1 Hopfield-like network for image classification

Inspired by the simplicity of Hopfield networks, we consider the idea of using the same kind of Hebbian learning rule in a feed-forward architecture.

Consider a 1-layer network with N input neurons σ_i, P output neurons S_j and synapses ω_{ji}

$$S_j = \sum_i^N \omega_{ji}\sigma_i \,. \tag{1}$$

In the training phase, when a pattern σ_i^1 is presented to the input neurons, it is stored in the synapses connected to the first output neuron. A second pattern σ_i^2 is stored in the synapses connected to the second output neuron and so on:

$$\omega_{i1} = \sigma_i^1 \,, \qquad \omega_{i2} = \sigma_i^2 \,, \qquad \omega_{ij} = \sigma_i^j \,. \tag{2}$$

The network will then be able to store P patterns.

When a new pattern σ_i^k is presented to the input layer, the network response will be stronger at the output neuron S_j corresponding to the stored pattern σ_i^j more similar to σ_i^k. Note that this is true only for binary patterns ($\sigma_i = \pm 1$). To prove this statement, simply consider that the product of 2 neurons σ_i, σ_i' is maximum when they are equal. So the less neurons differ between the input and the stored pattern, the higher the network response will be.

When using continuous input neurons, the network response tends to fall into the "brightest" stored patterns. In this case, one should store the n-bit representations of the patterns.

The network can be also propagate signals backwards

$$\sigma_i = \sum_j^P \omega_{ij} S_j \,. \tag{3}$$

By firing a single neuron k in the output layer, the response of the network in the input layer will be the corresponding stored pattern σ_i^k

$$S_j = \delta_j^k \longrightarrow \sum_j^P \omega_{ij}\delta_j^k = \omega_{ik} = \sigma_i^k \,. \tag{4}$$

So, as in the original Hopfield model, the network is able to remember the stored patterns.

An advantage of this approach over the standard Hopfield implementation is that the capacity of the network is independent of the input layer dimension. If necessary the output size can be increased to accomodate more patterns without affecting the already stored ones. Also, the interference between stored patterns that is observed in the Hopfield model due to the overlaps in the synaptic weights is avoided in this architecture since each synapses contains information regarding only a single pattern.

1.1 Tests

To test this approach, we consider a network consisting of a convolutional encoder that preprocess and reduces the dimension of the input samples, followed

by a 2-layers Hopfield network. The output s_i of the encoder is transformed into a N_b-bit spin representation $s_i^\alpha = \pm 1$ ($\alpha = 0, \dots N_b - 1$), so that it can be stored and retrieved by the Hopfield layers.

We want to use the network for classification and we pretrain the convolutional encoder as part of an autoencoder to reconstruct the samples from the dataset.

The training procedure for the Hopfield layers is as follow: when a new pattern is presented, if it is correctly classified nothing happens, else the pattern is stored in the first layer and connected to the correct category in the second layer.

For a total number of stored patterns N_p, the Hopfield weights in the 2 layer are ω_{ji}^α, $\hat{\omega}_{nj}$ with $\alpha = 0, \dots N_b - 1$ the bit index, $j = 0, \dots N_p - 1$ the stored pattern index, i the (encoded) input index and n the output category index. An input pattern s_i propagates through the first layer as

$$S_j^\alpha = \sum_i \omega_{ji}^\alpha s_i^\alpha \,, \tag{5}$$

The components of the output S_j^α are then summed over the bit index α, $S_j = \sum_\alpha S_j^\alpha$ and the resulting vector converted to a one-hot vector S'_j indicating the position of the maximum of S: $S'_j = 1$ if $j = \text{argmax}(S)$ else 0.

S'_j is then feeded into the second layer:

$$C_n = \sum_{j=0}^{N_p - 1} \hat{\omega}_{nj} S'_j \,, \tag{6}$$

to produce the category prediction $i_{\text{cat}} = \text{argmax}(C)$.

To assess the performance of the network we test it on the MNIST dataset and compare it to 2 other networks:

- a 2 linear layers network on top of the same pretrained convolutional encoder (where only the linear layers are trained)

- a full convolutional network given by the convolutional encoder + 2 linear layers (all layers trained from scratch).

The network architecture for the Hopfield model is:

$$\boxed{\begin{matrix}\text{Input}\\(1,32,32)\end{matrix}} \to \boxed{\begin{matrix}\text{Conv2d}(k3,s2)\\(100,15,15)\end{matrix}} \xrightarrow{ACT} \boxed{\begin{matrix}\text{Conv2d}(k3,s2)\\(50,7,7)\end{matrix}} \xrightarrow{ACT} \boxed{\begin{matrix}\text{Conv2d}(k4,s2)\\(25,2,2)\end{matrix}}$$
$$\xrightarrow{ACT} \boxed{\begin{matrix}\text{Flatten + Bin}\\(N_b \times 100)\end{matrix}} \to \boxed{\begin{matrix}\text{Hopfield}\\(N_b \times 100 \times N_p)\end{matrix}} \to \boxed{\begin{matrix}\text{Hopfield}\\(N_b \times N_p \times 10)\end{matrix}} \to \boxed{\begin{matrix}\text{Output}\\(10)\end{matrix}}, \tag{7}$$

and for the CNN:

$$\boxed{\begin{array}{c}\text{Input}\\(1,32,32)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Conv2d}(k3,s2)\\(100,15,15)\end{array}} \xrightarrow{ACT} \boxed{\begin{array}{c}\text{Conv2d}(k3,s2)\\(50,7,7)\end{array}} \xrightarrow{ACT} \boxed{\begin{array}{c}\text{Conv2d}(k4,s2)\\(25,2,2)\end{array}}$$
$$\xrightarrow{ACT} \boxed{\begin{array}{c}\text{Flatten}\\(100)\end{array}} \rightarrow \boxed{\begin{array}{c}\text{Linear}\\(100\times 50)\end{array}} \xrightarrow{ACT} \boxed{\begin{array}{c}\text{Linear}\\(50\times 10)\end{array}} \xrightarrow{ACT} \boxed{\begin{array}{c}\text{Output}\\(10)\end{array}}, \tag{8}$$

The full CNN achieves the best results (accuracy 99%) followed by the 2 linear layers (accuracy 97.5%). The 2 layers Hopfield network only manages to reach an accuracy of 86.5%.

We expected that the convolutional encoder would smooth out the differences between category elements so that the Hopfield network could more easily associate the inputs to the stored category patterns.

However, this does not seem to be the case: when training on the 60000 samples, the network still stores around 10000 different patterns (instead of only 10 corresponding to the MNIST categories). When running the same network without encoder the number of stored patterns is only slightly higher (around 11500) and the accuracy only slightly lower (84.2%).

We need to investigate if it is possible to improve the network performance by using different kind of encoders.