Frameworks for SNNs: a Review of Data Science-oriented Software and an Expansion of SpykeTorch

Davide L. Manna[1], Alex Vicente-Sola[1], Paul Kirkland[1], Trevor J. Bihl[2], and Gaetano Di Caterina[1]

[1] Neuromorphic Sensor Signal Processing (NSSP) Lab,
University of Strathclyde, G1 1XW, Glasgow, UK
davide.manna@strath.ac.uk
https://nssp.eee.strat.ac.uk

[2] Air Force Research Laboratory (AFRL),
Wright-Patterson AFB, Ohio, USA

Abstract. The Neuromorphic (NM) field has seen significant growth in recent years, especially in the development of Machine Learning (ML) applications. Developing effective learning systems for such applications requires extensive experimentation and simulation, which can be facilitated by using software frameworks that provide researchers with a set of ready-to-use tools. The NM technological landscape has witnessed the emergence of several new frameworks in addition to the existing libraries in neuroscience fields. This work reviews nine frameworks for developing Spiking Neural Networks (SNNs) that are specifically oriented towards data science applications. We emphasize the availability of spiking neuron models and learning rules to more easily direct decisions on the most suitable frameworks to carry out different types of research. Furthermore, we present an extension to the SpykeTorch framework that enables users to incorporate a broader range of neuron models in SNNs trained with Spike-Timing-Dependent Plasticity (STDP). The extended code is made available to the public, providing a valuable resource for researchers in this field.

Keywords: frameworks · spiking neural networks · spiking neurons · neuromorphic · software · machine learning · unsupervised learning

1 Introduction

The development of Deep Learning (DL) algorithms was greatly eased by the introduction of purposely developed software packages. These packages, or frameworks, usually offer a wide range of software tools that aim to speed up the development of Machine Learning (ML) pipelines as well as make the algorithms available to a larger audience. When referring to conventional DL, i.e. non Neuromorphic (NM), several famous libraries exist, such as TensorFlow (TF) [2], PyTorch [43] or Caffe [32]. The field of Neuromorphic engineering has recently

This is a peer-reviewed, accepted author manuscript of the following paper: Manna, D. L., Vicente Sola, A., Kirkland, P., Bihl, T. J., & Di Caterina, G. (Accepted/In press). Frameworks for SNN: a review of data science-oriented software and an expansion of SpykeTorch. Paper presented at Engineering Applications and Advances of Artificial Intelligence, León, Spain.

seen the emergence of several new software frameworks thanks to the renewed interest in its potential. However, these frameworks are often in an early development stage when compared to their conventional DL counterpart, being limited in the tools they offer, their documentation, and the support from the community. Some more established frameworks also exist, but they are often directed towards particular communities and use cases [48], or they are neuroscience-oriented frameworks rather than NM-ML development tools. Furthermore, effective data science algorithms that can close the gap with other conventional methodologies still need to be developed. Indeed, algorithms employing Spiking Neural Networks (SNNs) are already more energy efficient than conventional Convolutional Neural Networks (CNNs) [25], however, they are not as effective on ML tasks in terms of accuracy. Hence the importance of having good software frameworks that enable customization, simulation and deployment of SNNs. This requires combining a number of key elements into a pipeline such as learning rules, connectivity patterns, and spiking neurons. Regarding the spiking neurons, emerging NM chips such as Loihi 2 [41] allow the use of customized models. It has been shown in the literature that different types of neuron models can solve certain tasks more effectively than other models [22,36]. Therefore it can be beneficial for researchers to use a framework that enables seamless experimentation with different types of neurons.

This work contributes by providing a review of data science-oriented frameworks and highlighting the key features they offer. By restricting our review to this kind of frameworks, we hope to help boosting new research in NM for ML applications. Further to this, we develop an expansion[3] of the SpykeTorch [38] framework that enables the user to experiment on a wider variety of different spiking neuron models. By doing this, we aim to enlarge the scope of the research in SNNs to different spiking neuron models, and to thus build new algorithms that can leverage the latest advances in the NM hardware.

2 Related Works

When presenting a new software framework, authors often report other similar works and draw comparisons with them [38,27]. In these instances, differences in terms of offered features are highlighted, as well as the advantages of using the newly presented software over the existing ones. Other works specifically focus on reviewing the existing frameworks for the development of SNNs. One example is given by [45], where the authors make a subdivision of the software packages into three main groups depending on whether they are NM chips toolchains, SNN simulation frameworks or frameworks that integrate SNNs and DNNs. Another work [25] gives an introductory overview of SNNs and then reviews some prominent simulation frameworks. The authors also define a simple classification task and compare accuracy and execution time obtained by using the different frameworks. These previous two works consider frameworks regardless of their research orientation, i.e. they consider both neuroscience-oriented

[3] Code available at https://www.github.com/daevem/SpykeTorch-Extended

Table 1. Key elements of the reviewed frameworks. The “A-” stands for adaptive, whereas “H-” stand for heterogeneous.

Framework	**Nengo**	**Lava**	**SNN Toolbox**	**Norse**	**PySNN**	**snnTorch**	**SpikingJelly**	**BindsNet**	**SpykeTorch**
Spiking Neurons	LIF A-LIF IZ	LIF RF* A-LIF* A-RF* A-IZ* $\Sigma-\Delta$	IF	LIF AdEx EIF IZ LSNN	IF LIF A-LIF	LIF Recurrent LIF 2nd Order LIF LSNN	IF LIF pLIF QIF EIF	IF LIF A-LIF IZ SRM	IF LIF** QIF** EIF** AdEx** IZ** H-Neurons**
Learning Rules	Oja BCM BP	SLAYER STDP 3-Factor	Pre-trained	SuperSpike STDP	STDP MSTDP MSTDPET	BPTT RTRL	BP	STDP Hebbian MSTDPET	STDP R-STDP
Conversion from	TF/Keras	PyTorch	TF/Keras PyTorch Caffe Lasagne	-	-	-	PyTorch	PyTorch	-
Destination Backend/Platform	Loihi FPGA SpiNNaker MPI CPU/GPU	Loihi CPU/GPU	SpiNNaker Loihi pyNN Brian2 MegaSim	CPU/GPU	CPU/GPU	CPU/GPU	CPU/GPU	CPU/GPU	CPU/GPU

* Only available in Lava-DL.
** Added in this work.

and data science-oriented frameworks. In this work, we specifically highlight software packages that are data science-oriented and developed in Python or with a Python interface. Furthermore, we also include in our review other different frameworks and highlight some key features and neuron models that they offer for developing SNNs.

3 Software Frameworks

Many of the software libraries for the development of SNNs are oriented toward the needs of the neuroscience and neurobiology fields [25]. Because SNNs process inputs and communicate information in a way similar to the human brain, they are particularly suited for simulations of brain areas activations. Nevertheless, the recent emergence of NM engineering as a field for developing ML algorithms has highlighted the need for suitable frameworks. Consequently, following, we will present some of the most prominent software packages to develop data science-oriented SNNs along with their main features, which are also summarized in Table 1.

3.1 Nengo

Nengo [5] is a Python package for building and deploying neural networks. It is composed of several sub-packages to be used in case of different needs and destination platforms. NengoDL is to be used when aiming to convert a CNN built using TF/Keras into its Nengo spiking version. NengoLoihi allows to deploy NNs natively built in the Nengo Core package onto Loihi chips. Other packages are NengoFPGA, NengoSpiNNaker, NengoOCL and NengoMPI. Nengo builds on top of a theoretical framework called the Neural Engineering Framework

(NEF) [50]. Computations are based on the three principles of the NEF: neural representation, transformation, and neural dynamics. Neurons in Nengo are organized in Ensembles, and different types of neuron models are available, among which the Leaky Integrate-and-Fire (LIF) [33], and Izhikevich's (IZ)[30] models. Connections between ensembles are designed to allow a transformation of the information from one ensemble to another. Training in Nengo is possible with the Oja [40], BCM [7] and backpropagation (BP) learning rules. Using Nengo as a tool for the development of SNNs has the main advantage of having the possibility to target a wide variety of backends and to convert conventional DNNs into a spiking equivalent [25]. Nengo also allows for a certain degree of customization of the components; however, it remains very oriented towards the NEF structure.

3.2 SNN Toolbox

SNN Toolbox [46] provides a set of tools to perform automated conversion from conventional Artificial Neural Network (ANN) models into SNNs. Conversion is possible from three different DL frameworks, namely TF/Keras, PyTorch, Caffe and Lasagne [16]. The framework supports conversion to models for PyNN [15], Brian2 [51], MegaSim [35], SpiNNaker [24], and Loihi [14] where the SNN can be simulated or deployed. However, depending on the components used in the original ANN, some of the target platforms might not be available. During the conversion phase, Integrate-and-Fire (IF) neurons are used for a one-to-one substitution. These are then tuned so that their mean firing rate approximates the activation of the corresponding neuron in the original ANN. Neural networks must be pre-trained in their original framework. Tuning conversion parameters and performing inference is possible either through the command line or through a simple GUI.

3.3 Lava

Lava [1] is a relatively recent framework built by Intel's Neuromorphic Computing Lab (NCL). The framework results from an evolution from the Nx SDK software for Loihi chips, but aims to target other hardware platforms as well. Lava is composed of 4 main packages, namely Lava (core), Lava-DL, Lava Dynamic Neural Fields (DNF) and Lava Optimization. The current state of the platform includes the development of deep SNNs trained with SLAYER [49], and of SNNs converted from PyTorch. On-chip training through SLAYER is currently not available. Instead, models need to be trained off-chip, and weights must be exported to be used within the Lava core package. Within Lava-DL, a number of neuron models are defined, such as the LIF, Resonate-and-Fire (RF) [31], RF Izhikevich, Adaptive LIF [26], Adaptive RF, and Sigma-Delta [12] modulation models. The core package currently supports LIF and Sigma-Delta modulation neurons. Recent developments in the framework have seen the implementation of on-chip learning functionalities through STDP and customized 3-factor learning rules.

3.4 PyTorch-based Frameworks

Norse

Norse [44] is a relatively recent PyTorch-based framework. It was developed with the aim of easing the construction of SNNs for ML solutions. This framework offers a wide range of neuron models, such as the LIF, LIF variants and extensions, and Izhikevich's model. It also provides a LSNN [6], a spiking version of the LSTM (Long Short-Term Memory) [28]. Norse has a functional programming style. Neurons are mainly implemented as functions and do not hold an internal state. Instead, the previous state of the neuron needs to be provided as an argument at each iteration. The framework mainly allows for two types of learning: STDP [37], and SuperSpike [53]. Therefore, both local unsupervised learning and surrogate gradient learning are possible. Overall, Norse provides a good degree of flexibility and allows leveraging all of the features of PyTorch, such as GPU acceleration.

PySNN

PySNN [10] is another framework based on PyTorch aimed at developing ML algorithms. Similarly to Nengo, connections between two neurons are modelled as separate objects that have properties and can affect the transmission of a signal. For instance, they can explicitly account for connection delays. Neuron models in PySNN embed the concept of spike trace, which can be used for learning purposes. Some available neuron models are the IF, LIF and ALIF. Concerning the learning rules, it is possible to use either STDP or MSTDPET (Modulated STDP with Eligibility Traces) [20]. The framework also provides some useful utilities to load some NM datasets. A downside of using PySNN is that the documentation is not complete.

SnnTorch

SnnTorch [18] also bases its architecture on PyTorch. Connectivity between layers is enabled by leveraging PyTorch standard layers. Spiking neurons are thought to be used as intermediate layers between these. Spiking neurons are modelled as classes that hold their own internal state. Available models include LIF-based models, second-order LIF models, recurrent LIF models, and LSTM memory cells. Learning in snnTorch takes place with BP Through Time (BPTT) using surrogate gradient functions to calculate the gradient of the spiking neurons. The framework also offers the possibility to use a Real-Time Recurrent Learning (RTRL) rule, which applies weight updates at each time step, rather than at the end of a sequence of inputs. The network output can be interpreted using both a rate-based approach and a time-to-first-spike (TTFS) approach. Finally, snnTorch provides access to the N-MNIST [42], DVS Gestures [3], and the Spiking Heidelberg Digits [13] datasets, and includes useful network activity visualization tools.

SpikingJelly

SpikingJelly [19] is a framework using PyTorch as a backend and adopting its coding style throughout. It provides implementations of IF, LIF, parametric LIF (pLIF), Quadratic IF (QIF), and Exponential IF neuron [21] models. The firing of neurons in SpikingJelly is approximated by a surrogate function (such as the sigmoid) that allows differentiation. The framework provides several utilities to read NM and non-NM datasets. Concerning the NM datasets, it is possible to both read them with a fixed integration time-window and with a fixed number of frames. Among the available datasets, there are the CIFAR10-DVS [34] dataset, the DVS Gestures dataset, the N-Caltech101 [42] dataset, and the N-MNIST dataset. Finally, SpikingJelly also provides functionality for ANN to SNN conversion from PyTorch.

BindsNet

BindsNet [27] is a library for the development of biologically inspired SNNs. Despite having PyTorch as a backend, the coding style differs slightly. Execution is implemented by running the network for a certain amount of time on some input rather than explicitly looping through the dataset. BindsNet supports several types of neuron models: IF, LIF, LIF with adaptive thresholds, Izhikevich's, and Spike Response Model (SRM)-based [26] models. Connections are modelled explicitly and link one node of the network with another. Recurrent connections are also possible. The provided learning rules are biologically inspired and can be either two-factor (STDP or Hebbian) or three-factor (MSTDPET); hence no BP-based learning rule is proposed. Through sub-classing, it is possible to customize neurons, input encoding and learning rules. The framework also provides utility tools to load datasets, such as the spoken MNIST, and DAVIS [8] camera-based datasets. Finally, BindsNet includes a conversion system to convert neural networks developed in PyTorch into SNNs.

SpykeTorch

SpykeTorch is PyTorch-based library for building SNNs with at most one spike per neuron. This means that for each sequence of inputs, each neuron is allowed to fire only once. Because of this, tensor operations can be easily used to compute neuron activations. Because NM data includes the concept of time, what is normally treated as the batch dimension in PyTorch, it is interpreted as the time dimension in SpykeTorch. The framework is built to support STDP and Reward-modulated STDP (R-STDP) with a Winner Takes All (WTA) paradigm, and using convolutions as a connection scheme. The only available neuron model is the IF, which is provided as a function. Finally, the framework provides functionalities to encode non-NM input through difference of Gaussians and intensity to latency transforms, as well as some inhibition functions.

4 SpykeTorch Spiking Neurons

For the purpose of developing NM-ML algorithms based on STDP, SpykeTorch allows a high degree of customization and flexibility to the user. However, as men-

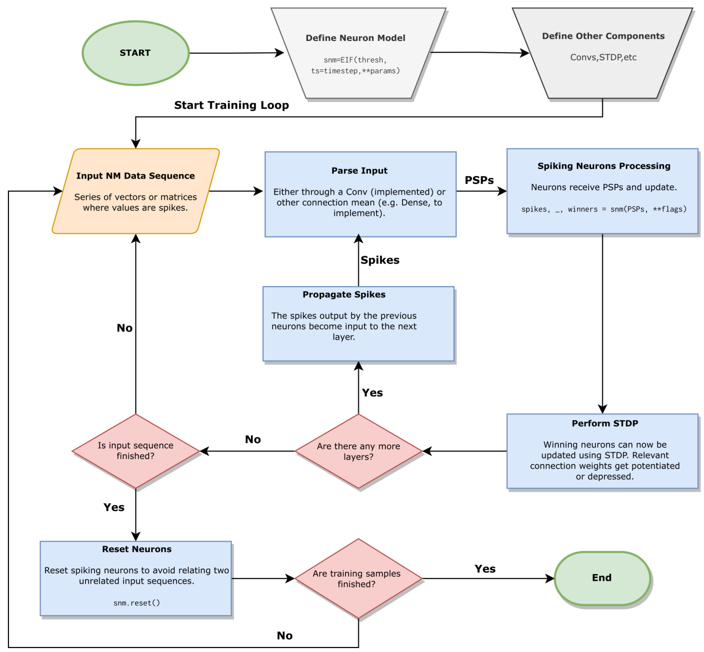

Fig. 1. Example flowchart for SpykeTorch-Extended. After the definition of the components of the SNN, each data sample is required to be decomposed into its forming time steps before being processed by the SNN. This ensures that learnt parameters will influence the result of the next iteration.

tioned in 3.4, the framework originally provides a single spiking neuron model, the IF. This does not have a voltage leakage factor, which means that its internal state can only increase until it is reset. In order to augment the usage potential of SpykeTorch, we expand the library by implementing a new set of spiking neuron models, for a total of 8 new models, as show in Table 2. By introducing more complex neuron models, the original workflow and implementation patterns adopted in the original framework cannot be easily utilized. Therefore, the following are some details about the differences introduced to accommodate such neuron models in the library. We refer to the framework resulting from our changes as SpykeTorch-Extended.

4.1 Spiking Neurons Implementation Details

In our implementation of spiking neurons, we consider a subset from the phenomenological family of neuron models due to their computational efficiency [36]. This includes: Leaky IF (LIF) [33], Exponential IF (EIF)[21], Quadratic IF (QIF)[17], Adaptive Exponential IF (AdEx) [9], Izhikevich's [30], Heterogeneous Neurons.

The LIF model is a single-variable neuron model and is the most widely used for the development of NM-ML systems [36,52,39,29,23]; the EIF and QIF models are other single-variable models that include different types of complexities in their equation, and are also the base for more complex models, the AdEx and Izhikevich's respectively; the AdEx and Izhikevich's models are two-variable neuron models that have also been widely studied and employed in the literature [47,4,11].

Due to the greater complexity of the newly introduced neurons, we deviate from the original implementation and adopt an object-oriented approach for the neurons. This allows them to retain an internal state and other properties. Nevertheless, to maintain compatibility, neuron objects are callable and share the same output format as in the original implementation. Furthermore, we do not restrict neurons to firing only once per input sequence. This only depends on the choice of parameters for a given neuron, such as the refractory period. Another difference with the previous implementation is that the neurons are expected to receive events one time-step at a time. While this introduces a overhead on the computational time, it allows to simulate real-time processing, and also ensures the decay of the membrane potential and that weight updates due to STDP affect every subsequent moment in time, thus making the system more realistic. A neuron layer in SpykeTorch-Extended is characterized by at least the set of parameters of a LIF neuron; however, more complex neuron models will require more parameters. A layer of neurons in this system can be better depicted as a set of neuronal populations. The number and size of the population reflect that of the input that is processed by the layer. Thus, a single population is intended as the group of neurons corresponding to one of the feature maps produced by a convolutional layer.

As a result of the changes above, the standard workflow in SpykeTorch-Extended requires some adjustments with respect to the original version. In Figure 1, we report an example flowchart of how a pipeline using the new neuron models could look like. As the flowchart highlights, each input is expected to be unravelled into all the time steps it is composed of and, for each time step, all the events that took place in such a time span are to be fed forward to the SNN.

4.2 Heterogeneous Neuron Classes

The implemented neuron classes create a layer of spiking neurons that share the same hyper-parameters. We refer to this as being a homogeneous layer of neurons because they all react in the same way to the same sequence of inputs. However, it might be useful to have neurons reacting differently to one input, since this could mean being able to learn different kinds of temporal patterns within the same layer. Because of this, we further implement heterogeneous neuron classes for the `LIF`, `EIF`, and `QIF` classes. Specifically, they provide a set of τ_{rc} values that are uniformly distributed within a range specified by the user through the parameter `tau_range`. We limited the current implementation to a uniform distribution for simplicity, and limit the heterogeneity to the τ_{rc} parameter since this directly influences the time scale to which the neuron is sensitive. Nevertheless, future developments will consider other types of distributions and parameters.

Table 2. Summary of newly added spiking neurons to SpykeTorch. All the neurons share a base set of parameters with the LIF, but they may require more depending on the neuron type, which are briefly reported in the short description.

Neurons	Short Description
LIF [33]	Uses the integral solution to the differential equation in [26].
EIF [21]	Single-variable model with an exponential dependency. Has parameters `delta_t` for the sharpness of the curve, and `theta_rh` as a cut-off threshold for the upswing of the curve [26].
QIF [17]	Single-variable model with a quadratic dependency. Has parameters `a` for the steepness of the quadratic curve, and `u_c` as the negative-to-positive updates crossing point of the membrane potential [26].
AdEx [9]	Two-variables model similar to the EIF, but with an adaptation variable. It adds parameters `a` and `b`, respectively for adaptation-potential coupling and adaptation increase upon spike emission.
IZ [30]	Two-variables model similar to the QIF, but with an adaptation variable. It adds parameters `a` for the time scale of the adaptation variable, `b` for the sub-threshold sensitivity of the adaptation, and `d` for the adaptation increase upon spike emission.
H-Neurons	Heterogeneous versions of LIF, EIF, and QIF neurons with uniformly distributed `tau_rc` parameter.

5 Conclusions

In this work we have presented a review of 9 Python frameworks for the development of spiking neural networks oriented towards data science applications. We have seen that several of them use PyTorch as a base to leverage the GPU acceleration, to exploit the existing functionalities it offers, and to ease the transition for users that come from a conventional DL background. Nevertheless, they all differ slightly in their implementations and in the SNN development tools they offer. Other frameworks like Nengo and Lava do not have such a base, but provide conversion methods to increase usability. This review also highlights how, despite restricting our field of view to data science-oriented libraries, there is a wide variety of frameworks. This is possibly due to growing interest that SNNs have lately received, however this also reflects the lack of an established and widespread framework like in the case of PyTorch or TF/Keras for conventional DL. Finally, we report our extension to a specific framework, SpykeTorch, that includes several new spiking neurons to use for simulations. Our additions require a modification of the original workflow, but enable real-time processing simulation with STDP. By doing this, we hope to promote and speed up future research in this direction, as well as to contribute to the development of richer software frameworks.

References

1. Lava: A software framework for neuromorphic computing. https://github.com/lava-nc/lava (2021)

2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16). pp. 265–283 (2016)
3. Amir, A., Taba, B., Berg, D., Melano, T., McKinstry, J., Di Nolfo, C., Nayak, T., Andreopoulos, A., Garreau, G., Mendoza, M., Kusnitz, J., Debole, M., Esser, S., Delbruck, T., Flickner, M., Modha, D.: A low power, fully event-based gesture recognition system. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 7388–7397 (2017)
4. Barton, A., Volna, E., Kotyrba, M.: The application perspective of izhikevich spiking neural model – the initial experimental study. In: Recent Advances in Soft Computing, pp. 223–232. Springer International Publishing (aug 2018)
5. Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T.C., Rasmussen, D., Choo, X., Voelker, A.R., Eliasmith, C.: Nengo: a python tool for building large-scale functional brain models. Frontiers in Neuroinformatics **7** (2014)
6. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., Maass, W.: Long short-term memory and learning-to-learn in networks of spiking neurons. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
7. Bienenstock, E.L., Cooper, L.N., Munro, P.W.: Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex. Journal of Neuroscience **2**(1), 32–48 (1982)
8. Brandli, C., Berner, R., Yang, M., Liu, S.C., Delbruck, T.: A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor. IEEE Journal of Solid-State Circuits **49**, 2333–2341 (2014)
9. Brette, R., Gerstner, W.: Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. Journal of Neurophysiology **94**(5), 3637–3642 (nov 2005)
10. Büller, B.: Pysnn. https://github.com/BasBuller/PySNN (2019)
11. Chaturvedi, S., Titre, R.N., Sondhiya, N.: Review of handwritten pattern recognition of digits and special characters using feed forward neural network and izhikevich neural model. In: 2014 International Conference on Electronic Systems, Signal Processing and Computing Technologies. IEEE (jan 2014)
12. Cheung, K., Tang, P.: Sigma-delta modulation neural networks. In: IEEE International Conference on Neural Networks. pp. 489–493 vol.1 (1993)
13. Cramer, B., Stradmann, Y., Schemmel, J., Zenke, F.: The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. IEEE Transactions on Neural Networks and Learning Systems pp. 1–14 (2020)
14. Davies, M., Srinivasa, N., Lin, T.H., Chinya, G., Cao, Y., Choday, S.H., Dimou, G., Joshi, P., Imam, N., Jain, S., Liao, Y., Lin, C.K., Lines, A., Liu, R., Mathaikutty, D., McCoy, S., Paul, A., Tse, J., Venkataramanan, G., Weng, Y.H., Wild, A., Yang, Y., Wang, H.: Loihi: A neuromorphic manycore processor with on-chip learning. IEEE Micro **38**(1), 82–99 (2018)
15. Davison, A.P.: PyNN: a common interface for neuronal network simulators. Frontiers in Neuroinformatics **2** (2008)
16. Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S.K., Nouri, D., Maturana, D., Thoma, M., Battenberg, E., Kelly, J., Fauw, J.D., Heilman, M., de Almeida, D.M., McFee, B., Weideman, H., Takács, G., de Rivaz, P., Crall, J., Sanders, G., Rasul, K., Liu, C., French, G., Degrave, J.: Lasagne: First release. (Aug 2015)

17. Ermentrout, G.B., Kopell, N.: Parabolic bursting in an excitable system coupled with a slow oscillation. SIAM Journal on Applied Mathematics **46**(2), 233–253 (apr 1986)
18. Eshraghian, J.K., Ward, M., Neftci, E., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D.S., Lu, W.D.: Training spiking neural networks using lessons from deep learning. arXiv preprint arXiv:2109.12894 (2021)
19. Fang, W., Chen, Y., Ding, J., Chen, D., Yu, Z., Zhou, H., Tian, Y., other contributors: Spikingjelly. https://github.com/fangwei123456/spikingjelly (2020)
20. Florian, R.V.: Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. Neural computation **19**(6), 1468–1502 (2007)
21. Fourcaud-Trocmé, N., Hansel, D., van Vreeswijk, C., Brunel, N.: How spike generation mechanisms determine the neuronal response to fluctuating inputs. The Journal of Neuroscience **23**(37), 11628–11640 (dec 2003)
22. Frady, E.P., Sanborn, S., Shrestha, S.B., Rubin, D.B.D., Orchard, G., Sommer, F.T., Davies, M.: Efficient neuromorphic signal processing with resonator neurons. Journal of Signal Processing Systems **94**(10), 917–927 (may 2022)
23. Friedl, K.E., Voelker, A.R., Peer, A., Eliasmith, C.: Human-inspired neurorobotic system for classifying surface textures by touch. IEEE Robotics and Automation Letters **1**(1), 516–523 (jan 2016)
24. Furber, S., Bogdan, P.: SpiNNaker: A Spiking Neural Network Architecture. now publishers, Inc. (2020)
25. García-Vico, Á.M., Herrera, F.: A preliminary analysis on software frameworks for the development of spiking neural networks. In: Lecture Notes in Computer Science, pp. 564–575. Springer International Publishing (2021)
26. Gerstner, W., Kistler, W.M., Naud, R., Paninski, L.: Neuronal Dynamics. Cambridge University Press (2009)
27. Hazan, H., Saunders, D.J., Khan, H., Patel, D., Sanghavi, D.T., Siegelmann, H.T., Kozma, R.: BindsNET: A machine learning-oriented spiking neural networks library in python. Frontiers in Neuroinformatics **12** (dec 2018)
28. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
29. Hunsberger, E., Eliasmith, C.: Spiking deep networks with lif neurons. arXiv preprint arXiv:1510.0882 (Oct 2015)
30. Izhikevich, E.: Simple model of spiking neurons. IEEE Transactions on Neural Networks **14**(6), 1569–1572 (nov 2003)
31. Izhikevich, E.M.: Resonate-and-fire neurons. Neural Networks **14**(6-7), 883–894 (jul 2001)
32. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe. In: Proceedings of the 22nd ACM international conference on Multimedia. ACM (nov 2014)
33. Lapicque, L.: Recherches quantitatives sur l'excitation electrique des nerfs traitée comme une polarization. Journal de Physiologie et Pathologie General **9**, 620–635 (1907)
34. Li, H., Liu, H., Ji, X., Li, G., Shi, L.: Cifar10-dvs: An event-stream dataset for object classification. Frontiers in Neuroscience **11** (2017)
35. Linares-Barranco, B.: Modular event-driven growing asynchronous simulator (megasim). https://bitbucket.org/bernabelinares/megasim (2018)
36. Manna, D.L., Vicente-Sola, A., Kirkland, P., Bihl, T., Di Caterina, G.: Simple and complex spiking neurons: perspectives and analysis in a simple stdp scenario. Neuromorphic Computing and Engineering (2022)

37. Masquelier, T., Thorpe, S.J.: Unsupervised learning of visual features through spike timing dependent plasticity. PLoS computational biology **3**(2), e31 (2007)
38. Mozafari, M., Ganjtabesh, M., Nowzari-Dalini, A., Masquelier, T.: SpykeTorch: Efficient simulation of convolutional spiking neural networks with at most one spike per neuron. Frontiers in Neuroscience **13** (jul 2019)
39. Mozafari, M., Kheradpisheh, S.R., Masquelier, T., Nowzari-Dalini, A., Ganjtabesh, M.: First-spike-based visual categorization using reward-modulated STDP. IEEE Transactions on Neural Networks and Learning Systems **29**(12), 6178–6190 (dec 2018)
40. Oja, E.: Simplified neuron model as a principal component analyzer. Journal of mathematical biology **15**(3), 267–273 (1982)
41. Orchard, G., Frady, E.P., Rubin, D.B.D., Sanborn, S., Shrestha, S.B., Sommer, F.T., Davies, M.: Efficient neuromorphic signal processing with loihi 2. In: 2021 IEEE Workshop on Signal Processing Systems (SiPS). pp. 254–259. IEEE (2021)
42. Orchard, G., Jayawant, A., Cohen, G.K., Thakor, N.: Converting static image datasets to spiking neuromorphic datasets using saccades. Frontiers in Neuroscience **9** (nov 2015)
43. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)
44. Pehle, C., Pedersen, J.E.: Norse - A deep learning library for spiking neural networks (Jan 2021), documentation: https://norse.ai/docs/
45. Qu, P., Yang, L., Zheng, W., Zhang, Y.: A review of basic software for brain-inspired computing. CCF Transactions on High Performance Computing (mar 2022)
46. Rueckauer, B., Lungu, I.A., Hu, Y., Pfeiffer, M., Liu, S.C.: Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. Frontiers in Neuroscience **11** (dec 2017)
47. Schemmel, J., Billaudelle, S., Dauer, P., Weis, J.: Accelerated analog neuromorphic computing. ArXiv **abs/2003.11996** (2020)
48. Schuman, C.D., Kulkarni, S.R., Parsa, M., Mitchell, J.P., Date, P., Kay, B.: Opportunities for neuromorphic computing algorithms and applications. Nature Computational Science **2**(1), 10–19 (jan 2022)
49. Shrestha, S.B., Orchard, G.: SLAYER: Spike layer error reassignment in time. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 1419–1428. Curran Associates, Inc. (2018)
50. Stewart, T.C.: A technical overview of the neural engineering framework. Tech. rep., Centre for Theoretical Neuroscience (2012)
51. Stimberg, M., Brette, R., Goodman, D.F.: Brian 2, an intuitive and efficient neural simulator. eLife **8** (aug 2019)
52. Vicente-Sola, A., Manna, D.L., Kirkland, P., Caterina, G.D., Bihl, T.: Keys to accurate feature extraction using residual spiking neural networks. Neuromorphic Computing and Engineering **2**(4), 044001 (sep 2022)
53. Zenke, F., Ganguli, S.: Superspike: Supervised learning in multilayer spiking neural networks. Neural computation **30**(6), 1514–1541 (2018)