Research Article Vol. 32, No. 9 / 22 Apr 2024 / Optics Express 16260

Optics EXPRESS

Surrogate gradient learning in spiking networks trained on event-based cytometry dataset

MUHAMMED GOUDA,[1,*] STEVEN ABREU,[2] AND PETER BIENSTMAN[1]

[1] *Photonics Research Group, Ghent University, Ghent, Belgium*
[2] *Bernoulli Institute, University of Groningen, Groningen, The Netherlands*
[*] *MuhammedGoudaAhmed.Gouda@UGent.be*

Abstract: Spiking neural networks (SNNs) are bio-inspired neural networks that - to an extent - mimic the workings of our brains. In a similar fashion, event-based vision sensors try to replicate a biological eye as closely as possible. In this work, we integrate both technologies for the purpose of classifying micro-particles in the context of label-free flow cytometry. We follow up on our previous work in which we used simple logistic regression with binary labels. Although this model was able to achieve an accuracy of over 98%, our goal is to utilize the system for a wider variety of cells, some of which may have less noticeable morphological variations. Therefore, a more advanced machine learning model like the SNNs discussed here would be required. This comes with the challenge of training such networks, since they typically suffer from vanishing gradients. We effectively apply the surrogate gradient method to overcome this issue achieving over 99% classification accuracy on test data for a four-class problem. Finally, rather than treating the neural network as a black box, we explore the dynamics inside the network and make use of that to enhance its accuracy and sparsity.

© 2024 Optica Publishing Group under the terms of the Optica Open Access Publishing Agreement

1. Introduction

Spiking neural networks are a particular type of neural network designed to closely imitate a biological brain. In contrast to neurons in an artificial neural network (ANN), a spiking neuron does not have a non-linear activation function. Rather, a leaky-integrate-and-fire (LIF) spiking neuron has an internal state often referred to as membrane potential. When a neuron receives an input spike from pre-synaptic neuron, its membrane potential grows and then decays with a certain time constant, then it rises again if another spike is received. Once the membrane potential is high enough to exceed a certain threshold, the neuron fires a spike [1–4].

In addition to being more biologically plausible, spiking neural networks are attractive since their behaviour allows a hardware implementation with low power consumption, because synapses are only active during the short time a spike is fired. Moreover, in some applications where the temporal signal is already event-based (as is the case in this work), SNNs become the best fit for processing such signals due to their temporal and sparse nature [5].

Nevertheless, the main downside of working with SNNs is the difficulty of training them. Several works in the literature have been carried out targeting training SNNs both in supervised and unsupervised settings. One of the most popular algorithms for unsupervised learning, known as spike time-dependent plasticity (STDP) [6–8], exploits the relative delay between spikes. This is thought to be the closest to how our brain learns [9]. For supervised learning on the other hand, the approach researchers take is similar to the one used for ANNs, which involves backpropagation [10–12].

However, since applying backpropagation involves computing the gradient of the loss with respect to the network weights in the backward path, the following problem arises. A spiking neuron only fires a spike when its membrane potential exceeds a certain threshold. This means that the output of the neuron is a non-linear Heaviside function of its hidden state. Differentiating

#518323 https://doi.org/10.1364/OE.518323
Journal © 2024 Received 18 Jan 2024; revised 3 Apr 2024; accepted 3 Apr 2024; published 18 Apr 2024

such a function gives zero everywhere except at the threshold. Therefore, we end up with a vanishing gradient which prevents the learning algorithm from finding the optimum values for the weights. To overcome this issue, one can keep the non-linear Heaviside function in the forward path when computing the loss, but in the backward path approximate its gradient using a smoother function. This algorithm is known as surrogate gradient learning, and is discussed in more detail in section 2. [1,13].

The input to SNNs in this work is provided by an event-based camera. Contrary to traditional CMOS cameras, an event camera does not capture all the information in the scene in a frame-based manner. Rather, it only generates spikes whenever the intensity of the captured light changes [14]. This process, although simple, opened the door for numerous applications in computer vision and machine learning [14–28].

The application we study here is flow cytometry, which is an analytical technique concerned with the sorting of cells and micro-particles based on their physical characteristics. A particular category of flow cytometers is imaging flow cytometry. Here consecutive images are captured from flowing cells, and then neural networks are trained on the stored images. This approach however leads to vast amounts of data and significant delay due to the deep neural networks utilized. In a recent study [29], we showed that an event-based sensor could enhance classification accuracy while reducing both data storage and the latency of the system. These improvements were mainly attributed to the automatic background subtraction that occurs at the sensor level. The problem addressed in that study was applied to a binary classification problem using two classes of spherical PMMA micro-particles. With a simple logistic regression model, we could reach above 98% accuracy. However, since our objective is to apply the system to a broader range of cells where morphological differences may be less distinct, a more complex machine-learning model such as the SNNs treated here would be necessary.

The rest of the manuscript is structured as follows:

- In section 2, the experiments performed using four classes of micro-particles are presented. Moreover, the architecture of the spiking networks trained in this work is discussed. We explore the effect of different parameters on the performance of the neural networks.
- Section 3 reports different results obtained with the current networks. We focus mainly on the accuracy of the networks by testing them on data coming from experiments other than those used in training. We will also discuss limiting the temporal activity of the hidden layer's neurons, and We will address the sparsity of the network by regularizing its hidden layer's activity.
- In section 4 we discuss the implementation of some spiking neural networks trained with our cytometry dataset on hardware.
- Section 5 sums up the current work and points out to future plans.

2. Methods

2.1. Experiment

The experimental setup, similar to that in [29], comprises a laser source that emits light with a wavelength of 632.8 nm. The light passes through a lens and a 25 µm pinhole before being focused onto a PMMA microfluidic channel. The microfluidic channel utilized is from Chipshop (Fluidic 156) and is a straight-channel chip integrating foru parallel channels where only one channel is used. The channel dimensions are $200\ \mu m \times 200\ \mu m \times 58.5\ mm$.

The channel contains flowing microparticles that are pumped using a manual syringe pump connected to the upper port, while a liquid reservoir is connected to the other port as shown in Fig. 1.

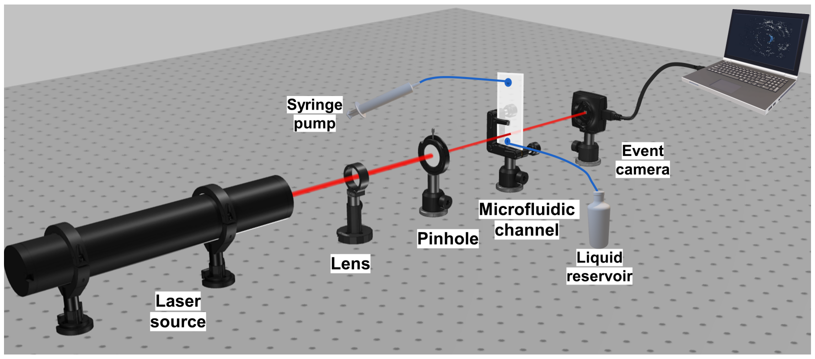


Fig. 1. Experimental setup for generating training and test datasets. A 633 nm He-Ne laser is focused using a lens and directed towards a 25 μm pinhole. The laser light passes through the pinhole before entering a vertically-mounted PMMA microfluidic channel, within which microparticles flow downward. The resulting diffraction and scattering patterns caused by the flowing particles are captured by an event-based camera connected to a laptop running dedicated software for recording the events detected at different pixels.

Four different classes of spherical micro-particles which differ in size according to Table 1 were used in the experiments. The particles were purchased from PolyAn GmbH.

Table 1. Particle classes and sizes

Particle Class	Diameter (μm)
A	9
B	12
C	16
D	20

To prepare the samples, 0.6 ml of particles were diluted with 30 ml of water, and a drop of X-triton surfactant was added to prevent particle clustering. Multiple separate experiments were performed for each particle class. Before pumping particles from a certain class, the microfluidic channel was flushed with water in both directions. This flushing procedure was repeated a number of times between measurements. The fluctuations in the laser signal due to the diffraction and scattering effects produced by the flowing particles were recorded using a Prophesee EVK-1-VGA event camera [30]. This interaction between the light source and the moving particles translates the features to a higher dimensional space where classification can be done more efficiently. This resembles the kernel method in machine learning [31] where functions acting on data features are computed. The advantage here is that such computations are done at high speed in the optical domain, which accelerates the overall system.

2.2. SNN architecture

During a recording session involving one of the four classes of particles, an event camera captures the movement of these particles as they enter the field of view illuminated by a laser. The event camera detects and records events in the form of both positive and negative signals. When a particle enters the field of view, the camera starts firing events, indicating its presence. The number of fired events reaches a peak as the particle moves through the field of view, and then gradually drops as the particle exits as shown in the bottom left corner of Fig. 2. The camera sensor has a temporal resolution of $\Delta t = 1\mu s$. Taking into account the fact that a particle takes

$10ms$ to pass through the field of view, we would end up with $10ms/1\mu s = 10,000$ time steps if we wanted to use the same temporal resolution for the neural network as for the camera. Therefore, for computational efficiency, a temporal compression was applied by clustering events into just 70 time bins. For each pixel, and for each simulation step corresponding to such a time bin, we input a single spike in the network in case the camera generated one or more spikes during that interval for that pixel.

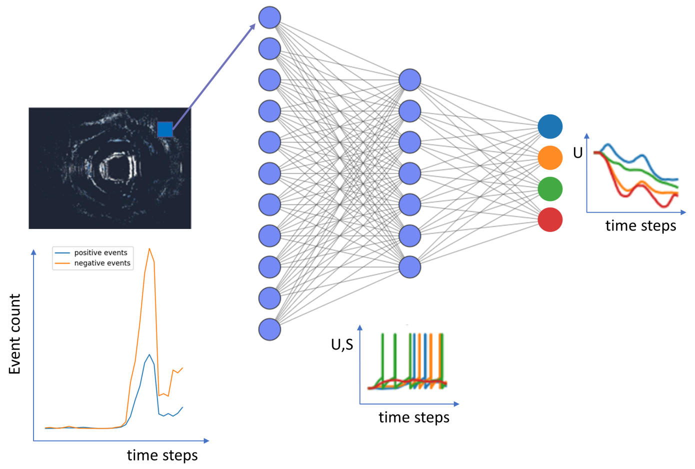


Fig. 2. The architecture of the neural network used in this work. It consists of three layers: an input layer, a hidden layer consisting of 100 neurons, and an output layer of 4 neurons corresponding to the four classes of particles. The event count during the passage of a particle through the field of view is a bell-like curve. This traversal period is split into multiple time bins. During each bin, the corresponding events are sent to the network. Then, the hidden layer neurons receive a weighted sum of the input spikes. If the internal state U of a neuron in the hidden layer is excited enough, a spike S is generated and sent to the output layer. The neuron with the maximum potential is chosen to be the predicted class.

What makes spiking neural networks unique is that they possess memory, even in a feedforward architecture. Indeed, the network's internal state and the activation of each neuron depend not only on the current events but also on past events from the flowing particle. This memory enables the network to capture temporal dependencies and learn patterns in the particle's movement over time.

The sensor employed possesses a resolution of 640 pixels in width and 480 pixels in height, resulting in a total of 307,200 individual pixels. In order to facilitate data processing, we downscale this to 64 pixels in width and 48 pixels in height, yielding a reduced pixel count of 3,072. To further analyze and process the captured data, the positive and negative events originating from these pixels are directed to separate neurons, thereby necessitating the presence of 6,144 neurons in the input layer of our neural network architecture.

Spikes from the input layer of the network are then multiplied by synaptic weights and fed to the hidden layer of the network (100 neurons were used in this layer). At each time step, neurons in the hidden layer with high enough membrane potential will fire spikes which are weighted and sent to the output layer of the network.

Four neurons are used in the output of the network, corresponding to the four classes of particles in the experiment. Figure 2 shows the output neurons and their hidden states with different colors. At the end of the simulation time for each sample, a decision is made as follows: the neuron with the highest average membrane potential is considered to be the predicted neuron/class. This is based on the convention in Spytorch [32]. Some other implementations in literature base a decision instead on the firing rate/time of the output spikes [33–36]. However, since the output spike of a neuron is directly related to its membrane potential, both approaches should give similar results.

2.3. Neuron dynamics

We focus here on a very specific model of spiking neurons, which is the leaky-integrate-and-fire model. Such neurons have a membrane potential which resembles the hidden state of a neuron in an RNN. The network consists of a number of layers, each layer is made up of multiple neurons. The equation governing the membrane potential U_i^l of neuron i in layer l is as follows:

$$\tau_{mem}\frac{dU_i^l}{dt} = -(U_i^l - U_{reset}) + RI_i^l \tag{1}$$

In this first-order differential equation, τ_{mem} is the time constant for the membrane potential. U_{reset} is the reset potential, and R and I are the input resistance and current.

Whenever its membrane potential U_i^l exceeds a chosen threshold U_{th}, we have it emit a spike. In discrete time, a spike train is represented as follows:

$$S_i^l(t) = \Sigma_{k\in C_i^l}\delta(t - t_i^k) \tag{2}$$

This is a series of Dirac delta functions with C_i^l representing the time indices of all the spikes fired by neuron i in layer l. t_i^k is then the actual time at which this neuron fired a specific spike. A spike is transmitted to its postsynaptic neurons through a current I which is governed by the following differential equation:

$$\frac{dI_i^l(t)}{dt} = -\frac{I_i^l(t)}{\tau_{syn}} + \Sigma_j W_{ij}^l S_j^{l-1}(t) \tag{3}$$

Here, W are the feed-forward weights. Solving the above linear ODEs yields (after some steps, see [32]) the following equation

$$I_i^l(t+1) = \alpha I_i^l(t) + \Sigma_j W_{ij}^l S_j^{(l-1)}(t) \tag{4}$$

which gives the development of the synaptic current. For the membrane potential, one obtains:

$$U_i^l(t+1) = \beta U_i^l(t) + I_i^l(t) - S_i^l(t) \tag{5}$$

α is the synaptic decay rate, and β is the decay rate of the membrane potential. Both are linked to the time constants discussed above through the following relations:

$$\alpha = \exp\left(-\frac{\Delta t}{\tau_{syn}}\right) \tag{6}$$

$$\beta = \exp\left(-\frac{\Delta t}{\tau_{mem}}\right) \tag{7}$$

The output spike in relation to the membrane potential is represented by the following equation:

$$S_i^l = \Theta(U_i^l(t) - U_{th}) \tag{8}$$

where Θ is the Heaviside function shown in Fig. 3.

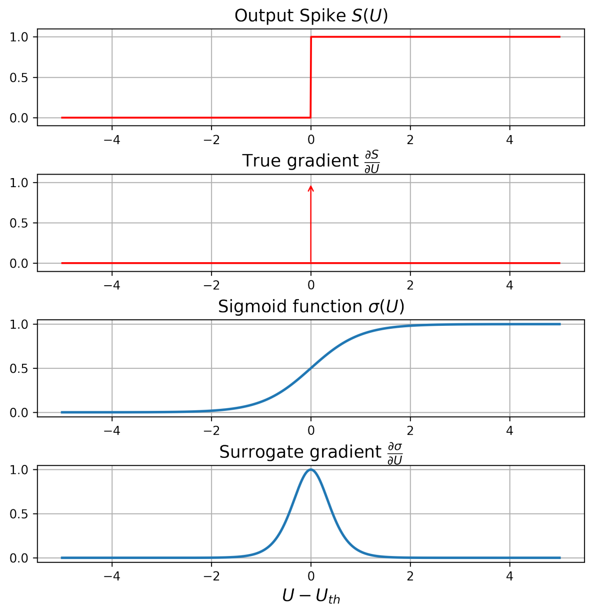


Fig. 3. A plot showing the output of the neuron as a function of its membrane potential. The derivative of this function is vanishing almost everywhere which creates the dead neuron problem. The bottom plot shows the derivative of a sigmoid function which is the approximation of the gradient used in the surrogate training algorithm.

2.4. Dead neuron problem and surrogate gradient

Applying backpropagation requires taking the derivative of the Heaviside function in (8) which yields a delta function with zero gradients almost everywhere. This phenomenon is known as the dead neuron problem and prevents the learning algorithm from finding a minimum of the loss function. To overcome this issue, Neftci et al. (2019) proposed a technique known as surrogate gradient learning in their study [32]. The surrogate gradient involves replacing the gradient of the Heaviside with an alternative gradient like the derivative of the sigmoid function in Fig. 3. Note that this only happens in the backward path. In the forward path, the Heaviside function is retained.

2.5. Relating the decay rate of the neurons to the event rate from the experiment

One of the crucial parameters in SNNs is the time constant τ_{mem} of the neuron. As discussed earlier in this article, the membrane potential decays to zero (or to a reset value) after it receives a spike from a pre-synaptic neuron. How fast this decay happens impacts how often this post-synaptic neuron will emit spikes to the next layer in the network. In some cases, the post-synaptic neuron might not emit any spikes if its membrane potential decays too quickly. If no spikes are fired between layers, no learning will happen. Therefore, a good value for the decay rate has to be found.

One way to choose a good value for the neuron's decay rate (or time constant) is by including it in the list of hyper-parameters and doing a grid search over the whole list of hyperparameters. This however will increase the time it takes to find an optimum network. In this work, we decided to choose an approximate value based on the firing rate of the event sensor governed by the following inequality

$$\tau_{decay} > \Delta t$$

where τ_{decay} is the decay time constant of the spiking neuron and Δt is the average time difference between two consecutive events generated by the sensor. Note that the latter parameter is not a characteristic of the sensor itself, rather it depends on how fast the particles move through the field of view which in turn depends on the flow rate generated by the pump and the dimensions of the microfluidic channel. The value that was finally chosen for the membrane time constant is 10 ms.

2.6. Number of time-steps and the trade-off between latency and accuracy

Another parameter that has a significant impact on the network performance is the number of time steps. As mentioned earlier, events from each sample are sent to the network over a number of time steps. Here we experimented with different values for that parameter. It has been observed that there is a minimum number below which the network performance deteriorates significantly. On the other hand, using a large number of time steps to simulate the network increases the latency of the overall system. The value we ended up choosing is 70 timesteps.

3. Results

3.1. Training with the true gradient vs training with a surrogate gradient

The network in Fig. 2 was trained using data from multiple experiments with four different classes of particles. The network's weights were optimized using a gradient-descent-based optimizer, namely the Adam optimizer [37], on the log-loss function. Figure 4 shows the progression of the training loss over multiple training epochs. The training was performed twice. In the first approach, the true gradient of the spike with respect to the membrane potential was used. This gradient is vanishing everywhere except at the threshold point as we discussed in the previous section. Hence, this did not lead the network to learn the important features present in the data as evidenced by the almost constant train loss. On the other hand, when the surrogate gradient of a sigmoid function was used, the loss significantly decreased after 50 epochs showing that the network was able to optimize its parameters in the direction of minimum loss.

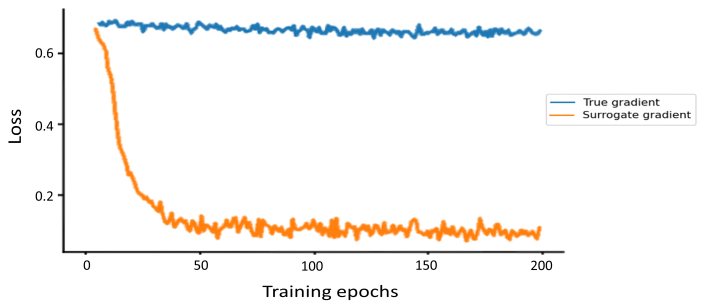


Fig. 4. Evolution of the loss with the training epochs. Training using a surrogate gradient results in a significant drop in the loss after only 50 epochs, while using the true gradient spike with respect to the membrane potential of the spiking neuron does not assist the network in learning the classes of particles. This is due to the vanishing gradient problem discussed in the text.

The effect of batch size on both the validation and test accuracy was also investigated. Our findings indicate that a relatively small batch size, such as 10 samples, can attain minimum train loss within fewer than 50 epochs (see Fig. 5). However, this approach is typically accompanied by poor validation accuracy. On the other hand, large batch sizes, such as 300, require a substantially greater number of epochs to attain minimum train loss, while also avoiding overfitting on the

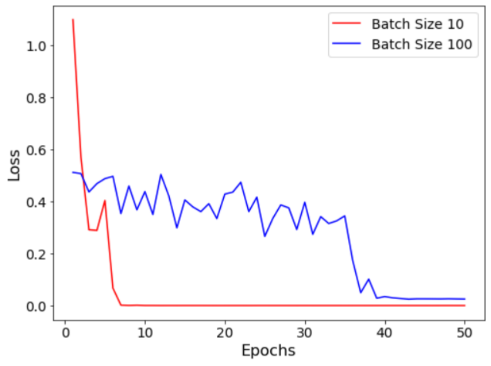


Fig. 5. Learning curve when different batch sizes (10 and 100) are used. For larger batch size, the training algorithm needs more epochs to reach minimum loss. On the contrary, using a very small batch requires a smaller number of training epochs but leads to over-fitting on training data.

training data. Thus, a batch size of 30 was determined to offer a favorable compromise between the aforementioned metrics.

3.2. Regularizing hidden layer activity

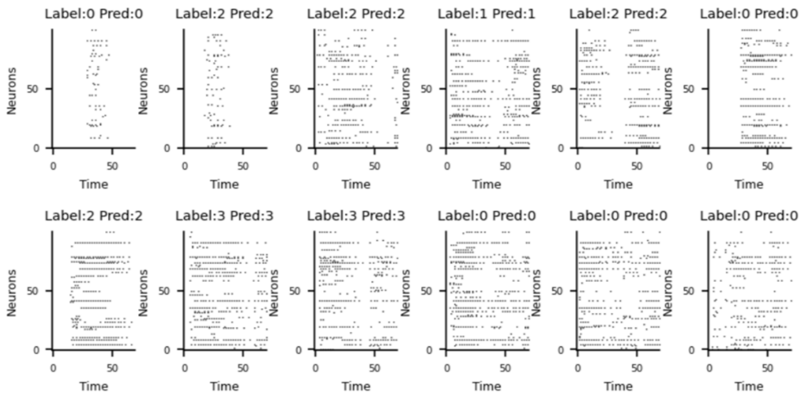
One of the main characteristics of an SNN that makes it attractive for applications requiring low power consumption, is the fact that these networks are only active when spikes are emitted. To maximize this benefit, one should aim at making the neurons silent most of the time. Therefore, we modifying the loss function by including a regularization term.

$$L = -\Sigma_{j=1}^{M} y_j \log(\hat{y}_j) + \lambda \Sigma_{k=1}^{N} S_k \tag{9}$$

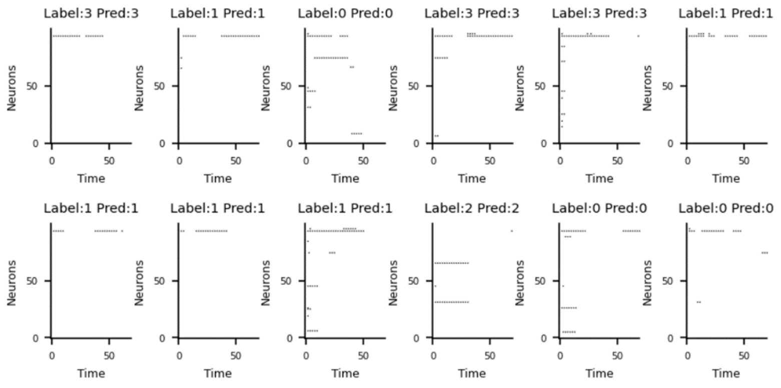
In the previous equation, the first term corresponds to the traditional negative log-likelihood loss, which runs over all the samples from $j = 1$ to $j = M$, where M is the number of samples. $\hat{y}$ is the predicted value by the network, while y is the actual label of the sample. The second term is the regularization term with its regularization parameter λ and S_k representing the total number of spikes fired by neuron k. Note that we only include neurons in the hidden layer.

Figure 6 shows the effect of adding such a regularization term on the dynamics of the network. The activity of the neurons in the hidden layer is shown using a raster plot. It resembles a binary heat-map with a black pixel in case there is a spike fired and a white pixel when the membrane potential is not high enough for a spike to be fired. The network with no regularization shows a large number of spikes generated at different time steps. If this were to be implemented on hardware, it would mean that the chip would require relatively high power for inference. On the other hand, the regularized network suppresses such high activity by penalizing the total number of fired spikes in the hidden layer.

The regularization parameter λ controls the strength to which the activity in the hidden layer is suppressed. When a relatively high value of 10^{-3} was used, no spikes were fired at all. Having nothing to be propagated to the output layer, the network could not learn anything, since it focuses on minimizing the regularization term while ignoring the logloss term. After optimising hyperparameters, It was found that a value of $6.7 * 10^{-6}$ gives a good compromise between the two terms in the loss function, achieving high classification accuracy while resulting in sparse network dynamics.



(a)

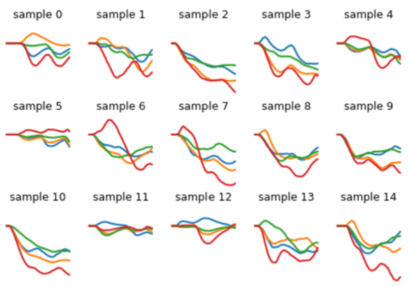


(b)

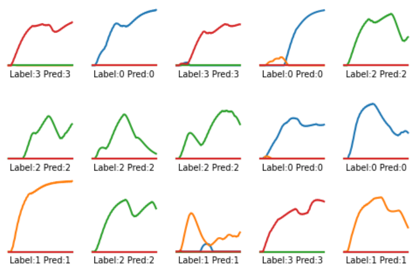
Fig. 6. Raster plot showing the activity in the hidden layer of two different trained neural networks. In **(a)**, the network was trained using a loss function consisting only of the logloss term. On the other hand, the network in **(b)** has a regularized loss function. These samples were drawn randomly from the dataset. Both networks achieve the same goal of predicting the correct label of a sample, but the second network is much more sparse, making it more power efficient.

3.3. Results for optimal hyperparameters

After running a grid search over the learning rate and the regularization parameter, their optimal values were found. Then, a final training loop was performed on the combination of training and validation splits. The resulting network was tested on data coming from a newly unseen experiment to assess its generalization capabilities. Figure 7 shows the output membrane potential for randomly-chosen samples both with suboptimal ($\lambda = 10^{-5}$, learning rate = 10^{-3}) and optimal ($\lambda = 6.7\ 10^{-3}$, learning rate = 10^{-3}) hyperparameters. The optimized network resulted in much higher classification accuracy, which is also illustrated by the confusion matrix in Fig. 8. Furthermore, the significant difference between the membrane potential of the correctly predicted neuron and those of the other three neurons indicates that the network gives its prediction with high confidence.

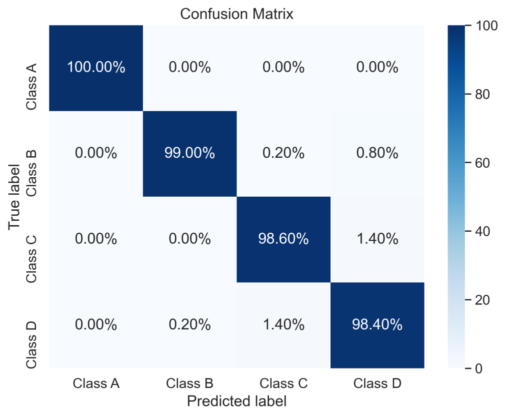


(a)



(b)

Fig. 7. Membrane potentials of the four nodes in the output layer for different randomly-chosen samples. The output in **(a)** is obtained from a network with parameters less optimal than those of the network in **(b)**. This resulted in the fact that the first network gave more errors. Moreover, the difference between the membrane potentials in a random sample is more significant in the second network meaning that it is also more confident of its predictions than the first network.


Confusion Matrix

True label \ Predicted label	Class A	Class B	Class C	Class D
Class A	100.00%	0.00%	0.00%	0.00%
Class B	0.00%	99.00%	0.20%	0.80%
Class C	0.00%	0.00%	98.60%	1.40%
Class D	0.00%	0.20%	1.40%	98.40%

Fig. 8. Confusion matrix of the text experiment. The maximum percentages located on the diagonal imply that the network is accurate on data coming from experiments unseen in the training folds.

4. SNN hardware implementation on Intel's Loihi chip

Finally, we investigated implementing the inference stage of the SNN on specialised electronic hardware using the second generation of Intel's Loihi chip [38]. This neuromorphic chip was used only for inference of the SNN, the training was still done conventionally on a GPU. We tested two different networks, one consisting of a single hidden layer with 512 neurons and another consisting of two hidden layers with 512 neurons each.

Regarding the mapping of the output obtained in Fig. 1 onto the Intel's Loihi chip, to prepare our event data for processing on the Loihi chip, we implement two key pre-processing steps:

1. **Downsampling:** We initially downsample the event camera's resolution by 20x, resulting in a 32x24 pixel grid with two channels. During this process, duplicate events are retained.
2. **Leaky Integrate-and-Fire (LIF) Neurons:** Subsequently, we pass all samples through a layer of LIF neurons. Each downsampled pixel corresponds to a neuron indexed by x (ranging from 0 to 31), y (ranging from 0 to 23), and p (representing the two channels). The membrane potential $v_{xy}(t)$ of each neuron evolves over time according to the number of spikes $n_{xy}(t)$ received at time t, with parameters $\sigma = 0.9$, $w = 1.0$, and $v_{\text{thr}} = 3.0$. Additionally, a refractory period $T_{\text{refr}} = 2$ ensures that all incoming spikes are ignored for a brief duration after a spike occurrence. After each spike at time t^*, the membrane potential is reset to $v_{xy}(t^* + 1) = v_{\text{rest}} = 0$.

These pre-processing steps lead to approximately 85% compression of the dataset. Importantly, our entire pre-processing pipeline is designed to be compatible with the neuromorphic chip utilized in this experiment.

The Loihi 2 chip comprises 128 neurocores, each of which may simulate up to 8192 neurons. The 1-layer SNN occupies 44 neurocores, utilizing 6,620 synapses in total. On the other hand, the 2-layer SNN takes up 57 neurocores and uses 13,039 synapses overall. The energy consumption of the inference on the chip scales proportionally to the number of synaptic operations. When analyzing a single sample of 10ms at a sampling time of 10μs, the 1-layer SNN needed 392,200 synaptic operations. On the other hand, the 2-layer SNN required 6,072,829 synaptic operations, thus 15 times more than the 1-layer network. The time it takes to run a sample through the 2-layer SNN is however only 4 times longer than for the 1-layer network since many of the operations in the 2-layer network are executed in parallel and not sequentially. The accuracy of both networks is comparable, with the 1-layer network actually outperforming the larger 2-layer network. The 1-layer network showed an average test accuracy of 98.13% while the 2-layer network showed an average test accuracy of 97.44%, averaged over different data splits.

4.1. Latency estimation

Estimating the inference latency on Intel's Loihi chip involves several considerations. Regarding computation on the chip itself, based on our estimations considering the number of time steps per sample and the clock frequency, we anticipate the computation latency to be less than 1 millisecond.

However, a significant concern lies in the communication to the chip, which has been challenging to precisely gauge due to several factors. Notably, we encountered limitations in estimating communication latency. At the time, the real event-based communication interface was still under development, thus we had to resort to using an alternative, albeit slower, interface. This resulted in an approximate latency of around 20 seconds per sample. We plan to conduct experiments with the new communication interface to gain better insights into the true latency characteristics when interfacing with the Loihi chip.

For more information on the implementation on Loihi, see the paper by Abreu *et al.* [39].

5. Conclusion

To summarize, this work integrates spiking neural networks and event-based vision sensors for the purpose of classifying micro-particles in label-free flow cytometry. While our previous work used a simple logistic regression with binary labels and achieved high accuracy, this study recognizes the need for a more advanced machine-learning model to adapt to a wider range of cells. To overcome the challenge of training such networks, the surrogate gradient method was effectively applied resulting in classification accuracy of more than 99% on test data. The study also explores the dynamics inside the network and focuses on enhancing not only the accuracy but also the power efficiency through regularizing the loss function. The results demonstrate the potential of this integrated approach for improving the classification of micro-particles through and suggest possibilities for future research in flow cytometry.

Funding. Horizon 2020 Framework Programme (871330); Prometheus (101070195); Post-Digital (860360); Fonds Wetenschappelijk Onderzoek (G006020N).

Acknowledgements. This work was performed in the context of the European H2020 projects Neoteric (grant agreement 871330), Prometheus (grant agreement 101070195) and Post-Digital (grant agreement 860360), and in the Flemish FWO project G006020N and the Belgian EOS project G0H1422N.

Disclosures. The authors declare that there are no conflicts of interest regarding the publication of this manuscript.

Data availability. The raw files containing the events recorded during different experiment sessions are accessible upon request. Please contact the corresponding author for access to the data.

References

1. F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks," Neural computation **33**(4), 899–925 (2021).
2. Y. Li, Y. Guo, S. Zhang, *et al.*, "Differentiable spike: Rethinking gradient-descent for training spiking neural networks," Advances in Neural Information Processing Systems **34**, 23426–23439 (2021).
3. S. Deng, Y. Li, S. Zhang, *et al.*, "Temporal efficient training of spiking neural network via gradient re-weighting," arXiv, arXiv:2202.11946 (2022).
4. K. M. Stewart and E. O. Neftci, "Meta-learning spiking neural networks with surrogate gradient descent," Neuromorph. Comput. Eng. **2**(4), 044002 (2022).
5. E. Ledinauskas, J. Ruseckas, A. Juršėnas, *et al.*, "Training deep spiking neural networks," arXiv, arXiv:2006.04436 (2020).
6. H. Markram, W. Gerstner, and P. J. Sjöström, "A history of spike-timing-dependent plasticity," Front. Synaptic Neurosci. **3**, 4 (2011).
7. N. Caporale and Y. Dan, "Spike timing–dependent plasticity: a hebbian learning rule," Annu. Rev. Neurosci. **31**, 25–46 (2008).
8. M. C. Van Rossum, G. Q. Bi, and G. G. Turrigiano, "Stable hebbian learning from spike timing-dependent plasticity," J. Neurosci. **20**(23), 8812–8821 (2000).
9. K. Yamazaki, V.-K. Vo-Ho, D. Bulsara, *et al.*, "Spiking neural networks and their applications: A review," Brain Sci. **12**(7), 863 (2022).
10. B. J. Wythoff, "Backpropagation neural networks: a tutorial," Chemom. Intell. Lab. Syst. **18**(2), 115–155 (1993).
11. R. J. Erb, "Introduction to backpropagation neural network computation," Pharm. Res. **10**(2), 165–170 (1993).
12. R. Rojas and R. Rojas, "The backpropagation algorithm," Neural networks: a systematic introduction pp., 149–182 (1996).
13. K. Che, L. Leng, K. Zhang, *et al.*, "Differentiable hierarchical and surrogate gradient search for spiking neural networks," Advances in Neural Information Processing Systems **35**, 24975–24990 (2022).
14. G. Gallego, T. Delbrück, G. Orchard, *et al.*, "Event-based vision: A survey," IEEE Trans. Pattern Anal. Mach. Intell. **44**(1), 154–180 (2020).
15. T. Delbrück, B. Linares-Barranco, E. Culurciello, *et al.*, "Activity-driven, event-based vision sensors," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, (IEEE, 2010), pp. 2426–2429.
16. B. Li, H. Meng, Y. Zhu, *et al.*, "Enhancing 3-d lidar point clouds with event-based camera," IEEE Trans. Instrum. Meas. **70**, 1–12 (2021).
17. S. Shiba, Y. Aoki, and G. Gallego, "Fast event-based optical flow estimation by triplet matching," IEEE Signal Process. Lett. **29**, 2712–2716 (2022).
18. F. Tschopp, C. von Einem, A. Cramariuc, *et al.*, "Hough^2 map–iterative event-based hough transform for high-speed railway mapping," IEEE Robot. Autom. Lett. **6**(2), 2745–2752 (2021).
19. F. Tschopp, "Visual positioning for railway vehicles," Ph.D. thesis, ETH Zurich (2021).
20. P. N. McMahon-Crabtree and D. G. Monet, "Commercial-off-the-shelf event-based cameras for space surveillance applications," Appl. Opt. **60**(25), G144–G153 (2021).

21. Y. Ng, Y. Latif, T.-J. Chin, *et al.*, "Asynchronous kalman filter for event-based star tracking," in *Computer Vision–ECCV 2022 Workshops: Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part I*, (Springer, 2023), pp. 66–79.
22. S. Tulyakov, D. Gehrig, S. Georgoulis, *et al.*, "Time lens: Event-based video frame interpolation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, (2021), pp. 16155–16164.
23. W. He, K. You, Z. Qiao, *et al.*, "Timereplayer: Unlocking the potential of event cameras for video interpolation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (2022), pp. 17804–17813.
24. G. Chen, H. Cao, J. Conradt, *et al.*, "Event-based neuromorphic vision for autonomous driving: A paradigm shift for bio-inspired visual sensing and perception," IEEE Signal Process. Mag. **37**(4), 34–49 (2020).
25. F. Bergner, E. Dean-Leon, and G. Cheng, "Event-based signaling for large-scale artificial robotic skin-realization and performance evaluation," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (IEEE, 2016), pp. 4918–4924.
26. J.-A. Sahel, E. Boulanger-Scemama, C. Pagot, *et al.*, "Partial recovery of visual function in a blind patient after optogenetic therapy," Nat. Med. **27**(7), 1223–1229 (2021).
27. G. Gallego, J. E. Lund, E. Mueggler, *et al.*, "Event-based, 6-dof camera tracking from photometric depth maps," IEEE Trans. Pattern Anal. Mach. Intell. **40**(10), 2402–2412 (2017).
28. C. Yang, P. Liu, G. Chen, *et al.*, "Event-based driver distraction detection and action recognition," in *2022 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, (IEEE, 2022), pp. 1–7.
29. M. Gouda, A. Lugnan, J. Dambre, *et al.*, "Improving the classification accuracy in label-free flow cytometry using event-based vision and simple logistic regression," IEEE J. Sel. Top. Quantum Electron. **29**(2), 1–8 (2023).
30. Prophesee, "Vga-cd event-based evaluation kit," (n.d.).
31. T. Hofmann, B. Schölkopf, and A. J. Smola, "A review of kernel methods in machine learning," Mac-Planck-Institute Technical Report **156**, 1 (2006).
32. 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," IEEE Signal Process. Mag. **36**(6), 51–63 (2019).
33. N. Dennler, G. Haessig, M. Cartiglia, *et al.*, "Online detection of vibration anomalies using balanced spiking neural networks," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, (IEEE, 2021), pp. 1–4.
34. S. R. Kulkarni and B. Rajendran, "Spiking neural networks for handwritten digit recognition–supervised learning and network optimization," Neural Networks **103**, 118–127 (2018).
35. S. Pande, F. Morgan, G. Smit, *et al.*, "Fixed latency on-chip interconnect for hardware spiking neural network architectures," Parallel computing **39**(9), 357–371 (2013).
36. A. Stöckel and C. Eliasmith, "Passive nonlinear dendritic interactions as a computational resource in spiking neural networks," Neural computation **33**(1), 96–128 (2021).
37. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv, arXiv:1412.6980 (2014).
38. M. Davies, N. Srinivasa, T.-H. Lin, *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," IEEE Micro **38**(1), 82–99 (2018).
39. S. Abreu, M. Gouda, A. Lugnan, *et al.*, "Flow cytometry with event-based vision and spiking neuromorphic hardware," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (2023), pp. 4139–4147.

157.193.240.5 {ts '2024-05-14 05:28:12'}