Hebbian Learning, Principal Component Analysis, and Independent Component Analysis

15-496/782: Artificial Neural Networks

Kornel Laskowski

Spring 2004

(based on slides by D. Touretzky, Spring 2002)

Hebbian Learning

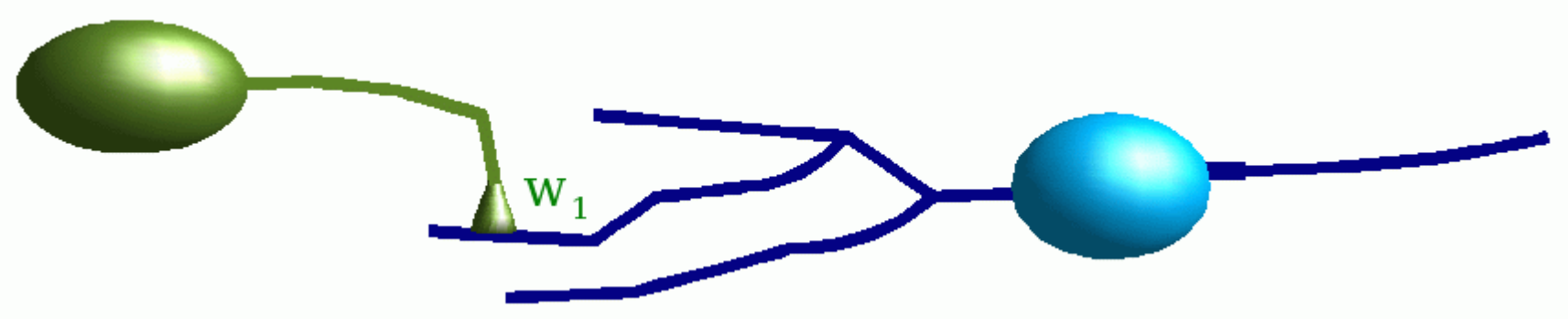
Donald Hebb wrote in 1949:

When an axon in cell A is near enough to excite cell B and repeatedly and persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency in firing B is increased.

Today this growth process is known as *Hebbian learning*.

The Hebbian Synapse

A biological system which exhibits Hebbian learning.



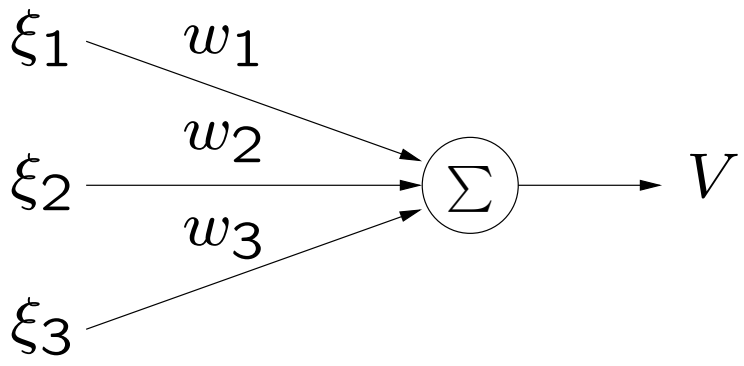
$$\Delta w_1(t) \;\propto\; x(t)\, y(t) \tag{1}$$

The Hebbian Neuron

A computational system which implements Hebbian learning.

Let's assume a linear unit; experiment shows this is largely sufficient:

$$V = \sum_j w_j \xi_j = \bar{w}^T \bar{\xi} \tag{2}$$

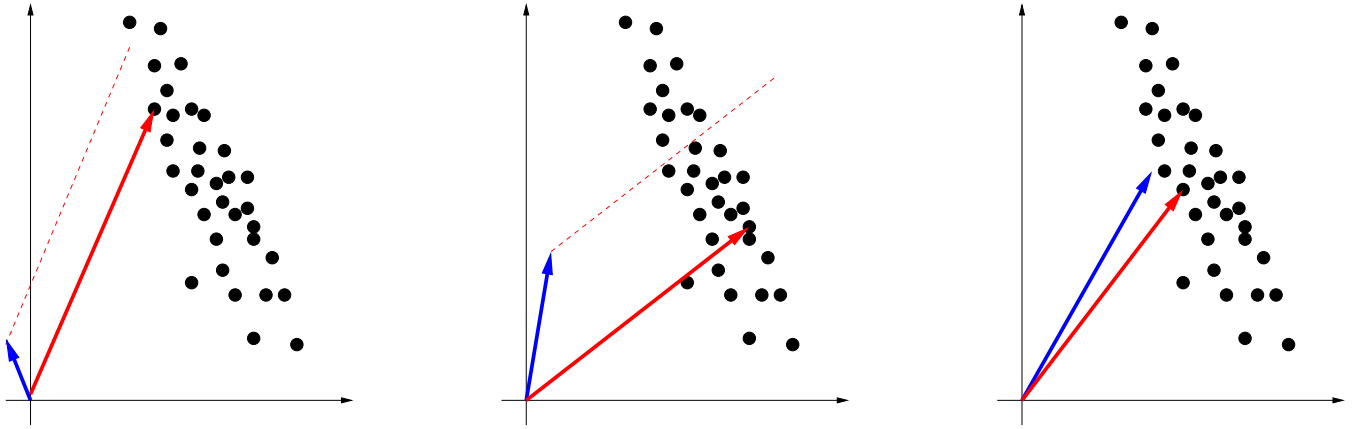


Plain Hebbian learning:

$$\Delta \bar{w} = \eta \, V \, \bar{\xi} \tag{3}$$

Hebbian Learning, Non-zero-mean Data

Recall that $\Delta\bar{w} = \eta V \bar{\xi}$ and that $V = \bar{w}^T \bar{\xi}$.

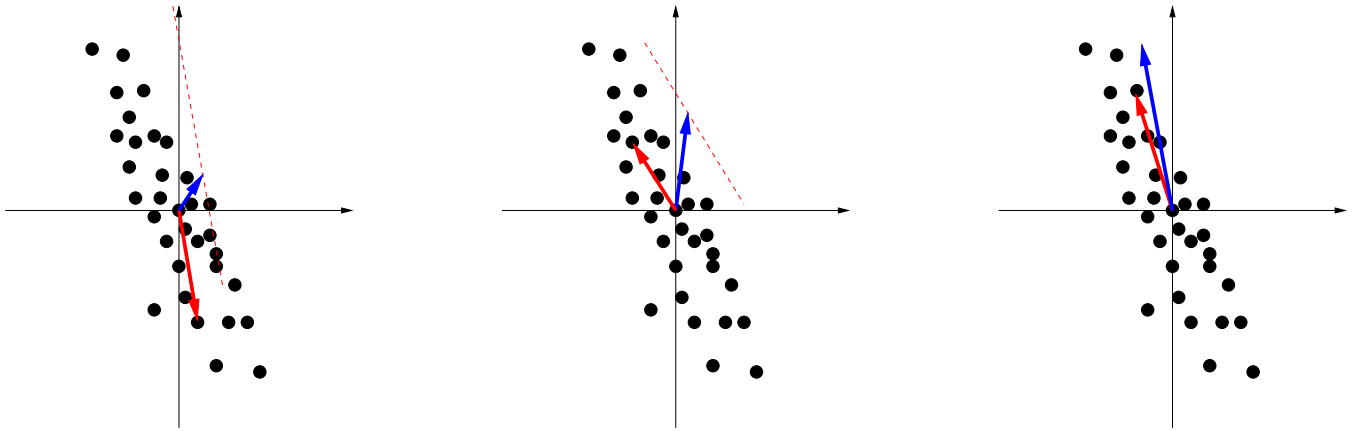


But note that if the initial $\bar{w}$ points away from the cluster, the dot product with $\bar{\xi}$ will be negative, and $\bar{w}$ will ultimately point in exactly the opposite direction.

Interpretation awkward ...

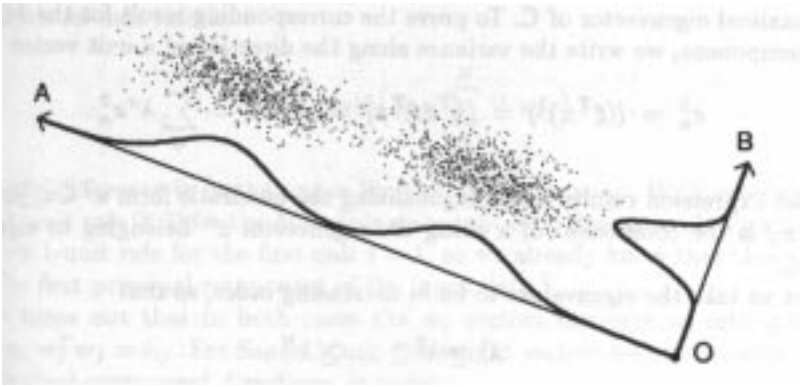
Plain Hebbian Learning, Zero-mean Data

Recall that $\Delta\bar{w} = \eta V \bar{\xi}$ and that $V = \bar{w}^T \bar{\xi}$.



In this case, the weight vector will ultimately align itself with the *direction of greatest variance* in the data.

The Direction of Greatest Variance



If the data has zero mean, Hebbian learning will adjust the weight vector $\bar{w}$ so as to maximize the variance in the output V.

When projected onto this direction, the data $\{\bar{\xi}\}$ will exhibit variance greater than in any other direction.

This direction is also known as the largest *principal component* of the data.

Locality of Plain Hebbian Learning

Looking at one synapse at a time (from input unit j)

$$\Delta w_j \;=\; \eta\, V\, \xi_j \tag{4}$$

This is a *local* update rule.

The Hebbian synapse, characterized by a weight w_j, is modified as a function of the activity of only the two units it connects.

By contrast, backprop is *non-local*. The update rule involves the backpropagation of a distant error signal, computed (potentially many) layers above it.

This makes Hebbian learning a likely candidate for biological systems.

Hebbian vs Competitive Learning

Plain Competitive Learning	Plain Hebbian Learning

Similarities

Unsupervised training methods (no teacher, no error signal).
Frequently used online .
Strong connection to biological systems.
Linear units: $V = \bar{w}^T \bar{\xi}$

Differences

Exactly one output must be active.	No constraint imposed by neighboring units.
Only winner's weights updated at every epoch.	All weights updated at every epoch.
$\Delta\bar{w} = \eta\,\bar{\xi}$	$\Delta\bar{w} = \eta\, V\, \bar{\xi}$

A Consequence of $\Delta\bar{w} = \eta\, V\, \bar{\xi}$

Let's look at the update rule Eq 3 given our expression for V in Eq 2:

$$\begin{aligned} \Delta\bar{w} &= \eta\, V\, \bar{\xi} \\ &= \eta \left(\bar{w}^T\, \bar{\xi}\right) \bar{\xi} \qquad \text{(inner product)} \\ &\equiv \eta \left(\bar{\xi}\, \bar{\xi}^T\right) \bar{w} \qquad \text{(outer product)} \end{aligned} \tag{5}$$

Given a current value of the weight vector $\bar{w}$, the weight update $\Delta\bar{w}$ will be a function of the outer product of the input pattern with itself.

Note that learning is incremental; that is, a weight update is performed each time the neuron is exposed to a training pattern $\bar{\xi}$.

We can compute an expectation for $\Delta\bar{w}$ if we take into account the distribution over patterns, $P(\bar{\xi})$.

An Aside: $\left(\bar{w}^T\,\bar{\xi}\right)\bar{\xi} = \left(\bar{\xi}\,\bar{\xi}^T\right)\bar{w}$

$$
\begin{aligned}
\left(\bar{w}^T\bar{\xi}\right)\bar{\xi} &= (w_1\xi_1 + w_2\xi_2 + \cdots + w_N\xi_N)\begin{bmatrix}\xi_1\\ \xi_2\\ \vdots\\ \xi_N\end{bmatrix} \\
&= \begin{bmatrix} w_1\xi_1^2 & + & w_2\xi_1\xi_2 & + & \cdots & + & w_N\xi_1\xi_N \\ w_1\xi_2\xi_1 & + & w_2\xi_2^2 & + & \cdots & + & w_N\xi_2\xi_N \\ & & & \vdots & & & \\ w_1\xi_N\xi_1 & + & w_2\xi_N\xi_2 & + & \cdots & + & w_N\xi_N^2 \end{bmatrix} \\
&= \begin{bmatrix} \xi_1^2 & \xi_1\xi_2 & \cdots & \xi_1\xi_N \\ \xi_2\xi_1 & \xi_2^2 & \cdots & \xi_2\xi_N \\ \vdots & \vdots & \ddots & \vdots \\ \xi_N\xi_1 & \xi_N\xi_2 & \cdots & \xi_N^2 \end{bmatrix}\begin{bmatrix} w_1\\ w_2\\ \vdots\\ w_N\end{bmatrix} \\
&= \left(\bar{\xi}\bar{\xi}^T\right)\bar{w}
\end{aligned}
$$

Or just $\left(\bar{w}^T\bar{\xi}\right)\bar{\xi} = \bar{\xi}\left(\bar{w}^T\bar{\xi}\right) = \bar{\xi}\left(\bar{\xi}^T\bar{w}\right) = \bar{\xi}\bar{\xi}^T\bar{w} = \left(\bar{\xi}\bar{\xi}^T\right)\bar{w}.$

The Correlation Matrix

Taking the expectation of Eq 5 under the input distribution $P(\bar{\xi})$:

$$\begin{aligned} \langle \Delta \bar{w} \rangle &= \eta \left\langle \bar{\xi}\, \bar{\xi}^T \right\rangle \bar{w} \\ &\equiv \eta\, \mathbf{C}\, \bar{w} \end{aligned} \tag{6}$$

where we have defined the *correlation matrix* as

$$\mathbf{C} \equiv \left\langle \bar{\xi}\, \bar{\xi}^T \right\rangle \tag{7}$$

$$= \begin{bmatrix} \langle \xi_i^2 \rangle & \langle \xi_1 \xi_2 \rangle & \cdots & \langle \xi_1 \xi_N \rangle \\ \langle \xi_2 \xi_1 \rangle & \langle \xi_2^2 \rangle & \cdots & \langle \xi_2 \xi_N \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \xi_N \xi_1 \rangle & \langle \xi_N \xi_2 \rangle & \cdots & \langle \xi_N^2 \rangle \end{bmatrix}$$

and N is the number of inputs to our Hebbian neuron.

More on the Correlation Matrix

Similar to the covariance matrix

$$\Sigma = \left\langle \left(\bar{\xi} - \bar{\mu}\right)\left(\bar{\xi} - \bar{\mu}\right)^T \right\rangle \tag{8}$$

$\mathbf{C}$ is the second moment of the input distribution about the origin ($\mathbf{\Sigma}$ is the second moment of the input distribution about the mean) If $\bar{\mu} = \bar{0}$, then $\mathbf{C} = \mathbf{\Sigma}$.

Like $\mathbf{\Sigma}$, $\mathbf{C}$ is symmetric. Symmetry implies that all eigenvalues are real and all eigenvectors are orthogonal.

Additionally, because $\mathbf{C}$ is an outer product, it's positive-definite. This means that all eigenvalues are not just real, they're also all non-negative.

The Stability of Hebbian Learning

Recalling Eq 6,

$$\langle \Delta \bar{w} \rangle \;=\; \eta \, \mathbf{C} \, \bar{w}$$

From a mathematical perspective, this is a discretized version of a linear system of coupled first-order differential equations,

$$\frac{d\bar{w}}{dt} \;=\; \mathbf{C} \, \bar{w} \tag{9}$$

whose natural solutions are

$$\bar{w}(t) \;=\; e^{\lambda t} \, \bar{u} \tag{10}$$

where λ is a scalar and $\bar{u}$ is a vector independent of time. If we wanted to solve this system of equations, we'd still need to solve for both λ and $\bar{u}$.

There turn out to be multiple pairs $\{\lambda, \bar{u}\}$; then $\bar{w}(t)$ is a linear combination of terms like the one in Eq 10.

Note that if any $\lambda_i > 0$, $\bar{w}(t)$ blows up.

Hebbian Learning Blows Up

To see that Eq 10 represents solutions to Eq 9,

$$\begin{aligned} \mathbf{C}\,\bar{w} &= \frac{d\bar{w}}{dt} \\ \mathbf{C}\,\left(e^{\lambda t}\,\bar{u}\right) &= \frac{d}{dt}\left(e^{\lambda t}\,\bar{u}\right) \\ &= \lambda\, e^{\lambda t}\,\bar{u} \\ \Rightarrow \mathbf{C}\,\bar{u} &= \lambda\,\bar{u} \end{aligned} \tag{11}$$

So the λ_i's are the eigenvalues of the correlation matrix $\mathbf{C}$!

1. $\mathbf{C}$ is an outer product $\Rightarrow$ all $\lambda \geq 0$.
2. From Eq 10, if any $\lambda > 0$ then $\bar{w} \rightarrow \infty$.
3. The λ_i's cannot all be zero (true only for the zero matrix).

So plain Hebbian learning blows up.

Plain Hebbian Learning: Conclusions

We've shown that for zero-mean data, the weight vector aligns itself with the direction of maximum variance as training continues.

This direction corresponds to the eigenvector of the correlation matrix $\mathbf{C}$ with the largest corresponging eigenvalue. If we decompose the current $\bar{w}$ in terms of the eigenvectors of $\mathbf{C}$, $\bar{w} = \sum_i \alpha_i \, \bar{u}_i$, then the expected weight update rule

$$\begin{aligned} \langle \Delta \bar{w} \rangle &= \eta \, \mathbf{C} \, \bar{w} \\ &= \eta \, \mathbf{C} \sum_i \alpha_i \, \bar{u}_i \\ &= \eta \sum_i \alpha_i \, \lambda_i \, \bar{u}_i \end{aligned} \tag{12}$$

will move the weight vector $\bar{w}$ towards eigenvector $\bar{u}_i$ by a factor proportional to λ_i. Over many updates, the eigenvector with the largest λ_i will drown out the contributions from the others.

But because of this mechanism, the magnitude of $\bar{w}$ blows up.

Stabilizing Hebbian Learning

How to keep $\bar{w}$ from blowing up?

We could renormalize $\bar{w}$:

$$\bar{w}_*^{(\tau+1)} = \bar{w}^{(\tau)} + \Delta\bar{w}^{(\tau)} \tag{13}$$

$$\bar{w}^{(\tau+1)} = \frac{\bar{w}_*^{(\tau+1)}}{\|\bar{w}_*^{(\tau+1)}\|} \tag{14}$$

This ensures that $\bar{w}$ always has unit length.

But it obliges us to give up our claim that learning is local — every synapse (w_i) must know what every other synapse of the unit is doing.

Oja's Rule

An alternative, suggested by Oja, is to add *weight decay*.

The weight decay term is proportional to proportional to V^2 (recall $V = \bar{w}^T \bar{\xi}$ is the (output) activation of our Hebbian neuron).

$$\Delta \bar{w} = \eta \, V \left(\bar{\xi} - V \bar{w} \right) \tag{15}$$

$\bar{w}$ approaches unit length, without any additional explicit effort.

But it retains the property that it points in the direction of maximum variance in the data (the largest principal component).

Sejnowski's Hebbian Covariance Rule

$$\Delta\bar{w} \;=\; \eta\,(V - \langle V\rangle)\left(\bar{\xi} - \langle\bar{\xi}\,\rangle\right) \tag{16}$$

Then under the input distribution $P(\bar{\xi})$,

$$\begin{aligned}
\langle\Delta\bar{w}\rangle &= \eta\left\langle (V - \langle V\rangle)\left(\bar{\xi} - \langle\bar{\xi}\,\rangle\right)\right\rangle \\
&= \eta\,\langle V\bar{\xi} - V\langle\bar{\xi}\,\rangle - \bar{\xi}\langle V\rangle + \langle V\rangle\langle\bar{\xi}\,\rangle\rangle \\
&= \eta\left(\langle V\bar{\xi}\,\rangle - \langle V\rangle\langle\bar{\xi}\,\rangle - \langle V\rangle\langle\bar{\xi}\,\rangle + \langle V\rangle\langle\bar{\xi}\,\rangle\right) \\
&= \eta\left(\langle V\bar{\xi}\,\rangle - \langle V\rangle\langle\bar{\xi}\,\rangle\right) \\
&= \eta\left(\langle\left(\bar{w}^T\bar{\xi}\right)\bar{\xi}\,\rangle - \langle\left(\bar{w}^T\bar{\xi}\right)\rangle\langle\bar{\xi}\,\rangle\right) \\
&= \eta\left(\langle\bar{\xi}\bar{\xi}^T\rangle - \langle\bar{\xi}\rangle\langle\bar{\xi}^T\rangle\right)\bar{w}
\end{aligned} \tag{17}$$

Subtracting the mean allows $\Delta\bar{w}$ to be negative, reducing the weight vector.

Principal Component Analysis (PCA)

A tool from statistics for data analysis.

Can reveal structure in high-N-dimensional data that is not otherwise obvious.

Like Hebbian learning, it discovers the direction of maximum variance in the data. But then in the $(N-1)$-dimensional subspace perpendicular to that direction, it discovers the direction of maximum *remaining* variance, and so on for all N.

The result is an ordered sequence of principal components. These are equivalently the eigenvectors of the correlation matrix $\mathbf{C}$ for zero-mean data, ordered by magnitude of eigenvalue in descending order. They are mutually orthogonal.

PCA: Batch Algorithm

Because of this, eigenvalue decomposition is an effective means of performing principal component analysis, *if we can afford to retain all the data and do the computation offline*.

This is typically the case with PCA.

```
[nData,nDim] = size(xi);
xi0 = xi - ones(nData,1) * mean(xi,1);
C = cov(xi0);
[u,lambda] = eig(C);
```

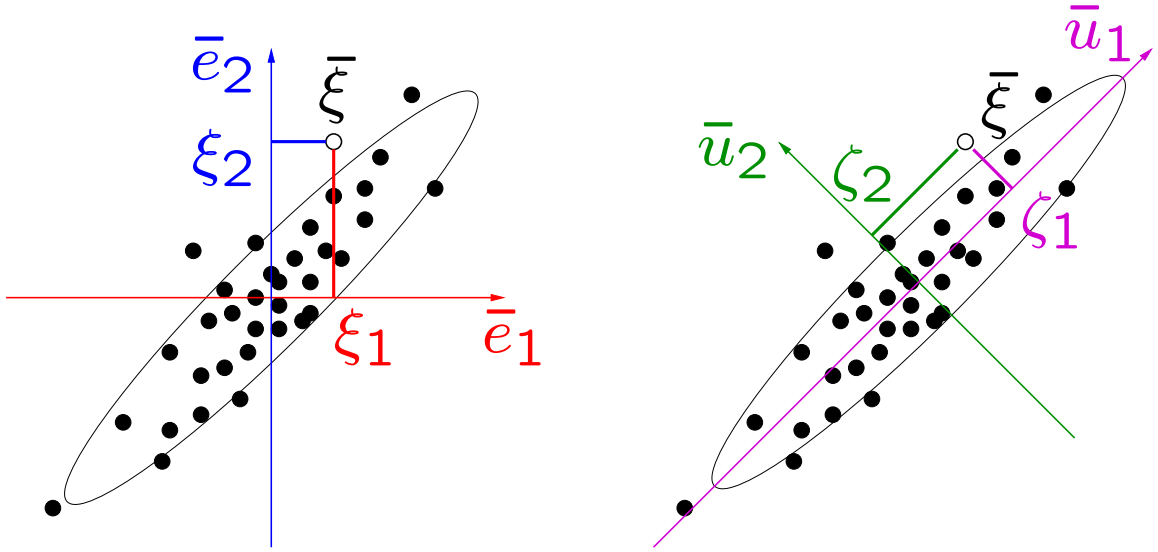
PCA Decorrelates Data

In producing the N principal components (which are orthogonal), PCA provides an alternate, complete, orthonormal basis of N-space.

$$\bar{\xi} \;=\; \sum_{i=1}^{N} \xi_i \, \bar{e}_i \;=\; \sum_{i=1}^{N} \zeta_i \, \bar{u}_i \tag{18}$$

$$\xi_1 \, \bar{e}_1 + \xi_2 \, \bar{e}_2 \;=\; \zeta_1 \, \bar{u}_1 + \zeta_2 \, \bar{u}_2 \tag{19}$$

In this basis, the data are uncorrelated.



The Karhunen-Loève Transform (KLT)

How do we get the ζ_i in Eq 18? Recall that the $\bar{u}_i$'s are orthogonal:

$$\begin{aligned} \bar{u}_j^T \bar{\xi} &= \bar{u}_j^T \sum_{i=1}^{N} \zeta_i \bar{u}_i \\ &= \sum_{i=1}^{N} \zeta_i \bar{u}_j^T \bar{u}_i \\ &= \zeta_j \end{aligned} \tag{20}$$

Let's organize the $\bar{u}_i$'s as columns of a square matrix $\mathbf{U}$. Then to transform all the elements of $\bar{\xi}$ at once:

$$\bar{\zeta} = \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ \vdots \\ \zeta_N \end{bmatrix} = \underbrace{\begin{bmatrix} \cdots & \bar{u}_1^T & \cdots \\ \cdots & \bar{u}_2^T & \cdots \\ & \vdots & \\ \cdots & \bar{u}_N^T & \cdots \end{bmatrix}}_{\mathbf{U}^T} \underbrace{\begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix}}_{\bar{\xi}} \tag{21}$$

PCA: Geometric Interpretation

The Karhunen-Loève transform operates on zero-mean data; it is a function of the covariance matrix Σ only.

The covariance matrix is a *second-order* statistic. This makes PCA a second-order method; it ignores higher-order statistics of the data.

We start with the original covariance matrix in the original orthonormal basis.

PCA applies *a set of rotations* to the original orthonormal basis, resulting in another orthonormal basis, such that the off-diagonal entries of the covariance matrix become zero and the diagonal entries of the covariance matrix take on the maximum values possible.

Data Whitening/Sphering

The Karhunen-Loève Transform $\bar{\zeta} = \mathbf{U}^T \bar{\xi}$ decorrelates data. This means that covariance matrix of $\{\bar{\zeta}\}$ is *diagonal*.

We can additionally ensure the transformed data has unity covariance by scaling each dimension of $\bar{\zeta}$ (the eigenvalues of a diagonal $\mathbf{\Sigma}$ are the variances) by the inverse of its standard deviation:

$$\bar{\psi} = \underbrace{\begin{bmatrix} \frac{1}{\sqrt{\lambda_1}} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sqrt{\lambda_2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sqrt{\lambda_N}} \end{bmatrix}}_{\sqrt{\Lambda^{-1}}} \mathbf{U}^T \bar{\xi} \tag{22}$$

Used extensively everywhere. In particular, used as a preprocessing step when training backprop networks.

Data Compression with PCA

If we wanted to transmit our original zero-mean dataset $\{\bar{\xi}\}$, we could send the zero-mean, diagonal-covariance dataset $\{\bar{\zeta}\}$ if the receiver knew the KL transform $\mathbf{U}^T$ we used to decorrelate it.

Suppose that several, say P, of the eigenvalues were very small, ie. that the contribution of the corresponding eigenvectors was (almost) statistically insignificant.

Then we wouldn't need to send the coefficients ζ_{N-P+1} through ζ_N of every datapoint for (almost) perfect reconstruction of the original datapoint $\bar{\xi}$.

This results in a compression factor of $\frac{N-P}{N}$ (plus the overhead of communicating the $N \times N$ transform if necessary).

PCA with Neural Networks

We've seen that Hebbian learning, with appropriate provisions for preventing blow up, extracts the largest principal component.

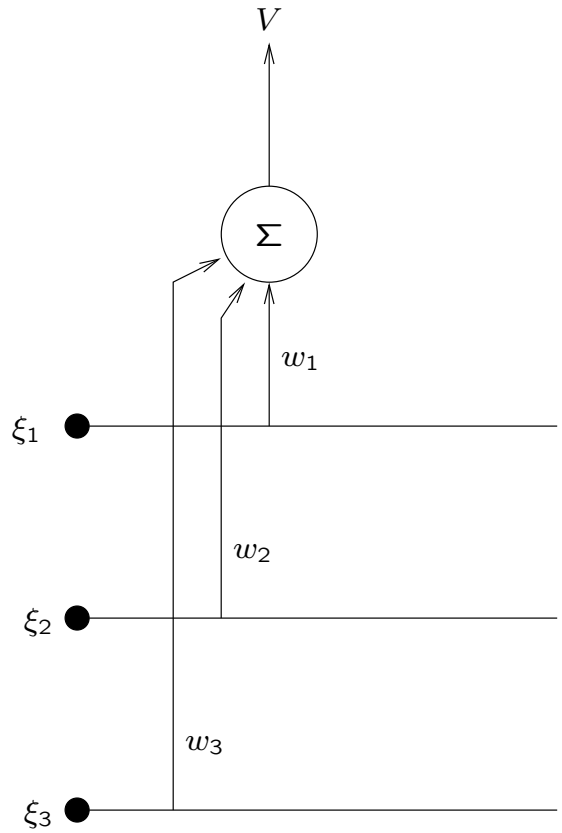
Let's take a look at two different neural network architectures capable of extracting more of them.

1. Cascading multiple Hebbian neurons.

2. Autoencoder networks (we've already seen these before).

The Hebbian Neuron, Revisited

An alternate diagram of a Hebbian neuron:



Extracting Multiple Principal Components

Algorithm:

1. Subtract the contribution of the first principal component.
2. Drive the difference into another Hebbian neuron.
3. This extracts the next principal component.
4. Subtract its contribution. Goto step 2.

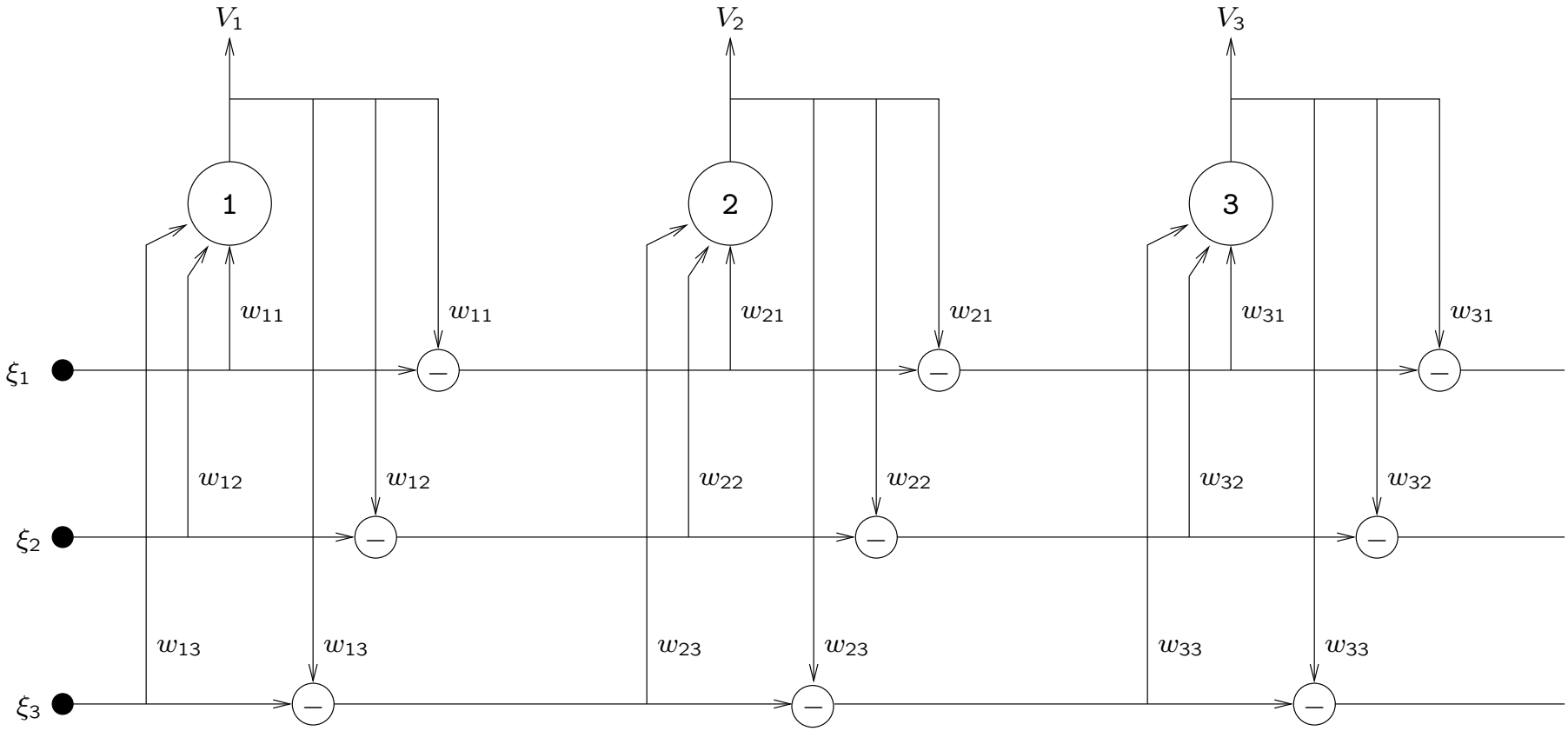
With N Hebbian neurons, we'll get all N principal components.

Embodied in Sanger's rule:

$$\Delta w_{ij} = \eta V_i \left(\xi_j - \sum_{k=1}^{i} V_k w_{kj} \right) \tag{23}$$

This is a generalization of Oja's rule.

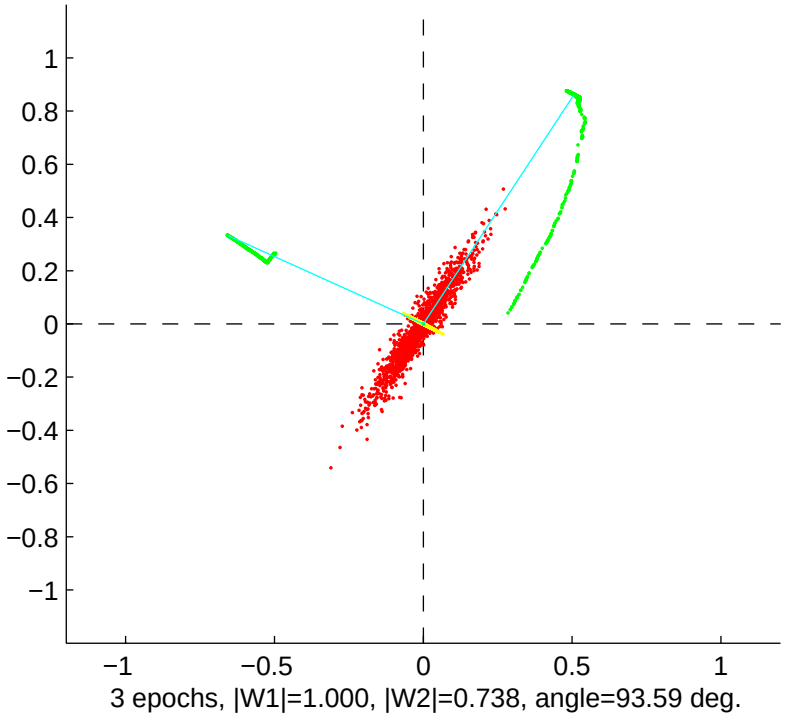
Sanger's Rule in Action



pcademo

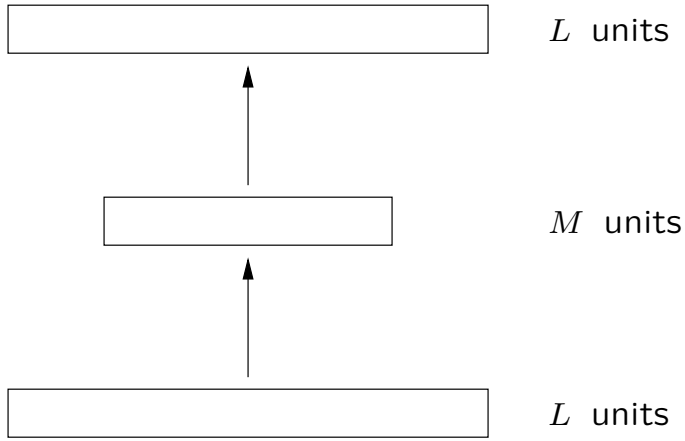
In `matlab/pca`: `pcademo`.

Set the flag `zeromean = 1` (or, incorrectly, leave `zeromean` to be zero for non-zero-mean data).



Autoencoder Networks

We've already seen another neural network implementation of PCA, which we trained with backprop. It consisted of exclusively linear units.



The network tries to reproduce the input in the output, inducing an short encoding in the hidden layer. This encoding retains the maximum amount of information about the input in a smaller dimensional space such that the input can be reconstructed.

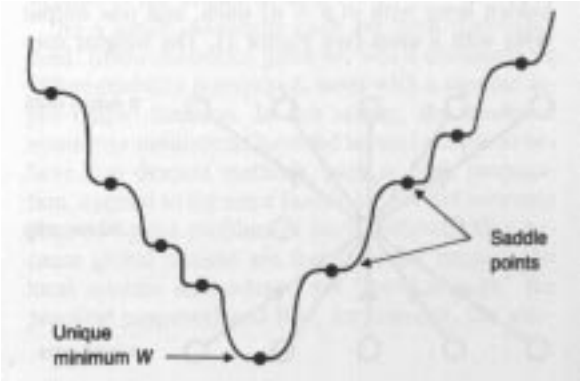
Autoencoder Networks Do PCA

It turns out that the M hidden units extract the first M principal components.

Autoencoder networks can be used for dimensionality reduction, compression, etc.

Because all units are linear, there is a single global minimum.

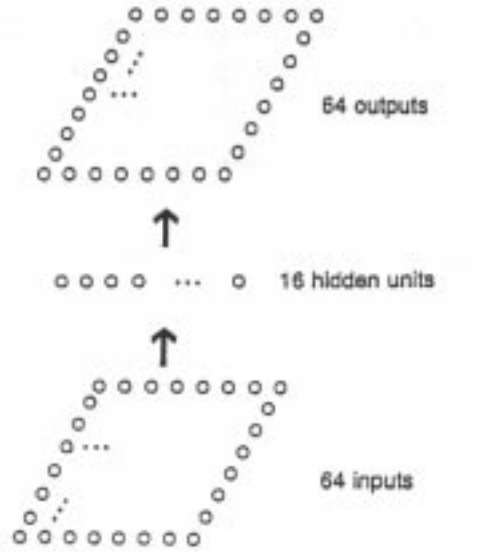
Baldi (1989) has shown that there are also multiple saddle points.



Cottrell (1988): Image Compression

Applied the autoencoder PCA algorithm to the compression of an image.

Training data consisted of 8×8 randomly selected patches from anywhere in the image. Tested on entire image applying it consecutively to all 8×8 non-overlapping patches.



Used logistic non-linearity in the units; but claim that although the hidden units do not compute the principal components, the space they do compute is spanned by the 16 principal components.

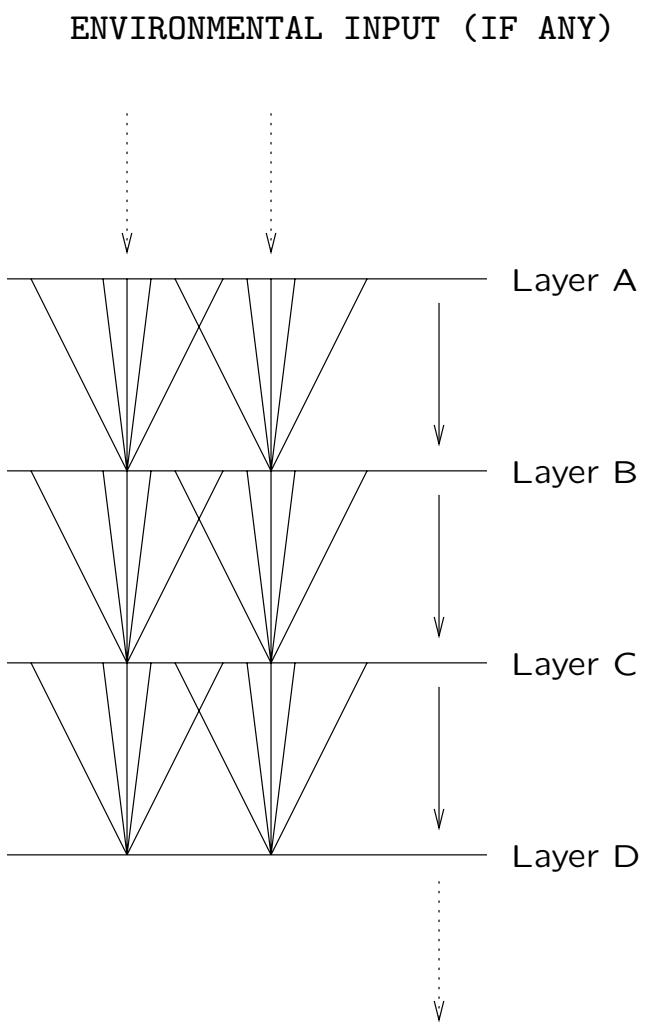
Cottrell (1988): Results

The original is on the left. The right figure is a reproduction using 8 hidden units and 32 quantization levels in the hidden units. With 16 hidden units and no quantization, it is hard to tell the difference.

Linsker (1988): Self-Adaptive Feature Detection in Visual Cortex

Constructed a multi-layered architecture with Hebbian neurons to simulate early processing stages in the visual system.

Each cell excited by an overlying neighborhood of cells in the previous layer.



Linsker (1988): Modified Hebb Rule

All cells linear, computing:

$$V = a_1 + \sum_j w_j \xi_j \tag{24}$$

Update rule:

$$\Delta w_i = a_2 V \xi_i + a_3 \xi_i + a_4 V + a_5 \tag{25}$$

Cells in each layer trained to maturity before training next layer.

Layer A exposed only to white noise, to understand how feature-analyzing cells could emerge even before birth in mammals.

Linsker (1988): Findings

Found that cells in Layer B, once matured, compute the local average of the activity in the overlying region of Layer A.

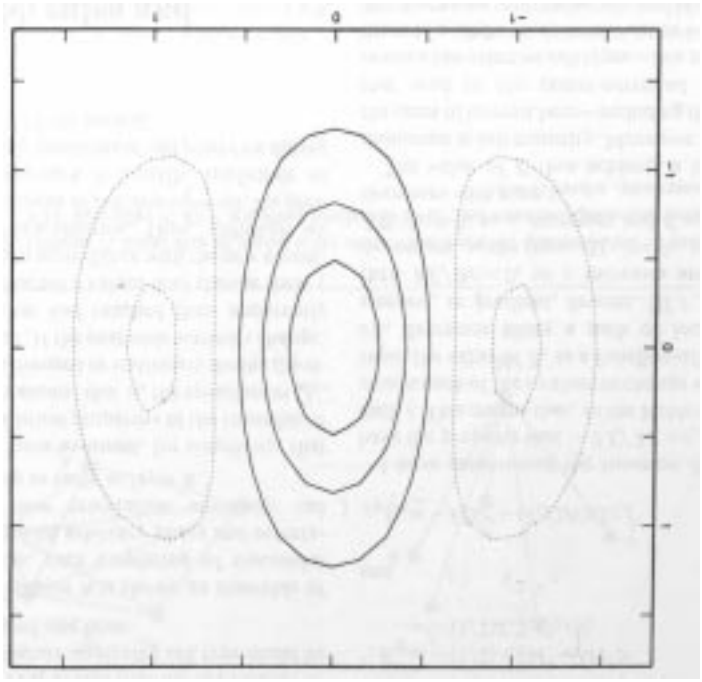
Due to neighborhood overlap, if one cell's activity in Layer B happens to be high, its neighbors' activities are also high.

In Layer C, *center-surround* type cells emerge — responding maximally to a bright spot in layer B surrounded by a dark region, and vice versa.

In Layer D, *orientation-selective* cells emerge — responding maximally to a bright edge against a dark background, and vice versa.

Linsker (1988): Conclusions

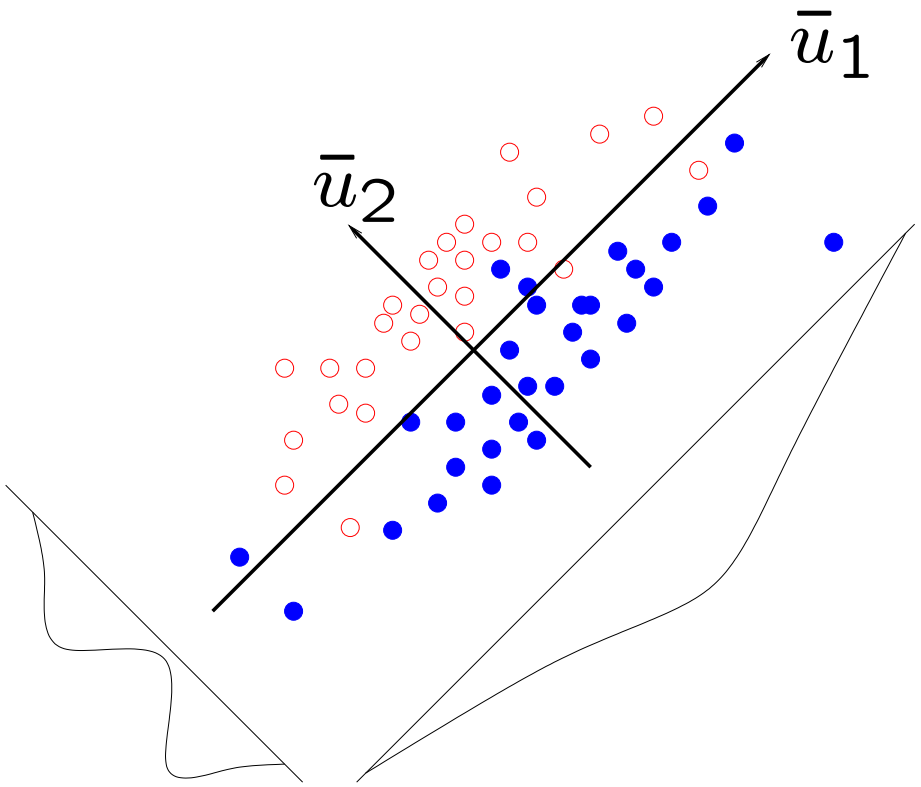
Receptive field map of a computed orientation-selective cell:



This layered construction of feature detectors is collectively due to each cell's individual efforts to maximize the variance of its output. Each cell's output optimally preserves the information contained in its set of input activities.

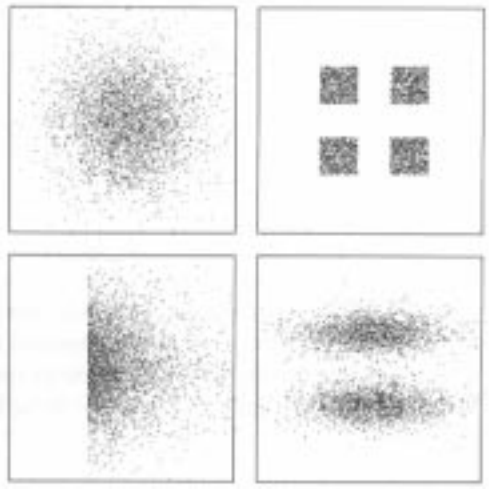
A Limitation of the Principle of Variance Maximization

One possibility: when the variance due to noise is high. If used for dimensionality reduction, may "accidentally" eliminate useful components and keep uninformative ones.



A More Serious Limitation

Recall that PCA looks at the covariance matrix only. What if the data is not well described by the covariance matrix?



The only distribution which is uniquely specified by its covariance (with the subtracted mean) is the Gaussian distribution. Distributions which deviate from the Gaussian are poorly described by their covariances.

Faithful vs Meaningful Representations

Even with non-Gaussian data, variance maximization leads to the most faithful representation in a reconstruction error sense (recall that we trained our autoencoder network using a mean-square error in an input reconstruction layer).

The mean-square error measure implicitly assumes Gaussianity, since it penalizes datapoints close to the mean less that those that are far away.

But it does not in general lead to the most meaningful representation.

We need to perform gradient descent in some function other than the reconstruction error.

A Criterion Stronger than Decorrelation

The way to circumvent these problems is to look for components which are *statistically independent*, rather than just uncorrelated.

For statistical independence, we require that

$$p(\xi_1, \xi_2, \cdots, \xi_N) \;=\; \prod_{i=1}^{N} p(\xi_i) \tag{26}$$

For uncorrelatedness, all we required was that

$$\langle \xi_i \xi_j \rangle - \langle \xi_i \rangle \langle \xi_j \rangle \;=\; 0\,, \quad i \neq j \tag{27}$$

Independence is a stronger requirement; under independence,

$$\langle g_1(\xi_i) g_2(\xi_j) \rangle - \langle g_1(\xi_i) \rangle \langle g_2(\xi_j) \rangle \;=\; 0\,, \quad i \neq j \tag{28}$$

for *any* functions g_1 and g_2.

Independent Component Analysis (ICA)

Like Principal Component Analysis, except that we're looking for a transformation subject to the stronger requirement of independence, rather than uncorrelatedness.

In general, no analytic solution (like eigenvalue decomposition for PCA) exists, so ICA is implemented using neural network models.

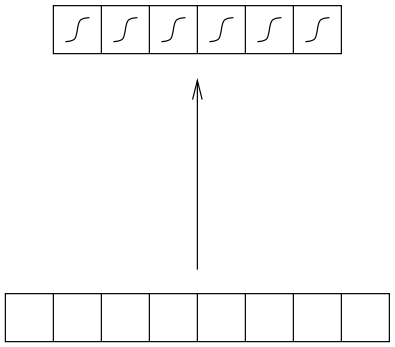
To do this, we need an architecture and an objective function to descend/climb in.

Leads to N independent (or as independent as possible) components in N-dimensional space; they need not be orthogonal.

When are independent components identical to uncorrelated (principal) components? When the generative distribution is uniquely determined by its first and second moments. This is true of only the Gaussian distribution.

Neural Network for ICA

Single layer network:



Patterns $\{\bar{\xi}\}$ are fed into the input layer.

Inputs multiplied by weights in matrix $\mathbf{W}$.

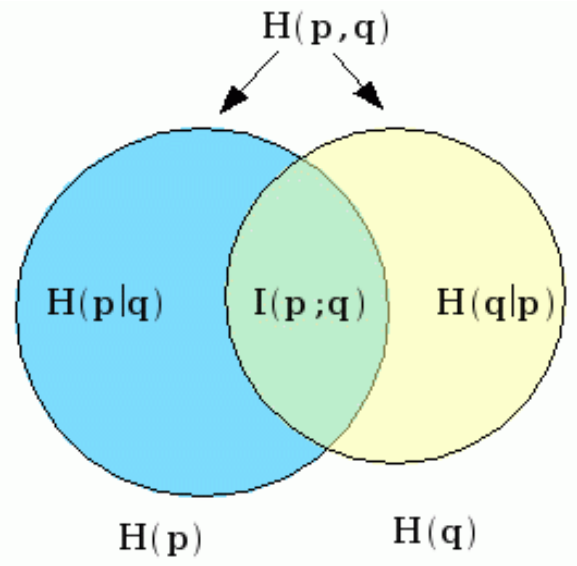
Output logistic (vector notation here):

$$\bar{y} = \frac{1}{1 + e^{\mathbf{W}^T \bar{\xi}}} \tag{29}$$

Objective Function for ICA

Want to ensure that the outputs y_i are maximally independent.

This is identical to requiring that the mutual information be small. Or alternately that the joint entropy be large.



$$
\begin{aligned}
H(p) &= \text{entropy of distribution } p \text{ of first neuron's output} \\
H(p|q) &= \text{conditional entropy} \\
I(p;q) &= H(p) - H(q|p) \\
&= H(q) - H(p|q) \\
&= \text{mutual information}
\end{aligned}
$$

Gradient ascent in this objective function is called *infomax* (we're trying to maximize the enclosed area representing information quantities).

Blind Source Separation (BSS)

The most famous application of ICA.

Have K sources $\{s_k[t]\}$, and K signals $\{x_k[t]\}$. Both $\{s_k[t]\}$ and $\{x_k[t]\}$ are time series (t is a discrete time index).

Each signal is a linear mixture of the sources

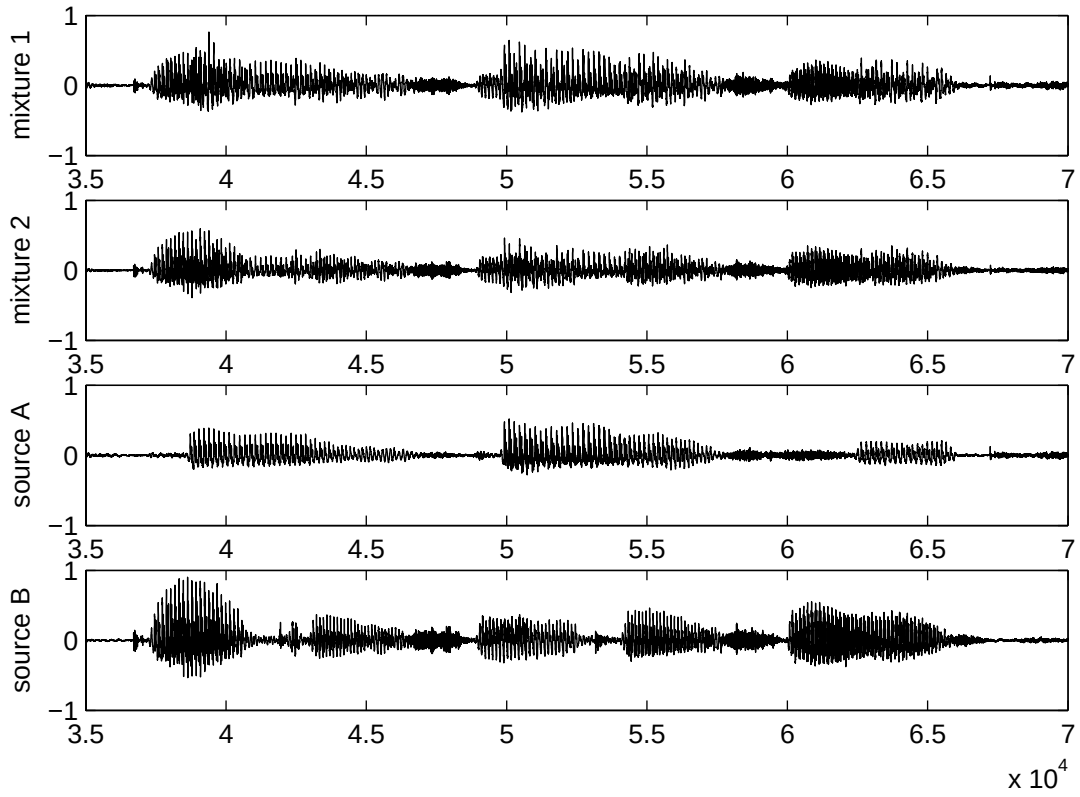
$$x_k[t] \;=\; \mathbf{A}s_k[t] + n_k[t] \tag{30}$$

where $n_k[t]$ is the noise contribution in the kth signal $x_k[t]$, and $\mathbf{A}$ is a mixture matrix.

The problem: given $x_k[n]$, determine $\mathbf{A}$ and $s_k[n]$.

The Cocktail Party

Want to separate individual voices from a cocktail party. Here's a 2-speaker equivalent:



ica1,ica2,ica3

In `matlab/pca`.

Also go to `http://web.media.mit.edu/~paris/ica.html`. This page has links to most of the people around the world working on ICA and BSS.