Lecture 14
Introduction to Deep Unsupervised Learning

CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

May 15, 2017

Unsupervised Learning

- So far we have only looked at *discriminative* models i.e. we model $Y = f(X; \theta)$ or $P(Y|X)$

Unsupervised Learning

- So far we have only looked at *discriminative* models i.e. we model $Y = f(X; \theta)$ or $P(Y|X)$
- Recall: $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$

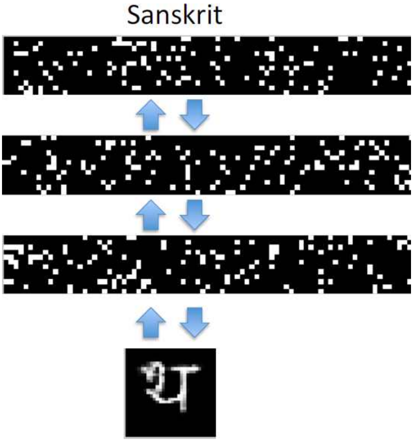
Unsupervised Learning

- So far we have only looked at *discriminative* models i.e. we model $Y = f(X;\theta)$ or $P(Y|X)$
- Recall: $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$
- $P(X)$ is defined in terms of $P(X|Y)$ or the best model of X (unsupervised learning) must involve the labels Y as a latent factor

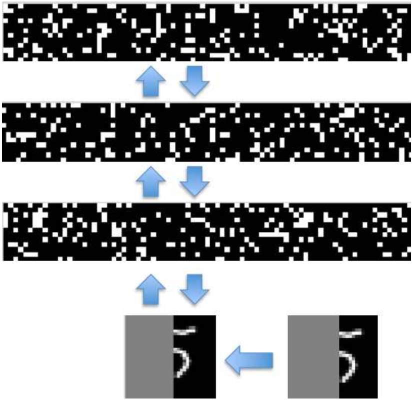
Unsupervised Learning

- So far we have only looked at *discriminative* models i.e. we model $Y = f(X;\theta)$ or $P(Y|X)$
- Recall: $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$
- $P(X)$ is defined in terms of $P(X|Y)$ or the best model of X (unsupervised learning) must involve the labels Y as a latent factor
- The idea of representation learning is to uncover the *latent variables* that *explain* X

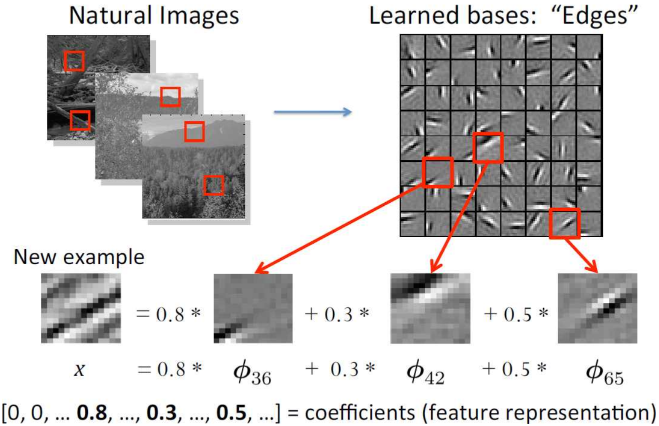
Unsupervised Learning



Unsupervised Learning

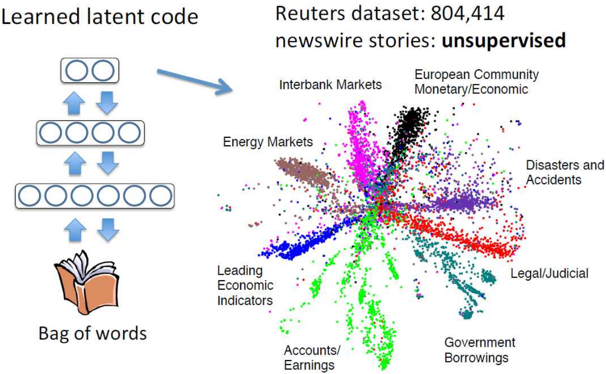


Unsupervised Learning



Slide credit: Honglak Lee

Unsupervised Learning



G. Hinton and R. Salakhutdinov, "Semantic Hashing", 2006

Unsupervised Learning

- Distributed representations (constraints on experts, compare to localist representations (e.g. Big Yellow Volk))

Unsupervised Learning

- Distributed representations (constraints on experts, compare to localist representations (e.g. Big Yellow Volk))
- Intrinsic latent dimensions

Unsupervised Learning

- Distributed representations (constraints on experts, compare to localist representations (e.g. Big Yellow Volk))
- Intrinsic latent dimensions
- Visualization

Unsupervised Learning

- Distributed representations (constraints on experts, compare to localist representations (e.g. Big Yellow Volk))
- Intrinsic latent dimensions
- Visualization
- Figuring explanatory factors

Unsupervised Learning

- Distributed representations (constraints on experts, compare to localist representations (e.g. Big Yellow Volk))
- Intrinsic latent dimensions
- Visualization
- Figuring explanatory factors
- Learning features for classification

Unsupervised Learning

- Distributed representations (constraints on experts, compare to localist representations (e.g. Big Yellow Volk))
- Intrinsic latent dimensions
- Visualization
- Figuring explanatory factors
- Learning features for classification
- Semi-supervised learning

Unsupervised Deep Learning

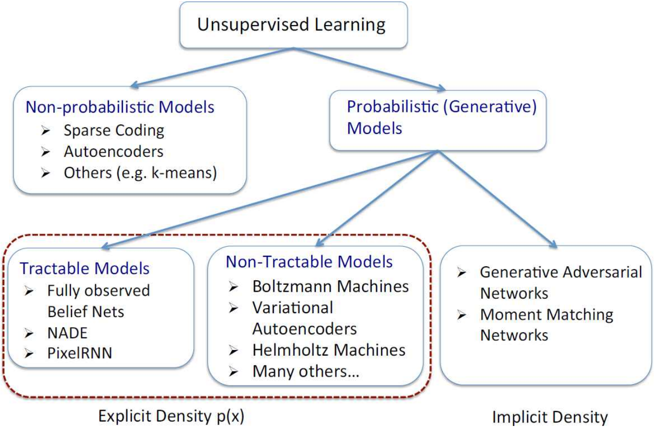


Figure: Ruslan Salakhutdinov

Warm Up

Linear Projections

- Suppose we have a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$

Linear Projections

- Suppose we have a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We don't have labels!

Linear Projections

- Suppose we have a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We don't have labels!
- We want to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

Linear Projections

- Suppose we have a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We don't have labels!
- We want to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

$$\tilde{\mathbf{x}}_i = \sum_{j=1}^{p} \alpha_{i,j} \mathbf{h}_j$$

Linear Projections

- Suppose we have a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We don't have labels!
- We want to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

$$\tilde{\mathbf{x}}_i = \sum_{j=1}^{p} \alpha_{i,j} \mathbf{h}_j$$

- We want to minimize:

Linear Projections

- Suppose we have a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We don't have labels!
- We want to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

$$\tilde{\mathbf{x}}_i = \sum_{j=1}^{p} \alpha_{i,j} \mathbf{h}_j$$

- We want to minimize:

$$Error = \sum_{i=1}^{N} (\tilde{\mathbf{x}}_i - \mathbf{x}_i)^2$$

Linear Projections

- Note that for bases $\mathbf{h}_1, \ldots, \mathbf{h}_N$

Linear Projections

- Note that for bases $\mathbf{h}_1, \ldots, \mathbf{h}_N$

$$\mathbf{x}_i = \sum_{j=1}^{N} \alpha_{i,j} \mathbf{h}_j$$

Linear Projections

- Note that for bases $\mathbf{h}_1, \ldots, \mathbf{h}_N$

$$\mathbf{x}_i = \sum_{j=1}^{N} \alpha_{i,j} \mathbf{h}_j$$

- We can now re-write the error:

$$Error = \sum_{i=1}^{N} \left(\sum_{j=1}^{p} \alpha_{i,j} \mathbf{h}_j - \sum_{j=1}^{N} \alpha_{i,j} \mathbf{h}_j \right)^2$$

Linear Projections

- Note that for bases $\mathbf{h}_1, \ldots, \mathbf{h}_N$

$$\mathbf{x}_i = \sum_{j=1}^{N} \alpha_{i,j} \mathbf{h}_j$$

- We can now re-write the error:

$$Error = \sum_{i=1}^{N} \left(\sum_{j=1}^{p} \alpha_{i,j} \mathbf{h}_j - \sum_{j=1}^{N} \alpha_{i,j} \mathbf{h}_j \right)^2$$

- After some basic manipulation ($\alpha_{i,j} = \mathbf{x}_i \mathbf{h}_j$):

Linear Projections

- Note that for bases $\mathbf{h}_1, \ldots, \mathbf{h}_N$

$$\mathbf{x}_i = \sum_{j=1}^{N} \alpha_{i,j} \mathbf{h}_j$$

- We can now re-write the error:

$$Error = \sum_{i=1}^{N} \left(\sum_{j=1}^{p} \alpha_{i,j} \mathbf{h}_j - \sum_{j=1}^{N} \alpha_{i,j} \mathbf{h}_j \right)^2$$

- After some basic manipulation ($\alpha_{i,j} = \mathbf{x}_i \mathbf{h}_j$):

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} \alpha_{i,j}$$

Linear Projections

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} \alpha_{i,j}$$

Linear Projections

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} \alpha_{i,j}$$

- Note that $\alpha_{i,j} = \mathbf{h}_j \mathbf{x}_i$, therefore:

Linear Projections

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} \alpha_{i,j}$$

- Note that $\alpha_{i,j} = \mathbf{h}_j \mathbf{x}_i$, therefore:

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} (\mathbf{h}_j \mathbf{x}_i)^2$$

Linear Projections

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} \alpha_{i,j}$$

- Note that $\alpha_{i,j} = \mathbf{h}_j \mathbf{x}_i$, therefore:

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} (\mathbf{h}_j \mathbf{x}_i)^2$$

- Which is just:

$$Error = \sum_{i=1}^{N} \sum_{j=p+1}^{N} \mathbf{h}_j^T \mathbf{x}_i \mathbf{x}_i^T \mathbf{h}_j$$

Linear Projections

$$Error = \sum_{j=p+1}^{N} \mathbf{h}_j^T \Sigma \mathbf{h}_j$$

- Now to find the minimizer, solve:

$$\min_{\mathbf{u}} \mathbf{u}\Sigma\mathbf{u} + \lambda(1 - \mathbf{u}^T\mathbf{u})$$

Linear Projections

$$Error = \sum_{j=p+1}^{N} \mathbf{h}_j^T \Sigma \mathbf{h}_j$$

- Now to find the minimizer, solve:

$$\min_{\mathbf{u}} \mathbf{u} \Sigma \mathbf{u} + \lambda(1 - \mathbf{u}^T \mathbf{u})$$

- The extra terms enforces orthonormality

Linear Projections

$$Error = \sum_{j=p+1}^{N} \mathbf{h}_j^T \Sigma \mathbf{h}_j$$

- Now to find the minimizer, solve:

$$\min_{\mathbf{u}} \mathbf{u} \Sigma \mathbf{u} + \lambda(1 - \mathbf{u}^T \mathbf{u})$$

- The extra terms enforces orthonormality
- Take derivative, and set to zero:

$$\mathbf{u}_i \Sigma = \lambda_i \mathbf{u}_i$$

Linear Projections

$$Error = \sum_{j=p+1}^{N} \mathbf{h}_j^T \Sigma \mathbf{h}_j$$

- Now to find the minimizer, solve:

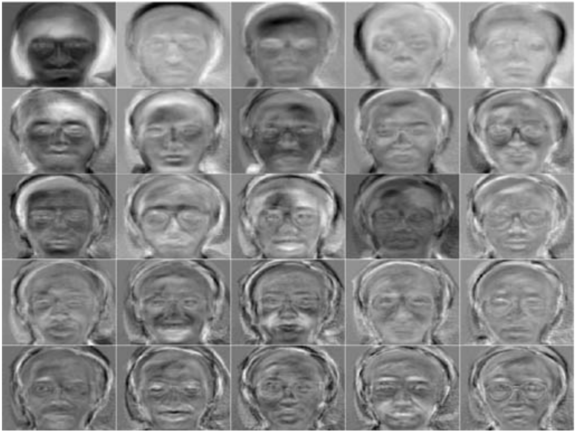
$$\min_{\mathbf{u}} \mathbf{u}\Sigma\mathbf{u} + \lambda(1 - \mathbf{u}^T\mathbf{u})$$

- The extra terms enforces orthonormality
- Take derivative, and set to zero:

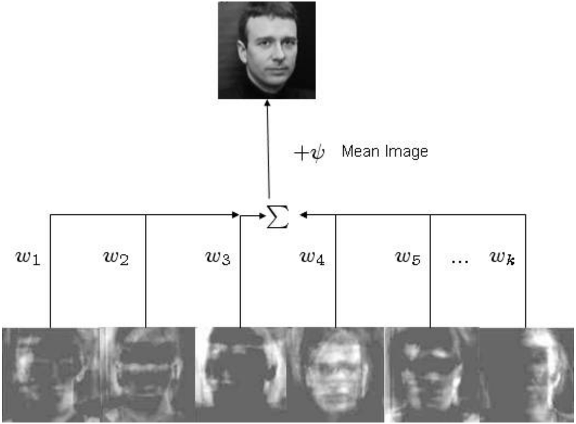
$$\mathbf{u}_i\Sigma = \lambda_i\mathbf{u}_i$$

- Solutions are eigenvectors!

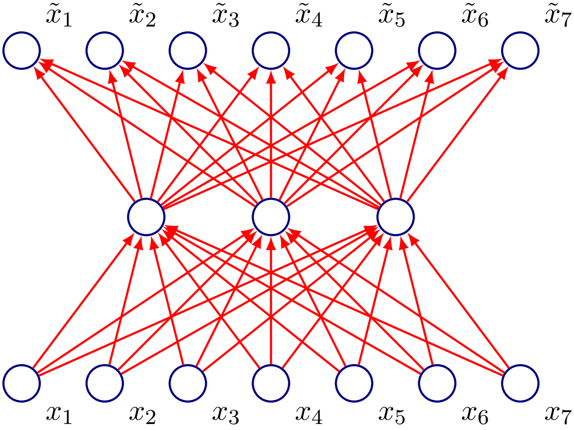
PCA on Face Images: Eigenfaces



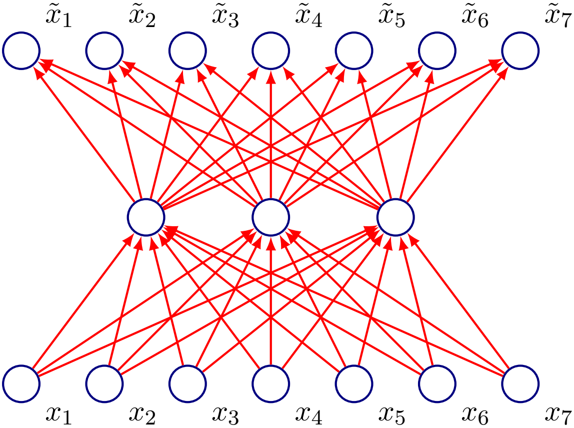
Eigenfaces: Features



A Linear Neural Network

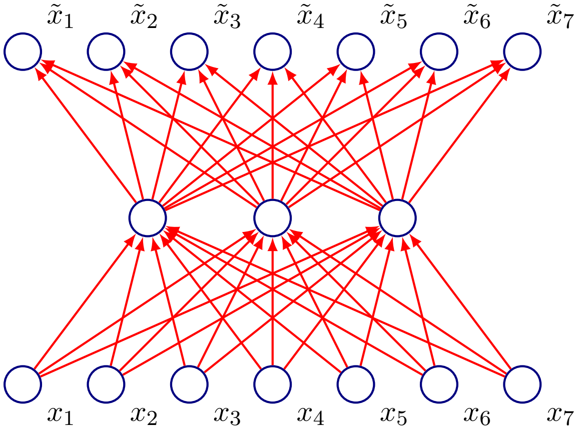


A Linear Neural Network



- Encoding: $\mathbf{x} \to \mathbf{h} = W\mathbf{x}$. Decoding: $\mathbf{h} \to \tilde{\mathbf{x}} = V\mathbf{h}$

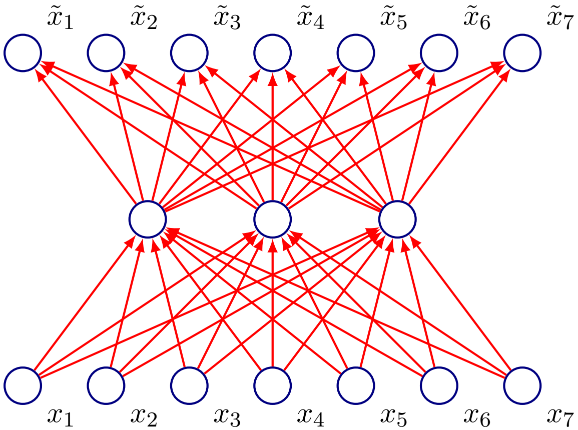
A Linear Neural Network



- Objective:

$$\min_{W,V} \|\mathbf{x} - VW\mathbf{x}\|_2^2$$

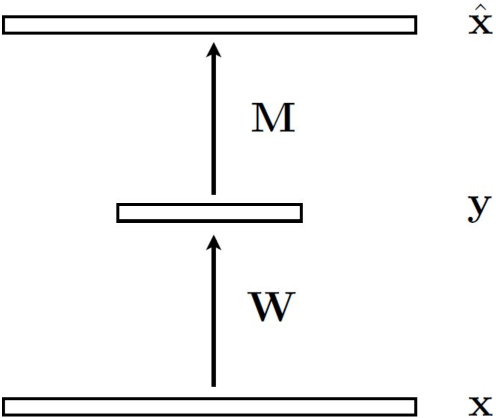
A Linear Neural Network



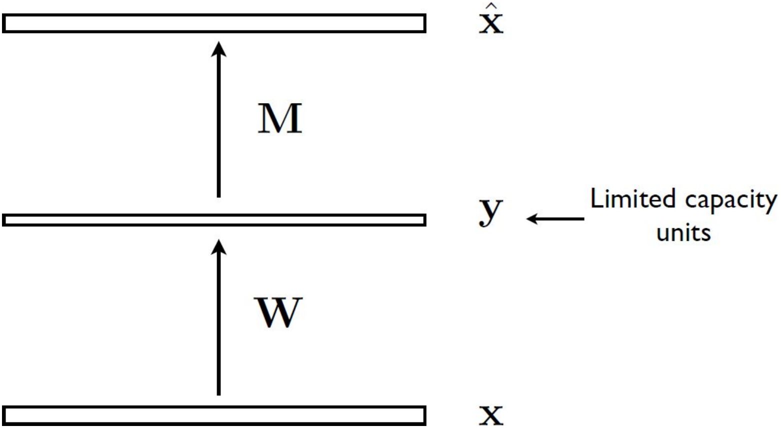
- This is a linear Autoencoder

Autoencoder: Non-Linear PCA

$$\min_{\mathbf{W},\mathbf{M}} |\mathbf{x} - \hat{\mathbf{x}}|^2$$

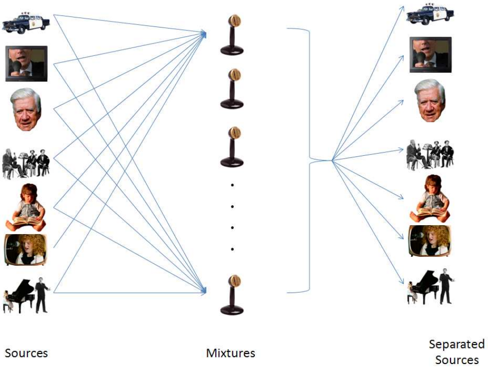


Autoencoder: Implicit Bottleneck



Another Linear Model: ICA

- Canonical example: Cocktail party problem



Another Linear Model: ICA

- Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_7$ are the microphone signals

Another Linear Model: ICA

- Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_7$ are the microphone signals
- Each $\mathbf{x}_i$ is a result of linear mixing between the sources $\mathbf{h}_i$

Another Linear Model: ICA

- Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_7$ are the microphone signals
- Each $\mathbf{x}_i$ is a result of linear mixing between the sources $\mathbf{h}_i$

$$\mathbf{x}_i = \sum_i \mathbf{a}_i \mathbf{h}_i \text{ or } X = AH$$

Another Linear Model: ICA

- Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_7$ are the microphone signals
- Each $\mathbf{x}_i$ is a result of linear mixing between the sources $\mathbf{h}_i$

$$\mathbf{x}_i = \sum_i \mathbf{a}_i \mathbf{h}_i \text{ or } X = AH$$

- Task: Only X is observed, A is unknown, recover H

Another Linear Model: ICA

- Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_7$ are the microphone signals
- Each $\mathbf{x}_i$ is a result of linear mixing between the sources $\mathbf{h}_i$

$$\mathbf{x}_i = \sum_i \mathbf{a}_i \mathbf{h}_i \text{ or } X = AH$$

- Task: Only X is observed, A is unknown, recover H
- Here the bases are *independent* of each other

Difference with PCA

- In PCA $X = AH$ with $H^T H = I$ i.e. bases are orthogonal

Difference with PCA

- In PCA $X = AH$ with $H^T H = I$ i.e. bases are orthogonal
- In ICA $X = AH$ with A invertible

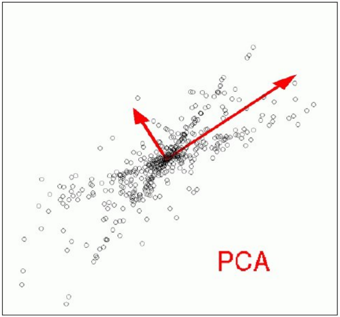
Difference with PCA

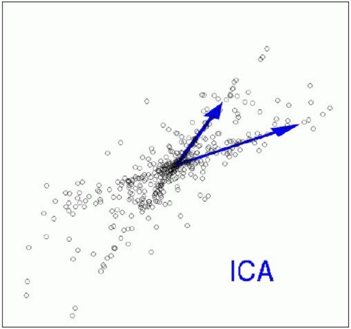
- In PCA $X = AH$ with $H^T H = I$ i.e. bases are orthogonal
- In ICA $X = AH$ with A invertible
- PCA does compression, ICA doesn't do any compression ($p = d$)

Difference with PCA

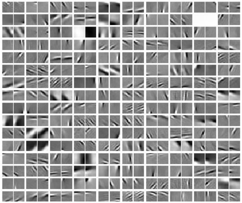
- In PCA $X = AH$ with $H^T H = I$ i.e. bases are orthogonal
- In ICA $X = AH$ with A invertible
- PCA does compression, ICA doesn't do any compression ($p = d$)
- Some PCs are more important than others, not in the case with ICA

Difference with PCA





Filters



Sparse Coding

- Objective: Given a set of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, learn a dictionary of bases $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p$ such that:

$$\mathbf{x}_i = \sum_{k=1}^{p} a_{ik} \mathbf{h}_k$$

Sparse Coding

- Objective: Given a set of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, learn a dictionary of bases $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p$ such that:

$$\mathbf{x}_i = \sum_{k=1}^{p} a_{ik}\mathbf{h}_k$$

- This such that most a_{ik} are zero i.e. very few bases *explain* $\mathbf{x}_i$

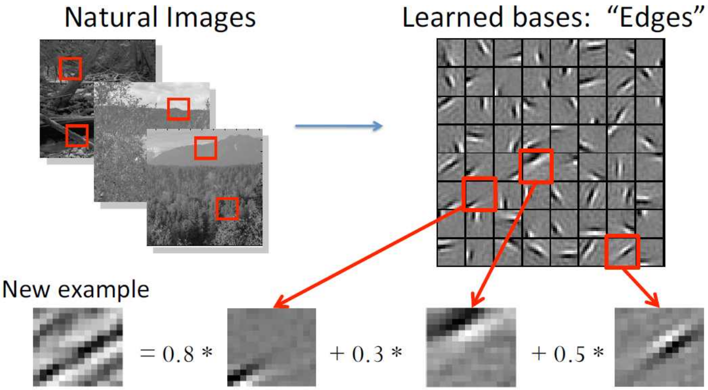
Sparse Coding

- Objective: Given a set of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, learn a dictionary of bases $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p$ such that:

$$\mathbf{x}_i = \sum_{k=1}^{p} a_{ik} \mathbf{h}_k$$

- This such that most a_{ik} are zero i.e. very few bases *explain* $\mathbf{x}_i$
- Like before, but data is now a *sparse* linear combination of bases

Sparse Coding



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

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

Sparse Coding

- Optimization: Given $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, learn dictionary $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p \in \mathbb{R}^d$ (arranged as $H = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p] \in \mathbb{R}^{d \times p}$) such that:

$$\min_{\mathbf{a}_1, \ldots, \mathbf{a}_N, H} \sum_{i=1}^{N} \|\mathbf{x}_i - H\mathbf{a}_i\|_2^2 + \lambda \sum_{i=1}^{N} \|\mathbf{a}_i\|_1$$

Sparse Coding

- Optimization: Given $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, learn dictionary $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p \in \mathbb{R}^d$ (arranged as $H = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p] \in \mathbb{R}^{d \times p}$) such that:

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_N, H} \sum_{i=1}^{N} \|\mathbf{x}_i - H\mathbf{a}_i\|_2^2 + \lambda \sum_{i=1}^{N} \|\mathbf{a}_i\|_1$$

- Reconstruction term: $\|\mathbf{x}_i - H\mathbf{a}_i\|_2^2$

Sparse Coding

- Optimization: Given $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, learn dictionary $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p \in \mathbb{R}^d$ (arranged as $H = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_p] \in \mathbb{R}^{d \times p}$) such that:

$$\min_{\mathbf{a}_1, \ldots, \mathbf{a}_N, H} \sum_{i=1}^{N} \|\mathbf{x}_i - H\mathbf{a}_i\|_2^2 + \lambda \sum_{i=1}^{N} \|\mathbf{a}_i\|_1$$

- Reconstruction term: $\|\mathbf{x}_i - H\mathbf{a}_i\|_2^2$
- Sparsity term: $\|\mathbf{a}_i\|_1$

Sparse Coding

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_N,H} \sum_{i=1}^{N} \|\mathbf{x}_i - H\mathbf{a}_i\|_2^2 + \lambda \sum_{i=1}^{N} \|\mathbf{a}_i\|_1$$

Sparse Coding

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_N,H} \sum_{i=1}^{N} \|\mathbf{x}_i - H\mathbf{a}_i\|_2^2 + \lambda \sum_{i=1}^{N} \|\mathbf{a}_i\|_1$$

- Optimization:
  1. Initialize $\mathbf{a}_1, \ldots, \mathbf{a}_N$ and $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$ randomly

Sparse Coding

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_N,H} \sum_{i=1}^{N} \|\mathbf{x}_i - H\mathbf{a}_i\|_2^2 + \lambda \sum_{i=1}^{N} \|\mathbf{a}_i\|_1$$

- Optimization:
  1. Initialize $\mathbf{a}_1,\ldots,\mathbf{a}_N$ and $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$ randomly
  2. Fix bases $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$ and optimize for codes $\mathbf{a}_i$

Sparse Coding

$$\min_{\mathbf{a}_1,\ldots,\mathbf{a}_N,H} \sum_{i=1}^{N} \|\mathbf{x}_i - H\mathbf{a}_i\|_2^2 + \lambda \sum_{i=1}^{N} \|\mathbf{a}_i\|_1$$

- Optimization:
  1. Initialize $\mathbf{a}_1, \ldots, \mathbf{a}_N$ and $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$ randomly
  2. Fix bases $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$ and optimize for codes $\mathbf{a}_i$
  3. Fix codes $\mathbf{a}_i$ and optimize for H (convex)

Sparse Coding:Test Time

- Given a new patch $\tilde{\mathbf{x}} \in \mathbb{R}^d$ and learned dictionary $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$, we find the code $\tilde{\mathbf{a}}$ as:

Sparse Coding:Test Time

- Given a new patch $\tilde{\mathbf{x}} \in \mathbb{R}^d$ and learned dictionary $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$, we find the code $\tilde{\mathbf{a}}$ as:

$$\min_{\tilde{\mathbf{a}}} \|\tilde{\mathbf{x}} - H\tilde{\mathbf{a}}\|_2^2 + \lambda \|\tilde{\mathbf{a}}\|_1$$

Sparse Coding:Test Time

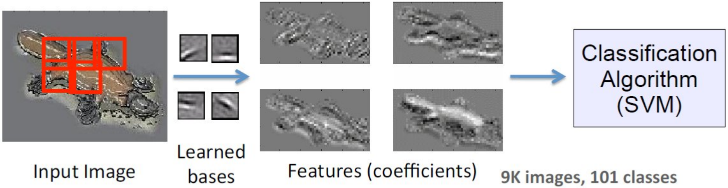
- Given a new patch $\tilde{\mathbf{x}} \in \mathbb{R}^d$ and learned dictionary $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$, we find the code $\tilde{\mathbf{a}}$ as:

$$\min_{\tilde{\mathbf{a}}} \|\tilde{\mathbf{x}} - H\tilde{\mathbf{a}}\|_2^2 + \lambda \|\tilde{\mathbf{a}}\|_1$$

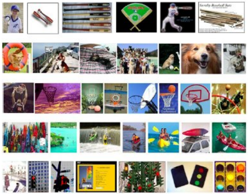
- $\tilde{\mathbf{a}}$ will be a sparse representation for $\tilde{\mathbf{x}}$

Image Classification

Evaluated on Caltech101 object category dataset.



9K images, 101 classes



Algorithm	Accuracy
Baseline (Fei-Fei et al., 2004)	16%
PCA	37%
Sparse Coding	**47%**

Slide Credit: Honglak Lee

Lee, Battle, Raina, Ng, 2006

Features for Faces

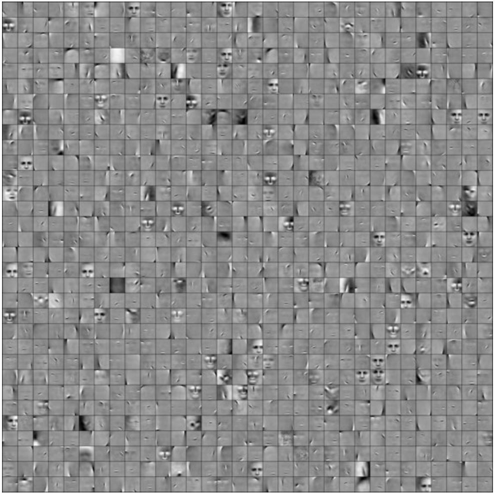


Figure: Charles Cadieu

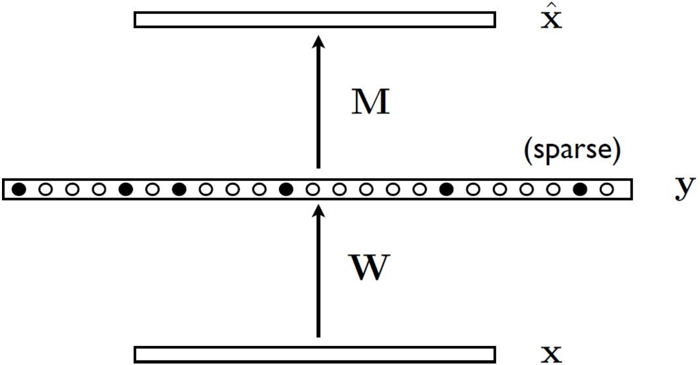
Encoding-Decoding

- Encoding: Implicit non-linear (in $\mathbf{x}$) encoding
- Decoding: Explicit linear decoding
- Can be overcomplete

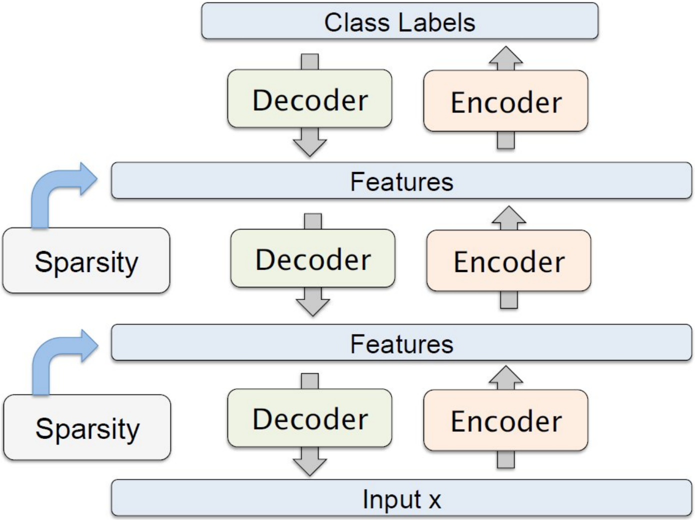
Encoding-Decoding

Simple Neural Network

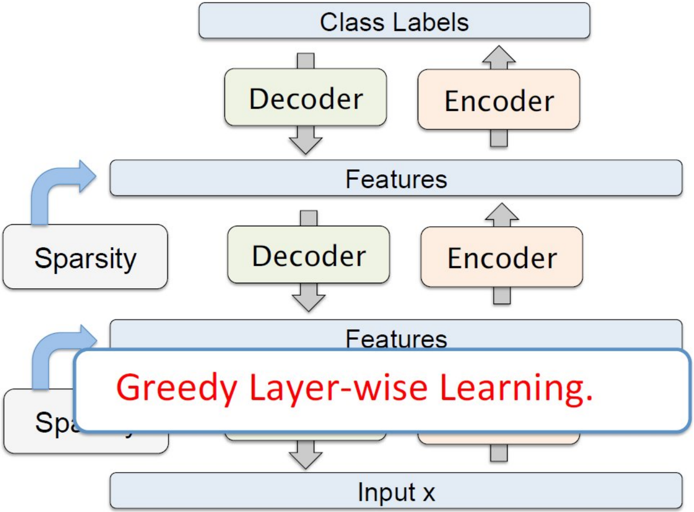
Sparse Autoencoders



Stacked Autoencoders



Pre-Training



Deep Autoencoders (2006)

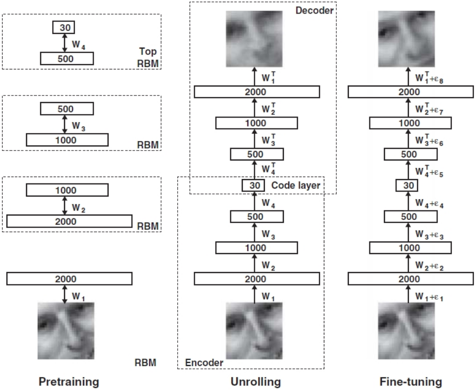
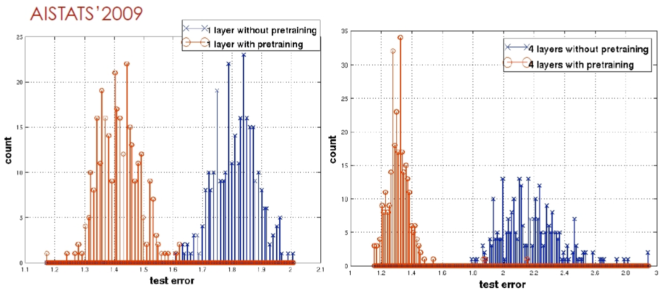


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

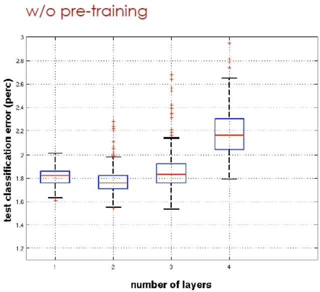
G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science, 2006

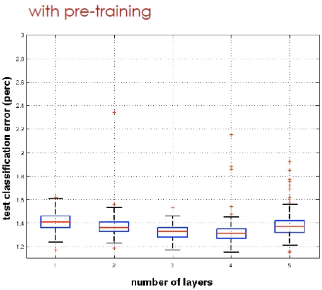
It was hard to train deep feedforward networks from scratch in 2006!

Effect of Unsupervised Pre-training



Effect of Unsupervised Pre-training





Why does Unsupervised Pre-training work?

- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$

Why does Unsupervised Pre-training work?

- Regularization. Feature representations that are good for $P(x)$ are good for $P(y|x)$
- Optimization: Unsupervised pre-training leads to better regions of the space i.e. better than random initialization

More Autoencoders

- De-noising Autoencoders: Input is corrupted by noise, but we attempt to reconstruct the uncorrupted image

More Autoencoders

- De-noising Autoencoders: Input is corrupted by noise, but we attempt to reconstruct the uncorrupted image
- Contractive Autoencoders: The regularization term penalizes for the derivative:

$$\Omega(\mathbf{h}, \mathbf{x}) = \lambda \sum_i \|\nabla_{\mathbf{x}} \mathbf{h}_i\|_2^2$$

De-Noising Autoencoder: Intuition

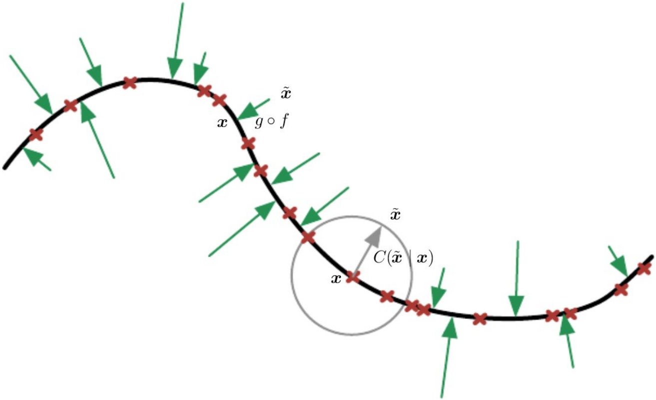


Figure: Goodfellow *et al.*

Back to Simple Models

Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$

Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$
- Often we might be interested in latent factors $\mathbf{h}$ that *explain* $\mathbf{x}$

Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$
- Often we might be interested in latent factors $\mathbf{h}$ that *explain* $\mathbf{x}$
- We then care about the marginal:

$$\tilde{P}(\mathbf{x}) = \mathbb{E}_{\mathbf{h}} \tilde{P}(\mathbf{x}|\mathbf{h})$$

Linear Factor Model

- We want to build a probabilistic model of the input $\tilde{P}(\mathbf{x})$
- Often we might be interested in latent factors $\mathbf{h}$ that *explain* $\mathbf{x}$
- We then care about the marginal:

$$\tilde{P}(\mathbf{x}) = \mathbb{E}_{\mathbf{h}} \tilde{P}(\mathbf{x}|\mathbf{h})$$

- $\mathbf{h}$ is a *representation* of the data

Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data

Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise

Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise
- Formally: Suppose we sample the latent factors from a distribution $\mathbf{h} \sim P(\mathbf{h})$

Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise
- Formally: Suppose we sample the latent factors from a distribution $\mathbf{h} \sim P(\mathbf{h})$
- Then: $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$

Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise
- Formally: Suppose we sample the latent factors from a distribution $\mathbf{h} \sim P(\mathbf{h})$
- Then: $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- How do we figure *good* representations that explain the data well?

Linear Factor Model

- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise
- Formally: Suppose we sample the latent factors from a distribution $\mathbf{h} \sim P(\mathbf{h})$
- Then: $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$
- How do we figure *good* representations that explain the data well?
- What would explaining the data mean?

Factor Analysis

- Suppose we fix the latent factor prior to be the unit Gaussian:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

Factor Analysis

- Suppose we fix the latent factor prior to be the unit Gaussian:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- Now, we need to specify a noise model. Assume it comes from a Gaussian with covariance $\Sigma = diag(\sigma_i^2)$

Factor Analysis

- Suppose we fix the latent factor prior to be the unit Gaussian:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- Now, we need to specify a noise model. Assume it comes from a Gaussian with covariance $\Sigma = diag(\sigma_i^2)$
- For this simple model, $\mathbf{x}$ is also a multivariate Gaussian:

Factor Analysis

- Suppose we fix the latent factor prior to be the unit Gaussian:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- Now, we need to specify a noise model. Assume it comes from a Gaussian with covariance $\Sigma = diag(\sigma_i^2)$
- For this simple model, $\mathbf{x}$ is also a multivariate Gaussian:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x}; b, WW^T + \Sigma)$$

Probabilistic PCA

- We only need to make a small change in our general factor analysis model

Probabilistic PCA

- We only need to make a small change in our general factor analysis model
- Still sample $\mathbf{h}$ as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

Probabilistic PCA

- We only need to make a small change in our general factor analysis model
- Still sample $\mathbf{h}$ as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- But now we assume a noise model which is a Gaussian with covariance $\sigma_i^2 I$

Probabilistic PCA

- We only need to make a small change in our general factor analysis model
- Still sample $\mathbf{h}$ as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- But now we assume a noise model which is a Gaussian with covariance $\sigma_i^2 I$
- Then, the conditional distribution becomes:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x}; b, WW^T + \sigma^2 I)$$

Probabilistic PCA

- We only need to make a small change in our general factor analysis model
- Still sample $\mathbf{h}$ as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- But now we assume a noise model which is a Gaussian with covariance $\sigma_i^2 I$
- Then, the conditional distribution becomes:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x}; b, WW^T + \sigma^2 I)$$

- Or $\mathbf{x} = W\mathbf{h} + \mathbf{b} +$ noise

Probabilistic PCA

- We only need to make a small change in our general factor analysis model
- Still sample $\mathbf{h}$ as before:

$$\mathbf{h} \sim \mathcal{N}(\mathbf{h}; 0, I)$$

- But now we assume a noise model which is a Gaussian with covariance $\sigma_i^2 I$
- Then, the conditional distribution becomes:

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x}; b, WW^T + \sigma^2 I)$$

- Or $\mathbf{x} = W\mathbf{h} + \mathbf{b} +$ noise
- Approaches PCA as $\sigma \to 0$

Energy Based Models and PoE

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration

Energy Based Models and PoE

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration
- Learning: Change the energy function so that its final shape has some desirable properties

Energy Based Models and PoE

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration
- Learning: Change the energy function so that its final shape has some desirable properties
- We can define a probability distribution through an energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

Energy Based Models and PoE

- Energy-Based Models assign a scalar energy with *every configuration* of variables under consideration
- Learning: Change the energy function so that its final shape has some desirable properties
- We can define a probability distribution through an energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

- Energies are in the log-probability domain:

$$\text{Energy}(\mathbf{x}) = \log \frac{1}{(ZP(\mathbf{x}))}$$

Energy Based Models and PoE

- $$P(\mathbf{x}) = \frac{\exp^{-(\mathsf{Energy}(\mathbf{x}))}}{Z}$$

Energy Based Models and PoE

- $$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

- Z is a normalizing factor called the Partition Function

$$Z = \sum_{\mathbf{x}} \exp(-\text{Energy}(\mathbf{x}))$$

Energy Based Models and PoE

- $$P(\mathbf{x}) = \frac{\exp^{-(\text{Energy}(\mathbf{x}))}}{Z}$$

- Z is a normalizing factor called the Partition Function

$$Z = \sum_{\mathbf{x}} \exp(-\text{Energy}(\mathbf{x}))$$

- How do we specify the energy function?

Product of Experts Formulation

- In this formulation, the energy function is:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x})$$

Product of Experts Formulation

- In this formulation, the energy function is:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x})$$

- Therefore:

$$P(\mathbf{x}) = \frac{\exp^{-(\sum_i f_i(\mathbf{x}))}}{Z}$$

Product of Experts Formulation

- In this formulation, the energy function is:

$$\text{Energy}(\mathbf{x}) = \sum_i f_i(\mathbf{x})$$

- Therefore:

$$P(\mathbf{x}) = \frac{\exp^{-(\sum_i f_i(\mathbf{x}))}}{Z}$$

- We have the product of experts:

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert f_i can be seen as enforcing a constraint on $\mathbf{x}$

Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert f_i can be seen as enforcing a constraint on $\mathbf{x}$
- If f_i is large $\implies$ $P_i(\mathbf{x})$ is small i.e. the expert thinks $\mathbf{x}$ is implausible (constraint violated)

Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert f_i can be seen as enforcing a constraint on $\mathbf{x}$
- If f_i is large $\implies P_i(\mathbf{x})$ is small i.e. the expert thinks $\mathbf{x}$ is implausible (constraint violated)
- If f_i is small $\implies P_i(\mathbf{x})$ is large i.e. the expert thinks $\mathbf{x}$ is plausible (constraint satisfied)

Product of Experts Formulation

$$P(\mathbf{x}) \propto \prod_i P_i(\mathbf{x}) \propto \prod_i \exp^{(-f_i(\mathbf{x}))}$$

- Every expert f_i can be seen as enforcing a constraint on $\mathbf{x}$
- If f_i is large $\implies P_i(\mathbf{x})$ is small i.e. the expert thinks $\mathbf{x}$ is implausible (constraint violated)
- If f_i is small $\implies P_i(\mathbf{x})$ is large i.e. the expert thinks $\mathbf{x}$ is plausible (constraint satisfied)
- Contrast this with mixture models

Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$

Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$
- The probability then becomes:

$$P(\mathbf{x}, \mathbf{h}) = \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$
- The probability then becomes:

$$P(\mathbf{x}, \mathbf{h}) = \frac{\exp^{-(\mathsf{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We only care about the marginal:

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\mathsf{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden factors that *explain* $\mathbf{x}$
- The probability then becomes:

$$P(\mathbf{x},\mathbf{h}) = \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We only care about the marginal:

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

Latent Variables

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\mathsf{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

Latent Variables

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\text{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We introduce another term from statistical physics: free energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\text{FreeEnergy}(\mathbf{x}))}}{Z}$$

Latent Variables

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{\exp^{-(\textsf{Energy}(\mathbf{x},\mathbf{h}))}}{Z}$$

- We introduce another term from statistical physics: free energy:

$$P(\mathbf{x}) = \frac{\exp^{-(\textsf{FreeEnergy}(\mathbf{x}))}}{Z}$$

- Free Energy is just a marginalization of energies in the log-domain:

$$\textsf{FreeEnergy}(\mathbf{x}) = -\log \sum_{\mathbf{h}} \exp^{-(\textsf{Energy}(\mathbf{x},\mathbf{h}))}$$