. etc.
4. <u>Appendix:</u>
    a. Codes
5. <u>References:</u>
    a. Any references used should be properly cited with full details.
6. <u>(**Graduate Students Only**) Hardware Implementation</u>
    a. Circuit Implementation
        i. Describe how you implemented an output neuron using Cadence Virtuoso with schematics and a summary of device/capacitor sizes for repeatability.
        ii. Explain how you modified the Integrate-and-Fire circuit to include:
            1. Input spikes (from raster plots)
            2. 3-bit weighted synapses (from your trained model in Python model)
    b. Output Comparison
        i. Compare the spike outputs from the hardware circuit and Python model using three sample images for each digit (0–9) by quantifying similarity (e.g., spike rate).
    c. Power Analysis
        i. Simulated current consumption of the hardware neuron.
        ii. Estimated total power if the full network were implemented in hardware.

Final Presentation Requirements (10 points)

(Slide submission deadline: Sunday May 4th, 11:59PM.)

(Presentation date/time: Monday, May 5th, 5:30PM-6:45PM)

Each team will give a brief, time-limited presentation to summarize their project results:

a. Undergraduates: **2-minute** presentation per team
b. Graduates: **3-minute** per team.
c. Timing will be strictly enforced.
d. This is **NOT** a summary of your full report. Instead, focus on presenting:
    a. Your best-performing model and its results (all students)
    b. Circuit simulation results (graduate students only)
e. Therefore, be concise and use visuals while highlighting the key points.

References

[1] Fang, Wei, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. "Spikingjelly: An open-source machine learning infrastructure platform for spike-based intelligence." Science Advances 9, no. 40 (2023): eadi1480.

[2] Fang, W., SpikingJelly: An open-source machine learning framework for spiking neural networks. GitHub. https://github.com/fangwei123456/spikingjelly