The Benefits of Self-Supervised Learning for Training Physical Neural Networks

Jeremie Laydevant[*, †] **Aaron Lott**[*] **Davide Venturelli** [*] **Peter L. McMahon** [‡]

Abstract

Physical Neural Networks (PNNs) are energy-efficient alternatives to their digital counterparts. Because they are inherently variable, noisy and hardly differentiable, PNNs require tailored training methods. Additionally, while the properties of PNNs make them good candidates for edge computing, where memory and computational ressources are constrained, most of the training algorithms developed for training PNNs focus on supervised learning, though labeled data could not be accessible on the edge. Here, we propose to use Self-Supervised Learning (SSL) as an ideal framework for training PNNs[4] : 1. SSL globally eliminates the reliance on labeled data and 2. as SSL enforces the network to extract high-level concepts, networks trained with SSL should result in high robustness to noise and device variability. We investigate and show with simulations that the later properties effectively emerge when a network is trained on MNIST in the SSL settings while it does not when trained supervisely. We also explore and show empirically that we can optimize layer-wise SSL objectives rather than a single global one while still achieving the performance of the global optimization on MNIST and CIFAR-10. This could allow local learning without backpropagation at all, especially in the scheme we propose with stochastic optimization. We expect this preliminary work, based on simulations, to pave the way of a robust paradigm for training PNNs and hope to stimulate interest in the community of unconventional computing and beyond.

1 Introduction

Unconventional hardware holds the potential for substantial energy savings and faster inference compared to standard digital hardware to sustain the current boom of deep learning [20]. However, PNNs, being analog, variable, and noisy, demand tailored training methods. Currently, three approaches co-exist: first training a digital model on a digital hardware and then transfer them on analog systems for doing inference[25], in-situ training, and mixed training with hardware-in-the-loop [34]. However, it is important to notice for our study that most prior work focuses on supervised learning[5] where a global supervised objective, often a classification loss function, is optimized through different methods.

In this paper, we question the standard approach of using end-to-end supervised learning for training PNNs, for two main reasons. First, constraining physical systems as mere classifiers may underutilize their potential benefits. PNNs can exhibit diverse, complex behaviors, akin to feature extractors, rather than classifiers, as demonstrated in reservoir computing and in-sensor computing. This subtle nuance could drive significant advancements in the field but also lead to hardware more robust to noise. The second is more a practical concern but not least. While edge computing is a prominent application for PNNs, collecting labeled data on the edge for on-device training may be unrealizable.

[*]USRA Research Institute for Advanced Computer Science, Mountain View, CA 94035, USA
[†]School of Applied and Engineering Physics, Cornell University, Ithaca, NY 14853, USA
[‡]Kavli Institute at Cornell for Nanoscale Science, Cornell University, Ithaca, NY 14853, USA

[4]We focus here on computer vision tasks.

[5]Or unsupervised learning, but with a noticeable cost on the performance of the PNN.

37th First Workshop on Machine Learning with New Compute Paradigms at NeurIPS 2023(MLNPCP 2023).

Self-supervised learning (SSL) is a multifaceted learning framework that is used to train neural networks as feature extractors rather than classifiers. SSL achieves this by solving pretext tasks tailored to downstream tasks. For the specific case of computer vision, SSL relies on strong data augmentation applied on input data to extract semantic concepts rather than high-frequency features that do not generalize well.

In this paper, we explore and report preliminary results on the potential benefits of using the SSL framework in the realm of training and using PNNs. Our contributions are the following:

1. **Question and raise awareness of the unconventional computing community about the majority use of supervised learning methods for training PNNs.**
2. **Investigate and highlight the desirable emergent properties of SSL for trainng PNNs:** On a toy task, training a MLP on MNIST, we show that using SSL instead of supervised learning directly leads to a network that is more robust to device variability, internal noise and sparser architecture. All these features are highly desirable for PNNs as they often suffer from device variability, device noise.
3. **Propose a path toward backprop-free layer-wise training of PNNs:** We investigate a path toward 100% backpropagation-free optimization by proposing to optimize layer-wise SSL objectives. We perform simulations on MNIST and CIFAR-10 and report accuracies obtained with networks trained layer-wise on par with networks trained with a global SSL objective. This highlights the potential of such methods for local-learning. While SSL relies on non-linear projectors and thus still requires local backpropagation, we finally propose a way toward backpropagation-free local optimization.

The results presented in this paper are preliminary but are surprising and convincing enough that we hope it will trigger interest in the unconventinal computing community, as was our first intention writing this paper.

2 Related works

Self-supervised learning (for computer vision): Instead of explicitly training a network in the supervised way on a specific task, self-supervised learning [1] aims at compressing the information contained in input data while focusing on their semantic [8, 28]. A network is "pre-trained" on a pretext task that allows to extract semantic features that are used as the input for downstream tasks: e.g. classification, segmentation, etc. Recent works heavily rely on joint-embedding architectures [5, 37, 2] (Fig. 1.a) where the network have jointly access to two augmented views (that conserve the semantics of the input) of the same input data. The SSL training procedure simultaneously maximizes a similarity objective computed between the resulting two embeddings and optimize a regularization objective that avoids the feature collapse issue. During training, the representation vectors ($h = f(x)$) are fed to a non-linear projector to produce embedding vectors ($z = g(h)$) that has been shown to improve the performance [5], and is discarded after pre-training.

Training Physical Neural Networks: Training PNNs is a wide field that includes noise-aware [36, 13, 19], quantization-aware and hardware-aware training methods [25], methods that embrace the no-ideal and noisy nature of the actual hardware. It also includes methods to train PNNs in-situ with backpropagation [34] while it is generally impossible to do.

Local learning with global feedback: Many works attempt to get rid of backpropagation for training neural networks while still optimizing a global objective by using local updates, whether it is for more bio-plausibility, for computational efficiency or for hardware constraints. [16, 17, 23, 7, 6] propose local learning with random projections of either the target or the error signal to the layer level. [27] leverage the property of convergent energy-based models to backpropagate the error signal through the symmetric bi-directional synapses and compute local updates. [14] propose to project targets or learning signals along with the data in a forward way and optimize the similarity between intermediate representation vectors and the targets at each layer.

Greedy local learning without global feedback: Other works on local learning propose to get rid of the feedback signal and propose different greedy layer-wise training algorithms. Early works [3, 32] were motivated for such greedy layer-wise trainings methods when no good initialization schemes for the parameters were known which impeded end-to-end training with backpropagation. Recently there is a renew interest for greedy layer(block)-wise training methods for two reasons. Training very large models on multiple devices requires to synchronize the different devices for

the forward and the backward passes when using end-to-end backpropagation. Local learning [24, 9, 35] allows desynchronization of the different forward and backward passes on different devices hence faster training. But greedy layer-wise training methods are also very appealing for training unconventional hardware with simple local learning rules [11, 14]. In a very recent work [22], the authors propose to use Forward-Forward-like learning rule to optimize digital linear layers placed in-between fixed-non-linear physical transformations which allows them do deep physical reservoirs computing by optimizing intermediate objectives.

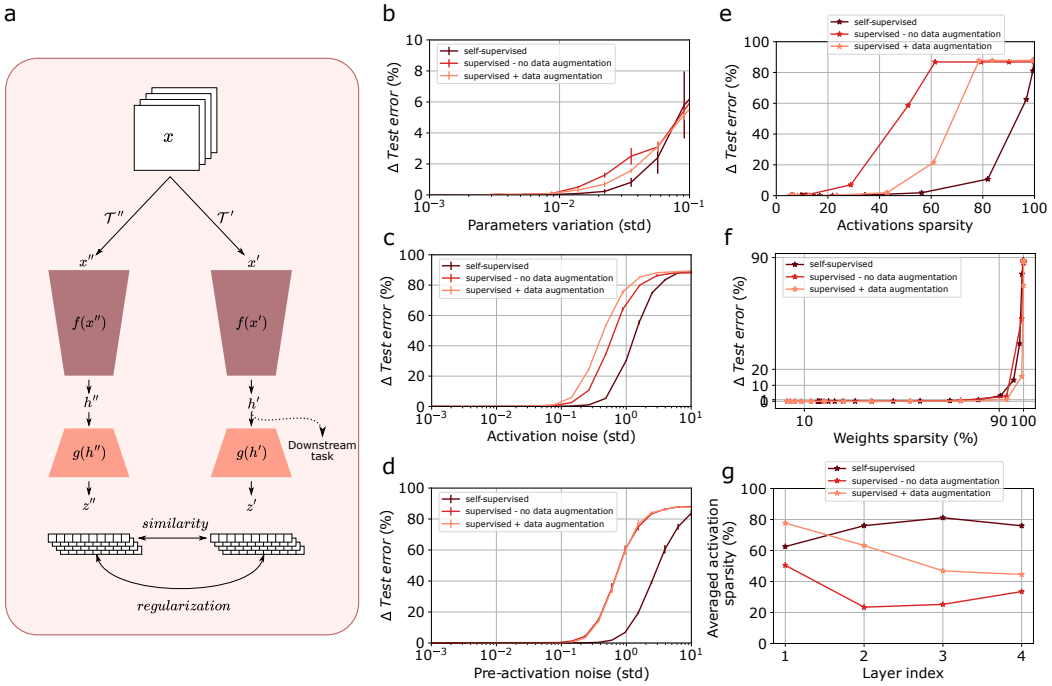

Figure 1: Self-supervised learning paradigm: a. Joint-embeddings non-contrastive self-supervised learning framework. b-e: We trained three MLPs on MNIST with different frameworks: 1. supervised learning without data augmentations, 2. supervised learning with data augmentations and 3. self-supervised learning (VicREG [2] objective) with heavy data augmentations. We evaluate different metrics on the trained networks and report the test error evaluated on the test set of MNIST. All noise injected here is centered gaussian noise with varying standard deviation. b. Robustness to parameters deviation from ideality after training. c. Robustness to activation noise after training. d. Robustness to pre-activation noise after training. e. Robustness of the networks to mean activation magnitude-based pruning. f. Robustness of the networks to weights magnitude-based pruning. g. Averaged activation sparsity of the networks.

3 Self-supervised learning for PNNs

Self-supervised learning focus on learning semantic information rather than high-frequency features which a network trained in the supervised settings could overfit on. Hence, a network trained in the SSL settings is more robust to some input perturbations [10] but we can also expect the same network trained with SSL to be more robust to internal perturbations - either on the parameters or the activations: SSL could serve as an implicit noise-aware training paradigm. Additionally, because of the redundancy reduction objectives optimized in methods such as Barlow Twins [37] or VicREG [2], we can expect both the final representation vector and the internal state of the network to be sparse. All these potential features are highly sought after for training (and deploying for inference) PNNs as they imply less parameters, less neurons (so less area and less energy consumption) and robustness to device variability and noise, which are common feautre in today's PNNs. In this section we investigate these features and show the promising use of SSL for training PNNs.

Experimental setup: We focus our investigation on a toy task: training a 4-layers Multilayer Perceptron (MLP) on MNIST, each layer has 1000 neurons and we use ReLU as the activation function. We train it with 3 different methods:

1. End-to-end supervised backpropagation - we minimize the Cross-entropy loss between the output on the network and the target of the input. We do not use data augmentation techniques for this experiment.
2. End-to-end supervised backpropagation with data augmentation - same as above but with data augmentation techniques for this experiment (see Supplementary for details).
3. End-to-end self-supervised learning with backpropagation - we optimize the VicREG objective: we project the activations of the last layer to an embedding space with a 3 layers MLP (256 neurons per layer, all layers except the last one use ReLU) (see Fig. 1.a for a schematic of the architecture). We extensively use data augmentation techniques (denoted as $\mathcal{T}$ in Fig. 1.a) as required for SSL.

For all the experiments, we proceed as follow: we first train the networks within their specific training framework (SSL or supervised learning). Once the networks are trained, we discard the last classifier layer of the networks trained in the supervised settings, the last hidden layer acting as the representation vector similarly to the network trained with SSL. This allow to investigate the behavior of each network for feature extraction. Except for computing the intrisinc sparsity of the networks, we supervisely train a new linear classifier on the activations vector of the last hidden layer. We do it three times so we report the mean error and the standard deviation. For evaluating the robustness to noise, we add centered gaussian noise to both the parameters and the neurons. This type of noise is general enough for describing many cases and already used for real-life unconventional devices such as memristors [4], but in future work focused on a specific hardware, we should use the relevant type of noise. While we report the lowest test error on MNIST with the network trained with SSL ($\sim 0.6\%$) compared to the supervised settings without data augmentation ($\sim 1.8\%$) and to the supervised settings with data augmentation ($\sim 0.8\%$), we do not report the actual test error if Fig. 1b-f but rather the absolute variation of the test error compared to that obtained after the initial training of the networks. This allows to compare the trend of each network rather than the absolute performance of each training settings.

Results:

- **Sparsity of the activations:** We compute the average number of non-activated neurons (ie neurons whose activation is 0) in each layer on the entire test set of MNIST. We report the results in Fig. 1.g. The network trained with SSL is much sparser than the networks trained in the supervised settings. This indicates that SSL alone allows to trained sparser networks which results in more hardware efficient models for deployment.
- **Robustness to parameters variations:** We apply a random additive perturbations to the trained weights before training the linear classifiers. This allows to simulate the non-ideal deployment of the trained weights on hardware for performing inference, we do not mimic random noise at each inference. In Fig. 1.b we report the variation in test error for an increasing value of the standard deviation of the non-ideality. While not too significant, we see that the network trained with SSL can tolerate a bit more variability at programming time than the networks trained in the supervised settings. We provide more explanation on this behavior in Appendice A.
- **Robustness to (pre-)activation noise:** At each inference, we apply a random additive perturbations to the pre-activations or activations of the neurons (except in the linear classifiers). This allows to simulate noisy parameters or neurons during inference. In Fig. 1.c-d we report the variation in test error for an increasing value of the standard deviation of the noise. In both cases, the network trained with SSL can handle a significant higher level of noise compared to the networks trained in the supervised settings. SSL allows the network to be much more tolerant to the internal random noise and hence should be a robust way to train a PNNs which obviously exhibits internal random noise (that obviously differs depending on the physics of the hardware [19]).
- **Robustness to neurons pruning:** We prune the neurons based on their average activation computed on the whole training set compared to a pruning threshold. We show in Fig. 1.e the significant advantage of using SSL for training a network compared to supervisely train it as the network trained with SSL can tolerate much more neuron-sparse graph than the supervisely trained-counterparts. This finding is in line with the first observation in Fig. 1.g which shows the sparsity of the activations of the different networks.
- **Robustness to parameters pruning:** We prune the parameters with a vanilla absolute magnitude-based pruning method for different magnitude thresholds, that we translate into

a sparsity metric. In Fig. Fig. 1.f. we report the variation of test for different weight sparsity levels. Here, we do not see any benefits of using SSL to trained a network compared to supervisely learning.

Overall, we have shown that using SSL for training a neural networks naturaly lead to features that are highly desirable for PNNs such a robustness to parameters variability, (pre-)activation noise and sparsity of the graph. While we have only worked on a toy vision task we expect those results to hold for deeper networks trained on harder tasks. Not only SSL can improve the training of standard PNNs with respect to the hardware caracteristics, but it could also be proved to improve "smart-sensors" [33]which are essentially designed to extract the most important features of the input. SSL can already be integrated with the Physics-Aware-Training framework [34] and used to train actual PNNs. However, end-to-end backpropagation, as used in this Section, could be not realizable on the edge because of the memory and computational ressources required. In the next section we investigate alternatives to end-to-end optimization with backpropagation and alternatives to backpropagation altogether.

4 Toward local learning with Layer-wise SSL

Training neural networks with local objectives instead of a global one have always garnered attention in the ML community. While initially motivated for initializing the parameters of the network when end-to-end backpropagation from scratch was failing [32, 3], the optimization of local objectives is recently experiencing a revival of interest for two distincts reasons: training large-scale models on distributed devices without the requirement on the synchronization of the forward and backward passes on the different devices that dramatically slows down the training process [15, 9], and for a reason that interest us most: training Physical Neural Networks that we cannot directly trained with end-to-end backpropagation. Thus, recent proposals for such "Forward-forward" training algorithms [11], ie greedy layer-wise training schemes are very appealing. Yet, most of these algorithms optimize empirical objectives [11, 6, 14] and except the later, do not scale well to tasks harder than MNIST. Meanwhile, [31, 18, 24, 35, 12, 26, 29] realize greedy layer or block-wise by optimizing well-defined objectives based or inspired by representation learning. Recent works in the SSL field [37, 2] shed new insights on these "forward-forward" algorithms: they could simply be special derivations of the general self-supervised learning framework. Needless to say that the positive vs. negative data used in the Forward-forward paper [11] is a contrastive self-supervised algorithm. But, more insightful, the proposed maximization (or minimization) of the goodness function, which the sum of the square activity of each layer, is an equivalent objective of the self-covariance objective optimized in VicREG [2]. SoftHebb [12] also optimizes such a kind of objective via the soft-winner-take-all applied as the activation of each layer. SignalProp can also be regarded as a special case of a joint-embedding SSL only in that case, one representation vector is fixed and similar for data of the same class which eventually enforce each layer to cluster data from the class together. Finally, [24] optimizes a layer-wise similarity objective between input of the class, which allows to cluster similar input in each layer while an additional parallel layer-wise supervised classification loss is also optimized. The similarity objective between inputs of the same classe enforces invariance as the corresponding term in self-supervised methods.

Those insights motivated us to further explore the implementation of greedy layer-wise training through the optimization of local layer-wise non-contrastive SSL objectives, which objectives (variance, invariance and covariance) could be more directly translated to actual physical properties than contrastive methods. This idea follows that of [29] that proposed earlier this year to optimize blocks of a ResNet with block-wise SSL objectives rather than an global optimization. Importantly, the authors achieve near state-of-the-art accuracy on ImageNet without a global optimization. In this Section we present our ongoing efforts to implement layer-wise SSL training and preliminary results.

Experimental setup: All the simulations we have performed follow the following protocol (see Fig. 2.a): we pre-train a network (MLP or CNN) by sequentially optimizing layer-wise VicREG objectives [2], starting from the first layer to the last one. The activations vectors of each layer are fed to non-linear projectors whose outputs are used to compute the layer-wise SSL objectives. Between each layer of the network we detach the computational graph so the gradient is local-only. In parallel to the pre-training optimization, we train linear classifiers on top of each layer to assess the performance of each layer. We provide the simulations details in Appendix C.

	MNIST			CIFAR-10
	MLP (4 fc)	CNN (2c)	MLP - SPSA (1fc)	CNN (4c)
Test accuracy (%)	99.25	99.36	96.6	77 (76)

Table 1: Test accuracy on MNIST and CIFAR-10 for MLPs and CNNs trained with a layer-wise SSL objective. *fc* stands for the number of fully-connected layers in the MLPs models and *c* denotes the number of convolutional layers in the CNN models. In parenthesis we report the accuracy we got with a end-to-end optimization with backpropagation of a global SSL objective).

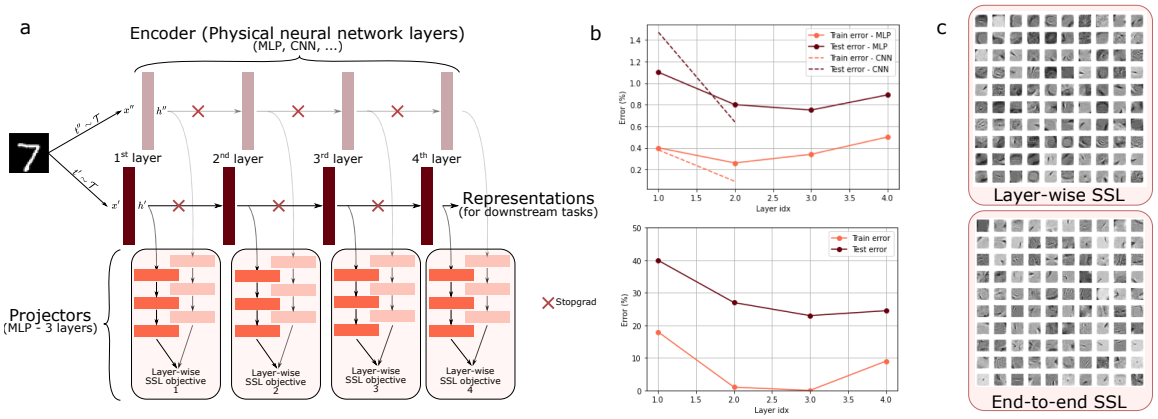

Figure 2: **Principle and implementation of Layer-wise SSL:** a. We compute a SSL objective for each layer of the network to be trained. The brown layers denote potentially PNN layers while the orange ones denote the layers of the non-linear projectors (still digital layers). b. Train and test error (%) as a function of the depth of the different neural networks (MLPS and CNNs) trained on MNIST (top) and CIFAR-10 (bottom). c. Trained weights of the first layer of two networks trained with VicREG. Top is with layer-wise optimization, bottom is end-to-end optimization.

Results: The global accuracy obtained on each task is reported in Table 1 and the accuracy-layer dependance for each task is reported in Fig. 1.b-c. **MLP - MNIST:** The network achieves a performance close to that of the end-to-end SSL optimization (reported in Section 3) on MNIST. We also see that stacking layers improve the test accuracy which indicates that even a greedy layer-wise optimization allows to train a relatively deep (4 layers) MLP on MNIST. Interestingly, the weights learnt with layer-wise SSL are qualitatively similar to the weights learnt with end-to-end global SSL (Fig. 2.c). **CNN - MNIST:** Again we achieved a very high accuracy on MNIST and show that stacking 2 layers perform better than a single one. **CNN - CIFAR-10:** Again, we achieve a test accuracy similar to that achieved with end-to-end global SSL optimization (reported between the parentesis in Table 1) on a 4-convolutional layers CNN which indicates the promising use of layer-wise SSL trainings of deep neural networks. **MLP - MNIST - SPSA (no backprop)** While the previous results still rely on local backpropagation as we used trained non-linear projectors, here we investigate the use of stochastic optimization for the parameters of layer. We use fixed non-linear projectors and used Simultaneaous Perturbations Stochastic Optimization (SPSA) [30] to update the weights of each layer. However, optimizing a multi-objectives SSL objective such as VicREG does not work well with SPSA as there is always a dominant objective which we optimize while do not the others. So we optimized the Barlow Twins [37] with SPSA for those simulations. We achieved a relatively higher test error in that case with a single layer MLP trained on MNIST. Nevertheless, the error is lower than if we don't train the layer which indicates that such local optimization can work.

Going forward: A possible way to improve the greedy layer-wise SSL training would be to train interlocked blocks as done in [9, 35]. We can also implement Forward gradients-based optimization as shown in [26] with contrastive SSL to improve on SPSA. Finally, deriving SSL objectives given the actual physics of the PNNs to be trained, a la "physics informed neural networks" [21], should be proven to be more powerful and practical than general-purpose SSL objectives for training PNNs.

5 Conclusion and looking forward

With this short paper we emphasized the multi-faceted assets that self-supervised learning has for training PNNs with preliminary simulations. It turns out that using SSL leads to sparser networks more robust to noise than networks trained with supervised learning although no sparsity objective or noise-aware training techniques are explicitely employed. Furthermore, SSL allows networks to be trained locally with layer-wise well-defined objectives and could allow for greedily trained deep PNNs. The next step is to actually train PNNs with SSL by including more physics knowledge in the training algorithm.

6 Aknowledgements

This work has been supported by the National Science Foundation (award no. CCF-1918549: P.L.M., D.V., A.L., J.L.; award no. CBET-2123862: P.L.M.) and a David and Lucile Packard Foundation Fellowship (P.L.M.). We thank the reviewers for constructive feedback that improved the quality of this paper.

References

[1] Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, Avi Schwarzschild, Andrew Gordon Wilson, Jonas Geiping, Quentin Garrido, Pierre Fernandez, Amir Bar, Hamed Pirsiavash, Yann LeCun, and Micah Goldblum. A cookbook of self-supervised learning, 2023.

[2] Adrien Bardes, Jean Ponce, and Yann LeCun. VICReg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations*, 2022.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, NIPS'06, page 153–160, Cambridge, MA, USA, 2006. MIT Press.

[4] Djohan Bonnet, Tifenn Hirtzlin, Atreya Majumdar, Thomas Dalgaty, Eduardo Esmanhotto, Valentina Meli, Niccolò Castellani, Simon Martin, Jean-Francois Nodin, Guillaume Bourgeois, Jean-Michel Portal, Damien Querlioz, and Elisa Vianello. Bringing uncertainty quantification to the extreme-edge with memristor-based bayesian neural networks, 2023.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[6] Giorgia Dellaferrera and Gabriel Kreiman. Error-driven input modulation: Solving the credit assignment problem without a backward pass. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 4937–4955. PMLR, 17–23 Jul 2022.

[7] Maxence M Ernoult, Fabrice Normandin, Abhinav Moudgil, Sean Spinney, Eugene Belilovsky, Irina Rish, Blake Richards, and Yoshua Bengio. Towards scaling difference target propagation by learning backprop targets. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5968–5987. PMLR, 17–23 Jul 2022.

[8] Marco Federici, Anjan Dutta, Patrick Forré, Nate Kushman, and Zeynep Akata. Learning robust representations via multi-view information bottleneck. In *International Conference on Learning Representations*, 2020.

[9] Aidan N Gomez, Oscar Key, Kuba Perlin, Stephen Gou, Nick Frosst, Jeff Dean, and Yarin Gal. Interlocking backpropagation: Improving depthwise model-parallelism. *The Journal of Machine Learning Research*, 23(1):7714–7741, 2022.

[10] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. *Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[11] Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

[12] Adrien Journé, Hector Garcia Rodriguez, Qinghai Guo, and Timoleon Moraitis. Hebbian deep learning without feedback. In *The Eleventh International Conference on Learning Representations*, 2023.

[13] Sanjay Kariyappa, Hsinyu Tsai, Katie Spoon, Stefano Ambrogio, Pritish Narayanan, Charles Mackin, An Chen, Moinuddin Qureshi, and Geoffrey W. Burr. Noise-resilient dnn: Tolerating noise in pcm-based ai accelerators via noise-aware training. *IEEE Transactions on Electron Devices*, 68(9):4356–4362, 2021.

[14] Adam Kohan, Edward A Rietman, and Hava T Siegelmann. Signal propagation: The framework for learning and inference in a forward pass. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[15] Michael Laskin, Luke Metz, Seth Nabarro, Mark Saroufim, Badreddine Noune, Carlo Luschi, Jascha Sohl-Dickstein, and Pieter Abbeel. Parallel training of deep networks with local updates, 2021.

[16] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 498–515. Springer International Publishing, 2015.

[17] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1), November 2016.

[18] Sindy Löwe, Peter O'Connor, and Bastiaan Veeling. Putting an end to end-to-end: Gradient-isolated learning of representations. In *Advances in Neural Information Processing Systems*, pages 3039–3051, 2019.

[19] Shi-Yuan Ma, Tianyu Wang, Jérémie Laydevant, Logan G. Wright, and Peter L. McMahon. Quantum-noise-limited optical neural networks operating at a few quanta per activation, 2023.

[20] Danijela Marković, Alice Mizrahi, Damien Querlioz, and Julie Grollier. Physics for neuromorphic computing. *Nature Reviews Physics*, 2(9):499–510, July 2020.

[21] Grégoire Mialon, Quentin Garrido, Hannah Lawrence, Danyal Rehman, Yann LeCun, and Bobak Kiani. Self-supervised learning with lie symmetries for partial differential equations. In *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

[22] Ali Momeni, Babak Rahmani, Matthieu Malléjac, Philipp del Hougne, and Romain Fleury. Backpropagation-free training of deep physical neural networks. *Science*, 0(0):eadi8474, 2023.

[23] Arild Nø kland. Direct feedback alignment provides learning in deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[24] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4839–4850. PMLR, 09–15 Jun 2019.

[25] Malte J. Rasch, Charles Mackin, Manuel Le Gallo, An Chen, Andrea Fasoli, Frédéric Odermatt, Ning Li, S. R. Nandakumar, Pritish Narayanan, Hsinyu Tsai, Geoffrey W. Burr, Abu Sebastian, and Vijay Narayanan. Hardware-aware training for large-scale and diverse deep learning inference workloads using in-memory computing-based accelerators. *Nature Communications*, 14(1), August 2023.

[26] Mengye Ren, Simon Kornblith, Renjie Liao, and Geoffrey Hinton. Scaling forward gradient with local losses. In *The Eleventh International Conference on Learning Representations*, 2023.

[27] Benjamin Scellier and Yoshua Bengio. Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. *Frontiers in Computational Neuroscience*, 11, 2017.

[28] Ravid Shwartz-Ziv and Yann LeCun. To compress or not to compress - self-supervised learning and information theory: A review. *CoRR*, abs/2304.09355, 2023.

[29] Shoaib Ahmed Siddiqui, David Krueger, Yann LeCun, and Stéphane Deny. Blockwise self-supervised learning at scale, 2023.

[30] J.C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[31] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019.

[32] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. 11:3371–3408, dec 2010.

[33] Tianyu Wang, Mandar M Sohoni, Logan G Wright, Martin M Stein, Shi-Yuan Ma, Tatsuhiro Onodera, Maxwell G Anderson, and Peter L McMahon. Image sensing with multilayer nonlinear optical neural networks. *Nature Photonics*, 17(5):408–415, 2023.

[34] Logan G Wright, Tatsuhiro Onodera, Martin M Stein, Tianyu Wang, Darren T Schachter, Zoey Hu, and Peter L McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, 2022.

[35] Yuwen Xiong, Mengye Ren, and Raquel Urtasun. Loco: Local contrastive representation learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[36] Xiaoxuan Yang, Changming Wu, Mo Li, and Yiran Chen. Tolerating noise effects in processing-in-memory systems for neural networks: A hardware–software codesign perspective. *Advanced Intelligent Systems*, 4(8), May 2022.

[37] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320. PMLR, 2021.

The appendices contain additional information regarding why SSL allows for more noise-robustness and the simulation details. Additionally, we provide a link to a github repository[6] that contains the code to reproduce the different experiments.

A Mechanisms underlying SSL as an implicit noise-aware training method

In this section we attempt to provide an explanation for why SSL as a learning framework allows for more noise robustness when noise is applied on the parameters (weights) of the model.

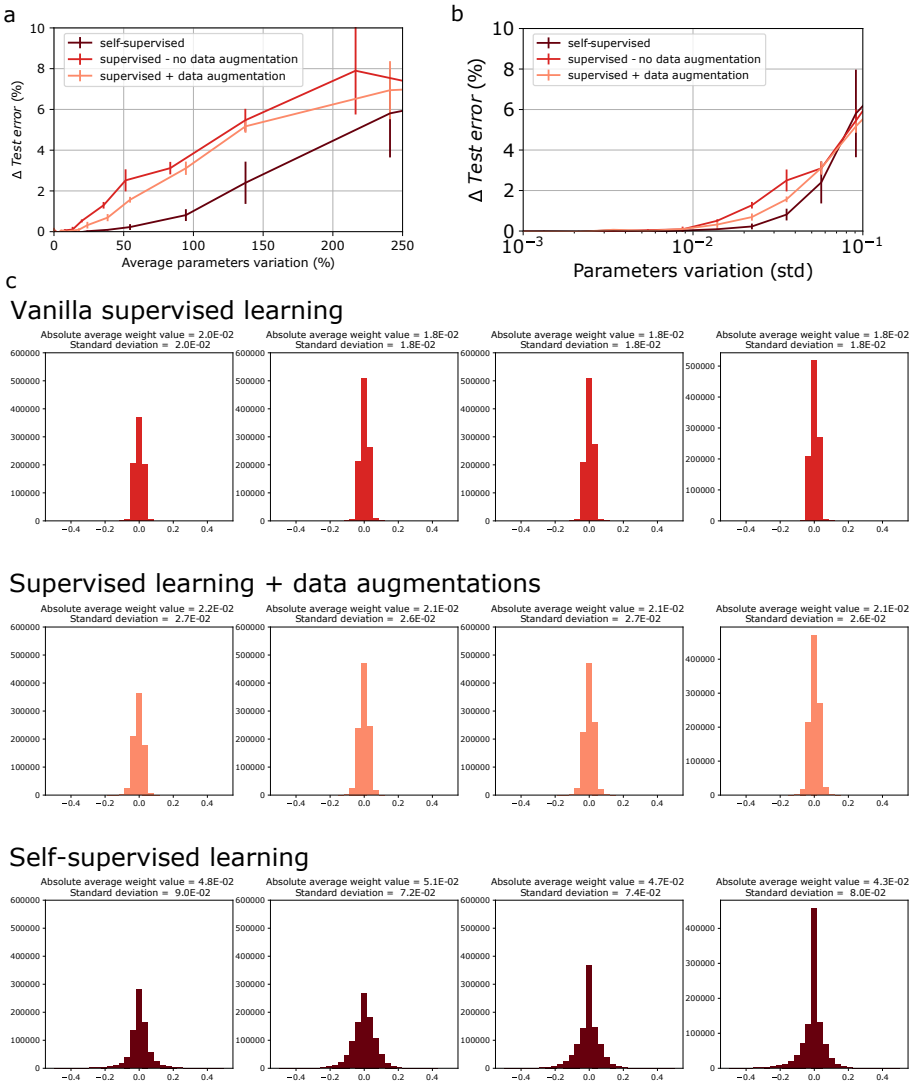

Figure 3: **More insights on noise-robustness:** a. We plot the absolute variation in test error with respect to the average absolute parameters variation for different corresponding noise levels. b. Same as a. (and Fig. 1.b but we directly report the standard deviation of the noise. c. We plot the histograms of the weights of the different layers of the different models trained: in the supervised way, in the supervised way with data augmenetation and with self-supervised learning. We used the same bining for all the histograms to better compare the different distributions.

[6]https://github.com/mcmahon-lab/The-Benefits-of-Self-Supervised-Learning-for-Training-Physical-Neural-Networks

In Fig. 1 (and Fig. 3.b) we plotted as the x-axis the varying standard deviation of the gaussian noise we applied to the parameters but it is true that standard deviation alone is not sufficient to explain why networks trained with SSL seem more robut to noise than network trained in the supervised way. Here (Fig. 3.a) we have converted the noise levels to an averaged relative variation from the original value of the parameters (in %), which gives more insights, especially if one knows the original statistics of the parameters of the different models. In Fig. 3.c we report the statistics of the weights for each model for each layer.

Interstingly, it appears that the network trained with SSL have weights that have a broader distribution which means that most of the parameters have a larger magnitude than the parameters of the models trained in the supervised way. Furthermore, it seems that the width of the parameters distributions grows when we add data augmentations to the training compared to the vanilla supervised case without data augmentations.

It is then natural that the model trained with SSL can handle higher levels of noise because the relative variation of the parameters resulting from a given noise level is lower than for the networks trained in the supervised settings.

B Simulations details: end-to-end training

In this section we present the simulation details for the results presented in Section 3. All the simulations focus on training a Multilayer Perceptron (MLP) which has 4 hidden layers on MNIST, each layer has 1000 neurons and we use ReLU as the non-linear activation function.

B.1 Supervised learning of Multilayer perceptron on MNIST

For the supervised learning settings, we minimize the Cross-entropy loss computed on an additional output layer between the output of the network and the target vector.

For the appropriate simulations, we use the following data augmentations - with the torchvision nomenclature:

- RandomRotation(degrees = 5, fill=0)
- RandomCrop((28,28), padding = 2)
- torchvision.transforms.RandomAffine(degrees=(0, 0), translate=(0.0, 0.0), scale=(0.9, 1.1))

We use ADAM with a learning rate of $1e-4$ with the default β parameters. We use mini-batches of size 256.

We train the network in the supervised settings for 50 epochs, which ensure a 0% train error and $\sim 1.8\%$ test error. The network clearly overfit without data augmentation.

When we use data augmentations, we train the network for 100 epochs. This allow the network to achieve $\sim 0.6\%$ train error and $\sim 0.8\%$ test error. 100 epochs is when the network starts to overfit and the test loss/ error starts to increase again.

B.2 Self-supervised learning of Multilayer perceptron on MNIST

For the self-supervised learning settings, we optimize the VicREG objective [2] with standard hyperparameters ($\alpha_{sim} = \alpha_{std} = 25$ and $\alpha_{covar} = 1$). We use a 3 layers MLP as the non-linear projector. Each layer has 256 neurons. All the layers, except the last one, use ReLU. We compute the VicREG objective given the output layer of the projector. Once the network (the main MLP with 4 layers) is trained, we discard the projector and use the representation vector, which is the output of the last hidden layer of the main network as the input for a downstream taks (classification in our case).

We use the following data augmentations - with the torchvision nomenclature:

- RandomRotation(degrees = 15, fill=0)
- RandomCrop((28,28), padding = 2)

- RandomAffine(degrees=(0, 0), translate=(0.0, 0.0), scale=(0.9, 1.1))
- ColorJitter(brightness=0, contrast = 0.5, saturation=0, hue = 0.5)

For the self-supervised learning step, we use ADAM with a learning rate of $1e-4$ with the default β parameters. We use mini-batches of size 256.

For supervisely train a linear classifier on top, we use ADAM with a learning rate of $1e-3$ with the default β parameters. We use mini-batches of size 64.

We train the network in the self-supervised settings for 1000 epochs, which ensure a 0% train error and $\sim 0.6\%$ test error.

When we use data augmentations, we train the network for 100 epochs. This allow the network to achieve $\sim 0.6\%$ train error and $\sim 0.8\%$ test error. 100 epochs is when the network starts to overfit and the test loss/ error starts to increase again.

C Simulations details: layer-wise training

In this section we present the simulation details for the results presented in Section 4.

C.1 Multilayer perceptron on MNIST

We first train a MLP which has 4 hidden layers on MNIST, each layer has 1000 neurons and we use ReLU as the non-linear activation function.

Here, we optimize layer-wise VicREG objectives [2] with standard hyperparameters ($\alpha_{sim} = \alpha_{std} = 25$ and $\alpha_{covar} = 1$). To that end, we use 4 3-layers MLPs as the non-linear projectors. Each layer has 256 neurons. All the layers, except the last one, use ReLU. We compute the VicREG objectives given the layer layer of the projector. Once layer (the main MLP with 4 layers) is trained, we discard its projectors. We use the representation vector of each layer, as the input for downstream taks (classification in our case). By training linear classifiers on top of each layer, we can know their individual performance. To avoid the gradient to flow from one layer to another we detach the computational graph between each layer of the main network to train in a layer-wise fashion.

We use the following data augmentations - with the torchvision nomenclature:

- RandomRotation(degrees = 15, fill=0)
- RandomCrop((28,28), padding = 2)
- RandomAffine(degrees=(0, 0), translate=(0.0, 0.0), scale=(0.9, 1.1))
- ColorJitter(brightness=0, contrast = 0.5, saturation=0, hue = 0.5)

For the self-supervised learning step, we use ADAM with a learning rate of $1e-4$ with the default β parameters. We use mini-batches of size 256.

For traininng a linear classifier with supervised learning on top, we use ADAM with a learning rate of $1e-3$ with the default β parameters. We use mini-batches of size 64. We train each layer for 1000 epochs.

C.2 Convolutional neural network on MNIST

We train a 2-layers convolutional neural network (CNN), again with layer-wise VicREG objectives [2] with standard hyperparameters ($\alpha_{sim} = \alpha_{std} = 25$ and $\alpha_{covar} = 1$). In order to match the depth-dependance of the different loss observed in networks trained with end-to-end SSL (as also observed inn [29]), we tried to introduce a sigmoidal parametrization for the different VicREG coefficient, but we haven't find yet the ideal set of parameters.

To that end, we use 2 3-layers MLPs as the non-linear projectors. Each layer has 256 neurons. All the layers, except the last one, use ReLU. We compute the VicREG objectives given the layer layer of the projector.

The network has 2 convolutional layers, with respectively 64 and 128 channels. Each layer has a kernel of size 5, a padding of 2 and a stride of 1. For each layer, we apply the convolutional operation first, then the non-linearity, then a batch-normalization and finally we apply a 2x2 max-pooling operations.

The main difference here, is that we need to add an additional layer between the feature map resulting from each convolutional layer and the input of the projectors because their size do not match directly. Similarly to [29], we use a trainable 1x1 convolution with the number of output channels being the size of the input of the projector. Once this 1x1 convolution is applied, we average each channel and feed the resuling vector to the projector.

We use the following data augmentations - with the torchvision nomenclature:

- RandomRotation(degrees = 15, fill=0)
- RandomCrop((28,28), padding = 2)
- RandomAffine(degrees=(0, 0), translate=(0.0, 0.0), scale=(0.9, 1.1))
- ColorJitter(brightness=0, contrast = 0.5, saturation=0, hue = 0.5)

For the self-supervised learning step, we use ADAM with a learning rate of $1e-4$ with the default β parameters. We use mini-batches of size 256.

For training a linear classifier with supervised learning on top of each activation vector (before the 1x1 convolution), we use ADAM with a learning rate of $1e-3$ with the default β parameters. We use mini-batches of size 64. We train each layer for 1000 epochs.

C.3 Convolutional neural network on CIFAR-10

We train a 4-layers convolutional neural network (CNN), again with layer-wise VicREG objectives [2] with standard hyperparameters ($\alpha_{sim} = \alpha_{std} = 25$ and $\alpha_{covar} = 1$). To that end, we use 2 3-layers MLPs as the non-linear projectors. Each layer has 2048 neurons. All the layers, except the last one, use ReLU. We compute the VicREG objectives given the layer layer of the projector.

The network has 4 convolutional layers, with respectively 128, 256, 512 and 1024 channels. Each layer has a kernel of size 5, a padding of 2 and a stride of 1. For each layer, we apply the convolutional operation first, then the non-linearity, then a batch-normalization and finally we apply a 2x2 max-pooling operations.

We also used trainable 1x1 convolutions followed by a channel-wise average operation to project each feature map to the input layer of the projectors.

We use the following data augmentations - with the torchvision nomenclature:

- RandomCrop(size=[32,32], padding = 2, padding_mode='edge')
- RandomHorizontalFlip(p=0.5)
- RandomAffine(degrees=(0, 0), translate=(0.0, 0.0), scale=(1, 1.3))
- ColorJitter(brightness=0.4, contrast=0.4, saturation=0.2, hue=0.5)
- RandomApply([transforms.Grayscale(num_output_channels=3)], p=0.2)
- RandomSolarize(0.0, p=0.1)
- Normalize((0.4914,0.4822,0.4465),(0.247,0.243,0.261))

We also tried asymmetric data augmentation, especially for the Grayscale and the Solarization, as done in the Barlow Twins [37], VicREG [2] and Block-wise SSL [29] but without anny effect on the accuracy. We did not use Gaussian blur, which seem to improve a lot the training because the torchvision implementation is very slow and the trainings were very slow. In future work we will implement this augmentation.

For the self-supervised learning step, we use ADAM with a learning rate of $1e-4$ with the default β parameters. We use mini-batches of size 256.

For training a linear classifier with supervised learning on top of each activation vector (before the 1x1 convolution), we use ADAM with a learning rate of $1e-3$ with the default β parameters. We use mini-batches of size 64. We train each layer for 1000 epochs.