Applied Neuromorphic Intermediate Representation

Victor Mazanov
Innopolis University
Email: v.mazanov@innopolis.university

Artyom Grishin
Innopolis University
Email: ar.grishin@innopolis.university

Remal Gareev
Innopolis University
Email: r.gareev@innopolis.university

Vladislav Merkulov
Innopolis University
Email: v.merkulov@innopolis.university

***Abstract*—This chapter presents an application of the Neuromorphic Intermediate Representation (NIR) framework. We trained a spiking convolutional neural network (SCNN) on the N-MNIST dataset and subsequently used NIR to convert the trained model into the Norse framework. The model performance was evaluated in terms of accuracy and loss, with results indicating either zero or negligible loss values, suggesting effective conversion and keeping the correctness of model.**

***Index Terms*—Spiking Neural Networks, Neuromorphic Computing, Neuromorphic Intermediate Representation, Norse Framework, Nature Inspiring Computing**

I. Introduction

The human brain continues to be the most efficient and powerful computational system we know, surpassing modern machine learning algorithms in performance while consuming only a small amount of energy. One of the key reasons for this efficiency is the way neurons communicate by the discrete electrical events known as spikes. A spike, or action potential, is an all-or-nothing event characterized by a brief and rapid change in voltage across the neuronal membrane. [1] This occurs when the membrane potential crosses a specific threshold.

Biologically inspired models such as the leaky integrate-and-fire (LIF) and the simpler leaky integrator (LI) capture these essential neuronal dynamics. The LIF model, in particular, characterizes neuron behavior by integrating incoming currents over time while accounting for the passive leakage of electrical charge, and then triggering a spike when a certain voltage threshold is exceeded. In contrast, the LI model offers a more streamlined approach to account for the integration process without the detailed dynamic resetting, providing a useful abstraction in certain computational contexts.

Spiking convolutional network (SCNN) uses this idea, in which neurons in the SCNN do not transfer information during every propagation cycle (unlike standard multilayer perceptron in Artificial Neural Networks(ANN)), but instead convey information only when a membrane potential—an inherent characteristic of the neuron associated with its membrane's electrical charge, hits a certain level, known as the threshold [2].

In parallel with software development, neuromorphic hardware platforms such as Intel's Loihi and IBM's TrueNorth aim to execute SNNs efficiently at the hardware level.

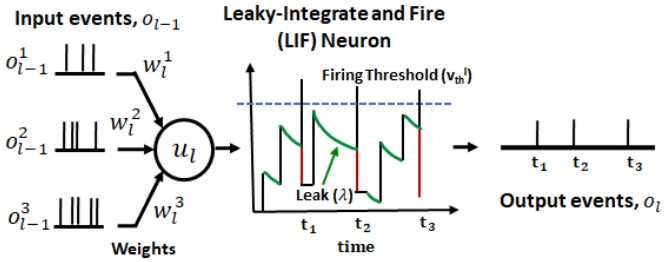

Fig. 1. Leaky Integrate and Fire.

In advancing the state-of-the-art in spiking neural network (SNN) research, training these temporally dynamic models poses unique challenges. One widely adopted technique is backpropagation through time (BPTT), which adapts the conventional backpropagation algorithm to handle time-dependent sequences by unrolling the network over temporal dimensions and computing gradients accordingly. This method, while computationally intensive, is essential for optimizing networks that operate over sequences of spikes.

Modern software frameworks have been developed to support the design, simulation, and training of spiking neural networks. Libraries such as Norse and snntorch integrate seamlessly with PyTorch, providing intuitive tools to model SNNs based on LIF and LI dynamics and to implement training routines that leverage BPTT. These frameworks facilitate not only the simulation of biologically plausible spiking behavior but also the practical aspects of machine learning workflows, enabling researchers to experiment with both the architectural nuances and the training algorithms specific to spiking networks. [3]

The neuromorphic intermediate representation (NIR) is a tool that supports converting models from different libraries to analogous models. [4] NIR plays an important role in ensuring cross-compatibility in neuromorphic research. It abstracts the specifics of various frameworks, allowing for easier integration and deployment of models across platforms. This reduces the development overhead and fosters collaboration among researchers using different tools or hardware backends.

NIR graph is a graph-based intermediate representation that takes the architecture and parameters of an SNN. It is used to

reconstruct some network across different frameworks. Graph nodes contain network layers when edges represent inputs. Moreover, it contains NIR primitives as the elements of the network.

In this chapter, we apply this framework's primitives to convert a model from the SNN Torch to the NIR graph and then convert it to the Norse framework. This conversion process underscores the versatility and interoperability of current neuromorphic tools, facilitating the pathway from design and visualization to simulation and analysis on various platforms.

A. Information about datasets

MNIST [5]- is a dataset of 70,000 grayscale images of handwritten digits (0 through 9). Train part of 60,000 and test set of 10,000 images of size 28 x 28. Other two datasets are from Tonic- Python library that contains a list of neuromorphic datasets and provided functionality to preprocess and augment them: N-MNIST [6] dataset- is a version of the default MNIST dataset that was made for testing Spiking Neural Networks (SNNs) and other event-based models. POKERDVS [7]- is a neuromorphic dataset of event-based recordings of playing cards (Ace, King, Queen, etc.) that was captured using a Dynamic Vision Sensor (DVS) camera.

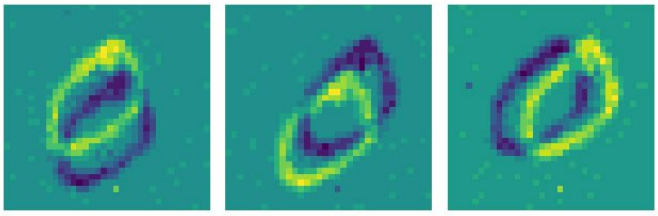

Fig. 2. N-MNIST dataset

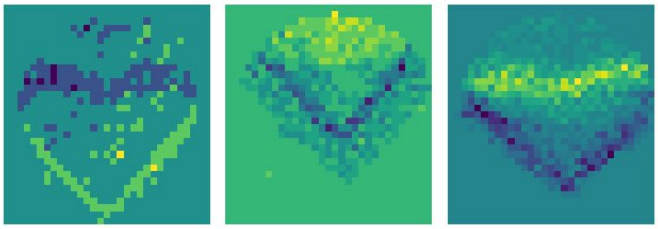

Fig. 3. POKERDVS

II. Related work

We mostly relied on the NIR article where authors created a framework that provides an opportunity of writing code for neuromorphic systems on different hardware platforms and transform it to multiple formats that can be ran on this platforms. Researchers tested their work with SCNN on N-MNIST dataset by transforming code to different frameworks and got over 97

In our work we firstly wrote code for SCNN model on one platform, trained it on this platform on MNIST, N-MNIST and POKERDVS datasets, transformed it with NIR to other neuromorphic platform and tested model on it. We also compared the accuracy on different platforms.

Also we lean on the work of snnTorch [8] where researchers where training SCNN on MNIST and testing it.

Our work has tests of SCNN writen in snnTorch and trained not only on N-MNIST, but MNIST and POKERDVS datasets with saving membrane potential and spike of each neuron in test data for creating informative plots with statistics.

III. Methodology

This study investigates a complete training-transfer workflow for SNNs, focusing on interoperability between different neuromorphic frameworks. The proposed methodology involves training an SNN using snnTorch, exporting the model to NIR, and importing it into Norse for execution.

A. Dataset Preparation

Neuromorphic datasets were preprocessed into frame-based tensors to enable compatibility with time-unfolded SNN architectures. Dataset-specific parameters such as time window duration and padding schemes were handled during the evaluation phase.

B. Network Architecture

The proposed model consists of a two-layer feedforward SNN:

- **fc1:** A fully connected layer that maps the flattened DVS input to hidden units.
- **lif1:** A synaptic neuron layer modeled as a LI, with fixed decay parameters (α, β).
- **fc2:** A second fully connected layer projecting the hidden state to all output classes.
- **lif2:** A synaptic neuron layer modeled as LIF with learnable decay parameters (α, β), as well as a trainable threshold vector.

The network unfolds over time using a time-stepped loop. At each timestep, the input is propagated through the network, and both membrane potentials and spiking outputs are recorded.

C. Training Procedure

The SNN is trained using backpropagation through time (BPTT), where the loss is accumulated across temporal steps and applied to the membrane potential dynamics. Synaptic parameters in the second layer are optimized along with network weights to capture temporal dependencies in the spiking data.

D. Export to NIR

Following training, the model is exported to NIR using the `export_to_nir` function provided by snnTorch. NIR serves as a framework-agnostic intermediate format that captures layer definitions, weights, neuron parameters, and time-based unrolling logic.

E. Import into Norse

The exported NIR model is loaded into the Norse framework using `from_nir()`. A custom inference function applies inputs to the imported model step-by-step, maintaining the hidden state across timesteps. This validates that Norse can replicate the trained SNN behavior initially developed in snnTorch.

F. Summary

The methodology enables the following:

- Training of a temporally dynamic, partially learnable SNN in snnTorch.
- Export of the trained model to a portable NIR format.
- Reuse and execution of the trained model in Norse without retraining.

The proposed workflow was applied across multiple neuromorphic datasets to validate both functional correctness and cross-framework portability.

IV. GitHub Link

GitHub Project.

V. Experiments and Evaluation

A. Datasets

Two neuromorphic datasets were used for evaluation: N-MNIST, POKERDVS, and one static MNIST. N-MNIST dataset was converted into frame-based tensors using a fixed time window of 60 ms via the `ToFrame` transformation from the Tonic library. To ensure consistent sequence lengths within each batch, input tensors were padded using `PadTensors`, and data loading was accelerated via disk caching. The same was done for POKERDVS but with only 1 ms due to the small size of the dataset (its timesteps). And to make MNIST compatible with spiking neural networks, synthetic temporal structure must be introduced — typically by replicating frames over multiple timesteps

B. Training Configuration

The model was trained using the Adam optimizer with a learning rate of 10^{-3} and a batch size of 64 for NMNIST and MNIST and batch size of 8 for POKERDVS. The cross-entropy loss function was applied to the membrane potential at each timestep and accumulated across the full temporal sequence. Training was performed for one epoch for demonstration, but already have good results

All experiments were conducted using Google Colab with T4 GPU.

C. Evaluation Metrics

Accuracy was computed based on the total spike count over all timesteps, selecting the output neuron with the highest spike count as predicted class. This evaluation strategy was consistenly applied to both the original snnTorch model and the transferred Norse model.

D. Plots

We changes process of testing SCNN model to record information about membrane potential, spikes activation, labels taking and output spikes on Synaptic Neurone, LIFNode and Lapicque's Neuron Model Input Spikes. By this plots we can watch the work of our model deeper (not only by digits and text) on physical procesess of neuromorphic computing idea.

E. Transfer Validation

To assess framework compatibility, the trained snnTorch model was exported to NIR and re-imported into Norse without modification. Inference in Norse was conducted using a custom step-wise loop to preserve hidden states. The results showed that the Norse model replicated the expected spiking behavior. The most important thing in transferring to Norse is to set the parameter of dt to 0.0001, otherwise the model will give the random accuracy, dt is the simulation step width.

F. Accuracy

The accuracy hasn't changed after trasnfering from snnTorch to Norse.

TABLE I
CLASSIFICATION ACCURACY ON DIFFERENT DATASETS AFTER TRANSFER TO NORSE.

Dataset	Accuracy (%)
N-MNIST	90.0
MNIST	95.0
Poker-DVS	90.0

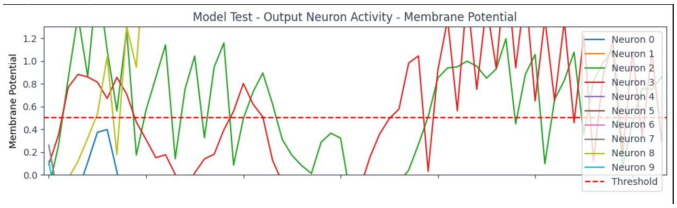

Fig. 4. Graph on MNIST

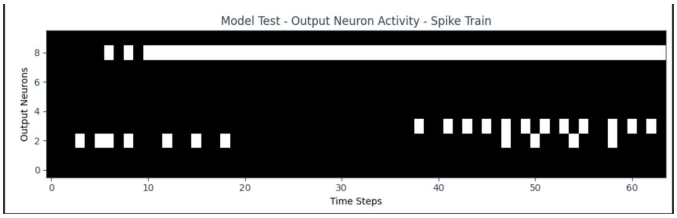

Fig. 5. Graph on NMNIST

VI. Analysis and Observations

During the evaluation phase of our code, we identified that the NIR framework currently does not support the conversion MaxPool layers from Norse to SCNN format. We show this issue to the NIR maintainers, who acknowledged the observation and expressed their appreciation for the feedback.

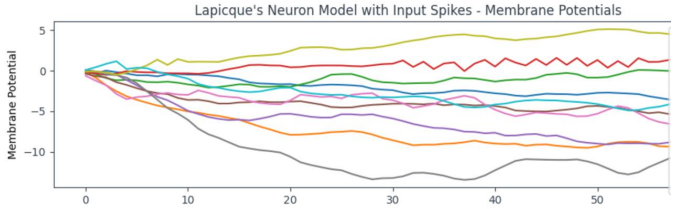

Fig. 6. Graph on MNIST

Also, graphics that we got by saving membrane potential and spikes of all neurons during test process of SCNN on MNIST and N-MNIST dataset shows that our model perfoms generally good, spikes activates and ”firing”.

Furthermore, our analysis indicates that SCNN implementations generally exhibit slower performance compared to ANN. This is primarily due to the recurrent nature of SCNNs, in which the output from each timestep must be reintroduced as input for subsequent timesteps. In addition, SNNs typically involve complex equations that incur additional computational overhead, further contributing to reduced processing speeds.

VII. Conclusion

To elucidate the dynamics of spiking neural networks, we utilized a neuromorphic dataset and analyzed how the data is represented and processed within our network framework. Initially, we selected the Norse library, a deep learning library for SNNs to construct a network aimed at solving a basic digit recognition problem. For this purpose, we used the NMNIST dataset, an enhanced version of MNIST that incorporates spike events and temporal sequences.

After training the network in Norse, we employed NIR to convert the network into a format compatible with another library, SnnTorch. During this conversion process, we encountered an issue: Norse currently does not support the conversion of MaxPooling layers. We reported this discrepancy, which led to the opening of an issue on the Norse GitHub repository here.

Subsequently, we developed a network using SnnTorch without MaxPooling layers to train on the NMNIST dataset. This model achieved an accuracy of 90%. The next step involved converting this SnnTorch model into a graph format via NIR and importing it into Norse for a comparative analysis. We observed a discrepancy in time-step recognition between the libraries due to differing discretization methods: SnnTorch and Norse each employ their own approach. After resolving these issues and improving network training, we achieved an accuracy of 95%.

References

[1] Mark Humphries. What it’s like to be a spike: What we’re learning in the golden age of neuroscience. https://press.princeton.edu/ideas/whats-it-like-to-be-a-spike, 2023. Accessed: 2023-10-01.

[2] Wikipedia contributors. Spiking neural network. https://en.wikipedia.org/wiki/Spiking_neural_network, 2023. Accessed: 2023-10-01.

[3] Adarsh Kosta and Kaushik Roy. Adaptive-spikenet: Event-based optical flow estimation using spiking neural networks with learnable neuronal dynamics. *arXiv*, 2022.

[4] Jens E. Pedersen, Steven Abreu, Matthias Jobst, Gregor Lenz, Vittorio Fra, Felix Christian Bauer, Dylan Richard Muir, Peng Zhou, Bernhard Vogginger, Kade Heckel, Gianvito Urgese, Sadasivan Shankar, Terrence C. Stewart, Sadique Sheik, and Jason K. Eshraghian. Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing. https://doi.org/10.1038/s41467-024-52259-9, 2024.

[5] Yann LeCun and Corinna Cortes. Mnist handwritten digit database. https://www.kaggle.com/datasets/hojjatk/mnist-dataset, 2010. Accessed: 2023-10-01.

[6] Garrick Orchard, Ajinkya Jayawant, Gregory Cohen, and Nitish Thakor. N-mnist: Neuromorphic-mnist dataset. https://www.garrickorchard.com/datasets/n-mnist, 2015. Accessed: 2023-10-01.

[7] A. Author and B. Researcher. Pokervds: Poker vision dataset suite. https://www.pokervds.com/dataset, 2023. Accessed: 2023-10-01.

[8] Jason K. Eshraghian, Max Ward, Emre Neftci, Xinxin Wang, Gregor Lenz, Girish Dwivedi, André van Schaik, Chee Loo, Yulia Sandamirskaya, Terrence Stewart, et al. snntorch: A python package for simulating spiking neural networks. https://snntorch.readthedocs.io/, 2023. Version 0.6.3.