Introduction to Artificial Neural Networks

Lecture 8:

Hebbian Learning and Principal Components Analysis (PCA)

By: Ali Motie Nasrabadi

Outline

- **Introduction**
- **Some Intuitive Principles of Self- Organization**
- **Principal Components Analysis**
- **Hebbian-Based Maximum Eigenfilter**
- **Hebbian-Based Principal Components Analysis**

Introduction

- **The purpose of an algorithm for *self-organized learning* or *unsupervised learning* is to discover features in the input data *without* a teacher.**
- **To do so, the algorithm is provided with rules of a *local* nature; the term "local" means that the change applied to the weight is confined to the immediate neighborhood of that neuron.**
- **This chapter is restricted to Hebbian learning. The primary focus of the chapter is *principal components analysis.***

Some Intuitive Principles of Self-Organization

- **PRINCIPLE 1. Modifications in synaptic weights tend to self-amplify.**
- **PRINCIPLE 2. Limitation of resources leads to competition among synapses and therefore the selection of the most vigorously growing synapses (i.e., the fittest) at the expense of the others.**
- **PRINCIPLE 3. Modifications in synaptic weights tend to cooperate.**
- **Note that all of the above three principles relate only to the neural network itself.**

Intuitive Principles

- **PRINCIPLE 4. Order and structure in the activation patterns represent *redundant* information that is acquired by the neural network in the form of knowledge, which is a necessary prerequisite to self-organized learning.**

- **Example: Linsker's model of the mammalian visual system**

Principal Components Analysis (1)

- **Goal: Find an invertible linear transformation T such that the truncation of Tx is optimum in the mean-squared error sense.**

- ***Principal components* analysis provides the right answer.**

PCA (2)

$\mathbf{X}$ an m-dimensional random vector with zero mean

$\mathbf{q}$ a unit vector

$A = \mathbf{X}^T\mathbf{q} = \mathbf{q}^T\mathbf{X}$ the projection of $\mathbf{X}$ onto $\mathbf{q}$

Then the mean of A is also zero, and the variance of A is

$s^2 = E[A^2] = \mathbf{q}^T\mathbf{R}\mathbf{q}$,

where $\mathbf{R}$ is the correlation matrix of $\mathbf{X}$, i.e.,

$\mathbf{R} = \mathrm{E}[\mathbf{X}\mathbf{X}^T]$.

Let $y(\mathbf{q}) = s^2$ be the variance probe, as a function of $\mathbf{q}$.

GOAL : Find extrema of $y(\mathbf{q})$, subject to $\|\mathbf{q}\| = 1$.

PCA (3)

- **The answer can be found by Lagrange multiplier method. We obtain the following:**

The variance probe $y(\mathbf{q})$ has an extremum l when and only when the unit vector is an eigenvector of $\mathbf{R}$ corresponding to the eigenvalue l.

- **Note that all eigenvalues are real and nonnegative since the correlation matrix is symmetric and positive semi-definite. More precisely, we have the following**

PCA (4)

- **(Finite-Dimensional, Baby) Spectral Theorem**

Let $\mathbf{R}$ be an $m \times m$ symmetric matrix with real entries.

Then $\mathbf{R}$ is diagonalizable, that is, there exist m eigenvalues $l_1 \geq l_2 \geq ... \geq l_m$ and corresponding unit eigenvectors $\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_m$ such that

(i) $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_m]$ is an orthogonal matrix, i.e., $\mathbf{Q}^{-1} = \mathbf{Q}^T$;

(ii) $\mathbf{R} = \mathbf{Q}\Lambda\mathbf{Q}^T$.

PCA (5)

- **Since the unit eigenvectors form an orthonormal basis of the data space, we can rewrite any vector in terms of this basis:**

$$\mathbf{x} = \sum_{j=1}^{m} a_j \mathbf{q}_j, \text{ where } a_j = \mathbf{q}_j^T \mathbf{x}.$$

- **Then we approximate the data vector by truncating the expansion:**

$$\hat{\mathbf{x}} = \sum_{j=1}^{l} a_j \mathbf{q}_j, \text{ where } l \leq m.$$

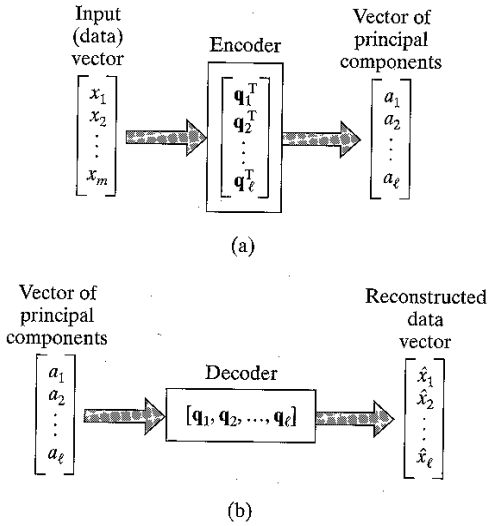
- **The *approximation error vector* is the difference of these two:**

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$$

- **Note that the error vector is orthogonal to the approximating data vector. This is known as the *principle of orthogonality.***

PCA (6)

- **The linear projecton given by truncation represents an *encoder* for the approximate representation of the data vector. See the figure.**



1

Hebbian-Based Maximum Eigenfilter

- **A single liner neuron with a Hebbian-type adaptation rule can evolve into a filter for the first principal component of the input.**

- **Formulation of the algorithm:**

  Let $\mathbf{x}(n) = [x_1(n), x_2(n), ..., x_m(n)]^T$ be the input vect or and

  $\mathbf{w}(n) = [w_1(n), w_2(n), ..., w_m(n)]^T$ be the synaptic weight ve ctor at discrete time n.

  The output of the neuron is

  $$y(n) = \mathbf{x}^T(n)\mathbf{w}(n).$$

  Update the weights by

  $$\mathbf{w}(n+1) = \mathbf{w}(n) + hy(n)[\mathbf{x}(n) - y(n)\mathbf{w}(n)].$$

Lecture 8-12

Maximum Eigenfilter

- **How do we derive the above algorithm from Hebb's postulate of learning?**
  - **Hebb's postulate can be written as**

$$w_i(n+1) = w_i(n) + \eta y(n) x_i(n) \quad \text{for } i = 1,2,\ldots,m.$$

  - **However, this rule leads to the *unlimited* growth of the weights. We may overcome this problem by incorporating *saturation* or *normalization* in the learning:**

$$w_i(n+1) = \frac{w_i(n) + \eta y(n) x_i(n)}{\left(\sum_{i=1}^{m} [w_i(n) + \eta y(n) x_i(n)]^2\right)^{1/2}} \quad \text{for } i = 1,2,\ldots,m.$$

Maximum Eigenfilter

  - **Expand this into power series in the learning parameter, and ignoring higher degree terms,**

$$w_i(n+1) = w_i(n) + \eta y(n)[x_i(n) - y(n) w_i(n)].$$

  - **We may think of**

$$x_i'(n) = x_i(n) - y(n) w_i(n)$$

as the *effective input*. Then we may rewrite the above equation

$$w_i(n+1) = w_i(n) + \eta y(n) x_i'(n).$$

Maximum Eigenfilter

- **We can rewrite the last equation by substituting the output value, and obtain a *nonlinear stochastic difference equation*:**

$$\mathbf{w}(n+1) = \mathbf{w}(n) + h[\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n) - \mathbf{w}^T(n)\mathbf{x}(n)\mathbf{x}^T(n)\mathbf{w}(n)\mathbf{w}(n)].$$

- **Does our algorithm converge?**

Maximum Eigenfilter

- **Consider a generic stochastic approximation algorithm:**

$$\mathbf{w}(n+1) = \mathbf{w}(n) + h(n)h(\mathbf{w}(n), \mathbf{x}(n)).$$

- **We *can* (but won't in detail) analyze the convergence of such algorithm by associating a *deterministic ordinary differential equation* to it.**

Maximum Eigenfilter

- **Rough explanation: Assume certain conditions. See the textbook, page 407 for details. The key conditions are:**

  4. The limit $\bar{h}(w) = \lim_{n\to\infty} E[h(\mathbf{w},\mathbf{X})]$ exists for each $\mathbf{w}$.

  5. There exists a locally asymptotic ally stable solution to the ODE

  $$\frac{d}{dt}\mathbf{w}(t) = \bar{h}(\mathbf{w}(t))$$

  where t denotes continuous time.

- **We maintain a savvy silence on what we mean by local asymptotical stability. See Chapter 14 if interested.**

Maximum Eigenfilter

- **Asymptotic Stability Theorem. Under the above conditions,** $\lim_{n\to\infty} \mathbf{w}(n) = \mathbf{q}_1$

almost always.

- **We can check the maximum eigenfilter satisfies all of the conditions of the above theorem.**

- **In summary,**
  - **The variance of the model output approaches the largest eigenvalue of the correlation matrix:**

    $$\lim_{n\to\infty} s^2(n) = l_1$$

  - **The synaptic weight vector of the model approaches the associated unit eigenvector:** $\lim_{n\to\infty} \mathbf{w}(n) = \mathbf{q}_1$

Hebbian-Based Principal Components Analysis

- **Expanding a single linear neuron model into a feedforward network with a ***single layer of linear neurons*****, we can perform principal components analysis of arbitrary size.**

- **Fix notations:**

The network has m inputs $x_1(n), x_2(n), ..., x_m(n)$,

common to all l neurons.

The weights of the neuron j at time n is

$\mathbf{w}_j(n) = [w_{j1}(n), w_{j2}(n), ..., w_{jm}(n)]^T$.

The output of the neuron j at time n is $y_j(n)$

Hebbian-Based PCA

- **Generalized Hebbian Algorithm:**
  - **1. Initialize the weights to a small random values at n=1. Assign a small positive value to the learning-rate parameter.**
  - **2. Compute**

$$y_j(n) = \sum_{i=1}^{m} w_{ji}(n) x_i(n),$$

$$\Delta w_{ji}(n) = h\left[y_j(n) x_i(n) - y_j(n) \sum_{k=1}^{j} w_{ki}(n) y_k(n) \right],$$

for $j = 1,2,...,l$, and $i = 1,2,...,m$..

  - **3. Increment n by 1, and go to step 2. Repeat until all synaptic weights reach their steady-state values.**

Hebbian-Based PCA

- The GHA computes weights convergent to the first l eigenvectors. The convergence can be proven as in the case of maximum eigenfilter.

- Furthermore, in the limit,

$$\hat{x}(n) = \sum_{k=1}^{l} y_k(n)\mathbf{q}_k.$$

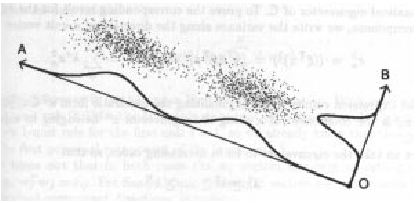
This equation may be viewed as one of ***data reconstruction***.

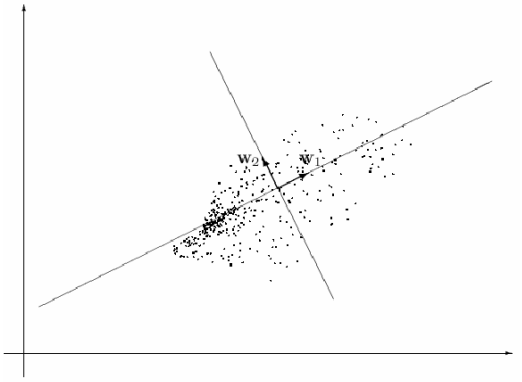
Hebbian-Based PCA

- Note that it is not necessary to compute the correlation matrix. The resulting computational savings can be enormous when m is very large, and l is a small fraction of m.

- Although the convergence is proven for a time-varying learning-rate parameter, in practice we set the learning-rate parameter to a small constant.

A two dimentional pattern with Principle directions

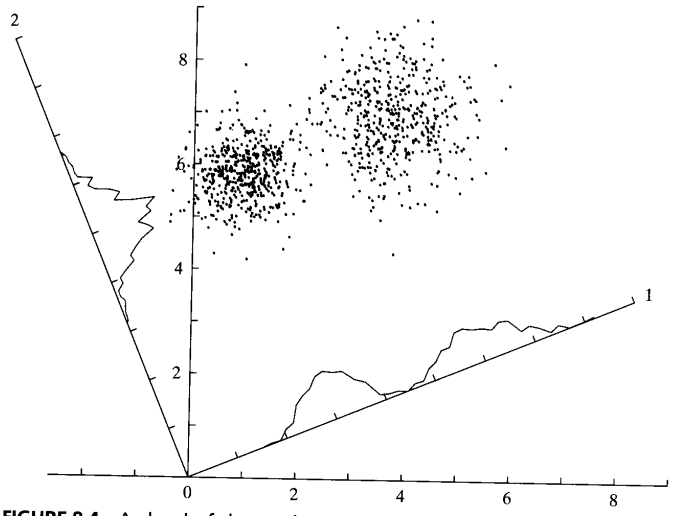




Summary

- **How useful is PCA?**
  - **For good data compression, PCA offers a useful self-organized learning procedure.**
  - **When a data set consists of several *clusters*, the principal axes found by PCA usually pick projections with good separation. PCA provides an effective basis for *feature extraction* in this case. See the figure.**
  - **Therefore, PCA is applicable as the *preprocessor* for a supervised network: Speed up convergence of the learning process by decorrelating the input data.**

Summary



- **PCA doesn't seem to be sufficient in biological perceptual systems.**