# Exploiting Sparsity for Accelerated SNN Training on Graphcore IPUs

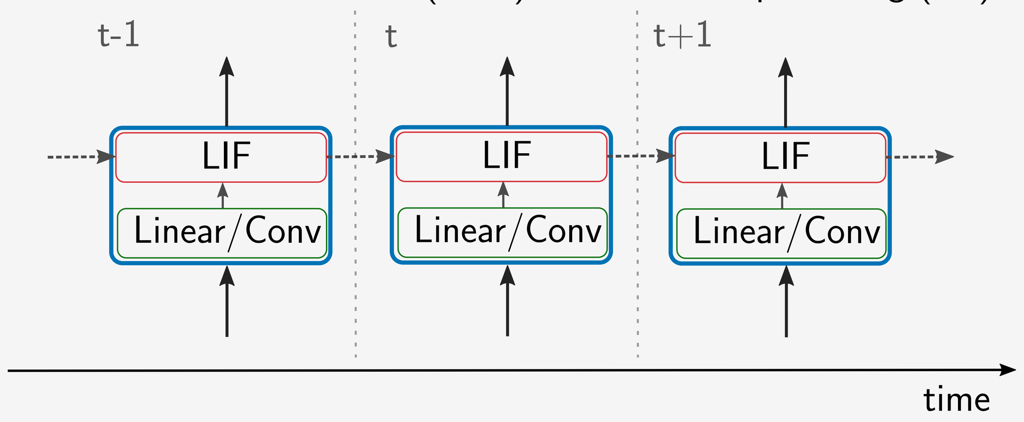
Jan Finkbeiner, Emre Neftci
jan.finkbeiner@fz-juelich.de, e.neftci@fz-juelich.de

## SNNs and Deep Learning

- Time discretized dynamics, of Leaky Integrate and Fire (LIF) neurons for the membrane potential $u$:

$$u_i[t] = \alpha\, u_i[t-1](1 - S_i^{\text{out}}[t-1]) + (1-\alpha)\,\frac{1}{C}\, I_i[t-1]$$
$$S_i^{\text{out}}[t] = \Theta(u_i[t] - \vartheta_i)$$
$$I_i[t] = f(I_i[t-1], S^{\text{in}}[t], S^{\text{out}}[t-1]) = \sum w_{ij}\, S_j^{\text{in}}[t-1]$$

- The computational graph of a time discretized SNN is the one of a recurrent neural network (RNN) as used in deep learning (DL)




⇒ **SNNs are RNNs** with the Heaviside function $\Theta$ as activation function

- Minimization of objective function $L$ by using some form of stochastic gradient descent with respect to the parameters $w_{ij}$:

$$w_{ij} = w_{ij} - \eta \frac{\partial L}{\partial w_{ij}} = w_{ij} - \eta \frac{\partial L}{\partial S_n} \frac{\partial S_n}{\partial u_k} \frac{\partial u_k}{\partial w_{ij}}$$

- To alleviate the zero-gradient issue of the Heaviside function we use the **surrogate gradient method** [4], where a smooth and differentiable surrogate function is used for the gradient computation in the backward pass:

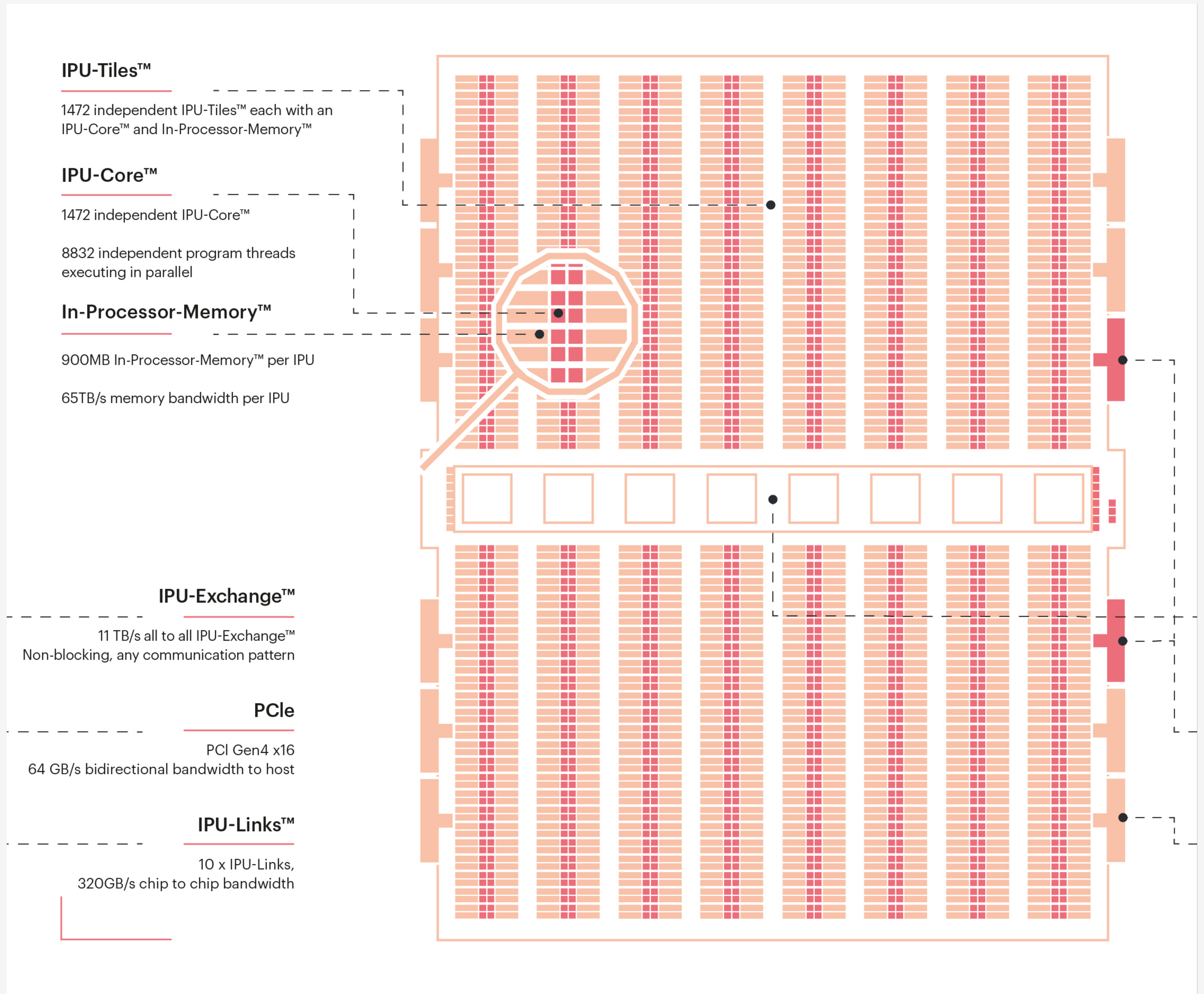
$$S_i^{\text{out}}[t] = \Theta(u_i - \vartheta_i) := \begin{cases} 1, & u_i - \vartheta_i \geq 0 \\ 0, & u_i - \vartheta_i < 0 \end{cases}$$
$$\frac{\partial S_i}{\partial u_i} = \frac{\Theta(u_i - \vartheta_i)}{\partial u_i} = \frac{1}{(\beta |u_i - \vartheta_i| + 1)^2}$$

There are many possible choices for the surrogate function [8]. Here we use the SuperSpike surrogate function [7].

- In order to train the SNNs we use the standard backpropagation through time (**BPTT**) algorithm.

## The Graphcore IPU™

The Intelligence Processing Unit (IPU™) from the UK based company Graphcore is designed to be a massively parallel compute architecture with distributed local memory ideally suited for multiple data multiple instruction (MIMD) workloads. The latest generation of the IPU, the BOW-2000 IPU Machine, can deliver up to 1.4 PFLOPS/s of FP16 AI compute.

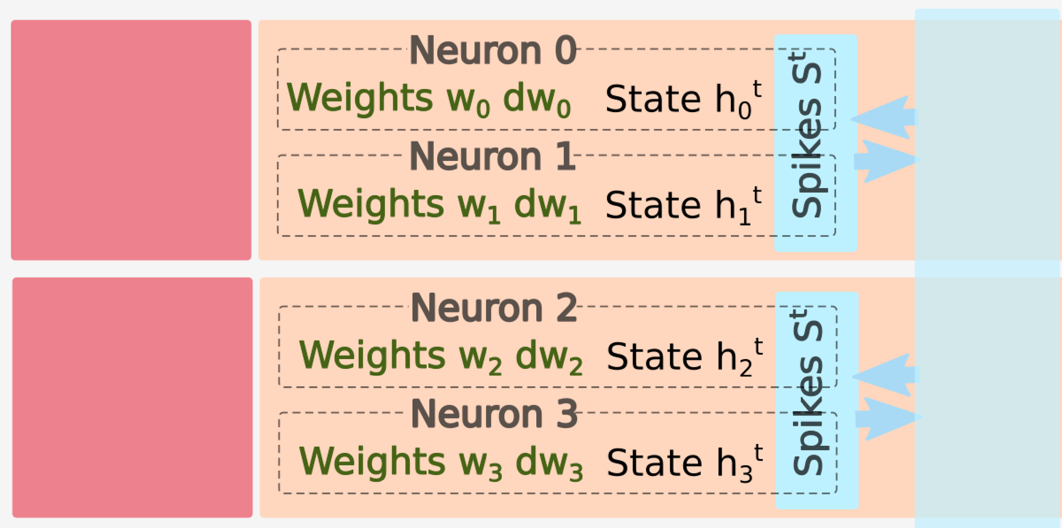



The IPU programming paradigm [3] is based on the Bulk Synchronous Parallel (BSP) model, which features the sequential execution of multiple supersteps composed of:

- a local computation phase,
- a communication phase,
- a barrier synchronization phase.

During each computation phase every IPU-Core™ has access only to its dedicated In-Processor-Memory™, which is 624kB of SRAM. The two together form a so called IPU-Tile™.

## SNNs on the Graphcore IPU™




- Due to the recurrent structure of SNNs we can efficiently distribute the neurons onto dedicated tiles for the whole training. Data transfer between tiles is only necessary for the sparse input spikes and output spike generation. This maximally utilizes the fast access to locally stored parameters and minimizes data transfer.
- As every processor can execute truly independent programs during the computation phases with complex control flow, we believe the IPU is better suited than GPUs to handle sparsity both in activation as well as in connectivity.
- The IPU features very fast intra- and inter-IPU interconnect with high bandwidth and low latency. This feature becomes especially useful when designing large scale, distributed SNNs.
- The IPU is programmable via the C++ based Poplar® SDK for very fine grained control and via the Python API of all major machine learning libraries (Pytorch, Tensorflow, ONNX, ...).

**Sparse Spike representation:**

In order to make use of the sparse activation of SNNs we use a sparse spike representation that only stores the indices and the number of neurons that spiked. A naive implementation would lead to inaccurate gradients in the backward pass. To circumvent this issue we take a very similar approach as in [6], by also propagating some information for neurons that did not spike, but that were close to the threshold.

- membrane potential: [0.8, 1.2, -0.5, 0.6, 0.95, 0.1, 0.98]
- dense spike representation: [ 0, 1, 0, 0, 0, 0, 0]
- sparse spike representation: ([ 1, 4, 6, nan], [1, 3])

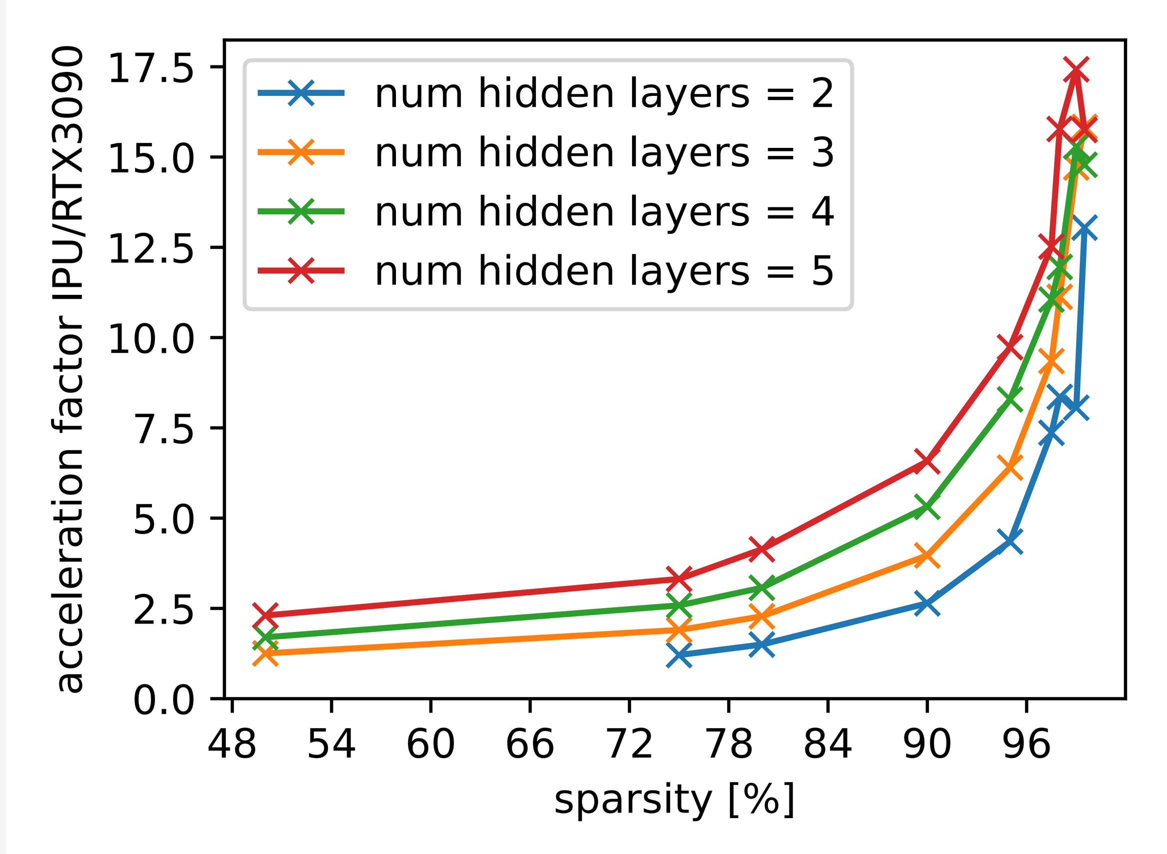
## Performance Benchmarking

In order to benchmark the SNN implementation for the IPU we evaluate the throughput of multiple SNN models using the SHD [2], N-MNIST [5] and DVSGesture [1] dataset and compare to an equivalent dense implementation on a GPU (NVIDIA GeForce RTX 3090, NVIDIA V100, NVIDIA A100). We demonstrate behavior for fully-connected multi-layer SNN architectures.

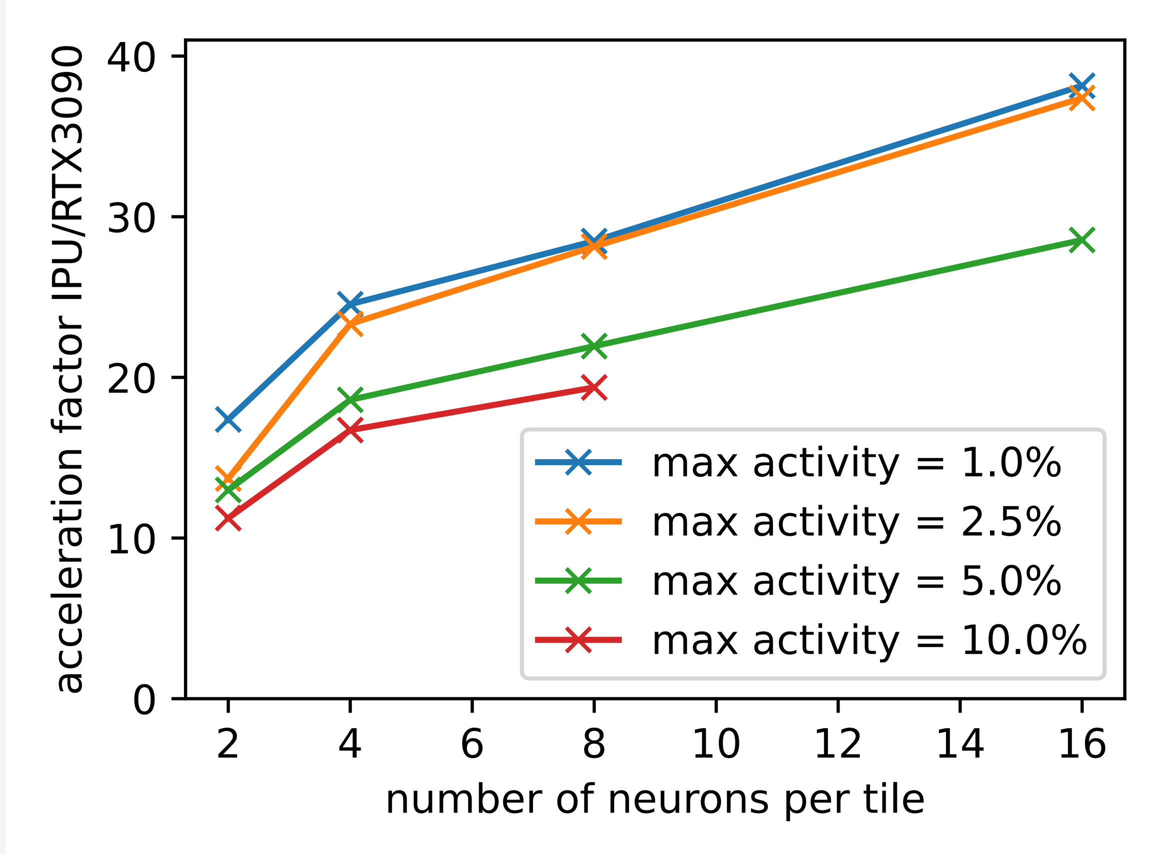
As expected, we observed **increasing throughput with increasing sparsity**. Currently, we can achieve at least **5-10x higher throughput** for realistic training scenarios with the IPU compared to a high-end GPUs.

When **increasing the number of neurons per tile** and therefore per IPU we oberve and at least constant or even textbfincreasing acceleration factor. This build a promising basis for future work on SNNs on increasing size.

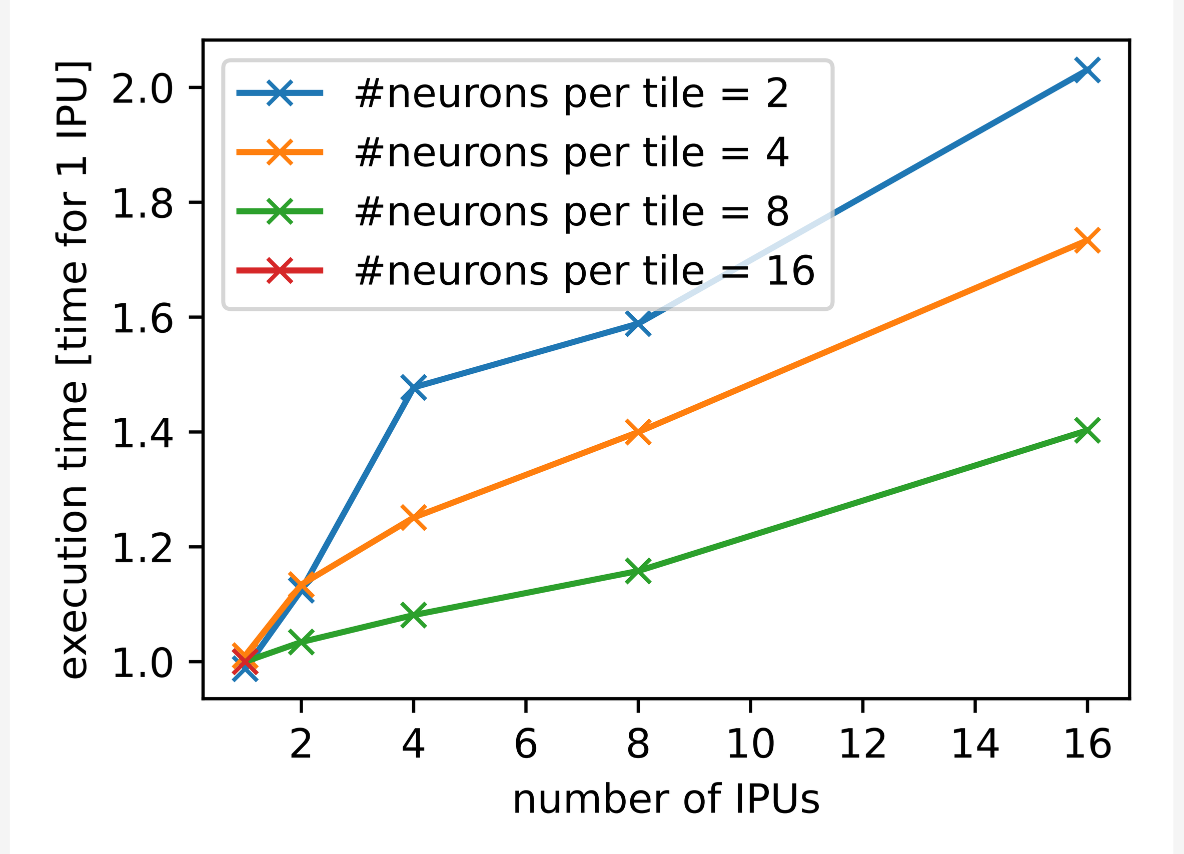
The SNN implementation on **multiple IPUs in a model parallel fashion** demonstrates **good scaling behavior**, which gives the hope for large scale distributed SNN training on large multi-IPU systems.




Spiking Heidelberg Dataset, 100 timesteps, ADAM optimizer




SHD, 10 timesteps, SGD optimizer, hidden layer size ≈ 982




SHD, 10 timesteps, SGD optimizer, max activity = 5.0%

## Execution Traces

Using an analysis tool developed from Graphcore for the IPU, it is possible to get detailed information about the execution and memory allocation of the executed programs. Here, you can see an example of the execution traces, meaning cycle by cycle information which operations were performed. **The left figure displays the execution trace for all operations performed during one timestep in the forward pass, the right figure for the backward pass.** Red color depicts a local computation phase, blue a communication and exchange phase, and yellow a synchronization phase.

## References


- Arnon Amir et al. "A Low Power, Fully Event-Based Gesture Recognition System". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2017.
- Benjamin Cramer et al. "The Heidelberg Spiking Data Sets for the Systematic Evaluation of Spiking Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.7 (2022).
- Zhe Jia et al. *Dissecting the Graphcore IPU Architecture via Microbenchmarking.* 2019.
- Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. "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).
- Garrick Orchard et al. *Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades.* 2015.
- Nicolas Perez-Nieves and Dan Goodman. "Sparse Spiking Gradient Descent". In: *Advances in Neural Information Processing Systems.* Ed. by M. Ranzato et al. 2021.
- Friedemann Zenke and Surya Ganguli. "SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks". In: *Neural Computation* 30.6 (June 2018).
- Friedemann Zenke and Tim P. Vogels. "The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks". In: *Neural Computation* 33.4 (Mar. 2021).