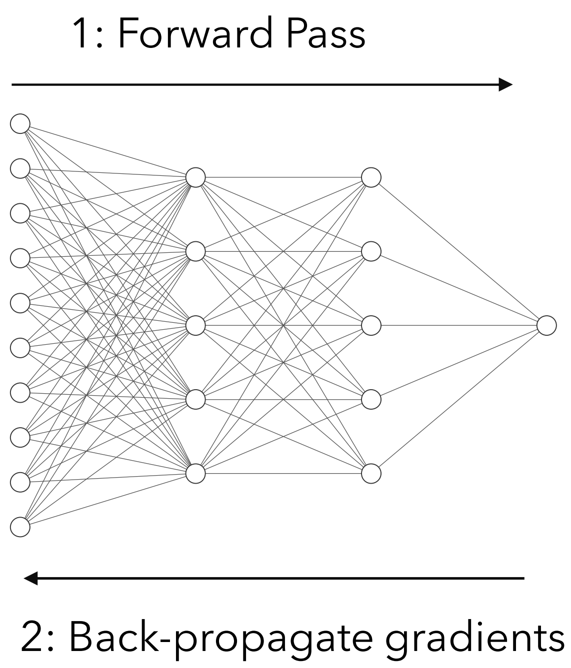
Probabilistic Spiking Neural Networks

Cameron Barker

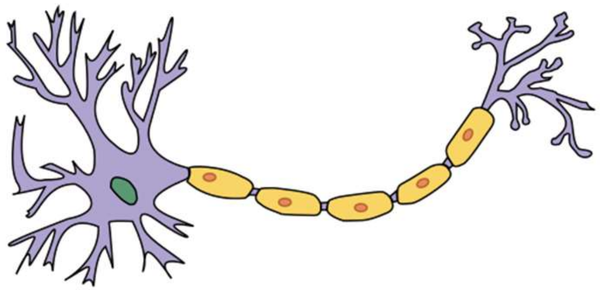
Artificial Neural Networks & Back-Propagation

- Connectionist – Composed linear transforms and non-linear activation functions
- Gradient based learning
- GPU Acceleration



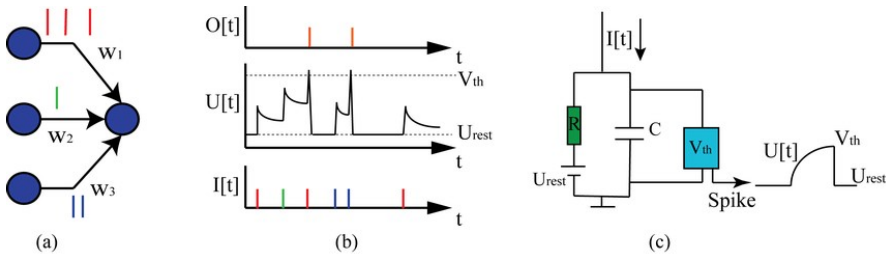
Biological Neural Networks

- ANN – Abstract representation of biological neuron
- ANNs use Float32/64 – BNN emit asynchronous impulses
- ANN usually layers – BNN graphs



Spiking Neural Networks

- Biologically plausible neuron models – LIF
- Sparse – Energy efficient
- Neuromorphic hardware
- Activation function → Usually non-differentiable

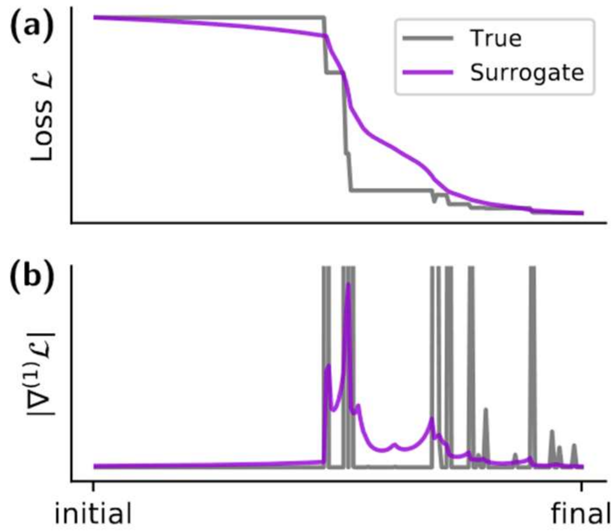


Chowdhury, Sayeed & Lee, Chankyu & Roy, Kaushik. (2020)
Towards Understanding the Effect of Leak in Spiking Neural Networks.

$$Spike(U(t)) = Heaviside(U(t) - V_{th})$$

Surrogate Gradient Learning

- Model SNN as RNN
- Implement BPTT using surrogate gradients
- PyTorch implementable (Norse)



E. O. Neftci, H. Mostafa and F. Zenke, "Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks," in *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51-63, Nov. 2019, doi: 10.1109/MSP.2019.2931595.

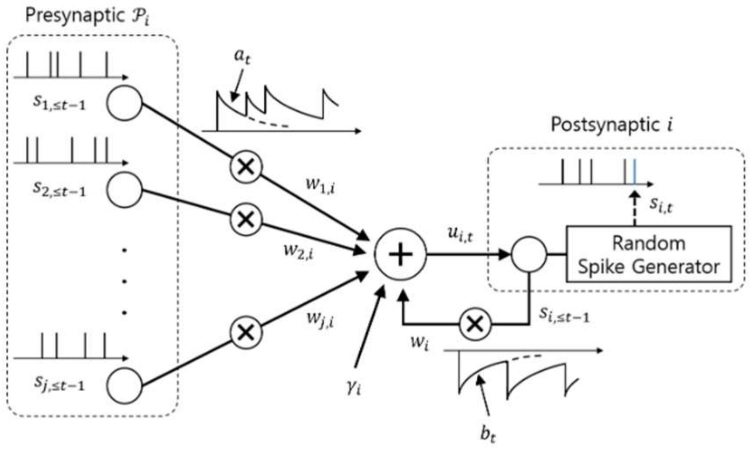
Probabilistic Spiking Neural Networks

- Convolve spikes / Running average

  Different kernels → Neuron Dynamics

- Combine presynaptic inputs

- $P(s_{i,t} = 1) = \sigma(u_{i,t})$



H. Jang, O. Simeone, B. Gardner and A. Gruning, "An Introduction to Probabilistic Spiking Neural Networks: Probabilistic Models, Learning Rules, and Applications," in *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 64-77, Nov. 2019, doi: 10.1109/MSP.2019.2935234.
H. Jang and O. Simeone, "Multisample Online Learning for Probabilistic Spiking Neural Networks," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 5, pp. 2034-2044, May 2022, doi: 10.1109/TNNLS.2022.3144296.

Learning Rules

- Variational learning + ML with SDG

$$e_i \leftarrow \begin{cases} \nabla \omega_{j,i} = \sum_{t=0}^{T}((s_{i,t} - \sigma(u_{i,t})) \cdot \overrightarrow{s}_{i,t-1}) \\ \nabla \omega_i = \sum_{t=0}^{T}((s_{i,t} - \sigma(u_{i,t})) \cdot \overleftarrow{s}_{i,t-1}) \\ \nabla \gamma_i = \sum_{t=0}^{T}(s_{i,t} - \sigma(u_{i,t})) \end{cases}$$

$$\ell = \sum_{t=0}^{T} \sum_{i \in \mathcal{X}} \log p(x_{i,t} | u_{i,t}),$$

$$\Theta_i \leftarrow \Theta_i + \eta \cdot \begin{cases} e_i, & \text{if } i \in \text{ Observed Neurons} \\ \ell e_i, & \text{if } i \in \text{ Hidden Neurons} \end{cases}$$

where ℓ is global learning signal

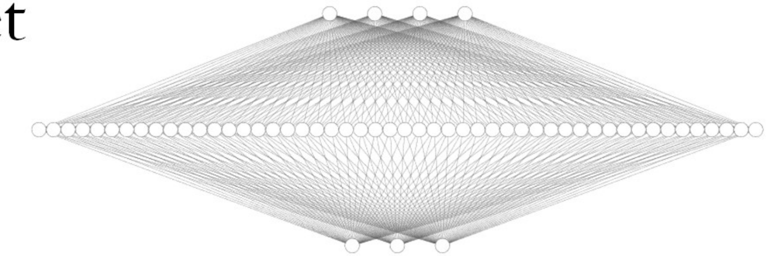
where $u_{i,t}$ is sampled from the feedforward distribution (variational posterior)

- Adaptive learning through momentum
- Multiple compartments $\ell^k \leftarrow \sigma_{SM}(\ell^k)$

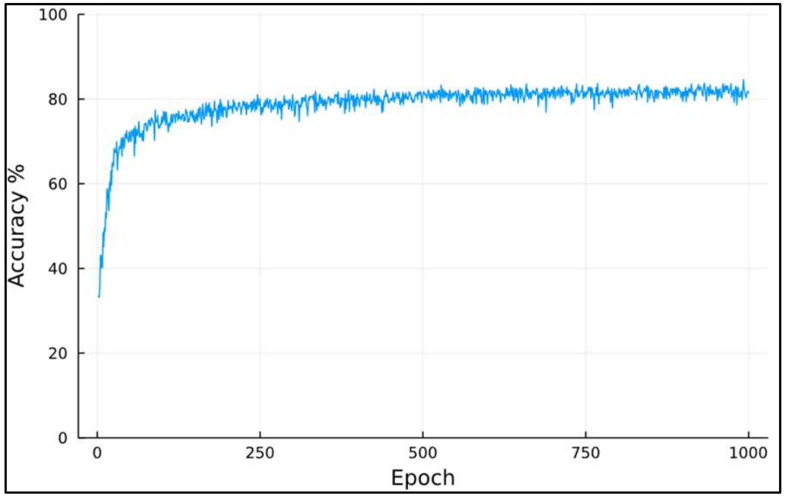
Implementation

- Fully Documented Module for Julia programming language
- Fully vectorized with no-copy array memory access
- Poisson Rate Encoded Dataset
- Rate Decoded
- CUDA – Memory bottleneck

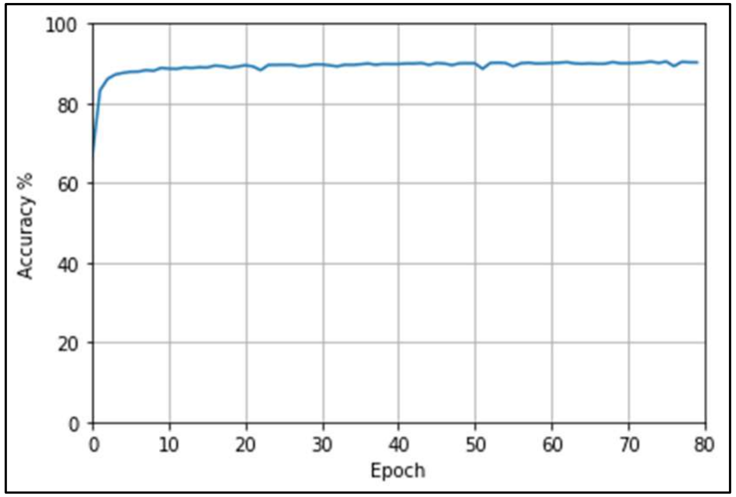
Experimental Results: Iris Dataset



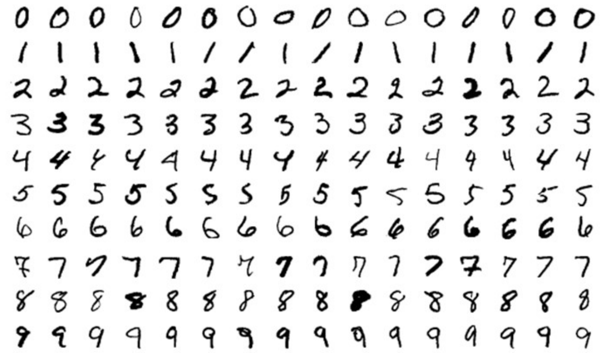
PSNN – 82% in 23 minutes



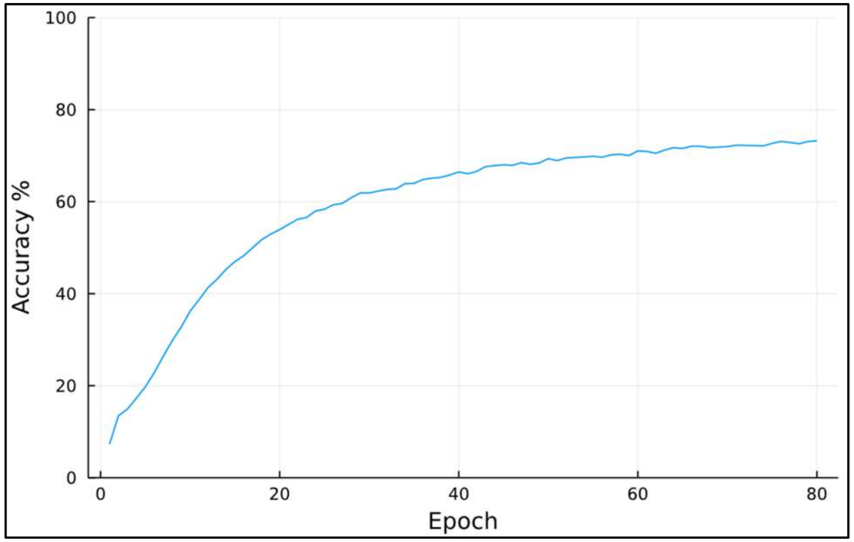
Norse (BP) – 90% in 5 minutes



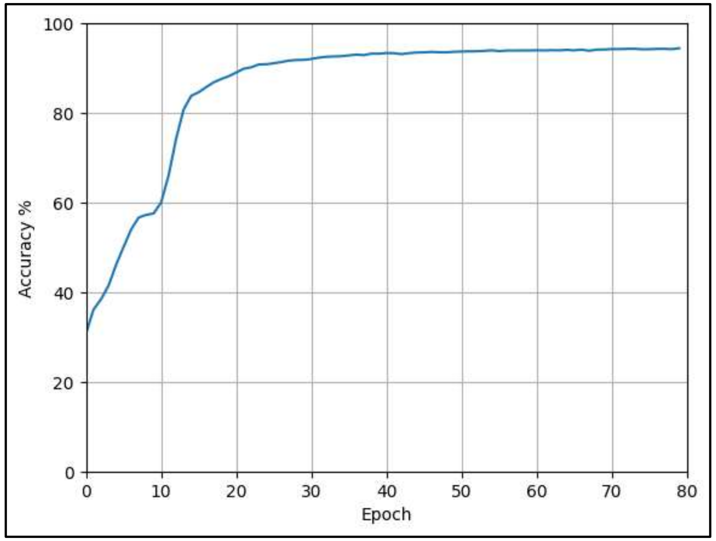
Experimental Results: MNIST Dataset



PSNN – 72%
120m CPU with 3 Compartments



Norse – 95%
5m GPU – 15m CPU



Future Work

- Parallelize compartments
- Port to CUDA or PyTorch/JAX
- Reparameterization trick
- Convolutional Architecture

Thank you!

Questions?