Linear Factor Models

Vahid Tarokh

ECE 685D, Fall 2025

Linear Factor Models

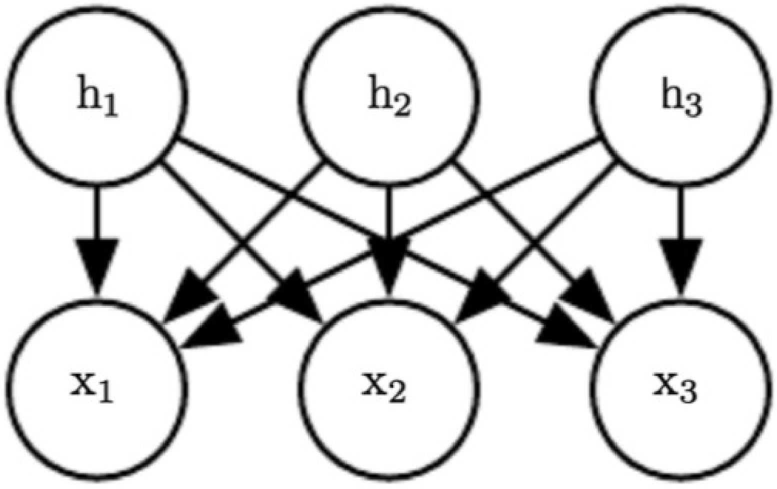
- As discussed before, many probabilistic models have latent variables.
- **Latent variables**, as opposed to **observable variables**, are variables that are not observed but instead inferred from observed variables
- **Linear factor models** are some of the simplest probabilistic models with latent variables.
- We may not implement any linear factor models to solve state-of-art problems, but LFMs provide a good building block for mixture models or deeper probabilistic models.
- Many of the approaches we discuss today are necessary to build generative models that more advanced deep models will expand upon

Linear Factor Models

- LFMs postulates that the data generation process is as follows:
  - ➢ Sample the explanatory factors h from a distribution p(h).
  - ➢ Sample the real-valued observable variables given the factors:

$$x = Wh + b + n$$

- Components of h are typically assumed to be i.i.d. and the noise n is assumed to be zero-mean Gaussians with independent (not necessarily i.i.d.) components.



$\mathbf{x} = \boldsymbol{W}\mathbf{h} + \boldsymbol{b} + \text{noise}$

Linear Factor Models

- In this setting W is a $q \times d$ matrix and h is d dimensional.
- The motivation is that if $q >> d$ then the latent variables h give a more parsimonious explanation of dependencies between components of the observations x.
- This must be reminiscent of PCA for the students in this course.
- In fact as we will see, LFM includes a many well-known techniques as special cases.

Factor Analysis

- Assume that the components of h are i.i.d. standard Gaussian. Then W colors x and b adds bias.
- Assume that the i-th noise component has variance σ_i^2.
- Then show that x is a multivariate normal with mean zero, mean b and variance

$$WW^T + \text{diag}(\sigma_1^2, \sigma_2^2, \ldots.., \sigma_q^2).$$

- W and $\sigma_1^2, \sigma_2^2, \ldots.., \sigma_q^2$ can be estimated using the MLE methods. Since no closed form exists, then iterative methods are commonly applied.
- Key issue is that given h components of x are ***independent*** but ***do not necessarily have the same variance.***
  - ➢ This is the key difference with classical PCA.
  - ➢ The subspace generated by the MLE estimate of W will not necessarily correspond to the principal subspace of the data.

Linear Factor Models

- Different types of LFMs make different choices about the form of the noise and of the prior p(h)
- The most important LFMs that we will discuss are:
  - PCA and Probabilistic PCA
  - Independent component analysis (ICA)
  - Slow feature analysis
  - Sparse coding

Probabilistic PCA

- Principal Component Analysis (PCA) is a old technique that is very well-established.
- Tipping and Bishop (1999) build on the work of Young (1940) and Whittle (1952) to propose a probabilistic version of PCA from LFM and factor analysis.
- Key assumption $\sigma_1^2 = \sigma_2^2 = \ldots\ldots = \sigma_q^2 = \sigma^2$
- In this case, the MLEs in factor analysis have a closed form, and model parameter estimation can be performed iteratively and efficiently. Define:

$$C = WW^T + \sigma^2 I .$$

C: the model covariance under PPCA. This is the covariance predicted by the PPCA model

$$S = [\sum_{i=1}^{N} (x_i - b)(x_i - b)^T]/N$$

S: the sample covariance.
This is exactly the empirical covariance of your dataset after subtracting the mean b

Then PPCA finds parameters: W(ML), σ^2(ML), b(ML) such that: C≈S. This is Maximum Likelihood estimation under PPCA.

Probabilistic PCA

- MLE for b is the mean of data, and for W is given by

$$W_{ML} = U_q\left(\Lambda_q - \sigma^2 I_d\right)^{1/2} R,$$

- With the q column vectors in the $q \times d$ matrix U_q denoting the principle eigenvectors of S with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$ in the $d \times d$ diagonal matrix Λ_d and R is an arbitrary $d \times d$ orthogonal matrix:

$$\sigma_{ML}^2 = \frac{1}{q-d} \sum_{j=d+1}^{q} \lambda_j.$$

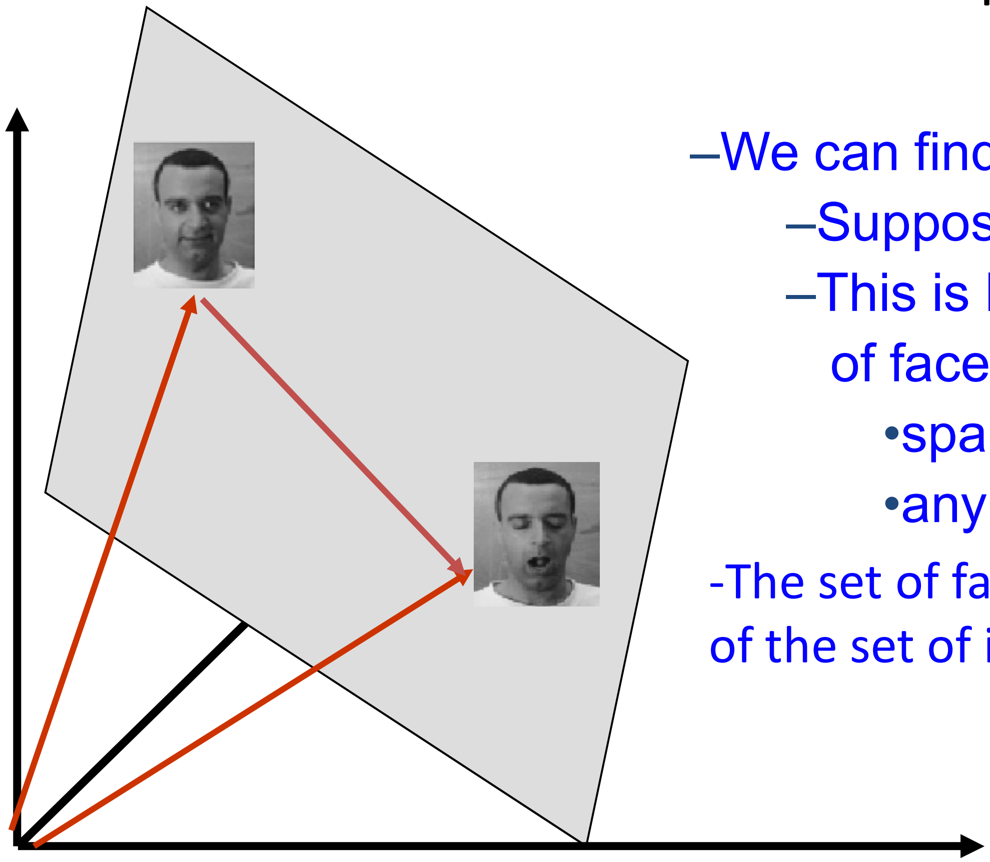
- Question: why does the existence of R makes sense in the above.

Probabilistic PCA

- Interestingly as $\sigma^2 \to 0$ we recover the classical PCA.
- This means that both PPCA and PCA are special cases of LFMs.

Application of PCA

- **Face Recognition**
- An image is a point in a high-dimensional space
  - An $N\ x\ M$ image is a point in R^{NM}
  - We can define vectors in this space



–We can find the best subspace using PCA

–Suppose this is a K-dimensional subspace

–This is like fitting a "hyper-plane" to the set of faces

•spanned by vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_K$

•any face $\mathbf{x} \approx a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + , \ldots, + a_K\mathbf{v}_K$

-The set of faces is a "subspace" of the set of images.

Application of PCA

- Let $F_1, F_2, \ldots, F_M$ be a set of training face images.
- Let F be their mean and $\Phi_i = F_i$ – F
- Use principal components to compute the eigenvectors and eigenvalues of the covariance matrix of the $\Phi_i's$
- Choose the vector u of most significant M eigenvectors to use as the basis.
- Each face is represented as a linear combination of eigenfaces
- $u = (u_1, u_2, u_3, u_4, u_5);$
- $F_{27} = a_1 * u_1 + a_2 * u_2 + \ldots + a_5 * u_5$

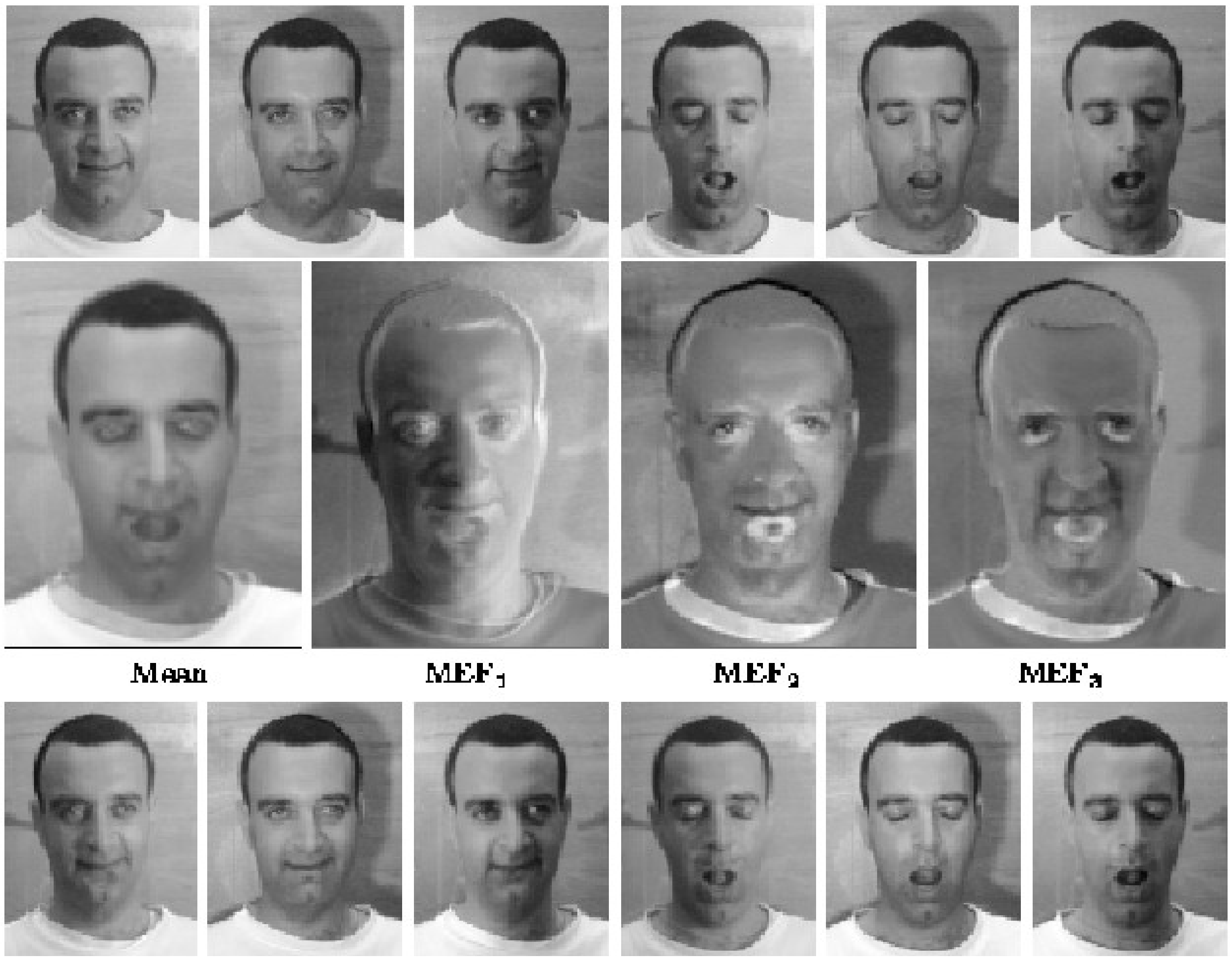
$$\begin{tikzcd}
\text{unknown face image} \arrow[r] & \boxed{\text{convert to its eigenface representation}} \arrow[r] & \Omega = (\Omega_1, \Omega_2, \ldots, \Omega_m)
\end{tikzcd}$$

Find the face class k that minimizes

$$\varepsilon_k = ||\Omega - \Omega_k||$$

Application of PCA

training images



mean image

3 eigen-images

linear approxi-mations

ICA

- Independent component analysis can also be cast as a special case of LFMs.
- In our setting, we assume $b = 0$ (this can be achieved by translation) and $n = 0$ (for simplicity):

$$\mathrm{x} = Ah.$$

  - $\mathrm{x} \in \mathbb{R}^n$ is called mixture vector.
  - $\mathrm{h} \in \mathbb{R}^n$ is called hidden, factor, or latent vector.
  - $\mathrm{A} \in \mathbb{R}^{n \times n}$ is called mixing matrix. It can also be generalized to rectangular matrix.
  - This can be thought of mixing sources given by components of h using mixing matrix A.
  - Both A and h are **unknown** (blind source (signal) separation)

ICA

- Here, we consider the **linear ICA**. The nonlinear version is related to the Autoencoders.
- **Assumptions of ICA:**
  - Each component of h is assumed to be statistically **independent**.
  - Independent components must have **non-Gaussian** distribution.
- **Ambiguities of ICA:**
  - Estimating the latent components up to a scaling factor

    $$x = (\alpha A)(\frac{1}{\alpha} h) \quad \text{for some } \alpha > 0.$$

    - Can force $\mathrm{E}\left(h_i^2\right) = 1$; however, there is still an ambiguity for the sign of hidden components.
    - Fortunately, this is insignificant in most applications.
  - Estimating the components up to the order of them

    $$x = AP^{-1}Ph \quad \text{where } P \text{ is a permutation matrix.}$$

ICA

- **Why non-Gaussian assumption?**
  - ➢ If h is jointly Gaussian, vector Rh is also Gaussian.
  - ➢ The mixing matrix can be any matrix given by AR^{-1} (A is unidentifiable) .

- ICA finds the independent components by maximizing (in some measure) distances of the estimated components from the Gaussian distribution.
  - ➢ Given **observed random vectors,** $\mathbf{x_i} \in \mathbb{R}^{\boldsymbol{n}}$, for $i = 1,2, \dots m$, the goal is to find an unmixing matrix W and laten vector h.
  - ➢ Once W is estimated, the latent components of a given test mixture vector, x^* is computed by $h^* = W^{-1}\,\mathrm{x}^*$.
  - ➢ **How to estimate $\boldsymbol{W}$ matrix:**
    - ▪ W is estimated such that $W^{-1}\,\mathrm{x}$ is far from Gaussian.

ICA

- **Measure of non-Gaussianity:**
  - The absolute or squared kurtosis (the fourth-order cumulant).
  - For a R.V. Y the kurtosis is defined as follows:

$$\boldsymbol{kurt(Y) = E\left(Y^4\right) - 3\left(E(Y^2)\right)^2}$$

  - If Y is a Gaussian r.v., then kurt(Y) $= 0$.
  - For most of non-Gaussian r.v's, the kutosis is non-zero.

- **Methods of estimating the independent components:**
  - Maximizing Neg-entropy
  - Maximization of non-Gaussianity (Using Kurtosis)

- Exercise: Refresh your knowledge of both PCA and ICA.

Application of ICA
Blind source separation

Consider 3 sources, defined as follows:

- Source 1: $S_1 = Sin(2t)$
- Source 2: $S_2 = Squarewave(3t)$
- Source 3: $S_3 = Sawtooth(2\pi t)$

Also, assume the mixing matrix is given by $W = \begin{bmatrix} 1 & 1 & 1 \\ 0.5 & 2 & 1 \\ 1.5 & 1 & 2 \end{bmatrix}$; as a result, observation is $X = WS^T$.

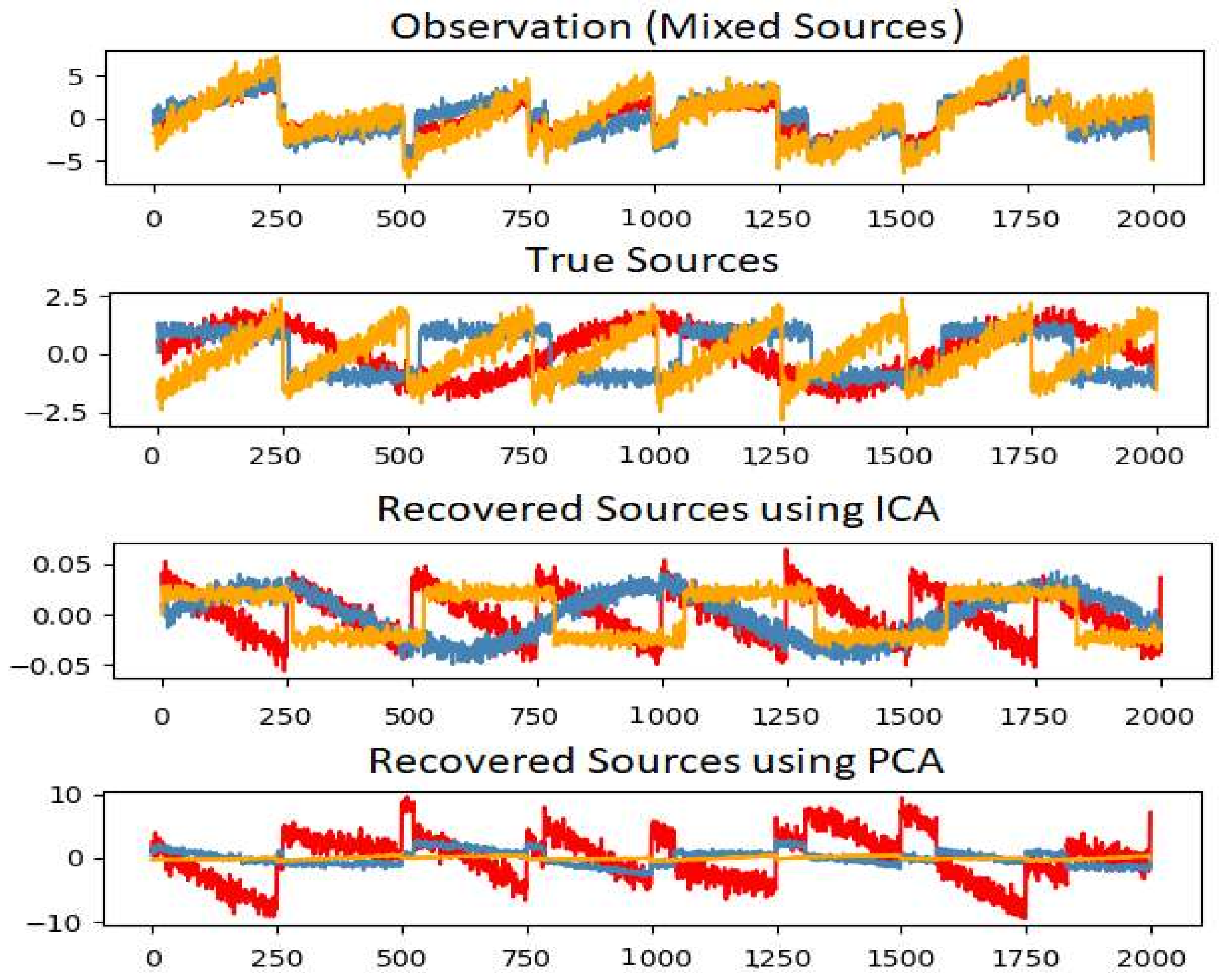
Now, our goal is to separate the three sources from X using ICA.

```
•   n_samples = 2000                              # Number of Samples
•   time = np.linspace(0, 8, n_samples)           # Time range

•   s1 = np.sin(2 * time)                         # Source 1: sinusoidal signal
•   s2 = np.sign(np.sin(3 * time))                # Source 2: square signal
•   s3 = signal.sawtooth(2 * np.pi * time)        # Source 3: saw tooth signal

•   S = np.c_[s1, s2, s3]
•   S += 0.2 * np.random.normal(size=S.shape)  # Add noise

•   S /= S.std(axis=0)                                          # Standardizing data
•   W = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])   # Mixing matrix, W_3X3
•   X = np.dot(W, S.T)                                          # Observations (Mixed Sources)
•   X =X.T


•   ica = FastICA(n_components=3)                               # Compute ICA (Using FastICA [1] algorithm)

•   Ŝ = ica.fit_transform(X)                                    # Recovered Sources
•   Ŵ  = ica.mixing_                                            # Estimated mixing matrix
```

[1]. A. Hyvärinen, "Fast and Robust Fixed-Point Algorithms for Independent Component Analysis", IEEE Transactions on Neural Networks 10(3):626-634, 1999

Application of ICA

- **Blind source separation**



https://scikit-learn.org/stable/auto_examples/decomposition/plot_ica_blind_source_separation.html#sphx-glr-auto-examples-decomposition-plot-ica-blind-source-separation-py

Slow Feature Analysis

- Slow feature analysis is motivated by a general principle called the **slowness principle**.
  - The idea is that the important characteristics of scenes change very slowly compared to the individual measurements that make up a description of a scene.
  - For example, in computer vision, individual pixel values can change very rapidly. If a zebra moves from left to right across the image, an individual pixel will rapidly change from black to white and back again as the zebra's stripes pass over the pixel. By comparison, the feature indicating whether a zebra is in the image will not change at all, and the feature describing the zebra's position will change slowly.
  - We therefore may wish to regularize our model to learn features that change slowly over time.

Slow Feature Analysis

- The slowness principle predates slow feature analysis and has been applied to a wide variety of models.

- In general, we can apply the slowness principle to any differentiable model trained with gradient descent. The slowness principle may be introduced by adding a term to the cost function of the form

$$\lambda \sum_t L(f(\boldsymbol{x}^{(t+1)}), f(\boldsymbol{x}^{(t)})),$$

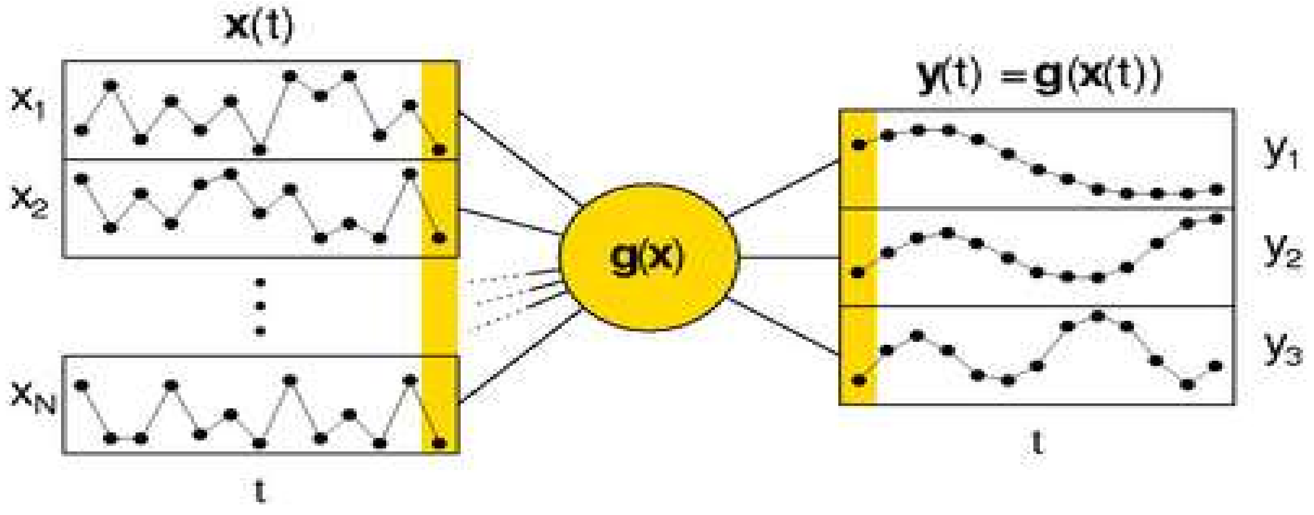
where λ is a hyper-parameter determining the strength of the slowness regularization term, t is the index into a time sequence of examples, f is the feature extractor to be regularized, and L is a loss function.

Slow Feature Analysis

- Given a set of time-varying input signals, **x**(t), SFA learns instantaneous, non-linear functions **g**(**x**) that transform **x** into slowly-varying output signals, **y**(t).

- The optimization procedure guarantees that SFA returns the global optimum for **g** (i.e., the slowest output signal) in a given function space.

- As the transformations must be instantaneous, trivial solution like low-pass filtering are not possible.



Slow Feature Analysis

- If we restrict the nonlinear functions $g_j(\cdot)$ to a finite dimensional function space, such as all polynomials of degree two, we can transform the variational problem into a more conventional optimization over the coefficients of the basis of the function space (e.g., all monomials of degree 1 and 2).

- In this way, the problem becomes simpler to solve, and one can use algebraic methods, which are the basis of the slow feature analysis algorithm.

Slow Feature Analysis

- Consider as a simple example the two-dimensional input signal $x_1(t) = \sin(t) + \cos(11t)^2$ and $x_2(t) = \cos(11t)$.

- Both components are quickly varying but hidden in the signal is the slowly varying $y(t) = x_1(t) - x_2^2(t)$, which can be extracted with a polynomial of degree two, namely $g(\mathbf{x}) = x_1 - x_2^2$.

- **To find the function g(x) that extracts the slow feature from the input signal, one proceeds as follows.**

- **The non-linear problem is transformed to a linear one by expanding the input into the space of nonlinear functions under consideration.**

- **Subsequently we seek for a linear transform** f(x; θ).

Slow Feature Analysis

- SFA algorithm (Wiskott and Sejnowski, 2002) then finds the linear transformation f(x; θ) , then solving the optimization problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_t (f(\boldsymbol{x}^{(t+1)})_i - f(\boldsymbol{x}^{(t)})_i)^2$$

Subject to

$$\mathbb{E}_t f(\boldsymbol{x}^{(t)})_i = 0$$

and

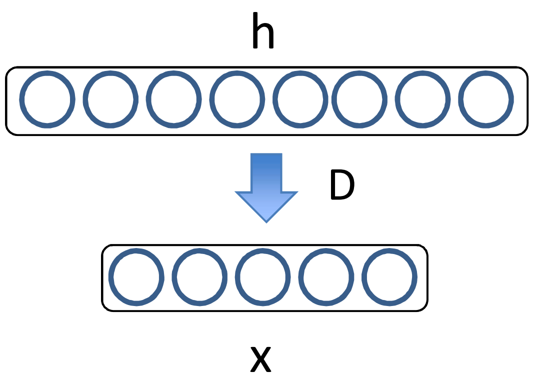
$$\mathbb{E}_t [f(\boldsymbol{x}^{(t)})_i^2] = 1.$$

- Question: Why are all these constraints necessary?
- We will provide examples in HW.

Sparse Coding

- Sparse coding (Olshausen & Field, 1996). Originally developed to explain early visual processing in the brain (edge detection).

- For each input $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - it is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - we can good reconstruct the original input $\mathbf{x}^{(t)}$



Sparse Coding

- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - it is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - we can good reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

Reconstruction: $\widehat{\mathbf{x}}^{(t)}$

Sparsity vs. reconstruction control

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \underbrace{\frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2}_{\text{Reconstruction error}} + \underbrace{\lambda ||\mathbf{h}^{(t)}||_1}_{\text{Sparsity penalty}}$$

Sparse Coding

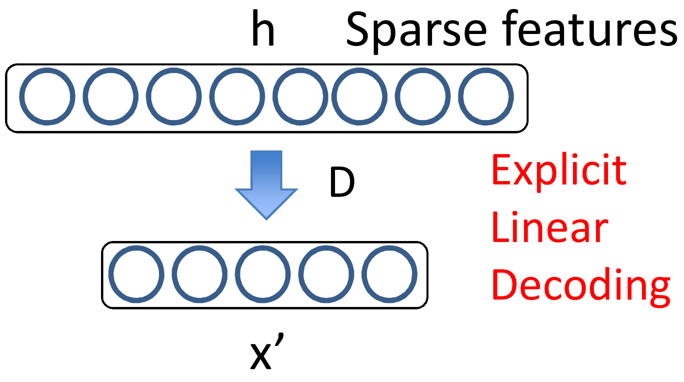
- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - it is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - we can good reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

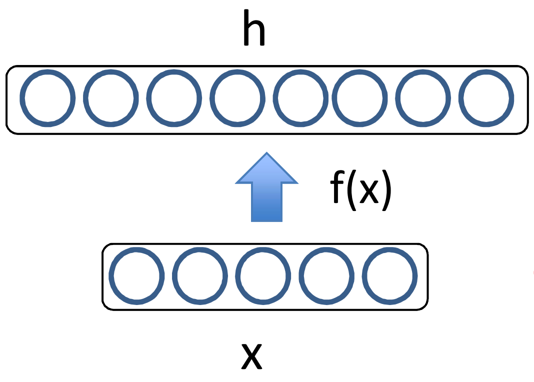
$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\ \mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

Interpreting Sparse Coding

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$



Explicit
Linear
Decoding



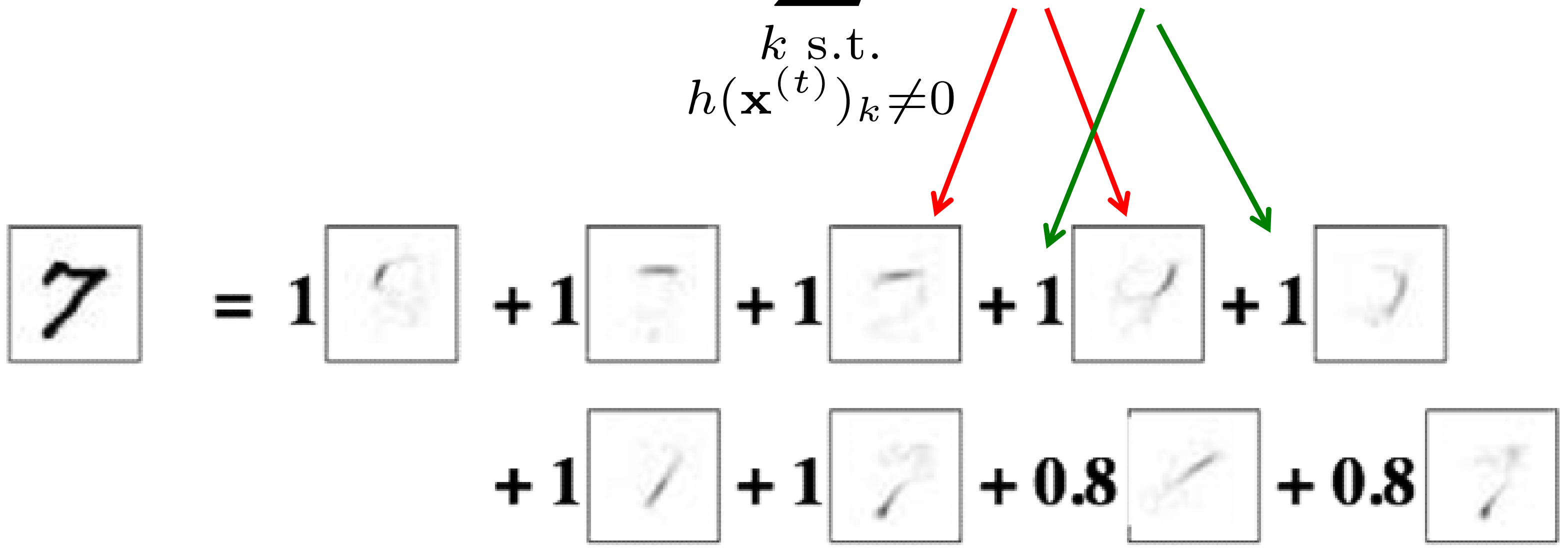
Implicit
nonlinear
encoding

- Sparse, over-complete representation **h.**
- Encoding **h** = f(**x**) is implicit and nonlinear function of **x**.
- Reconstruction (or decoding) **x'** = Dh is linear and explicit.

Sparse Coding

- We can also write:

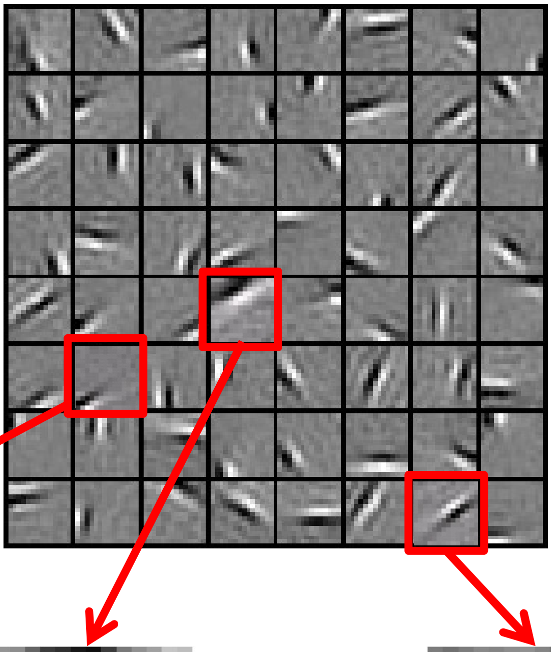
$$\widehat{\mathbf{x}}^{(t)} = \mathbf{D}\,\mathbf{h}(\mathbf{x}^{(t)}) = \sum_{\substack{k \text{ s.t.}\\ h(\mathbf{x}^{(t)})_k \neq 0}} \mathbf{D}_{\cdot,k}\, h(\mathbf{x}^{(t)})_k$$



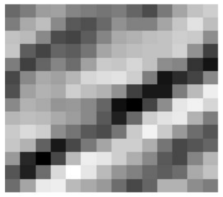
Sparse Coding

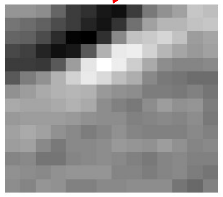
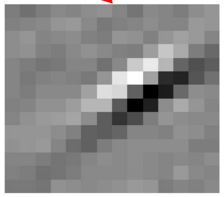
Natural Images

Learned bases: "Edges"



New example


$= 0.8 *$

$+ 0.3 *$

$+ 0.5 *$


$$x = 0.8 * \phi_{36} + 0.3 * \phi_{42} + 0.5 * \phi_{65}$$

[0, 0, … **0.8**, …, **0.3**, …, **0.5**, …] = coefficients (feature representation)

Slide Credit: Honglak Lee

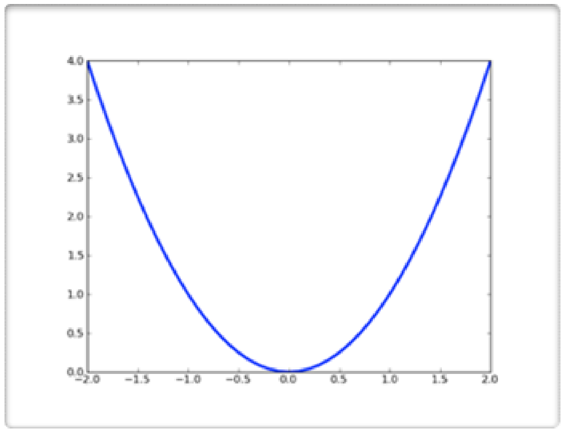
Inference

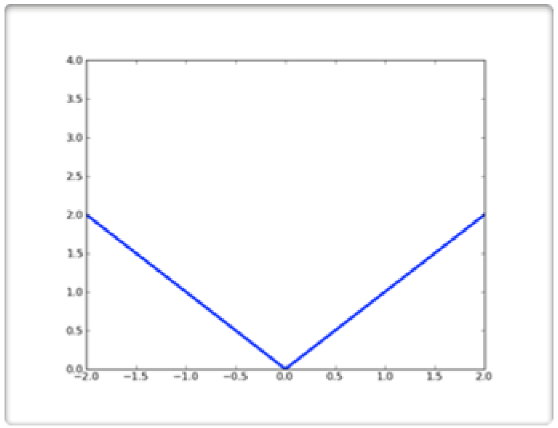
•How do we compute $\mathbf{h}(\mathbf{x}^{(t)})$?

- We need to optimize:

$$l(\mathbf{x}^{(t)}) = \frac{1}{2}||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda||\mathbf{h}^{(t)}||_1$$

- This is Lasso.





- We could use a gradient descent method:

$$\nabla_{\mathbf{h}^{(t)}} l(\mathbf{x}^{(t)}) = \mathbf{D}^{\top}(\mathbf{D}\,\mathbf{h}^{(t)} - \mathbf{x}^{(t)}) + \lambda\,\mathrm{sign}(\mathbf{h}^{(t)})$$

Inference

- For a single hidden unit:

$$\frac{\partial}{\partial h_k^{(t)}} l(\mathbf{x}^{(t)}) = (\mathbf{D}_{\cdot,k})^\top (\mathbf{D}\,\mathbf{h}^{(t)} - \mathbf{x}^{(t)}) + \lambda\,\mathrm{sign}(h_k^{(t)})$$

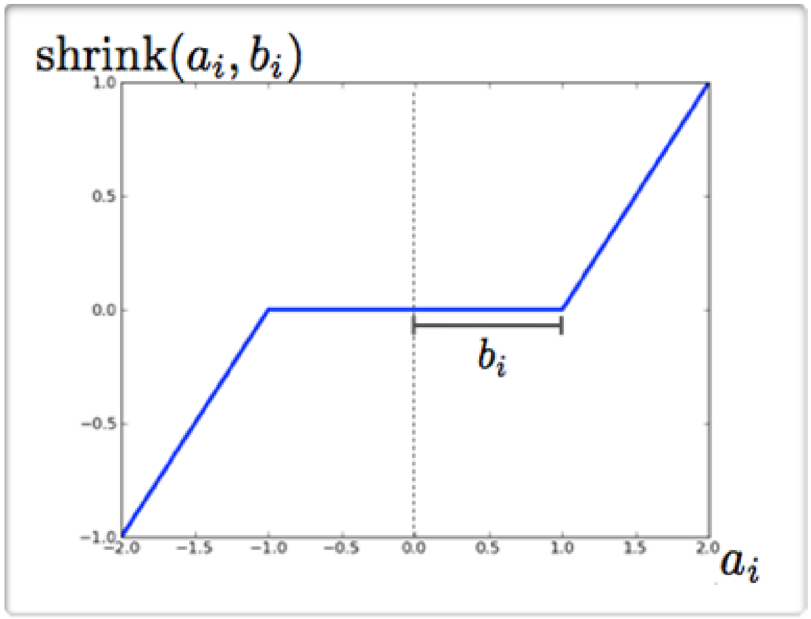
  - issue: L_1 norm not differentiable at 0
  - very unlikely for gradient descent to "land" on $h_k^{(t)} = 0$ (even if it's the solution)

- Solution: if $h_k^{(t)}$ changes sign because of L_1 norm gradient, clamp to 0.

ISTA Algorithm

- This process corresponds to the ISTA (Iterative Shrinkage and Thresholding) Algorithm:

```
➢ Initialize $\mathbf{h}^{(t)}$ (for example to 0)
➢ While $\mathbf{h}^{(t)}$ has not converged
    $\mathbf{h}^{(t)} \Longleftarrow \mathbf{h}^{(t)} - \alpha\, \mathbf{D}^\top (\mathbf{D}\,\mathbf{h}^{(t)} - \mathbf{x}^{(t)})$
    $\mathbf{h}^{(t)} \Longleftarrow \mathrm{shrink}(\mathbf{h}^{(t)}, \alpha\, \lambda)$
```



where

$$\mathrm{shrink}(\mathbf{a}, \mathbf{b}) = [\ldots, \mathrm{sign}(a_i)\ \max(|a_i| - b_i, 0), \ldots]$$

- Will converge if $\dfrac{1}{\alpha}$ is bigger than the largest eigenvalue of $\mathbf{D}^\top\mathbf{D}$