CRIPAC
智能感知与计算研究中心
Center for Research on Intelligent Perception and Computing

中国科学院自动化研究所
Institute of Automation
Chinese Academy of Sciences

2023

"Deep Learning Lecture"

Lecture 7: Generative Model (1)

Yan Huang

Center for Research on Intelligent Perception and Computing (CRIPAC)

State Key Laboratory of Multimodal Artificial Intelligence Systems(MAIS)

Institute of Automation, Chinese Academy of Science (CASIA)

Outline

1. Course Review
2. Linear Factor Model
3. Autoencoder
4. DBN and RBM

Review: Regularization Strategies

- Parameter Norm Penalties
- Dataset Augmentation
- Noise Robustness
- Early Stopping
- Parameter Tying and Parameter Sharing
- Multitask Learning
- Bagging and Other Ensemble Methods
- Dropout
- Adversarial Training

Review: Parameter Norm Penalties

- This approach limits the capacity of the model by adding the penalty $\Omega(\theta)$ to the objective function resulting in:

$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

- $\alpha \in [0,1)$ is a hyper-parameter that weights the relative contribution of the norm penalty to the value of the objective function
- L2 Norm Parameter Regularization

$$\Omega(\theta) = \frac{1}{2}\|w\|_2^2 = \frac{1}{2}w^T w$$

- L1 Norm Parameter Regularization

$$\Omega(\theta) = ||\mathbf{w}|| = \sum_i |w_i|$$

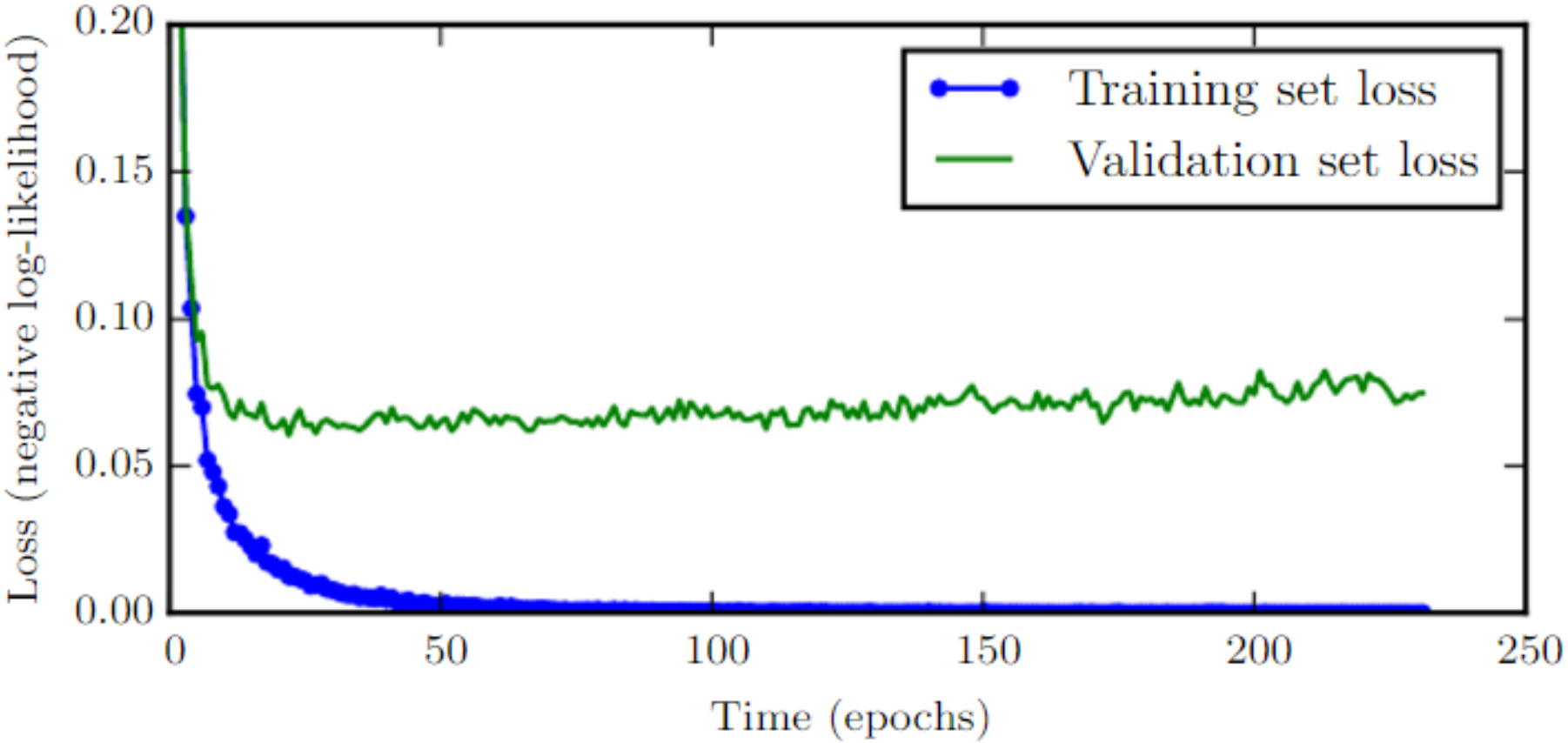
Review: Dataset Augmentation

Color jitter

Horizontal flipping

Relative performance improvement is limited!

Review: Early Stopping



Review: Parameter Sharing and Parameter Tying

- **Parameter Sharing** imposes much stronger assumptions on parameters through forcing the parameter sets to be equal
- Examples would be Siamese networks, convolution operators, and multitask learning

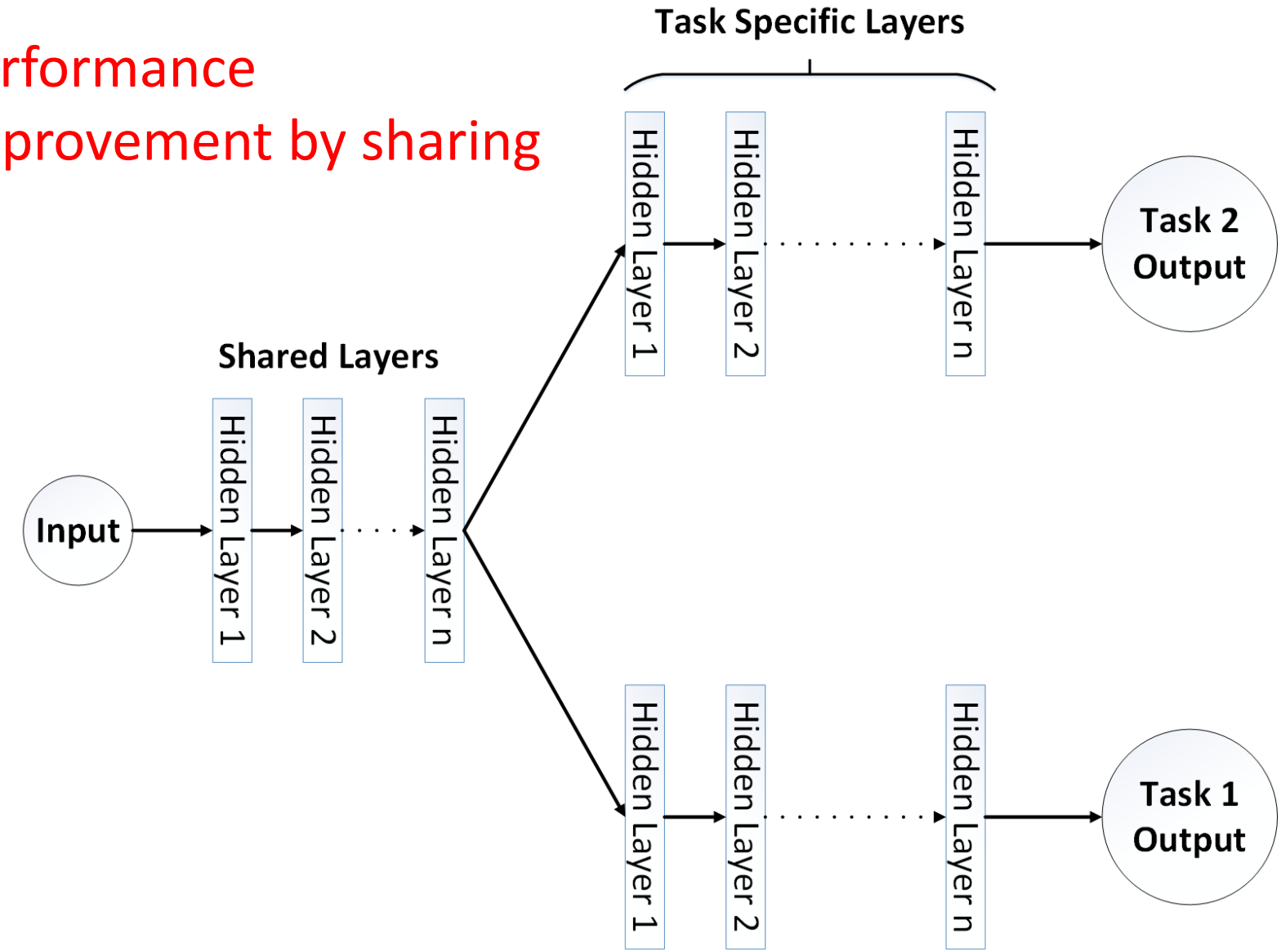
- **Parameter Tying** refers to explicitly forcing the parameters of two models to be close to each other, through the norm penalty:

$$||\mathbf{w}^{(A)} - \mathbf{w}^{(B)}||$$

- Here, $w^{(A)}$ refers to the weights of the first model while $w^{(B)}$ refers to those of the second one

Review: Multitask Learning

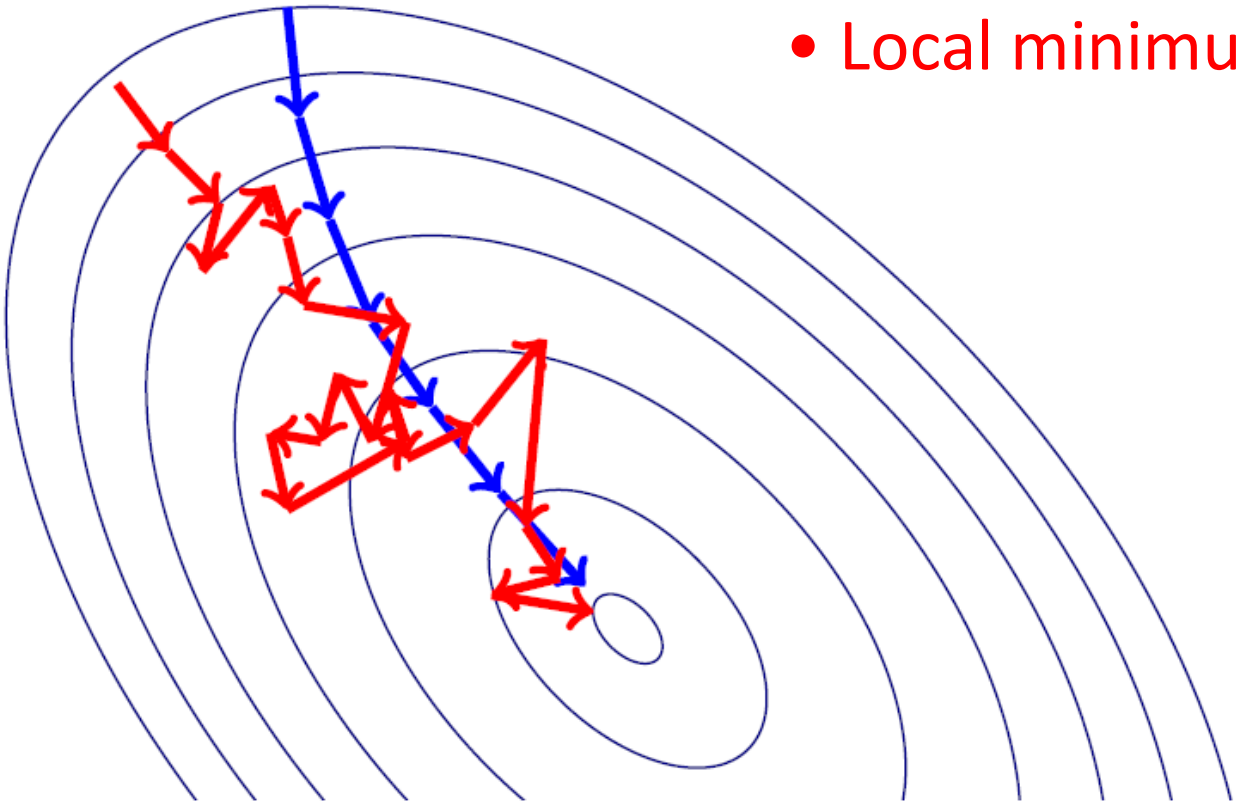
Performance improvement by sharing



Review: Optimization

- Stochastic Gradient Descent
- Momentum Method and the Nesterov Variant
- Adaptive Learning Methods (AdaGrad, RMSProp, Adam)
- Batch Normalization
- Initialization Heuristics

Review: SGD Vs BGD



- Slow convergence
- Local minimum

Review: Comparison

SGD: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \mathbf{v}$

Nesterov: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)}) \right)$ then $\theta \leftarrow \theta + \mathbf{v}$

AdaGrad: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ then $\theta \leftarrow \theta + \Delta\theta$

RMSProp: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

Adam: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$ then $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ then $\theta \leftarrow \theta + \Delta\theta$

Review: Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Why 0-Mean?

"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Ioffe and Szegedy 2015

Outline

1. Course Review
2. Linear Factor Model
3. Autoencoder
4. DBN and RBM

Probabilistic Modeling

- We want to build a probabilistic model of the input $P(\mathbf{x})$
- Like before, we are interested in latent factors $\mathbf{h}$ that explain $\mathbf{x}$
- We then care about the marginal:

$$P(\mathbf{x}) = E_{\boldsymbol{h}} P(\mathbf{x}|\mathbf{h})$$

- The latent factor $\mathbf{h}$ is an encoding of the data

Unsupervised Learning ⟷ Probabilistic Modeling

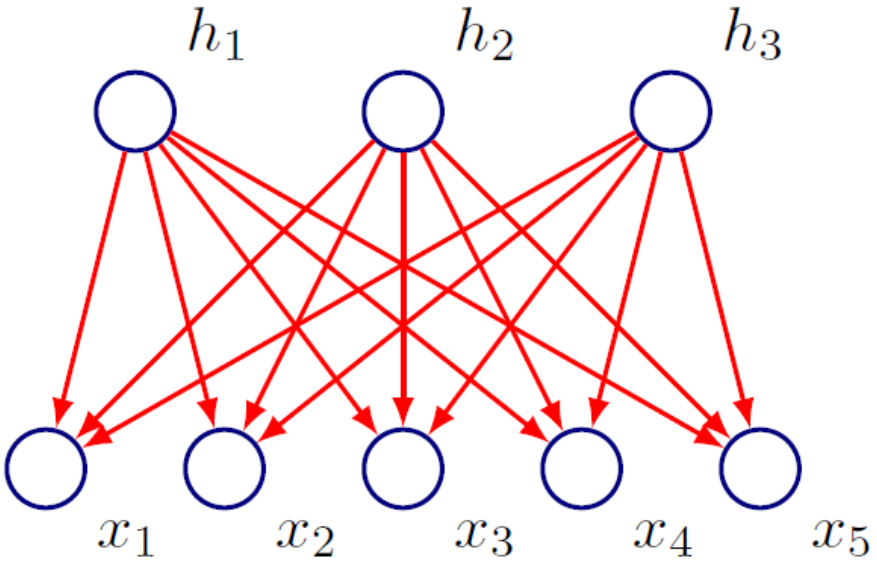
Linear Factor Model

- Simplest probabilistic model: Get $\mathbf{x}$ after a linear transformation of $\mathbf{h}$ with some noise

- Formally: Suppose we sample the latent factors from a factorial distribution $\mathbf{h} \sim P(\mathbf{h}) = \prod P(\mathbf{h}_i)$

- Then: $\mathbf{x} = W\mathbf{h} + \mathbf{b} + \varepsilon$

Linear Factor Model



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

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

Probabilistic PCA

- Suppose underlying latent factor has a Gaussian distribution

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

- For the noise model: assume $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$
- Then:

$$P(\mathbf{x}|\mathbf{h}) = \mathcal{N}(\mathbf{x}|W\mathbf{h} + \mathbf{b}, \sigma^2 I)$$

- We care about the marginal P(x) (predictive distribution):

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, WW^T + \sigma^2 I)$$

Probabilistic PCA

$$P(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{b}, WW^T + \sigma^2 I)$$

- How do we learn the parameters? (EM, ML Estimation)
- Let's look at the ML Estimation:
  - Let $C = WW^T + \sigma^2 I$
  - We want to maximize $\ell(\theta; X) = \sum_i \log P(\mathbf{x}_i|\theta)$

Probabilistic PCA: ML Estimation

$$\ell(\theta; X) = \sum_i \log P(\mathbf{x}_i|\theta)$$
$$= -\frac{N}{2}\log|C| - \frac{1}{2}\sum_i (\mathbf{x}_i - \mathbf{b})C^{-1}(\mathbf{x}_i - \mathbf{b})^T$$
$$= -\frac{N}{2}\log|C| - \frac{1}{2}Tr[\ C^{-1}\sum_i (\mathbf{x}_i - \mathbf{b})(\mathbf{x}_i - \mathbf{b})^T]$$
$$= \frac{N}{2}\log|C| - \frac{1}{2}Tr[\ C^{-1}S]$$

- S is the variance matrix: $S = \frac{1}{N}\sum(x_i - b)\,(x_i - b)^T$
- Now tune the parameters θ= W, b, σ to maximize log-likelihood
- Can also use EM

Details can be found in: https://blog.csdn.net/janehong1314/article/details/84918269

Factor Analysis

- Fix the latent factor prior to be the unit Gaussian as before:

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

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

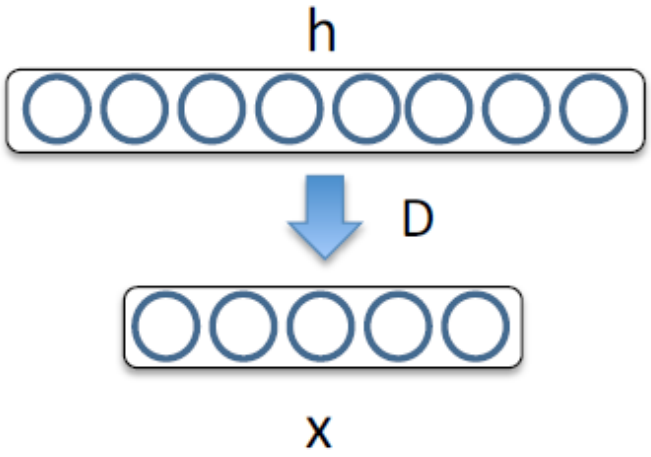
$$\Psi = diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_d^2])$$

- Still consider linear relationship between inputs and observed variables: Marginal

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

Sparse Coding

- Sparse coding is 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 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 reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

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

Sparsity vs. reconstruction control

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

Reconstruction error

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 reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

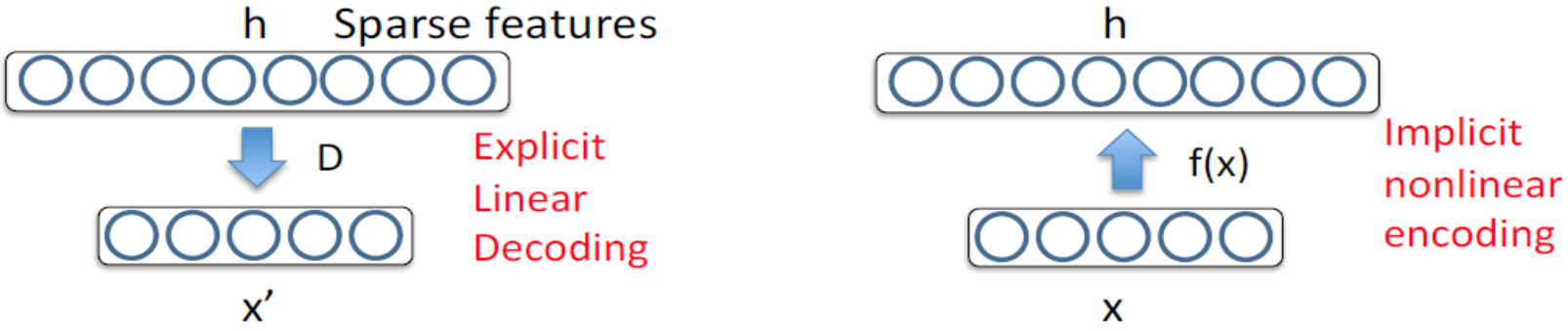
$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

  - we also constrain the columns of $\mathbf{D}$ to be of norm 1
  - otherwise, $\mathbf{D}$ could grow big while $\mathbf{h}$ becomes small to satisfy the L1 constraint

Interpreting Sparse Coding

Interpreting Sparse Coding

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

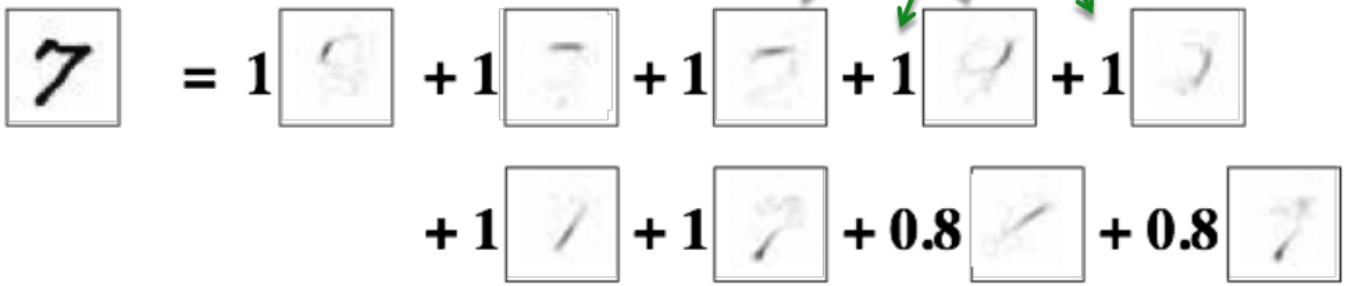


- Sparse, over-complete representation $\mathbf{h}$
- Encoding $\mathbf{h} = f(\mathbf{x})$ is implicit and nonlinear function of $\mathbf{x}$
- Reconstruction (or decoding) $\mathbf{x}' = \mathbf{Dh}$ is linear and explicit

Interpreting Sparse Coding

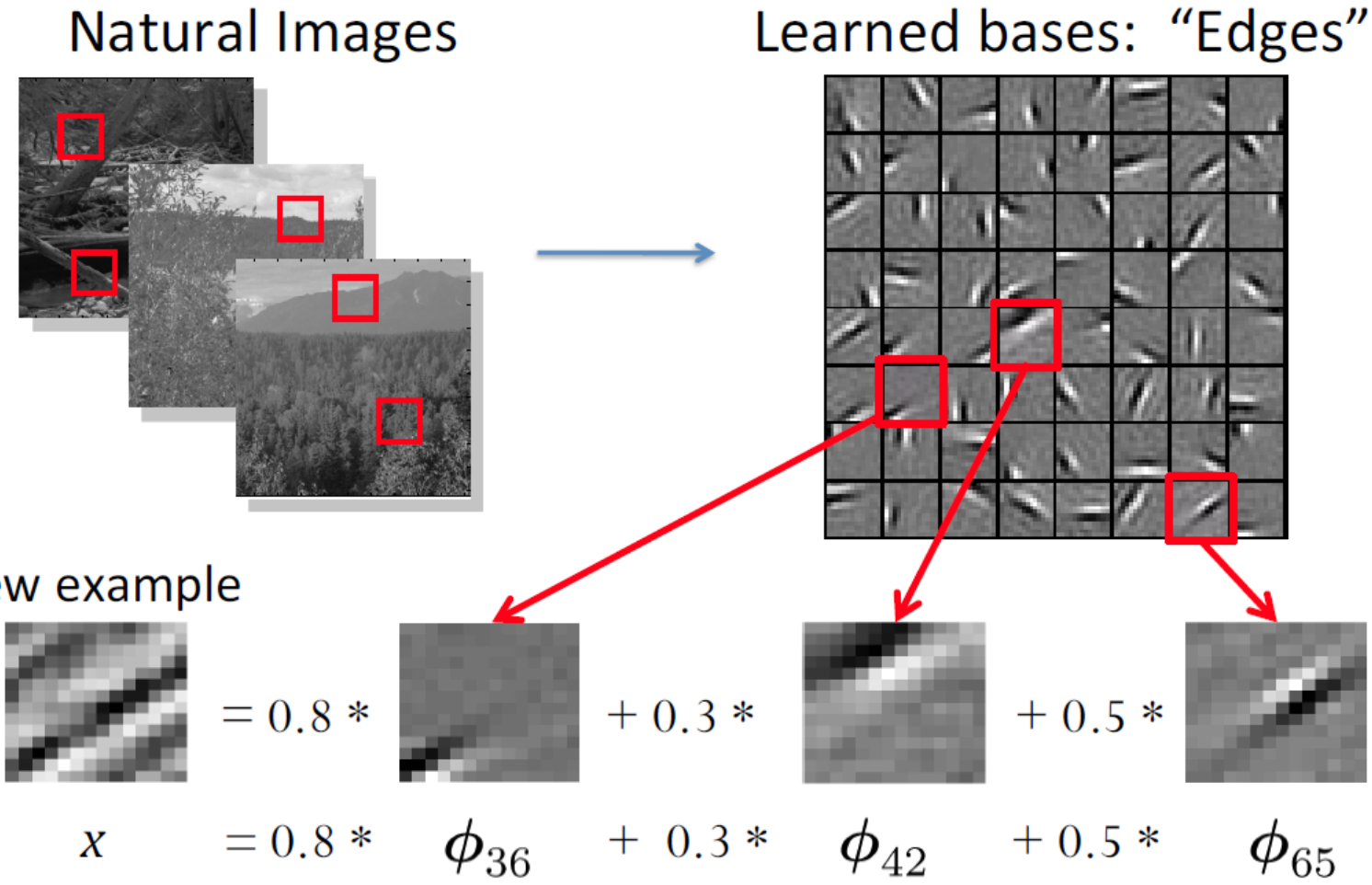
- We can also write:

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



- $\mathbf{D}$ is often referred to as Dictionary
- In certain applications, we know what dictionary matrix to use, while in many cases, we have to learn it

Interpreting Sparse Coding



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

Inference

- Given dictionary $\mathbf{D}$, how do we compute $\mathbf{h}(\mathbf{x}^{(t)})$
- We need to optimize:

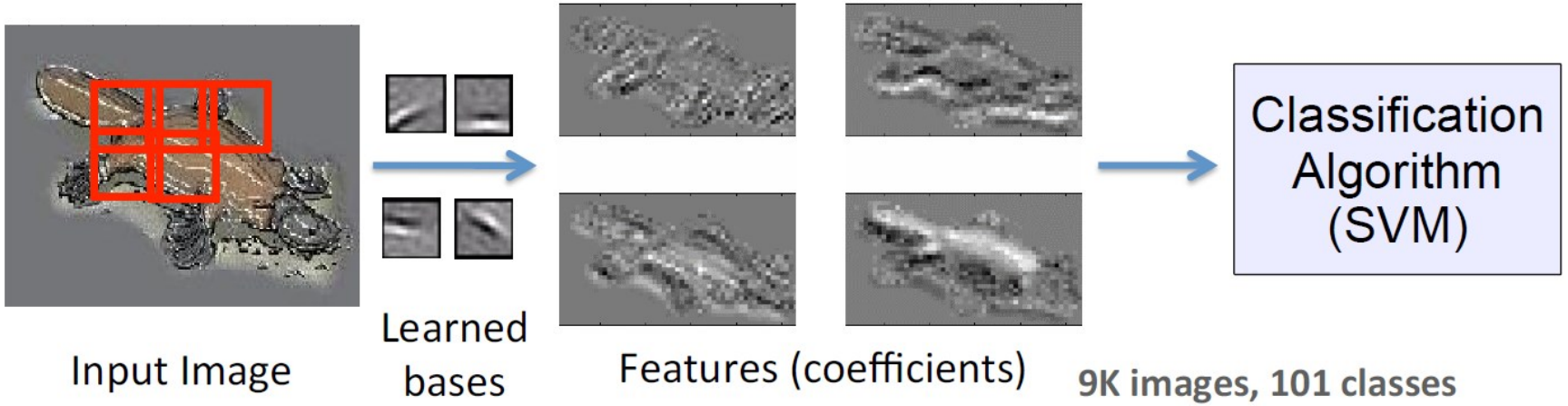
$$l(\mathbf{x}^{(t)}) = \frac{1}{2}||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda||\mathbf{h}^{(t)}||_1$$

- We could use a gradient descent method:

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

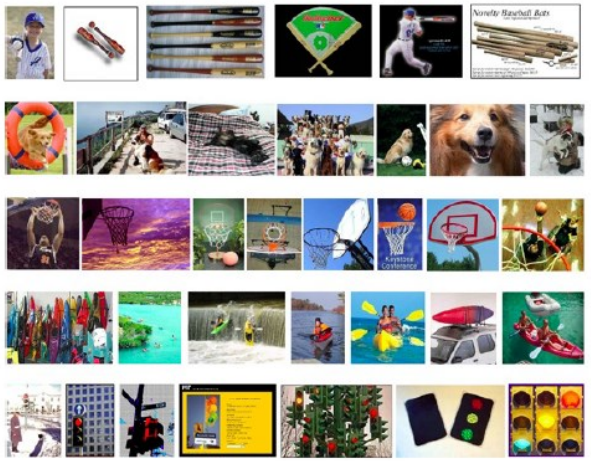
Image Classification

- Evaluated on Caltech101 object category dataset



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

9K images, 101 classes



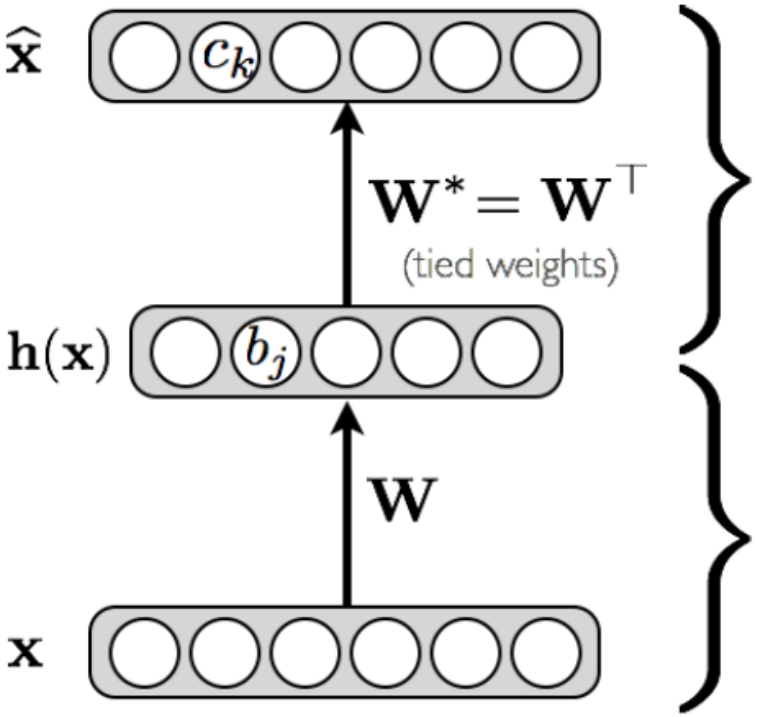
Lee et al., NIPS 2006

Outline

1. Course Review
2. Linear Factor Model
3. Autoencoder
4. DBN and RBM

Autoencoder

- Feed-forward neural network is trained to reproduce its input at the output layer



Decoder

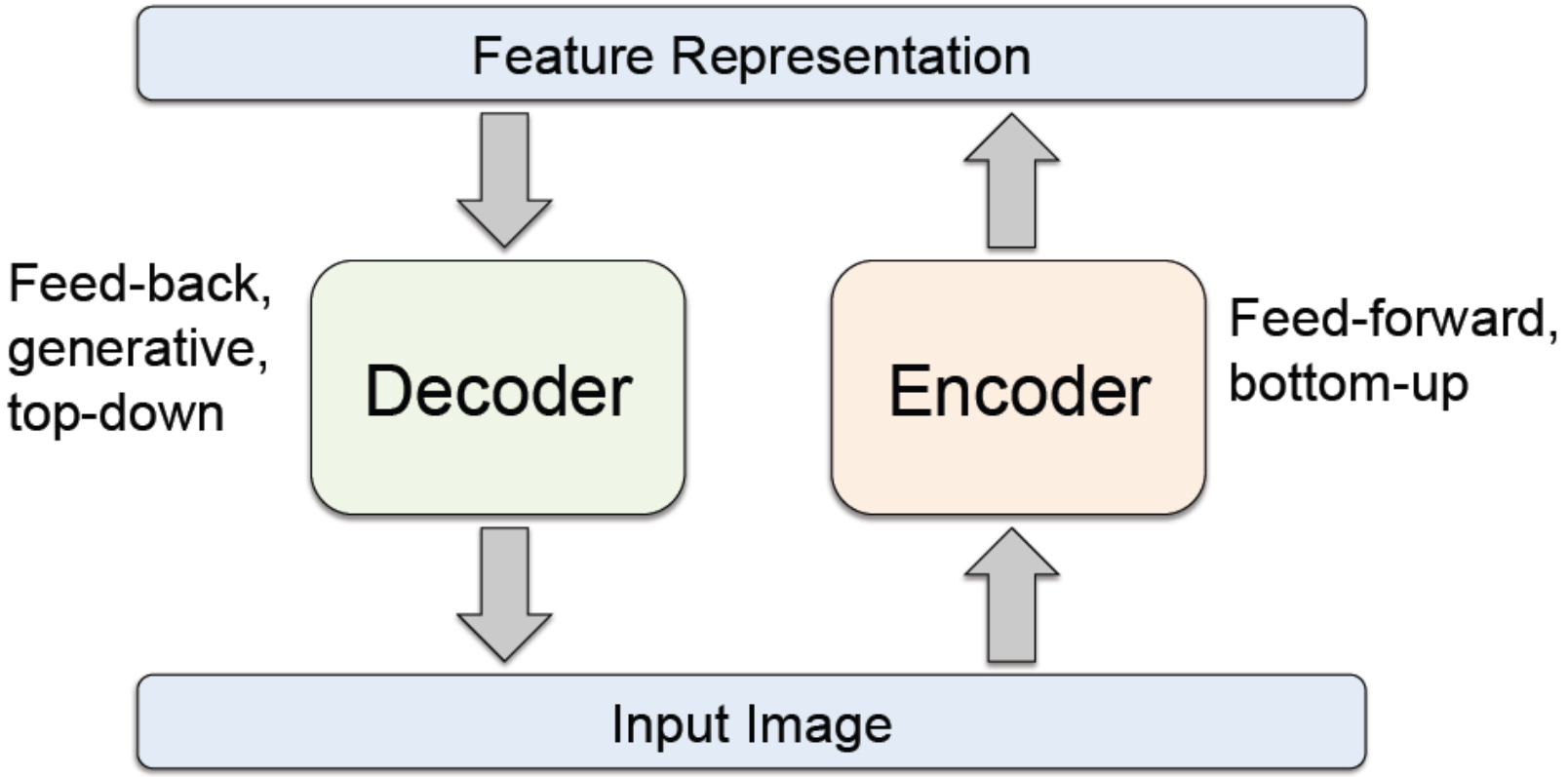
$$\hat{\mathbf{x}} = o(\hat{\mathbf{a}}(\mathbf{x})) = \underbrace{\text{sigm}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))}$$

For binary units

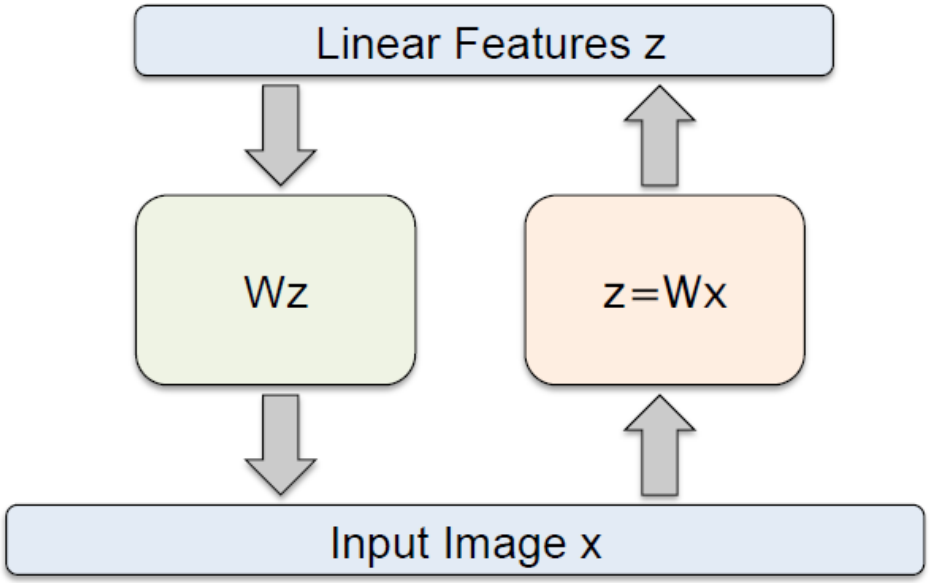
Encoder

$$\mathbf{h}(\mathbf{x}) = g(\mathbf{a}(\mathbf{x})) = \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$$

Autoencoder (Another View)



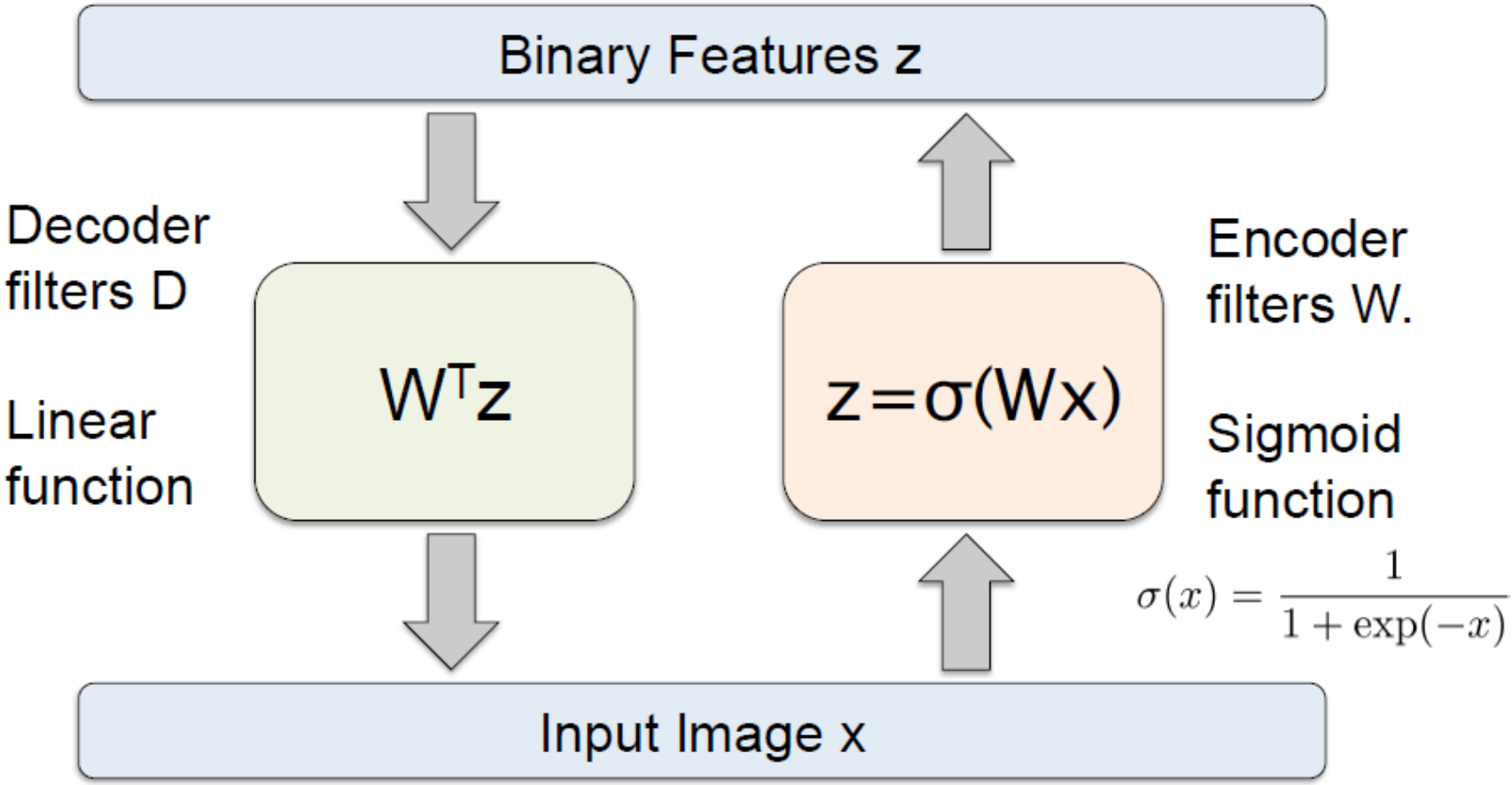
Linear Autoencoder & PCA



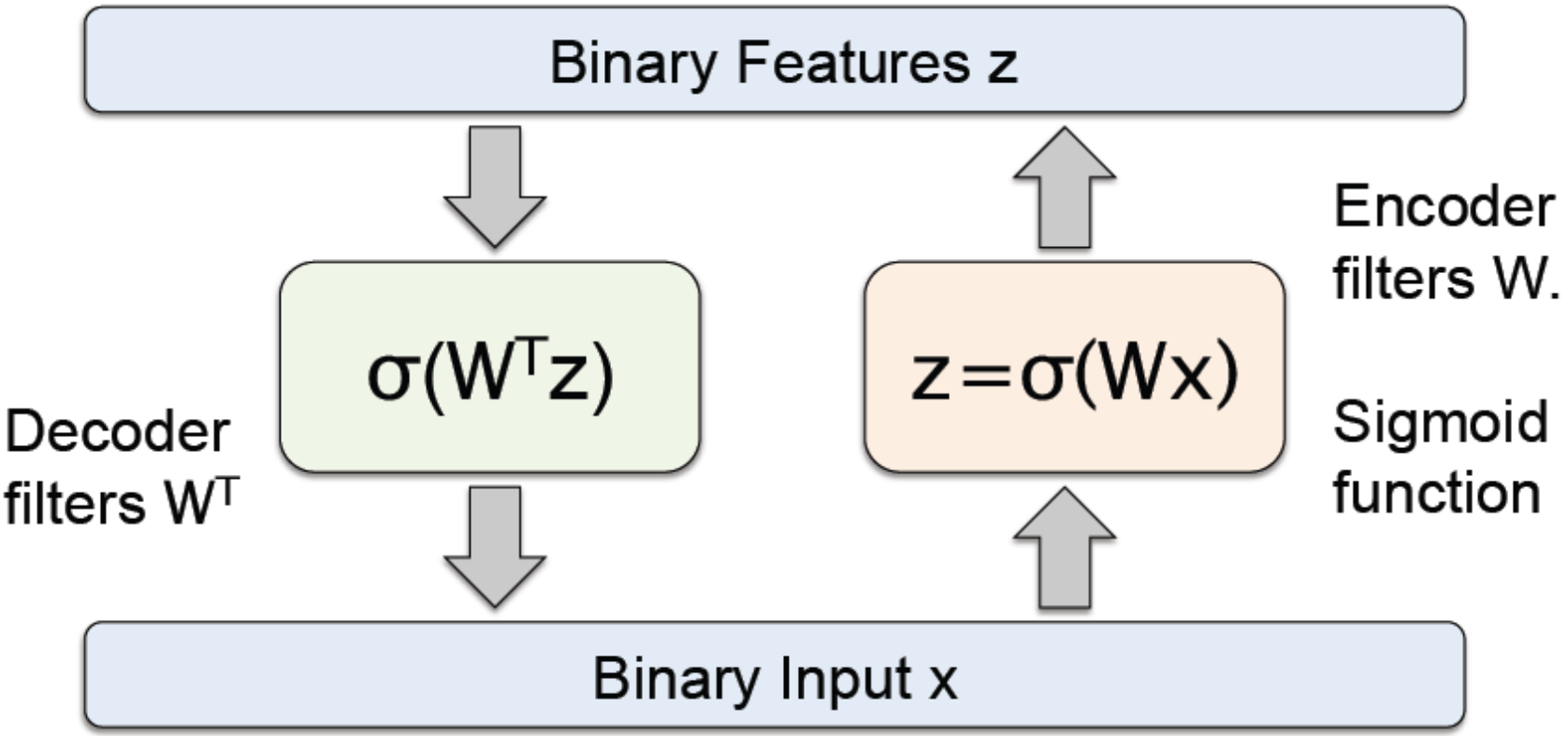
- If the hidden and output layers are linear, it will learn hidden units that are a linear function of the data and minimize the squared error.
- The K hidden units will span the same space as the first k principal components. The weight vectors may not be orthogonal.

- With nonlinear hidden units, we have a nonlinear generalization of PCA

Autoencoder (Real-valued Input)

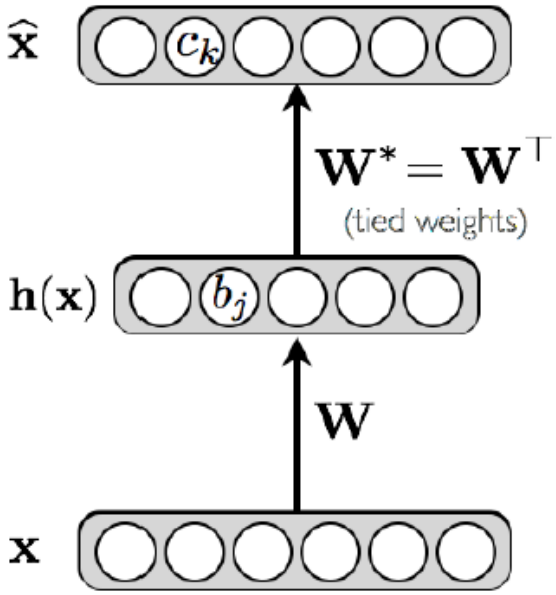


Autoencoder (Binary Input)

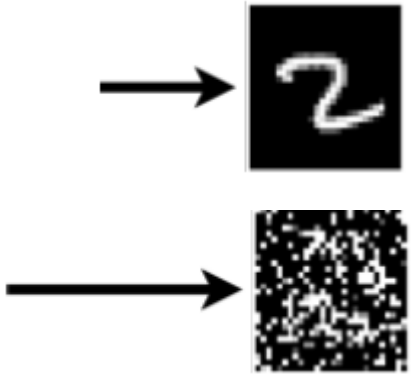


Undercomplete Representation

- Hidden layer is undercomplete if smaller than the input layer (bottleneck layer, e.g. dimensionality reduction):
  - hidden layer "compresses" the input
  - will compress well only for the training distribution

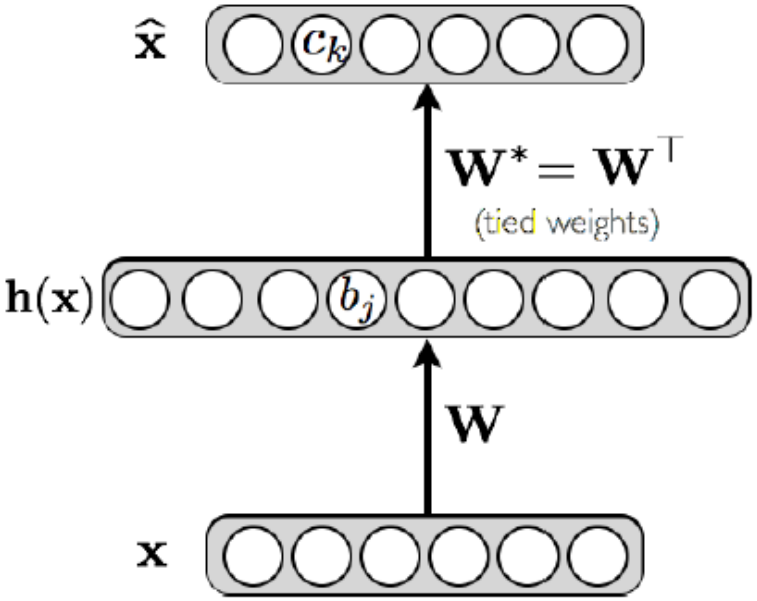


- Hidden units will be
  - good features for the training distribution
  - will not be robust to other types of input



Overcomplete Representation

- Hidden layer is overcomplete if greater than the input layer
  - no compression in hidden layer
  - each hidden unit could copy a different input component
- No guarantee that the hidden units will extract meaningful structure



Loss Function

- Loss function for binary inputs

$$l(f(\mathbf{x})) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k) \log(1 - \widehat{x}_k)\right)$$

  - Cross-entropy error function (reconstruction loss) $f(\mathbf{x}) \equiv \widehat{\mathbf{x}}$

- Loss function for real-valued inputs

$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (\widehat{x}_k - x_k)^2$$

  - Sum of squared differences (reconstruction loss)
  - We use a linear activation function at the output

Loss Function

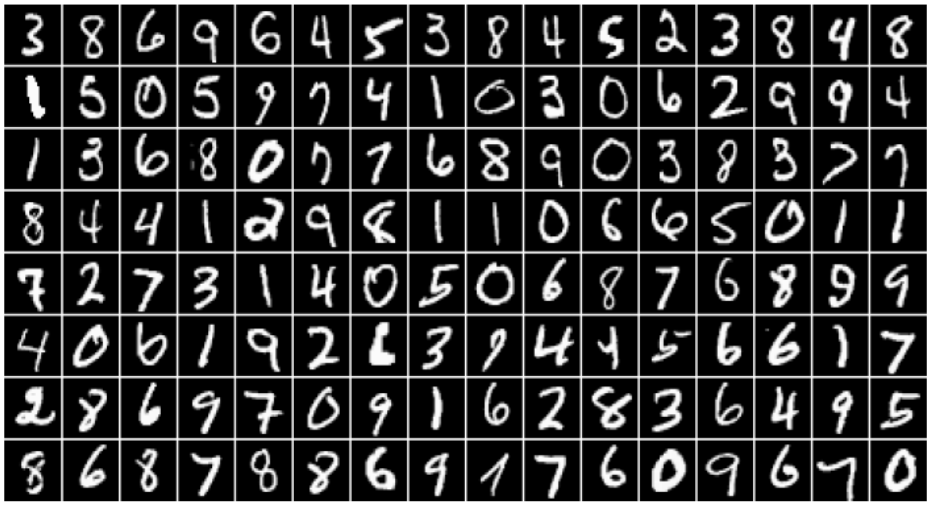
- For both cases, the gradient $\nabla_{\widehat{\mathbf{a}}(\mathbf{x}^{(t)})} l(f(\mathbf{x}^{(t)}))$ has a very simple form:

$$\nabla_{\widehat{\mathbf{a}}(\mathbf{x}^{(t)})} l(f(\mathbf{x}^{(t)})) = \widehat{\mathbf{x}}^{(t)} - \mathbf{x}^{(t)} \qquad f(\mathbf{x}) \equiv \widehat{\mathbf{x}}$$

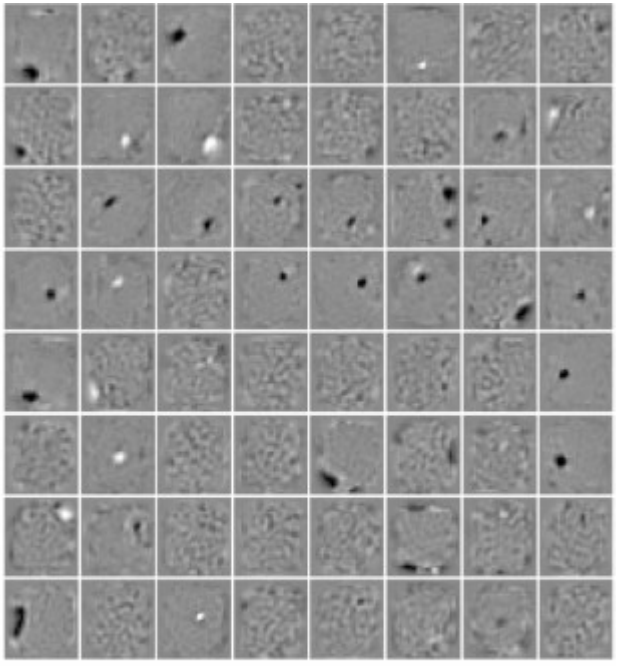
- Parameter gradients are obtained by backpropagating the gradient $\nabla_{\widehat{\mathbf{a}}(\mathbf{x}^{(t)})} l(f(\mathbf{x}^{(t)}))$ like in a regular network

  - important: when using tied weights ($\mathbf{W}^* = \mathbf{W}^\top$), $\nabla_{\mathbf{W}} l(f(\mathbf{x}^{(t)}))$ is the sum of two gradients
  - this is because $\mathbf{W}$ is present in the encoder and in the decoder

Learned Features

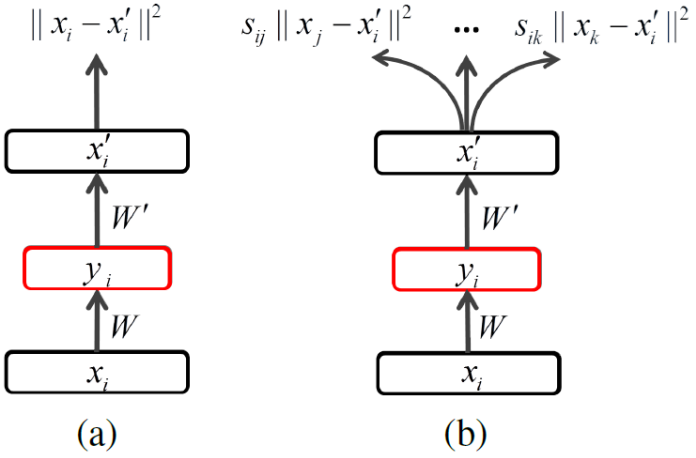


MNIST dataset

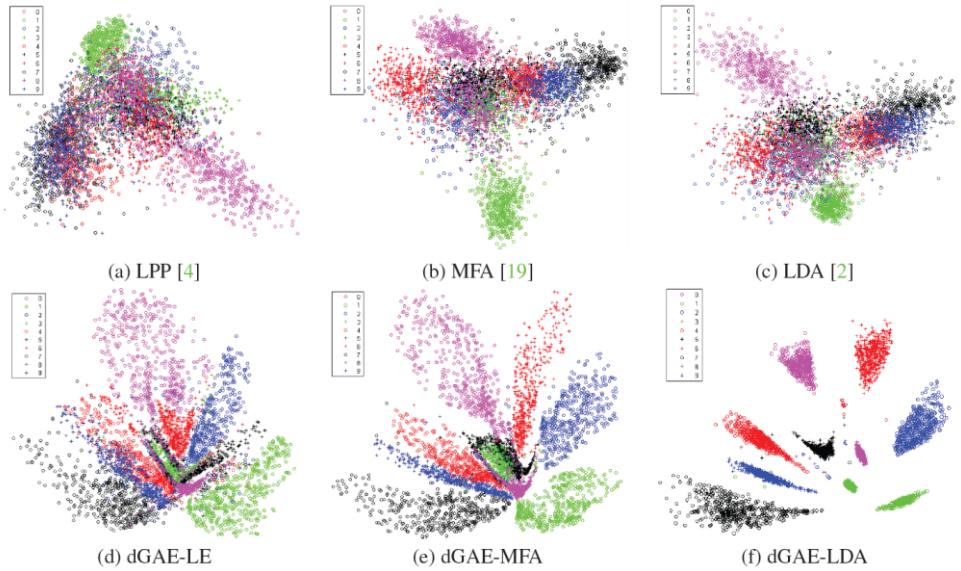


Autoenncoder

Generalized Autoencoder



(a) Traditional autoencoder
(b) Generalized autoencoder



2D visualization of the learned digit image manifold

Method	Reconstruction Set	Reconstruction Weight
GAE-PCA	$j = i$	$s_{ij} = 1$
GAE-LDA	$j \in \Omega_{c_i}$	$s_{ij} = \frac{1}{n_{c_i}}$
GAE-ISOMAP	$j : x_j \in X$	$s_{ij} \in S = -H\Lambda H/2$
GAE-LLE	$j \in N_k(i)$,	$s_{ij} = (M + M^T - M^T M)_{ij}$ if $i \neq j$;
	$j \in (N_k(m) \cup m), j \neq i$ if $\forall m, i \in N_k(m)$	0 otherwise
GAE-LE	$j \in N_k(i)$	$s_{ij} = \exp\{-\|x_i - x_j\|^2/t\}$
GAE-MFA	$j \in \Omega_{k_1}(c_i)$,	$s_{ij} = 1$
	$j \in \Omega_{k_2}(\bar{c}_i)$	$s_{ij} = -1$

Generalized Autoencoder

ICCV2013: 223 rejected

R3: "ICCV may not be the most appropriate conference to publish this paper because of the lack of real computer vision applications"

R3: "For deep neural networks, in general quite a bit of tuning is needed in order to get good results because of the existence of many local optima…"

R3&5 "it is more interesting to discuss the proposed generalized autoencoder method in the nonlinear case"

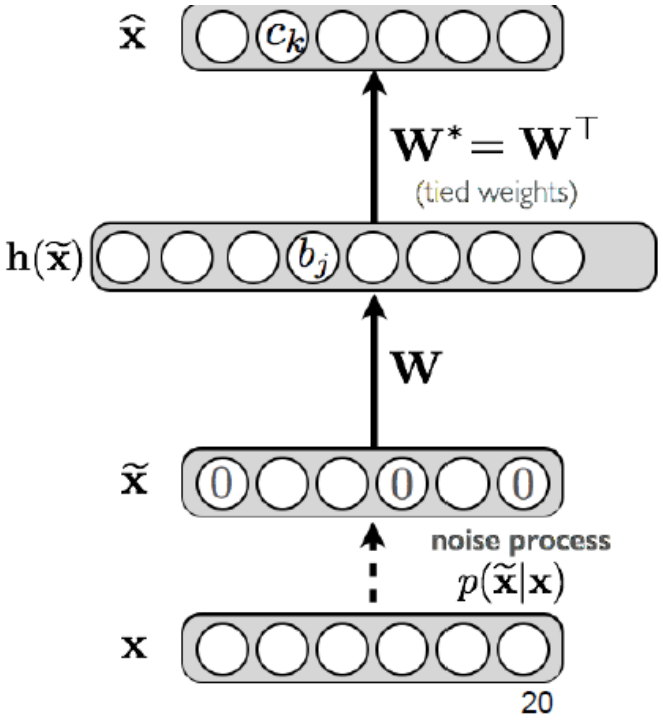
R6: "how about the performance comparison between the dGAEs and original autoencoder?"

Denoising Autoencoder

- Idea: representation should be robust to introduction of noise:
  - Random assignment of subset inputs to 0, with probability
  - Similar to dropouts on the input layer
  - Gaussian additive noise

• Reconstruction $\widehat{\mathbf{x}}$ computed from the corrupted input $\widetilde{\mathbf{x}}$

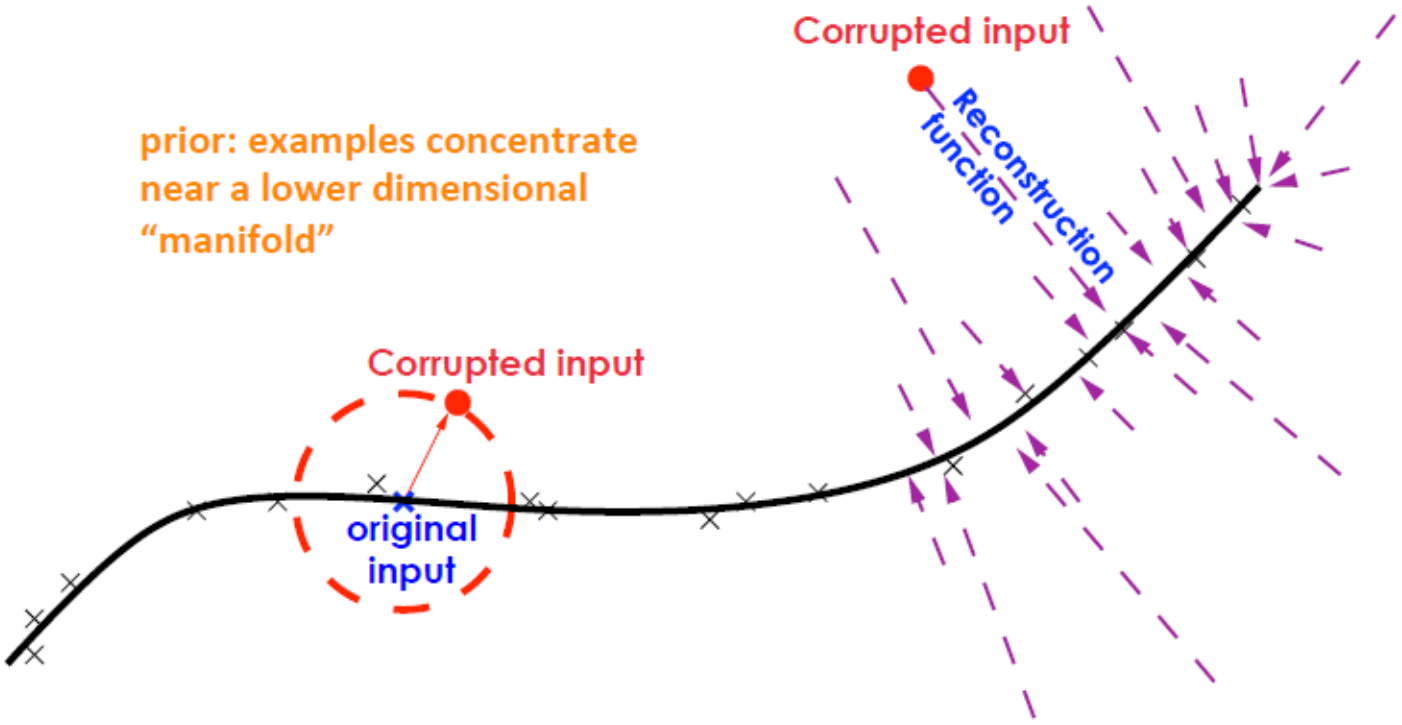
• Loss function compares $\widehat{\mathbf{x}}$ reconstruction with the noiseless input $\mathbf{x}$



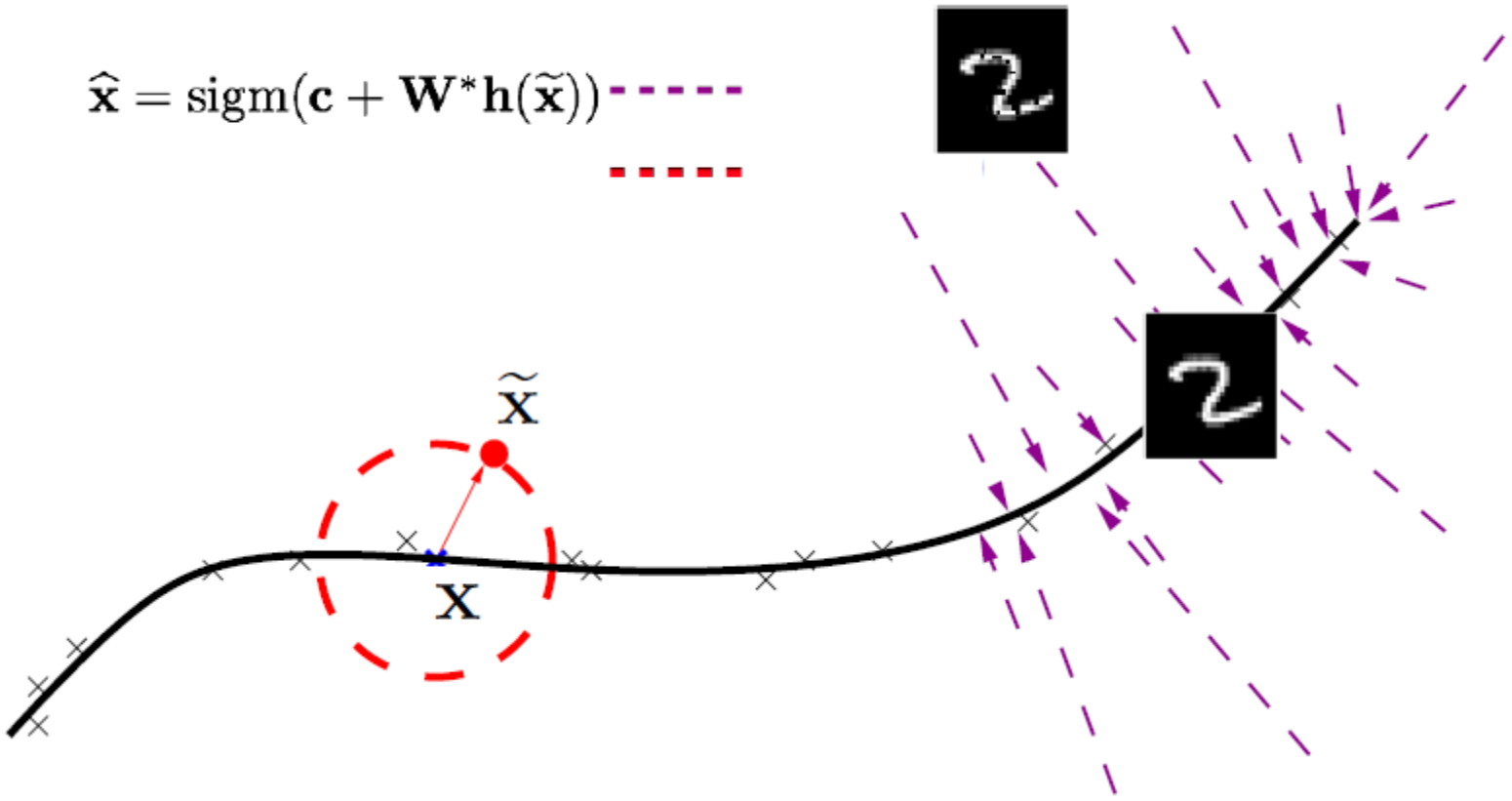
(Vincent et.al., ICML 2008)

Denoising Autoencoder

- The learner must capture the structure of the input distribution in order to optimally undo the effect of the corruption process
- Learn a reconstruction function that corresponds to a vector field pointing towards high-density regions (the manifold where examples concentrate)

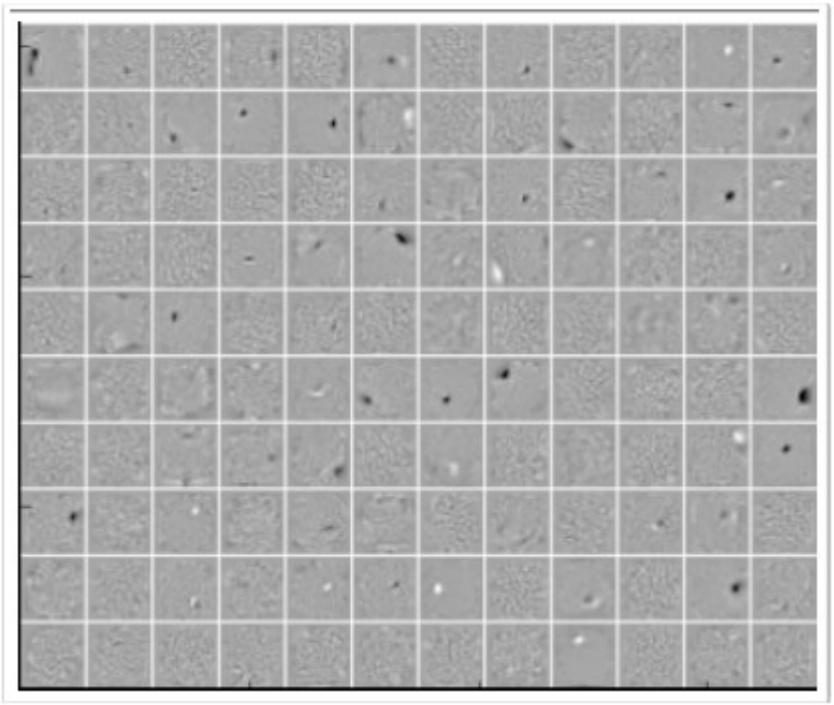


Denoising Autoencoder

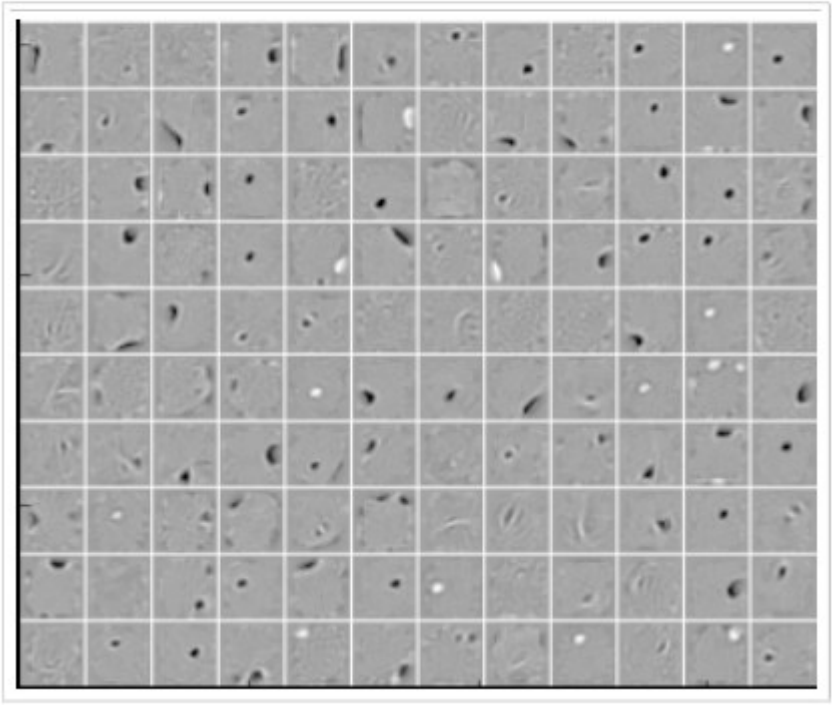


Learned Filters

Non-corrupted

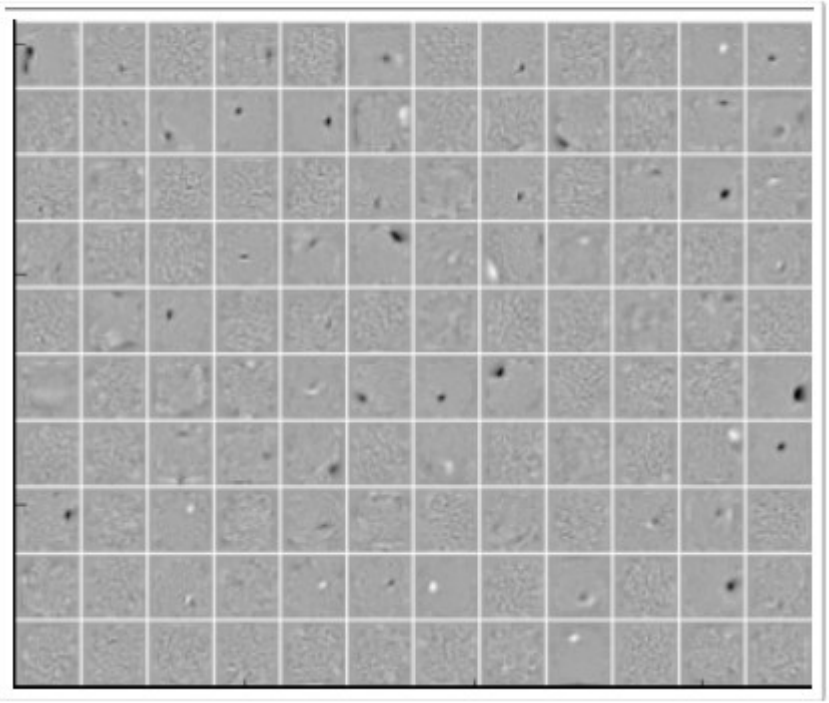


25% corrupted input

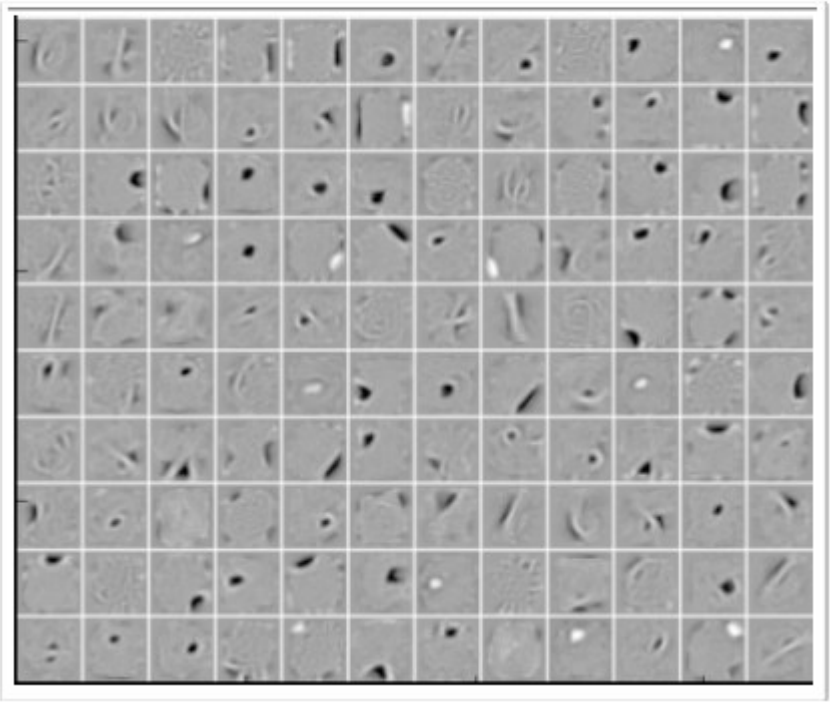


Learned Filters

Non-corrupted

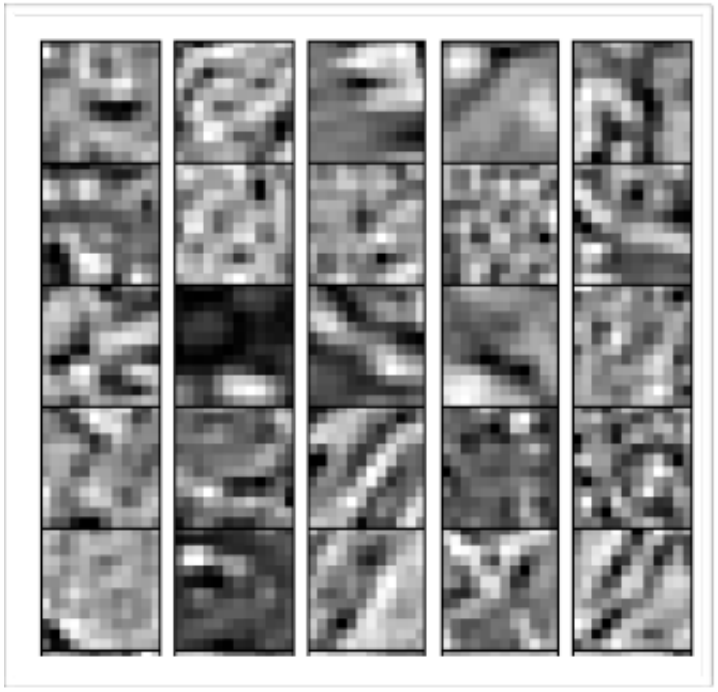


50% corrupted input

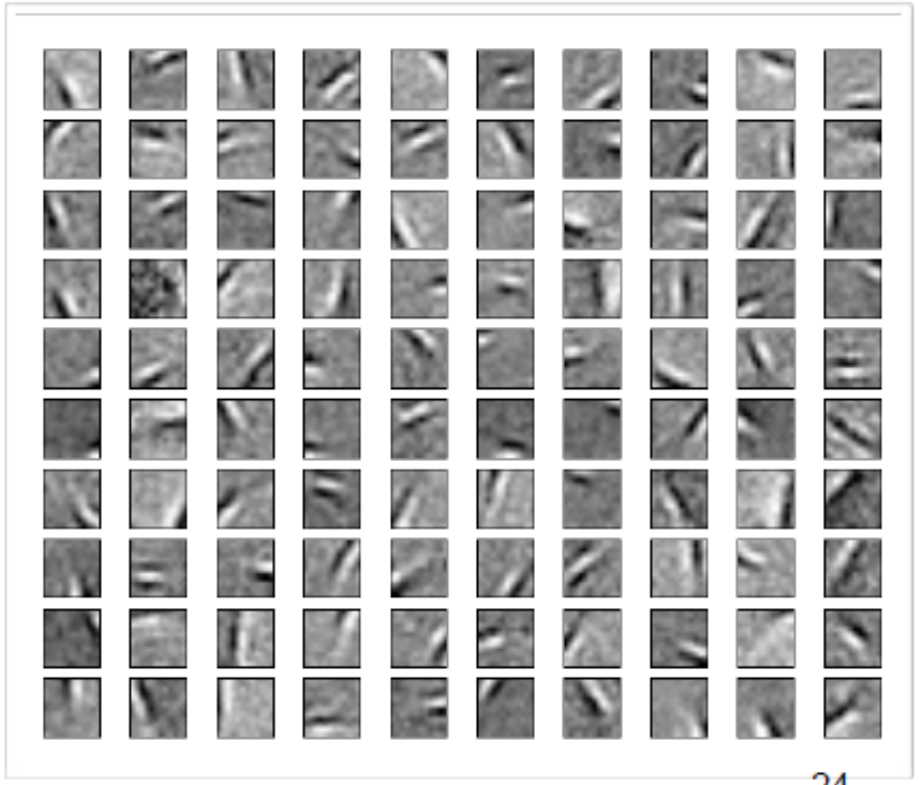


Squared Error Loss

- Training on natural image patches, with squared loss



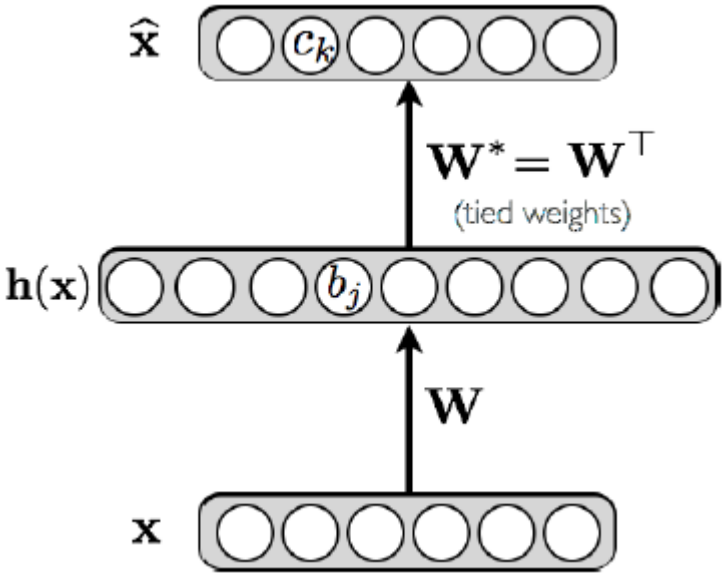
Data



Filters

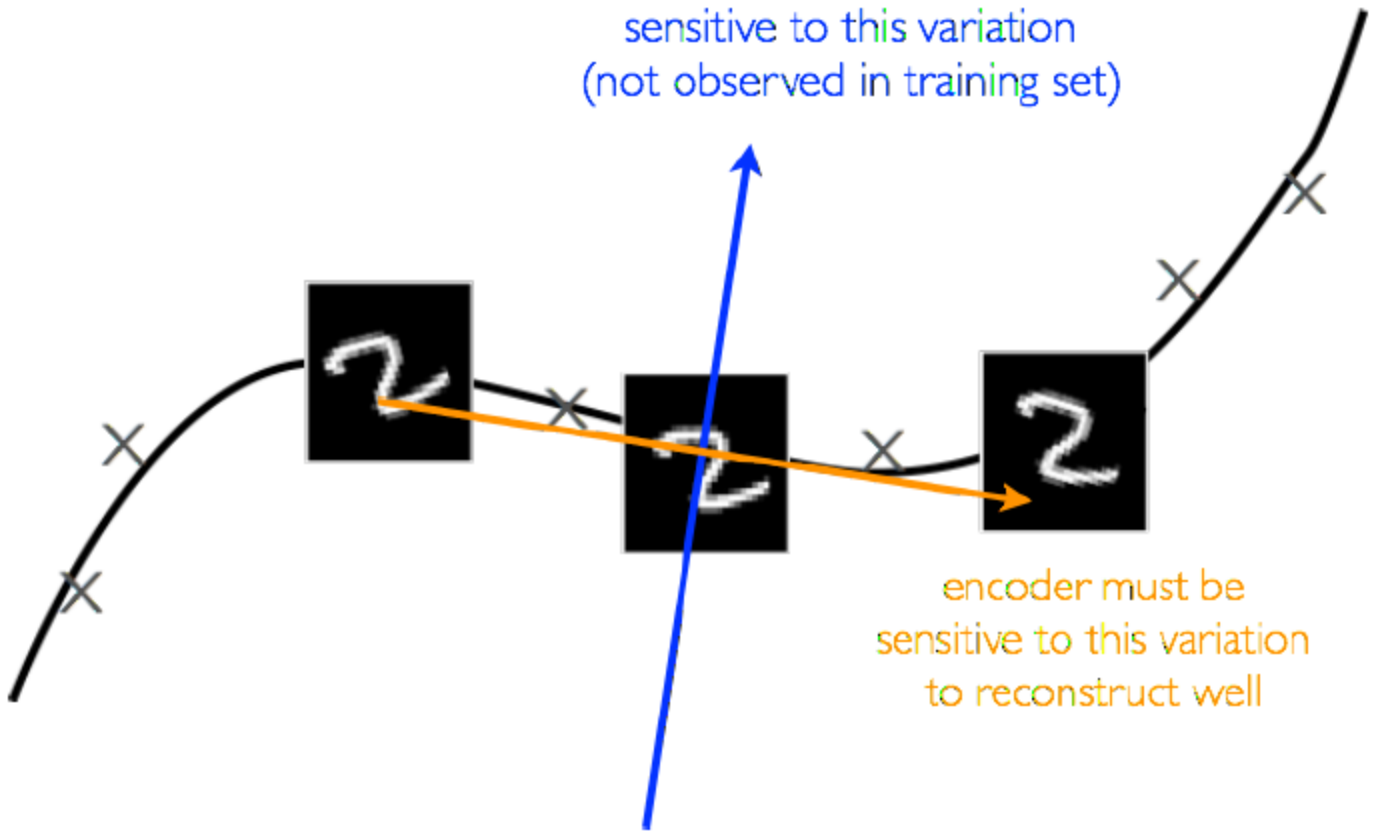
Contractive Autoencoder

- Alternative approach to avoid uninteresting solutions
  - add an explicit term in the loss that penalizes that solution
- We wish to extract features that only reflect variations observed in the training set
  - invariant to the other variations



Contractive Autoencoder

- Illustration:



Contractive Autoencoder

- Consider the following loss function:

$$\underbrace{l(f(\mathbf{x}^{(t)}))}_{\text{Reconstruction Loss}} + \lambda \underbrace{||\nabla_{\mathbf{x}^{(t)}} \mathbf{h}(\mathbf{x}^{(t)})||_F^2}_{\text{Jacobian of Encoder}}$$

- For the binary observations:

$$l(f(\mathbf{x}^{(t)})) = -\sum_k \left(x_k^{(t)} \log(\widehat{x}_k^{(t)}) + (1 - x_k^{(t)}) \log(1 - \widehat{x}_k^{(t)}) \right)$$

$$||\nabla_{\mathbf{x}^{(t)}} \mathbf{h}(\mathbf{x}^{(t)})||_F^2 = \sum_j \sum_k \left(\frac{\partial h(\mathbf{x}^{(t)})_j}{\partial x_k^{(t)}} \right)^2$$

Autoencoder attempts to preserve all information

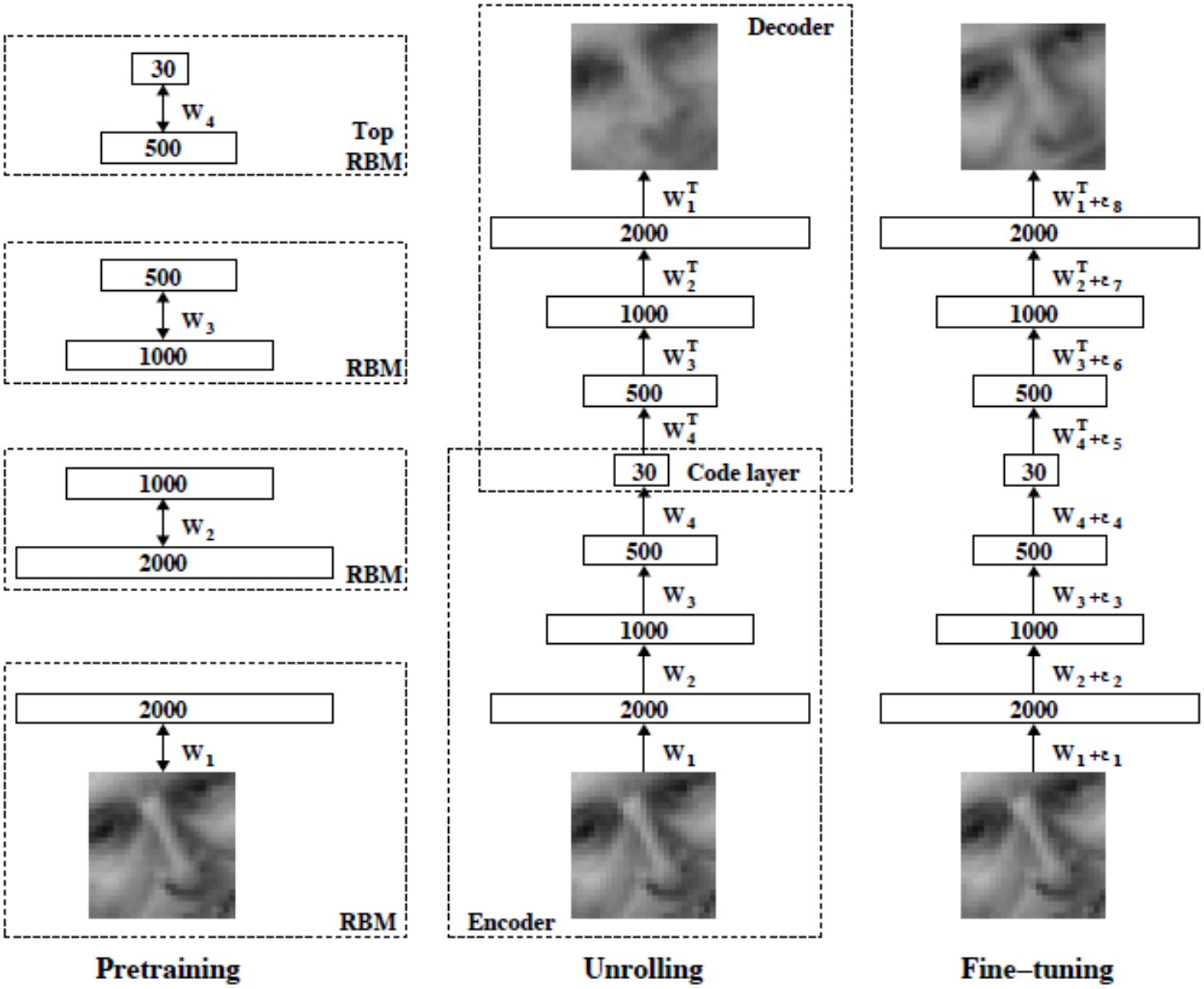
Encoder throws away all information

Pros and Cons

- Advantage of denoising autoencoder: simpler to implement
  - require adding one or two lines of code to regular autoencoder
  - no need to compute Jacobian of hidden layer

- Advantage of contractive autoencoder: gradient is deterministic
  - can use second order optimizers (conjugate gradient, LBFGS, etc.)
  - might be more stable than denoising autoencoder, which uses a sampled gradient

Deep Autoencoder



Deep Autoencoder

- Use 25x25 – 2000 – 1000 – 500 – 30 autoencoder to extract 30-D real-valued codes for Olivetti face patches

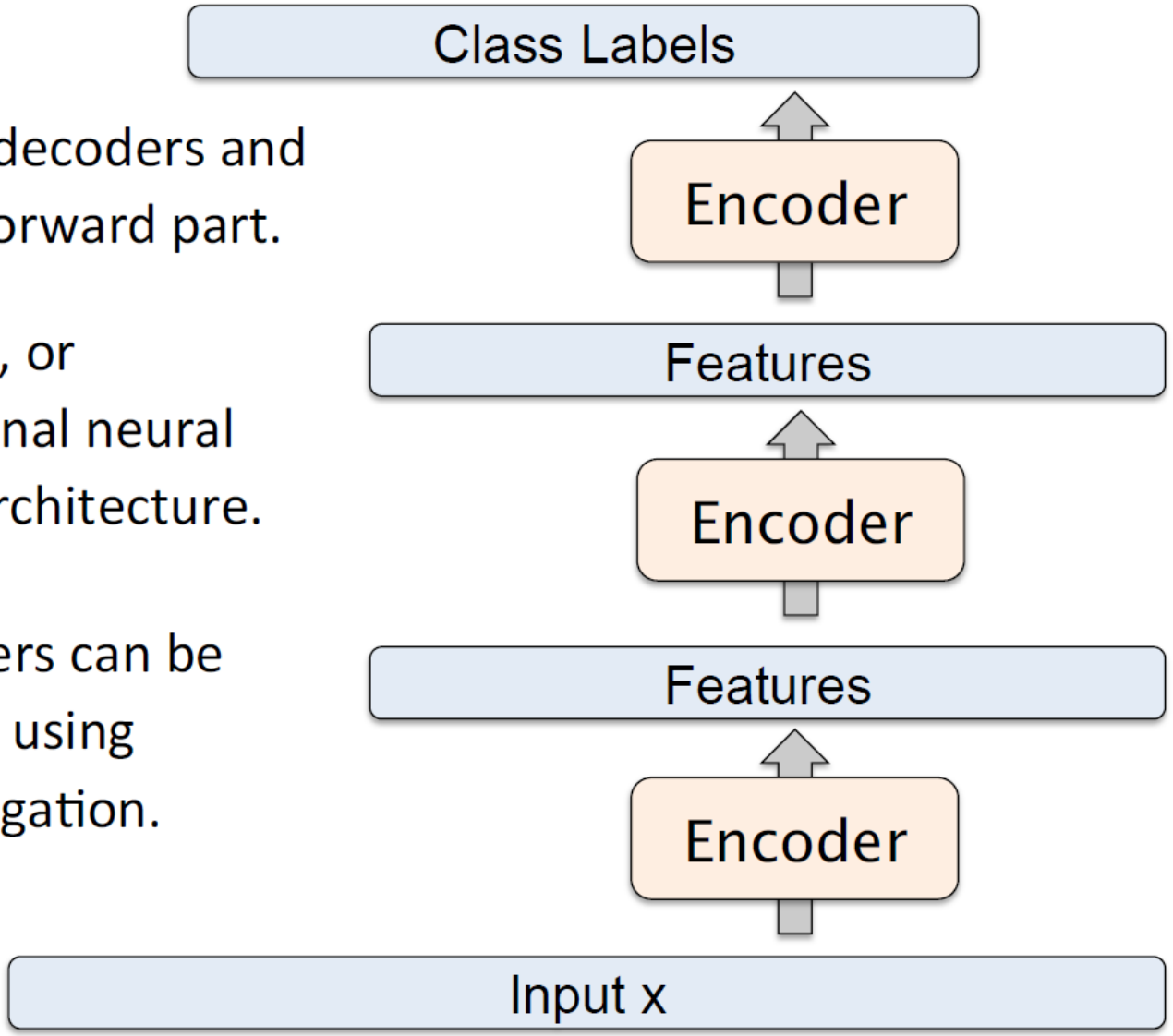
- **Top**: Random samples from the test dataset
- **Middle**: Reconstructions by the 30-dimensional deep autoencoder
- **Bottom**: Reconstructions by the 30-dimensional PCA

Deep Autoencoder

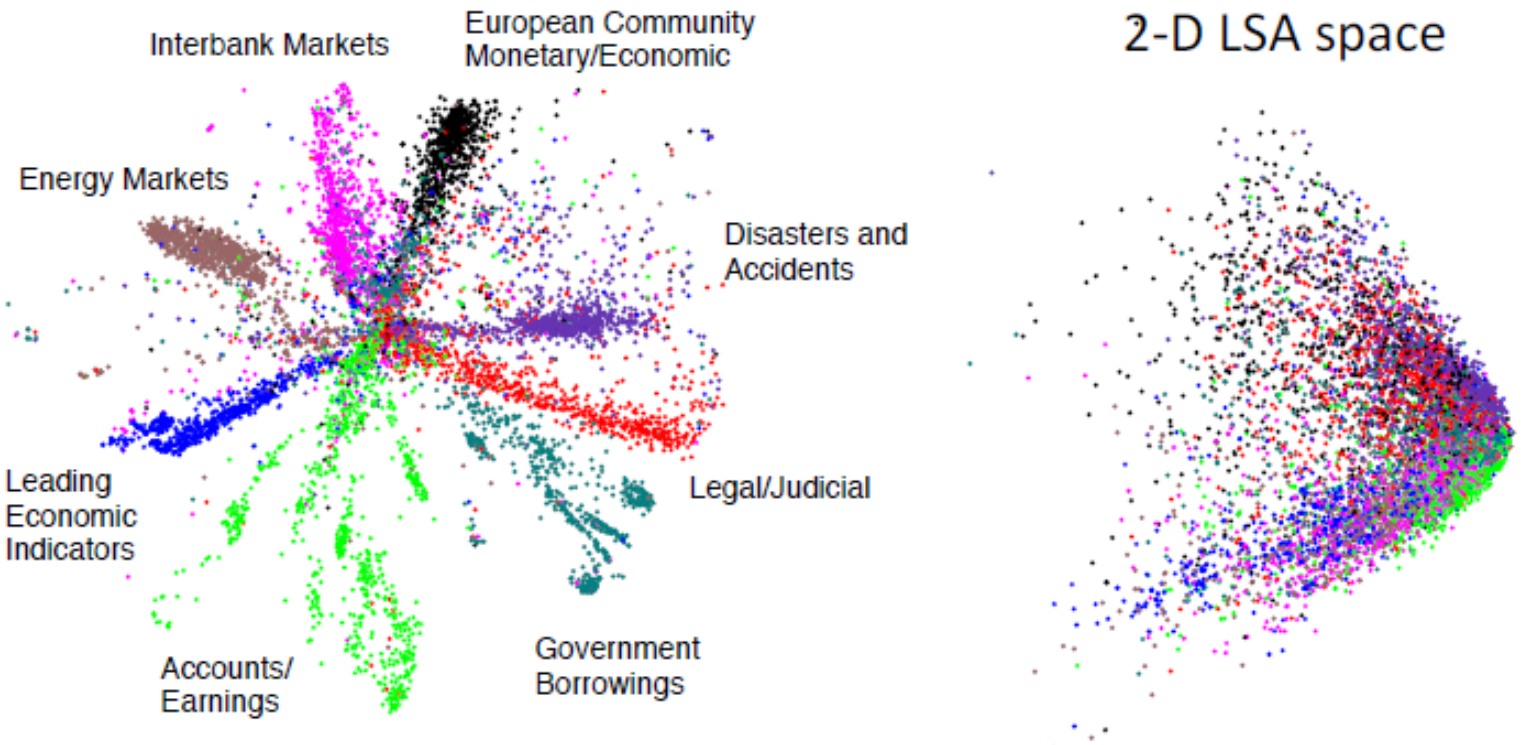
- Very difficult to optimize deep autoencoders using backpropagation
- Pre-training + fine-tuning
  - First train a stack of autoencoders
  - Then “unroll” them
  - Then fine-tune with backpropagation

Deep Autoencoder

- Remove decoders and use feed-forward part.
- Standard, or convolutional neural network architecture.
- Parameters can be fine-tuned using backpropagation.

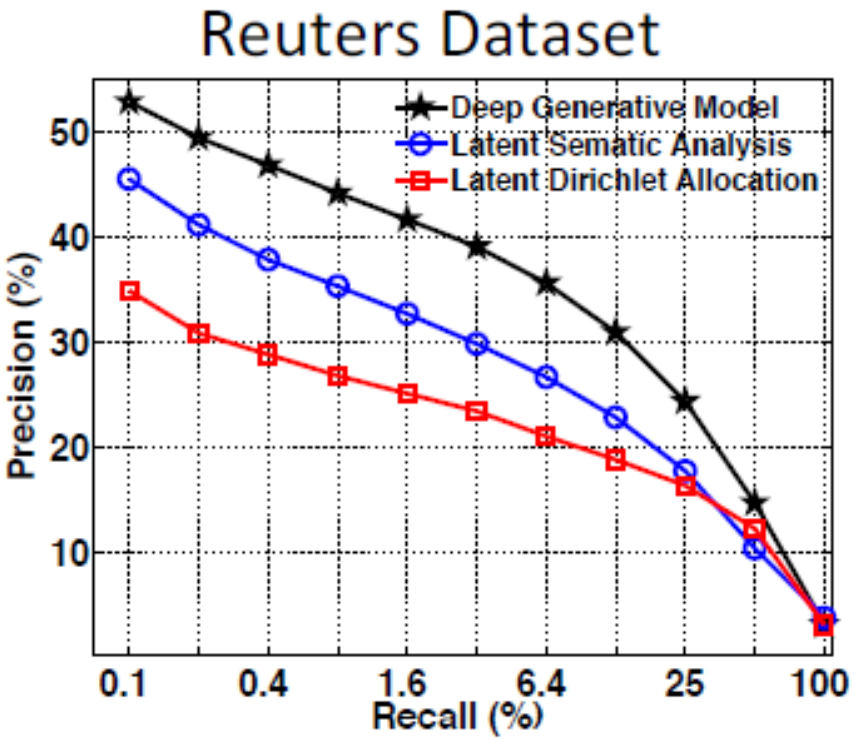


Information Retrieval



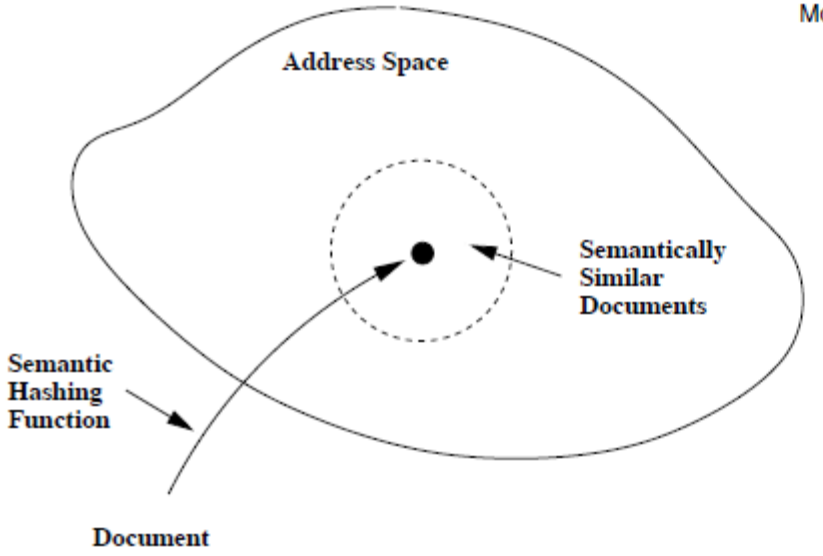
- The Reuters Corpus Volume II contains 804,414 newswire stories (randomly split into **402,207 training and 402,207 test)**
- "Bag-of-words" representation: each article is represented as a vector containing the counts of the most frequently used 2000 words in the training set

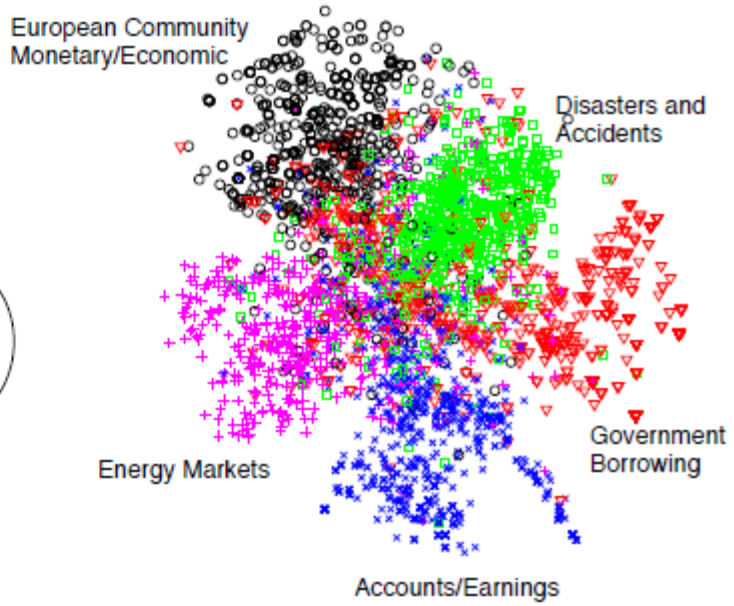
Information Retrieval



- Reuters dataset: 804,414 newswire stories
- Deep generative model significantly outperforms LSA and LDA topic models

Semantic Hashing

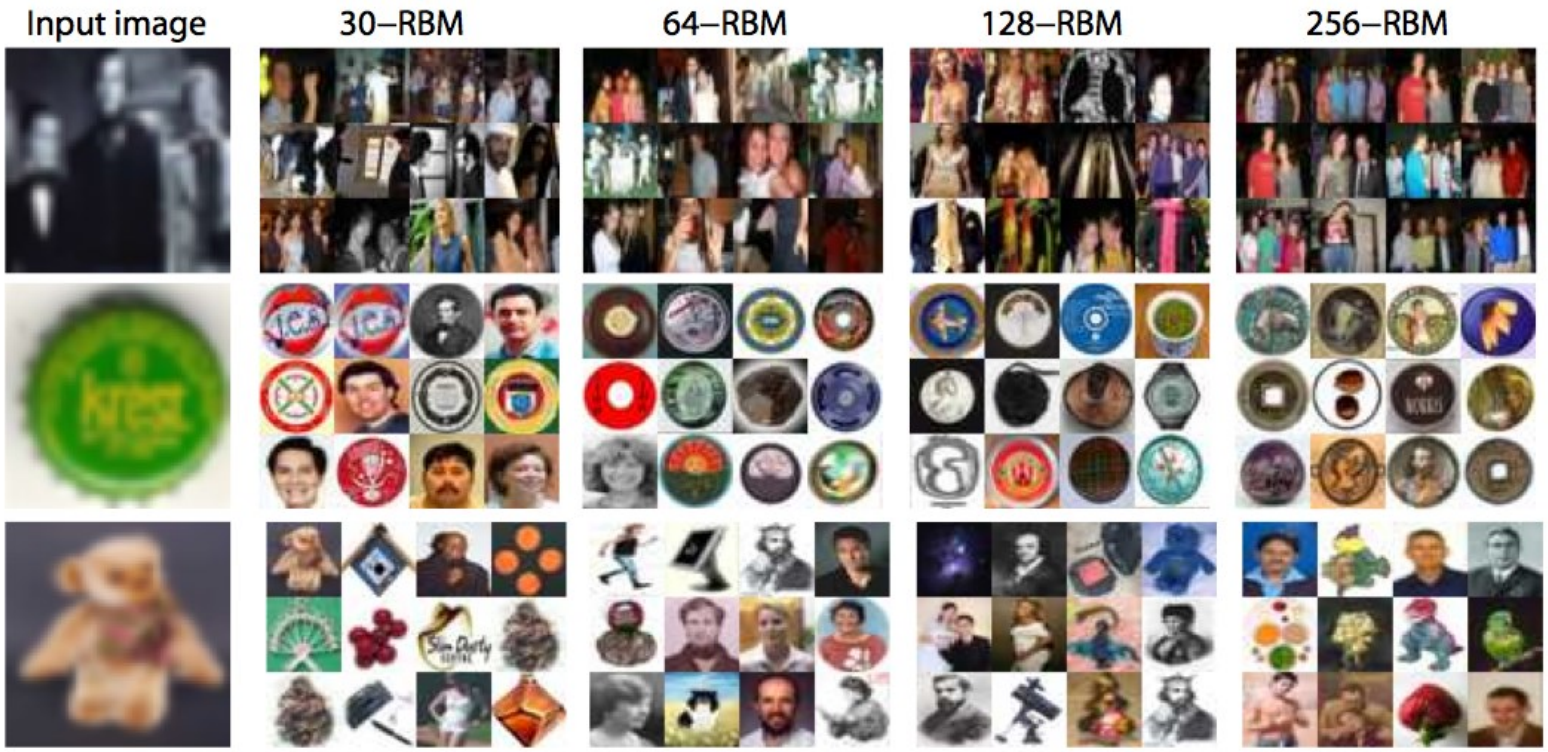




- Learn to map documents into **semantic 20-D binary codes**
- Retrieve similar documents stored at the nearby addresses **with no search at all**

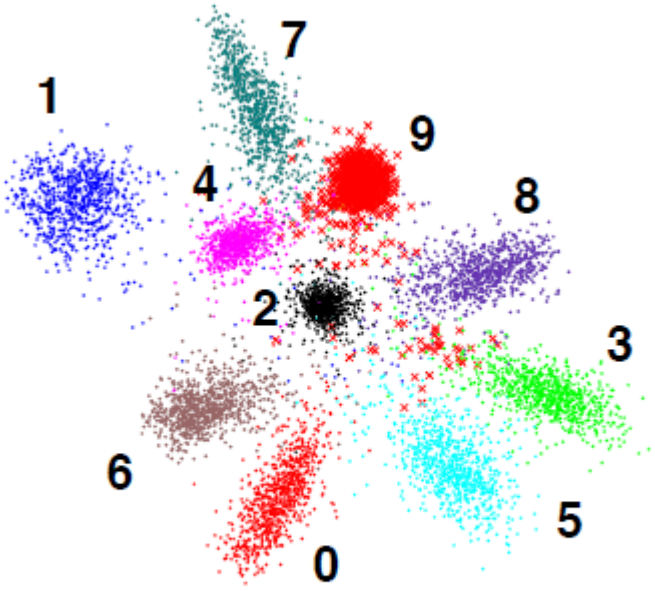
Searching Image Database Using Binary Codes

- Map images into binary codes for fast retrieval

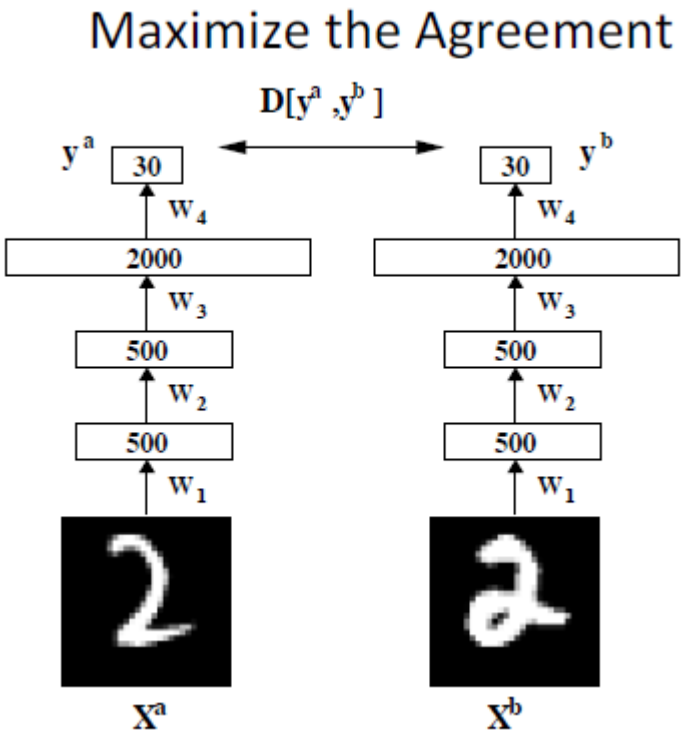


- Small Codes, Torralba, Fergus, Weiss, CVPR 2008
- Spectral Hashing, Y. Weiss, A. Torralba, R. Fergus, NIPS 2008
- Kulis and Darrell, NIPS 2009, Gong and Lazebnik, CVPR 20111
- Norouzi and Fleet, ICML 2011

Learning Similarity Measures

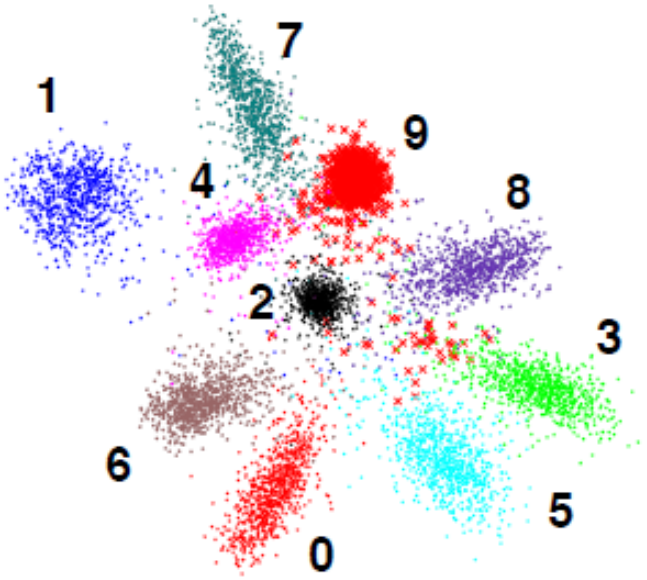


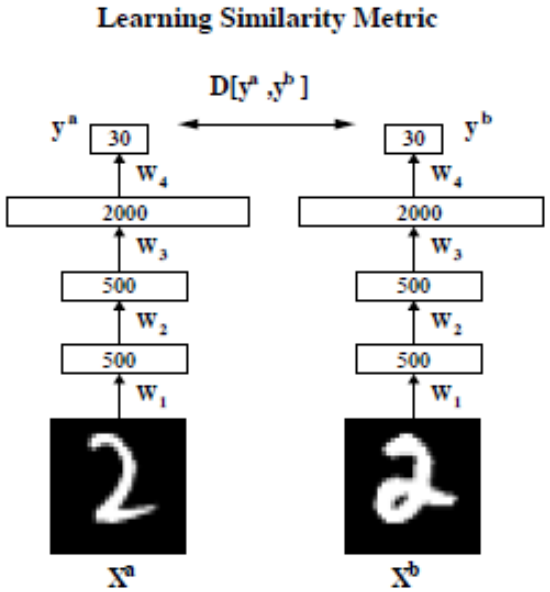
Related to Siamese Networks of LeCun.



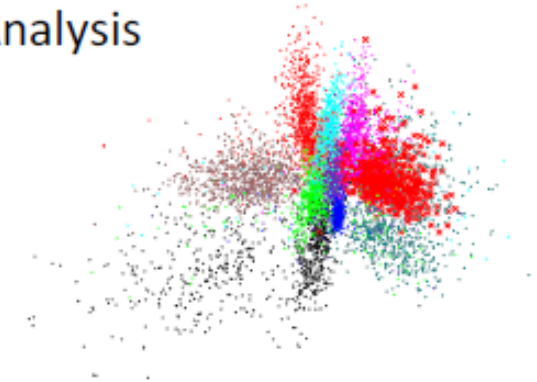
- Learn a nonlinear transformation of the input space
- Optimize to make KNN perform well in the low-dimensional feature space

Learning Similarity Measures

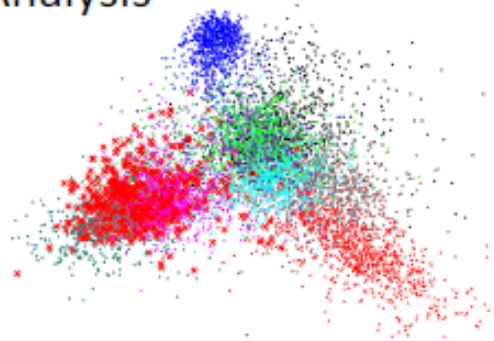




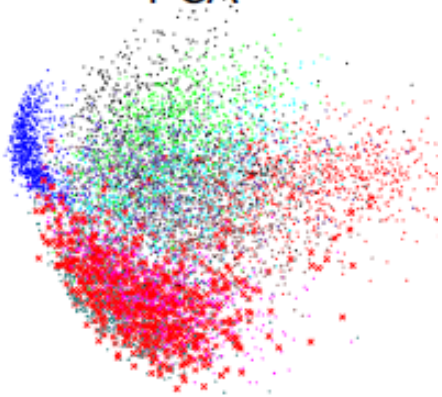
Neighborhood Component Analysis



Linear discriminant Analysis

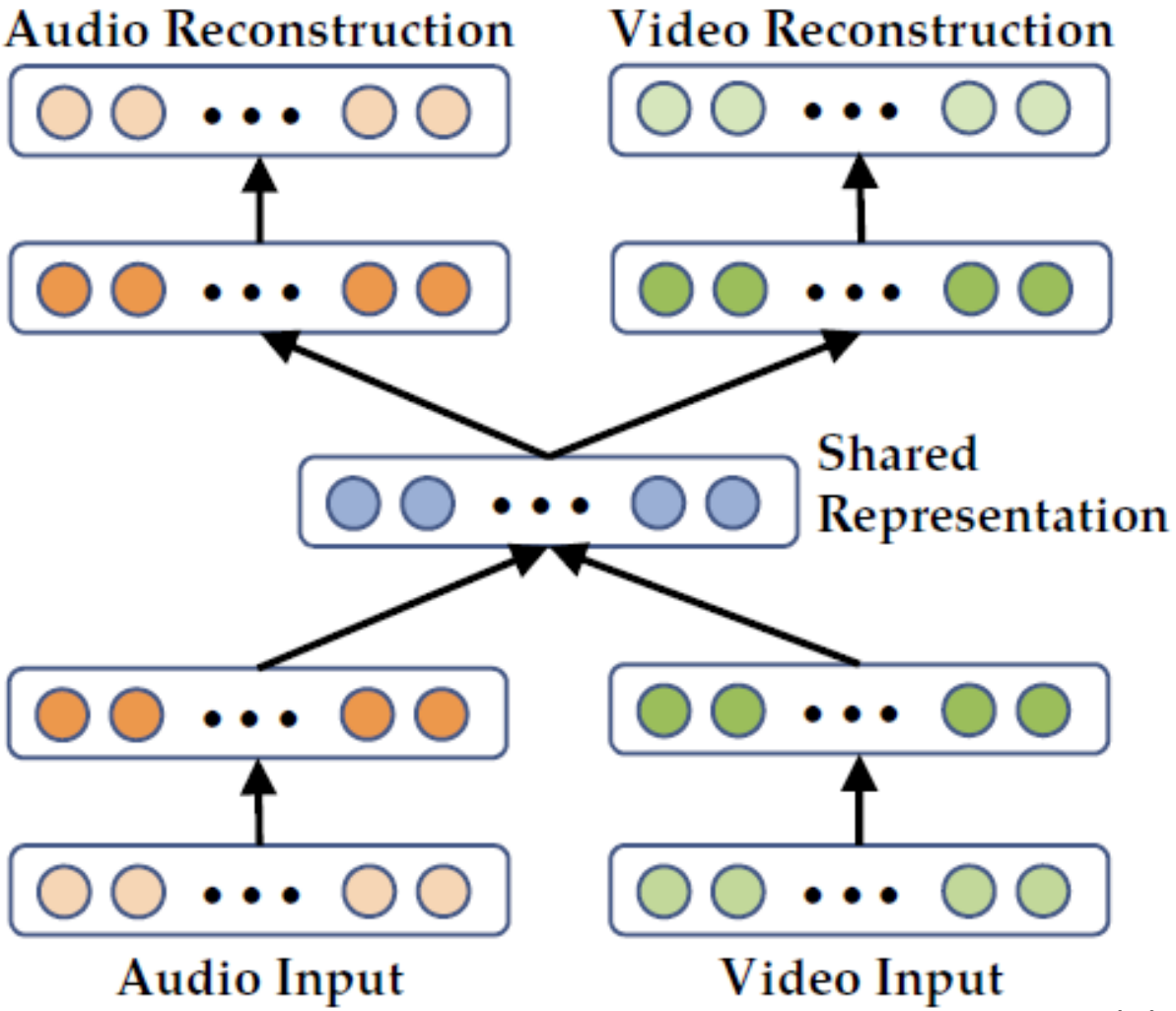


PCA



Multimodal Learning

- Bimodal deep autoencoder



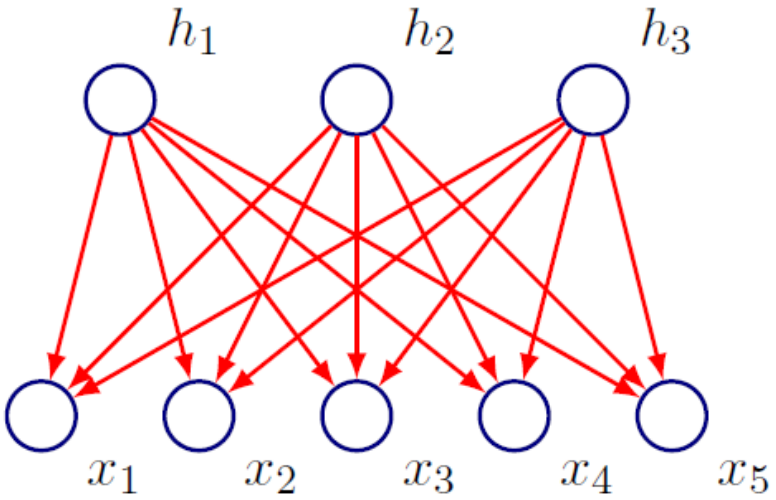
Slide Courtesy: J Ngiam

Outline

1. Course Review
2. Linear Factor Model
3. Autoencoder
4. DBN and RBM

Earlier we had:

- Generative models can be modeled as linear graphical models
- The nodes represent random variables and arcs indicate dependency
- All variables are factorable and have a nice Gaussian form

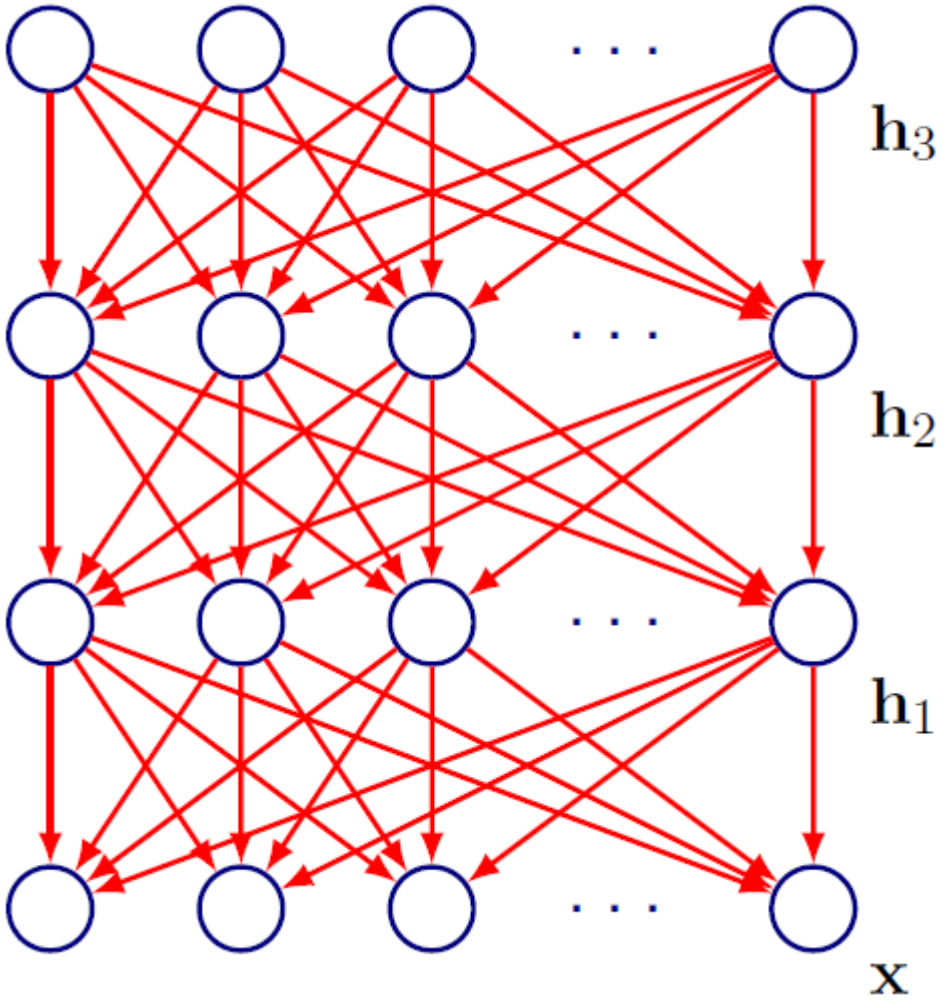


More General Models

- Suppose $P(h)$ cannot be assumed to have a nice Gaussian form
- The decoding of the input from the latent states can be a complicated non-linear function
- Some of the random variables are observed, others are hidden

- Learning and inference can get complicated!

Sigmoid Belief Networks



- Just like a feedforward network, but with arrows reversed

Sigmoid Belief Networks

- Let x = h^0, consider binary activations, then:

$$P(\mathbf{h}_i^k = 1|\mathbf{h}^{k+1}) = sigm(b_i^k + \sum_j W_{i,j}^{k+1}\mathbf{h}_j^{k+1})$$

- The joint probability factorizes as:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l)\Big(\prod_{k=1}^{l-1} P(\mathbf{h}^k|\mathbf{h}^{k+1})\Big)P(\mathbf{x}|\mathbf{h}^1)$$

- Marginalization yields *P*(x), intractable in practice except for very small models

Sigmoid Belief Networks

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l)\Big(\prod_{k=1}^{l-1} P(\mathbf{h}^k|\mathbf{h}^{k+1})\Big)P(\mathbf{x}|\mathbf{h}^1)$$

- The top level prior is chosen as factorizable:

$$P(\mathbf{h}^l) = \prod_i P(\mathbf{h}_i^l)$$

- A single (Bernoulli) parameter is needed for each h_i in case of binary units

Sigmoid Belief Networks

- General case models are called Helmholtz Machines
- Two key references:
  - G. E. Hinton, P. Dayan, B. J. Frey, R. M. Neal: The Wake-Sleep Algorithm for Unsupervised Neural Networks, In Science, 1995
  - R. M. Neal: Connectionist Learning of Belief Networks, In Artificial Intelligence, 1992

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 (only consider x for simplicity):

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

- Energies are in the log-probability domain:

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

Energy Based Models

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

- 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 (sum formulation) 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 $\mathbf{x}$
- If f_i is large => $P_i(\mathbf{x})$ is small i.e. the expert thinks $\mathbf{x}$ is implausible (constraint violated)
- If f_i is small => $P_i(\mathbf{x})$ is large i.e. the expert thinks $\mathbf{x}$ is plausible (constraint satisfied)

Latent Variables

- $\mathbf{x}$ is observed, let's say $\mathbf{h}$ are hidden variables 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

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

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

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

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

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

Data Log-Likelihood Gradient

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

- We have an expression for $P(\mathbf{x})$ and hence for the data log-likelihood $\log P(\mathbf{x})$
- The gradient is simply working from the above:

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta} + \frac{1}{Z}\sum_{\tilde{\mathbf{x}}} \exp^{-(\text{FreeEnergy}(\tilde{\mathbf{x}}))} \frac{\partial \text{FreeEnergy}(\tilde{\mathbf{x}})}{\partial \theta}$$

Data Log-Likelihood Gradient

- The expected log-likelihood gradient over the training set has the following form:

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

Data Log-Likelihood Gradient

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

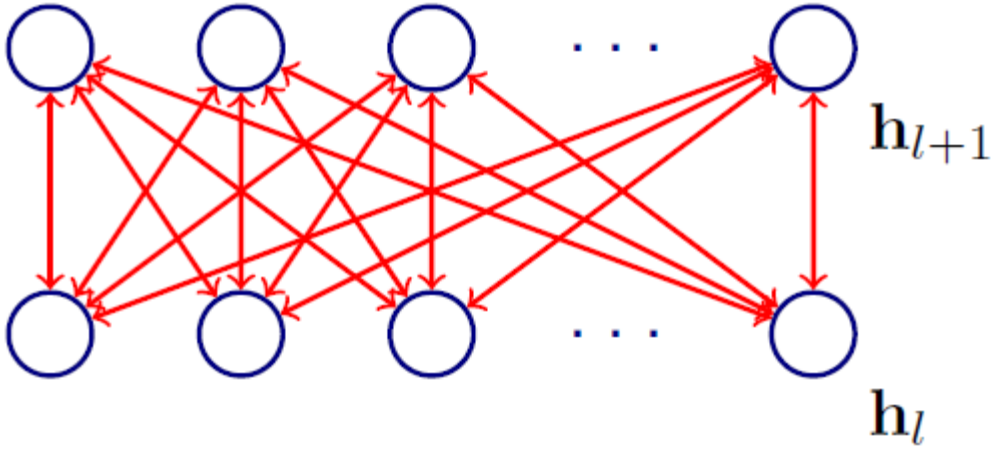
- $\tilde{P}$ is the empirical training distribution
- Easy to compute!

Data Log-Likelihood Gradient

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- P is the model distribution (exponentially many configurations!)
- Usually very hard to compute!
- Resort to Markov Chain Monte Carlo to get a stochastic estimator of the gradient

Restricted Boltzmann Machines

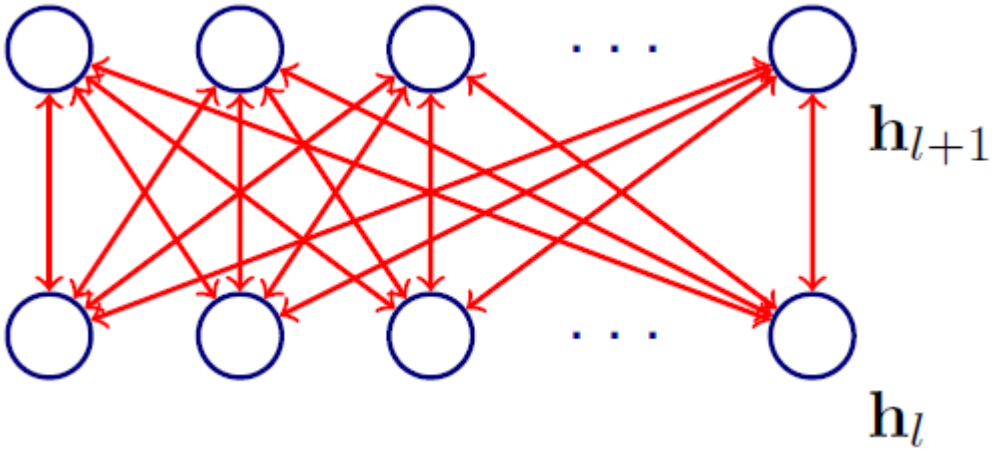


- Recall the form of energy:

$$\text{Energy}(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T W\mathbf{x}$$

- Originally proposed by Smolensky (1987) who called them Harmoniums as a special case of Boltzmann Machines

Restricted Boltzmann Machines



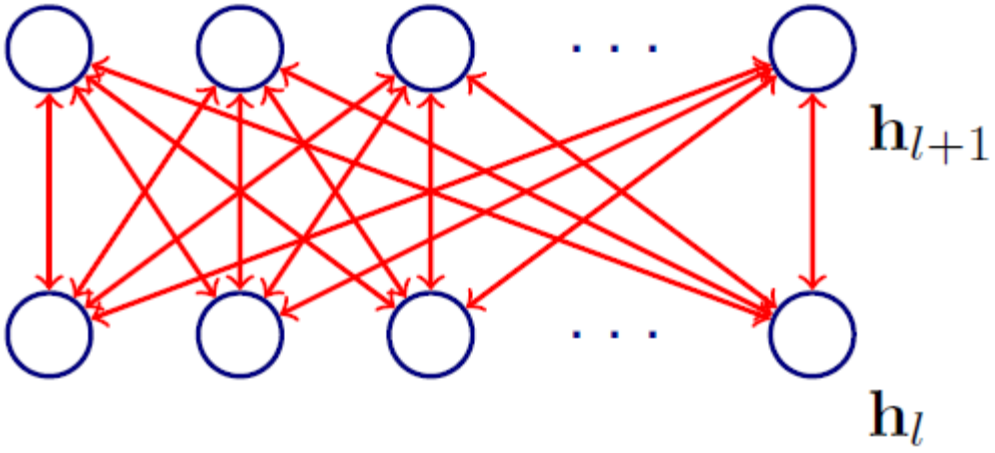
- As seen before, the Free Energy can be computed efficiently:

$$\text{FreeEnergy}(\mathbf{x}) = -\mathbf{b}^T\mathbf{x} - \sum_i \log \sum_{\mathbf{h}_i} \exp^{\mathbf{h}_i(\mathbf{c}_i + W_i\mathbf{x})}$$

- The conditional probability:

$$P(\mathbf{h}|\mathbf{x}) = \frac{\exp\left(\mathbf{b}^T\mathbf{x} + \mathbf{c}^T\mathbf{h} + \mathbf{h}^T W\mathbf{x}\right)}{\sum_{\tilde{\mathbf{h}}} \exp\left(\mathbf{b}^T\mathbf{x} + \mathbf{c}^T\tilde{\mathbf{h}} + \tilde{\mathbf{h}}^T W\mathbf{x}\right)} = \prod_i P(\mathbf{h}_i|\mathbf{x})$$

Restricted Boltzmann Machines



- $\mathbf{x}$ and $\mathbf{h}$ play symmetric roles:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

- The common transfer (for the binary case):

$$P(\mathbf{h}_i = 1|\mathbf{x}) = \sigma(\mathbf{c_i} + W_i\mathbf{x})$$

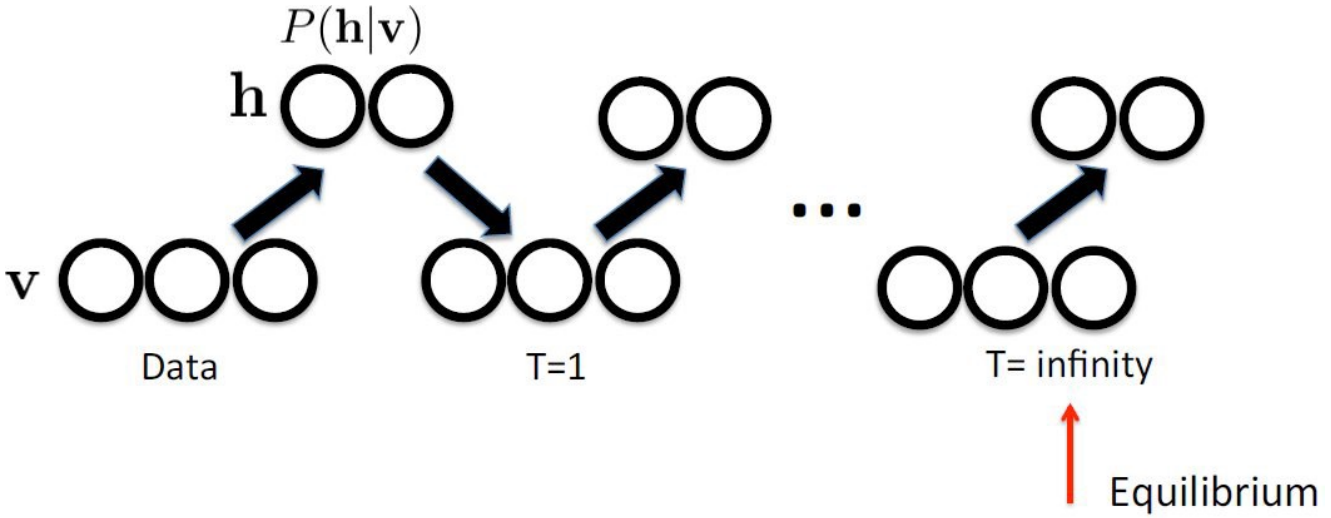
$$P(\mathbf{x}_j = 1|\mathbf{h}) = \sigma(\mathbf{b_j} + W_{:,j}^T\mathbf{h})$$

Approximate Learning and Gibbs Sampling

$$\mathbb{E}_{\tilde{P}}\left[\frac{\partial \log P(\mathbf{x})}{\partial \theta}\right] = \mathbb{E}_{\tilde{P}}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right] + \mathbb{E}_{P}\left[\frac{\partial \text{FreeEnergy}(\mathbf{x})}{\partial \theta}\right]$$

- We saw the expression for Free Energy for a RBM. But the second term was intractable. How do learn in this case?
- Replace the average over all possible input configurations by sampling
- Run Markov Chain Monte Carlo (Gibbs Sampling):
  - First sample $\mathbf{x}_1 \sim P(\mathbf{x})$, then $\mathbf{h}_1 \sim P(\mathbf{h}|\mathbf{x}_1)$, then $\mathbf{x}_2 \sim \mathrm{P}(\mathbf{x}|\mathbf{h}_1)$, then $\mathbf{h}_2 \sim \mathrm{P}(\mathbf{h}|\mathbf{x}_2)$ till $\mathbf{x}_{k+1}$

Approximate Learning and Gibbs Sampling



- We have already seen:

$$P(\mathbf{x}|\mathbf{h}) = \prod_i P(\mathbf{x}_i|\mathbf{h})$$

$$P(\mathbf{h}|\mathbf{x}) = \prod_i P(\mathbf{h}_i|\mathbf{x})$$

- With: $P(\mathbf{h}_i = 1|\mathbf{x}) = \sigma(\mathbf{c_i} + W_i\mathbf{x})$ and

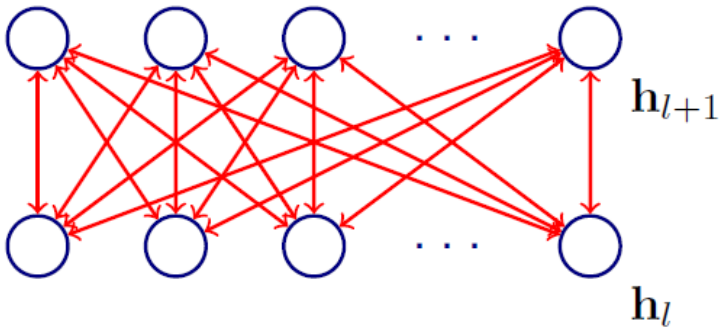
$$P(\mathbf{x}_j = 1|\mathbf{h}) = \sigma(\mathbf{b_j} + W_{:,j}^T\mathbf{h})$$

Training RBM: Contrastive Divergence Algorithm

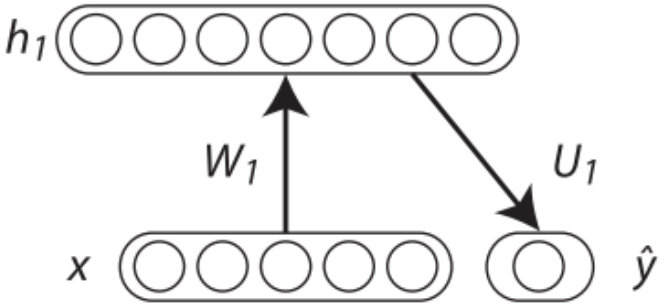
1. Start with a training example on the visible units
2. Update all the hidden units in parallel
3. Update all the visible units in parallel to obtain a reconstruction
4. Update all the hidden units again
5. Update model parameters

- Aside: Easy to extend RBM (and contrastive divergence) to the continuous case

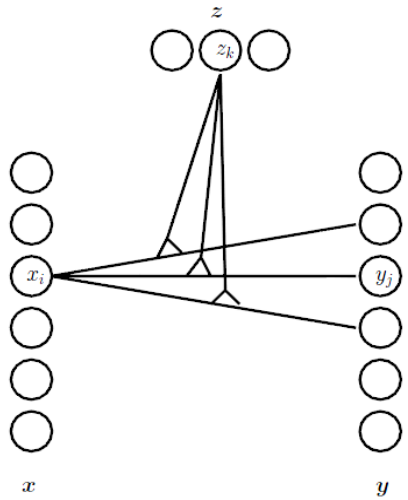
High-order RBM



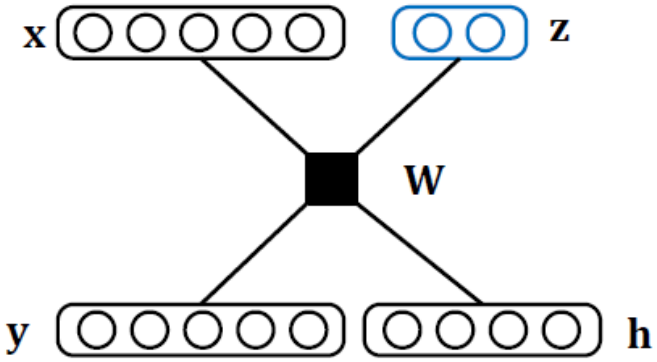
Original RBM



Supervised RBM



High-order RBM



Supervised high-order RBM

High-order RBM

IJCAI2014: 873 rejected

R1: However, it is not clear at all from the paper why is this extension necessary ? Why is this gated modification necessary ?

R1: Expect the adding of z label. This is quite marginal, plus not clear why and under-what context is this necessary ?

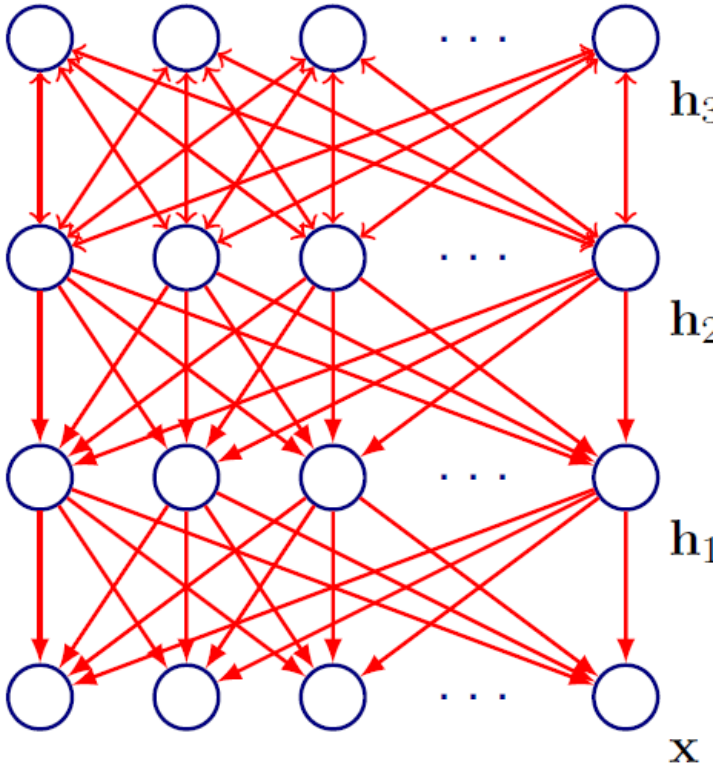
R1: Authors compared the pro-posed model with unsupervised model baselines. This is certain not fair since the supervised model use much more information.

ICCV2015: 144 accepted

R1: There are many kinds of Boltzmann machines. It is not clear how different these methods, and advantages

R2: this paper seems like a fusion of these two models ([Memisevic and Hinton, 2010] - M2010 and [Nair and Hinton, 2009] - N2009). There are no new technical hurdles overcome or novel learning or inference methods introduced.

Deep Belief Networks



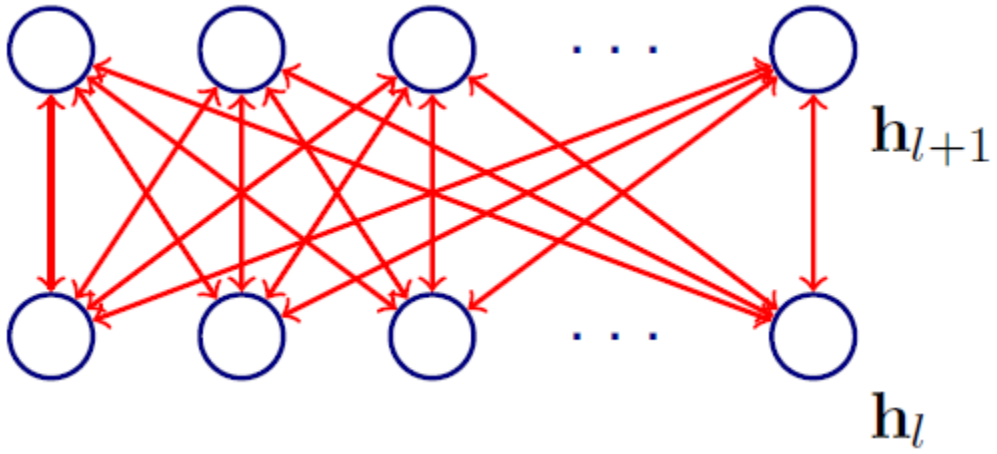
- Deep Belief Networks are like Sigmoid Belief Networks except for the top two layers
- The top two layers now have undirected edges

Deep Belief Networks

- The joint probability changes as:

$$P(\mathbf{x}, \mathbf{h}^1, \ldots, \mathbf{h}^l) = P(\mathbf{h}^l, \mathbf{h}^{l-1}) \left(\prod_{k=1}^{l-2} P(\mathbf{h}^k | \mathbf{h}^{k+1}) \right) P(\mathbf{x} | \mathbf{h}^1)$$

Deep Belief Networks



- The top two layers are a Restricted Boltzmann Machine (RBM)
- A RBM has the joint distribution:

$$P(\mathbf{h}^{l+1}, \mathbf{h}^l) \propto \exp(\mathbf{b}'\mathbf{h}^{l-1} + \mathbf{c}'\mathbf{h}^l + \mathbf{h}^l W \mathbf{h}^{l-1})$$

Greedy Layer-wise Training of DBNs

Reference: G. E. Hinton, S. Osindero and Y-W Teh: A Fast Learning Algorithm for Deep Belief Networks, In Neural Computation, 2006.

1. Construct a RBM with input $\mathbf{x}$ and a hidden layer $\mathbf{h}$, train the RBM
2. Stack another layer on top of the RBM to form a new RBM. Fix $\mathbf{W}^1$, sample from $P(\mathbf{h}^1|\mathbf{x})$, train $\mathbf{W}^2$ as RBM
3. Continue till k layers
4. Implicitly defines $P(\mathbf{x})$ and $P(\mathbf{h})$ (variational bound justifies layerwise training)
5. Can then be discriminatively fine-tuned using backpropagation

Deep Learning Since 2006

materials are identical for all configurations. The blue bars in Fig. 1 summarize the measured SHG signals. For excitation of the *LC* resonance in Fig. 1A (horizontal incident polarization), we find an SHG signal that is 500 times above the noise level. As expected for SHG, this signal closely scales with the square of the incident power (Fig. 2A). The polarization of the SHG emission is nearly vertical (Fig. 2B). The small angle with respect to the vertical is due to deviations from perfect mirror symmetry of the SRRs (see electron micrographs in Fig. 1). Small detuning of the *LC* resonance toward smaller wavelength (i.e., to 1.3-μm wavelength) reduces the SHG signal strength from 100% to 20%. For excitation of the Mie resonance with vertical incident polarization in Fig. 1D, we find a small signal just above the noise level. For excitation of the Mie resonance with horizontal incident polarization in Fig. 1C, a small but significant SHG emission is found, which is again po-

Reducing the Dimensionality of Data with Neural Networks

G. E. Hinton* and R. R. Salakhutdinov

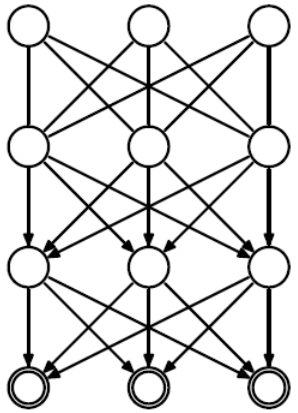
High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Gradient descent can be used for fine-tuning the weights in such "autoencoder" networks, but this works well only if the initial weights are close to a good solution. We describe an effective way of initializing the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data.

Dimensionality reduction facilitates the classification, visualization, communication, and storage of high-dimensional data. A simple and widely used method is principal components analysis (PCA), which finds the directions of greatest variance in the data set and represents each data point by its coordinates along each of these directions. We describe a nonlinear generalization of PCA that uses an adaptive, multilayer "encoder" network

28 JULY 2006 VOL 313 SCIENCE www.sciencemag.org

Deep neural network + RBM pre-training

Image Classification



Deep Belief Network

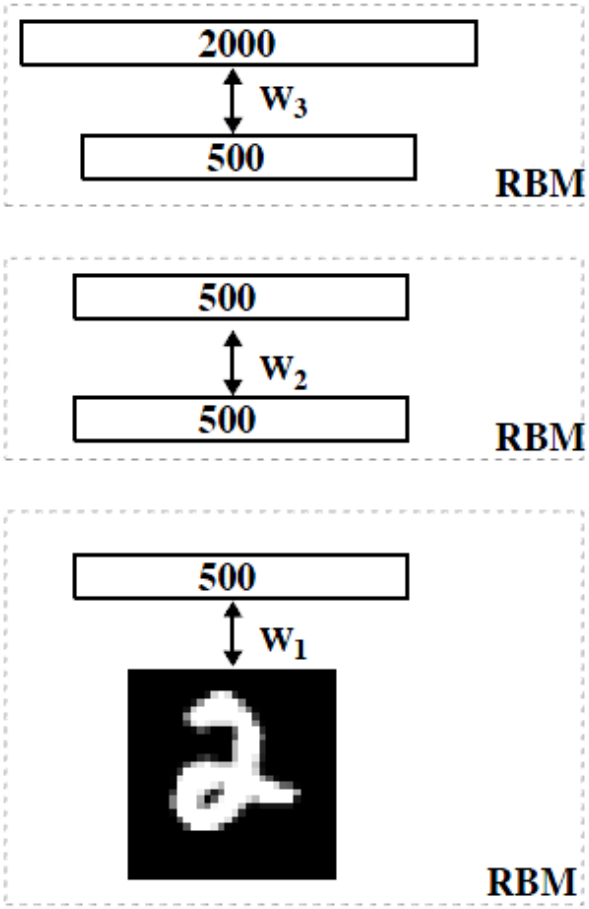
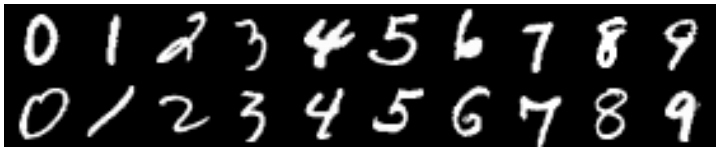
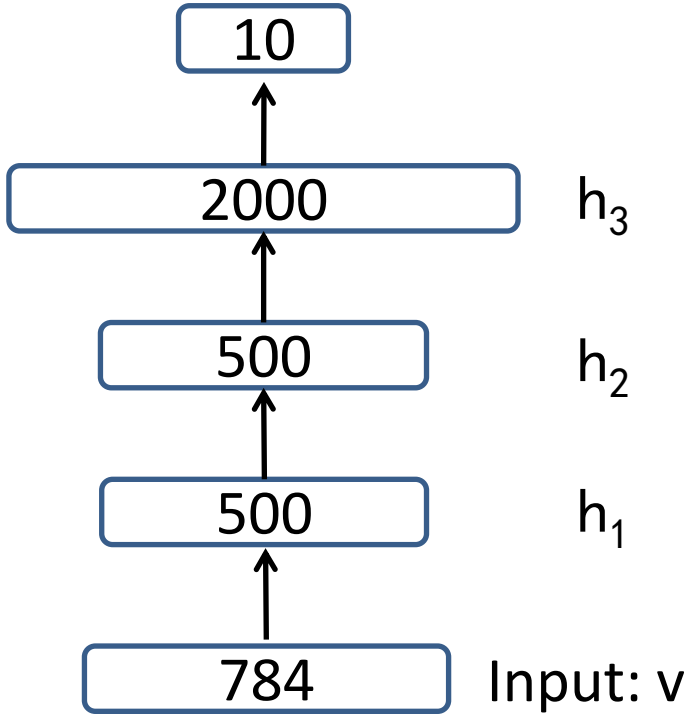


Image Classification

- **Back-prop net** (Platt et al.)
  - 1.6%
- **SVM** (Decoste et al. 2002)
  - 1.4%
- **Generative pretraining** and back-propagation (Hinton et al. 2006)
  - 1.15%
  - 0.39% (more training data)

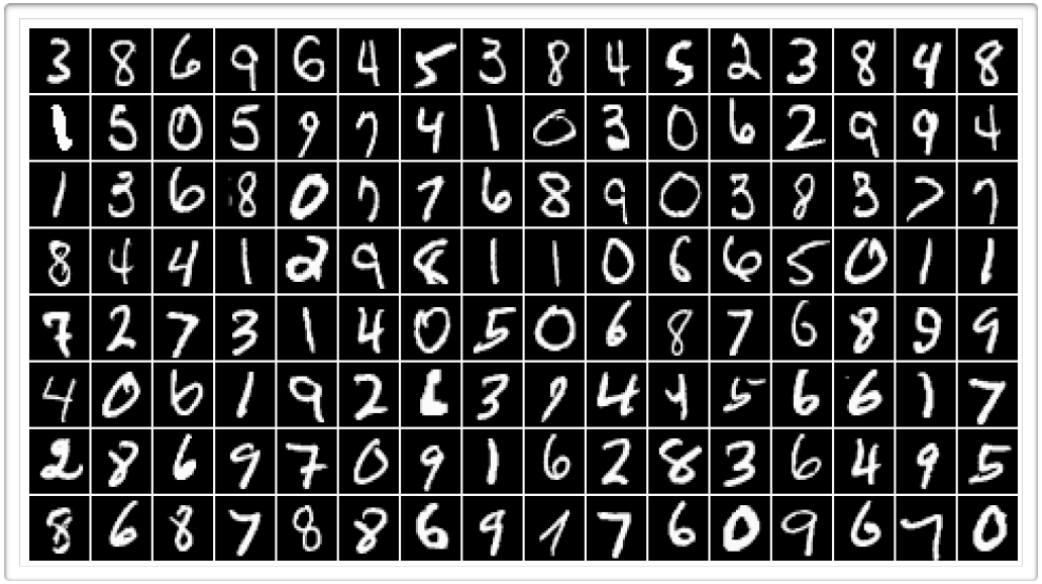


Labels: 0-9

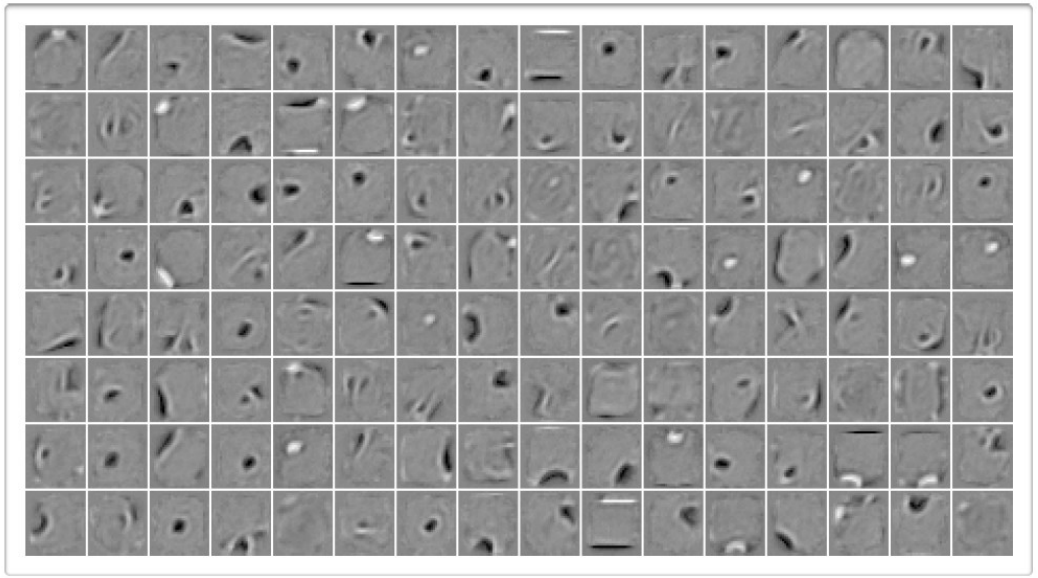


Example: MNIST

Original images:



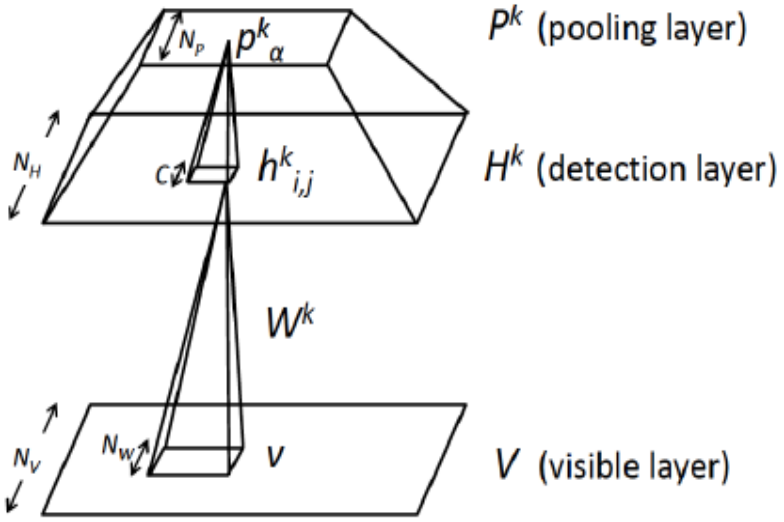
Learned features:

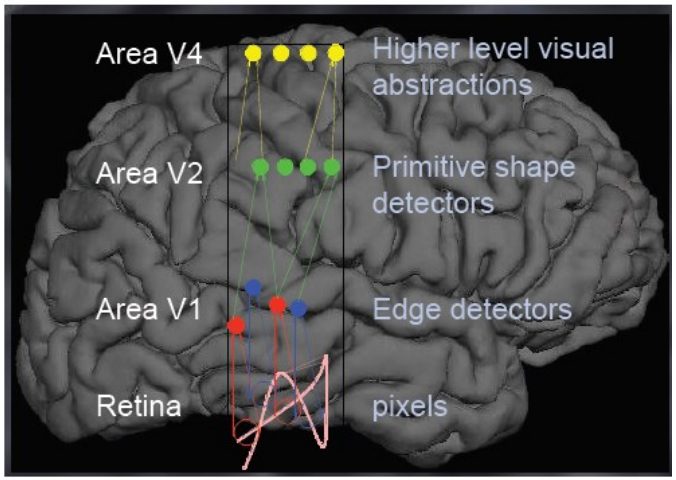


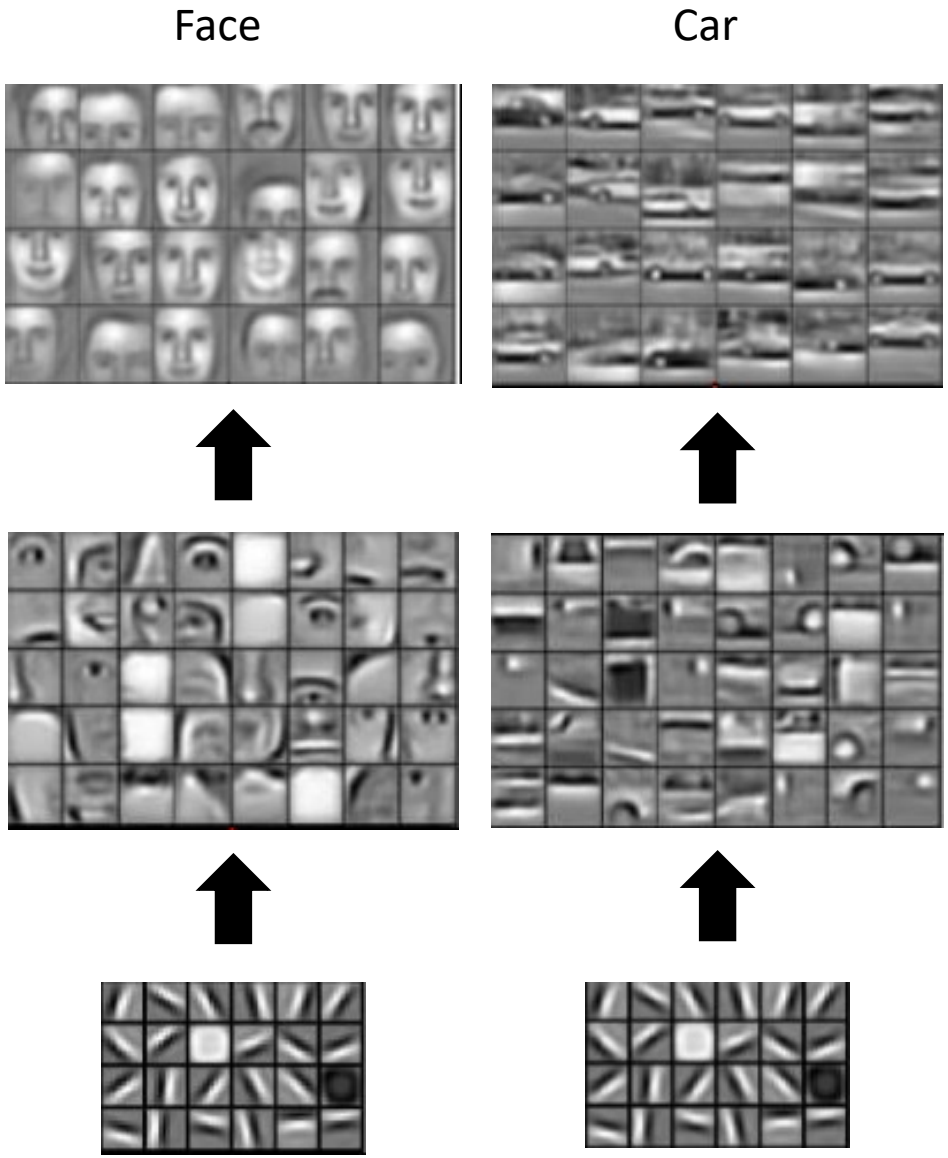
(Larochelle et al., JMLR 2009)

Hierarchical Feature Learning

- Convolutional DBN



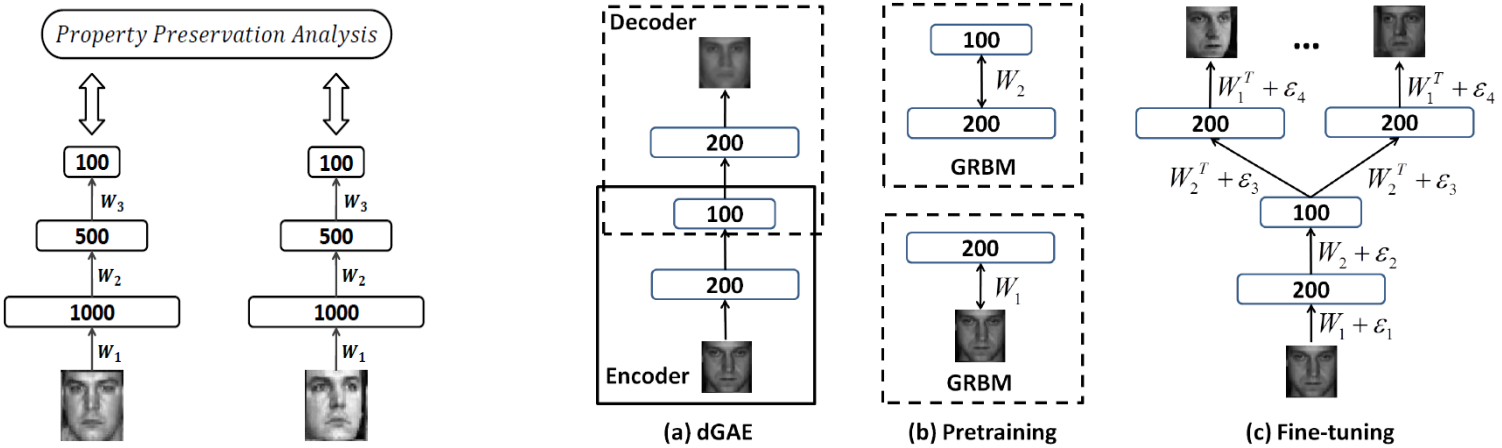


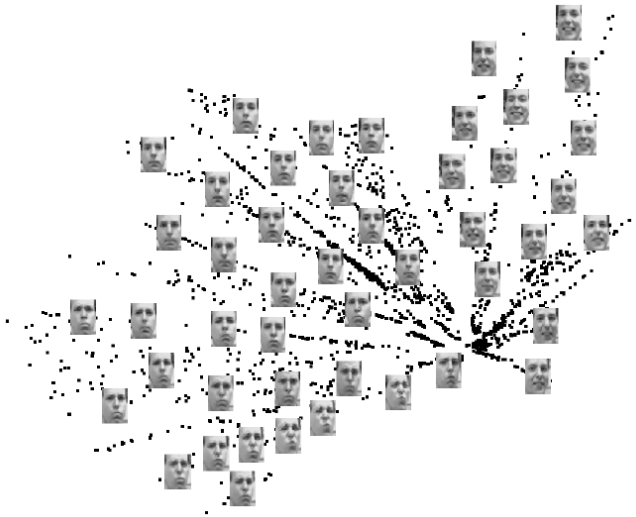


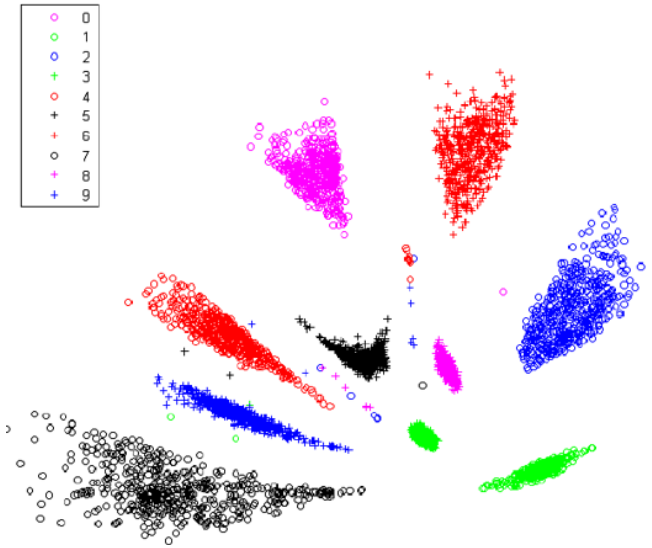
[Lee, Grosse, Ranganath & Ng, 2009]

Dimensionality Reduction

- Generalized Auto-Encoder

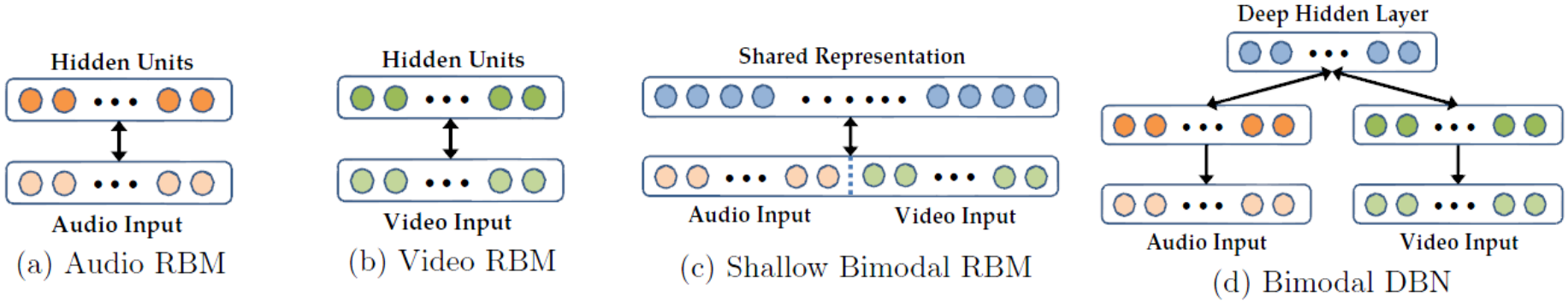


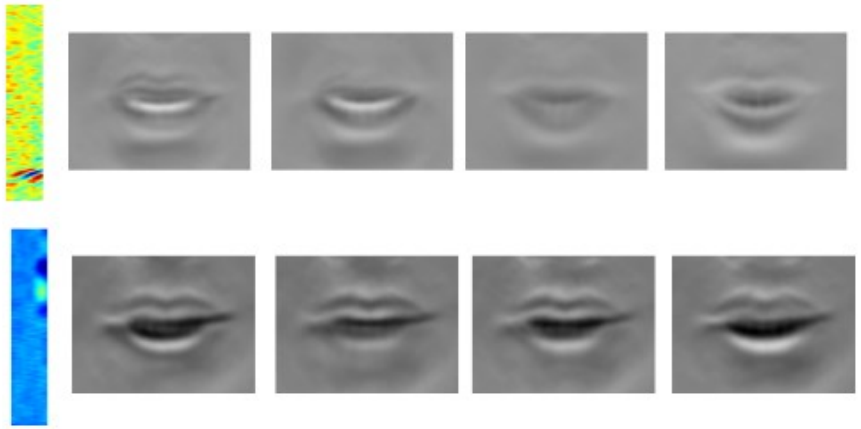


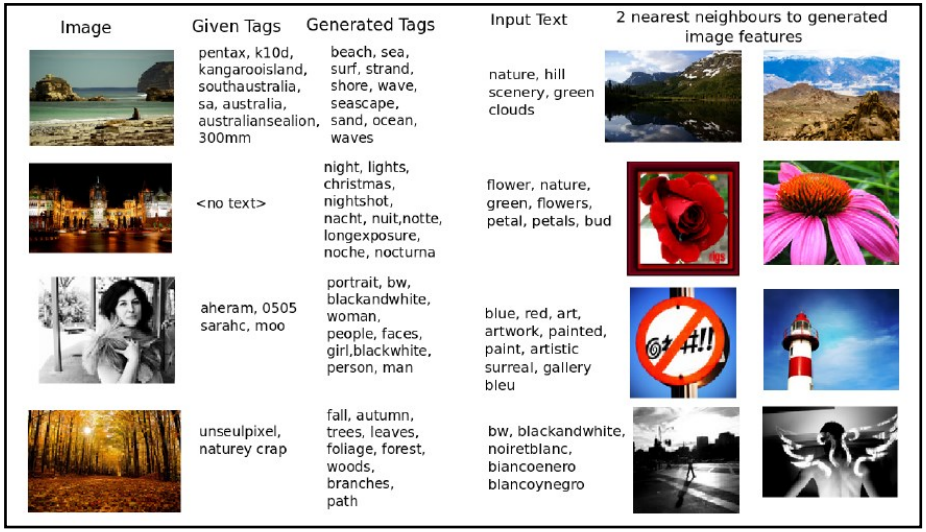


Multimodal Learning

- Bimodal deep Boltzmann machine







Slide Courtesy: J Ngiam

Acknowledgement

Some of the materials in these slides are drawn inspiration from:

- Shubhendu Trivedi and Risi Kondor, University of Chicago, Deep Learning Course
- Hung-yi Lee, National Taiwan University, Machine Learning and having it Deep and Structured course
- Xiaogang Wang, The Chinese University of Hong Kong, Deep Learning Course
- Fei-Fei Li, Standord University, CS231n Convolutional Neural Networks for Visual Recognition course

Next Time

- Generative Model (2)

Questions?

Thank You !