Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

Towards robust, interpretable neural networks via Hebbian/anti-Hebbian learning: A software framework for training with feature-based costs

Metehan Cekic [*], Can Bakiskan, Upamanyu Madhow

Department of Electrical and Computer Engineering, University of California Santa Barbara, Santa Barbara, CA 93106, United States of America

A R T I C L E   I N F O

Keywords:
Interpretable ML
Hebbian learning
Neuro inspired
Machine learning

A B S T R A C T

Conventional deep neural network (DNN) training with an end-to-end cost function is unable to exert control on, or to provide guarantees regarding the features extracted by the layers of a DNN. Thus, despite the pervasive impact of DNNs, there remain significant concerns regarding their (lack of) interpretability and robustness. In this work, we develop a software framework in which end-to-end costs can be supplemented with costs which depend on layer-wise activations, permitting more fine-grained control of features. We apply this framework to include Hebbian/anti-Hebbian (HaH) learning in a discriminative setting, demonstrating promising gains in robustness for CIFAR10 image classification.

Code metadata

Current code version	v0.0.5
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2022-68
Permanent link to Reproducible Capsule	https://codeocean.com/capsule/0731065/tree/v1
Legal Code License	MIT License
Code versioning system used	Git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Python >= 3.8.2 with the following dependencies PyTorch 1.10.2; Numpy 1.19.2
If available Link to developer documentation/manual	https://github.com/metehancekic/HaH
Support email for questions	metehancekic@ucsb.edu

1. Introduction

Deep neural networks (DNNs) have gone well beyond their image classification roots to impact an increasing variety of applications [1–4]. Conventional top-down training employs a cost function based on the DNN output, thus providing little insight and no guarantees on the features extracted by the layers of the DNN. The use of the resulting "black box" DNNs in many safety- and security-critical applications is blocked by concerns about their lack of interpretability and robustness. While data augmentation has been shown to enhance robustness to some extent (e.g., the use of adversarial examples generated on the fly during adversarial training), it is only a partial solution.

In this paper, we develop and open-source a training framework aimed at shaping the features generated by a DNN layer, by supplementing end-to-end costs with costs that depend on the activations at each layer. We apply this framework to neuro-inspired Hebbian/anti-Hebbian (HaH) learning, seeking to generate sparse activation patterns with a small fraction of large activations, instead of the large proportion of small activations produced by a standard DNN. With HaH costs, neurons which are strongly activated by an input take a step in the direction of the input, while the remaining neurons take a step in the direction opposite to the input. We also introduce changes in the inference framework, replacing a standard DNN layer with a HaH block which includes implicit weight normalization for each neuron, allowing us to interpret its output as a projection, and divisive output normalization, allowing us to suppress weak activations using strong ones. These result in invariance to input and weight scaling.

We implement our ideas via a publicly available extension module to PyTorch [5] called HaH (Hebbian/Anti-Hebbian): a neuro-inspired

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: metehancekic@ucsb.edu (M. Cekic), canbakiskan@ucsb.edu (C. Bakiskan), madhow@ucsb.edu (U. Madhow).

https://doi.org/10.1016/j.simpa.2022.100347
Received 17 May 2022; Received in revised form 22 June 2022; Accepted 28 June 2022

2665-9638/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

DNN toolbox, with new components such as Cost, Regularizer, Divisive normalizer, and Threshold. Experiments on CIFAR10 [6] image classification, including HaH Blocks into VGG16, show that training with HaH costs is more robust to noise and adversarial perturbations than the baseline, even without data augmentation.

1.1. Related work

Since the neocognitron proposed about four decades ago [7], there have been sporadic attempts to introduce Hebbian learning in artificial neural networks, including recent work on deep architectures [8]. Divisive normalization is also a well-known concept in neuroscience [9,10], and was shown to be competitive with standard batch norm in [11]. However, as far as we know, ours is the first work to show how to employ a combination of these concepts to obtain sparser activations and demonstrable gains in DNN robustness. We use neuro-inspiration for extracting high-level design principles to guide software architectures for DNN training and inference, in contrast to recent work which attempts to directly apply known features from mammalian vision, such as [12] (Gabor filters and stochasticity) and [13] (DNN regularization based on neural activity measurements from mice). Our HaH-based approach to sparse activations can be realized with standard stochastic gradient training, unlike the iterative computations required for sparse coding with a reconstruction-based objective [14]. A recent study [15] shows the promise of weight and activation sparsity in the context of adversarial robustness. While our HaH-based framework enhances robustness via large, sparse activations, it is of interest to explore potential enhancements to its robustness by imposing weight sparsity.

2. Description

HaH training is governed by the HaH cost, which is incorporated into standard backpropagation based training. The HaH block is designed to take advantage of the features shaped by the HaH cost, introducing competition between neurons and enforcing scale-invariance.

2.1. HaH cost

We design the HaH cost to reward large elements of a given tensor along a dimension while penalizing the remainder of the tensor. While any DNN tensor, including weight tensors, layer outputs, and layer inputs, can be fed to the HaH cost to promote sparsity, we use it on convolutional layer outputs, since we wish to promote sparse, strong activations. After sorting the activations along the channel dimension for a given layer output, we define HaH cost as follows:

$$L_{block}(loc) = \frac{1}{K}\sum_{k=1}^{K} y^{(k)}(loc) - \lambda \frac{1}{N-K}\sum_{k=K+1}^{N} y^{(k)}(loc) \tag{1}$$

where y corresponds to the activations of k'th layer for a single image, loc corresponds to the spatial location of the activation, $block$ is the layer we want to have layer-wise HaH cost, and λ is a hyperparameter determining how much to emphasize the anti-Hebbian component.

2.2. HaH block

We design the HaH block such that it can replace a standard block in popular DNNs (e.g., VGG, ResNet, EfficientNet). The key components, shown in Fig. 1, are described in the following.

Implicit weight normalization: If we represent a given activation as a vector product of a filter weight $\mathbf{w}$ and input patch $\mathbf{x}$, then the output of a convolutional layer after implicit normalization is given by

$$y = \frac{\langle \mathbf{w}, \mathbf{x}\rangle}{\|\mathbf{w}\|_2} \tag{2}$$

which implicitly normalizes each filter's weight tensor to unit ℓ_2 norm without requiring weight decay, ℓ_2 regularization or explicitly setting each filter to unit norm $\|.\|_2 = 1$ after backpropagation.

Custom Divisive Normalization: We normalize each activation by the activity of a pool of neighboring neurons in the spatial dimension as suggested by [11]. However, we also suppress (or promote) the activation depending on the activity along the channel dimension. Specifically, the output of the kth divisive normalization block is given by

$$z_k(loc) = \frac{y_k(loc)}{\sigma M_{max} + (1-\sigma)M(loc)}\ ,\ k = 1,\dots,N \tag{3}$$

where $M(loc) = \frac{1}{N}\sum_{k=1}^{N} y_k(loc)$ is the local suppression field, and $M_{max} = \max_{loc} M(loc)$ is the maximum of this suppression field across the layer. The hyperparameter $0 \le \sigma \le 1$ can be separately tuned for each HaH block. For $\sigma > 0$, the use of M_{max} in the denominator helps us suppress weak "noise" blocks for which $M(loc)$ and $z_k(loc)$ are both small. Since all of these quantities scale with the layer input, the output remains scale-invariant.

Fractional Thresholding: We apply neuron-specific thresholding after divisive normalization, in order to keep only the significant outputs. The output of the kth neuron at location loc is given by

$$o_k(loc) = \begin{cases} z_k(loc) & \text{if } z_k(loc) \ge \tau_k \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where the channel-specific threshold τ_k is also input-dependent. For example, if we wish to limit the activation rate of each channel to 10%, we may set τ_k to the 90th percentile of the statistics of z_k.

2.3. Insight into HaH updates

In this section, we provide quick analytical and geometric insight into the HaH framework. By rewarding large activations y, the HaH cost targets learning weight tensors more aligned with input tensors. To see this, consider an activation $y = \langle \mathbf{w}, \mathbf{x}\rangle/\|\mathbf{w}\|_2$ which among the top K, and is therefore receiving a Hebbian update. The gradient along which $\mathbf{w}$ is to be updated can be computed as

$$\frac{\partial y}{\partial \mathbf{w}} = \frac{\langle \mathbf{w}, \mathbf{w}\rangle \mathbf{x} - \langle \mathbf{w}, \mathbf{x}\rangle \mathbf{w}}{\|\mathbf{w}\|_2^3} = \frac{\mathcal{P}_{\mathbf{w}}^{\perp}\mathbf{x}}{\|\mathbf{w}\|_2} \tag{5}$$

where $\mathcal{P}_{\mathbf{w}}^{\perp}\mathbf{x}$ denotes the projection of the input $\mathbf{x}$ orthogonal to the one-dimensional subspace spanned by $\mathbf{w}$. The update $\Delta \mathbf{w} = \eta \frac{\partial y}{\partial \mathbf{w}}$ is therefore proportional to this orthogonal component, and moving in this direction reduces the angle between $\mathbf{w}$ and $\mathbf{x}$, provided that η is small enough. We skip details due to lack of space, but note the following geometric interpretation. Because of implicit normalization, the original activation can be written as

$$y = \langle \mathbf{w}, \mathbf{x}\rangle/\|\mathbf{w}\|_2 = \|\mathbf{x}\|_2 \cos\theta \tag{6}$$

where θ is the angle between $\mathbf{w}$ and $\mathbf{x}$. By reducing θ via the weight update, we increase the implicitly normalized activation. Thus,

$$y_{new} = \langle \mathbf{w} + \Delta\mathbf{w}, \mathbf{x}\rangle/\|\mathbf{w} + \Delta\mathbf{w}\|_2 > y_{old} = \langle \mathbf{w}, \mathbf{x}\rangle/\|\mathbf{w}\|_2 \tag{7}$$

Note that exactly the opposite phenomenon occurs for an anti-Hebbian update: those weight vectors become less aligned with the input.

This procedure trains the neurons such that, during inference, the highly activated neurons at a given location tend to be better aligned with the input. Not only does this make these top activations more resilient to the impact of noise or perturbations, but larger activations also help attenuate the impact of noise on smaller activations by virtue of divisive normalization. In fact, many of these smaller noisy activations get eliminated via the thresholding applied after divisive normalization.

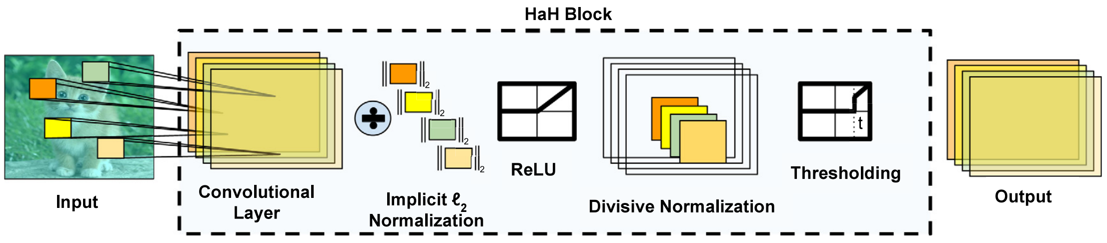

Fig. 1. HaH block consists of a convolutional layer, implicit ℓ_2 normalization, ReLU activation, divisive normalization, and a thresholding layer.

Table 1
CIFAR10 classification: Performance comparison between a HaH trained network and a standard network for different input corruptions on the test set. Clean corresponds to the test set without any corruption, Noisy corresponds to Gaussian noise injection on the test set with a standard deviation of 0.1, Adv(ℓ_∞) corresponds to adversarial attack on inputs bounded by the ℓ_∞ norm with budget $\epsilon = 2/255$, and Adv(ℓ_2) corresponds to the adversarial attack on inputs bounded by the ℓ_2 norm with budget $\epsilon = 0.25$. For all of the adversarial attacks, we use AutoAttack [16] which is an ensemble of parameter-free attacks.

	Clean	Noisy ($\sigma = 0.1$)	Adv (ℓ_∞) ($\epsilon = 2/255$)	Adv (ℓ_2) ($\epsilon = 0.25$)
Standard VGG16	**92.5%**	26.6%	10.4%	13.9%
HaH VGG16	87.3%	**64.0%**	**21.5%**	**27.6%**

2.4. Additional utilities

We provide additional utility functions to extract and use the layer outputs. We provide a wrapper class for *torch.nn.Module* which utilizes forward hooks from PyTorch to extract all the outputs and inputs of a specific layer type from a DNN while training. The wrapper expects a model and the layer type and makes the layer outputs and inputs accessible whenever an input is fed to the neural network.

3. Impact

Given the shortcomings of standard DNNs in terms of both interpretability and robustness, we believe it is time to explore how to better control the features generated within a DNN. We present our HaH framework as a first step in this direction. Our proposed approach plugs seamlessly into standard end-to-end training with the addition of HaH costs. The HaH blocks are defined so as to directly replace standard DNN blocks consisting of filters, ReLU and batch norm. Given the importance of understanding the features generated by each layer, the HaH module also includes several utilities for feature analysis.

Our initial results show promising gains in robustness. Table 1 compares baseline VGG16 against VGG16 with HaH layers for CIFAR10 image classification. These results show that a model armed with the HaH module (HaH blocks and HaH regularizer) performs significantly better against corruptions such as Gaussian noise and ℓ_p norm bounded adversarial attacks, while incurring a small decrease in accuracy for clean inputs. See [17] for more detail.

We hope that these promising initial results, and the release of our software, motivates the community to explore these ideas further. Although our experiments are limited to an image classification task, the HaH module, with small modifications, can be incorporated into any DNN.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by the Army Research Office, United States of America under grant W911NF-19-1-0053, and by the National Science Foundation, United States of America under grants CIF-1909320 and CIF-2224263.

References

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al., Solving rubik's cube with a robot hand, 2019, arXiv preprint arXiv:1910.07113.
[2] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, 2020, arXiv preprint arXiv:2005.14165.
[3] A.W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A.W. Nelson, A. Bridgland, et al., Improved protein structure prediction using potentials from deep learning, Nature 577 (7792) (2020) 706–710.
[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al., A general reinforcement learning algorithm that masters chess, shogi, and go through self-play, Science 362 (6419) (2018) 1140–1144.
[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8024–8035.
[6] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, 2009.
[7] K. Fukushima, S. Miyake, T. Ito, Neocognitron: A neural network model for a mechanism of visual pattern recognition, IEEE Trans. Syst. Man Cybern. SMC-13 (5) (1983) 826–834, http://dx.doi.org/10.1109/TSMC.1983.6313076.
[8] G. Amato, F. Carrara, F. Falchi, C. Gennaro, G. Lagani, Hebbian learning meets deep convolutional neural networks, in: E. Ricci, S. Rota Bulò, C. Snoek, O. Lanz, S. Messelodi, N. Sebe (Eds.), Image Analysis and Processing – ICIAP 2019, Springer International Publishing, Cham, ISBN: 978-3-030-30642-7, 2019, pp. 324–334.
[9] M. Carandini, D.J. Heeger, Normalization as a canonical neural computation, Nat. Rev. Neurosci. 13 (1) (2012) 51–62.
[10] M.F. Burg, S.A. Cadena, G.H. Denfield, E.Y. Walker, A.S. Tolias, M. Bethge, A.S. Ecker, Learning divisive normalization in primary visual cortex, PLoS Comput. Biol. 17 (6) (2021) e1009028.
[11] M. Ren, R. Liao, R. Urtasun, F.H. Sinz, R.S. Zemel, Normalizing the normalizers: Comparing and extending network normalization schemes, 2016, arXiv preprint arXiv:1611.04520.
[12] J. Dapello, T. Marques, M. Schrimpf, F. Geiger, D. Cox, J. DiCarlo, Simulating a primary visual cortex at the front of CNNs improves robustness to image perturbations, 2020, http://dx.doi.org/10.1101/2020.06.16.154542, BioRxiv, URL https://www.biorxiv.org/content/early/2020/06/17/2020.06.16.154542.
[13] Z. Li, W. Brendel, E. Walker, E. Cobos, T. Muhammad, J. Reimer, M. Bethge, F. Sinz, Z. Pitkow, A. Tolias, Learning from brains how to regularize machines, Adv. Neural Inf. Process. Syst. 32 (2019).
[14] B.A. Olshausen, D.J. Field, Sparse coding with an overcomplete basis set: A strategy employed by V1? Vis. Res. 37 (23) (1997).
[15] Y. Guo, C. Zhang, C. Zhang, Y. Chen, Sparse dnns with improved adversarial robustness, Adv. Neural Inf. Process. Syst. 31 (2018).
[16] F. Croce, M. Hein, Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks, in: International Conference on Machine Learning, PMLR, 2020, pp. 2206–2216.
[17] M. Cekic, C. Bakiskan, U. Madhow, Neuro-inspired deep neural networks with sparse, strong activations, 2022, in press, ICIP 2022, arXiv:2202.13074.