Hebbian Learning Meets Deep Convolutional Neural Networks*

Giuseppe Amato[1] [0000−0003−0171−4315], Fabio Carrara[1] [0000−0001−5014−5089], Fabrizio Falchi[1] [0000−0001−6258−5313], Claudio Gennaro[1] [0000−0002−0967−5050], and Gabriele Lagani[2]

[1] CNR Pisa, Italy
{giuseppe.amato, fabio.carrara, fabrizio.falchi, claudio.gennaro}
@isti.cnr.it
[2] University of Pisa, Italy
gabriele.lagani@gmail.com

Abstract. Neural networks are said to be biologically inspired since they mimic the behavior of real neurons. However, several processes in state-of-the-art neural networks, including Deep Convolutional Neural Networks (DCNN), are far from the ones found in animal brains. One relevant difference is the training process. In state-of-the-art artificial neural networks, the training process is based on backpropagation and Stochastic Gradient Descent (SGD) optimization. However, studies in neuroscience strongly suggest that this kind of processes does not occur in the biological brain. Rather, learning methods based on Spike-Timing-Dependent Plasticity (STDP) or the Hebbian learning rule seem to be more plausible, according to neuroscientists. In this paper, we investigate the use of the Hebbian learning rule when training Deep Neural Networks for image classification by proposing a novel weight update rule for shared kernels in DCNNs. We perform experiments using the CIFAR-10 dataset in which we employ Hebbian learning, along with SGD, to train parts of the model or whole networks for the task of image classification, and we discuss their performance thoroughly considering both effectiveness and efficiency aspects.

Keywords: Hebbian learning · deep learning · computer vision · convolutional neural networks

E. Ricci et al. (Eds.): ICIAP 2019, LNCS 11751, pp. 324-334, 2019.
Final authenticated publication: https://doi.org/10.1007/978-3-030-30642-7_29

* This work was partially supported by "Automatic Data and documents Analysis to enhance human-based processes" (ADA), CUP CIPE D55F17000290009, and by the AI4EU project, funded by the EC (H2020 - Contract n. 825619). We gratefully acknowledge the support of NVIDIA Corporation with the donation of a Tesla K40 GPU used for this research.

1 Introduction

Backpropagation is the most common learning rule for artificial neural networks. Despite being initially developed for biologically inspired artificial networks, it is commonly known by neuroscience that this process is unlikely to be implemented by nature. Seeking for more plausible models that mimic biological brains, researchers introduced several alternative learning rules for artificial networks.

In this work, we explore one of these alternative learning rules — *Hebbian learning* — in the context of modern deep neural networks for image classification. Specifically, the concept of Hebbian learning refers to a family of learning rules, inspired by biology, according to which the weight associated with a synapse increases proportionally to the values of the pre-synaptic and post-synaptic stimuli at a given instant of time [2,4].

Different variants of Hebbian rules can be found in the literature. In this work, we investigate two main Hebbian learning approaches, that is the *Winner-Takes-All* competition [3,12] and the *supervised Hebbian learning* solution. We apply these rule to train image classifiers that we extensively evaluate and compare with respect to standard models trained with Stochastic Gradient Descent (SGD). Moreover, we experiment with hybrid models in which we apply Hebbian and SGD updates to different parts of the network. Experiments on the CIFAR-10 dataset suggest that the Hebbian approach is adequate for training the lower and the higher layers of deep convolutional neural networks, while current results suggest that it has some limitations when used in the intermediate layers[1]. On the other hand, the Hebbian approach is much faster than Gradient Descent in terms of numbers of epochs required for training. Moreover, Hebbian update rules are inherently local and thus fully parallelizable in the backward/update phase, and we think that strategies to enhance the scalability of current models can benefit from this property. The main contributions of this work are:

- the use of Hebbian learning in DCNN, with a novel proposal for weight updates in shared kernels (Section 3.4);
- the definition of various hybrid deep neural networks, obtained combining SGD and Hebbian learning in the various network layers (Section 4);
- extensive experimentation and analysis of the results (Section 5).

The paper is organized as follows. Section 2 gives a brief overview of other works in this context. Section 3 introduces the Hebbian learning model. Section 4 discusses the deep network architecture that we defined and how we set the experiments to assess the performance of the approach. Section 5 discusses the experiments and the results. Section 6 concludes.

2 Related works

Recently, several works investigated the Hebbian rule for training neural networks for image classification. In [15], the authors propose a deep *Convolutional*

[1] For the implementation details about the experiments described in this document and the related source code, the reader is referred to [8,9].

Neural Network (CNN) architecture consisting of three convolutional layers, followed by an SVM classifier. The convolutional layers are trained, without supervision, to extract relevant features from the inputs. This technique was applied on different image datasets, among which CIFAR-10 [6], on which the algorithm achieved above 75% accuracy with a three-layer network.

In [10], the authors obtain the Hebbian weight update rules by minimizing an appropriate loss function, defined as the *strain loss*. Intuitively, they aim at minimizing how the differences, between the similarity among input vectors and output vectors, get distorted when moving from the input space and the output space. Also in this case, the authors use CIFAR-10 to perform the experiments, achieving accuracy up to 80% with a single layer followed by an SVM classifier [1].

In the above approaches, the Hebbian rule application remains limited to relatively shallow networks. On the other hand, in our work, we explore the possibility of applying Hebbian learning rules to deeper network architectures and discuss the opportunities and limitations arisen in this context.

3 Hebbian learning model

The Hebbian plasticity rule can be expressed as

$$\Delta w = \eta \, y(x,\, w) \, x \,, \tag{1}$$

where x is the vector of input signals on the neuron synapses, w is the weight vector associated with the neuron, η is the learning rate coefficient, Δw is the weight update vector, and $y(x,\, w)$ is the post-synaptic activation of the neuron — a function of the input and the weights that is assumed to be non-negative (e.g. a dot product followed by a ReLU or sigmoid activation).

3.1 Weight decay

Rule 1 only allows weights to grow, not to decrease. In order to prevent the weight vector from growing unbounded, Rule 1 is extended by introducing a *weight decay* (forgetting) term [2] $\gamma(x,\, w)$

$$\Delta w = \eta \, y(x,\, w) \, x - \gamma(x,\, w) \,. \tag{2}$$

When the weight decay term is $\gamma(x,\, w) = \eta \, y(x,\, w) \, w$ [4], we obtain

$$\Delta w = \eta \, y(x,\, w) \, (x - w) \,. \tag{3}$$

If we assume that $\eta \, y(x,\, w)$ is smaller than 1, the latter equation obtains the following physical interpretation: at each iteration, the weight vector is modified by taking a step towards the input, the size of the step being proportional to the similarity between the input and the weight vector, so that if a similar input is presented again in the future, the neuron will be more likely to produce a stronger response. If an input (or a cluster of similar inputs) is presented repeatedly to the neuron, the weight vector tends to converge towards it, eventually acting as a matching filter. In other words, the input is *memorized* in the synaptic weights. In this perspective, the neuron can be seen as an entity that, when stimulated with a frequent pattern, learns to recognize it.

3.2 Competitive Hebbian Learning: Winner Takes All

Equations 3 can be also used in the context of *competitive learning* [14,3,5,12]. When more than one neuron is involved in the learning process, it is possible to introduce some forms of lateral interaction in order to force different neurons to learn different patterns. A possible scheme of interaction is *Winner Takes All* (WTA) competition [3,12] which works as follows:

1. when input is presented to the network, the neurons start a competition;
2. the winner of the competition is the neuron whose weight vector is the closest to the input vector (according to some distance metric, e.g. angular distance [3] or euclidean distance [4]), while all the other neurons get inhibited;
3. the neurons update their weights according to Eq. 3, where y is set to 0 for the inhibited neurons and to 1 for the winner neuron.

3.3 Supervised Hebbian Learning

Hebbian learning is inherently an unsupervised approach to neural network training because each neuron updates its weight without relying on labels provided with the data. However, it is possible to use a simple trick to apply Hebbian rules in a supervised fashion: the *teacher neuron* technique [13,11] involves imposing a teacher signal on the output of the neurons that we want to train, thus replacing the output that they would naturally produce. By doing so and by applying a Hebbian learning rule, neurons adapt their weights in order to actually reproduce the desired output when the same input is provided.

Applying this technique to the output layer is straightforward, as the teacher signal coincides with the output target, but the choice of the teacher signal for supervised training of internal neurons is not trivial. Similarly to [15], we use the following technique to guide the neurons to develop a certain class-specificity in intermediate layers: we divide the kernels of a layer in as many groups as the number of classes and associate each group with a unique class; in addition, we also devote a set of kernels to be in common to all the classes; when an input of a given class is presented to the network, a high teacher signal $y = 1$ is provided to all the neurons sharing kernels that belong to the group corresponding to the given class, while the others receive a low teacher signal $y = 0$ (neurons sharing kernels associated with the set common to all the classes always receive a high teacher signal).

3.4 Hebbian rule with shared kernels in DCNN

Equation 3 allows computing Δw for each neuron in a given layer. Due to weight sharing in convolutional neural networks, different neurons in the same convolutional layer that shares the same kernel might be associated with different

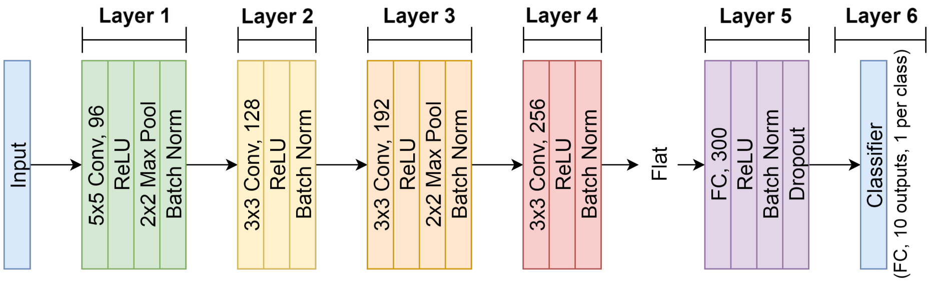
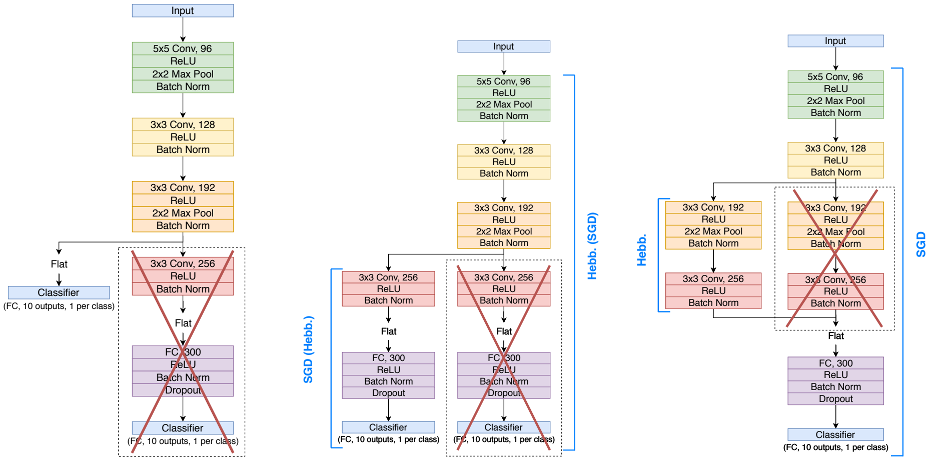


Fig. 1: Architecture of the Deep Convolutional Neural Network used in our experiments.

Δw's. In order to allow weight sharing in kernels of deep convolutional layers, we propose to perform an *aggregation* step in which the different Δw's, obtained at different spatial locations, are used to produce a global Δw_{agg} used for the update of the kernel. Δw_{agg} is computed as a weighted average of the Δw, where the weights are proportional to the coefficient that determines the step size (y when the basic Hebbian rule is used, 1 or 0 for winners and losers, respectively, when the WTA rule is used, and the teacher signal when supervised Hebbian learning is used).

4 Neural network architecture and experiment settings

To evaluate the Hebbian rule in deep learning, we designed a reference deep network architecture inspired to AlexNet [7]. The deep network structure, shown in Figure 1, is composed of four convolutional layers followed by a fully connected layer (layer 5). Layer 6 is a linear classifier with one output per class. We performed several experiments in which we combined Hebbian learning with SGD learning in various ways, and we measured the classification performance obtained on the CIFAR-10 dataset [6]. In the first and second experiment, discussed in Sections 5.1 and 5.2, we modified the architecture in Figure 1 as shown in Figure 2a. Specifically, we placed in turn Hebbian classifiers and SGD classifiers on top of feature extracted from various layers of, respectively, an SGD- and an Hebbian-trained network. In other words, the entire network was trained using a single approach (either Hebbian or SGD) and just the top layer (the classifier) was changed. In the third and fourth experiment, discussed in Sections 5.3 and 5.4, we placed SGD-trained layers on top of Hebbian-trained layers, and vice-versa, at various level of the network, as shown in Figure 2b. In the fifth experiment, discussed in Section 5.5, we put various Hebbian-trained layers in between SGD trained layers, as shown in Figure 2c. In all the experiments, when lower layers were trained with the Hebbian rule, we also pre-processed images with ZCA-whitening [6], which provided us with better performance.



(a) A classifier on top on the i-th layer of the network

(b) SGD layers on top of Hebbian-trained layers (and vice-versa)

(c) Hebbian trained layers in between SGD layers

Fig. 2: Architecture modifications applied in our experiments.

5 Experiments

As a baseline for the various experiments, we trained the defined network by applying the Stochastic Gradient Descent (SGD) algorithm to minimize the Cross-Entropy loss. Training was executed using the following configuration: SGD with Nesterov correction and momentum of 0.9, L2 penalty of 0.06, and a batch size of 64. The network was trained for 20 epochs on the first four of the five training batches of the CIFAR-10 dataset [6], corresponding to the first 40,000 samples, while the fifth batch, corresponding to the last 10,000 samples, was used for validation. Testing was performed on the CIFAR-10 test batch provided specifically for this purpose. Images were normalized to have zero mean and unit standard deviation. Early stopping was used so that, at the end of the 20 epochs, the network parameter configuration that we kept was the one achieving the highest accuracy. The learning rate was set to 10^{-3} for the first ten epochs, then halved every epoch for the next ten epochs. This baseline was used both for comparing with the performance obtained with Hebbian learning and to produce pre-trained SGD layers to be combined with Hebbian-trained layers. In the next sections, we discuss the results obtained by the various combinations of Hebbian-trained and SGD-trained layers, introduced in Section 4.

5.1 Hebbian vs SGD classifiers on SGD-trained layers

In this first experiment, we used the baseline trained network, discussed in Section 5, as a pre-processing module to extract features from an image, which

Table 1: Accuracy (%) of SGD- and Hebbian-trained classifiers built on top of various internal layers of an SGD- or Hebbian-trained network.

Classifier	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5
	SGD	SGD	SGD	SGD	SGD
SGD	60.71	66.30	72.39	82.69	84.95
Hebbian	46.58	56.59	67.79	82.18	84.88
	Hebbian	Hebbian	Hebbian	Hebbian	Hebbian
SGD	63.92	63.81	58.28	52.99	41.78

were then fed to a classifier as shown in Figure 2a. We compared both a classifier trained using SGD and the Hebbian rule.

To perform an exhaustive test, we measured the performance obtained by placing (and training) a classifier after every layer of the network. The SGD classifiers were trained with the same parameters used for the baseline, except that the L2 penalty is reduced to $5 \cdot 10^{-4}$. The Hebbian classifiers were trained with learning rate 0.1, the similarity between neuron inputs and weight vectors was measured in terms of angular distance (lower angular distance means higher similarity and vice-versa), and the activation function adopted was simply the scalar product between the input and the normalized weight vectors.

Table 1 (top two rows) reports the results of tested configurations. We can see that classifiers placed on top of higher layers and trained with the Hebbian rule achieve accuracy values practically overlapped to those of an SGD classifier. On the other hand, Hebbian classifiers placed on top of lower layers obtain a lower accuracy. However, it is worth mentioning that Hebbian classifiers can be trained in just a few epochs (usually one or two in our experiments), while classifiers trained with Gradient Descent need from five to ten epochs to converge.

5.2 SGD classifiers on Hebbian-trained layers

Experiments discussed in this section are complementary to those discussed above. The entire deep network is trained with Hebbian approach, and the features extracted from the various layers are fed to an SGD classifier.

The goal is to evaluate the performance of the Hebbian approach for training the feature extraction layers of the network. To train the network, we set the learning rate of the Hebbian weight update rule to 0.1. The similarity between neuron inputs and weight vectors was measured in terms of angular distance (lower angular distance means higher similarity and vice-versa), and the activation function used for neurons of Hebbian hidden layers was the cosine similarity between input and weight vector, followed by the ReLU non-linearity. We used the WTA approach (Section 3.2) for updating the weight of the internal layers, we applied ZCA-whitening (see Section 4) to the input images. We imposed a teacher signal on the layers of the network trained with Hebbian approach, even if they are not classification layers, according to the logic discussed Section 3.3.

In our experiments, we used 96 common kernels at layer 1, 8 kernels per class plus 16 common kernels at layer 2, 16 kernels per class plus 32 common kernels at layer 3, 24 kernels per class plus 16 common kernels at layer 4, and 28 kernels per class plus 20 common kernels at layer 5.

Table 1 (bottom row) reports the achieved resultsIt can be observed that the accuracy slowly degrades with the number of layers. We conclude that training with the basic Hebbian rule has some disadvantages when the depth of the network increases, while it is still competitive when the layers are less than 4.

5.3 Hybrid: SGD layers on top of Hebbian layers

In the previous experiments, the entire network was trained using a single approach and we changed just the classifier. Here, we created hybrid networks where the bottom layers were trained with the Hebbian rule and the top layers were trained with SGD. The goal of the following set of experiments is to assess the limits within which layers trained with the Hebbian algorithm can replace layers trained with SGD.

The architectures of these hybrid networks are as in Figure 2b, where a layer was chosen as the *splitting point* between Hebbian trained layers and SGD trained ones. All the layers from the first to the fifth were used in different experiments as splitting points. The features extracted from the Hebbian-trained layers up to the splitting point were fed to the remaining network which was re-trained from scratch with SGD on the Hebbian feature maps provided as input. During this re-training process, the Hebbian-trained network was kept in evaluation mode and its parameters were left untouched. As before, also in this case, when training the Hebbian layers, we used the WTA approach, and the images were processed with ZCA-whitening.

Table 2 (second group) shows the accuracy on CIFAR-10 of a network composed of bottom layers trained with the Hebbian algorithm and top layers trained with Gradient Descent. In addition, the accuracy of the baseline fully trained with Gradient Descent and that of the same network fully trained with the Hebbian algorithm are also shown for comparisons (Table 2, first group). We also report the results of a network where the bottom layers are left completely untrained (randomly initialized), so that it is possible to assess whether Hebbian training gives a positive contribution w.r.t. pure randomness or it is completely destructive (Table 2, third group).

It can be observed that the first layer trained with the Hebbian algorithm can perfectly replace the corresponding Gradient Descent layer. There is a certain accuracy loss when also the second layer is switched to Hebbian learning, however it is still competitive. Accuracy heavily degrades when further layers are set to Hebbian learning. As expected, Hebbian training is better than untrained layers.

5.4 Hybrid: Hebbian layers on top of SGD layers

The experiments presented in this section complement those discussed in Section 5.3: bottom layers are SGD-trained, while the top layers are Hebbian-trained. Table 2 (fourth group) shows the accuracy on CIFAR-10 of a network composed of bottom layers trained with Gradient Descent and top layers trained with the Hebbian algorithm. The table compares the accuracy of hybrid networks obtained by choosing layers 1, 2, ..., 5 to be the splitting point, i.e. all the layers on top of the first, second, ..., fifth (respectively) were trained with the Hebbian algorithm, while the rest of the network was kept in evaluation mode and its parameters were left untouched.

In this case, we can observe that the last layer trained with the Hebbian algorithm (i.e. the supervised Hebbian classifier) can perfectly replace the corresponding Gradient Descent layer. The fifth layer can also be replaced with a Hebbian layer with only a minimal accuracy decrease. We observe a slightly higher accuracy loss when also the fourth layer is replaced. Accuracy heavily degrades when further layers are replaced.

5.5 Hybrid: Hebbian layers between SGD layers

Finally, more complex configurations were also considered in which network layers were divided into three groups: bottom layers, middle layers, and top layers, as shown in Figure 2c. The bottom layers were trained with SGD, the middle layers with the Hebbian algorithm and the top layers again with SGD.

Table 2 (fifth to eighth group) shows the accuracy on CIFAR-10 of these configurations. The table compares the accuracy of hybrid networks obtained by choosing various combinations of inner layers to be converted to Hebbian training. It can be observed that a minor accuracy loss occurs when a single inner layer is switched to a Hebbian equivalent. A slightly larger accuracy loss occurs when two layers are replaced. Specifically, lower layers are more susceptible than higher layers. Accuracy degrades more when further layers are replaced. However, the replacement of inner layers has more influence on the resulting accuracy than the replacement of outer layers.

6 Conclusion

We explored the use of the Hebbian rules for training a deep convolutional neural network for image classification. We extended the Hebbian weight update rule to convolutional layers, and we tested various combinations of Hebbian and SGD learning to investigate the advantages and disadvantages of mixing the two.

Experiments on CIFAR-10 showed that the Hebbian algorithm can be effectively used to train a few layers of a neural network, but the performance decreases when more layers are involved. In particular, the Hebbian algorithm is adequate for training the lower and the higher layers of a neural network, but not for the intermediate layers, which lead to the main performance drops when switched from Gradient Descent to its Hebbian equivalent.

On the other hand, the algorithm is advantageous with respect to Gradient Descent in terms of the number of epochs needed for training. In fact, a stack of Hebbian layers (for instance the top portion of a hybridly-trained network, or

Table 2: Accuracy (%) on the CIFAR-10 test set of various configurations of learning rules. Columns 'L1-L5' and 'Classif' report the learning rule (G gradient descent, H Hebbian rule, R random init.) used to train respectively layers 1 to 5 and the final classifier.

Group	Description	L1	L2	L3	L4	L5	Classif	Accuracy
1	Full SGD	G	G	G	G	G	G	84.95
	Full Hebbian	H	H	H	H	H	H	28.59
2	1-bottom Hebbian	H	G	G	G	G	G	84.93
	2-bottom Hebbian	H	H	G	G	G	G	78.61
	3-bottom Hebbian	H	H	H	G	G	G	67.87
	4-bottom Hebbian	H	H	H	H	G	G	57.56
	5-bottom Hebbian	H	H	H	H	H	G	41.78
3	1-bottom Random	R	G	G	G	G	G	80.19
	2-bottom Random	R	R	G	G	G	G	71.87
	3-bottom Random	R	R	R	G	G	G	54.96
	4-bottom Random	R	R	R	R	G	G	45.56
	5-bottom Random	R	R	R	R	R	G	9.52
4	1-top Hebbian	G	G	G	G	G	H	84.88
	2-top Hebbian	G	G	G	G	H	H	83.16
	3-top Hebbian	G	G	G	H	H	H	71.18
	4-top Hebbian	G	G	H	H	H	H	50.43
	5-top Hebbian	G	H	H	H	H	H	32.95
5	Layer 1 Hebbian	G	H	G	G	G	G	80.36
	Layer 2 Hebbian	G	G	H	G	G	G	80.68
	Layer 3 Hebbian	G	G	G	H	G	G	80.92
	Layer 4 Hebbian	G	G	G	G	H	G	83.75
6	Layer 2-3 Hebbian	G	H	H	G	G	G	72.12
	Layer 3-4 Hebbian	G	G	H	H	G	G	74.98
	Layer 4-5 Hebbian	G	G	G	H	H	G	76.86
7	Layer 2-3-4 Hebbian	G	H	H	H	G	G	63.68
	Layer 3-4-5 Hebbian	G	G	H	H	H	G	62.43
8	Layer 2-3-4-5 Hebbian	G	H	H	H	H	G	47.24

even a full Hebbian network), can be trained in fewer epochs (e.g. one or two on the architecture we used) than a network trained with SGD, which needs twenty epochs. Although the performance of deep full Hebbian networks is not yet comparable to the one of gradient-based models, according to our results, current Hebbian learning approaches could be efficiently and effectively adopted in scenarios like fine-tuning and transfer learning, where Hebbian layers on top of pre-trained SGD layers can be re-trained fast and effectively.

Moreover, the local nature of the Hebbian rule potentially provides huge speed-ups for large models with respect to backpropagation, thus encouraging further research to improve current approaches.

References

1. Yanis Bahroun and Andrea Soltoggio. Online representation learning with multi-layer hebbian networks for image classification tasks. *arXiv preprint arXiv:1702.06456*, 2017.
2. Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge university press, 2002.
3. Stephen Grossberg. Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors. *Biological cybernetics*, 23(3):121–134, 1976.
4. Simon Haykin. *Neural networks and learning machines.* Pearson, 3 edition, 2009.
5. Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological cybernetics*, 43(1):59–69, 1982.
6. Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.
7. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.
8. Gabriele Lagani. Hebbian learning algorithms for training convolutional neural networks. Master's thesis, School of Engineering, University of Pisa, Italy, 2019.
9. Gabriele Lagani. Hebbian learning algorithms for training convolutional neural networks - project code. https://github.com/GabrieleLagani/HebbianLearningThesis, 2019.
10. Cengiz Pehlevan, Tao Hu, and Dmitri B Chklovskii. A hebbian/anti-hebbian neural network for linear subspace learning: A derivation from multidimensional scaling of streaming data. *Neural computation*, 27(7):1461–1495, 2015.
11. Filip Ponulak. Resume-new supervised learning method for spiking neural networks. technical report. *Institute of Control and Information Engineering, Poznan University of Technology*, 2005.
12. David E Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.
13. Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. pages 1999–2006, 2017.
14. Chr Von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100, 1973.
15. Aseem Wadhwa and Upamanyu Madhow. Learning sparse, distributed representations using the hebbian principle. *arXiv preprint arXiv:1611.04228*, 2016.