Neural Principal Component Analysis

October 17, 2009

On this page, I propose some demos using the Oja's or Sanger's learning rules to perform Principal Component Analysis. The demos are written in Python and need in addition matplotlib

1 Oja's rule : Getting the first principal component

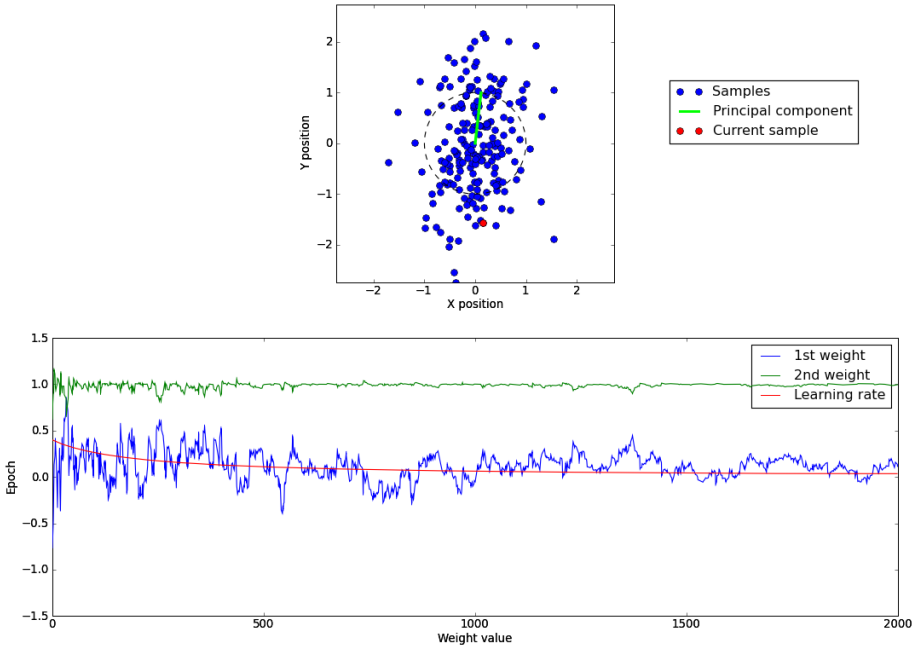
An introduction to the Oja's learning rule can be found on

Let's consider a set of N time varying input signals $x_i(t)$. In the following, we denote the input vectors as $\mathbf{X(t)} = (x_i(t))_{i\in[1;N]}$. We will also consider a neuron y which is connected with the N input neurons with a certain weights' vector W. Therefore, when one sample of the input signals $X(t)$ is presented to the network, the activity of neuron y is computed by $y = W^T.X$, where W^T denotes the transpose of W.

Updating the weights with the Oja's rule can then be formulated as : $\Delta W(t) = \alpha(t)(X(t).y(t) - y(t)^2W(t))$ ($y(t)$ being a scalar, $X(t)$ and $W(t)$ being two one-dimensional vectors). $\alpha(t)$ is the learning rate, which has to decrease with time t. Then, one can show that if the Oja's rule is converging (see the scholarpedia reference above), the vector $W(t)$ converges to one eigen-vector of the covariance matrix of the input signals corresponding to the first principal component of them, and its norm converges to one.

The following script provides a Python demo showing how the Oja's learning rule can be used to extract the first principal component of a distribution : oja.py

Below is an illustration of the demo. In the upper graph, the blue points represent the 2D samples, the red point the currently processed sample, the green line the weight vector of the output neuron and the dotted black circle is of unitary radius showing the normalization of the weights. On the second graph is represented the evolution of the two weights and of the learning rate through the learning epochs. In addition to using the previous learning rule, with a decreasing learning rate, the samples are also centered to have a zero mean.



2 Sanger's rule : Getting multiple principal components

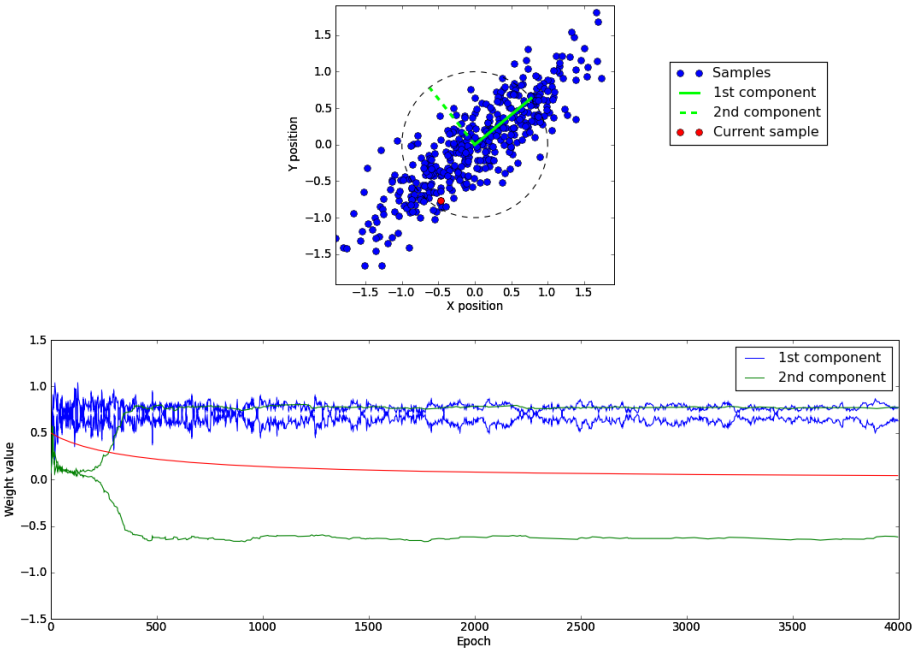
A description of the Sanger's learning rule (also called Generalized Hebbian Learning), used to extract several principal components can be found in the thesis of Terence Sanger.

The Sanger's learning rule can be considered as an extension of the Oja's learning rule, that can be applied to extract a subset of the principal components. Consider that you have N input signals and M output neurons. Let's denote W a NxM matrix $(M \leq N)$where the column j contains the weights connecting the input neurons to the output neurons y_j. When presenting, at time t, a sample $X(t) = (x_i(t))_{i \in [1;N]}$ to the network, the output $Y(t) = (y_j)_{j \in [1;M]}$ is computed as $Y(t) = W(t)^T X(t)$.

The update of the weights using the Sanger's learning rule is done by : $\Delta W = \alpha(X.Y^T - W.UT[YY^T])$. The time index is removed but all the vectors are time dependent (where each epoch corresponds to the presentation of one sample) as well as the learning rate (which has to converge to zero with some conditions on the series $\sum_t \alpha(t)^k$, see p. 40-41). The $UT[.]$ operator is the *Upper Triangular* operation which sets all the values below the diagonal to zero. Expanding the matrix formulation, it leads to :

$$\forall i \in [1;N], \forall j \in [1;M]; \Delta W_{ij} = \alpha(x_i y_j - y_j \sum_{k \leq j} W_{ik} y_k)$$

With the Generalized Hebbian Learning rule, the columns of W converges to the M principal components of the input signals.



In case you would define the weight matrix as a MxN where, now, each rows contains the input weights of one output neuron, you would use the following learning rule $\Delta W = \alpha(Y.X^T - LT[YY^T]W)$, where $LT[.]$ denotes the *Lower Triangular* operation which sets all the values above the diagonal to zero.

The following script provides a Python demo showing how the Sanger's learning rule can be used to extract the two principle components of 2D samples : sanger.py.

The legend is the same as for the previous graph, except that we now have two output neurons and therefore four weights to represent and two weight vectors. In addition to using the previous learning rule, with a decreasing learning rate, the samples are also centered to have a zero mean.