Visual feature learning

Winter 2013

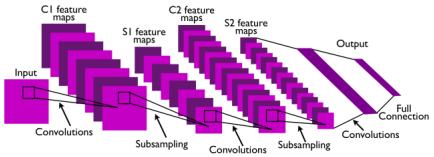
Roland Memisevic

Lecture 7, February 12, 2013

Last time

- Visual processing
- Receptive fields
- Complex cells and the energy mechanism
- Retinotopy, topography

Convolutional networks



- A variety of models are inspired by the presence of simple and complex cells in the brain. Eg.
  - Neocognitron (Fukushima, 1980)
  - Convolutional networks (LeCun et. al, 1998)
  - HMAX (Riesenhuber & Poggio, 1999)
- But “complex cells” in these models are not defined as energy models (they use pooling without squaring).

Why Hebbian learning can implement PCA

- Hebbian learning (linear neurons):

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \epsilon(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})\boldsymbol{x}$$

- Average weight changes:

$$\begin{aligned}\langle \Delta \boldsymbol{w} \rangle &= \langle \epsilon(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})\boldsymbol{x} \rangle \\ &= \langle \epsilon \boldsymbol{x}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} \rangle \\ &= C\boldsymbol{w}\end{aligned}$$

  where C is the covariance matrix of $\boldsymbol{x}$.
- Weight dynamics as differential equation:

$$\frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} = \eta C \boldsymbol{w}$$

- Solution:

$$\boldsymbol{w}(t) = e^{\eta C t}\boldsymbol{w}(0)$$

Why Hebbian learning can implement PCA

- Write $\boldsymbol{w}(t)$ in terms of the eigenvectors of C:

$$\begin{aligned}\boldsymbol{w}(t) = \sum_k a_k(t)\boldsymbol{v}_k &= e^{\eta C t}\boldsymbol{w}(0) \\ &= e^{\eta C t}\sum_k a_k(0)\boldsymbol{v}_k \\ &= \sum_k a_k(0)e^{\eta C t}\boldsymbol{v}_k \\ &= \sum_k a_k(0)e^{\eta \lambda_k t}\boldsymbol{v}_k\end{aligned}$$

Why Hebbian learning can implement PCA

- Assume $\lambda_1 > \ldots > \lambda_D$
- Then we have

$$\begin{aligned}\boldsymbol{w}(t) &= \sum_k a_k(t)e^{\eta \lambda_k t}\boldsymbol{v}_k \\ &= e^{\eta \lambda_1 t}\big[a_1(0)\boldsymbol{v}_1 + \sum_{k=2}^{D} a_k(0)e^{-\eta t(\lambda_1-\lambda_k)}\boldsymbol{v}_k\big] \\ &\approx a_1(0)e^{\eta \lambda_1 t}\boldsymbol{v}_1\end{aligned}$$

- So

$$\lim_{t\to\infty}\frac{\boldsymbol{w}(t)}{\|w(t)\|} = \pm\boldsymbol{v}_1$$

Why Hebbian learning can implement PCA

- "**But $\|\boldsymbol{w}\|$ will grow**."
  Solution: Renormalize after every step (Oja's rule).
- "**What about the other eigenvectors $\boldsymbol{v}_2, \ldots, \boldsymbol{v}_D$?**"
  Solution: Orthogonalize after each step (Sanger's rule).
- With some small modifications (for example, non-linear output nonlinearity) Hebbian learning can implement sparse coding, too.

Probabilistic PCA

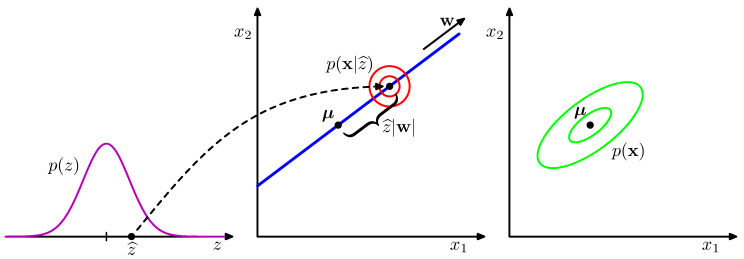
- One can define PCA also as a *probabilistic latent variable model* (Tipping, Bishop; 1999).
- Assume a Gaussian prior distribution over latent variables

$$p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}|\mathbf{0}, \mathbf{I})$$

and a Gaussian conditional distribution over images $\boldsymbol{x}$

$$p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{x}|W\boldsymbol{z} + \boldsymbol{\mu}, \sigma^2\mathbf{I})$$

Probabilistic PCA



- This defines the following generative model: First draw from a low-dimensional Gaussian in the latent space, and then draw the observation from a *D*-dimensional conditional Gaussian whose mean depends on the latent variable.

Probabilistic PCA

- The backward mapping follows from Bayes' rule:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{\int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}}$$

- Plugging in the Gaussian distributions yields

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\mathbf{M}^{-1}\mathbf{W}^{\mathrm{T}}(\mathbf{x}-\boldsymbol{\mu}), \sigma^{-2}\mathbf{M})$$

  with $\mathbf{M} = \mathbf{W}^{\mathrm{T}}\mathbf{W} + \sigma^2\mathbf{I}$

- The marginals turn out to be Gaussian, too:

$$p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\ \mathrm{d}\mathbf{z} = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \mathbf{W}\mathbf{W}^{\mathrm{T}} + \sigma^2\mathbf{I})$$

- This is simply a Gaussian, whose covariance matrix is the outer product of two low-rank matrices (plus noise).

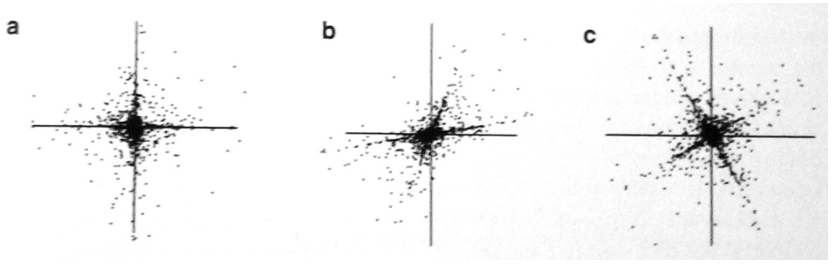
Gaussians and independence

- Spherical Gaussians have independent marginals:

$$\begin{aligned} p(x_1, \ldots, x_n) &= \frac{1}{(2\pi)^{\frac{n}{2}}} \exp\left(-\frac{1}{2}\|\mathbf{x}\|^2\right) \\ &= \prod_i \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x_i^2\right) \end{aligned}$$

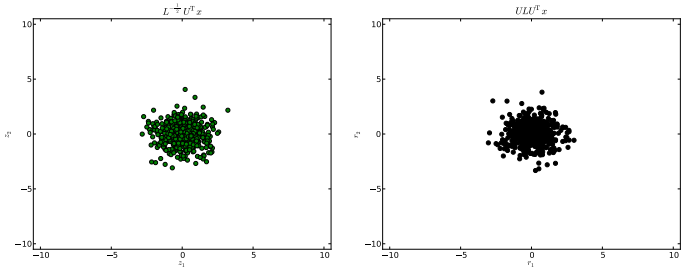
- Implications:
  1. For Gaussian distributed data, whitening amounts to extracting the independent components.
  2. For Gaussian distributed data, uncorrelatedness implies independence.
- For non-Gaussian distributions these are *not* true.

A counter example



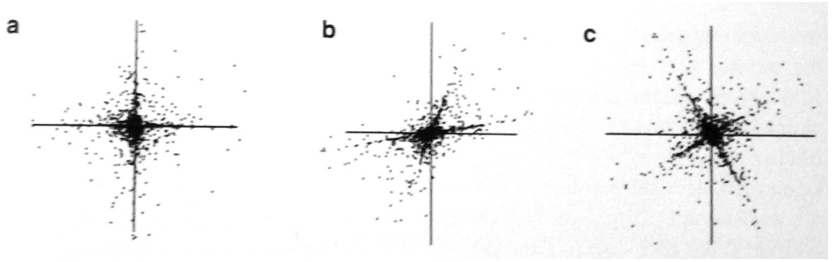
- a: Independent variables.
- b: A linear combination: not independent.
- c: Linear combination after whitening: not independent.

Uncorrelatedness is not independence



- Recall that any orthogonal transformation of white data is white.
- PCA and ZCA are two of the infinitely many example whitening matrices.
- How can we find the one that maximizes independence?

Finding independent components is whitening



- Independence does imply uncorrelatedness.
- So the linear transformation that makes data independent (if it exists) must still be a whitening matrix!

Independent components analysis

- Generative model: We have independent "source" variables s_i, which get mixed to yield the observed data:

$$\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{s}$$

  where $p(s_1, \ldots, s_n) = \prod_i p_i(s_i)$.
- The analysis equation is:

$$\boldsymbol{s} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{x}$$

- For images it can be convenient to write this as

$$I(x, y) = \sum_{i=1}^{n} A_i(x, y) s_i$$

$$s_i = \sum_{x,y} W_i(x, y) I(x, y)$$

- Learning: Find A and W as well as all the $\boldsymbol{s}$.

Independent components analysis

- Instead of the original images, I, we typically use the whitened components, $\boldsymbol{z}$, (from PCA or ZCA) as the input data.
- Multiplying any component s_i by some scalar will have no effect if we divide the corresponding A_i by the same number.
- One way to fix this is by constraining A.
- In general, we also have to assume that $A = W^{-1}$.
- A and W need to be square matrices. We will relax that assumption in the future.

Implication for feature learning

- Most popular feature learning models (auto-encoder networks, k-means, restricted Boltzmann machines, etc.) utilize "tied weights".
- This means that the encoder weights are the transpose of the decoder weights:

$$A = W^{\mathrm{T}}$$

- But this means that $W^{\mathrm{T}}W = \boldsymbol{I}$, so W is orthogonal.
- So these models are likely to work well only on whitened data!?

Maximum likelihood ICA

- Since we assume independence of the s_i, we can write the pdf of the observations as

$$p(\boldsymbol{x}) = \left|\det W\right| \prod_{i=1}^{n} p_i(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x})$$

- For IID observations, we get the following likelihood:

$$L(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n) = \prod_{t=1}^{T} p(\boldsymbol{x}_t) = \prod_{t=1}^{T} \left[\left|\det W\right| \prod_{i=1}^{n} p_i(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}_t) \right]$$

Densities under linear transformation

Under a linear transformation $\boldsymbol{y} = W\boldsymbol{x}$, the density $p_x(\boldsymbol{x})$ turns into

$$\left|\det W\right| p_x(W\boldsymbol{x})$$

Maximum likelihood ICA

- The log-likelihood is

$$\log L(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n) = T \log \left|\det W\right| + \sum_{i=1}^{n} \sum_{t=1}^{T} \log p_i(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}_t)$$

- If we apply this to pre-whitened data $\boldsymbol{z}$, then W must be orthonormal, so

$$\left|\det W\right| = 1$$

- In this case we may maximize:

$$\sum_{i=1}^{n} \sum_{t=1}^{T} \log p_i(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{z}_t)$$

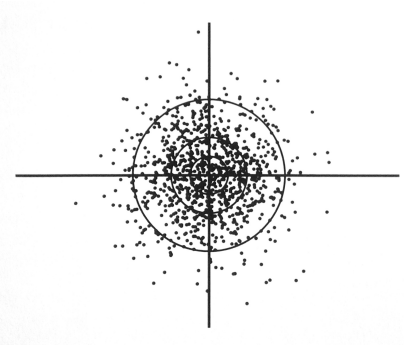
Maximum likelihood ICA

- We can rewrite the constrained optimization problem as

$$\begin{aligned} \text{Minimize} \quad & \sum_t \sum_i \phi(\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{z}_t) \\ \text{s.t.} \quad & W^{\mathrm{T}}W = I \end{aligned}$$

  where $\phi() = -\log p_i()$ is some negative, non-Gaussian log-pdf.
- We just need to choose appropriate source densities...

Maximum likelihood for Gaussian sources

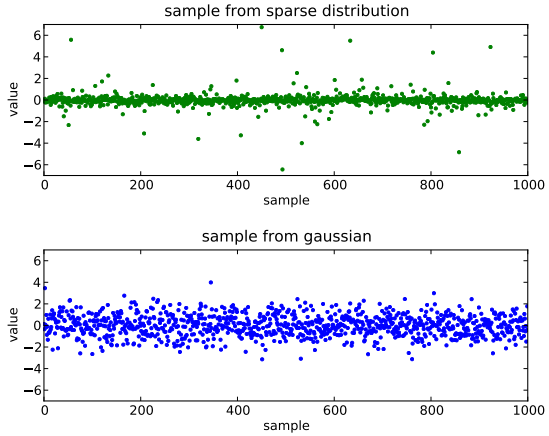


- The independent Gaussian is spherically symmetric.
- So rotation (orthogonal W) won't change the objective.
- For any *non*-Gaussian source distribution it will.

Forms non-Gaussianity

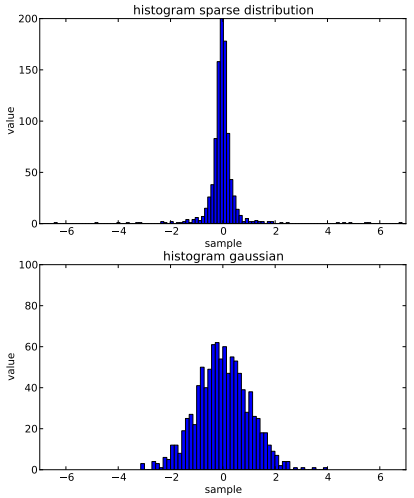
- To get beyond uncorrelatedness, we have to choose non-Gaussian p_i.
- Three forms of non-Gaussianity are
  1. Super-Gaussian (*sparsity*): Distribution is peaked at zero (positive kurtosis)
  2. Sub-Gaussian: Distribution is "flat" at zero (negative kurtosis)
  3. Skew: Distribution is unsymmetric.
- Of these, super-Gaussianity is generally assumed to be the best match for (most) image features.

Sparseness



- Top: Samples from a student-T-distribution (sparse)
- Bottom: Samples from a normal distribution of the same variance (not sparse).

Histograms



Sparse does not mean “small values”

- Sparsity can easily be mixed up with “small”.
- A normal distribution may be scaled to take on small values, too. This doesn’t make it sparse.
- Sparsity must be measured *relative to some standard deviation*.

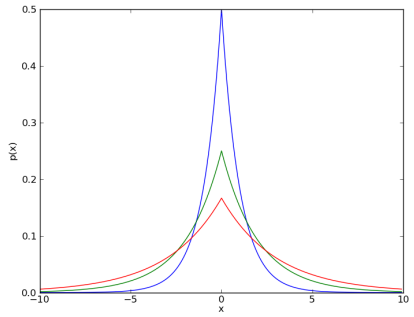
Gaussian scale mixtures

- One explanation for super-Gaussianity in natural images is that any one feature may occur at different (brightness-)scales.
- We can model an image patch using a Gaussian g_i whose value is modulated by some independent scale-variable d_i:

$$s_i = g_i d_i$$

- This yields a super-Gaussian distribution, because $p(s_i)$ will be a superposition of Gaussians each with different variance.
- In fact, contrast normalization seems to reduce sparsity (a bit).

Sparse source densities



- A popular choice for the sparse source density is the (zero-mean) Laplacian:

$$p(s_i) = \frac{1}{2b} \exp\left(-\frac{|s_i|}{b}\right)$$

- A differentiable alternative is the log cosh function.

Sparse coding

- With this source density, the optimization problem turns into

$$\begin{aligned} \text{Minimize} \quad & \sum_t \sum_i |\boldsymbol{w}_i^{\mathrm{T}} \boldsymbol{z}_t| \\ \text{s.t.} \quad & W^{\mathrm{T}} W = I \end{aligned}$$

- This can be solved by alternating gradient steps and projections that enforce the constraint.

Reconstruction error

- An alternative to solving a constrained optimization problem is to enforce the constraint

$$W^{\mathrm{T}} W = I \Leftrightarrow W^{-1} = W^{\mathrm{T}}$$

implicitly.

- By adding a reconstruction term we can encourage this as follows:

$$\text{Minimize} \quad \sum_t \| W W^{\mathrm{T}} \boldsymbol{z}_t - \boldsymbol{z}_t \|^2 + \sum_t \sum_i |\boldsymbol{w}_i^{\mathrm{T}} \boldsymbol{z}_t|$$

- We may even separate encoder and decoder weights:

$$\text{Minimize} \quad \sum_t \| A W^{\mathrm{T}} \boldsymbol{z}_t - \boldsymbol{z}_t \|^2 + \sum_t \sum_i |\boldsymbol{w}_i^{\mathrm{T}} \boldsymbol{z}_t|$$

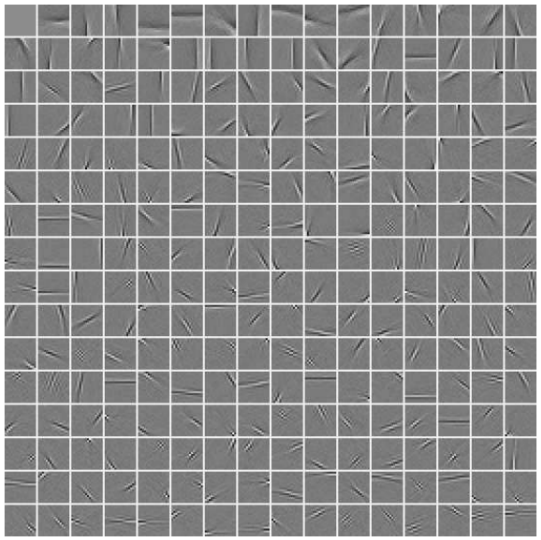
Reconstruction error

- In the literature one frequently finds

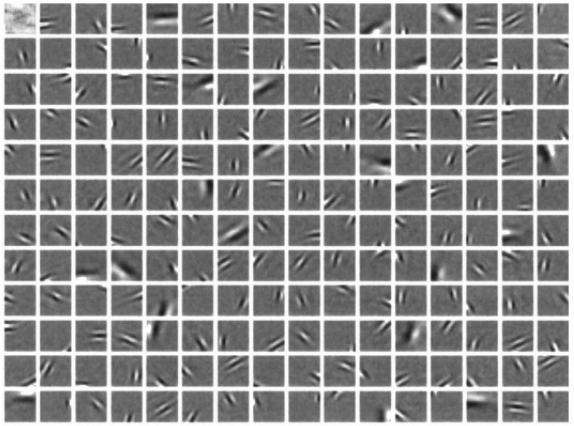
$$\text{Minimize} \quad \sum_t \| A \boldsymbol{s}_t - \boldsymbol{z}_t \|^2 + \sum_t \sum_i |s_i|$$

where the optimization is over *both* A and $\boldsymbol{s}$.

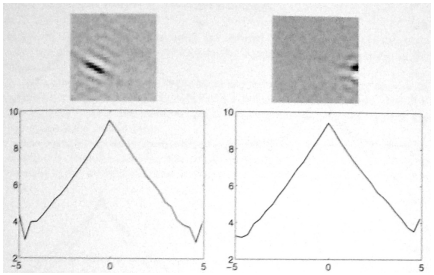
ICA filters



Sparse coding components (Olshausen/Field)



Estimating the source densities



- One can estimate the source densities from data as well.
- It turns out that not all components are sparse.
- The DC component, for example, tends to be sub-Gaussian.
- So to keep it or not can make a difference in practice!

Relation between analysis and synthesis weights

- If the s_i are independent and have unit variance, it holds (exercise):

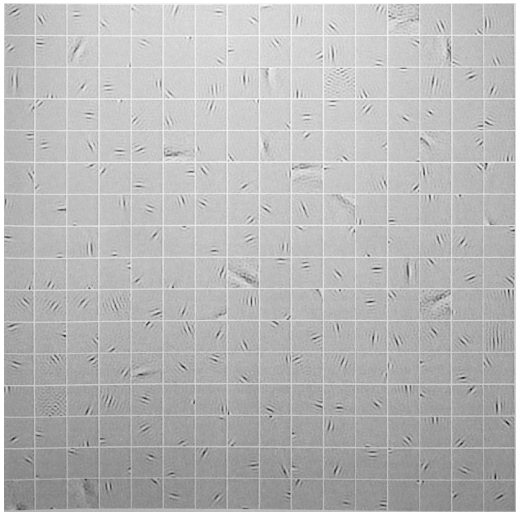
$$\mathrm{cov}\big((x,y),(x',y')\big) = \sum_i A_i(x,y)A_i(x',y')$$

- Therefore:

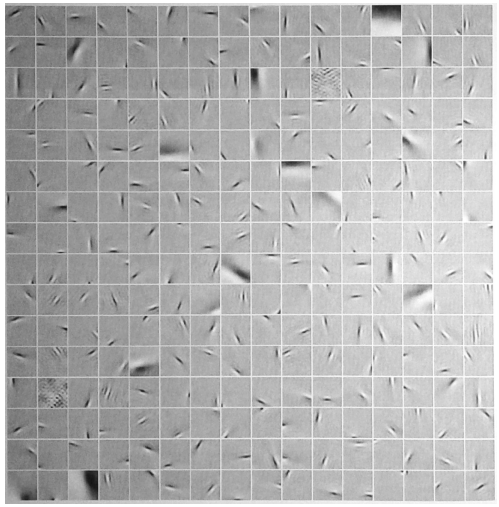
$$\begin{aligned}
&\sum_{x',y'} \mathrm{cov}\big((x,y),(x',y')\big)\, W_i(x',y') \\
= &\sum_{x',y'} \sum_j A_j(x,y)A_j(x',y')W_i(x',y') \\
= &\sum_j A_j(x,y) \sum_{x',y'} A_j(x',y')W_i(x',y') = A_i(x,y)
\end{aligned}$$

- Multiplying by the covariance matrix is lowpass-filtering:

Example analysis filters



Example synthesis filters



Information theoretic interpretation

- One can use the *mutual information* to measure independence of the s_i:

$$\begin{aligned} I(s_1, \ldots, s_n) &= \int_{s_1,\ldots,s_n} p(s_1, \ldots, s_n) \log \frac{p(s_1, \ldots, s_n)}{p(s_1) \cdots p(s_n)} \\ &= \sum_{i=1}^{n} H(s_i) - H(\boldsymbol{s}) \\ &= \sum_{i=1}^{n} H(\boldsymbol{w}_i^{\mathrm{T}} \boldsymbol{z}) - H(W\boldsymbol{z}) \end{aligned}$$

- For orthogonal W, the last term is constant.
- To minimize MI, minimize the entropy of individual components!
- To minimize their entropy, make them less Gaussian!