Leveraging PyTorch on BrainScaleS-2: Training a Real-World Application

Elias Arnold[1,*] Philipp Spilger[1] Eric Müller[3,1] Georg Böcherer[2]
Maxim Kuschnerov[2] Johannes Schemmel[1]

[1]Kirchhoff-Institute for Physics, Ruprecht-Karls-Universität Heidelberg, Heidelberg, Germany
[2]Munich Research Center, Huawei Technologies Duesseldorf GmbH, Munich, Germany
[3]European Institute for Neuromorphic Computing, Heidelberg University, Heidelberg, Germany

Introduction

The usability of novel neuromorphic hardware is dictated by software allowing non-expert users to describe experiments effortlessly [3]. Especially when approaching gradient-based optimization tasks with spiking neural networks (SNNs) on such hardware, support for model definition and automatic differentiation is indispensable.

- We discuss the abstraction of training on the accelerated mixed-signal neuromorphic hardware system BrainScaleS-2 (BSS-2) in our high-level PyTorch-based framework `hxtorch.snn` and motivate our design choices
- We demonstrate `hxtorch.snn` on a real-world application: The demapping of a nonlinearly impaired high-speed intensity modulation / direct detection (IM/DD) optical data link with SNNs on BSS-2 [1]

Methods

BrainScaleS-2 (BSS-2)

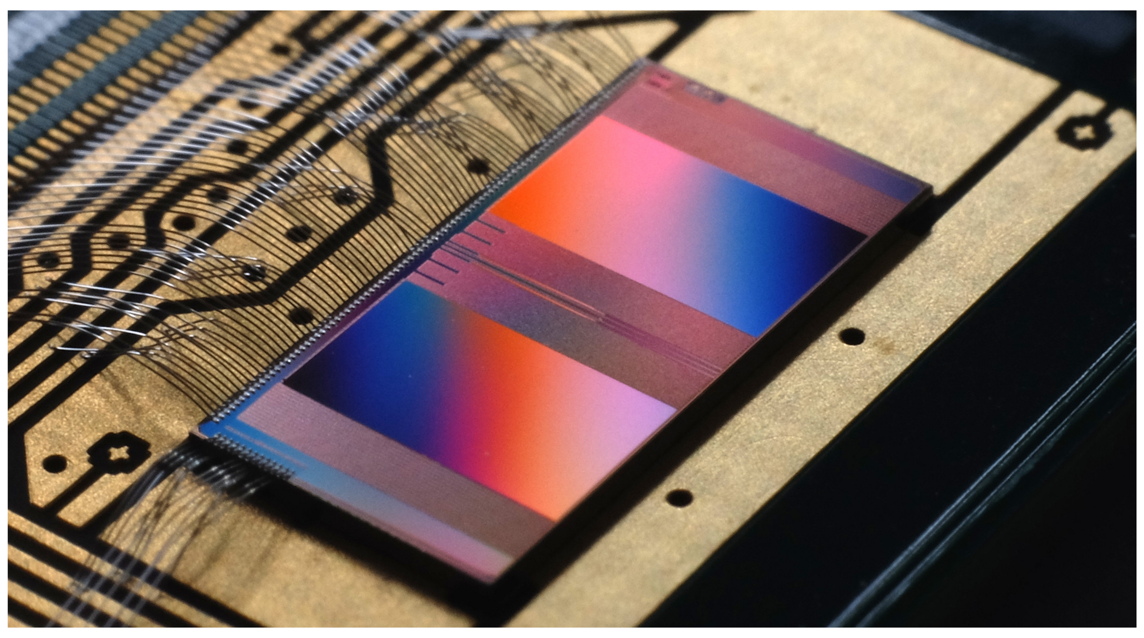


Figure 1. The BrainScaleS-2 analog neuromorphic system.

The mixed-signal neuromorphic BSS-2 system [6]:

- 512 spiking AdEx [2] neurons and 131k 6-bit plastic synapses in parallel analog circuits, emulated in continuous time
- Neural dynamics individually parameterized
- Support for arbitrary topologies
- Access to spike events and membrane potentials, sampled column-wise in parallel by analog-to-digital converters

Training SNNs with PyTorch

SNNs are studied for machine learning tasks using state-of-the-art software libraries typically deployed for artificial neural networks (ANNs), like PyTorch [5].

- The network model is described by a composition of PyTorch modules (layers)
- Easy computation of network gradient by the backpropagation through time algorithm
- Eager construction of a differentiable computational graph by executing all network operations successively and assigning a backward function to it → Gradient → Weight update

NOTE The non-differentiability of spiking neurons is often bypassed with surrogate gradients [4]

In-the-loop Learning on BSS-2

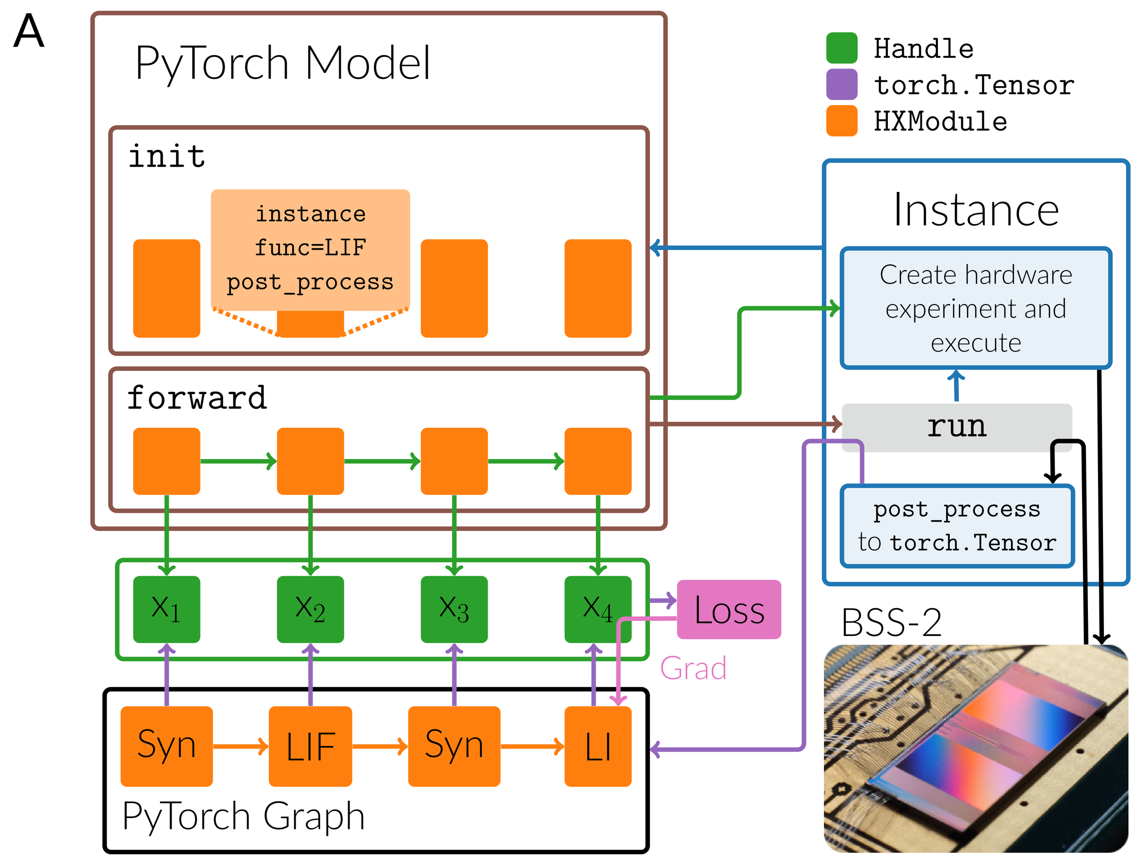
SNNs on BSS-2 can be trained in-the-loop within PyTorch's ecosystem:

1. Emulate the forward pass on BSS-2
2. Use the hardware observables to compute weight updates on the host computer

NOTE The SNN is executed on BSS-2 before the PyTorch graph is built to ensure that hardware data is present for the backward pass

`hxtorch.snn` Framework

A



B

```
ins = hxtorch.snn.Instance()

syn1 = hxtorch.snn.Synapse(ins, ...)
lif1 = hxtorch.snn.LIF(ins, ...)
syn2 = hxtorch.snn.Synapse(ins, ...)
li2 = hxtorch.snn.LI(ins, ...)

x1 = syn1(Handle(input))
x2 = lif1(x1)
x3 = syn2(x2)
x4 = li2(x3)

hxtorch.snn.run(ins, ...)

loss = f(x4)
loss.backward()
```

Figure 2. **A)** Schmeantic of `hxtorch.snn` framework. **B)** Code example of an feed-forward SNN. Both taken from [7].

- SNNs are defined as in PyTorch
- Modules are derived from a base-class `HXModule` representing entities on BSS-2, e.g., neuron populations `LIF` and `LI`, and projections `Syn`
- Modules hold a PyTorch-differentiable function `func(...)`, defining the backward pass
- Modules get `Handle`-typed inputs and return a `Handle`-typed output → References to data available after hardware execution
- Modules share an `Instance` (experiment on BSS-2) object in which they register themselves

Calling `run` → `Instance` extracts and executes the hardware experiment on BSS-2 → Hardware observables are `post_process`ed to `torch.Tensors` → The PyTorch graph is constructed by calling the modules' `func`tions, utilizing and returning the data tensors → Tensors are assigned to the modules' handles → PyTorch `backward` pass estimates the gradient of the SNN on BSS-2

NOTE For model development, `hxtorch.snn` also supports a simulation mode

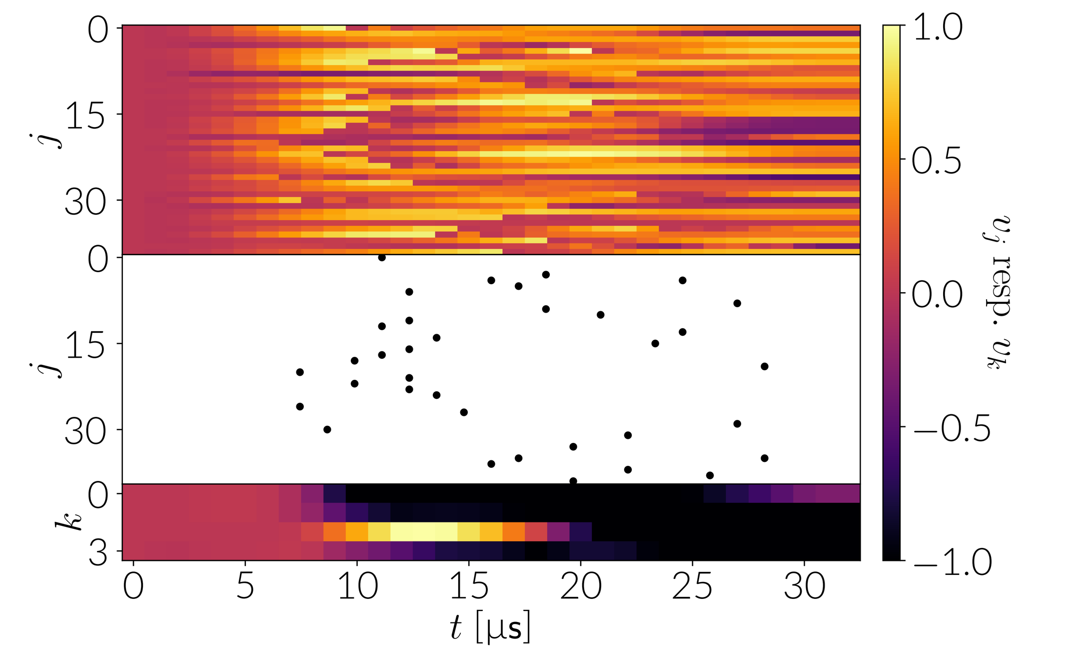


Figure 5. BSS-2 observables of the LIF (Upper: membrane, middle: spikes) and LIF (lower: membrane) layer while demapping two bits.

Example: Demapping an IM/DD Optical Link on BSS-2

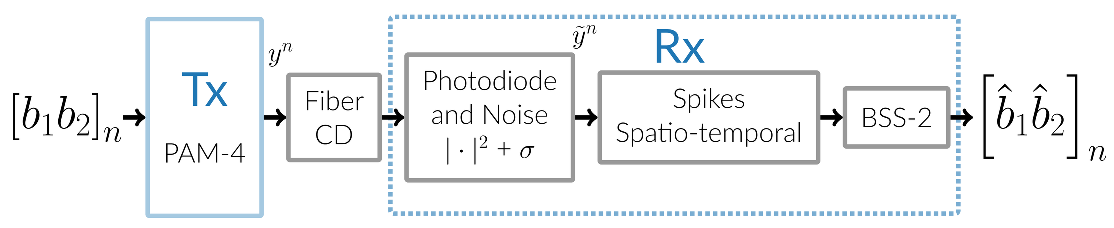


Figure 3. Demapping a simualted IM/DD link on BSS-2.

For demonstration, we train an SNN on BSS-2 to demap a simulated IM/DD link [1], see fig. 3. A bit sequence $[b_1 b_2]_n$ is modulated to a signal y^n, optically transmitted through a fiber, and measured at the receiver by a photodiode (PD). The resulting sequence $\tilde{y}^n$ is impaired by linear chromatic dispersion, the nonlinear PD, and noise (σ). The SNN equalizes and demaps $\tilde{y}^n$, translated into spikes, to the received bits $[\hat{b}_1 \hat{b}_2]_n$. The SNN, consisting of a hidden leaky-integrate and fire (LIF) and a leaky-integrator (LI) output layer, is trained to minimize the bit error rate (BER).

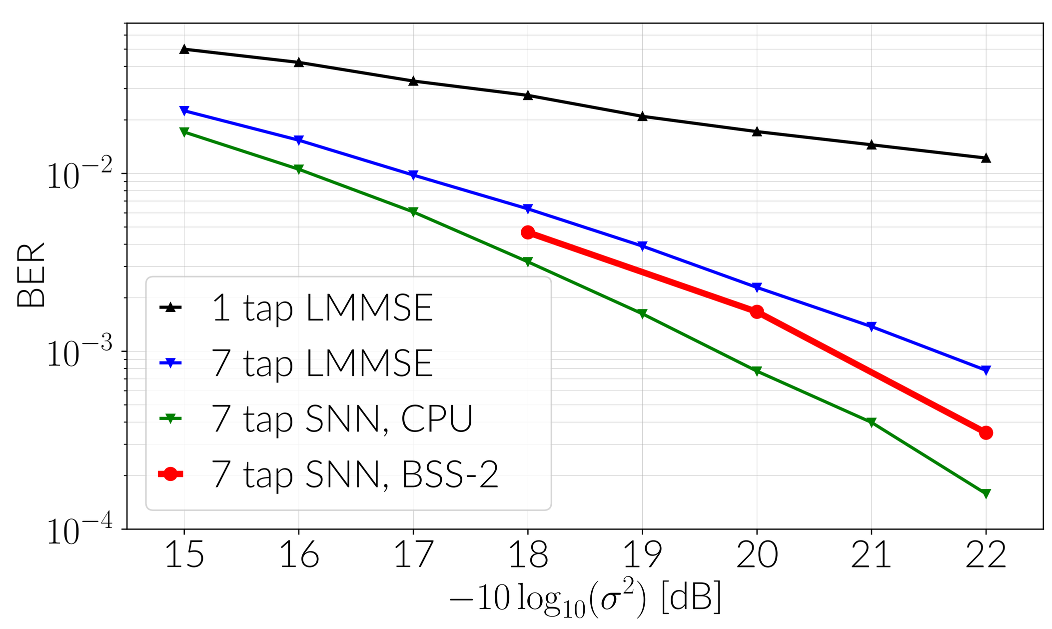


Figure 4. BER over noise in the IM/DD link.

In fig. 4, the BER is depicted over noise σ in the link. The simulated SNN and the SNN emulated on BSS-2 achieve lower BERs than the linear LMMSE equalizer. Figure 5 shows the LIF neurons' membrane voltages and their spikes together with the LI output traces are shown → The output neuron producing the highest voltage dictates the demapped bits.

Conclusion

- `hxtorch.snn` allows descriptions of high-level SNN models in PyTorch and their emulation on BSS-2
- By utilizing PyTorch data types, we can leverage its auto-differentiation mechanism and thus make learning on BSS-2 effortless
- API supports the definition of arbitrary networks (incl. feedback connections), the different neuron types on BSS-2 and different backward functions
- Integrated into the EBRAINS platform

References

[1] E. Arnold et al. "Spiking Neural Network Equalization on Neuromorphic Hardware for IM/DD Optical Communication". In: *European Conference on Optical Communication (ECOC) 2022*. Optica Publishing Group, June 2022, Th1C.5.

[2] Romain Brette and Wulfram Gerstner. "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity". In: *Journal of Neurophysiology* 94.5 (2005). PMID: 16014787, pp. 3637–3642. DOI: `10.1152/jn.00686.2005`.

[3] E. Müller et al. "A Scalable Approach to Modeling on Accelerated Neuromorphic Hardware". In: *Frontiers in Neuroscience* 16 (2022). DOI: `10.3389/fnins.2022.884128`.

[4] E. O. Neftci et al. "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63. DOI: `10.1109/MSP.2019.2931595`.

[5] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.

[6] C. Pehle et al. "The BrainScaleS-2 Accelerated Neuromorphic System With Hybrid Plasticity". In: *Frontiers in Neuroscience* 16 (2022). DOI: `10.3389/fnins.2022.795876`.

[7] P. Spilger et al. *hxtorch.snn: Machine-learning-inspired Spiking Neural Network Modeling on BrainScaleS-2*. 2022. DOI: `10.48550/arXiv.2212.12210`.

This work received funding from: BMBF (16ES1127), EU (H2020/2014-2020: 720270, 785907, 945539), Lautenschläger-Forschungspreis 2018, DFG (EXC 2181/1-390900948).

*elias.arnold@kip.uni-heidelberg.de