MNIST DIGIT CLASSIFICATION USING SPIKING NEURAL NETWORKS

BRAIN AND COGNITIVE SCIENCE CLUB, IIT KANPUR

PRATIK KUMAR MAITY

Step 1: Data Preprocessing (Spike Encoding)

In this stage, the MNIST dataset is preprocessed and converted into a temporal spike-based representation suitable for training a Spiking Neural Network (SNN). Unlike traditional Artificial Neural Networks (ANNs), SNNs operate on discrete spikes over time. Therefore, it is essential to encode static image data into temporal spike trains that mimic the way biological neurons transmit information.

Dataset Selection and Initialization

We use the standard MNIST dataset of handwritten digits, which contains 60,000 training images and 10,000 test images of size 28×28 in grayscale. The dataset is downloaded using the PyTorch `torchvision.datasets.MNIST` API. To ensure reproducibility of results, we initialize the random number generators in PyTorch, NumPy, and Python's standard library using a fixed seed.

Only a small subset of 5 images is selected randomly from the dataset to illustrate the spike encoding process. This is done to keep the visualization computationally light while demonstrating the effectiveness of the encoding method.

Poisson Rate Coding

The static pixel intensities of the grayscale images are converted into spike trains using a biologically plausible approach called **Poisson rate encoding**. In this method, a pixel's intensity is interpreted as the firing rate of a spiking neuron. More specifically, each pixel is simulated across $T = 20$ discrete time steps. At each time step, a spike is generated at a particular pixel location with a probability proportional to the pixel's brightness.

Mathematically, this is equivalent to sampling from a Bernoulli distribution at each time step where the success probability is given by the normalized pixel intensity. If the sampled value is below the intensity value, a spike (1) is generated; otherwise, no spike (0) is produced. Over multiple time steps, brighter pixels tend to produce more spikes, while darker pixels rarely spike. This encoding reflects the probabilistic, rate-based behavior of sensory neurons in biological systems.

Spike Train Generation

The spike trains generated from the Poisson encoding process form a three-dimensional tensor with dimensions (T, H, W) where:

- T is the number of time steps (e.g., 20),
- H and W are the height and width of the image (both 28 for MNIST).

Each frame in this temporal sequence represents a binary spike map, where pixels have either spiked (1) or not (0) at that particular time step. The full sequence captures the temporal dynamics of spiking behavior corresponding to the static image.

Visualization of Spiking Activity

To validate the correctness and interpretability of the spike encoding, two visualization techniques are employed:

(a) 2D Spike Visualization Across Time

For each image, we generate and display the spike map at each of the T time steps. This gives an intuitive understanding of which pixels are firing at each point in time. Brighter pixels in the original image will exhibit frequent spiking, and this can be observed by comparing the sequence of frames. This 2D view provides temporal insight on a frame-by-frame basis.

(b) 3D Raster Plot Visualization

To obtain a comprehensive view of the spike train over both space and time, a 3D raster plot is generated. Each spike is plotted in three-dimensional space where the x- and y-axes represent pixel coordinates and the z-axis represents the time step. The color gradient is used to indicate spike timing. This visualization is particularly useful for observing the sparsity and distribution of spiking activity across the entire simulation duration.

Biological Relevance and Intuition

Poisson rate coding is one of the simplest and most widely used spike encoding strategies in SNNs, inspired by the behavior of retinal ganglion cells and cortical neurons. It mimics how real neurons respond to continuous stimuli (such as light intensity) with stochastic spike trains. This encoding scheme is temporally rich, biologically meaningful, and computationally straightforward to implement.

Conclusion of Step 1

This preprocessing pipeline successfully transforms static grayscale images from the MNIST dataset into spatiotemporal spike trains using Poisson rate coding. These spike trains form the input to the subsequent spiking layers in the SNN architecture. The visualizations confirm that the encoded spikes maintain the structural information of the original digits, thus validating the correctness of the encoding step.

Step 2: SNN Model Implementation

In this step, a basic 3-layer Spiking Neural Network (SNN) is implemented from scratch to process and classify temporally encoded MNIST digit images. The model consists of an input layer that receives spike trains, a hidden layer that integrates information, and an output layer that accumulates spike activity to infer the predicted digit. The neurons in both hidden and output layers follow the biologically inspired **Leaky Integrate-and-Fire (LIF)** dynamics.

Network Architecture

The network is designed as follows:

- **Input layer**: Consists of 784 neurons corresponding to the 28×28 pixel inputs. This layer receives binary spike trains generated via Poisson rate coding over $T = 20$ time steps.
- **Hidden layer**: Contains 100 LIF neurons that integrate incoming spikes from the input layer and emit spikes based on a membrane potential threshold.
- **Output layer**: Contains 10 LIF neurons corresponding to the digit classes (0 to 9). This layer aggregates spike activity across time to decide the predicted class.

LIF Neuron Model

Each neuron in the hidden and output layers behaves according to the LIF dynamics:

- **Membrane potential update:** At each time step, the neuron's membrane potential is updated using a leaky integration rule:
$$V(t) = \beta \cdot V(t-1) + I(t)$$
where β is the leak constant and $I(t)$ is the input current.
- **Spike generation:** If the membrane potential crosses a predefined threshold, the neuron emits a spike (value of 1), and its potential is reset to 0.
- **Reset mechanism:** Spikes cause the neuron's membrane potential to reset to 0, preventing continuous firing and enabling temporal sparsity.

This process continues over all T time steps. At each time step, neurons accumulate and decay their membrane potential, mimicking how biological neurons operate under dynamic stimuli.

Weight Initialization and Inference

The network weights between the layers (`W1` for input-to-hidden and `W2` for hidden-to-output) are initialized with small random values from a Gaussian distribution. During inference:

- A single image is selected from the MNIST dataset, normalized, and encoded into a spike train using the Poisson encoding method from Step 1.
- The spike train is passed through the SNN using a custom `run_snn` function that simulates the LIF dynamics.
- For each output neuron, the total number of spikes over all time steps is recorded.
- The neuron with the highest spike count is selected as the predicted digit class.

Interpretation of Output

The output of the SNN is a vector of spike counts, one per output neuron. The class with the highest spike count is chosen as the final predicted label. This spike-based decision mechanism reflects the rate-based coding principle observed in neuroscience, where neurons firing more frequently are considered more activated.

Conclusion of Step 2

This step successfully demonstrates the forward computation of a spike-based neural network using biologically realistic neuron dynamics. The Leaky Integrate-and-Fire model captures temporal integration and threshold-based spiking, both essential for SNN operation. The network is ready for training using either supervised or unsupervised learning mechanisms, which will be discussed in the next step.

Step 3: Training and Evaluation

This step involves training the Spiking Neural Network (SNN) using the MNIST dataset and evaluating its classification performance. The training procedure is supervised and uses **surrogate gradient descent** to overcome the non-differentiability of spike-based neuron models.

Training Setup

The SNN architecture consists of three layers:

- **Input Layer:** Receives the spike trains generated from Poisson encoding of MNIST images.
- **Hidden Layer:** Contains 250 Leaky Integrate-and-Fire (LIF) neurons.
- **Output Layer:** Contains 10 LIF neurons corresponding to digit classes (0–9).

Key hyperparameters include:

- Time steps per input: $T = 50$
- Threshold potential: $V_{th} = 0.5$
- Membrane decay factor: $\beta = 0.98$
- Learning rate: 2×10^{-3}
- Training samples: 10,000 Test samples: 10,000

Learning with Surrogate Gradients

Since spiking activations are binary and non-differentiable, traditional backpropagation cannot be directly applied. To address this, the training uses a differentiable approximation called the **surrogate gradient**, applied to the membrane potential before thresholding.

The surrogate function used is a smoothed, clipped sigmoid derivative:

$$\sigma'(v) = \frac{\alpha e^{-\alpha(v-\theta)}}{(1 + e^{-\alpha(v-\theta)})^2}$$

where α is the sharpness parameter and θ is the threshold.

This allows for gradient-based updates to the weight matrices between layers:

- Gradients are computed at each time step by comparing output spikes to the true label.
- Weight updates are accumulated across time and averaged.
- Gradients are clipped to ensure stability.

Training Dynamics

During training, the network processes each input sample across T time steps. At each step:

1. Input spike trains are fed into the hidden layer.
2. Membrane potentials are updated using the LIF dynamics with leakage and thresholding.
3. Hidden layer spikes feed into the output layer.
4. If a neuron's potential crosses the threshold, it emits a spike and resets.

The error is computed using the difference between actual and target spike counts. Gradients are calculated using the surrogate function and used to update the weights with gradient descent.

Evaluation

After training, the model is evaluated on a separate test set:

- Each test image is encoded into a spike train using the same Poisson encoding strategy.
- The network processes these spikes without updating weights.
- The neuron in the output layer that accumulates the highest spike count over time is considered the predicted digit.

The final test accuracy is calculated by comparing predictions to true labels.

Visualization of Spike-Based Output

To better understand the classification behavior, a bar plot is generated for the first 10 test samples. For each image:

- A bar chart shows the total spike count across the 10 output neurons.
- The class with the highest spike count is considered the prediction.
- The plot compares the true label and predicted class.

This visualization confirms that the network learns to associate different spike patterns with different digits and that spike frequency plays a key role in decision-making.

Summary

This step demonstrates how an SNN can be trained end-to-end using surrogate gradients. Despite the binary nature of spikes, the network learns to classify digits by adjusting weights to control when and how often neurons fire. Compared to traditional ANNs, SNNs offer insights into temporal processing, event-driven computation, and biologically inspired learning mechanisms.

Challenges Faced and Resolutions

Throughout the development and training of the Spiking Neural Network (SNN) model for digit classification, several significant challenges were encountered. These challenges spanned both implementation-level difficulties and theoretical issues unique to training spiking systems. Below is a detailed discussion of these obstacles and the strategies employed to address them.

1. Hyperparameter Tuning

Training SNNs is highly sensitive to hyperparameters such as:

- Number of time steps (T)
- Membrane threshold (θ)
- Leakage constant (β)
- Learning rate
- Network architecture (hidden layer size)

Unlike traditional ANNs, small changes in these values can lead to drastically different spiking behavior. For instance:

- If θ is too high, neurons fail to spike.

- If β is too low, membrane potentials decay too quickly to integrate meaningful information.
- If T is too small, the network cannot capture enough temporal dynamics for accurate classification.

Resolution: Extensive experimentation was carried out by sweeping over different values. Validation accuracy was tracked epoch-wise. Optimal results were obtained at:

- $T = 50$
- $\theta = 0.5$
- $\beta = 0.98$
- Learning rate $= 2 \times 10^{-3}$
- Hidden layer size $= 250$ neurons

2. Long Training Time

Even with a modest network size, training took a significant amount of time due to:

- Sequential simulation over T time steps for every input
- Spike-based processing being inherently sparse yet expensive

Resolution: The code was accelerated using GPU-backed numpy operations (e.g., via CuPy in extended versions) and optimized memory allocation. Training was restricted to a manageable subset (e.g., 10,000 samples) for experimentation. Batch processing and parallel encoding were also considered for further optimization.

3. Surrogate Gradient Overflow

A major technical difficulty was the computation of surrogate derivatives. Since these involve exponential functions, numerical overflow errors occurred during training:

```
RuntimeWarning: overflow encountered in exp
```

Resolution: To stabilize the gradient, the argument passed to the exponential function was clipped within a bounded range:

$$x = \text{clip}(v - \theta, -10, 10)$$

This ensured gradients remained finite and the model learned stably.

4. Gradient Clipping

The unbounded growth of weight updates over time led to instability and divergence in the early stages of training. This was especially problematic in the presence of sparse spiking, which caused large gradient values when spikes were infrequent but mismatched.

Resolution: Gradient clipping was applied after each weight update:

```
np.clip(dW, -1, 1, out=dW)
```

This bounded the updates to a controlled range and improved convergence.

5. Debugging and Visualization of Spiking Behavior

Verifying that neurons were spiking correctly and that their dynamics matched theoretical expectations was non-trivial.

Resolution: Visual tools were developed to:

- Render 2D spike frames at each time step
- Plot 3D raster plots of spike activity across time
- Visualize output spike counts for digit classification

These tools helped confirm that the network was integrating inputs and firing appropriately, aiding in both debugging and interpretation.

Summary

Despite the inherent complexity of SNNs, the challenges of unstable gradients, long training times, and sensitive parameters were overcome through careful engineering, visualization, and mathematical regularization techniques. These insights can serve as a roadmap for future work involving large-scale SNNs or more biologically plausible architectures.

Reflections: Comparison Between SNNs and ANNs

Spiking Neural Networks (SNNs) differ significantly from traditional Artificial Neural Networks (ANNs) in both their computational mechanisms and philosophical inspiration. The following reflections outline the key distinctions observed during the implementation and training of SNNs on the MNIST dataset.

1. Information Representation and Processing

- **ANNs:** Neurons in ANNs process continuous-valued activations. Each neuron's output is typically a real-valued scalar passed through an activation function such as ReLU or sigmoid.
- **SNNs:** Neurons in SNNs communicate through discrete binary events (spikes). The timing of these spikes carries essential information, making the encoding inherently temporal. In this work, *Poisson encoding* was used to convert static images into spike trains over multiple time steps.

2. Temporal Dynamics and Memory

- **ANNs:** Classical feedforward ANNs are memoryless; each input is processed independently unless explicitly designed with recurrent connections or temporal memory.
- **SNNs:** Neurons maintain internal states in the form of membrane potentials. These potentials integrate input spikes over time and decay via a leakage term. This creates an inherent temporal memory without needing external mechanisms.

3. Learning Mechanisms

- **ANNs:** Training is typically performed using standard backpropagation with gradient descent, leveraging smooth and differentiable activation functions.
- **SNNs:** Due to the non-differentiable nature of spike events, standard backpropagation cannot be directly applied. In this project, *surrogate gradients* were used to approximate the gradient during training, enabling supervised learning.

4. Sparsity and Energy Efficiency

- **ANNs:** Every neuron computes a weighted sum and applies an activation at each forward pass, leading to dense computation even for sparse inputs.
- **SNNs:** Neurons only fire when membrane potential crosses a threshold, resulting in sparse computation. This sparsity can potentially lead to much lower energy consumption, especially in neuromorphic hardware.

5. Computational Complexity and Training Speed

- **ANNs:** Modern ANN architectures are highly optimized for GPUs and support batch processing, enabling fast training.
- **SNNs:** Due to the temporal dimension and lack of mature deep learning frameworks for spiking computation, training is slower and often sequential in software simulations.

6. Biological Plausibility

- **ANNs:** While inspired by biology, the mechanisms in ANNs (e.g., backpropagation) do not align closely with how real neurons work.
- **SNNs:** SNNs are considered the third generation of neural networks and are much closer to the operation of biological brains, where spike timing and synaptic dynamics are crucial.

Conclusion

While ANNs dominate practical machine learning due to their high performance and established toolchains, SNNs offer a compelling alternative for scenarios where energy efficiency, temporal processing, and biological realism are prioritized. This project demonstrated that, with surrogate gradient learning and proper encoding, SNNs can achieve competitive accuracy on tasks like MNIST classification, though challenges in training complexity and speed remain areas of active research.