APPENDIX
ADDITIONAL RESULTS

TABLE VI
THE TEST ERROR FOR BP, FA, DFA, AND SP (**BEST** VS BP)

Dataset	Network		BP	FA	DFA	SP
MNIST	FC	2x800	$\underline{1.60} \pm 0.06$	$\mathbf{1.64} \pm 0.03$	1.74 ± 0.08	1.71 ± 0.03
		3x800	1.75 ± 0.05	$\mathbf{1.66} \pm 0.09$	1.70 ± 0.04	1.70 ± 0.04
		4x800	1.92 ± 0.11	$\mathbf{1.70} \pm 0.04$	1.83 ± 0.07	$\mathbf{1.70} \pm 0.04$
		2x800 DO	$\underline{1.26} \pm 0.03$	1.53 ± 0.03	1.45 ± 0.07	$\mathbf{1.38} \pm 0.03$
CIFAR-10	FC	3x1000 DO	$\underline{42.20} \pm 0.2$	46.90 ± 0.3	42.90 ± 0.2	$\mathbf{42.62} \pm 0.16$
	CONV		$\underline{22.50} \pm 0.4$	27.10 ± 0.8	26.90 ± 0.5	$\mathbf{24.75} \pm 0.40$
CIFAR-100	FC	3x1000 DO	$\underline{69.80} \pm 0.1$	75.30 ± 0.2	73.10 ± 0.1	$\mathbf{70.30} \pm 0.19$
	CONV		$\underline{51.70} \pm 0.2$	60.50 ± 0.3	59.00 ± 0.3	$\mathbf{57.01} \pm 0.42$

We trained several networks using BP, FA, DFA, and SP on the MNIST, CIFAR-10, and and CIFAR-100. We used fully-connected architectures (FC), and a small convolutional architecture (CONV) architecture. Note, feedback alignment based algorithms (FA and DFA) do not scale well; they are combined them with LL, or another learning model, to achieve reasonable performance.

Signal Propagation: A Framework for Learning and Inference In a Forward Pass

Adam Kohan, Edward A. Rietman, Hava T. Siegelmann

***Abstract*—We propose a new learning framework, signal propagation (sigprop), for propagating a learning signal and updating neural network parameters via a forward pass, as an alternative to backpropagation. In sigprop, there is only the forward path for inference and learning. So, there are no structural or computational constraints necessary for learning to take place, beyond the inference model itself, such as feedback connectivity, weight transport, or a backward pass, which exist under backpropagation based approaches. That is, sigprop enables global supervised learning with only a forward path. This is ideal for parallel training of layers or modules. In biology, this explains how neurons without feedback connections can still receive a global learning signal. In hardware, this provides an approach for global supervised learning without backward connectivity. Sigprop by construction has compatibility with models of learning in the brain and in hardware than backpropagation, including alternative approaches relaxing learning constraints. We also demonstrate that sigprop is more efficient in time and memory than they are. To further explain the behavior of sigprop, we provide evidence that sigprop provides useful learning signals in context to backpropagation. To further support relevance to biological and hardware learning, we use sigprop to train continuous time neural networks with Hebbian updates, and train spiking neural networks with only the voltage or with biologically and hardware compatible surrogate functions.**

***Index Terms*—Local Learning, Neural Networks, Parallel Learning, Optimization, Biological Learning, Neuromorphics**

I. INTRODUCTION

THE success of deep learning is attributed to the backpropagation of errors algorithm [1] for training artificial neural networks. However, the constraints necessary for backpropagation to take place are incompatible with learning in the brain and in hardware, are computationally inefficient for memory and time, and bottleneck parallel learning. These learning constraints under backpropagation come from calculating the contribution of each neuron to the network's output error. This calculation during training occurs in two phases. First, the input is fed completely through the network storing the activations of neurons for the next phase and producing an output; this phase is known as the forward pass. Second, the error between the input's target and network's output is fed in reverse order of the forward pass through the network to compute parameter updates using the stored neuron activations; this phase is known as the backward pass.

These two phases of learning have the following learning constraints. The forward pass stores the activation of every neuron for the backward pass, increasing memory overhead. The forward and backward passes need to complete before receiving the next inputs, thereby pausing resources. Network learning parameters can only be updated after and in reverse order of the forward pass, which is sequential and synchronous. The backward pass requires its own feedback connectivity to every neuron, increasing structural complexity. The feedback connectivity needs to have weight symmetry with forward connectivity, known as the weight transport problem. The backward pass uses a different type of computation than the forward pass, adding computational complexity. In total, these constraints prohibit parallelization of computations during learning, increase memory usage, run time, and the number of computations, and bound the network structure.

These learning constraints under backpropagation are difficult to reconcile with learning in the brain [2], [3]. Particularly, the backward pass is considered to be problematic [2]–[6] as (1) the brain does not have the comprehensive feedback connectivity necessary for every neuron (2) neither is neural feedback known to be a distinct type of computation, separate from feedforward activity and (3) the feedback and feedforward connectivity would need to have weight symmetry.

These learning constraints also hinder efficient implementations of backpropagation and error based learning algorithms on hardware [7], [8]: (1) weight symmetry is incompatible with elementary computing units which are not bidirectional, (2) the transportation of non local weight and error information requires special communication channels in hardware, and (3) spiking equations are non-derivable, non-continuous. Hardware implementations of learning algorithms may provide insight into learning in the brain. An efficient, empirically competitive algorithm to backpropagation on hardware will likely parallel learning in the brain.

All of these constraints can be categorized by their overall effect on learning for a network as follows. (a) Backwardpass unlocking would allow for all parameters to be updated in parallel after the forward pass has completed. (b) Forwardpass unlocking would allow for individual parameters to be asynchronously updated once the forward pass has reached them, without waiting for the forward pass to complete. These categories directly reference parallel computation, but also have implications on network structure, memory, and run-time. For example, backwardpass locking implies top-down feedback connectivity. Similar terminology was used in [9], where (a) is backward locking and (b) is update locking. Alternative learning approaches to address backwardpass and forwardpass unlocking have been proposed, refer to Section II and Figure 1, but do not solve all of these constraints and are based on

Adam Kohan, Edward Rietman, and Hava Siegelmann are with the Biologically Inspired Neural and Dynamical Systems Laboratory, College of Information and Computer Sciences, University of Massachusetts Amherst (e-mail: akohan@cs.umass.edu, erietman@cs.umass.edu, hava@cs.umass.edu)

relaxing learning constraints under backpropagation.

We propose a new learning framework, signal propagation (SP or sigprop), for propagating a learning signal and updating neural network parameters via a forward pass. Sigprop has no constraints on learning, beyond the inference model itself, and is completely forwardpass unlocked. At its core, sigprop generates targets from learning signals and then re-uses the forward path to propagate those targets to hidden layers and update parameters. Sigprop has the following desirable features. First, inputs and learning signals use the same forward path, so there are no additional structural or computational requirements for learning, such as feedback connectivity, weight transport, or a backward pass. Second, without a backward pass, the network parameters are updated as soon as they are reached by a forward pass containing the learning signal. Sigprop does not block the next input or store activations. So, sigprop is ideal for parallel training of layers or modules. Third, since the same forwardpass used for inputs is used for updating parameters, there is only one type of computation. Compared with alternative approaches, sigprop addresses all of the above constraints, and does so with a global learning signal.

Our work suggests that learning signals can be fed through the forward path to train neurons. Feedback connectivity is not necessary for learning. In biology, this means that neurons who do not have feedback connections can still receive a global learning signal. In hardware, this means that global learning (e.g supervised or reinforcement) is possible even though there is no backward connectivity.

This paper is organized as follows. In Section II, we detail the improvements on relaxing learning constraints of sigprop over alternative approaches. In Section III, we introduce the signal propagation framework and learning algorithm. In Section IV, we describe experiments evaluating the accuracy, run time, and memory usage of sigprop. We also demonstrate that sigprop can be trained with a sparse learning signal. In Section V, we demonstrate that sigprop provides a useful learning signal that becomes increasingly similar to backpropagation as training progresses. We also demonstrate that sigprop can train continuous time neural networks, and with a Hebbian plasticity mechanism to update parameters in hidden layers, as further support of its relevance to biological learning. In Section VI, we demonstrate that sigprop directly trains Spiking Neural Networks, with or without surrogate functions, as further support of its relevance to hardware learning.

II. Relaxing Constraints on Learning

Signal propagation (sigprop) is a new approach that imposes no learning constraints, beyond the inference model itself, while providing a global learning signal. Alternative approaches, in contrast, are based on relaxing the learning constraints under backpropagation. This is a view by which we can arrive at sigprop: once the learning constraints under backpropagation are done away with, the simplest explanation to provide a global learning is to use the forward path, the path constructing the inference model; that is, project the learning signal through the same path as the inputs. Here, we discuss alternative approaches, compare the variations of constraints they relax, and see the difference of removing constraints entirely, which results in the improvements shown under sigprop. Refer to Fig 1 for a visual comparison.

Feedback Alignment (FA), Fig 1b uses fixed random weights to transport error gradient information back to hidden layers, instead of using symmetric weights [10]. It was shown that the sign concordance between the forward and feedback weights is enough to deliver effective error signals [7], [11], [12]. During learning, the forward weights move to align with the random feedback weights and have approximate symmetry, forming an angle below $90°$. FA addresses the weight transport problem, but remains forwardpass and backwardpass locked. Direct Feedback Alignment (DFA), Fig 1c propagates the error directly to each hidden layer and is additionally backwardpass unlocked [13]. Sigprop improves on DFA and is forwardpass unlocked. DFA performs similarly to backpropagation on CIFAR-10 for small fully-connected networks with dropout, but performs more poorly for convolutional neural networks. Sigprop performs better than DFA and FA for convolutional neural networks.

FA based algorithms also rely on systematic feedback connections to layers and neurons. Though it is possible [6], [10], [12], there is no evidence in the neocortex of the comprehensive level of connectivity necessary for every neuron (or layer) to receive feedback (reciprocal connectivity). With sigprop, we introduce an algorithm capable of explaining how neurons without feedback connections learn. That is, neurons without feedback connectivity receive feedback through their feedforward connectivity.

An alternative approach that minimizes feedback connectivity is Local Learning (LL), Fig 1f. In LL algorithms [14]–[16], layers are trained independently by calculating a separate loss for each layer using an auxiliary classifier per layer. LL algorithms have achieved performance close to backpropagation on CIFAR-10 and is making progress on ImageNet. It trains each layer and auxiliary classifier with backpropagation. At the layer level, it has the weight transport problem and is forwardpass and backwardpass locked. In [14], FA is used to backwardpass unlock the layers. It does not use a global learning signal, but learns greedily. In another approach, Synthetic Gradients (SG), Fig 1g are used to train layers independently [9], [17]. SG algorithms train auxiliary networks to predict the gradient of the backward pass from the input, the synthetic gradient. Similar to LL, SG methods trains the auxiliary networks using backpropagation. Until the auxiliary networks are trained, it has the weight transport problem and is forwardpass and backwardpass locked at the network level. In contrast, sigprop is completely forwardpass unlocked, combines a global learning signal with local learning, is compatible with learning in hardware where there is no backward connectivity, and compatible with models of learning in the brain where comprehensive feedback connectivity is not seen, including projections of the targets to hidden layers.

Forwardpass unlocked algorithms do not necessarily address the limitations in biological and hardware learning models, as they have different types of computations for inference and learning. In sigprop, the approach to having a single type of computation for inference and learning is similar to

across a spectrum of learning constraints, with backpropagation being the most constrained and signal propagation being the least constrained. Signal propagation has better efficiency, compatibility, and performance than more constrained learning algorithms not using backpropagation.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[2] A. H. Marblestone, G. Wayne, and K. P. Kording, "Toward an integration of deep learning and neuroscience," *Frontiers in computational neuroscience*, vol. 10, p. 94, 2016.

[3] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive science*, vol. 11, no. 1, pp. 23–63, 1987.

[4] F. Crick, "The recent excitement about neural networks." *Nature*, vol. 337, no. 6203, pp. 129–132, 1989.

[5] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training deep spiking neural networks using backpropagation," *Frontiers in neuroscience*, vol. 10, p. 508, 2016.

[6] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in computational neuroscience*, vol. 11, p. 24, 2017.

[7] E. O. Neftci, C. Augustine, S. Paul, and G. Detorakis, "Event-driven random back-propagation: Enabling neuromorphic deep learning machines," *Frontiers in neuroscience*, vol. 11, p. 324, 2017.

[8] M. Bouvier, A. Valentian, T. Mesquida, F. Rummens, M. Reyboz, E. Vianello, and E. Beigne, "Spiking neural networks hardware implementations and challenges: A survey," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, pp. 1–35, 2019.

[9] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1627–1635.

[10] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, p. 13276, 2016.

[11] Q. Liao, J. Z. Leibo, and T. A. Poggio, "How important is weight symmetry in backpropagation?" in *AAAI*, 2016, pp. 1837–1844.

[12] J. Guerguiev, T. P. Lillicrap, and B. A. Richards, "Towards deep learning with segregated dendrites," *eLife*, vol. 6, 2017.

[13] A. Nøkland, "Direct feedback alignment provides learning in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 1037–1045.

[14] A. Nøkland and L. H. Eidnes, "Training neural networks with local error signals," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4839–4850.

[15] E. Belilovsky, M. Eickenberg, and E. Oyallon, "Decoupled greedy learning of cnns," in *International Conference on Machine Learning*. PMLR, 2020, pp. 736–745.

[16] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.

[17] W. M. Czarnecki, G. Świrszcz, M. Jaderberg, S. Osindero, O. Vinyals, and K. Kavukcuoglu, "Understanding synthetic gradients and decoupled neural interfaces," in *International Conference on Machine Learning*. PMLR, 2017, pp. 904–912.

[18] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 2015, pp. 498–515.

[19] Y. Bengio, "How auto-encoders could provide credit assignment in deep networks via target propagation," *arXiv preprint arXiv:1407.7906*, 2014.

[20] B. Scellier and Y. Bengio, "Equivalence of equilibrium propagation and recurrent backpropagation," *arXiv preprint arXiv:1711.08416*, 2017.

[21] X. Xie and H. S. Seung, "Equivalence of backpropagation and contrastive hebbian learning in a layered network," *Neural computation*, vol. 15, no. 2, pp. 441–454, 2003.

[22] B. Scellier, A. Goyal, J. Binas, T. Mesnard, and Y. Bengio, "Extending the framework of equilibrium propagation to general dynamics," 2018.

[23] K. Hirasawa, M. Ohbayashi, M. Koga, and M. Harada, "Forward propagation universal learning network," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1. IEEE, 1996, pp. 353–358.

[24] R. J. Williams and D. Zipser, *Gradient-based learning algorithms for recurrent connectionist networks*. Citeseer, 1990.

[25] Y. Ohama, N. Fukumura, and Y. Uno, "A forward-propagation rule for acquiring neural inverse models using a rls algorithm," in *International Conference on Neural Information Processing*. Springer, 2004, pp. 585–591.

[26] ——, "A forward-propagation learning rule for neural inverse models using a method of recursive least squares," *Systems and Computers in Japan*, vol. 36, no. 8, pp. 71–80, 2005.

[27] A. P. Heinz, "Pipelined neural tree learning by error forward-propagation," in *Proceedings of ICNN'95-International Conference on Neural Networks*, vol. 1. IEEE, 1995, pp. 394–397.

[28] Y. Ohama and T. Yoshimura, "A parallel forward-backward propagation learning scheme for auto-encoders," in *International Conference on Neural Information Processing*. Springer, 2017, pp. 126–136.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[30] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1. Citeseer, 2013, p. 3.

[31] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[32] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[33] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[34] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[35] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[36] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the national academy of sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[37] J. K. Eshraghian, M. Ward, E. Neftci, X. Wang, G. Lenz, G. Dwivedi, M. Bennamoun, D. S. Jeong, and W. D. Lu, "Training spiking neural networks using lessons from deep learning," *arXiv preprint arXiv:2109.12894*, 2021.

[38] S. R. Kheradpisheh and T. Masquelier, "Temporal backpropagation for spiking neural networks with one spike per neuron," *International Journal of Neural Systems*, vol. 30, no. 06, p. 2050027, 2020.

[39] J. Backus, "Can programming be liberated from the von neumann style? a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[40] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[41] N. R. Mahapatra and B. Venkatrao, "The processor-memory bottleneck: problems and solutions," *Crossroads*, vol. 5, no. 3es, p. 2, 1999.

[42] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, no. 1, pp. 54–66, 2015.

[43] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International joint conference on neural networks (IJCNN)*. ieee, 2015, pp. 1–8.

[44] B. Rueckauer, I.-A. Lungu, Y. Hu, M. Pfeiffer, and S.-C. Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, p. 682, 2017.

[45] A. Mohemmed, S. Schliebs, S. Matsuda, and N. Kasabov, "Span: Spike pattern association neuron for learning spatio-temporal spike patterns," *International journal of neural systems*, vol. 22, no. 04, p. 1250012, 2012.

[46] S. Yin, S. K. Venkataramanaiah, G. K. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, and J.-s. Seo, "Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations," in *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, 2017, pp. 1–5.

[47] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[48] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

TABLE V
THE TEST ERROR FOR A SPIKING CONVOLUTIONAL NEURAL NETWORK.

	BP		SP	
	Surrogate	Shallow	Surrogate	Voltage
Fashion-MNIST	6.70	16.42	9.51	10.68
MNIST	0.84	7.24	1.01	2.63

B. Results

We compare four spiking models on the MNIST and Fashion-MNIST datasets - Table. V. The BP model propagates backward through the spiking equations at each layer using a differentiable surrogate. The Shallow model only trains the classification layer. The SP Surrogate model uses the same differentiable surrogate as BP does, but SP propagates forward through the network and therefore does not need to go through the spiking equation to deliver a learning signal. That is, the parameter update and surrogate are before or perpendicular to spiking, possibly as separate compartment. Finally, the SP voltage model uses the neuron's voltage (i.e. directly uses the spiking equation) to calculate the loss and update the parameters, no surrogate is used.

In contrast, BP based learning (without considerable modifications and additions) struggles when only using the voltage for learning [37], [38]. A differentiable nonlinear function estimating the spiking behavior (i.e. surrogate) is necessary for reasonable performance in BP learning. A surrogate is also necessary for sigprop to come close to BP surrogate performance. Even without a surrogate, the SP Voltage model is able to train the network significantly better than the Shallow model. To the best of our knowledge, sigprop is the only learning framework with a global supervised (unsupervised, reinforcement) learning signal that satisfies requirements for hardware (on-chip) learning [8], [48].

VII. DISCUSSION AND CONCLUSION

Alternative learning algorithms to backpropagation relax constraints on learning under backpropagation, such as feedback connectivity, weight transport, multiple types of computations, or a backward pass. This is done to improve training efficiency, lowering time or memory, or enabling distributed or parallel execution; and, to improve compatibility with biological and hardware learning models. However, relaxing constraints negatively impacts performance. So, alternatives try varying relaxations or supplementary modifications and additions in an attempt to retain the performance found under backpropagation. For instance, the best performing and least constrained alternative algorithm, LL-FA, uses a layer-wise loss and random feedback to relax constraints, but adds layer-wise auxiliary networks to retain performance. In contrast, sigprop has no constraints on learning, beyond the inference model, and without constraining (e.g. layer-wise) additions or modifications.

We demonstrated that sigprop has faster training times and lower memory usage than BP, LL-BP, and LL-FA. The reason sigprop is more efficient than BP is clear, sigprop is forwardpass unlocked while BP is backwardpass locked. For LL-BP and LL-FA, sigprop is more efficient as it has fewer layers for learning, it has no auxiliary networks. LL-BP has 2 auxiliary layers for every hidden layer. LL-FA has 3 auxiliary layers for every hidden layer. In Section IV-B, we showed that sparse targets, which have a much smaller size than the hidden layer outputs, are able to train the hidden layer as well as dense targets, which have the same size as the hidden layer outputs. A key feature of learning in the brain and biological neural networks is sparsity. A small fraction of the neurons weigh in on computations and decision making. It is encouraging that sigprop is able to learn just as well with a sparse learning signal.

In Section V, we applied sigprop to a time continuous model using a Hebbian plasticity mechanism to update weights, demonstrating sigprop has dynamical and structural compatibility with biological and hardware learning. With this continuous model, we also showed that sigprop is able to provide useful learning signals. While sigprop improves the performance of EP, the Fashion-MNIST results demonstrate that there is room for growth. One problem may be that the layers on the path from the input to the output have their weight updates dominated by the input, so are struggling to come into alignment with the loopback layer. In future work, we will compensate to increase alignment.

In Section VI, we demonstrated a key feature of sigprop not seen in other global learning algorithms: sigprop does not need to go through a non-derivable, non-continuous spiking equation to provide a learning signal to hidden layers. This makes sigprop ideal for hardware (on-chip) learning. Furthermore, sigprop is able to train an SNN using spikes (voltage), which backpropagation struggles to do, and at a reduced 16-bit precision. So, no additional complex circuitry is necessary. This makes on-chip global learning (e.g supervised or reinforcement) more plausible with sigprop, whereas the complex neuron and synaptic models of previous supervised learning algorithms are impractical [8], [48]. This is in addition to sigprop not having architectural requirements for learning and having the same type of computation for learning and inference, which on their own address hardware constraints restricting the use of previous supervised learning algorithms [8], [48]. We are working to implement sigprop on hardware neural networks.

We demonstrated signal propagation, a new learning framework for propagating a learning signal and updating neural network parameters via a forward pass. Our work shows that learning signals can be fed through the forward path to train neurons. In biology, this means that neurons who do not have feedback connections can still receive a global learning signal through their incoming connections. In hardware, this means that global learning (e.g supervised or reinforcement) is possible even though there is no backward connectivity. At its core, sigprop re-uses the forward path to propagate a learning signal and generate targets. With this combination, there are no structural or computational requirements for learning, beyond the inference model. Furthermore, the network parameters are updated as soon as they are reached by a forward pass. So, sigprop learning is ideal for parallel training of layers or modules. In total, we presented learning models

target propagation. Target Propagation (TP), Fig 1d [18], [19] generates a target activation for each layer instead of gradients by propagating backward through the network. It requires reciprocal connectivity and is forwardpass and backwardpass locked. In contrast, sigprop generates a target activation at each layer by going forward through the network. An alternative approach, Equilibrium Propagation (EP) is an energy based model using a local contrastive Hebbian learning with the same computation in the inference and learning phases [6], [20], [21]. The model is a continuous recurrent neural network that minimizes the difference between two fixed points: when receiving an input only and when receiving the target for error correction. EP is closer to a framework, wherein symmetric and random feedback (FA) weights work [22]. These models of EP still require comprehensive connectivity for each layer and are forwardpass locked. We demonstrate that sigprop works in the EP framework without these problems, more closely modeling neural networks in the brain.

Another approach that reuses the forward connectivity for learning, as is we do in sigprop, is Error Forward Propagation, Fig 1e [23]–[28]. Error forward propagation is for closed loop control systems or autoencoders. In either case, the output of the network is in the same space as the input of the network. These works calculate an error between the output and input of the network and then propagate the error forward through the network, instead of backward, calculating the gradient as in error backpropagation. Error forward propagation is backwardpass locked and forwardpass locked. It also requires different types of computation for learning and inference. In contrast, sigprop uses only a single type of computation and is backwardpass unlocked and forwardpass unlocked.

III. SIGNAL PROPAGATION

The premise of signal propagation (sigprop) is to reuse the forward path to map an initial learning signal into targets at each layer for updating parameters. The network is shown in Fig. 2a; notice that training uses the same forward path as inference, except that instead of only feeding the network the input x, we also feed it c the learning signal. The learning signal is some context c, e.g. the label in supervised learning. The learning signal and the input can have different shapes, e.g. a supervised label is a single integer and the input is an image. The target generator projects the learning signal c and the first hidden layer projects the input x to both have the same shape (dense signal) or concordant shapes (sparse signal Sec III-E) to be processed by the network, e.g. the target generator projects the label to have the same shape as the input or even the first hidden layer. After which, the forward pass during training proceeds the same way as inference, except with x and c as the new inputs instead of only the original input x.

We provide a framework for any given input x or learning signal c, not only for supervised learning with labels. For example, in regression tasks, the inputs x and outputs y commonly have the same type and shape; so, by using the output training targets y^* as the learning signal c, the target generator and first hidden layer can be the same (weight sharing). Nonetheless, the focus here is supervised learning.

In the following sub-sections, we start with the general training procedure III-A, then prediction for both training and inference III-B, the loss for training III-C, and details of target generators III-D.

A. Training

The forward pass starts with the input x, a learning signal c, and the target generator. Assume the network has two hidden layers, as shown in Figures 2a, where W_i and b_i are weight and bias for layer i. Let S_1 and d_1 be the weight and bias for the target generator. The activation function $f()$ is a non-linearity. Let (x, y^*) be a mini-batch of inputs and labels of m possible classes. We feed x into the first hidden layer to get h_1. We create a one-hot vector of each class c_m, this is our learning signal, and feed it into the target generator to get t_1. Notice that x and c_m have different shapes. Now, h_1 and t_1 have the same shape.

$$h_1, t_1 = f(W_1 x + b_1), f(S_1 c_m + d_1) \tag{1}$$
$$[h_2, t_2] = f(W_2[h_1, t_1] + b_2) \tag{2}$$
$$[h_3, t_3] = f(W_3[h_2, t_2] + b_2) \tag{3}$$

The outputted t_1 is a target for the output of the first hidden layer h_1. This target is used to compute the loss $L_1(h_1, t_1)$ for training the first hidden layer and the target generator. Then, the target t_1 and the output h_1 are fed to the next hidden layer. The forward pass continues this way until the final layer. The final layer and each hidden layer have their own losses:

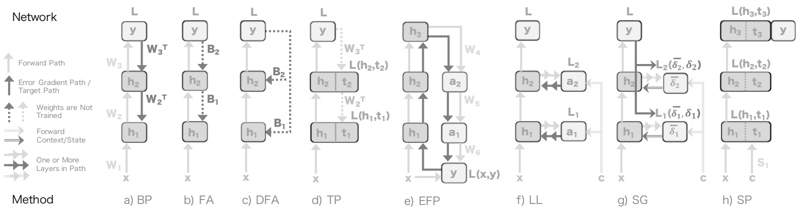
$$J = L(h_1, t_1) + L(h_2, t_2) + L(h_3, t_3) \tag{4}$$

where J is the total loss for the network. For hidden layers, the loss L can be a supervised loss, such as L_{pred} Eq. 9 which is used in Section IV. It can also be a Hebbian update rule, such as Eq. 14 which is used in Section V. For the final layer, the loss L is a supervised loss, such as L_{pred} Eq. 9.

In total, each layer processes its input and input-target to create an output and output-target. The layer compares its output with its output-target to update its parameters. In this way, the layer locally computes its update from a global learning signal. The layer then sends its output and output-target to the next layer which will compute its own update. This processes continues until the final layer has computed its update and produces the network's output (prediction). From this procedure collectively, the network learns to process the input to produce an output, and at the same time, learns to make an initial learning signal into a useful training target at each hidden layer and final layer. In other words, the network itself, which is the forward path, takes on the role of the feedback connectivity in producing a learning signal for each layer. This makes sigprop compatible with models of learning where backward connectivity is limited, such as in the brain and learning in hardware (e.g. neuromorphic chips).

B. Prediction for Training and Inference

In training, the prediction y is formed by comparing the final layer's output h_3 with its target t_3 (Output Target) - Fig 2a. For inference, the same procedure may be used if group targets,



Method	a) BP	b) FA	c) DFA	d) TP	e) EFP	f) LL	g) SG	h) SP
Forw. Locked	global	global	global	global	global	local∓	local*	none
Backw. Locked	global	global	none	global	global	local∓	none*	none
Backprop Error	yes	no	no	no	no	yes	yes	no
Backprop Target	no	no	no	yes	no	no	no	no
Direct Error	no	no	yes	no	no	no	no	no
Forprop Error	no	no	no	no	yes	no	no	no
Forprop Target	no	no	no	no	no	no	no	yes
Direct Target	no	no	no	no	no	yes°	yes°	no
Local Loss	no	no	no	yes	no	yes	no	yes
Global Signal	yes	yes	yes	yes	yes	no	no*	yes

Fig. 1. Comparison of Learning Algorithms Relaxing Learning Constraints Under Backpropagation. **a)** the backpropagation algorithm **b,c)** the feedback alignment and direct feedback alignment algorithms. FA based algorithms do not solve forwardpass locking and require additional connectivity. **d)** target propagation uses a single type of computation for training and inference, but is forwardpass locked and requires feedback connectivity. **e)** error forward propagation for closed loop systems or autoencoders reuses the forward connectivity to propagate error, but is otherwise similarly constrained as backpropagation. **f)** local learning with layer-wise training using auxiliary classifiers. ∓LL is forwardpass and backwardpass locked at the layer level as the auxiliary networks use backpropagation. Backpropagation in the auxiliary networks may be substituted with an alternative model, such as FA. **g)** the synthetic gradients algorithm. *SG based algorithms are only forwardpass and backwardpass unlocked after learning to predict the synthetic gradient. **h)** the signal propagation learning algorithm presented in this work. SP feeds the learning signal forward through the network to solve the weight transport and forwardpass locking problems without requiring additional connectivity requirements. For SP, taking t_3 with h_3 produces y, however a classification layer may also be used Fig. 2. **Table)** Direct Error and Direct Target means that a model uses the error or target directly at layer h_i. °Direct target can be substituted in LL and SG, with direct error or temporary use of backpropagation for example. Forprop stands for forward propagation. Forprop error and Forprop target means the model uses the error or target starting at the input layer, instead of starting at the output layer as is done in backpropagation. Global Signal means the learning signal is propagated through the network instead of sent directly to or formed at each hidden layer. **Networks)** The light grey arrows indicate the feed forward path. Dark grey arrows indicates error gradient or target paths. If the dark grey arrow pass through a layer, the weights are not trained by the error gradient or target. Dotted lines indicate the weights are not trained. Double lines, light or dark grey, are forwarding the context c or state h_i, without modification. Double arrows indicate going through one or more intermediate hidden layers. W_i and S_i are trained weights and B_i are fixed random weights. There are versions of these models where B_i is trained to be the transpose of W_i. The loss function is L and takes the output of the previous layer and possibly some target y^* when unspecified. The target generator layer S_1 generates the initial training target t_i from a learning signal, which is some privileged information or context c, usually the label in supervised learning. The gradient is δ and the synthetic gradient is $\hat{\delta}$. Auxiliary networks are represented by the double arrows going into a_i and $\hat{\delta}_i$.

such as class labels, are available. However, no target of any kind is needed for inference - Fig 2b. Instead, a classification layer may be used with no effect on performance (Classification Layer) Fig 2b. In general, the last layer may be any type of prediction layer, such as a classification layer or the output layer for regression tasks. With a prediction layer, inference for classification, regression, or any task proceeds as usual, without using a target. We describe both version of sigprop below.

Output Target, Fig 2a: The network's prediction y at the final layer is formed by comparing the output h_3 and outputted target t_3 (Fig 2a):

$$y = y_3 = O(h_3, t_3) \tag{5}$$

where O is a comparison function. Two such comparison functions are the dot product and L2 distance. We use the less complex O_{dot},

$$O_{dot}(h_i, t_i) = h_i \cdot t_i^T \tag{6}$$

$$O_{l2}(h_i, t_i) = \sum_k ||t_i[i,1,k] - h_i[1,j,k]||_2^2 \tag{7}$$

but both versions give similar performance using the losses in Section III-C. Each hidden layer can also output a prediction, these are known as early exits (faster responses from earlier layers during inference):

$$y = y_i = O(h_i, t_i) \tag{8}$$

Classification Layer, Fig 2b: The final layer of the network may be replaced with the standard output layer used in neural networks, e.g. the classification layer for supervised learning, as shown in Fig 2b. This simplifies predictions during inference, matching standard neural network design. In this case, the learning signal c (e.g. labels in supervised learning) would be projected to the final layer of the network, as per standard training of networks. The target t_3 is no longer used during inference to form y, so neither is the context generator.

C. Training Loss

In sigprop, losses compare neurons with themselves over different inputs and with each other. The L_{pred} is the basic loss we use.

Prediction Loss: The prediction loss is a cross entropy loss using a local prediction, Eq 8. The local prediction is from a dot product between the layer's local targets t_i and the layer's output h_i. Given a hidden layer's local targets $t_i = (t_i^1, \ldots, t_i^m)$

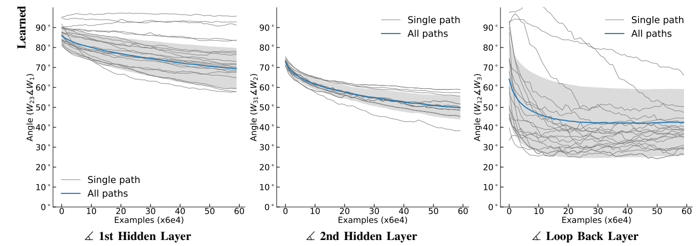


Fig. 4. Signal Propagation updates bring weights into alignment within 90°, approaching backpropagation symmetric weight alignment. Sigprop provide useful targets for learning. The weight alignment for a network with two hidden layers W_1 and W_2 and one loop back layer W_3 is shown. The weight matrices form a loop in the network and come into alignment with each other during training on the Fashion-MNIST dataset. Each weight matrix aligns with the product of the other two weights forming the network loop. $W_{xy} \measuredangle W_z$ means the angle between weight z and the matrix multiplication of the weights x and y. **Learned**) The loop back layer is trained. However, even a fixed loop back layer reaches a similar angle of alignment. **Layers**) The loop back layer converges before the 1st and 2nd hidden layers can. The 1st hidden layer is the least aligned with the 2nd hidden layer and the loop back layer because it is dominated by the input signal. The alignment angles are taken for every sample and error bars are one standard deviation.

VI. Spiking Neural Networks

We demonstrate that sigprop can train a spiking neural model with only the voltage (spike), and improves the hardware compatibility of surrogate functions by reducing them to local update rules. This is an improvement over backpropagation based approaches as they: struggle to learn with only the voltage; require going backward through non-derivable, non-continuous spiking equations; and require comprehensive feedback connectivity - all of which are problematic for hardware and biological models of learning [8], [37], [38].

Spiking is the form of neuronal communication in biological and hardware neural networks. Spiking neural networks (SNN) are known to be efficient by parallelizing computation and memory, overcoming the memory bottleneck of Artificial Neural Networks (ANN) [39]–[41]. However, SNNs are are difficult to train. A key reason is that spiking equations are non-derivable, non-continuous and spikes do not necessarily represent the internal parameters, such as membrane voltage of the neuron before and after spiking [8]. Spiking also has multiple possible encodings for communication when considering time which are non-trivial, whereas artificial neural networks (ANN) have a single rate value for communication [8]. One approach to training SNNs is to convert an ANN into a spiking neural network after training [42]–[44]. Another approach is to have an SNN in the forward path, but have a backpropagation friendly surrogate model in the backward path, usually approximately making the spiking differentiable in the backward path to update the parameters [8], [45], [46].

We trained SNNs with sigprop. The target is forwarded through the network with the input, so learning is done before the spiking equation. That is, we do not need to differentiate a non-derivable, non-continuous spiking equation to learn. Also, the SNN has the same dynamics in inference and learning and has no reciprocal feedback connectivity. This makes sigprop ideal for on-chip, as well as off-chip, training of spiking neural networks. We measure the performance of this model on the MNIST and Fashion-MNIST datasets.

A. Spiking Neural Network

We train a convolutional spiking neural network with Integrate-and-Fire (IF) nodes, which are treated as activation functions. The IF neuron can be viewed as an ideal integrator where the voltage does not decay. The subthreshold neural dynamics are:

$$v_i^t = v_i^{t-1} + h_i^t \tag{15}$$

where v_i^t is the voltage at time t for neurons of layer i and h_i^t is the layer's activations. The surrogate spiking function for the IF neuron is the arc tangent

$$g(x) = \frac{1}{\pi}\arctan(\pi x) + \frac{1}{2} \tag{16}$$

where the gradient is defined by

$$g'(x) = \frac{1}{1 + (\pi x)^2} \tag{17}$$

The neuron spikes when the subthreshold dynamics reach 0.5 for sigprop, and 1.0 for BP and Shallow models. All models is simulated for 4 time-steps, directly using the subthreshold dynamics. The SNN has 4 layers. The first two are convolutional layers, each followed by batch normalization, an If node, and a $2x2$ maxpooling. The last two layers are fully connected, with one being the classification layer. The output of the classification layer is averaged across all four time steps and used as the network output. ADAM was used for optimization [29]. The learning rate was set to $5e-4$. Cosine Annealing [47] was used as the learning rate schedule with the maximum number of iterations T_{max} set to 64. The models are trained on the MNIST and Fashion-MNIST datasets for 64 epochs using a batchsize of 128. We use automatic mixed precision for 16-bit floating operations, instead of the only the full 32-bit. The reduced precision is better representative of hardware limitations for learning. We use the classification layer version of sigprop Fig. 2b.

into alignment. In Section V-B, we measured the performance of this model on the MNIST and Fashion-MNIST datasets [34], [35].

A. A Continuous Recurrent Neural Network Model

The learning framework, Equilibrium Propagation (EP), proposed in [6] is one way to introduce physical time in deep continuous learning and have the same dynamics in inference and learning, avoiding the need for different hardware for each. EP has been used with symmetric or random feedback weights. We combine Sigprop with EP such that there are no additional constraints on learning, beyond the Hebbian update. We trained deep recurrent networks with a neuron model based on the continuous Hopfield model [36]:

$$\begin{aligned}\frac{\mathrm{d}s_j}{\mathrm{d}t} =& \frac{\mathrm{d}\rho(s_j)}{\mathrm{d}s_j}\Big(\sum_{i\to j} w_{ij}\rho(s_i) + \sum_{i\in O\to j\in I} w_{ij}\rho(s_i) + b_j\Big) \\ &- \frac{s_j}{r_j} - \beta\sum_{j\in O}(s_j - d_j)\end{aligned} \quad (13)$$

where s_j is the state of neuron j, $\rho(s_j)$ is a non-linear monotone increasing function of it's firing rate, b_j is the bias, β limits magnitude and direction of the feedback, O is the subset of output neurons, I is the subset of input receiving neurons, and d_j is the target for output neuron j. The input receiving neurons, $s_j \in I$, are the neurons with forward connections from the input layer. The networks are entirely feedforward except for the final feedback loop from the output neurons $s_i \in O$ to the input receiving neurons $s_j \in I$. All weights and biases are trained. The weights in the feedback loop connections may be fixed or trained. The output neurons receive the L_2 error as an additional input which nudges the firing rate towards the target firing rate d_j. The target firing rate d_j is the one-hot vector of the target value; all tasks in this section are classification tasks.

The EP learning algorithm can be broken into the free phase, the clamped phase, and the update rule. In the free phase, the input neurons are fixed to a given value and the network is relaxed to an energy minimum to produce a prediction. In the clamped phase, the input neurons remain fixed and the rate of output neurons $s_j \in O$ are perturbed toward the target value d_j, given the prediction s_j, which propagates to connected hidden layers. The update rule is a simple contrastive Hebbian (CHL) plasticity mechanism that subtracts $s_i^0 s_j^0$ at the energy minimum (fixed point) in the free phase from $s_i^\beta s_j^\beta$ after the perturbation of the output, when $\beta > 0$:

$$\Delta W_{ij} \propto \rho(s_i)\frac{\mathrm{d}}{\mathrm{d}\beta}(\rho(s_j)) \approx \frac{1}{\beta}\rho(s_i^0)(\rho(s_j^\beta) - \rho(s_j^0)) \quad (14)$$

The clamping factor β allows the network to be sensitive to internal perturbations. As $\beta \to +\infty$, the fully clamped state in general CHL algorithms is reached where perturbations from the objective function tend to overrun the dynamics and continue backwards through the network.

B. Signal Propagation Provides Useful Learning Signals

We look at the behavior of our model during training and how the feedback loop drives weight changes. Precise symmetric connectivity was thought to be crucial for effective error delivery [1]. Feedback Alignment, however, showed that approximate symmetry with reciprocal connectivity is sufficient for learning [10]–[12]. Direct Feedback Alignment showed that approximate symmetry with direct reciprocal connectivity is sufficient. In the previous sections, we showed that no feedback connectivity is necessary for learning. Here, we conduct an experiment to show that the same approximate symmetry is found in sigprop.

We provide evidence that sigprop brings weights into alignment within 90°, known as approximate symmetry. In comparison, backpropagation has complete alignment between weights, known as symmetric connectivity. Note that this is not a measure of approximation to backpropagation - sigprop is a new and different approach; instead, this is a measure of the quality of the learning signal in deeper layers, contextualized by observations of learning with backpropagation, particularly symmetry. In this experiment, the sigprop network architecture forms a loop, so all the weights serve as both feedback and feedforward weights. For a given weight matrix, the feedback weights are all the weights on the path from the downstream error to the presynaptic neuron. In general, this is all the other weights in the network loop. The weight matrices in the loop evolve to align with each other as seen in Fig. 4. More precisely, each weight matrix roughly aligns with the product of all the other weights in the network loop. In Fig. 4, the weight alignment for a network with two hidden layers W_1 and W_2 and one loop back layer W_3 is shown.

Information about W_3 and W_1 flows into W_2 as roughly W_3W_1, which nudges W_2 into alignment with the rest of the weights in the loop. From equation 14, $W_2 \propto \rho(\vec{s}_2^0)(\rho(\vec{s}_3^\beta) - \rho(\vec{s}_3^0))$ where $\vec{s}_2 \leftarrow \rho(\vec{s}_1)W_1$, which means information about W_1 accumulates in W_2. Similarly, $W_1 \propto \rho(\vec{s}_1^0)(\rho(\vec{s}_2^\beta) - \rho(\vec{s}_2^0))$, except since the network architecture is a feedforward loop, $\vec{s}_1 \leftarrow \rho(\vec{s}_3)W_3$, which means information about W_3 accumulates in W_1. The result is shown in column c of the bottom row of Fig. 4, where a weight matrix is fixed and the rest of the network's weights come into alignment with the fixed weight. Notice that W_3W_1 has the same shape as W_2^T and serves as it's 'feedback' weight.

C. Classification Results

We provide evidence that sigprop with EP has comparable performance to EP with symmetric weights, and report the performance results of the experiment in the previous section. A two and another three layer architecture of 1500 neurons per layer were trained. The two layer architecture was run for sixty epochs and the three layer for one hundred and fifty epochs. The best model during the entire run was kept. On the MNIST dataset [34], the generalization error is $1.85 - 1.90\%$ for both the two layer and three layer architectures, an improvement over EP's $2 - 3\%$. The best validation error is $1.76 - 1.80\%$ and the training error decreases to 0.00%. To demonstrate that sigprop provides useful learning signals in the previous section, we trained the network on the more difficult Fashion-MNIST dataset [35]. The generalization error is 11.00%. The best validation error is 10.95% and the training error decreases to 2%.

and a size n mini-batch of outputs $h_i = (h_i^1, \ldots, h_i^n)$ of the same hidden layer:

$$L_{pred}(h_i, t_i) = \mathrm{CE}(y_i^*, -O_{dot}(h_i, t_i)) \quad (9)$$

where h_i and t_i have the same size output dimension. The cross entropy loss (CE) uses y_i^*, which is a reconstruction of the labels y^* at each layer i from the positional encoding of the inputs x and context c_m., starting from the activations h_1 and targets t_1 formed at the first hidden layer. In particular, we form a new batch $[h_1, t_1]$ by interleaving h_1 and t_1 such that each sample's activations in h_1 is concatenated after its corresponding target t_1. Then, at each layer i, we assign a label for each sample h_{ij} depending on which target t_{ik} the sample came after, where $0 \leq k < j$. Many different encodings are available, depending on the task and target generator. An alternative is to use the approach in Section V which merges the context c, and therefore generated targets t_1, with the inputs x to form a single combined input xt, an input-target III-D, and then either compares them with each other or uses an update rule over multiple iterations. The second option is natural for continuous networks where multiple iterations (e.g time steps) can support robust update rules.

D. Target Generators

The target generator takes in a learning signal as some context c to condition learning on and then produces the initial target, which is fed forward through the network to produce targets at each hidden layer. There are many possible formulations of the target generator, such as: fixed or learned, projecting to input or first hidden layer, and sharing weights with the first hidden layer. We recommend deciding based on the task, selected learning signal(s), and implementation constraints. For example, in segmentation tasks where outputs have the same shape as the inputs, we can use the output training segmentation targets for the learning signal and have the target generator share weights with the first hidden layer. We describe three formulations below to address different learning scenarios, particularly hardware constrained, continuous, and spike-time learning.

Target-Only, Fig 2a,b: This is the version described in Eq. 1 and conditions only on the class label. This version of the target generator can interfere with batch normalization statistics as h_1 and t_1 do not necessarily have similar enough distribution. Batch normalization statistics may be disabled or be put in inference mode when processing the targets, therefore only collecting statistics on the input.

Target-Input, Fig 2a,b: Another context we condition on is the class label and input. We feed a one-hot vector of the labels y_m^* through the target generator to produce a scale and shift for the input. We take the scaled and shifted output as the target for the first hidden layer.

$$t_1 = h_1 f(S_1 c_m + d_1) + f(S_2 c_m + d_2) \quad (10)$$

The target t_1 is now more closely tied to the distribution of the input. We found that this formulation of the target works better with batch normalization. Even though this version has similar performance to Eq. 1, it increases memory usage as each input will have its own version of the targets.

Target-Loop, Fig 2c: The last option is to incorporate a form of feedback. The immediate choice is to condition on the activations of the predictions y_3 and labels y_m^*,

$$t_1 = f(S_1 y_3 + S_1 y_m^* + d_1) \quad (11)$$

or using the final layer's output and error e_3 with the target t_3 to correct it

$$\begin{aligned} t_1 &= f(S_1(h_3 - \eta e_3) + d_1) \quad (12) \\ &\triangleq f(S_1(h_3 - \eta\frac{\mathrm{d}L}{\mathrm{d}h_3}) + d_1)\end{aligned}$$

where η controls how much error e_3 to integrate. We use it in Section V for continuous networks.

E. Sparse Learning

Sigprop can be a form of sparse learning. We reformulate the target generator to produce a sparse target, which is a sparse learning signal. We make the targets t_i as sparse as possible such that at minimum, they can still be taken with each layer's weights W_i, via a convolution or dot-product, and then fed-forward through the network. To make the target sparse, we reduce the output size of S_i in the target generator. We use sparse learning throughout this paper, except when otherwise written.

For convolutional layers, the output size of S_i is made the same size as the weights. For example, let there be an input of $32x28x28$ and a convolutional hidden layer of $32x16x3x3$, where 32 is the in-channels, $28x28$ is the width and height of the input, 16 is the out-channels, and $3x3$ is the kernel. The dense target's shape is $32x28x28$. In contrast, the sparse target's shape is reduced to $10x32x3x3$. As a result, even though convolutional layers have weight sharing, there is no weight sharing when convolving with a sparse target.

For fully connected layers, the output size of S_i is made smaller than input size of the weights. For example, let there be an input of 1024 and a fully connected hidden layer of $1024x512$ features. The dense target's shape would be 1024. In contrast, the sparse target's shape is < 1024. Then, we resize the target to match the layer input size of 1024 by filling it with zeros. With the sparse target, the layer is no longer fully connected.

IV. Experiments

We compare sigprop (SP) with Feedback Alignment (FA) and Local Learning (LL). We also show results for backpropagation (BP) as reference. The models are shown in Figure 1. FA uses fixed random weights to transport error gradient information back to hidden layers, instead of using symmetric weights. For LL, we show results for two model versions. The first uses BP at the layer level (LL-BP), and the second uses FA in the auxiliary networks to have a backpropagation free model that relaxes learning constraints under backpropagation (LL-FA). LL-FA performs better than using FA or DFA alone. We use LL-BP and LL-FA with predsim losses on the VGG8b architecture [14]. We trained several network on the CIFAR-10,

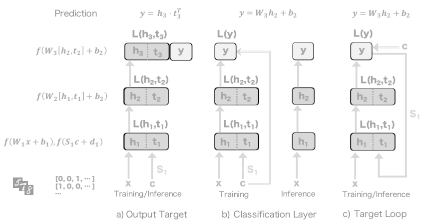


Fig. 2. Different Versions of sigprop (SP). **a)** For sigprop, the prediction y is formed by taking t_3 with h_3. sigprop does not need a classification layer. **b)** However, a classification layer may be used without effecting performance. In this case, the last hidden layer's outputs are sent to the classification layer. The classification layer has a benefit for inference. During inference, the target t_3 is no longer needed to make predictions, so the context c and target generator are not used. **c)** This is the version of sigprop used in Sections V for the continuous rate model. The classification layer feeds back into the input layer creating a feedback loop, so y is the context c: $y = c$. This feedback loop allows the target of hidden layers earlier in the network to incorporate information from hidden layers later in the network without incurring the overhead of reciprocal feedback to every neuron. Continuous networks have multiple iterations which is ideal for this version of sigprop. The other versions of sigprop may also be used.

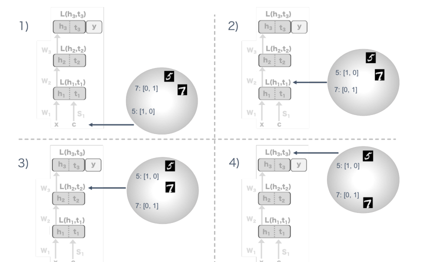


Fig. 3. Training in sigprop (SP). The learning signals c and inputs x are fed into the network. Then, each layer successively brings the learning signal $5 : [1, 0]$ closer to the images of 5, but farther away from learning signal $7 : [0, 1]$ and images of 7. The same is done for 7. Before the first layer **1)**, the images and learning signal of the same class are not closer to each other than to other classes. In the first layer **2)**, we nudge $5 : [1, 0]$ and the image of 5 closer; the same for 7. This continues in the following layer **3)** and then the final layer **4)**, at which point the learning signal and inputs of the same class are close each other, but farther from the other class. In general, each layer successively bring inputs x and there respective learning signals c closer together than all other inputs and learning signals.

CIFAR-100, and SVHN datasets. We used a VGG architecture. The experiments were run using the PyTorch Framework. All training was done on a single GeForce GTX 1080. For each layer to have a separate loss, the computational graph was detached before each hidden layer to prevent the gradient from propagating backward past the current layer. The target generator was conditioned on the classes, producing a single target for each class.

Results for BP, LL-BP, LL-FA, and SP A batch size of 128 was used. The training time was 100 epochs for SVHN, and 400 epochs for CIFAR-10 and CIFAR-100. ADAM was used for optimization [29]. The learning rate was set to $5e-4$. The learning rate was decayed by a factor of .25 at 50%, 75%, 89%, and 94% of the total epochs. The leaky ReLU activation with a negative slope of 0.01 was used [30]. Batch normalization was applied before each activation function [31] and dropout after. The dropout rate was 0.1 for all datasets. The standard data augmentation was composed of random cropping for all datasets and horizontal flipping for CIFAR-10 and CIFAR-100. The results over a single trial for VGG models.

The CIFAR-10 dataset [32] consists of 50000 $32x32$ RGB images of vehicles and animals with 10 classes. The CIFAR-100 dataset [32] consists of 50000 $32x32$ RGB images of vehicles and animals with 100 classes. The SVHN dataset [33] consists of $32x32$ images of house numbers. We use both the training of 73257 images and the additional training of 531131 images.

A. Efficiency

We measured training time and maximum memory usage on CIFAR-10 for BP, LL-BP, LL-FA, and SP. The version of SP used is 2b with the L_{pred} loss. The results are summarized in Table I. LL and SP training time are measured per layer as they are forwardpass unlocked and layers can be updated in parallel. However, BP is not forwardpass unlocked, so layers are updated sequentially and is therefore necessarily measured at the network level. Measurements are across all seven layers, which is the source of the high variance for LL and SP, and over four hundred epochs of training. To ensure training times are comparable, we compare the epochs at which SP, LL, and BP converge toward their lowest test error. We also include the first epochs that have performance within 0.5% of the best reported performance. All learning algorithms converge within significance of their best performance around the same epoch. Given efficiency per iteration, SP is faster than the other learning algorithms and has lower memory usage.

The largest bottleneck for speed of LL and SP is successive calls to the loss function in each layer. Backpropagation only needs to call the loss function once for the whole network; it optimizes the forward and backward computations for all layers and the batch. SP and LL would benefit from using a larger batch size than backpropagation. The batch size could be increased in proportion to the number of layers in the network. This is only pragmatic in cases where memory can be sacrificed for more speed (e.g. not edge devices). We also provide per layer measurements in Tables II. At the layer level, SP remains faster and more memory efficient than LL and backpropagation. It is interesting to note that LL and SP tend to be slower and faster in different layers even though both are using the same architecture. For memory, SP uses less memory than LL and BP regardless of the layer. However, there is a general trend for LL and SP: the layers closer to the input have more parameters, so are slower and take up more memory then layers closer to the output.

B. Sparse Local Targets

We demonstrate that sigprop (SP) can train train a network with a sparse learning signal. We use the larger VGG8b(2x) architecture to leave more room for possible improvement when using this sparse target. The version of sigprop is 2b with the L_{pred} loss. We use the CIFAR10 dataset with the same configuration as in Section IV. We see that the network's training speed increased and memory usage decreased Fig. III,IV, with negligible change in accuracy.

TABLE I
THE TRAINING TIME PER SAMPLE AND MAXIMUM MEMORY USAGE PER BATCH OVER ALL LAYERS FOR VGG8B

		Backprop		Alternative	
		BP	LL-BP	LL-FA	SP
Time (s)	CIFAR-10	12.29 ± 0.02	8.11 ± 14.40	8.50 ± 29.86	$\mathbf{5.91} \pm 7.40$
	CIFAR-100	15.34 ± 1.45	10.20 ± 28.98	9.44 ± 28.63	$\mathbf{6.25} \pm 7.33$
	SVHN	148.70 ± 2.23	95.51 ± 3617.90	89.32 ± 1767.26	$\mathbf{69.74} \pm 1048.54$
Mem (MiB)	CIFAR-10	22.00 ± 0.00	8.85 ± 8.06	13.03 ± 10.61	$\mathbf{6.19} \pm 1.57$
	CIFAR-100	27.16 ± 0.38	11.45 ± 106.02	5.51 ± 23.17	$\mathbf{5.19} \pm 16.72$
	SVHN	28.04 ± 2.68	11.41 ± 106.03	5.43 ± 23.04	$\mathbf{4.91} \pm 16.54$
Best Epoch	CIFAR-10	319(198)	266(164)	309(201)	313(207)
	CIFAR-100	350(306)	380(209)	339(264)	329(219)
	SVHN	98(11)	41(7)	93(23)	88(34)
Test Error (%)	CIFAR-10	5.99	**5.58**	9.02	<u>8.34</u>
	CIFAR-100	**26.20**	29.31	38.41	<u>34.30</u>
	SVHN	2.19	**1.77**	2.55	<u>2.15</u>

TABLE II
THE TRAINING TIME PER SAMPLE AND MAXIMUM MEMORY USAGE PER BATCH PER LAYER ON CIFAR-10 FOR VGG8B

	Backprop	Alternative	
Layer	LL-BP	LL-FA	SP
		Time (s)	
1	7.16 ± 0.04	6.21 ± 0.03	$\mathbf{4.48} \pm 0.05$
2	15.80 ± 0.07	15.15 ± 0.09	$\mathbf{8.95} \pm 0.15$
3	9.27 ± 0.04	$\mathbf{7.09} \pm 0.02$	10.13 ± 0.14
4	9.25 ± 0.30	18.40 ± 0.06	$\mathbf{7.27} \pm 0.25$
5	4.93 ± 0.01	5.66 ± 0.04	$\mathbf{4.71} \pm 0.05$
6	7.46 ± 0.01	3.93 ± 0.02	$\mathbf{3.44} \pm 0.02$
7	2.90 ± 0.00	3.00 ± 0.00	$\mathbf{2.36} \pm 0.03$
		Mem (MiB)	
1,6,7	6.12	10.98	**5.67**
2	14.50	18.18	**9.26**
3	9.70	18.18	**5.67**
4,5	9.70	10.97	**5.67**

TABLE III
EFFICIENCY OF TARGETS OVER ALL LAYERS ON CIFAR-10 FOR VGG8B(2X). TRAINING TIME PER SAMPLE, MAXIMUM MEMORY USAGE PER BATCH

	Dense	Sparse
Time (s)	14.48 ± 54.29	$\mathbf{9.56} \pm 29.02$
Mem (MiB)	14.04 ± 6.39	$\mathbf{10.74} \pm 65.10$
Best Epoch	273(207)	340(219)
Test Error (%)	7.60	7.71

TABLE IV
EFFICIENCY OF TARGETS PER LAYER ON CIFAR-10 FOR VGG8B(2X). TRAINING TIME PER SAMPLE AND MAXIMUM MEMORY USAGE PER BATCH

Layer	Time s (Mem MiB)			
	Dense		Sparse	
1	12.85 ± 5.66	(12.99)	$\mathbf{7.42} \pm 0.79$	(**6.34**)
2	21.51 ± 9.31	(**20.23**)	$\mathbf{19.70} \pm 0.18$	(27.53)
3	18.81 ± 5.50	(13.02)	$\mathbf{9.30} \pm 0.39$	(**9.41**)
4	25.30 ± 12.97	(**13.02**)	$\mathbf{14.19} \pm 0.12$	(15.99)
5	9.69 ± 1.86	(13.02)	$\mathbf{8.84} \pm 0.11$	(**9.10**)
6	8.11 ± 3.16	(13.02)	$\mathbf{5.24} \pm 0.08$	(**6.15**)
7	5.06 ± 1.61	(12.99)	$\mathbf{2.25} \pm 0.07$	(**0.68**)

V. IN CONTINUOUS TIME

We demonstrate that sigprop can train a neural model in the continuous setting using a Hebbian update mechanism, in addition to the discrete setting. Biological neural networks work in continuous time, have no indication of different dynamics in inference and learning, and use Hebbian based learning. Sigprop improves learning in this scenerio by bringing a global learning signal into Hebbian based learning, without the comprehensive feedback connectivity to neurons and layers that previous approaches require, not observed in biological networks. In addition, sigprop improves compatibility for learning in hardware, such as neuromorphic chips, which have resource and design constraints that limit backward connectivity.

In the model presented in this section, the target generator is conditioned on the activations of the output layer to produce a feedback loop - Fig. 2c. The feedback loop is always active, during training and inference. With this feedback loop, we demonstrate in section V-A that sigprop provides useful learning signals by bringing forward and feedback loop weights