A bio-inspired learning rule for deep architectures

Code at `https://github.com/Einlar/QLS-Project`

Francesco Manzali

University of Padova

March 22, 2021

Table of contents

- **Machine Learning** (ML) is the study of algorithms that **improve** automatically by using **data**.
- The **human brain** is a striking example of physical system capable of learning. Algorithms inspired by biology, such as *Artificial Neural Networks*, are now widely used in ML.
- In the 2010s, significant advances in computational power and data availability have lead to the rise of **Deep Learning**.

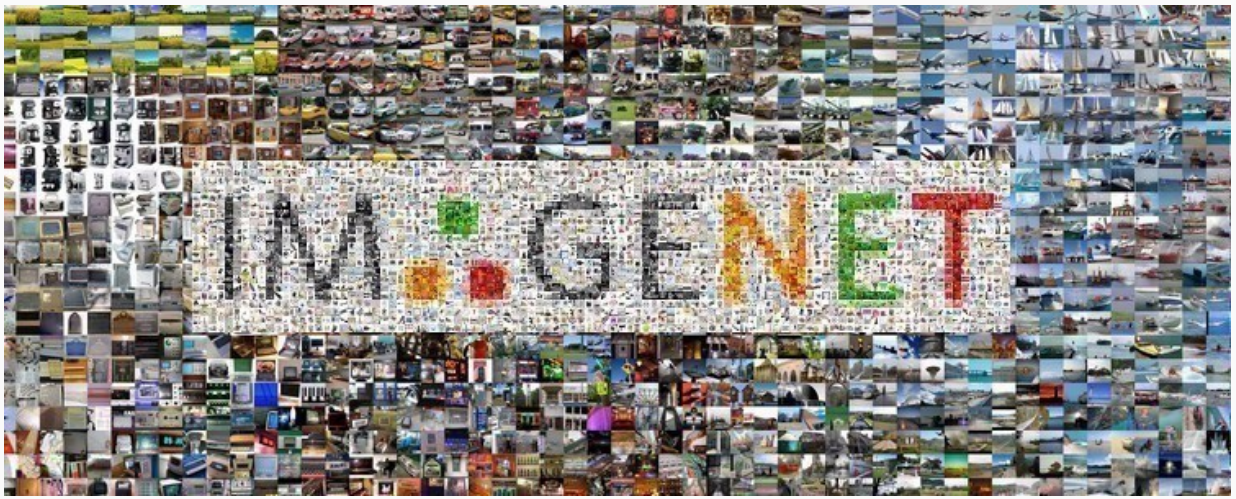
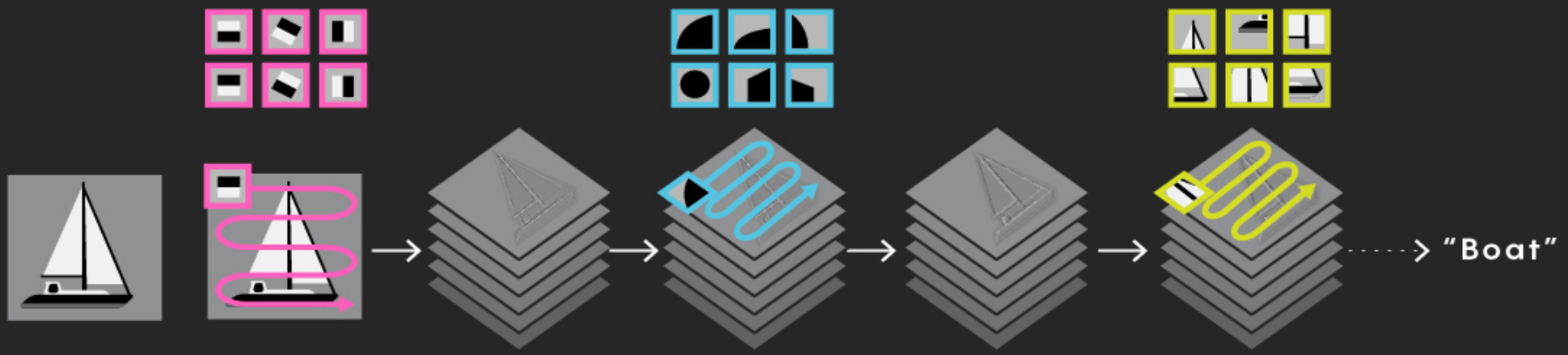


Figure 1 – **Imagenet** (`http://image-net.org/`, 2009) is one of the reference datasets for image classification. It consists of over 14m images, manually labelled into $\sim$ 20k categories. As of 2020, the state-of-the-art model on Imagenet is EfficientNet-L2 (`https://paperswithcode.com/paper/meta-pseudo-labels`), reaching a 98.8% top-5 accuracy (higher than humans!).

How Convolutional Neural Networks See



Input image

Convolution using edge-detecting filters

Resulting feature maps

Convolution using simple feature-detecting filters

Resulting feature maps

Convolution using complex feature-detecting filters

1 Feature-detecting filters slide across an input image, and the degree of match between each filter and each position in the image is logged, producing a set of feature maps.

2 After some processing, the feature maps are convolved again, using filters that detect increasingly high-level features.

3 Eventually, the network learns to recognize and correctly classify input images.

How to **learn** patterns?

- Most networks still use the **backpropagation** algorithm (1970).

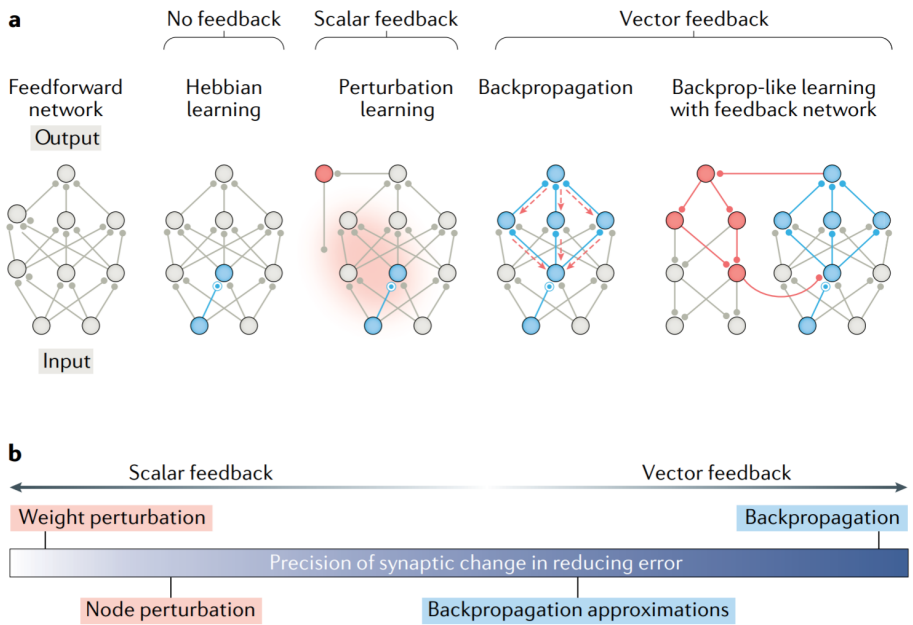


Figure 2 – Learning rule and flow of information. Neurons/weights undergoing learning are colored in blue [1].

Change in a weight depends on the activation of **all** neurons located after it. Backpropagation requires an **arbitrarily long** transfer of information.

The unreasonable effectiveness of backpropagation methods has **stalled** research on alternative learning rules after 2014.

Can a **different**, **bio-inspired** rule achieve **comparable results** to backpropagation?

"Unsupervised learning by competing hidden units", 2019, Dmitry Krotov and John J. Hopfield [2]

- Consider a **local** rule: change in a weight is determined only by the neurons it is connected to.
- (Mostly) **feedforward** network, with no top-down connections. All layers except the last are trained in an **unsupervised** way.

BioLearning vs Biology

The point is not to exactly model neurons, but to take **inspiration** from them.

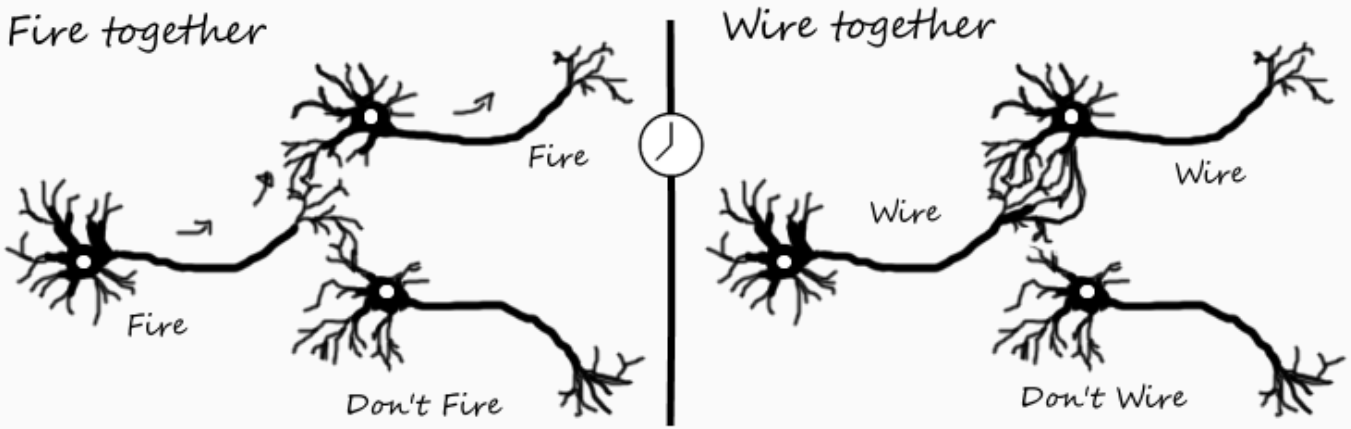
- Dale's law is not respected
- Top-down connections exist in brains.

BioLearning vs Backpropagation

- BioLearning is **local**, while backpropagatio is **not**.
- Backpropagation may be used in **unsupervised** learning.

1.a **Locality**. Changes in a synapse w_{AB} connecting A to B should be *locally determined*, i.e. proportional to the activity of the pre-synaptic cell (A) and (a function of) the activity of the post-synaptic cell (B).

Hebb's rule: *"when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" - D. O. Hebb [3]*



1.b Changes in a synapse w_{AB} may be **positive** (Long Term Potentiation) or **negative** (Long Term Depression).

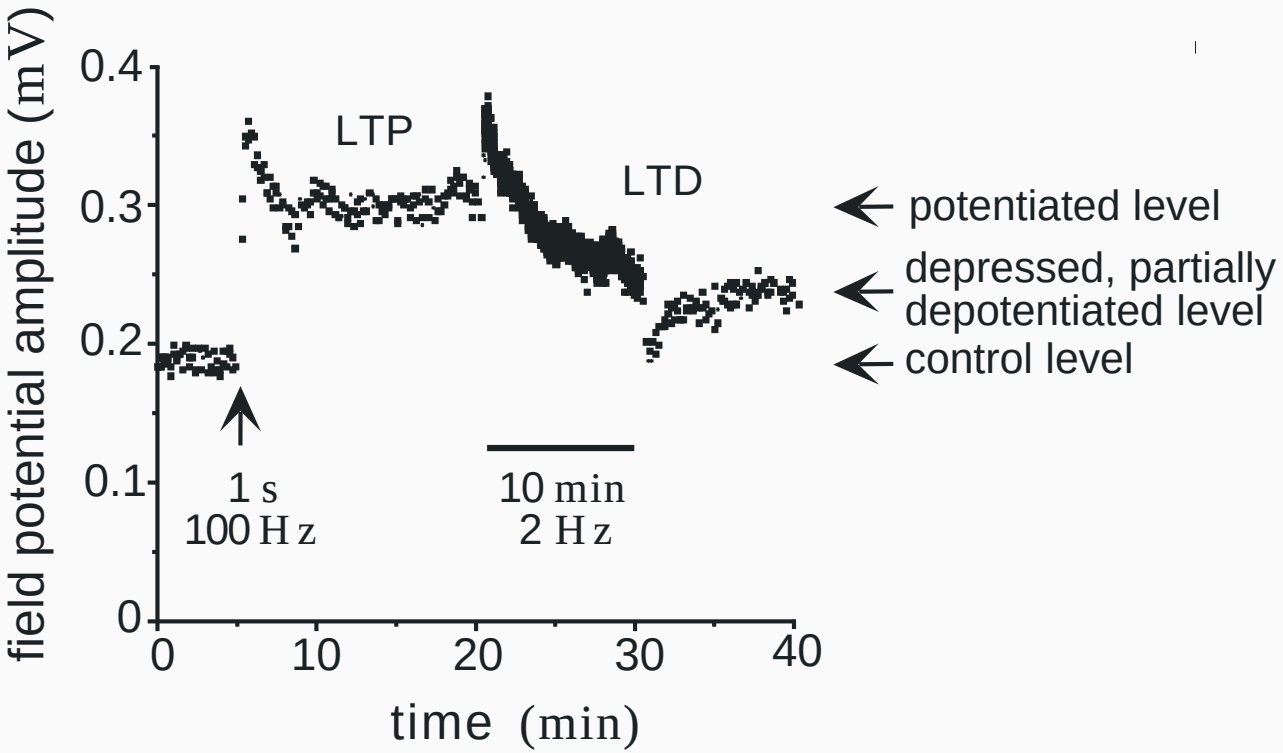
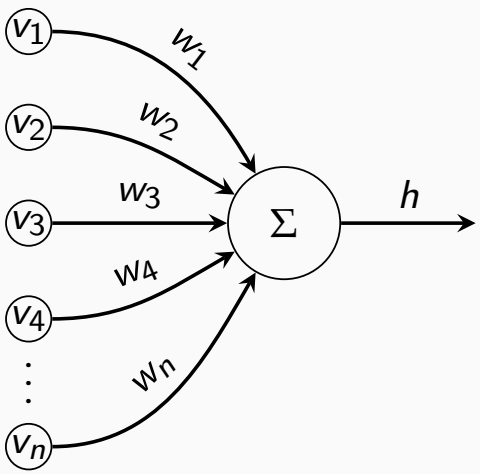


Figure 3 – Amplitude of the field potentials evoked by a constant amplitude stimulus in a rat hippocampal slice. At $\sim 5\,\mathrm{min}$, a *tetanus*, i.e. a short high-frequency stimulus ($1\,\mathrm{s}$, $100\,\mathrm{Hz}$) is applied. Then, the constant amplitude from before is re-applied. After a transient (Post-Tetanic Potentiation), the response stabilizes at a *higher* value (LTP). Applying a long but low-frequency stimulus ($10\,\mathrm{min}$, $2\,\mathrm{Hz}$) has the opposite effect, *decreasing* the response (LTD) [4, p. 291].

2. **Competition**. Weights cannot be all arbitrarily high, since resources for creating synapses are finite and shared (an organism must maintain *homeostasis*).
   Thus, rising a weight means lowering another, i.e. weights *compete* with each other.
   Mathematically, this means that weights should be **normalized**.
3. **Sparsity**. It is more efficient to have few *high* weights and many *zero* weights, than all *low* weights. Only the "important" connections should be maintained.
   Mathematically, the normalization condition from before should **emphasize** large weights more than low ones.
4. **Selectivity**. Different neurons should be strongly activated by different patterns. There should not be many neurons all reacting to the same pattern.
   In other words, if a neuron reacts strongly to a pattern, other neurons should not "specialize" on it.

One linear neuron with Hebbian weights



We need two rules:

- **Neuron dynamics**: use a simple neuron that computes a weighted sum of inputs
- **Synaptic plasticity**: implement **locality** (1.a) with the most basic Hebb's rule

Let $\boldsymbol{v} = (v_1, \dots, v_n)^T$, $\boldsymbol{w} = (w_1, \dots, w_n)^T$.

$$\begin{cases} \tau \dfrac{\mathrm{d}h}{\mathrm{d}t} = -h + \boldsymbol{w} \cdot \boldsymbol{v} & \text{(Linear neuron)} \\ \tau_w \dfrac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = h\,\boldsymbol{v} & \text{(Hebb's Rule)} \end{cases} \tag{1}$$

Assume $\tau_w \gg \tau$, so that **quasi-stationary** approximation may be used: $h^* = \boldsymbol{w} \cdot \boldsymbol{v}$.

! Satisfies (1.a), but weights **grow** indefinitely!

$$\frac{\mathrm{d}\|\boldsymbol{w}\|_2^2}{\mathrm{d}t} \equiv \frac{\mathrm{d}\boldsymbol{w}\cdot\boldsymbol{w}}{\mathrm{d}t} = 2\boldsymbol{w}\cdot\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} \underset{(1)}{=} 2\boldsymbol{w}\cdot\frac{1}{\tau}(h\boldsymbol{v}) \underset{\tau_L \gg \tau}{=} \frac{2}{\tau}h^2 \geq 0$$

- One discrete step of Hebb's rule is:

$$w_i(t+1) - w_i(t) = \frac{1}{\tau_w} h\, v_i(t)$$

- Rename $\tau_w^{-1} \equiv \gamma$ and rearrange into:

$$w_i(t+1) = w_i(t) + \gamma\, h\, v_i(t)$$

- **Normalize** with Euclidean norm:

$$w_i(t+1) = \frac{w_i(t) + \gamma h(t) v_i(t)}{\underbrace{\|\boldsymbol{w}(t) + \gamma h(t) \boldsymbol{v}(t)\|_2}_{\|\boldsymbol{w}(t+1)\|_2}}; \qquad \|\boldsymbol{a}\|_2 \equiv \sqrt{\sum_{i=1}^{n} a_i^2}$$

- Suppose $\gamma \ll 1$ (i.e. $\tau_w \gg 1$), meaning that weights change slowly. Then expand to first order in γ:

$$\Delta w_i \equiv w_i(t+1) - w_i(t) = \gamma h[v_i - h w_i] + O(\gamma^2) \qquad \textbf{Oja rule } [5]$$

Let's verify the **normalization**.

- Start from the Oja Rule in vector form:

$$\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = \gamma h(\boldsymbol{v} - h\boldsymbol{w})$$

- Now $\boldsymbol{w}$ tends to the surface of an n-dimensional Euclidean unit sphere:

$$\frac{\mathrm{d}\|\boldsymbol{w}\|_2^2}{\mathrm{d}t} = 2\boldsymbol{w} \cdot \gamma h[\boldsymbol{v} - h\boldsymbol{w}] = 2(\underbrace{\boldsymbol{w} \cdot \boldsymbol{v}}_{h}\gamma h - \gamma h^2 \boldsymbol{w} \cdot \boldsymbol{w}) = 2\gamma h^2(1 - \|\boldsymbol{w}\|_2^2)$$

- The *radius* of this sphere can be adjusted by adding a factor:

$$\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = \gamma h(R^2\boldsymbol{v} - h\boldsymbol{w})$$

So that, at stationarity:

$$|w_1|^2 + \cdots + |w_n|^2 = R^2 \qquad \textbf{Homeostatic Constraint}$$

But what **kind** of weights will be learned?

- Start from the stochastic equation for the Oja Rule:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \gamma h[\boldsymbol{v} - h\boldsymbol{w}] \xrightarrow[h=\boldsymbol{w}\cdot\boldsymbol{v}]{} \boldsymbol{w}(t+1) = \boldsymbol{w} + \gamma[\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{w} - (\boldsymbol{w}^T\boldsymbol{v}\boldsymbol{v}^T\boldsymbol{w})\boldsymbol{w}]$$

- Averaging ($\boldsymbol{v}$ and $\boldsymbol{m}$ stochastic and statistically independent):

$$\mathbb{E}[\boldsymbol{w}(t+1)|\boldsymbol{w}(t)] = \boldsymbol{w} + \gamma[\mathrm{C}\boldsymbol{w} - (\boldsymbol{w}^T\mathrm{C}\boldsymbol{w})\boldsymbol{w}] + O(\gamma^2); \quad \mathrm{C} = \mathbb{E}[\boldsymbol{v}\boldsymbol{v}^T] \quad (2)$$

Stochastic Approximation Theory

If the distribution of $\boldsymbol{v}$ satisfies some (weak) assumptions, and $\gamma \to 0$ as $1/t$, then (2) approximated by a (deterministic) differential equation:

$$\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = \mathrm{C}\boldsymbol{w}(t) - (\boldsymbol{w}^T\mathrm{C}\boldsymbol{w})\boldsymbol{w} \underset{\text{Stationarity}}{\Rightarrow} \mathrm{C}\boldsymbol{w} = (\boldsymbol{w}^T\mathrm{C}\boldsymbol{w})\boldsymbol{w}$$

So $\boldsymbol{w}$ converges to an **eigenvector** (the dominant, in fact) of the covariance matrix C, and so h is a **principal component** of the inputs.

- The model now implements **locality** (1.a) and **competition** (2):

$$\begin{cases} \tau \dfrac{\mathrm{d}h}{\mathrm{d}t} = -h + \boldsymbol{w} \cdot \boldsymbol{v} & \text{Linear neuron} \\ \tau_w \dfrac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = h[R^2 \boldsymbol{v} - h\,\boldsymbol{w}] & \text{Oja Rule} \end{cases}$$

- To add **sparsity** (3) it suffices to replace the Euclidean norm with a generic Lebesgue p-norm (any type of normalization works for the Oja Rule):

$$\begin{cases} \tau \dfrac{\mathrm{d}h}{\mathrm{d}t} = -h + \langle \boldsymbol{w}, \boldsymbol{v} \rangle_p \\ \tau_w \dfrac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = h[R^p \boldsymbol{v} - h\,\boldsymbol{w}] \end{cases} \qquad \langle \boldsymbol{w}, \boldsymbol{v} \rangle_p \equiv \sum_{i=1}^{n} \mathrm{sgn}(w_i) |w_i|^{p-1} v_i$$

$$\left[\langle \boldsymbol{a}, \boldsymbol{b} \rangle \equiv \sum_{i,j} \eta_{ij} a_i b_j; \quad \eta_{ij} = |w_i|^{p-2} \delta_{ij} \right]$$

- Now weights converge to a sphere of radius R defined with the L^p norm:

$$|w_1|^p + \cdots + |w_n|^p = R^p \qquad \textbf{Homeostatic Constraint} \qquad (3)$$

  For $p > 2$, only high weights matter in (3).

Model LTD (1.b)

If the inputs $\boldsymbol{v}$ do not activate "much" a neuron, weights $\boldsymbol{w}$ should **decrease** (LTD).

- This idea is implemented in the **BCM learning rule** [6], which is basically Hebb's rule with a *threshold* on the output:

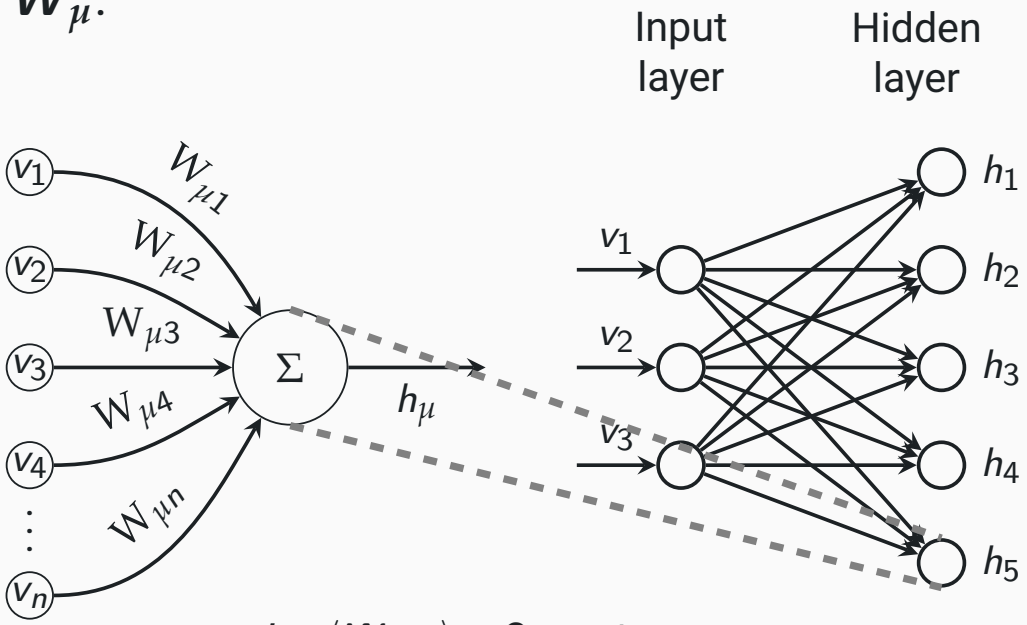
$$\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = \underbrace{(h - \theta_m)}_{\text{Threshold}}\, h\, \boldsymbol{v} - \overbrace{\epsilon \boldsymbol{w}}^{\text{Weight decay}}$$

- The same idea is used to modify the Oja's rule:

$$\tau_w \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = \underbrace{g(h)}_{\text{Activation function}}[R^p \boldsymbol{v} - h\, \boldsymbol{w}]; \qquad g(h) = \begin{cases} 0 & h < 0 \\ -\Delta & 0 \leq h < h_* \\ 1 & h_* \leq h \end{cases}$$

The function g implements **temporal competition** between patterns: each neuron learns to differentiate between patterns that strongly drive it, and patterns that do not.

Before implementing **selectivity** (4), we need to consider more than one neuron. Consider m hidden units. The weights are collected in an $m \times n$ matrix W, with m rows denoted as $\boldsymbol{W}_\mu$.



$I_\mu \equiv \langle \boldsymbol{W}_\mu, \boldsymbol{v} \rangle_p$: Currents

$$\begin{cases} \tau \dfrac{\mathrm{d}h_\mu}{\mathrm{d}t} = -h_\mu + \overbrace{\displaystyle\sum_{i=1}^{n} \mathrm{sgn}(\mathrm{W}_{\mu i}) |\mathrm{W}_{\mu i}|^{p-1} v_i}^{} & \text{Linear neuron with } p\text{-norm} \\ \tau_w \dfrac{\mathrm{dW}_{\mu i}}{\mathrm{d}t} = g(h_\mu)\left[R^p v_i - h_\mu \mathrm{W}_{\mu i}\right] & \text{Oja's rule + BCM} \end{cases}$$

Model Selectivity (4)

Selectivity (4) is implemented by adding **global lateral inhibition** between neurons. T That is, if a neuron μ is active ($h_\mu > 0$), it makes all other neurons *less* active (spatial competition).

$$\begin{cases} \tau \dfrac{\mathrm{d}h_\mu}{\mathrm{d}t} = I_\mu - h_\mu \overbrace{-w_{\text{inh}} \sum_{\nu \neq \mu} \max(h_\nu, 0)}^{\text{Global lateral inhibition}} \\ \tau_w \dfrac{\mathrm{d}W_{\mu i}}{\mathrm{d}t} = g(h_\mu)\left[R^p v_i - h_\mu W_{\mu i}\right] \end{cases} \qquad I_\mu = \langle \boldsymbol{W}_\mu, \boldsymbol{v} \rangle_p$$

This model implements **all** the requirements!

- **Locality** (1.a): Hebb's rule. LTD (1.b) is allowed by BCM activation $g(h_\mu)$.
- **Competition** (2): Oja's rule, homeostatic constraint.
- **Sparsity** (3): Lebesgue p-norm.
- **Selectivity** (4): Global lateral inhibition.

Simulate: $\tau \frac{\mathrm{d}h_\mu}{\mathrm{d}t} = \overbrace{\langle \boldsymbol{W}_\mu, \boldsymbol{v} \rangle_p}^{I_\mu} - h_\mu - w_{\text{inh}} \sum_{\nu \neq \mu} \max(h_\nu, 0)$

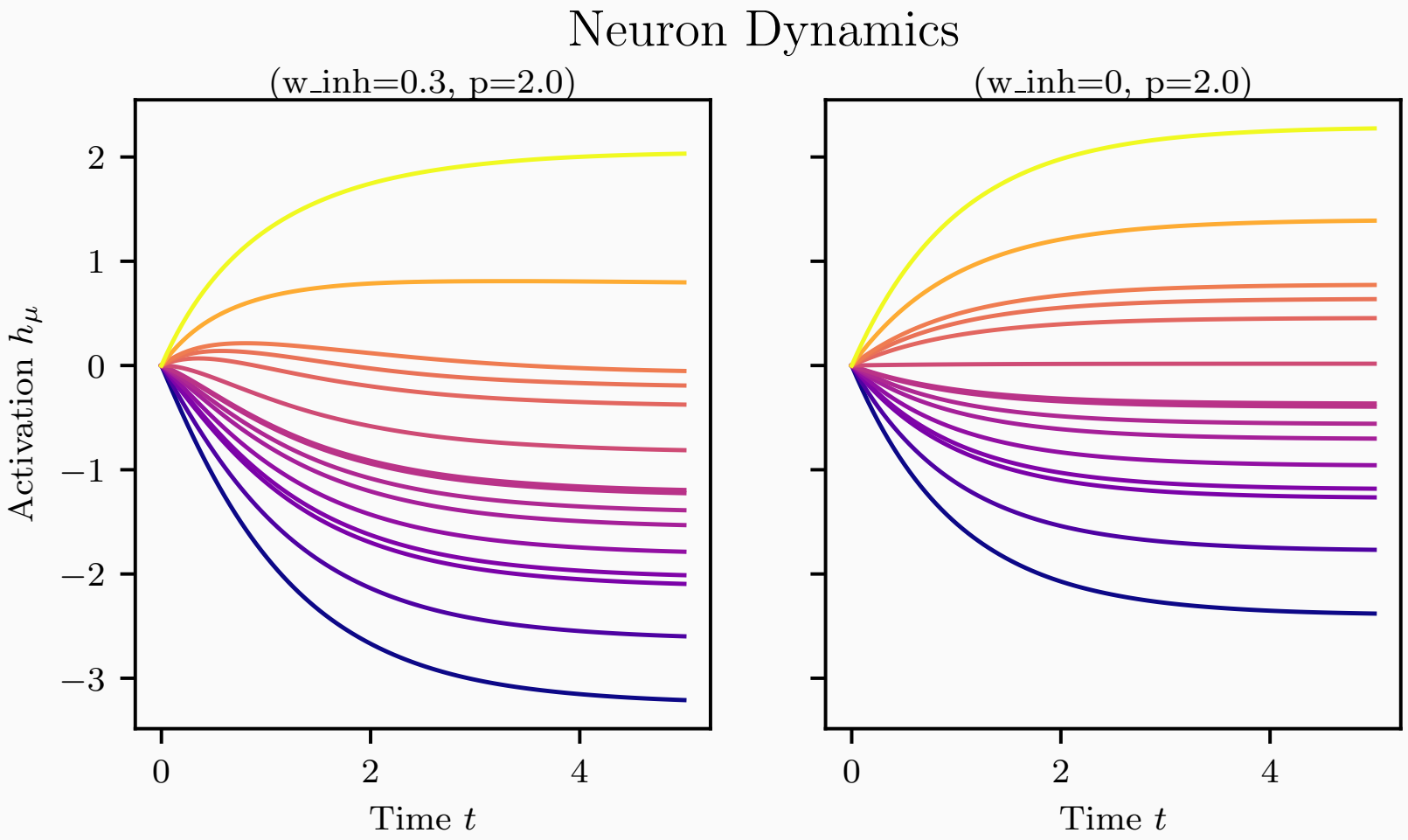


Figure 4 – Activations of $m = 15$ neurons, with $\mathrm{W} \sim \mathcal{N}(0, 1)$, and a 2-dimensional input $\boldsymbol{v} \sim \mathcal{U}(0, 1)$. When $w_{\text{inh}} = 0$, $h^*_\mu = I_\mu$, but if $w_{\text{inh}} > 0$, $h^*_\mu < I_\mu$. However, their **ranking order** is mantained.

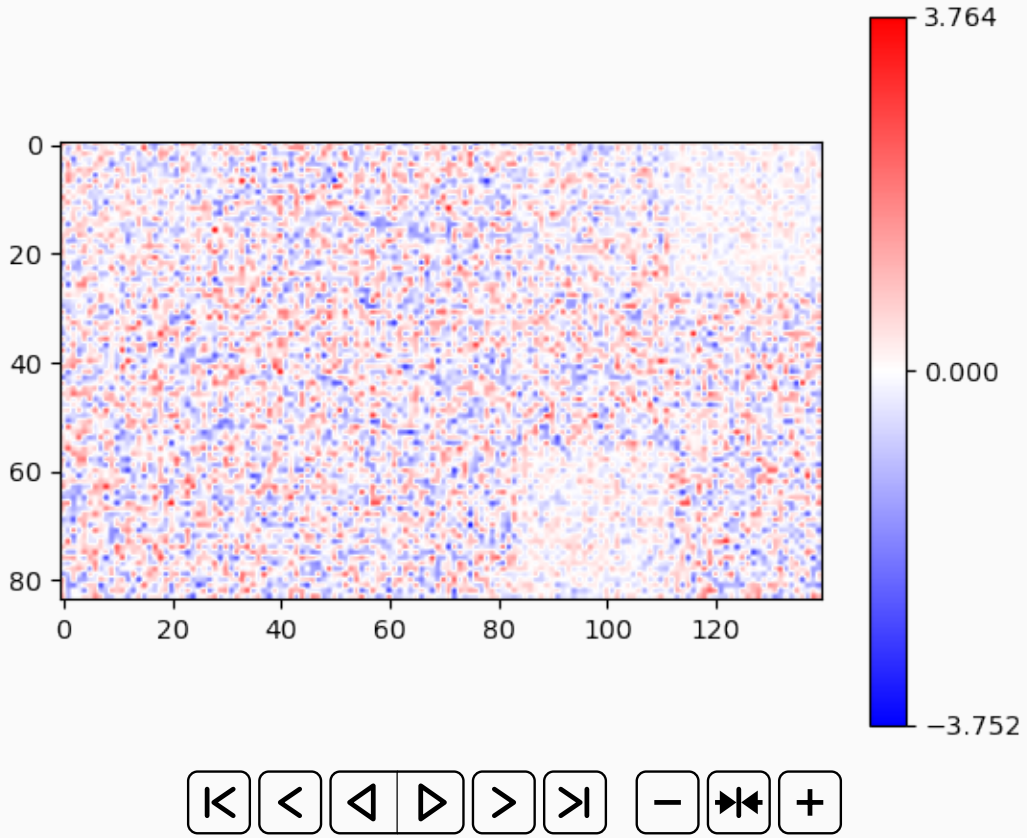


Figure 5 – Evolution of weights during unsupervised training on 300 MNIST samples chosen at random. Each sample $\boldsymbol{x}_i$ is "shown" to the model for a time $\tau_s < \tau_w$, by setting $\boldsymbol{v} = \boldsymbol{x}_i$ and solving $\boldsymbol{w}(t)$ for $0 \leq t \leq \tau_s$, in the quasi-stationary approximation ($\boldsymbol{h}(t) \equiv \boldsymbol{h}^*$). For $\tau_w = 100$, τ_s starts at 25 and decreases linearly to 5 (equivalent to a linear increase of τ_w), to aid convergence. Other parameters are: $n = 784$, $m = 15$, $w_{\text{inh}} = 15$, $p = 2$, $\Delta = 0.4$, $h_* = 0.8$, $\tau_w = 100$.

Model Problems

The model can learn useful representations! But...

- Parameters need to be **tuned** to make the model converge.

  For instance, w_{inh} should be s.t. a small $(5 - 15\%)$ fraction of neurons is initially active. But $w_{\text{inh}} \propto 1/m$ (inhibition is global), and also depends on p (higher p "suppresses" small weights). Similarly, h_* is tied to Δ. They may also be different for each dataset.

  $\rightarrow$ Difficult to choose *a priori*, many trials are needed!

- **Online** algorithm: samples need to be "shown" **one at a time**.
  Also, to solve for $\mathrm{W}(t)$, we need to compute $\boldsymbol{h}^*$ at stationarity at every step, which is computationally expensive.

  $\rightarrow$ Very slow!
  Previous simulation took $\sim 30\,\mathrm{min}$ just for 300 samples (whole MNIST has $60\,000$ samples).

The algorithm can be approximated to be much **faster**.

- Recall that $\boldsymbol{h}^*$ with $w_{\text{inh}} > 0$ maintain the same ordering of $\boldsymbol{I}$ (currents). Also, if $w_{\text{inh}} \sim 0$, $\boldsymbol{h}^* \sim \boldsymbol{I}$.
- So, let $\boldsymbol{h}^* = \boldsymbol{I}$, i.e. $h^*_\mu = \langle \boldsymbol{W}_\mu, \boldsymbol{v} \rangle$. Then **reorder** the units so that $h^*_1 > h^*_2 > \ldots h^*_n$. The activation function is then computed as follows:

$$g(h_\nu) = \begin{cases} 1 & i = 1 \text{ (h_1 is highest)} \\ -\Delta & i = 2 \text{ (or, in general, k)} \\ 0 & \text{otherwise} \end{cases}$$

  This can be applied to **batches** of data, and involves only matrix operations (which are efficient on GPUs).
- **Discretize** $\mathrm{dW} \,/\, \mathrm{d}t$, and compute a $\delta \mathrm{W}$ instead. Normalize it s.t. $\max |\delta \mathrm{W}| = 1$, and then update weights:

$$\mathrm{W}(t+1) = \mathrm{W}(t) + \eta \, \delta \mathrm{W}(t)$$

  where η is a (small) learning rate that decreases over time.

For each **epoch**:

Shuffle(training data)

For each **batch** $\{\boldsymbol{v_b}\}_{b=0,\ldots,\text{batch\_size}-1}$ in training data:

$$l_{\mu b} = \langle \boldsymbol{W_\mu}, \boldsymbol{v_b} \rangle_p$$

$$r_{s\mu} = \text{index } (b) \text{ of } s\text{-th highest element of } \{l_{\mu b}\}_{b=0,\ldots,\text{batch\_size}-1}$$

$$g(l_\mu) = \begin{cases} 1 & r_\mu = 0 \\ -\Delta & r_\mu = k-1 \\ 0 & \text{otherwise} \end{cases}$$

$$\delta \mathrm{W}_{\mu i} = g(l_\mu)[v_i - l_\mu \mathrm{W}_{\mu i}]$$

$$\mathrm{W}_{\mu i}(t+1) = \mathrm{W}_{\mu i}(t) + \eta\, \delta \mathrm{W}_{\mu i}(t)$$

Fast-AI Details

How does it **work**?

- The implementation is written in Python 3.9, using **Pytorch** [7], **Pytorch Lightning** [8] (for the unsupervised part), and **Pytorch Ignite** (for the supervised part).
- Works on **GPU**. The "bio"-learning rule can be scaled also to **clusters**, without changing any code!
- Can train "at once" more than one fully connected layer (one forward pass + iterate learning rule).
- **Modular** Learning rule is defined in the `BioLearningRule` class, which is separate from the model.

```
#inputs.shape=(input_size, batch_size)
#weights.shape=(output_size,input_size)
currents = (torch.sign(weights) * torch.abs(weigbs) ** (p-1)) @ inputs
_, ranks = currents.topk(k, dim=0) #Compute g()
post_act = torch.zeros_like(currents)
batch_idx = torch.arange(batch_size)
post_act[ranks[0], batch_idx] = 1.0
post_act[ranks[k-1], batch_idx] = -delta
d_weights = post_act @ inputs.t() #First term
norm = torch.sum(post_act * currents, dim = 1) #Second term
d_weights -= norm.unsqueeze(1) * weights #delta_weights
#Then normalize...
```

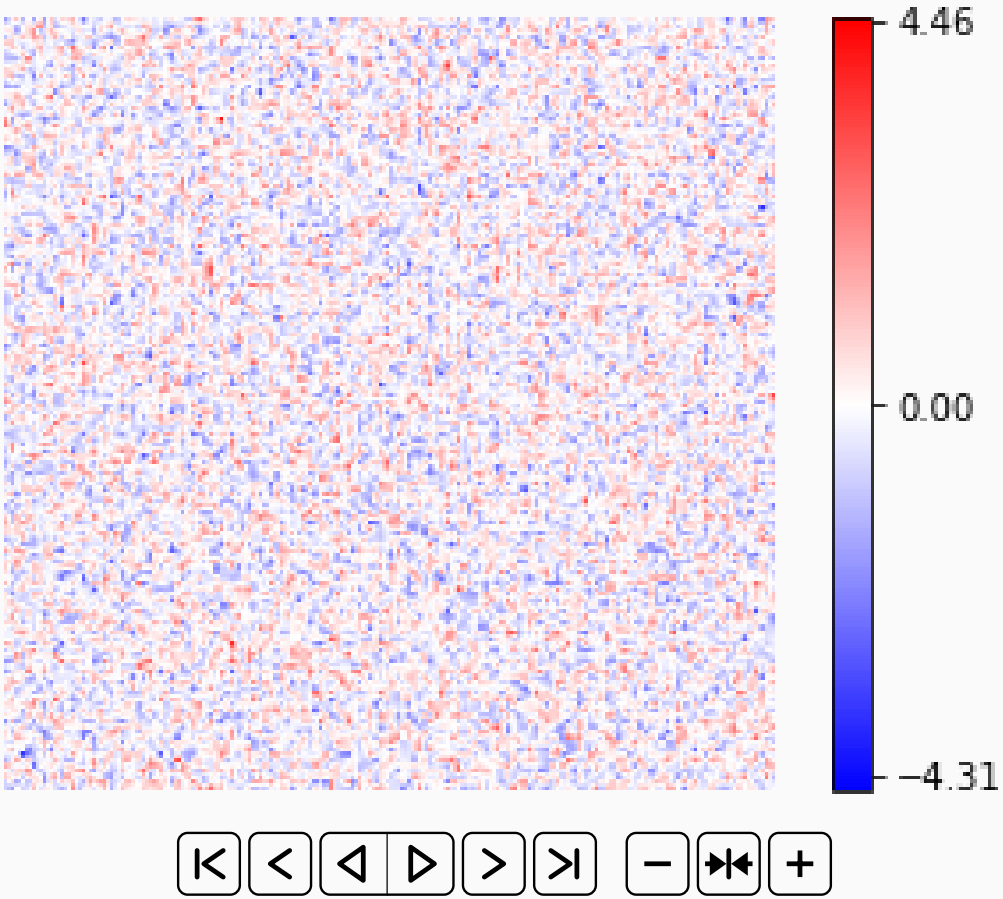


Figure 6 – Evolution of weights using the *fast* approximation. Hyperparameters are $\Delta = 0.4$, $p = 3$, $k = 2$. Learning rate η is 0.015 at the start, and decreases linearly to 0 over 200 epochs.

MNIST Dataset

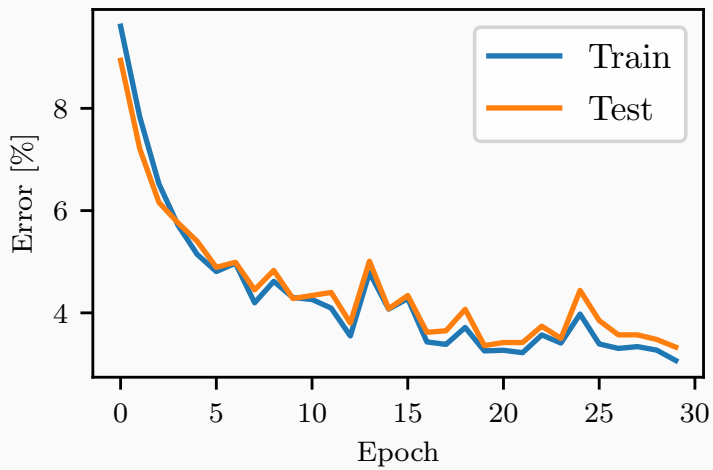


Figure 7 – Unsupervised Learning ($\Delta = 0.4$, $p = 2$, $k = 2$, $\eta = 0.02 \to 0$ in 200 epochs)
+ Supervised Learning (Adam optimizer, $\eta = 10^{-4}$, $m = 6$, $n = 4.5$, $\beta = 0.01$, 30 epochs)

Network:

$$\begin{cases} h_\mu = f(W_{\mu i} v_i) \\ c_\alpha = \tanh\left(\beta S_{\alpha\mu} h_\mu\right) \end{cases} \qquad f(x) = [\mathrm{ReLU}(x)]^n$$

- $W_{\mu i}$: unsupervised weights (first layer).
- $S_{\alpha\mu}$: supervised weights (top layer).

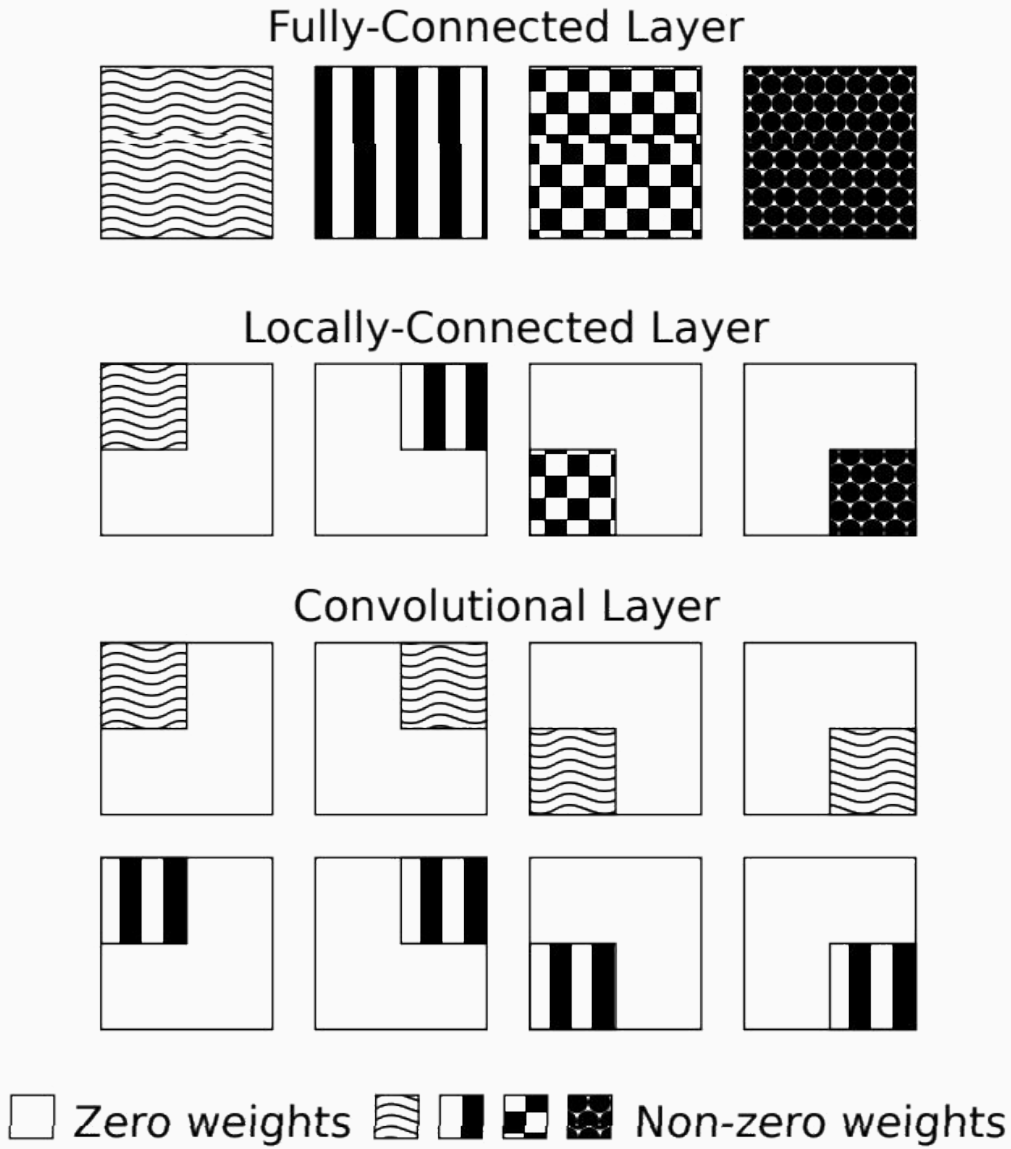
Loss function:

$$\ell(\boldsymbol{y}, \boldsymbol{y}^*) = \sum_{\text{batch}} \sum_{\alpha=1}^{10} |y_\alpha - y_\alpha^*|^m$$

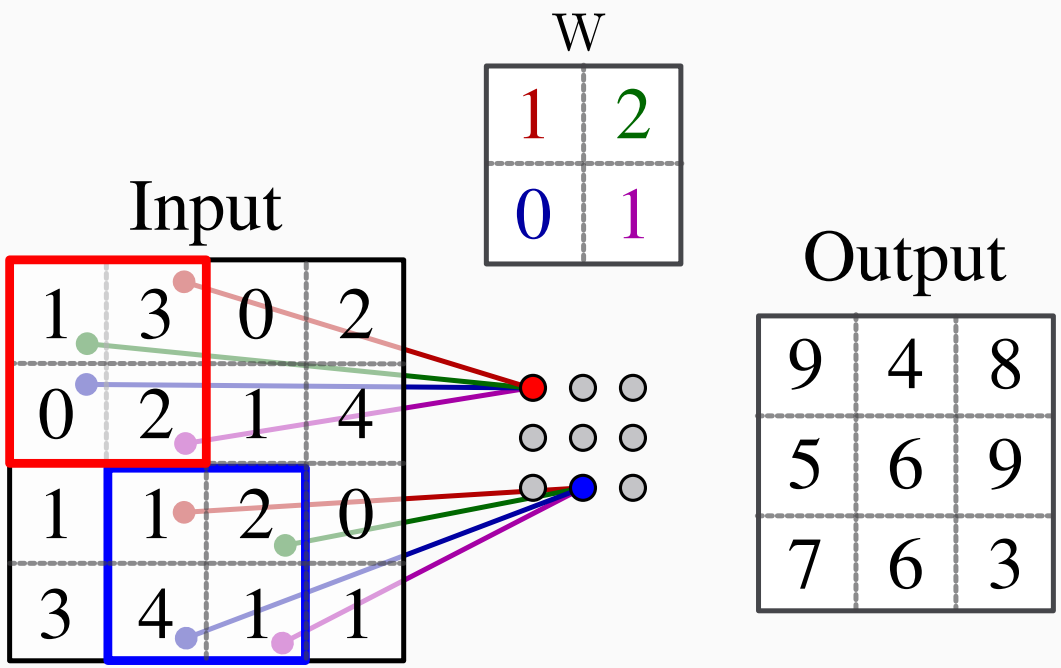
where $\boldsymbol{y}$ is the **prediction**, and $\boldsymbol{y}*$ the two category.
A sample of k-th category has $y_\alpha^* = -1 + 2\delta_{\alpha k} \in \{\pm 1\}$ ("spin-like" one-hot encoding).

Best accuracy (test): 96.7% (Lower than in [2], but no hyperparameters fine-tuning)
Comparable with end-to-end trained networks!

What about more modern **architectures**?



Convolutional Neural Network = **Locally connected** Weights + **Shared** Weights



- **Unsupervised** training on **MNIST** ($\Delta = 0.4$, $p = 4$, $k = 2$, $\eta = .015$).
- **Network** is:

```
self.conv1 = nn.Conv2d(1, 16, kernel_size=10, bias=False)
self.max_pool = nn.MaxPool2d(2)
self.conv2 = nn.Conv2d(16, 5, kernel_size=8, bias=False)
```

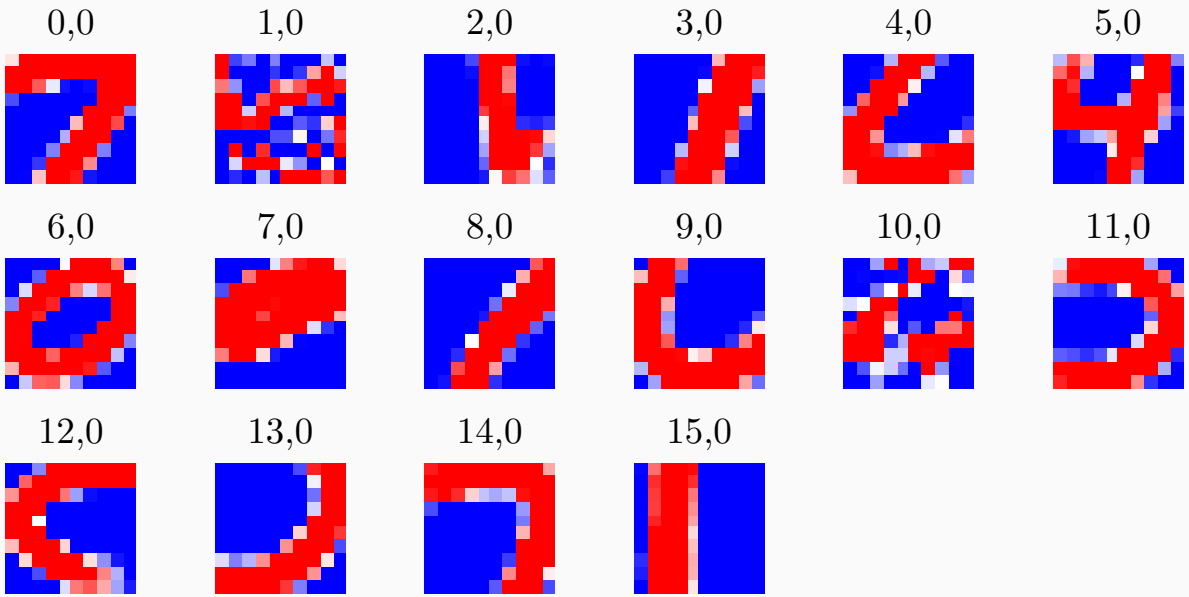


Figure 8 – Weights for the **first** convolutional layer. Basic parts of digits can be seen.

Generalization CNN weights

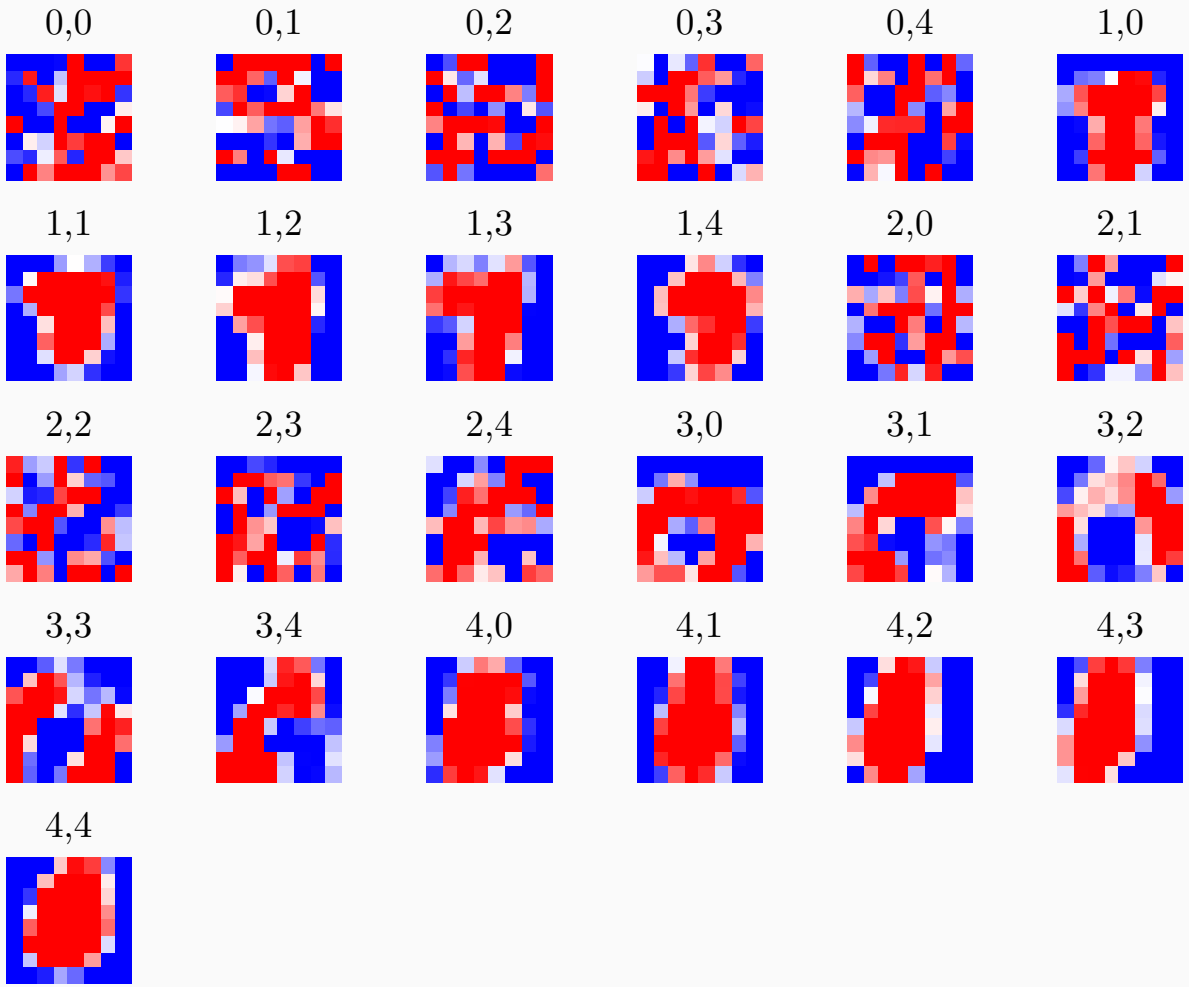


Figure 9 – Weights for the **second** convolutional layer. Edge/center detectors and more complex filters can be seen. Still, some filters stop training **prematurely**: the model should be generalized also to aid training of deeper layers.

Conclusions

1. Alternative learning rules deserve **attention** and research.

   → Performance is not the same as **backpropagation**, so changes in the algorithm may be needed to make it faster/more scalable.
2. PyTorch implementations of [2] exist (e.g. `https://bit.ly/30MvBuS`).

   → This is the first using Pytorch Lightning, which allows **scaling** on multiple GPUs/clusters.
3. "BioLearning" can be applied to more **complex** architectures (deep nets, CNNs).

   → Testing on more complex datasets is required (needs more computing power)

References I

[1] Timothy Lillicrap, Adam Santoro, Luke Marris, Colin Akerman, and Geoffrey Hinton.
Backpropagation and the brain.
Nature Reviews Neuroscience, 21, 04 2020.

[2] Dmitry Krotov and John J. Hopfield.
Unsupervised learning by competing hidden units.
Proceedings of the National Academy of Sciences, 116(16):7723–7731, 2019.

[3] Donald O. Hebb.
***The organization of behavior: A neuropsychological theory*.**
Wiley, New York, June 1949.

[4] Peter Dayan and L. F. Abbott.
***Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*.**
The MIT Press, 2001.

[5] Erkki Oja.
Simplified neuron model as a principal component analyzer.
Journal of Mathematical Biology, 15(3):267–273, November 1982.

References II

[6] EL Bienenstock, LN Cooper, and PW Munro.
Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex.
Journal of Neuroscience, 2(1):32–48, 1982.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala.
Pytorch: An imperative style, high-performance deep learning library.
In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] WA Falcon and .al.
Pytorch lightning.
GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning, 3, 2019.