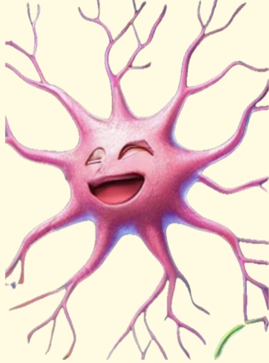
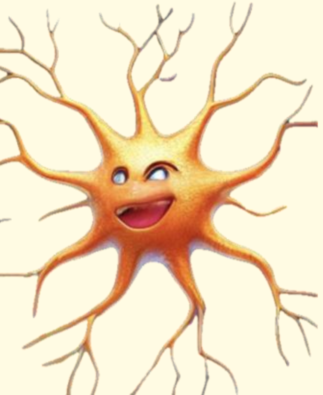


Unsupervised learning by competing hidden units

A research by Dmitry Krotov and John J. Hopfield

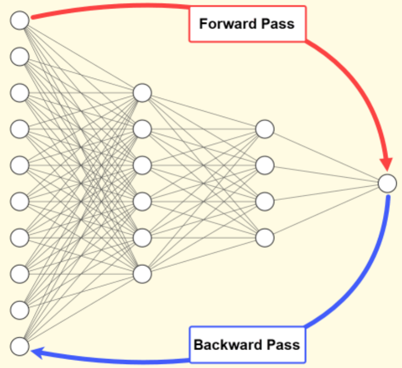


Course 20882 – Computational Neuroscience
Project by: Adam Amer & Alessandra Bontempi
Academic Year 2024/2025, Bocconi University

Background

Traditionally, end-to-end learning with backpropagation has been essential in complex classification tasks for learning good feature detectors in ANNs. However, this traditional approach is not biologically plausible:

In ANNs, learning is a global process. Thereofore, the adjustment of weights and biases for a neuron can depend on interactions involving nodes located across multiple layers, including those situated several layers earlier in the network. Instead, in human brains, neurons are only affected by other neurons that are locally nearby.



ANNs typically learn through supervised learning, meaning they need many labeled examples. Real brains learn mostly through unsupervised learning, which doesn’t rely on labeled data (e.g., babies learn to recognize objects by observing and interacting with them without being told the names of everything).

"Neurons that fire together, wire together"
Donald O. Hebb

Objective and architecture

IDEA: in this paper, the researchers investigate if early layer representations can be learned in an ANN without supervision and only leveraging an *ad hoc* local "biological" rule.

REASON: the motive behind this research is to replicate a biologically plausible model which does not rely on end-to-end training with backpropagation and still learns good feature detectors in early layers of the ANN.

ALGORITHM (overview): the network is built as follows:

- An input layer which takes images from the CIFAR-10 and MNIST datasets
- A hidden layer in which the hidden neurons compete against each other to represent the input features and weights are updated following local Hebbian-like updates
- An output layer which produces the final representation

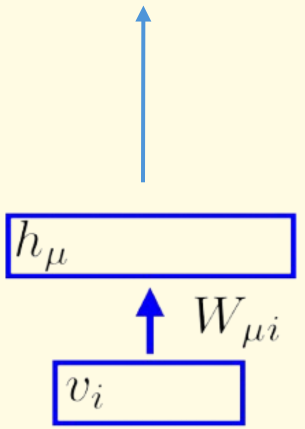
Mathematical framework and synaptic plasticity rule

Krotov and Hopfield's model employs a feed-forward architecture, where information flows sequentially from the input layer through multiple hidden layers to the output layer.

The learning of hidden layers is governed by locally defined update rules inspired by Hebbian plasticity, ensuring that synaptic weight changes depend only on the activities of connected neurons. Additionally, the hidden layers incorporate non-linear activation functions, which introduce complexity into the network and enable the detection of different patterns within the input data.

$$h_u = f\,(W_{ui} \cdot v_i)$$

where f is the ReLu function and v_i are the visible neurons.



The synaptic plasticity rule studied by the researchers governs the adjustment of synpatic weights W_{ij} between neurons i and j, ensuring learning remains local. The plasticity rule is:

$$\tau_L \frac{dW_i}{dt} = g(Q)\ (R^p v_i \ - <\mathbf{W},\boldsymbol{v}>\ W_i\ ) \quad \text{where } Q = \frac{<\mathbf{W},\boldsymbol{v}>}{<\mathbf{W},\mathbf{W}>^{(p\,-\,1)/p}}$$

Here, g(Q) is a non-linear learning activation function.
If p (Lebesgue parameter) is increased, only large values of the weights W_i contribute to the normalization condition, while small values become suppressed. As t goes to ∞, the weights W_i will converge to a sphere of radius R; this ensures homeostaticity.

A biologically inspired algorithm

The synaptic plasticity rule described earlier strengthens connections between co-activated neurons; however, it doesn't encourage competition or differentiation between hidden units. To maximize diversification in feature detection, the scientists developed the following equation:

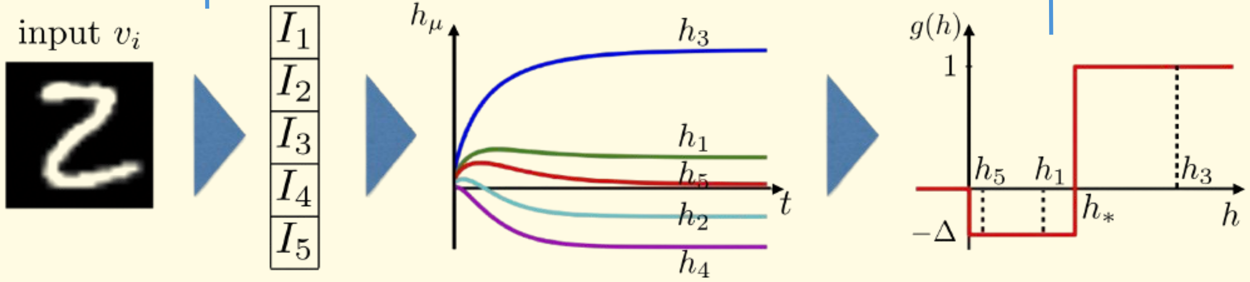
$$\tau \frac{d\,h_u}{dt} = I_u - w_{\text{inh}} \sum_{v \neq u} r(h_u) - h_v \qquad \text{where } I_u = <\mathbf{W}_u\,,\, \boldsymbol{v}>_u \ \text{ and } r(h_u) = \max(h_u, 0)$$

h_u represents the activites of hidden units, $r(h_u)$ are the firing rates, and I_u are input currents.

The pipeline of the unsupervised part of the algorithm starts with the input. Each raw input v_i is converted throught the weights W_{ui} into a set of input currents I_u.

We fix the strength parameter of the global inhibition. At the end, only a small fraction of hidden units have positive activity (here, h_1, h_3, and h_5).

g is a non-linear function that implements temporal competition between features. Here, the strongest synapses are pushed towards the inputs that activated them, resulting in neuron specialization.



Krotov and Hopfield's results ...

MNIST dataset

The image set is split in: 50k images for training and 10k validation images for finetuning. After the hyperparameters (p = 3, k = 7, Δ = 0.4, n = 4.5) are fixed, the whole set is used to train the final model.

2000 hidden units are used in the first experiment to find the weights **W**. As training progresses the weights eventually converge to the surface of a sphere. See Figure 1.1 (left).

After the unsupervised phase, **W** is frozen and used in a supervised layer with standard SGD. In Figure 1.1 (right), we notice that the "bio" algorithm reaches a similar error level of 1.46% of the well-known benchamarks. The training error is higher (0.40% instead of 0%).

The "bio" algorithm learns feature detectors that encode ink presence and absence, leveraging positive and negative weights. Furthermore, the algorithm doesn't simply match templates to classify data, rather each feature "votes" for certain images and against others.

CIFAR-10 dataset

The image set is split in 45k training images and 5k validation images, and each image is normalized to be a vector in the 32x32x3 space. The optimal hyperparameters found are: p = 3, k = 7, and Δ = 0.4

In Figure 1.2 (right), we notice that the "bio" algorithm reaches a training error of 44.95% and a test error of 49.25%. In comparison, the network trained end-to-end with backpropagation achieves 0% training error (after 200 epochs) and 44.74% on the test set.

In Figure 1.2 (left), we see the weights of 25 randomly selected hidden units, which represent the diverse features detected by the neurons. It's noticable that the weights have large negative elements, meaning the hidden units "connect" only with few features.

It's very noticable from Figure 1.2 (left) and Figure 1.2 (center) how the "bio" algorithm picks up on more general patterns (i.e., continuity of colors), even though the given is pixel-color invariant. Instead, the standard algorithm looks more speckled, meaning it focuses more on detecting features like edges.

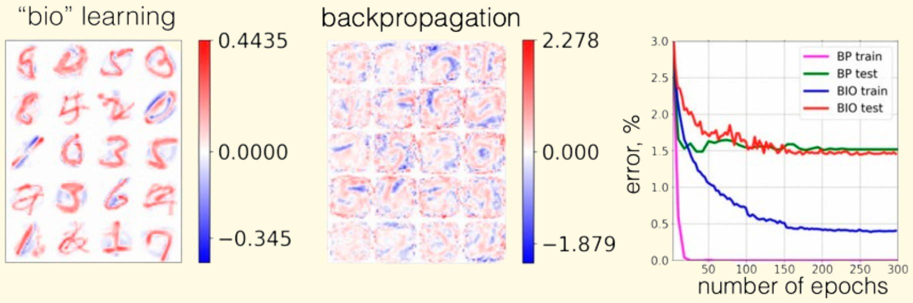


Figure 1.1: we see the weights learnt during the unsupervised part (left). We see training and testing errors during the supervised phase (right).

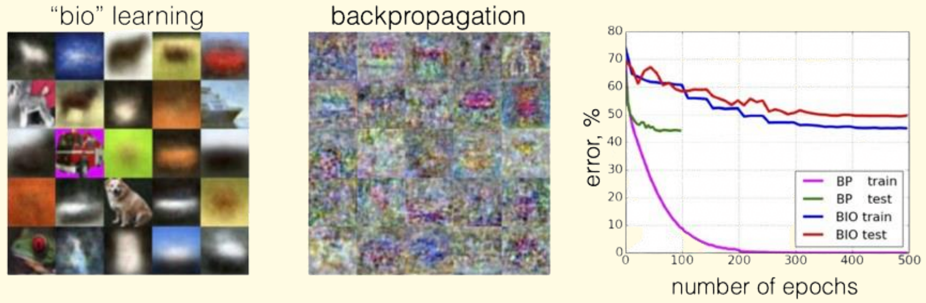
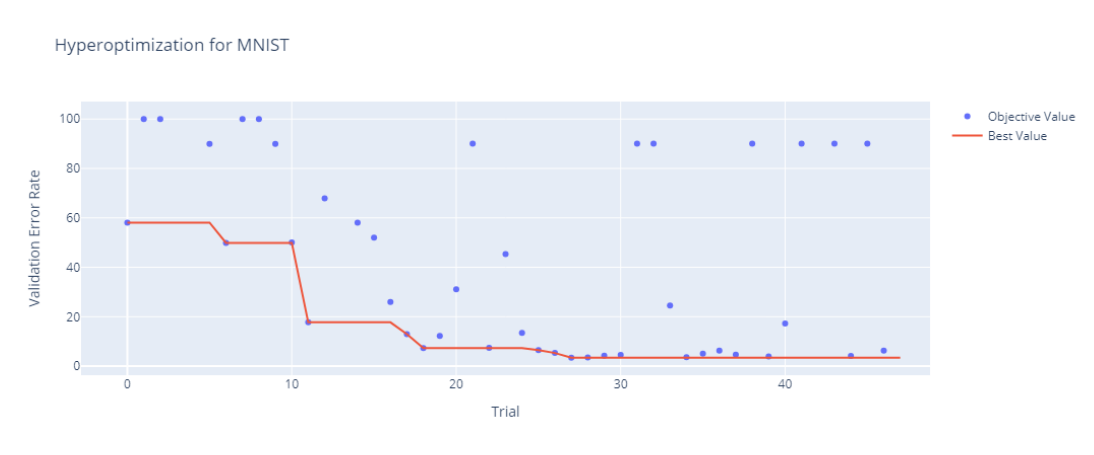


Figure 1.2: we see the weights learnt during the unsupervised part (left). We then see training and test sets errors as training progresses (right).

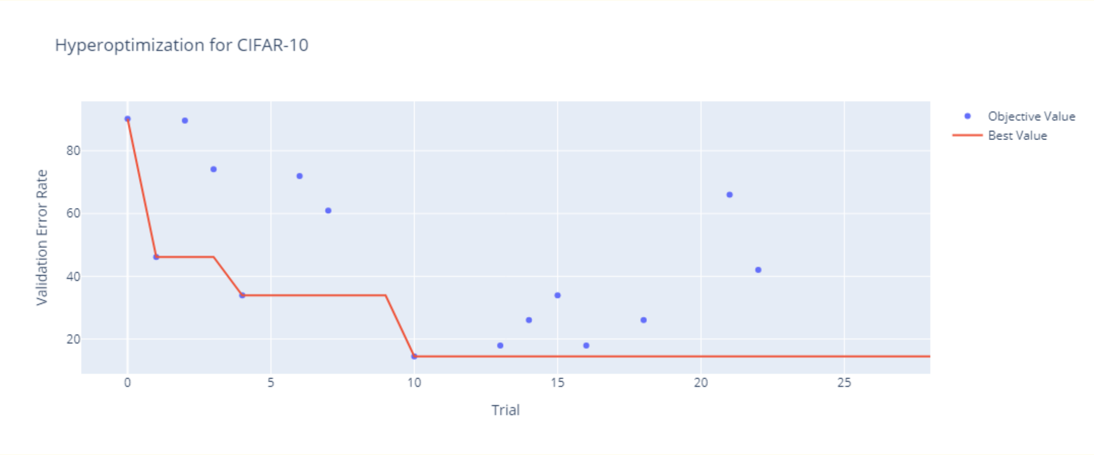
... our results

REPLICATION: initially, the replication of the paper's results was carried out with the same hyperparameter values found in the research. However, we were unable to achieve the same low error rates (although over a smaller number of epochs).

BAYESIAN HYPERPARAMETER OPTIMIZATION: consequently, we coded and ran our own Bayesian hyperparameter optimization (with the Tree-Structured-Parzen Estimator) to find a set of hyperparemeters that achieved an error rate similar to the paper over 100 unsupervised epochs and 20 supervised epochs.



In the MNIST dataset, we achieved a ~3% error rate with the following hyperparameters: k = 3, batch size = 55, p = 4, m = 9, n = 5, and Δ = 0.2



In the CIFAR-10 dataset, we achieved a ~14% error rate with the following hyperparameters: k = 1, batch size = 98, p = 5, m = 10, n = 3, and Δ = 0.7

Multi-dataset hyperparameter optimization

Given the high sensitivity of the error rate to small changes in the hyperparameters and the remarkably diverse results obtained by optimizing for CIFAR-10 and MNIST, we set out to find a set of global hyperparameters which work well for multiple datasets. We decided to finetune over datasets of varied nature and complexity - CIFAR-100, MNIST, and FashionMNIST, optimizing for **1)** final average error rate **2)** average improvement over epoch (to promote steady progress rather than overfitting and plateauing) and **3)** time taken to train (find Pareto efficient configurations).

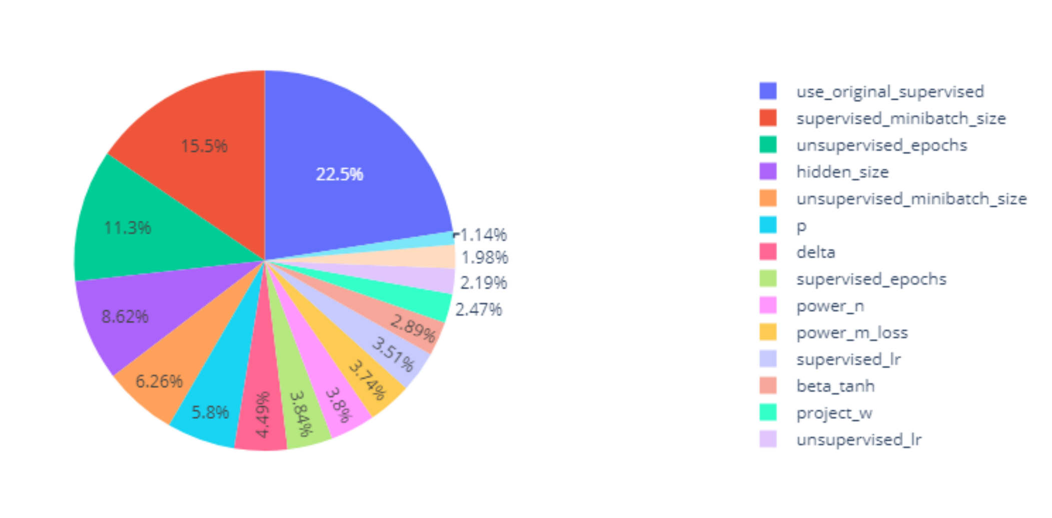
BEST TRIAL

By simply running the algorithm over 24 unsupervised and 2 supervised epochs, using just 744 hidden neurons, we were able to achieve an average ~8% error rate across the three datasets.

We used the following hyperparameters:

$\Delta = 0.35$	$p = 2$	$n = 1$
$k = 7$	$m = 6$	

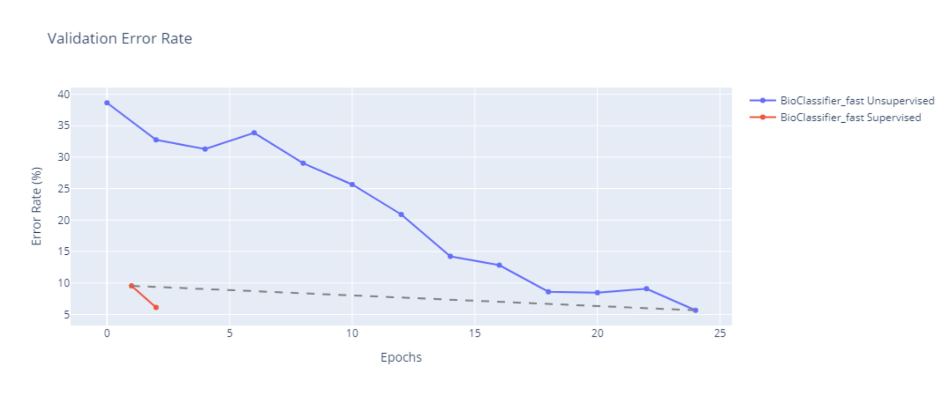
HYPERPARAMETER IMPORTANCE

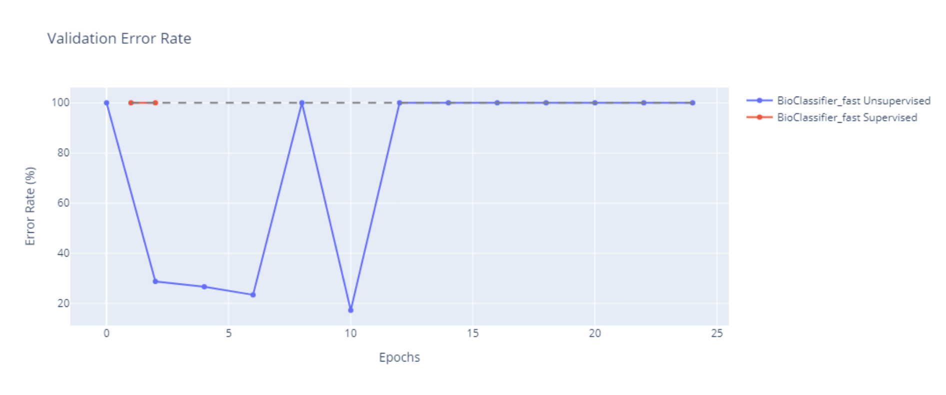


Extension: data-augmentation

DATA AUGMENTATION

We tried to compare whether error rate would improve on our best configuration, if we trained the model on CIFAR-100 with a learned augmentation policy.
The performance drammatically worsens (see right figure), inhibiting actual learning.





STEADY-STATE IMPLEMENTATION

We implemented the «slow» version of the algorithm but due to limited computational capacity, we weren't able to run the full hyperparameter optimization. However, we tried to test it out on CIFAR-10 with the optimal hyperparameters and achieved and error rate of ~18%

Extension: fully-unsupervised CNN

REASON: CNNs are highly accredited models for image classification tasks due to their efficiency derived by parameter sharing across various locations. This approach reduces significantly the number of parameters used compared to FCNs.

Extending Krotov and Hopfield's work, we decided to use this well-known benchmarked model to create a network that maintains biological plausibility while using less parameters.

ALGORITHM: the network follows Krotov and Hopfield's "bio" algorithm, where:

- An input layer takes images from the CIFAR-100, MNIST and FashionMNIST datasets
- Two hidden layers each consisting of a custom convolution, which applies competition among filters, and a diversification step carried out with Gram-Schmidt orthogonalization
- The output layer is a simple classifier head, trained with Adam optimiz. In this supervised phase (the only one trained with backpropagation), the convolutional filters remain frozen

RESULTS: the avg. validation accuracy is ~70% and it's achieved over 3 unsupervised and 3 supervised epochs with: batch size = 243, k = 4, Δ = 0.3, kernel = 7x7, padding = 2, stride = 1, and pooling = 5x5

For CIFAR-100, the CNN has 225k parameters, divided in 49% unsupervised and 51% supervised parameters. This corresponds to a 90% reduction in model w.r.t. the best performing FCN. We are certain that, with a deeper CNN, the performance can be improved with a low parameter count.

Further research directions

Due to computational constraints, we were unable to conduct additional experiments on certain intriguing aspects that emerged during our exploration of Krotov and Hopfield's research. Nevertheless, we have identified several promising ideas that could serve as valuable directions for future research in neuroscience:

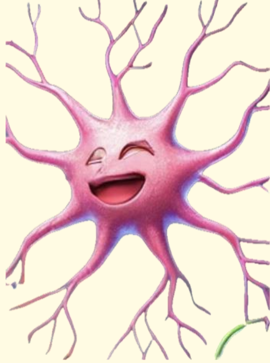
E-I neurons: in the research, it's written that the "bio" algorithm does not respect a principle rule: namely, neurons can be either inhibitory or excitatory, but never both. To address this concern, we propose a biological plausible model which also respects lateral inhibition across hidden neurons:

- we implement a plasticity rule similar to the one in the paper
- we impose the hidden neurons to be exclusively excitatory
- we connect a small layer of exclusively inhibitory neurons (possibly only locally), where the I neurons promote the E neuron with highest activation and "punish" the remaining k neurons. During this activity, we should enforce homeostaticity (i.e., R^p).

This algorithm should be biologically plausible and take into account the E/I constraint.

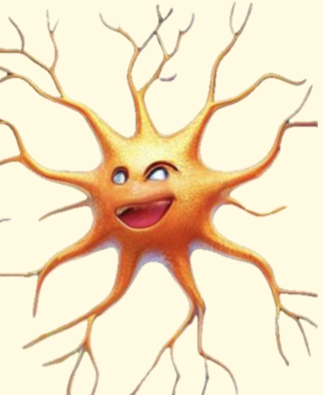
STDP: another neuronal behavior which was not explored in the paper was STDP. In this case, we thought of creating an algorithm that would process the input in batches and take a time t(*) for each node i. We'd then compute the average spiking time avg(T) and: if avg(T) > t_i , the activation for node i would be multiplied by k < 0; else, it would be multiplied by a positive k. Lastly, we'd compute the highest activation and set it > 0. All the other activations would be set to be < 0.

* TBD



Thank you for your attention

For any doubt or question, feel free to reach out



Adam Amer: adam.amer@studbocconi.it
Alessandra Bontempi: alessandra.bontempi@studbocconi.it