Lecture 15
Introduction to Deep Unsupervised Learning II

CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

May 17, 2017

Recap: Unsupervised Deep Learning

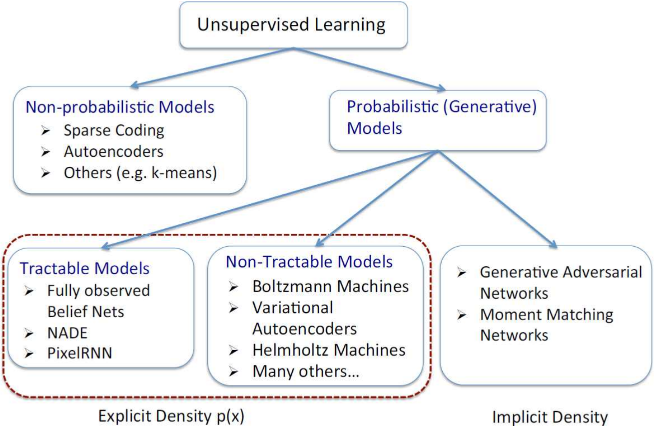


Figure: Ruslan Salakhutdinov

Recap: PCA

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

Recap: PCA

- For a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We wanted to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

Recap: PCA

- For a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We wanted to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

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

Recap: PCA

- For a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We wanted to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

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

- To minimize the error: $Error = \sum_{i=1}^{N}(\tilde{\mathbf{x}}_i - \mathbf{x}_i)^2$

Recap: PCA

- For a mean-centered dataset X with N datapoints $\mathbf{x}_1, \dots, \mathbf{x}_N \in \mathbb{R}^d$
- We wanted to find bases $\mathbf{h}_1, \dots, \mathbf{h}_p$ such that each:

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

- To minimize the error: $Error = \sum_{i=1}^{N} (\tilde{\mathbf{x}}_i - \mathbf{x}_i)^2$
- Solution: $\mathbf{h}_1, \dots, \mathbf{h}_p$ are the first p *eigenvectors* of the sample covariance matrix

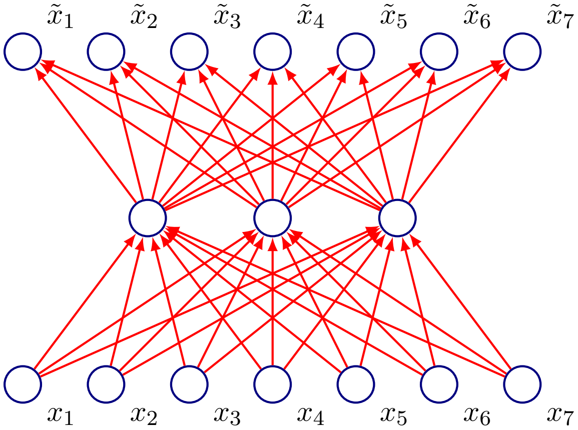
Recap: PCA

- For a mean-centered dataset X with N datapoints $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$
- We wanted to find bases $\mathbf{h}_1, \ldots, \mathbf{h}_p$ such that each:

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

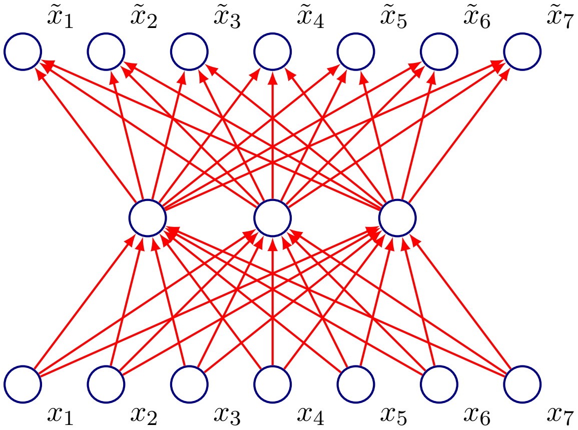
- To minimize the error: $Error = \sum_{i=1}^{N} (\tilde{\mathbf{x}}_i - \mathbf{x}_i)^2$
- Solution: $\mathbf{h}_1, \ldots, \mathbf{h}_p$ are the first p *eigenvectors* of the sample covariance matrix
- The bases are orthogonal. Coefficient vectors are dense

Recap: Linear Autoencoder



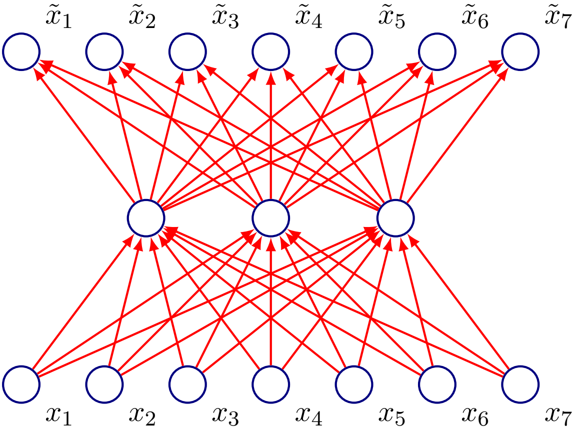
- Encoding: $\mathbf{x} \rightarrow \mathbf{h} = W\mathbf{x}$. Decoding: $\mathbf{h} \rightarrow \tilde{\mathbf{x}} = V\mathbf{h}$

Recap: Linear Autoencoder



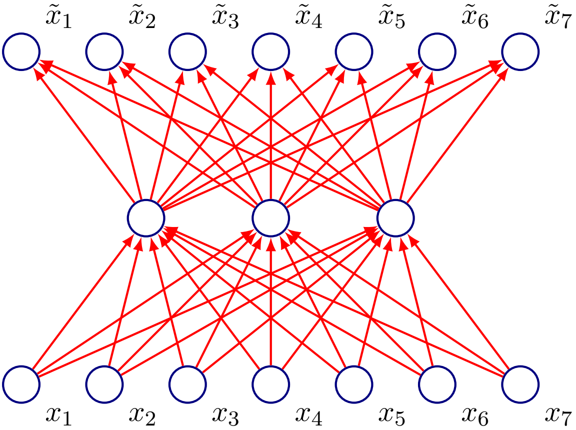
- Encoding: $\mathbf{x} \to \mathbf{h} = W\mathbf{x}$. Decoding: $\mathbf{h} \to \tilde{\mathbf{x}} = V\mathbf{h}$
- Objective: $\min_{W,V} \|\mathbf{x} - VW\mathbf{x}\|_2^2$

Recap: Simple Non-Linear Autoencoder



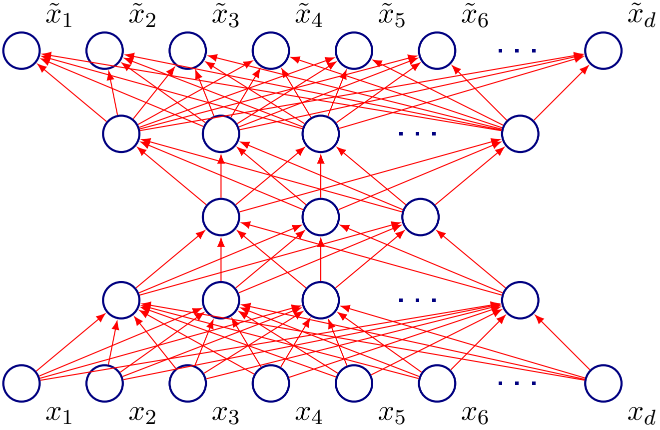
- Encoding: $\mathbf{x} \mapsto \mathbf{h} = \tanh(W\mathbf{x})$. Decoding: $\mathbf{h} \mapsto \tilde{\mathbf{x}} = \tanh(V\mathbf{h})$

Recap: Simple Non-Linear Autoencoder



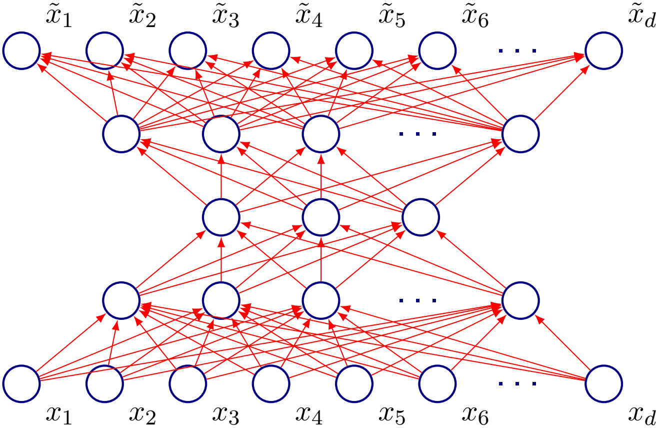
- Encoding: $\mathbf{x} \mapsto \mathbf{h} = \tanh(W\mathbf{x})$. Decoding: $\mathbf{h} \mapsto \tilde{\mathbf{x}} = \tanh(V\mathbf{h})$
- Objective: $\min_{W,V} \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2$

Recap: Non-Linear Autoencoder



- Encoding: $\mathbf{h} = \tanh(W_2 \tanh(W_1 \mathbf{x}))$. Decoding: $\tilde{\mathbf{x}} = \tanh(W_1' \tanh(W_2' \mathbf{h}))$

Recap: Non-Linear Autoencoder



- Encoding: $\mathbf{h} = \tanh(W_2 \tanh(W_1 \mathbf{x}))$. Decoding: $\tilde{\mathbf{x}} = \tanh(W_1' \tanh(W_2' \mathbf{h}))$
- Objective: $\min_{W_2, W_1, W_1', W_2'} \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2$

Recap: Non-Linear Autoencoders

- So far we have only seen auto-encoders that have a bottleneck

Recap: Non-Linear Autoencoders

- So far we have only seen auto-encoders that have a bottleneck
- The forms for encoding and decoding can be different than those specified

Recap: Non-Linear Autoencoders

- So far we have only seen auto-encoders that have a bottleneck
- The forms for encoding and decoding can be different than those specified
- We get non-linear projections or *representations* of the data

Recap: Non-Linear Autoencoders

- So far we have only seen auto-encoders that have a bottleneck
- The forms for encoding and decoding can be different than those specified
- We get non-linear projections or *representations* of the data
- Can be seen as a form of non-linear PCA

Recap: 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:

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

Recap: 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:

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

- Most a_{ik} values are zero i.e. very few factors *explain* $\mathbf{x}_i$

Recap: Sparse Coding

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

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

- Most a_{ik} values are zero i.e. very few factors *explain* $\mathbf{x}_i$
- In PCA, the bases $\mathbf{h}$'s were orthogonal and the codes for the $\mathbf{x}$ i.e. $\mathbf{a}$'s were dense.

Recap: 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:

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

- Most a_{ik} values are zero i.e. very few factors *explain* $\mathbf{x}_i$
- In PCA, the bases $\mathbf{h}$'s were orthogonal and the codes for the $\mathbf{x}$ i.e. $\mathbf{a}$'s were dense.
- Here, the bases need not be orthogonal, but the codes are sparse

Recap: Sparse Coding

- Optimization Problem: 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:

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

Recap: Sparse Coding

- Optimization Problem: 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: $\|\mathbf{x}_i - H\mathbf{a}_i\|_2^2$

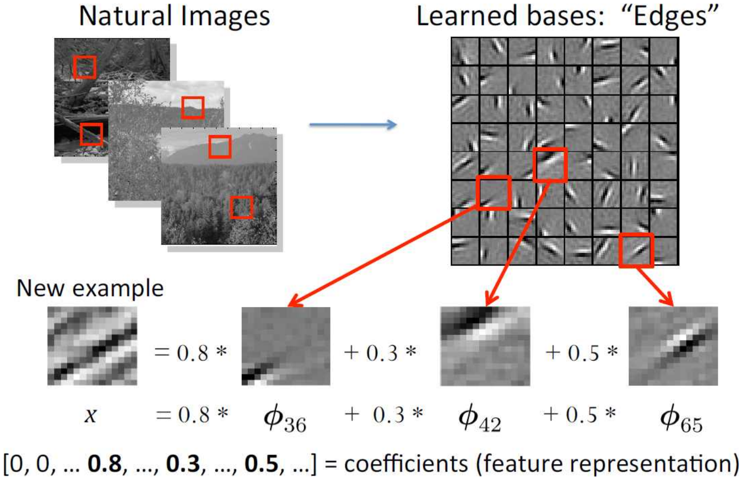
Recap: Sparse Coding

- Optimization Problem: 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



The ϕ's here are our $\mathbf{h}$
Figure: Honglak Lee

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 $\tilde{\mathbf{x}}$

Sparse Coding:Test Time

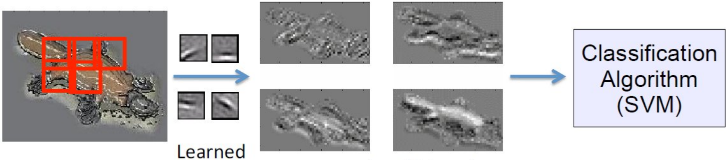
- Given a new patch $\tilde{\mathbf{x}} \in \mathbb{R}^d$ and learned dictionary $H = [\mathbf{h}_1 \ldots, \mathbf{h}_p]$, we find the code $\tilde{\mathbf{a}}$ as:

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

- $\tilde{\mathbf{a}}$ will be a sparse representation for $\tilde{\mathbf{x}}$
- Again, $\tilde{\mathbf{a}}$ is our representation or *code* for $\tilde{\mathbf{x}}$ that we can use as features for classification

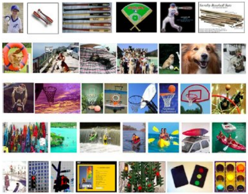
Image Classification

Evaluated on Caltech101 object category dataset.



Input Image — Learned bases — Features (coefficients)

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

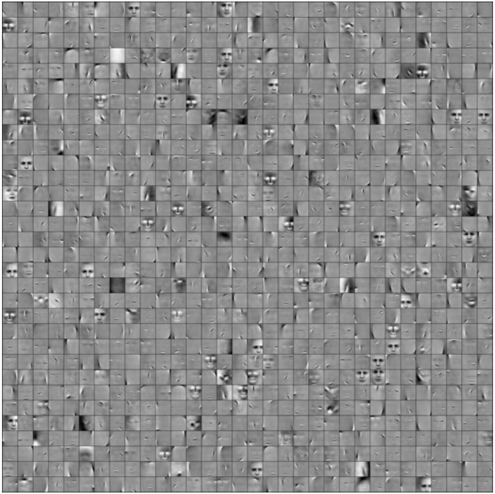


Figure: Charles Cadieu

Encoding-Decoding

- Encoding: Implicit encoding, non-linear (in $\mathbf{x}$)

Encoding-Decoding

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

Encoding-Decoding

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

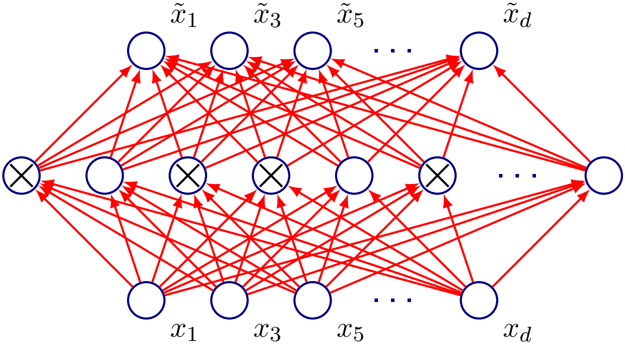
Encoding-Decoding

- Encoding: Implicit encoding, non-linear (in $\mathbf{x}$)
- Decoding: Explicit linear decoding
- Bases is overcomplete
- In PCA, plain autoencoders (i.e. Non-Linear PCA) overcomplete representations don't make much sense (can just copy input!)

- Like before, as in the case of PCA, let us try to write sparse coding as a neural network

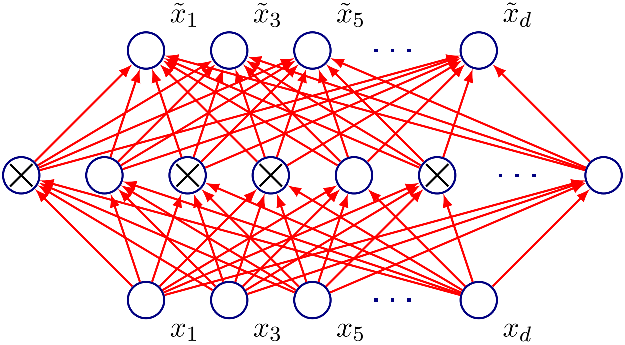
- Like before, as in the case of PCA, let us try to write sparse coding as a neural network
- Will lead to another kind of auto-encoder

Implicit Bottleneck



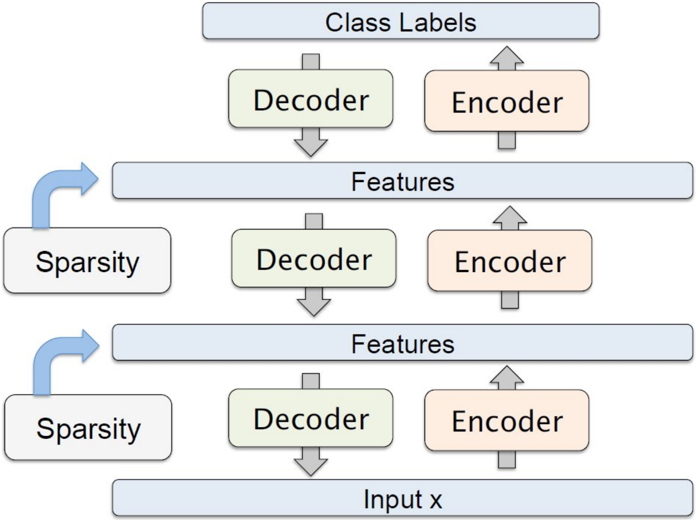
- Encoding: $\mathbf{h} = \tanh(W\mathbf{x})$. Decoding: $\tilde{\mathbf{x}} = V\mathbf{h}$

Implicit Bottleneck

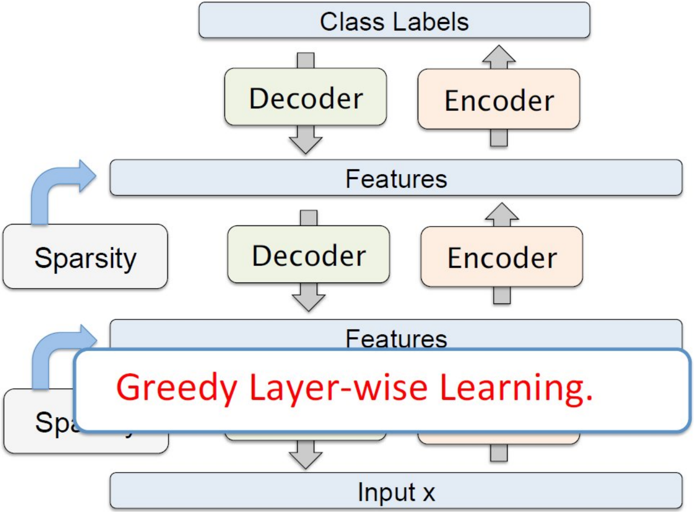


- Encoding: $\mathbf{h} = \tanh(W\mathbf{x})$. Decoding: $\tilde{\mathbf{x}} = V\mathbf{h}$
- PS: Modified model than the sparse coding model we saw (but to emphasize nonlinearity in encoding, and linear decoding)

Stacked Autoencoders



Pre-Training



Deep Autoencoders (2006)

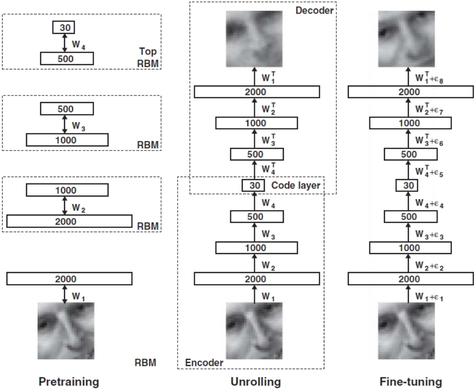
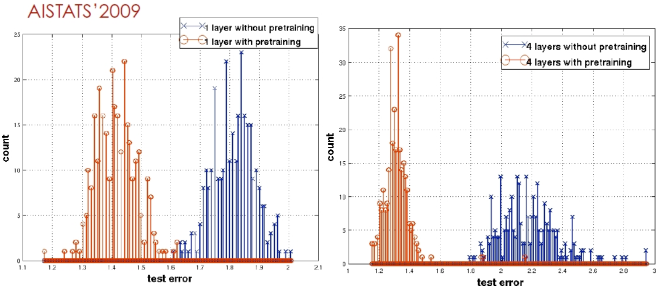


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the "data" for training the next RBM in the stack. After the pretraining, the RBMs are "unrolled" to create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

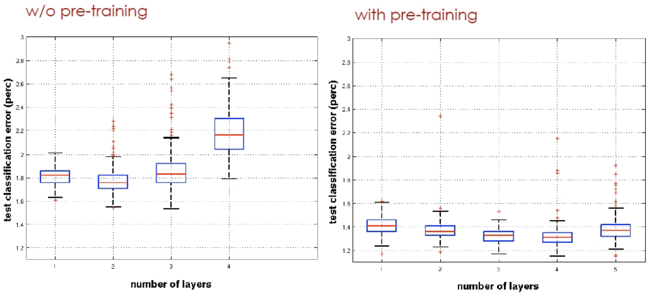
G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science, 2006

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

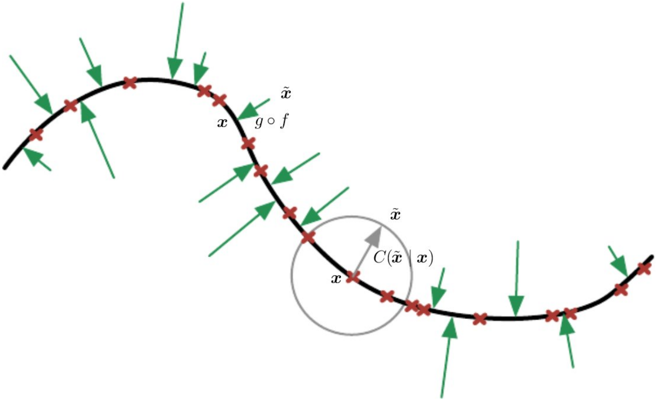
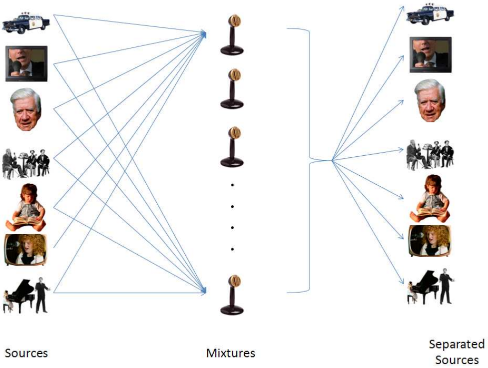


Figure: Goodfellow *et al.*

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

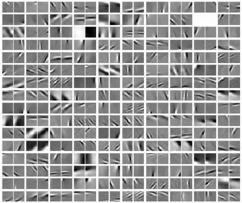
Difference with PCA

- 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

Filters



Short Digression: Distributed Representations

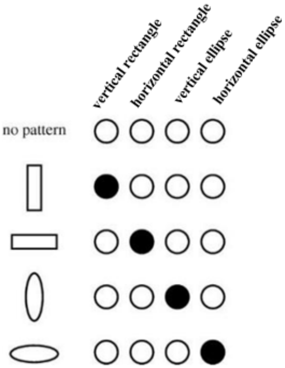
- All the models that we have seen so far today have something in common: They are distributed representations

- All the models that we have seen so far today have something in common: They are distributed representations
- PCA is a dense distributed representation unlike sparse coding

- All the models that we have seen so far today have something in common: They are distributed representations
- PCA is a dense distributed representation unlike sparse coding
- One of the reasons of the power of Deep Networks are distributed representations (which unlike these toy cases are highly non-linear)

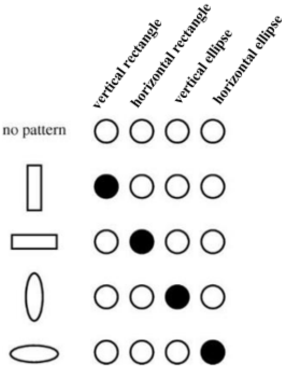
- All the models that we have seen so far today have something in common: They are distributed representations
- PCA is a dense distributed representation unlike sparse coding
- One of the reasons of the power of Deep Networks are distributed representations (which unlike these toy cases are highly non-linear)
- What are distributed representations?

Distributed Representations: Intuition



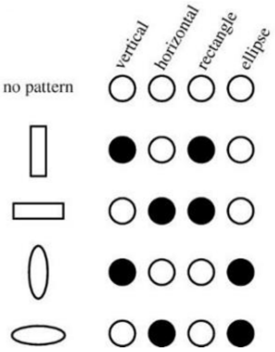
- This is a *localist* representation: Every concept gets a code that has local structure

Distributed Representations: Intuition



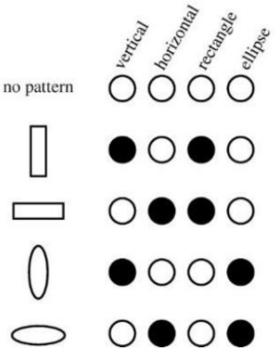
- This is a *localist* representation: Every concept gets a code that has local structure
- Very easy to code, and easy to learn (mixture models build representations like this)

Distributed Representations: Intuition



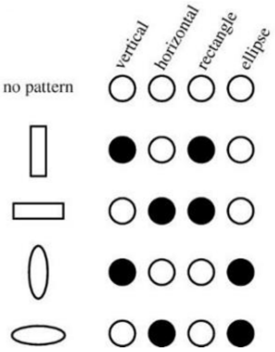
- This is a *distributed* representation:

Distributed Representations: Intuition



- This is a *distributed* representation:
- Each concept is represented by multiple neurons

Distributed Representations: Intuition



- This is a *distributed* representation:
- Each concept is represented by multiple neurons
- Given an exponential advantage in representational efficiency

Representations

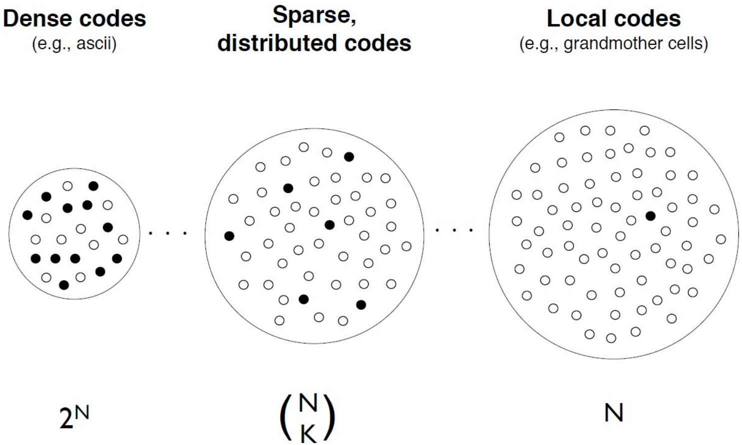
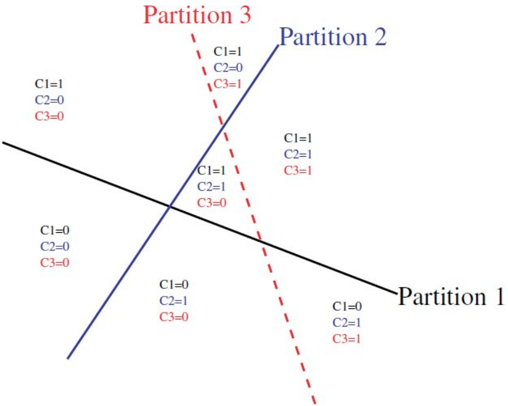


Figure: Bruno Olshausen

Representations



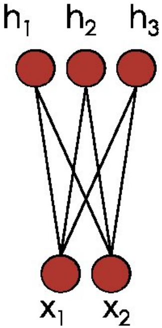


Figure: Yoshua Bengio (FTML Volume)

Exponential Advantage

- A Localist representation will be able to distinguish patterns linear in the dimension of the representation

Exponential Advantage

- A Localist representation will be able to distinguish patterns linear in the dimension of the representation
- A Distributed representation can distinguish patterns exponential in the dimension of the representation

Exponential Advantage

- A Localist representation will be able to distinguish patterns linear in the dimension of the representation
- A Distributed representation can distinguish patterns exponential in the dimension of the representation
- Exercise inspired from previous figure

Exponential Advantage

- A Localist representation will be able to distinguish patterns linear in the dimension of the representation
- A Distributed representation can distinguish patterns exponential in the dimension of the representation
- Exercise inspired from previous figure
- How many regions can lines carve in a 2-D space?

Exponential Advantage

- A Localist representation will be able to distinguish patterns linear in the dimension of the representation
- A Distributed representation can distinguish patterns exponential in the dimension of the representation
- Exercise inspired from previous figure
- How many regions can lines carve in a 2-D space?
  - Num. lines + num. intersections + 1

Exponential Advantage

- A Localist representation will be able to distinguish patterns linear in the dimension of the representation
- A Distributed representation can distinguish patterns exponential in the dimension of the representation
- Exercise inspired from previous figure
- How many regions can lines carve in a 2-D space?
  - Num. lines + num. intersections + 1
- How many regions can m hyperplanes carve in a d-dimensional space?

Exponential Advantage

- A Localist representation will be able to distinguish patterns linear in the dimension of the representation
- A Distributed representation can distinguish patterns exponential in the dimension of the representation
- Exercise inspired from previous figure
- How many regions can lines carve in a 2-D space?
    - Num. lines + num. intersections + 1
- How many regions can m hyperplanes carve in a d-dimensional space?
    - $1 + m + \binom{m}{2} + \cdots + \binom{m}{d}$

End of Digression

- Distributed representations are one of the major reasons of the success of Deep Learning methods in complicated tasks

End of Digression

- Distributed representations are one of the major reasons of the success of Deep Learning methods in complicated tasks
- Just a distributed representation is not enough (e.g. PCA is distributed)

End of Digression

- Distributed representations are one of the major reasons of the success of Deep Learning methods in complicated tasks
- Just a distributed representation is not enough (e.g. PCA is distributed)
- The representations are non-linear, hierarchical amongst other things

End of Digression

- Distributed representations are one of the major reasons of the success of Deep Learning methods in complicated tasks
- Just a distributed representation is not enough (e.g. PCA is distributed)
- The representations are non-linear, hierarchical amongst other things
- Note: This is not specific to just unsupervised deep learning

Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants

Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)

Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders

Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic

Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic
- The encoder and decoders have no stochasticity

Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic
- The encoder and decoders have no stochasticity
- We don't construct a probabilistic model of the data

Approach so far:

- We have considered simple models and then constructed their deep, non-linear variants
- Example: PCA (and Linear Autoencoder) to Nonlinear-PCA (Non-linear (deep?) autoencoders)
- Example: Sparse Coding (Sparse Autoencoder with linear decoding) to Deep Sparse Autoencoders
- All the models we have considered so far are completely deterministic
- The encoder and decoders have no stochasticity
- We don't construct a probabilistic model of the data
- Can't sample from the data

Representations

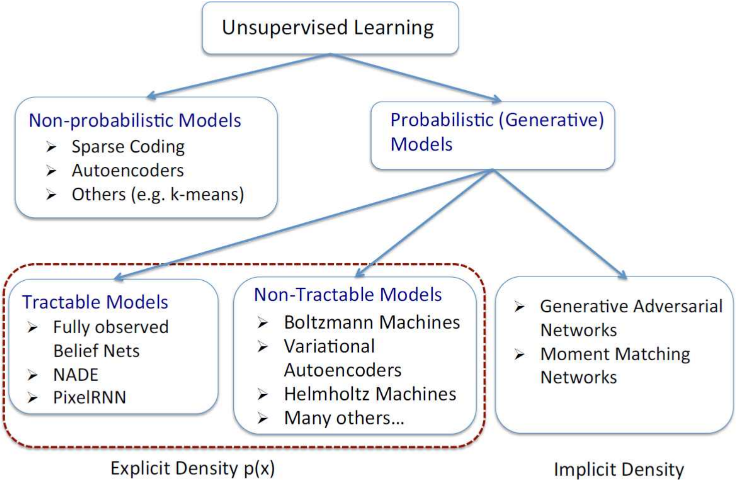


Figure: Ruslan Salakhutdinov

- To motivate Deep Neural Generative models, like before, let's seek inspiration from simple linear models first

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 $\tilde{P}(\mathbf{x})$
- Like before, we are interested in latent factors $\mathbf{h}$ that *explain* $\mathbf{x}$

Linear Factor Model

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

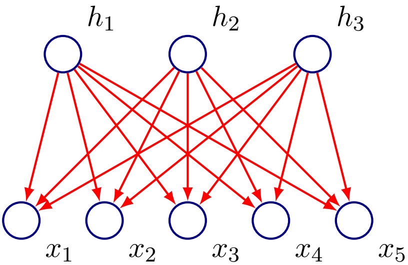
- The latent factors $\mathbf{h}$ are an *encoding* of the data
- Simplest decoding model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{x}$ with some noise
- Formally: Suppose we sample the latent factors from a distribution $\mathbf{h} \sim P(\mathbf{h})$

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

- $P(\mathbf{h})$ is a factorial distribution



$$\mathbf{x} = W\mathbf{h} + \mathbf{b} + \epsilon$$

- How do learn in such a model?
- Let's look at a simple example

Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

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

Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

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

- Now, we need to specify a noise model. Assume it comes from an isotropic Gaussian with covariance $\sigma^2 I$

Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

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

- Now, we need to specify a noise model. Assume it comes from an isotropic Gaussian with covariance $\sigma^2 I$
- For this simple model, $\mathbf{x}$ is also a multivariate Gaussian:

Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

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

- Now, we need to specify a noise model. Assume it comes from an isotropic Gaussian with covariance $\sigma^2 I$
- For this simple model, $\mathbf{x}$ is also a multivariate Gaussian:

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

Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

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

- Now, we need to specify a noise model. Assume it comes from an isotropic Gaussian with covariance $\sigma^2 I$
- For this simple model, $\mathbf{x}$ is also a multivariate Gaussian:

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

- Standard PCA: In the limit as $\sigma \to 0$

Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

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

- Now, we need to specify a noise model. Assume it comes from an isotropic Gaussian with covariance $\sigma^2 I$
- For this simple model, $\mathbf{x}$ is also a multivariate Gaussian:

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

- Standard PCA: In the limit as $\sigma \to 0$
- Gives a simple generative model for the data; can draw samples!

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:

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

- Noise is sampled from a Gaussian with a diagonal covariance:

$$\Psi = diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2])$$

Factor Analysis

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

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

- Noise is sampled from a Gaussian with a diagonal covariance:

$$\Psi = diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2])$$

- Still consider linear relationship between inputs and observed variables $\mathbf{x} \sim \mathcal{N}(\mathbf{x}; b, WW^T + \Psi)$

Factor Analysis

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

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

- Noise is sampled from a Gaussian with a diagonal covariance:

$$\Psi = diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2])$$

- Still consider linear relationship between inputs and observed variables $\mathbf{x} \sim \mathcal{N}(\mathbf{x}; b, WW^T + \Psi)$
- Already harder to analyze than PPCA

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 $\sigma \to 0$

More General Models

- Suppose $P(\mathbf{h})$ can not be assumed to have a nice Gaussian form

More General Models

- Suppose $P(\mathbf{h})$ can not be assumed to have a nice Gaussian form
- The decoding of the input from the latent states is a complicated non-linear function

More General Models

- Suppose $P(\mathbf{h})$ can not be assumed to have a nice Gaussian form
- The decoding of the input from the latent states is a complicated non-linear function
- Estimation and inference can get complicated!

More General Models

- Suppose $P(\mathbf{h})$ can not be assumed to have a nice Gaussian form
- The decoding of the input from the latent states is a complicated non-linear function
- Estimation and inference can get complicated!
- Let's look at an approach to write these problems in a general form

Energy Based Models

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

Energy Based Models

- 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

- 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

- 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

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

Energy Based Models

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

- Z is a normalizing factor called the Partition Function

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

Energy Based Models

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

- 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:

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

- We only care about the marginal:

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

Latent Variables

- $\mathbf{x}$ 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

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

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

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

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

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

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

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