Sparse coding via a locally competitive algorithm (LCA)

Bruno A. Olshausen

September 25, 2018

Recall that the sparse coding image model is of the form

\[ I(x, y) = \sum_i a_i \phi_i(x, y) + \epsilon(x, y) \]  

(1)

where \( I(x, y) \) denotes the pixel intensities within an image patch (\( x \) and \( y \) are spatial coordinates), and the \( \phi_i(x, y) \) are a set of basis functions or ‘features’ for describing the image. The \( a_i \) correspond to neural activity (the subscript \( i \) denotes the index of the neuron). The goal is to find a set of features that allow images to be described using as few non-zero \( a_i \) as possible.

The energy function governing this system is as follows:

\[
E = \frac{1}{2} \sum_{x,y} \left[ I(x, y) - \sum_i a_i \phi_i(x, y) \right]^2 + \lambda \sum_i C(a_i)
\]

(2)

The neural activities \( a_i \) for a given image \( I(x, y) \) are computed by finding a minimum of this energy function. In the original sparse coding algorithm (Olshausen & Field 1996), they were computed by direct gradient descent. However this is not very efficient in terms of a numerical (algorithmic) implementation. In addition, we would like to have a physical implementation that corresponds more closely to what a neural population could do.

In Rozell et al. (2008), we derived a dynamical system for efficiently computing the \( a_i \) which descends the energy, but not according to steepest descent per se. This system is described by the following equations:

\[
\tau \dot{u}_i + u_i = b_i - \sum_{j \neq i} G_{ij} a_j
\]

(3)

\[
a_i = g(u_i)
\]

(4)

where \( b_i = \sum_{x,y} \phi_i(x, y) I(x, y) \) and \( G_{ij} = \sum_{x,y} \phi_i(x, y) \phi_j(x, y) \). The nonlinearity \( g() \) is determined by the form of the cost function \( C() \). In the case where \( C \) is the L1 norm over the coefficients, i.e., \( C(a_i) = |a_i| \), then \( g \) is a ‘soft-thresholding’ function of the form

\[
g(u_i) = \begin{cases} 
    u_i - \lambda & u_i > \lambda \\
    0 & -\lambda < u_i < \lambda \\
    u_i + \lambda & u_i < -\lambda
\end{cases}
\]

(5)
This is plotted below for $\lambda = 1$. It is also called a “shrinkage” function because it shrinks the values below threshold to zero, and those above threshold by $\lambda$.

We called this method a “locally competitive algorithm” (LCA) because it computes the coefficients through a local competition (lateral inhibition) and thresholding. The LCA has a simple neural implementation as follows: each neuron is a leaky integrator that is driven by a feedforward term - the $b_i$, which is the inner product between the neuron’s receptive field $\phi_i(x,y)$ and image $I(x,y)$ - and inhibited by other neurons in the population by lateral connection strengths $G_{ij}$. The resulting subthreshold potential from the leaky integrator, $u_i$ is passed through a threshold function $g$ to give the neurons output $a_i$.

The learning rule for the $\phi_i(x,y)$ is the same as in the original algorithm. It is derived via direct gradient descent on $E$ using the coefficient values $\hat{a}_i$ computed from the LCA. This yields the update rule

$$\Delta \phi_i(x,y) = \eta r(x,y) \hat{a}_i$$  

(6)

where $r(x,y) = I(x,y) - \sum \hat{a}_i \phi_i(x,y)$ and $\eta$ is the learning rate. Note that the equations given above for LCA are conditioned on the basis functions having unit length, so after each update they must be renormalized so that $\sum_{x,y} \phi_i(x,y)^2 = 1$ $\forall i$.

To summarize the above, the LCA computes for each image

$$\hat{a} = \arg \min_a E(I,a;\Phi)$$  

(7)

and the learning rule seeks to solve

$$\Phi^* = \arg \min_{\Phi} \langle E(I,\hat{a};\Phi) \rangle$$  

(8)

where the brackets $\langle \rangle$ denote average over many (millions) images.