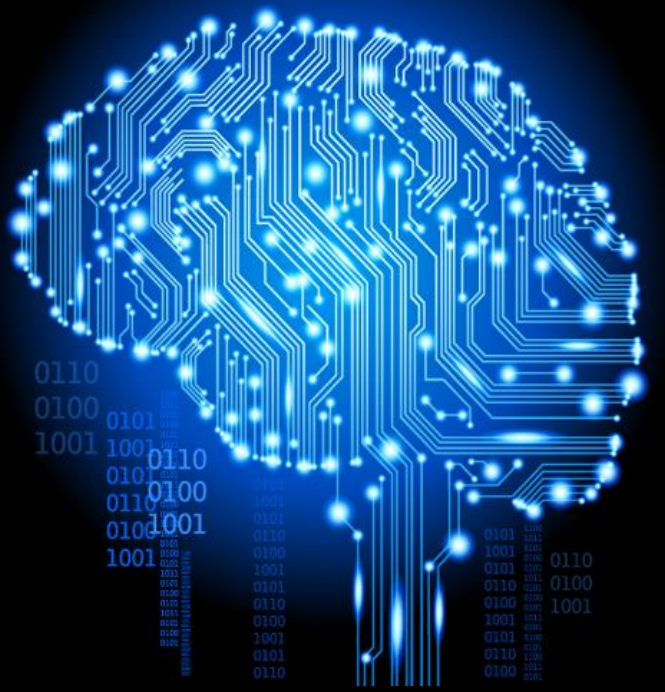
CENG 793

Advanced Deep Learning



© AlchemyAPI

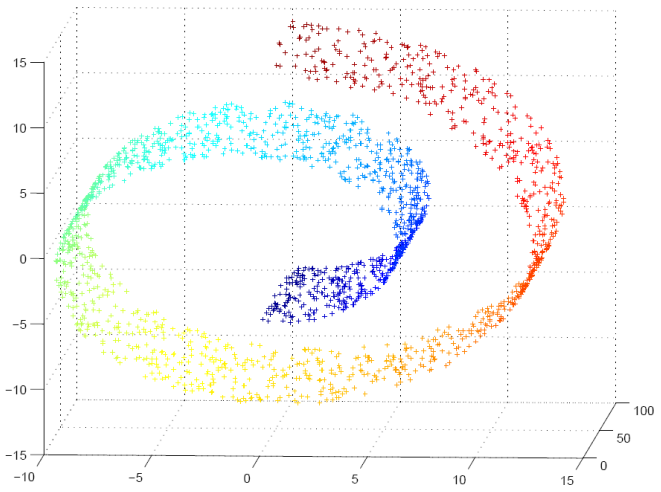
Auto-encoders

Sinan Kalkan & Emre Akbaş

today

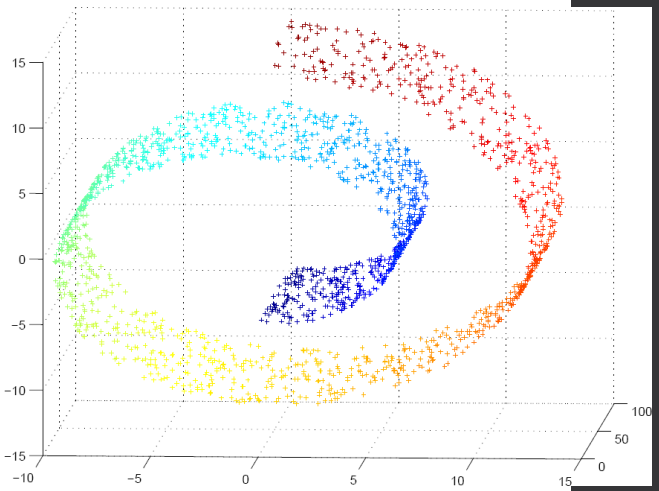
- Manifold Learning
  - Principle Component Analysis
  - Independent Component Analysis
- Autoencoders
- Sparse autoencoders
- K-sparse autoencoders
- Denoising autoencoders
- Contraction autoencoders

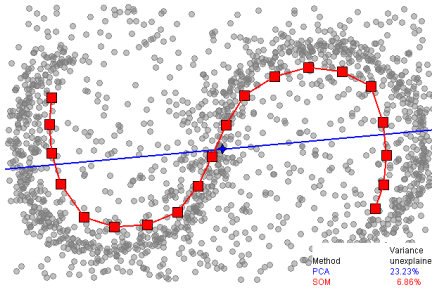
Manifold Learning

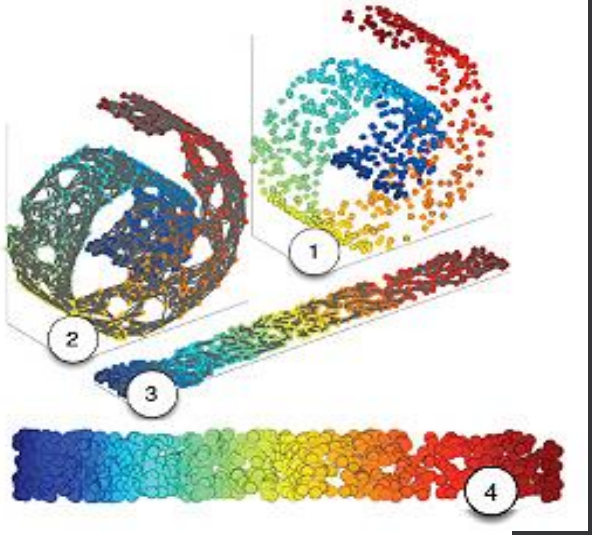


Manifold Learning

- Discovering the “hidden” structure in the high-dimensional space
- Manifold: “hidden” structure.
- Non-linear dimensionality reduction







http://www.convexoptimization.com/dattorro/manifold_learning.html

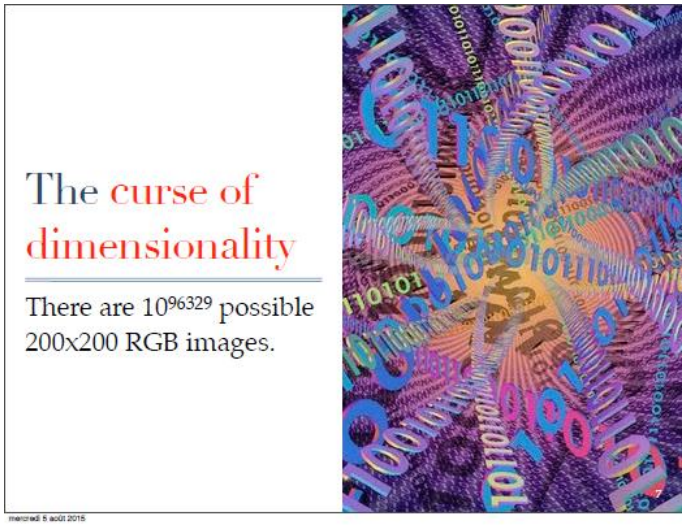
Manifold Learning

- Many approaches:
  - Self-Organizing Map (Kohonen map/network)
  - Auto-encoders
  - Principles curves & manifolds: Extension of PCA
  - Kernel PCA, Nonlinear PCA
  - Curvilinear Component Analysis
  - Isomap: Floyd-Marshall + Multidimensional scaling
  - Data-driven high-dimensional scaling
  - Locally-linear embedding
  - …

Manifold learning

- Autoencoders learn lower-dimensional manifolds embedded in higher-dimensional manifolds

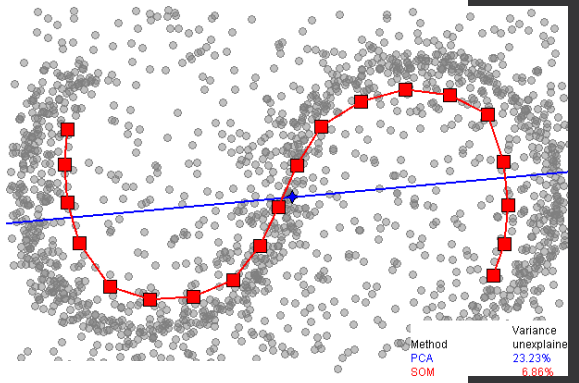
- Assumption: "Natural data in high dimensional spaces concentrates close to lower dimensional manifolds"
  - Natural images occupy a very small fraction in a space of possible images



(Pascal Vincent)

Manifold Learning

- Many approaches:
  - Self-Organizing Map (Kohonen map/network)



Algorithm [edit]

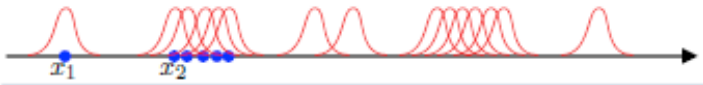
1. Randomize the map's nodes' weight vectors
2. Grab an input vector $\mathbf{D}(\mathbf{t})$
3. Traverse each node in the map
    1. Use the Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector
    2. Track the node that produces the smallest distance (this node is the best matching unit, BMU)
4. Update the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector
    1. $\mathbf{W_v}(s + 1) = \mathbf{W_v}(s) + \Theta(u, v, s)\, \alpha(s)(\mathbf{D}(t) - \mathbf{W_v}(s))$
5. Increase s and repeat from step 2 while $s < \lambda$

A variant algorithm:

1. Randomize the map's nodes' weight vectors
2. Traverse each input vector in the input data set
    1. Traverse each node in the map
        1. Use the Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector
        2. Track the node that produces the smallest distance (this node is the best matching unit, BMU)
    2. Update the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector
        1. $\mathbf{W_v}(s + 1) = \mathbf{W_v}(s) + \Theta(u, v, s)\, \alpha(s)(\mathbf{D}(t) - \mathbf{W_v}(s))$
3. Increase s and repeat from step 2 while $s < \lambda$

https://en.wikipedia.org/wiki/Self-organizing_map

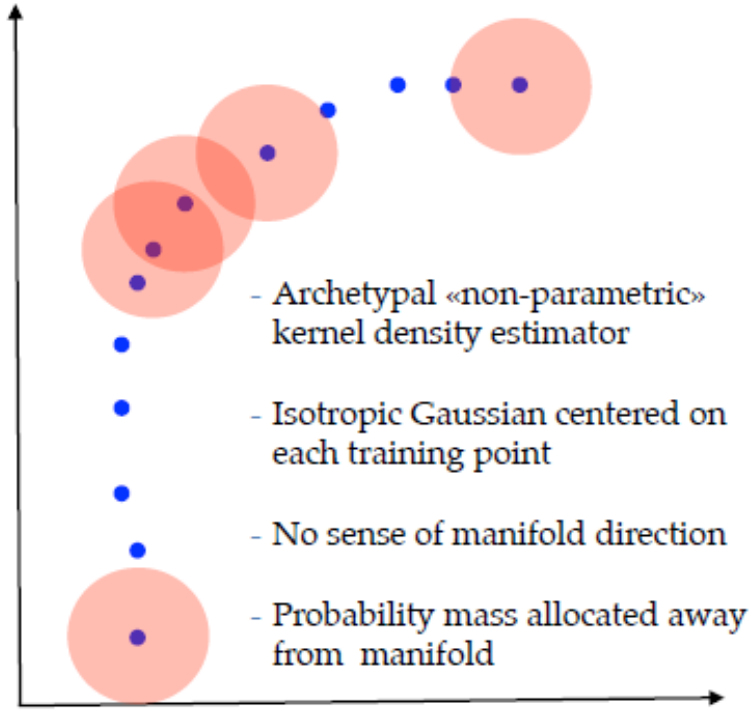
Non-parametric density estimation



$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(x; x_i, C_i)$$

Classical Parzen Windows

density estimator

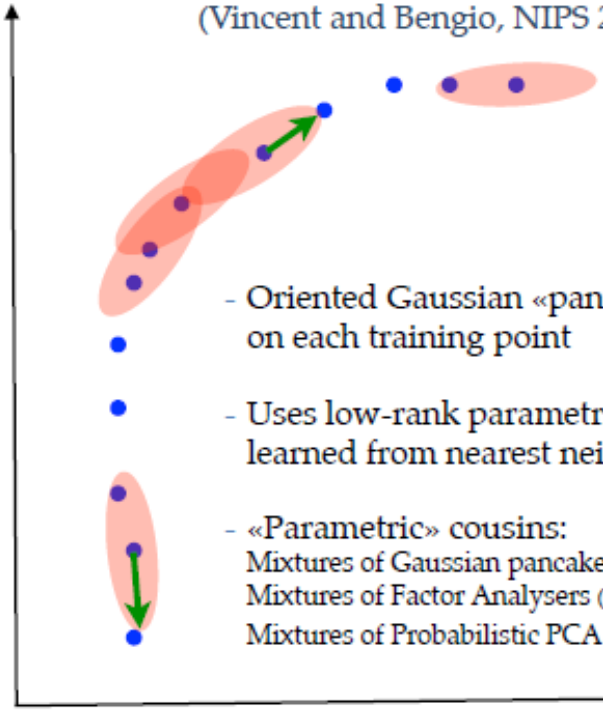


- Archetypal «non-parametric» kernel density estimator
- Isotropic Gaussian centered on each training point
- No sense of manifold direction
- Probability mass allocated away from manifold

Manifold Parzen Windows

density estimator

(Vincent and Bengio, NIPS 2003)



- Oriented Gaussian «pancake» centered on each training point
- Uses low-rank parametrization of C_i, learned from nearest neighbors (local PCA)
- «Parametric» cousins:
  Mixtures of Gaussian pancakes (Hinton et al. 95)
  Mixtures of Factor Analysers (Gharamani + Hinton 96)
  Mixtures of Probabilistic PCA (Tipping + Bishop 99)

11

mercredi 5 août 2015

(Pascal Vincent)

Non-local manifold Parzen windows

(Bengio, Larochelle, Vincent, NIPS 2006)

Isotropic Parzen:

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(x; x_i, \underbrace{\sigma^2 I}_{\text{isotropic}})$$

Manifold Parzen:
(Vincent and Bengio, NIPS 2003)

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(x; x_i, \underbrace{C_i}_{d_M \text{ high variance directions from PCA on } k \text{ nearest neighbors}})$$

Non-local manifold Parzen:
(Bengio, Larochelle, Vincent, NIPS 2006)

$$\hat{p}(x) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{N}(x; \underbrace{\mu(x_i), C(x_i)}_{d_M \text{ high variance directions output by neural network trained to maximize likelihood of } k \text{ nearest neighbors}})$$

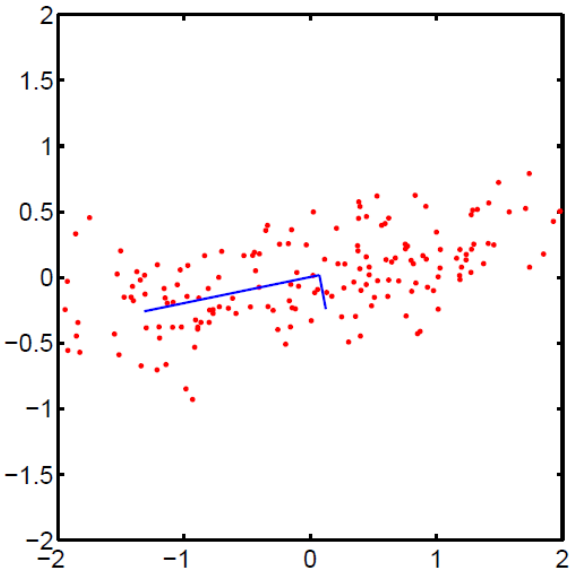
12

(Pascal Vincent)

[Pearson 1901] [Hotelling 1933]

Principle Component Analysis (PCA)

- Principle Components:
  - Orthogonal directions with most variance
  - Eigen-vectors of the co-variance matrix



(Ole Winther)

- Mathematical background:
  - Orthogonality:
    - Two vectors $\vec{u}$ and $\vec{v}$ are orthogonal iff
      $$\vec{u} \cdot \vec{v} = 0$$
  - Variance:

$$\sigma(X)^2 = Var(X) = E[(X-\mu)^2] = \sum_i p(x_i)(x_i-\mu)^2$$

where the (weighted) mean, $\mu = E[X] = \sum_i p(x_i)x_i$.

If $p(x_i) = 1/N$:

$$Var(X) = \frac{1}{N}\sum_i (x_i-\mu)^2$$

$$\mu = \frac{1}{N}\sum_i x_i$$

Mathematical background for PCA: Covariance

- Co-variance:
  - Measures how two random variables change wrt each other:

$$Cov(X,Y) = E[(X - E[X])(Y - E[Y])]$$
$$= \frac{1}{N}\sum_i (x_i - E[X])(y_i - E[Y])$$

  - If big values of X & big values of Y "co-occur" and small values of X & small values of Y "co-occur" ➔ high co-variance.
  - Otherwise, small co-variance.

Mathematical background for PCA: Covariance Matrix

- Co-variance Matrix:
  - Denoted usually by Σ
  - For an n-dimensional space:

$$\Sigma_{ij} = Cov(X_i, X_j) = E\left[(X_i - \mu_i)(X_j - m_j)\right]$$

$$\Sigma = \begin{bmatrix} \mathrm{E}[(X_1 - \mu_1)(X_1 - \mu_1)] & \mathrm{E}[(X_1 - \mu_1)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_1 - \mu_1)(X_n - \mu_n)] \\ \mathrm{E}[(X_2 - \mu_2)(X_1 - \mu_1)] & \mathrm{E}[(X_2 - \mu_2)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_2 - \mu_2)(X_n - \mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[(X_n - \mu_n)(X_1 - \mu_1)] & \mathrm{E}[(X_n - \mu_n)(X_2 - \mu_2)] & \cdots & \mathrm{E}[(X_n - \mu_n)(X_n - \mu_n)] \end{bmatrix}.$$

Mathematical background for PCA: Covariance Matrix

- Co-variance Matrix:

$$\Sigma_{ij} = Cov(X_i, X_j)$$
$$= E[(X_i - \mu_i)(X_j - m_j)]$$

- Properties

1. $\Sigma = \mathrm{E}(\mathbf{X}\mathbf{X}^{\mathrm{T}}) - \boldsymbol{\mu}\boldsymbol{\mu}^{\mathrm{T}}$
2. $\sum$ is positive-semidefinite and symmetric.
3. $\operatorname{cov}(\mathbf{A}\mathbf{X} + \mathbf{a}) = \mathbf{A}\ \operatorname{cov}(\mathbf{X})\ \mathbf{A}^{\mathrm{T}}$
4. $\operatorname{cov}(\mathbf{X}, \mathbf{Y}) = \operatorname{cov}(\mathbf{Y}, \mathbf{X})^{\mathrm{T}}$
5. $\operatorname{cov}(\mathbf{X}_1 + \mathbf{X}_2, \mathbf{Y}) = \operatorname{cov}(\mathbf{X}_1, \mathbf{Y}) + \operatorname{cov}(\mathbf{X}_2, \mathbf{Y})$
6. If $p = q$, then $\operatorname{var}(\mathbf{X} + \mathbf{Y}) = \operatorname{var}(\mathbf{X}) + \operatorname{cov}(\mathbf{X}, \mathbf{Y}) + \operatorname{cov}(\mathbf{Y}, \mathbf{X}) + \operatorname{var}(\mathbf{Y})$
7. $\operatorname{cov}(\mathbf{A}\mathbf{X} + \mathbf{a}, \mathbf{B}^{\mathrm{T}}\mathbf{Y} + \mathbf{b}) = \mathbf{A}\ \operatorname{cov}(\mathbf{X}, \mathbf{Y})\ \mathbf{B}$
8. If $\mathbf{X}$ and $\mathbf{Y}$ are independent or uncorrelated, then $\operatorname{cov}(\mathbf{X}, \mathbf{Y}) = \mathbf{0}$

(Wikipedia)

M is called *positive-semidefinite* (or sometimes *nonnegative-definite*) if

$$x^* M x \geq 0$$

for all x in $\mathbf{C}^n$ (or, all x in $\mathbf{R}^n$ for the real matrix). (Wikipedia)

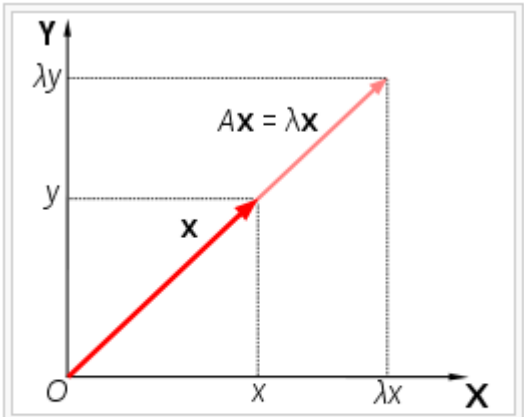
Mathematical background for PCA: Eigenvectors & Eigenvalues

- Eigenvectors and eigenvalues:
  - $\vec{v}$ is an eigenvector of a square matrix A if

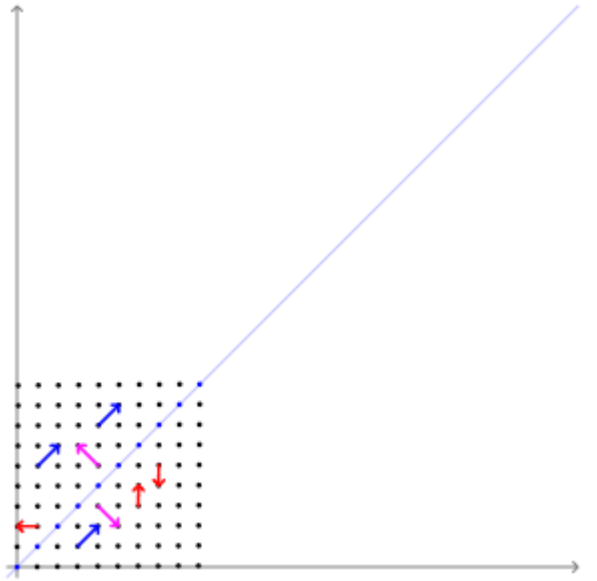
$$A\vec{v} = \lambda\vec{v}$$

where λ is the eigenvalue (scalar) associated with $\vec{v}$.

- Interpretation:
  - "Transformation" A does not change the direction of the vector.
  - It changes the vector's scale, i.e., the eigenvalue.
- Solution:
  - $(A - \lambda I)\vec{v} = 0$ ➔ has a solution when the determinant $|A - \lambda I|$ is zero.
  - Find the eigenvalues, then plug in those values to get the eigenvectors.



Matrix A acts by stretching the vector x, not changing its direction, so x is an eigenvector of A.


The transformation matrix $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ preserves the direction of vectors parallel to $\mathbf{v} = (1,-1)^T$ (in purple) and $\mathbf{w} = (1,1)^T$ (in blue). The vectors in red are not parallel to either eigenvector, so, their directions are changed by the transformation. See also: An extended

Mathematical background for PCA: Eigenvectors & Eigenvalues Example

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

- Setting the determinant $|A - \lambda I|$ to zero:

$$p(\lambda) = |A - \lambda I| = 3 - 4\lambda + \lambda^2 = 0,$$

- The roots: $\lambda = 1$ and $\lambda = 3$
- If you plug in those eigenvalues, for $\lambda = 1$:

$$(A - I)\mathbf{v} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{Bmatrix} v_1 \\ v_2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix},$$

which gives $\mathbf{v}_1 = \{1, -1\}$. For $\lambda = 3$:

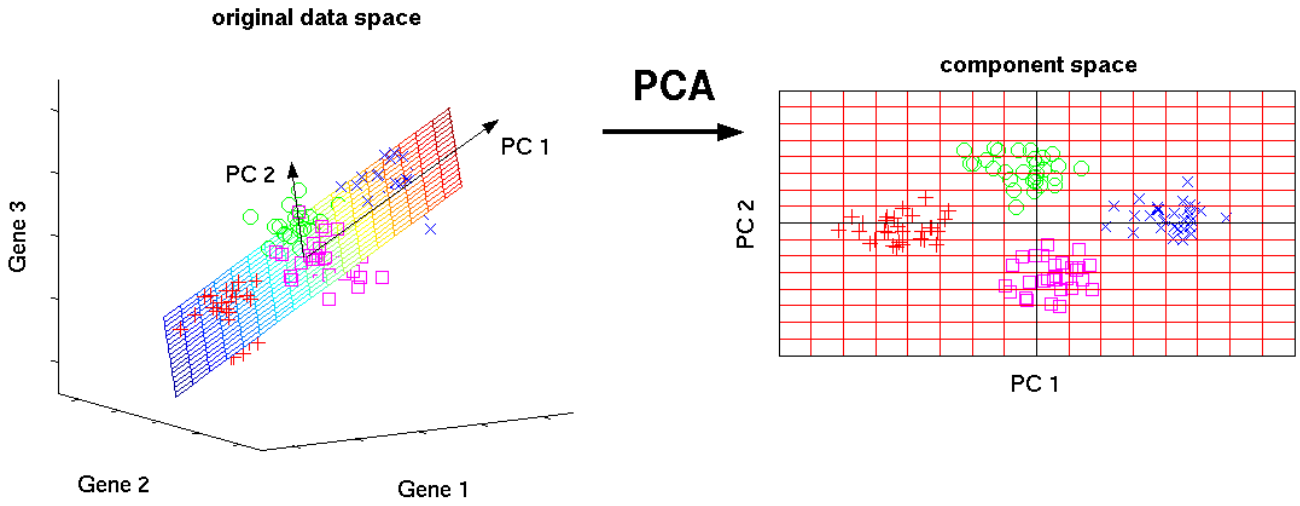
$$(A - 3I)\mathbf{w} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{Bmatrix} w_1 \\ w_2 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix},$$

which gives $\mathbf{v}_2 = \{1,1\}$.

Example from Wikipedia.

PCA allows also dimensionality reduction

- Discard components whose eigenvalue is negligible.



See the following tutorial for more on PCA:
http://www.cs.princeton.edu/picasso/mats/PCA-Tutorial-Intuition_jp.pdf

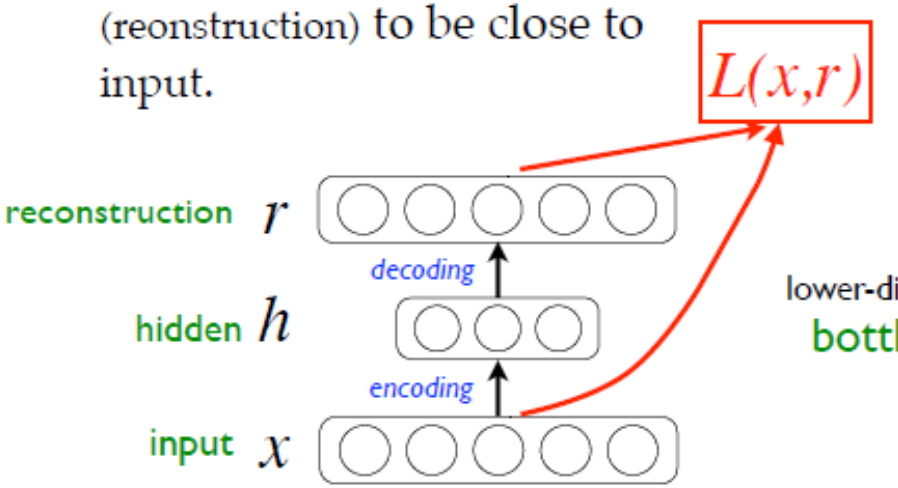
Autoencoders

Autoencoders

- Universal approximators
  - So are Restricted Boltzmann Machines
- Unsupervised learning
- Dimensionality reduction
- $\mathbf{x} \in \mathbb{R}^D \Rightarrow \mathbf{h} \in \mathbb{R}^M$ s.t. $M < D$

Autoencoders: MLPs used for «unsupervised» representation learning

- Make output layer same size as input layer
- Have target = input
- Loss encourages output (reonstruction) to be close to input.



Autoencoders are also called

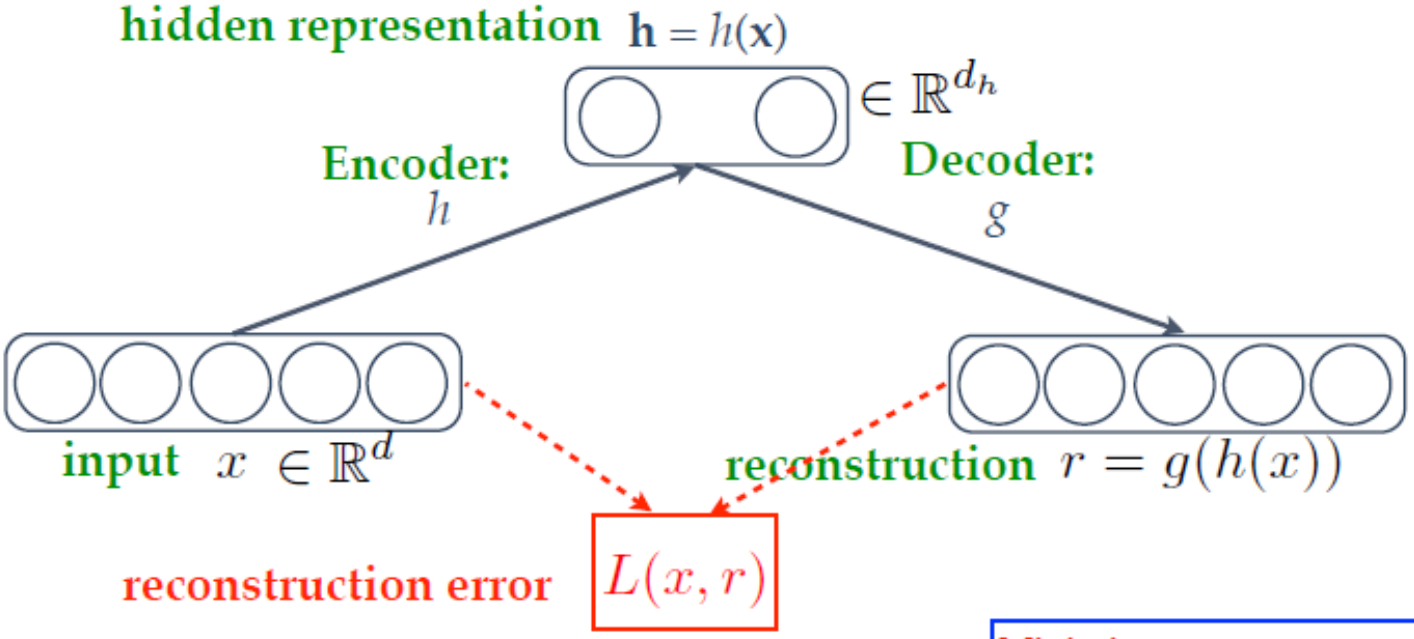
- Autoencoders
- Auto-associators
- Diabolo networks
- Sandglass-shaped net

lower-dimensional bottleneck

The Diabolo

(Pascal Vincent)

Auto-Encoders (AE)
for learning representations



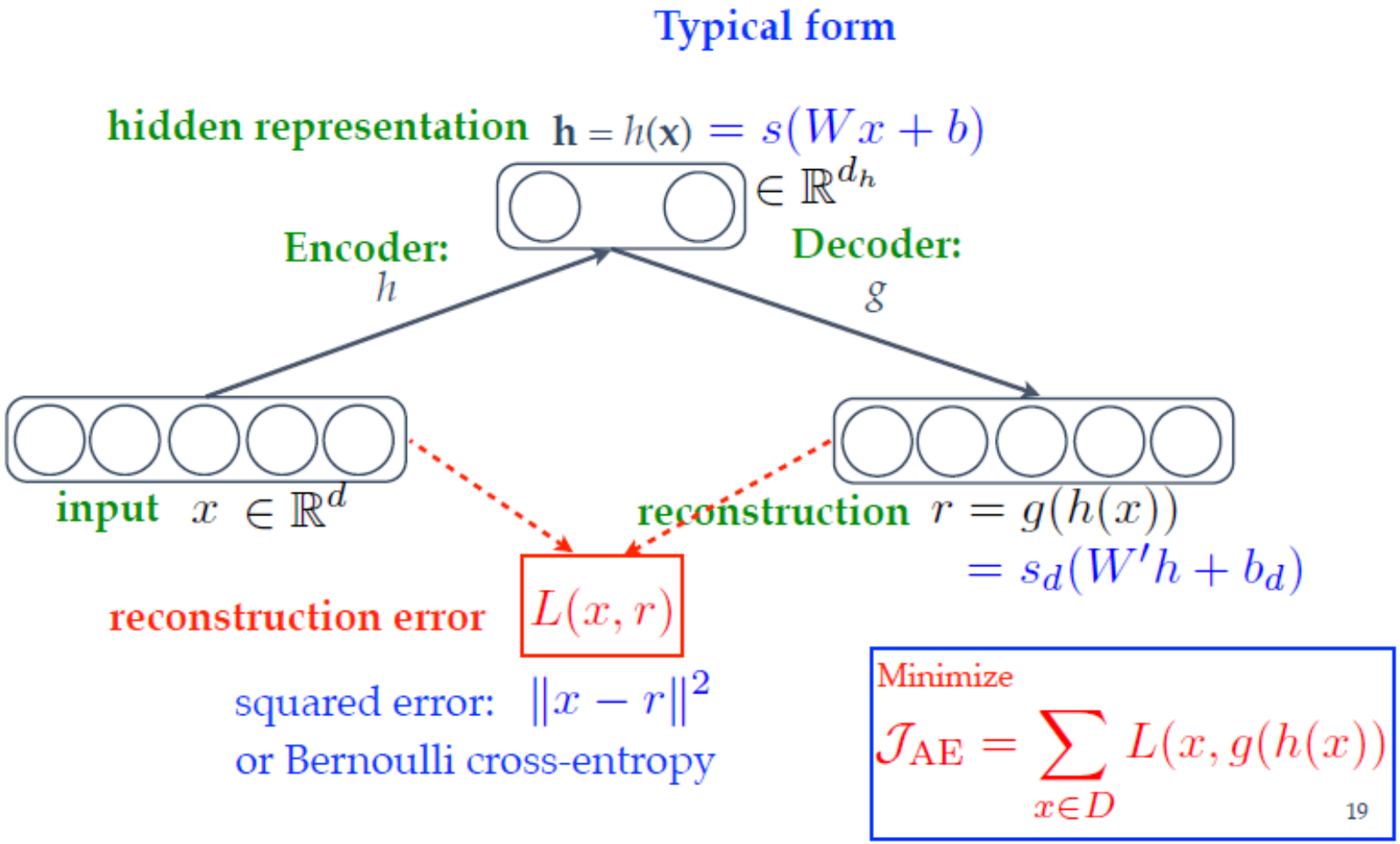
Minimize

$$\mathcal{J}_{\text{AE}} = \sum_{x \in D} L(x, g(h(x)))$$

19

(Pascal Vincent)

Auto-Encoders (AE)
for learning representations



(Pascal Vincent)

conection between

Linear auto-encoders and PCA

$d_h < d$ (bottleneck, undercomplete representation):

- With linear neurons and squared loss
  ➡ autoencoder learns same **suspace** as PCA
- Also true with a single sigmoidal hidden layer, if using linear output neurons with squared loss [Baldi& Hornik 89] and untied weights.
- Won't learn the exact same **basis** as PCA, but W will span the same **subspace**.

(Pascal Vincent)

similarity between

Auto-encoders and RBM

Consider an auto-encoder MLP

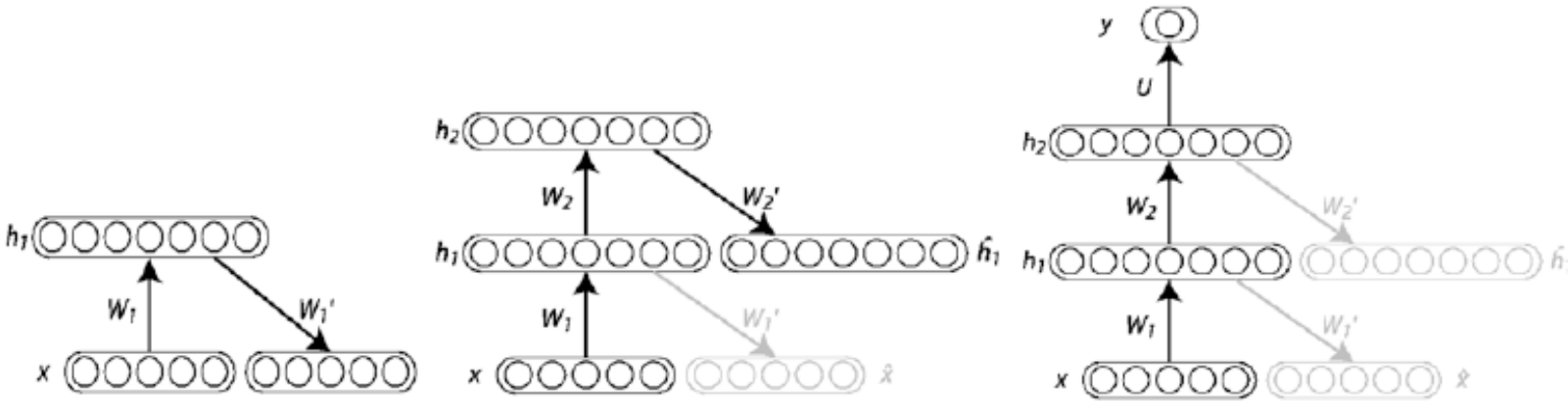
- with a single hidden layer with sigmoid non-linearity
- and sigmoid output non-linerity.
- Tie encoder and decoder weights: $W' = W^T$.

	Autoencoder:	RBM:
	$h_i = s(W_i x + b_i)$	$P(h_i=1 \mid v) = s(W_i v + c_i)$
	$r_j = s(W_j^T h + b_{dj})$	$P(v_j=1 \mid h) = s(W_j^T h + b_j)$
Differences:	deterministic mapping h is a function of x.	stochastic mapping h is a random variable

(Pascal Vincent)

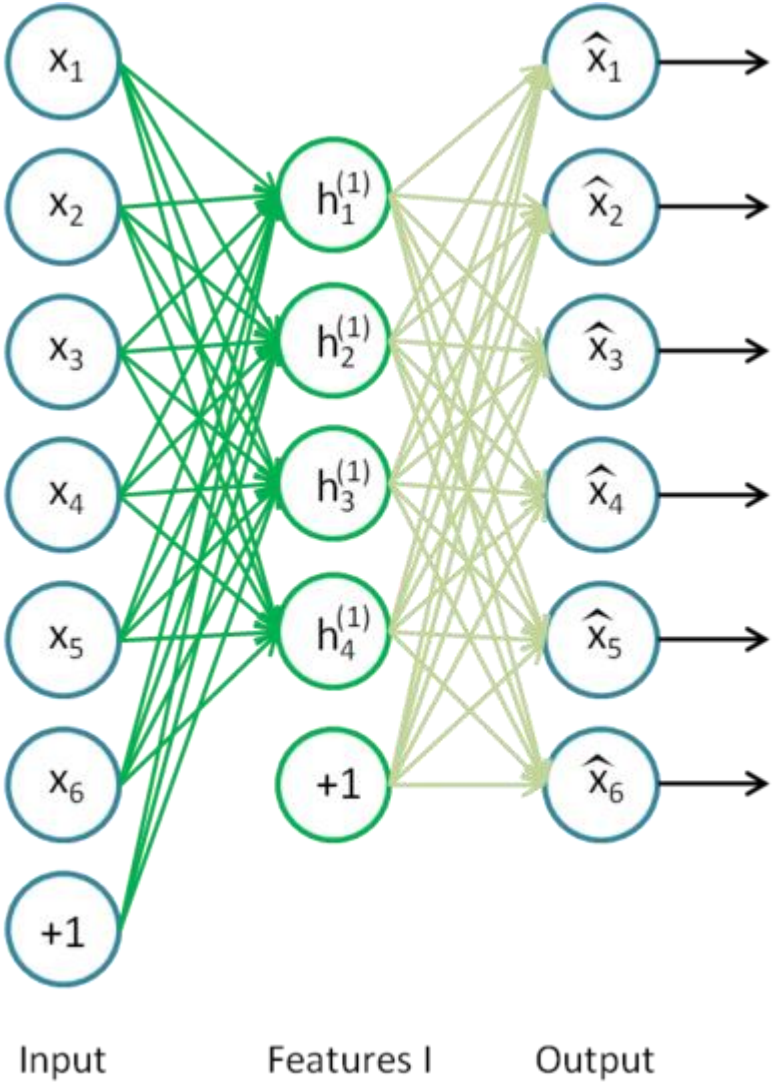
Greedy Layer-Wise Pre-training with Auto-Encoders

Stacking basic Auto-Encoders [Bengio et al. 2007]



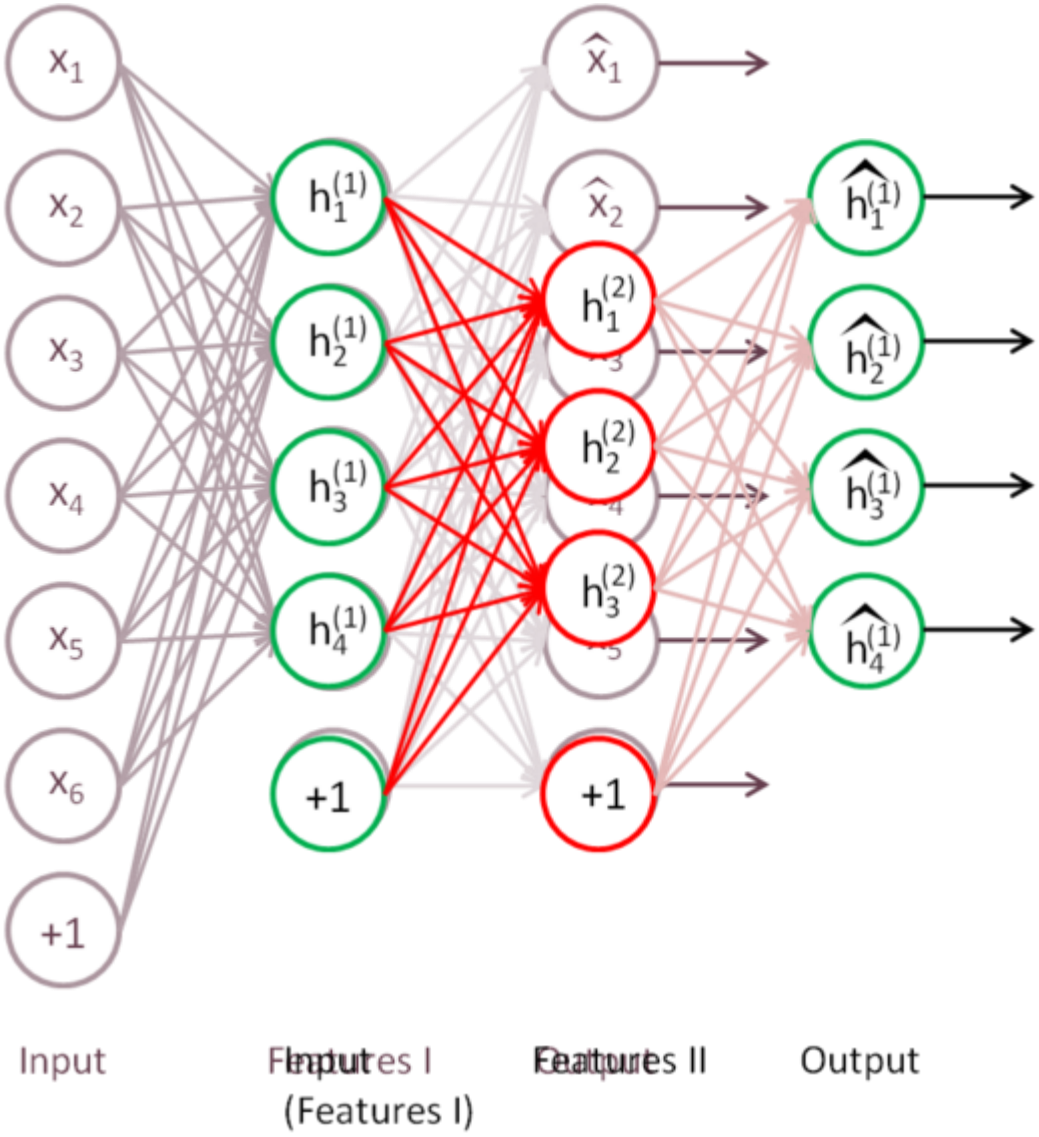
(Pascal Vincent)

Stacking autoencoders:
learn the first layer



http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

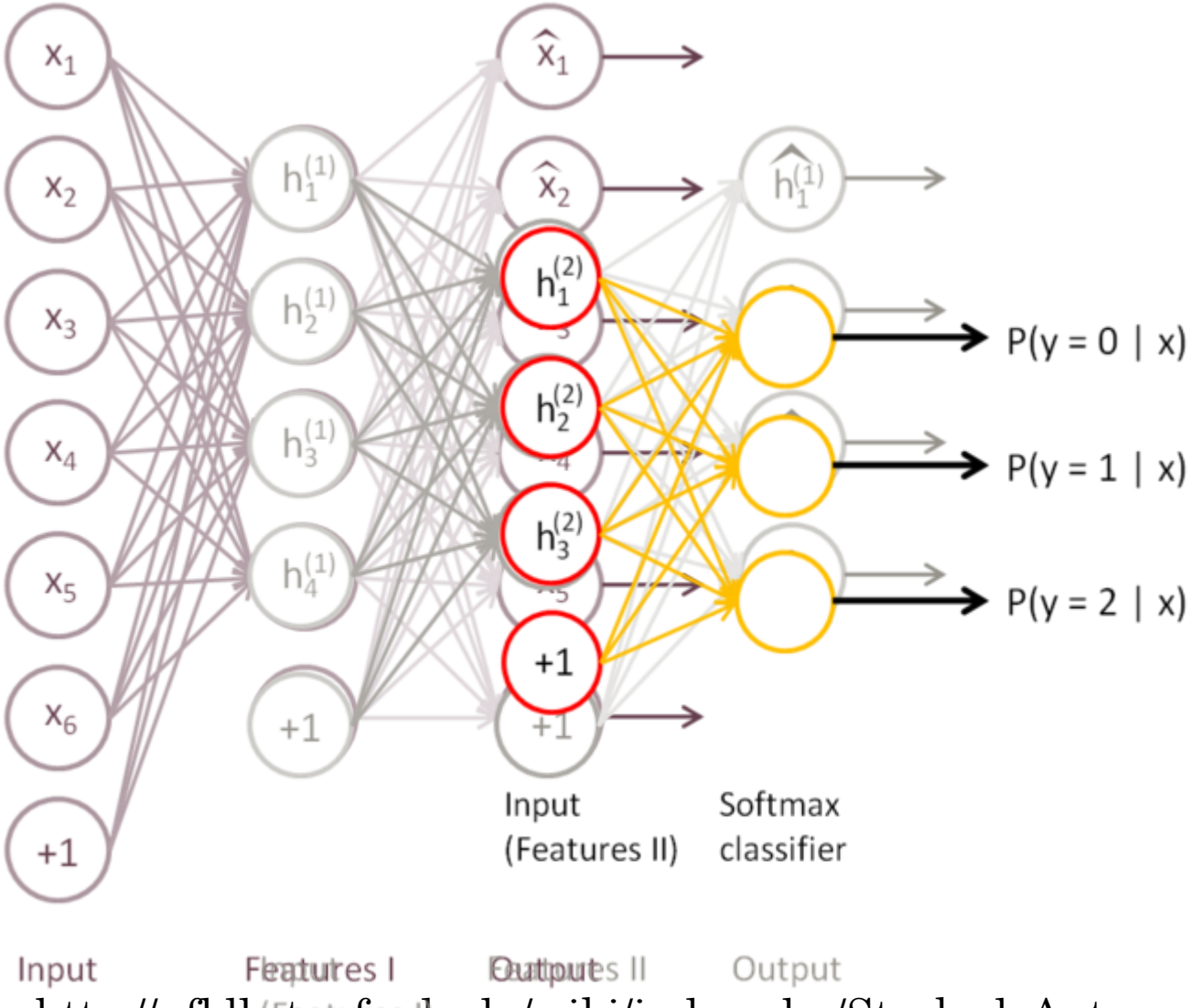
Stacking autoencoders: learn the second layer



http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

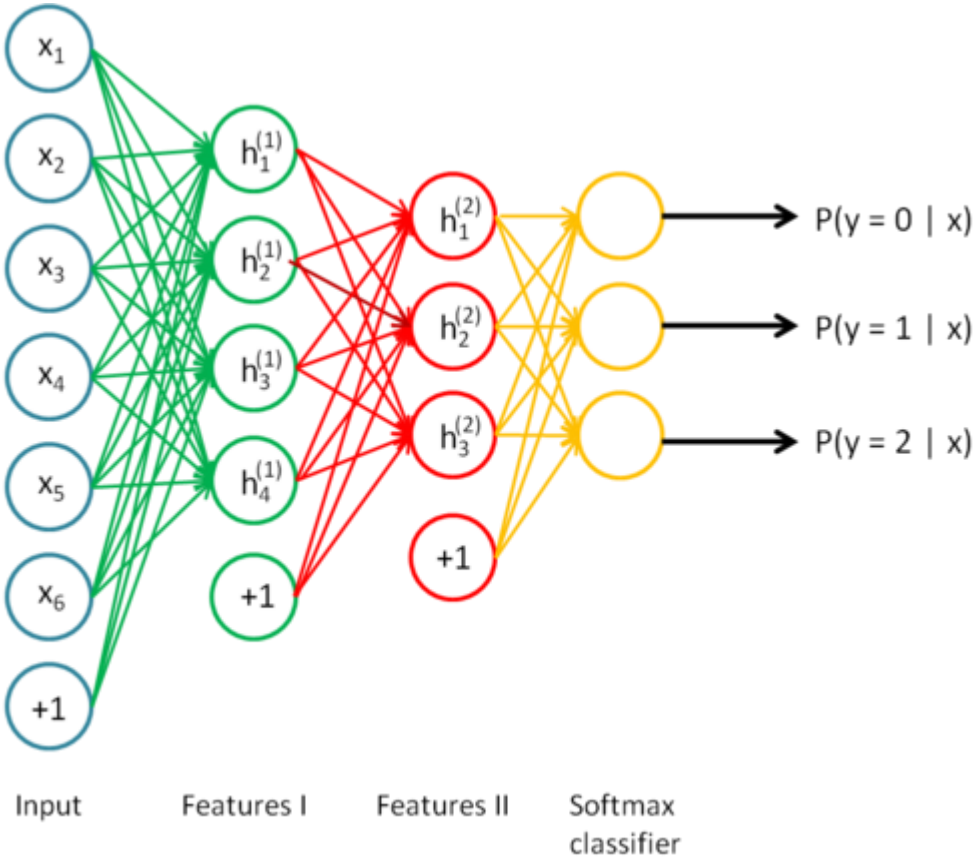
Stacking autoencoders:

Add, e.g., a softmax layer for mapping to output



http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

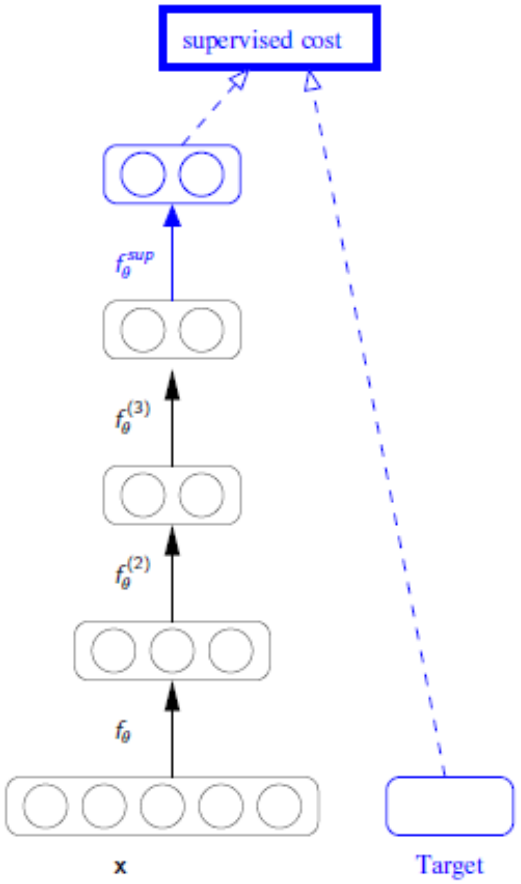
Stacking autoencoders: Overall



http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

Supervised fine-tuning

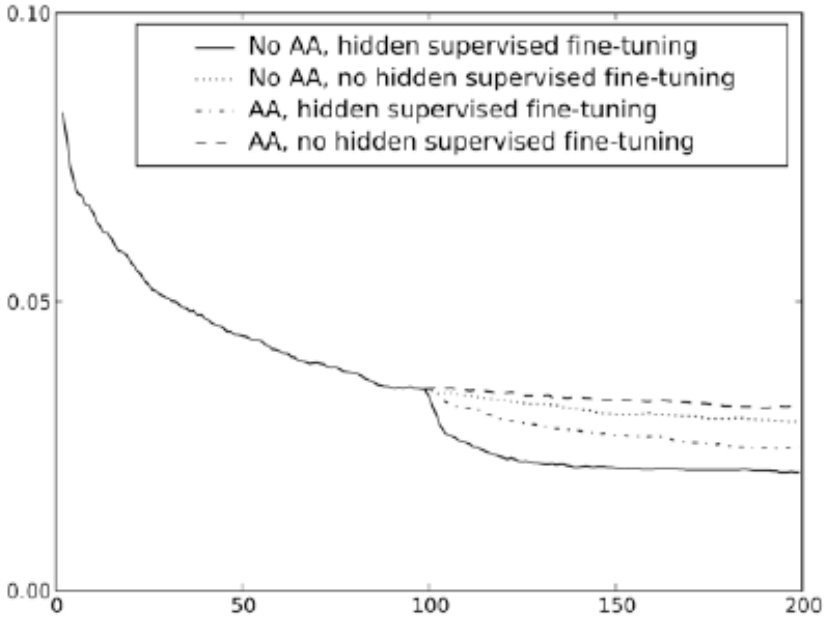
- Initial deep mapping was learnt in an unsupervised way.
- → initialization for a supervised task.
- Output layer gets added.
- Global fine tuning by gradient descent on supervised criterion.



(Pascal Vincent)

Supervised Fine-Tuning is Important

- Greedy layer-wise unsupervised pre-training phase with RBMs or auto-encoders on MNIST
- Supervised phase with or without unsupervised updates, with or without fine-tuning of hidden layers



Classiffication performance on benchmarks:

- Pre-training basic auto-encoder stack better than no pre-training
- Basic auto-encoder stack **almost** matched RBM stack...

(Pascal Vincent)

Basic auto-encoders not as good feature learners as RBMs...

What's the problem?

- Traditional autoencoders were for **dimensionality reduction** ($d_h < d_x$)
- Deep learning success seems to depend on ability to learn **overcomplete representations** ($d_h > d_x$)
- Overcomplete basic autoencoder yields trivial *useless* solutions: identity mapping!
- Need for alternative **regularization/ constraining**

(Pascal Vincent)

Making auto-encoders learn over-complete representations

That are not one-to-one mappings

Wait, what do we mean by over-complete?

- Remember distributed representations?

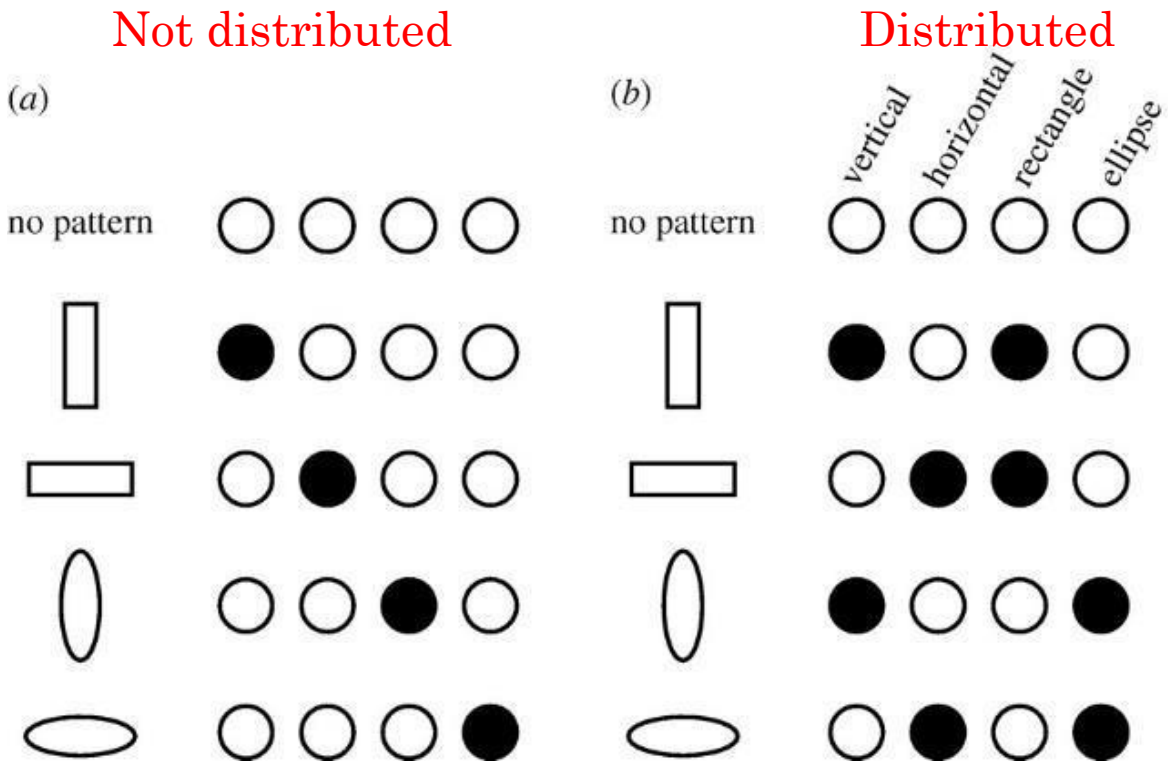
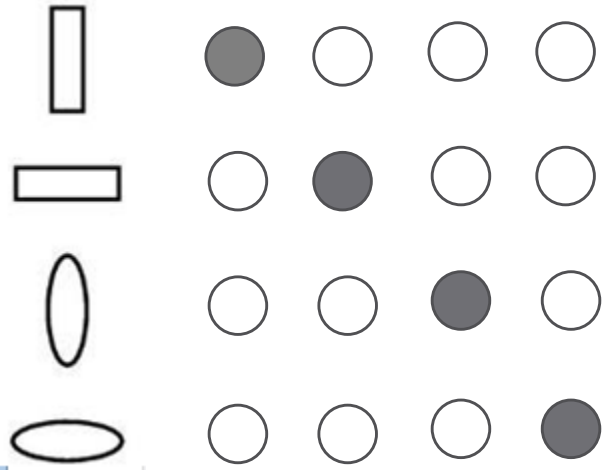


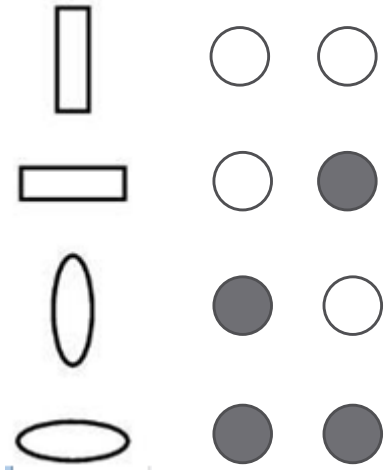
Figure Credit: Moontae Lee

Distributed vs. undercomplete vs. overcomplete representations

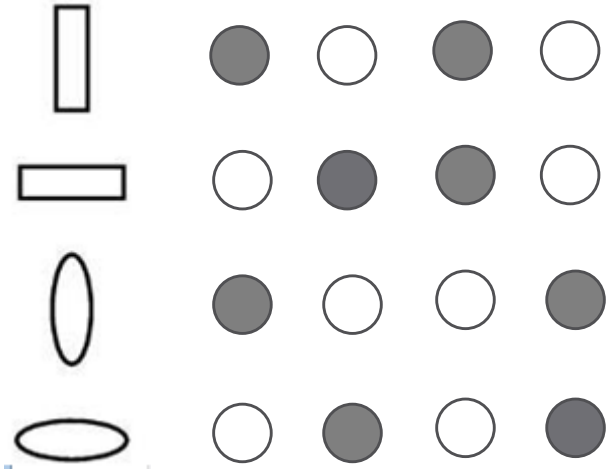
- Four categories could also be represented by two neurons:



Not Distributed
(over complete)



Distributed
(Under complete)



Distributed
(Over complete)

Over-complete = sparse (in distributed representations)

- Why sparsity?

1. Because our brain relies on sparse coding.
   - Why does it do so?

   a. Because it is adapted to an environment which is composed of and can be sensed through the combination of primitive items/entities.

   b. *"Sparse coding may be a general strategy of neural systems to augment memory capacity. To adapt to their environments, animals must learn which stimuli are associated with rewards or punishments and distinguish these reinforced stimuli from similar but irrelevant ones. Such task requires implementing stimulus-specific associative memories in which only a few neurons out of a population respond to any given stimulus and each neuron responds to only a few stimuli out of all possible stimuli."*

   – Wikipedia

   c. Theoretically, it has shown that it increases capacity of memory.

Over-complete = sparse (in distributed representations)

- Why sparsity?

2. Because of information theoretical aspects:
   - Sparse codes have lower entropy compared to non-sparse one.

3. It is easier for the consecutive layers to learn from sparse codes, compared to non-sparse ones.

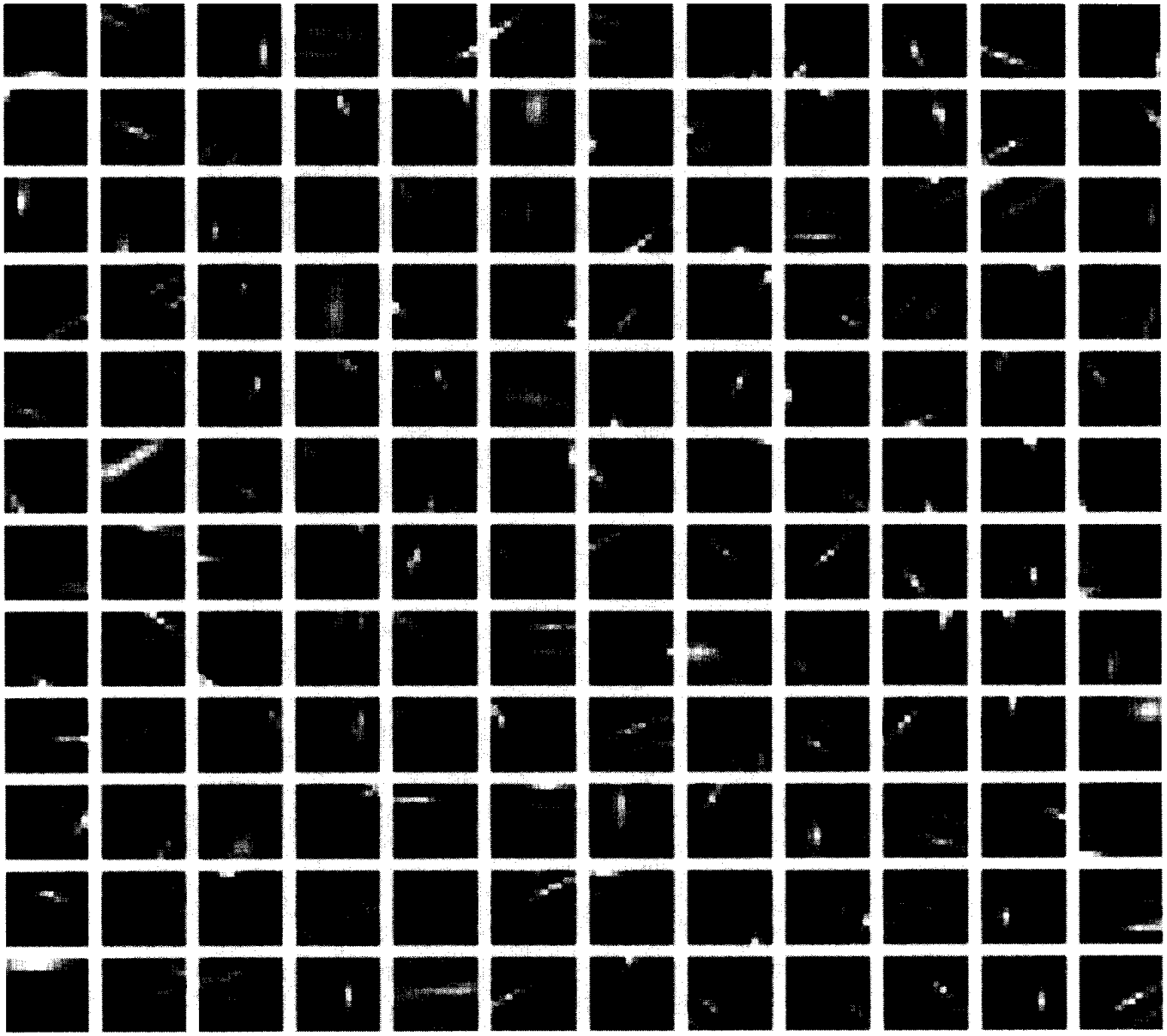


FIGURE 7. The set of 144 basis functions learned by the sparse coding algorithm. The basis functions are totally overlapping (i.e., the entire set codes for the same image patch). All have been normalized to fill the grey scale, but with zero always represented by the same grey level.

Olshausen & Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?", 1997

Mechanisms for enforcing over-completeness

- Use stochastic gradient descent
- Add sparsity constraint
  - Into the loss function (sparse autoencoder)
  - Or, in a hard manner (k-sparse autoencoder)
- Add stochasticisity / randomness
  - Add noise: Denoising Autoencoders, Contraction Autoencoders
  - Restricted Boltzmann Machines

Auto-encoders with SGD

Simple neural network

- Input: $\mathbf{x} \in \boldsymbol{R}^{\boldsymbol{n}}$
- Hidden layer: $\mathbf{h} \in \boldsymbol{R}^{\boldsymbol{m}}$

$$\mathbf{h} = \boldsymbol{f}_1(\boldsymbol{W}_1\mathbf{x})$$

- Output layer: $\mathbf{y} \in \boldsymbol{R}^{\boldsymbol{n}}$

$$\mathbf{y} = \boldsymbol{f}_2\big(\boldsymbol{W}_2\boldsymbol{f}_1(\boldsymbol{W}_1\mathbf{x})\big)$$

- Squared-error loss:

$$L = \frac{1}{2}\sum_{d\in D}\left|\left|\mathbf{x}_d - \mathbf{y}_{\boldsymbol{d}}\right|\right|^2$$

- For training, use SGD.
- You may try different activation functions for f_1 and f_2.

Sparse Autoencoders

Sparse autoencoders

- Input: $\mathbf{x} \in \boldsymbol{R}^{\boldsymbol{n}}$
- Hidden layer: $\mathbf{h} \in \boldsymbol{R}^{\boldsymbol{m}}$

$$\mathbf{h} = \boldsymbol{f}_1(\boldsymbol{W}_1\mathbf{x})$$

- Output layer: $\mathbf{y} \in \boldsymbol{R}^{\boldsymbol{n}}$

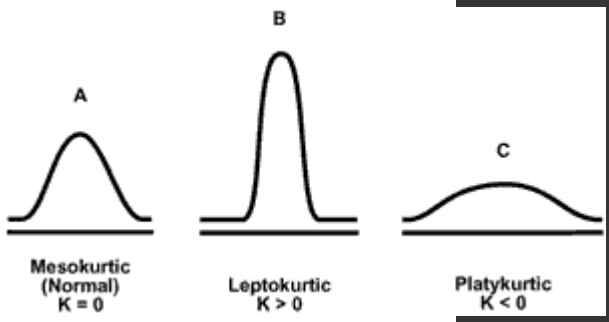
$$\mathbf{y} = \boldsymbol{f}_2\big(\boldsymbol{W}_2\boldsymbol{f}_1(\boldsymbol{W}_1\mathbf{x})\big)$$

Over-completeness and sparsity:

- Require
  - $m > n$, and
  - Hidden neurons to produce only little activation for any input ➔ i.e., sparsity.
- How to enforce sparsity?

Enforcing sparsity: alternatives

- How?
- Solution 1: $\lambda |w|$
  - We have seen before that this enforces sparsity.
  - However, this is not strong enough.
- Solution 2
  - Limit on the amount of average total activation for a neuron throughout training!
- Solution 3
  - Kurtosis: $\frac{\mu_4}{\sigma^4} = \frac{E[(X-\mu)^4]}{(E[(X-\mu)^2])^2}$
  - Calculated over the activations of the whole network.
  - High kurtosis ➔ sparse activations.
  - "Kurtosis has only been studied for response distributions of model neurons where negative responses are allowed. It is unclear whether kurtosis is actually a sensible measure for realistic, non-negative response distributions." - http://www.scholarpedia.org/article/Sparse_coding
- And many many other ways…



Enforcing sparsity: a popular choice

- Limit the amount of total activation for a neuron throughout training!
- Use ρ_i to denote the activation of neuron x on input i. The average activation of the neuron over the training set:

$$\hat{\rho}_i = \frac{1}{m}\sum_{i}^{m} \rho_i$$

- Now, to enforce sparsity, we limit to $\hat{\rho}_i = \rho_0$.
- ρ_0: A small value.
  - Yet another hyperparameter which may be tuned.
  - typical value: 0.05.
- The neuron must be inactive most of the time to keep its activations under the limit.

Enforcing sparsity

$$\hat{\rho}_i = \frac{1}{m}\sum_{i}^{m} \rho_i$$

- How to limit $\hat{\rho}_i = \rho_0$? How do we add integrate this as a penalty term into the loss function?
- Use Kullback-Leibler divergence:

$$\sum_i KL(\rho_0 \,||\, \hat{\rho}_i)$$

Or, equivalently as (since this is between two Bernoulli variables with mean ρ_0 and $\hat{\rho}_i$):

$$\sum_i \rho_0 \log\frac{\rho_0}{\hat{\rho}_i} + (1-\rho_0)\log\frac{1-\rho_0}{1-\hat{\rho}_i}$$

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}.$$

Backpropagation and training

$$S = \beta \sum_i \rho_0 \log \frac{\rho_0}{\hat{\rho}_i} + (1-\rho_0) \log \frac{1-\rho_0}{1-\hat{\rho}_i}$$

$$\frac{dS}{d\rho_i} = \beta \left(-\rho_0 \frac{1}{\hat{\rho}_i \ln 10} + (1-\rho_0) \frac{1}{(1-\hat{\rho}_i) \ln 10} \right)$$

- If you use ln in KL:

$$\frac{dS}{d\rho_i} = \beta \left(-\frac{\rho_0}{\hat{\rho}_i} + \frac{1-\rho_0}{1-\hat{\rho}_i} \right)$$

- So, if we integrate into the original error term:

$$\delta_h = o_h(1-o_h).\left(\left(\sum_k w_{kh} \delta k \right) + \beta \left(-\frac{\rho_0}{\hat{\rho}_i} + \frac{1-\rho_0}{1-\hat{\rho}_i} \right) \right)$$

- Need to change $o_h(1-o_h)$ if you use a different activation function.

Reminder

-For each hidden unit h, calculate its error term δ_h:

$$\delta_h = o_h(1-o_h) \sum_{k \in outputs} w_{kh} \delta_k$$

-Update every weight w_{ji}

$$w_{ji} = w_{ji} + \eta \delta_j x_{ji}$$

Backpropagation and training

$$S = \beta \sum_i \rho_0 \log \frac{\rho_0}{\hat{\rho}_i} + (1 - \rho_0) \log \frac{1 - \rho_0}{1 - \hat{\rho}_i}$$

- Do you see a problem here?
- $\hat{\rho}_i$ should be calculated over the training set.
- In other words, we need to go through the whole dataset (batch) once to calculate $\hat{\rho}_i$.

Loss & decoders & encoders

- Be careful about the range of your activations and the range of the output
- Real-valued input:
  - Encoder: use sigmoid
  - Decoder: no need for non-linearity.
  - Loss: Squared-error Loss
  - Vincent et al. (2010):

For real-valued $\mathbf{x}$, that is, $\mathbf{x} \in \mathbb{R}^d$: $X|\mathbf{z} \sim \mathcal{N}(\mathbf{z}, \sigma^2\mathbf{I})$, that is, $X_j|\mathbf{z} \sim \mathcal{N}(\mathbf{z}_j, \sigma^2)$.
This yields $L(\mathbf{x}, \mathbf{z}) = L_2(\mathbf{x}, \mathbf{z}) = C(\sigma^2)\|\mathbf{x} - \mathbf{z}\|^2$ where $C(\sigma^2)$ denotes a constant that depends only on σ^2 and that can be ignored for the optimization.

- Binary-valued input:
  - Encoder: use sigmoid.
  - Decoder: use sigmoid.
  - Loss: use cross-entropy loss:

$$-\sum_j [\mathbf{x}_j \log \mathbf{z}_j + (1 - \mathbf{x}_j) \log(1 - \mathbf{z}_j)]$$

Loss & decoders & encoders

- Kullback-Leibler divergence assumes that the variables are in the range [0,1].
  - I.e., you are bound to use sigmoid for the hidden layer if you use KL to limit the activations of hidden units.

Why Regularized Auto-Encoders learn Sparse Representation?

Devansh Arpit DEVANSHA@BUFFALO.EDU
Yingbo Zhou YINGBOZH@BUFFALO.EDU
Hung Q. Ngo HUNGNGO@BUFFALO.EDU
Venu Govindaraju GOVIND@BUFFALO.EDU

k-Sparse Autoencoder

- Note that it doesn't have an activation function!
- Non-linearity comes from k-selection.

k-Sparse Autoencoders:

Training:

1) Perform the feedforward phase and compute

$$\boldsymbol{z} = W^\top \boldsymbol{x} + \boldsymbol{b}$$

2) Find the k largest activations of $\boldsymbol{z}$ and set the rest to zero.

$$\boldsymbol{z}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_k(\boldsymbol{z})$$

3) Compute the output and the error using the sparsified $\boldsymbol{z}$.

$$\hat{\boldsymbol{x}} = W\boldsymbol{z} + \boldsymbol{b}'$$
$$E = \|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2$$

3) Backpropagate the error through the k largest activations defined by Γ and iterate.

Sparse Encoding:

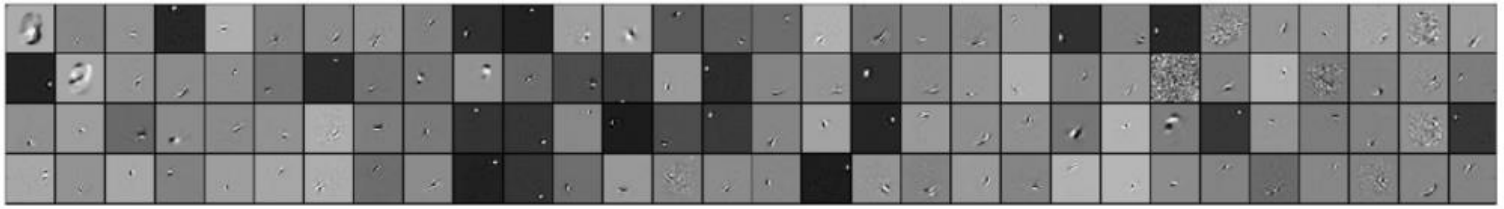
Compute the features $\boldsymbol{h} = W^\top \boldsymbol{x} + \boldsymbol{b}$. Find its αk largest activations and set the rest to zero.

$$\boldsymbol{h}_{(\Gamma)^c} = 0 \quad \text{where} \quad \Gamma = \text{supp}_{\alpha k}(\boldsymbol{h})$$

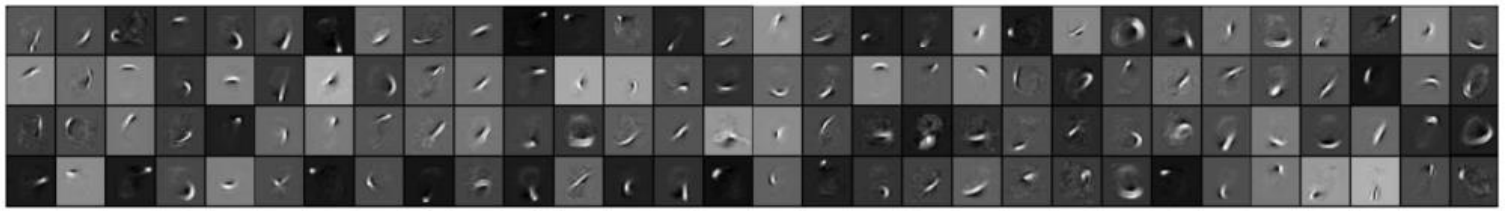
---

k-Sparse Autoencoders

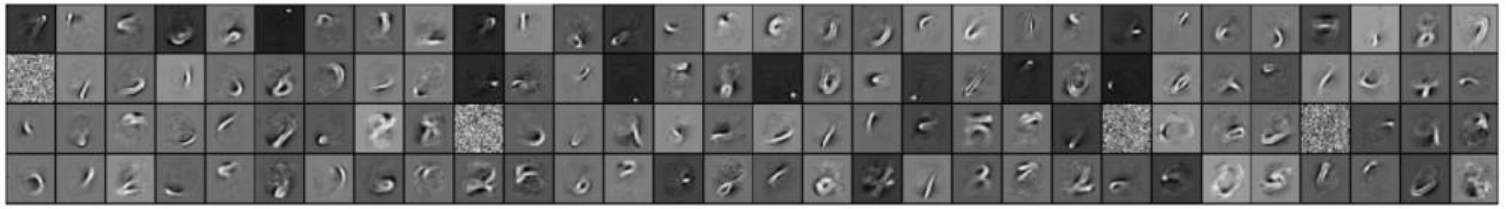
Alireza Makhzani MAKHZANI@PSI.UTORONTO.CA
Brendan Frey FREY@PSI.UTORONTO.CA
University of Toronto, 10 King's College Rd. Toronto, Ontario M5S 3G4, Canada



(a) $k = 70$



(b) $k = 40$



(c) $k = 25$

(d) $k = 10$

http://www.ericlwilkinson.com/blog/2014/11/19/deep-learning-sparse-autoencoders

Denoising Auto-encoders (DAE)

Journal of Machine Learning Research 11 (2010) 3371-3408 Submitted 5/10; Published 12/10

Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion

Pascal Vincent PASCAL.VINCENT@UMONTREAL.CA
Département d'informatique et de recherche opérationnelle
Université de Montréal
2920, chemin de la Tour
Montréal, Québec, H3T 1J8, Canada

Hugo Larochelle LAROCHEH@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario, M5S 3G4, Canada

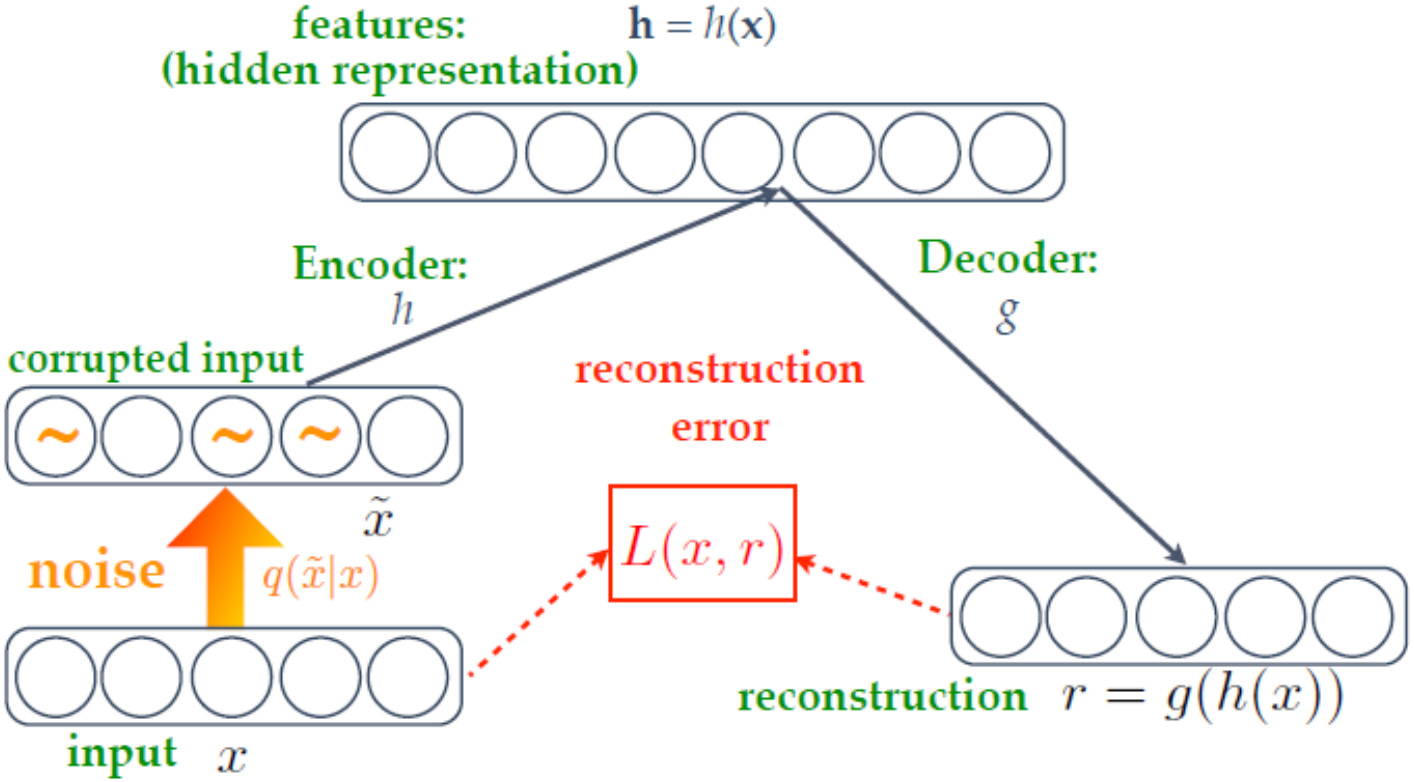
Isabelle Lajoie ISABELLE.LAJOIE.1@UMONTREAL.CA
Yoshua Bengio YOSHUA.BENGIO@UMONTREAL.CA
Pierre-Antoine Manzagol PIERRE-ANTOINE.MANZAGOL@UMONTREAL.CA
Département d'informatique et de recherche opérationnelle
Université de Montréal
2920, chemin de la Tour
Montréal, Québec, H3T 1J8, Canada

Denoising Auto-encoders

- Simple idea:
  - randomly corrupt some of the inputs (as many as half of them) – e.g., set them to zero.
  - Train the autoencoder to *reconstruct the input from a corrupted version of it*.
  - The auto-encoder is to *predict the corrupted (i.e. missing) values from the uncorrupted values*.
  - This requires capturing the joint distribution between a set of variables

- A stochastic version of the auto-encoder.

Denoising auto-encoder (DAE)



- learns robust & useful features
- easier to train than RBM features
- yield similar or better classification performance (as deep net pre-training)

Minimize:

$$\mathcal{J}_{\mathrm{DAE}}(\theta) = \sum_{x \in D} \mathbb{E}_{q(\tilde{x}|x)} \left[L(x, g(h(\tilde{x})) \right]$$

(Pascal Vincent)

Denoising auto-encoder (DAE)

- Autoencoder training minimizes:

$$\mathcal{J}_{\mathrm{AE}}(\theta) = \sum_{x \in D} L(x, g(h(\tilde{x}))$$

- Denoising autoencoder training minimizes

$$\mathcal{J}_{\mathrm{DAE}}(\theta) = \sum_{x \in D} \mathbb{E}_{q(\tilde{x}|x)} \left[L(x, g(h(\tilde{x})) \right]$$

Cannot compute expectation exactly
➪ use stochastic gradient descent,
sampling corrupted inputs $\tilde{x}|x$

Possible corruptions q:

- zeroing pixels at random
  (now called «dropout» noise)
- additive Gaussian noise
- salt-and-pepper noise
- ...

29

(Pascal Vincent)

Loss in DAE

- You may give extra emphasis on "corrupted" dimensions:

$$L_{2,\alpha}(\mathbf{x},\mathbf{z}) = \alpha\left(\sum_{j\in\mathcal{J}(\tilde{\mathbf{x}})}(\mathbf{x}_j-\mathbf{z}_j)^2\right)+\beta\left(\sum_{j\notin\mathcal{J}(\tilde{\mathbf{x}})}(\mathbf{x}_j-\mathbf{z}_j)^2\right),$$

where $\mathcal{J}(\tilde{\mathbf{x}})$ denotes the indexes of the components of $\mathbf{x}$ that were corrupted.

Or, in cross-entropy-based loss:

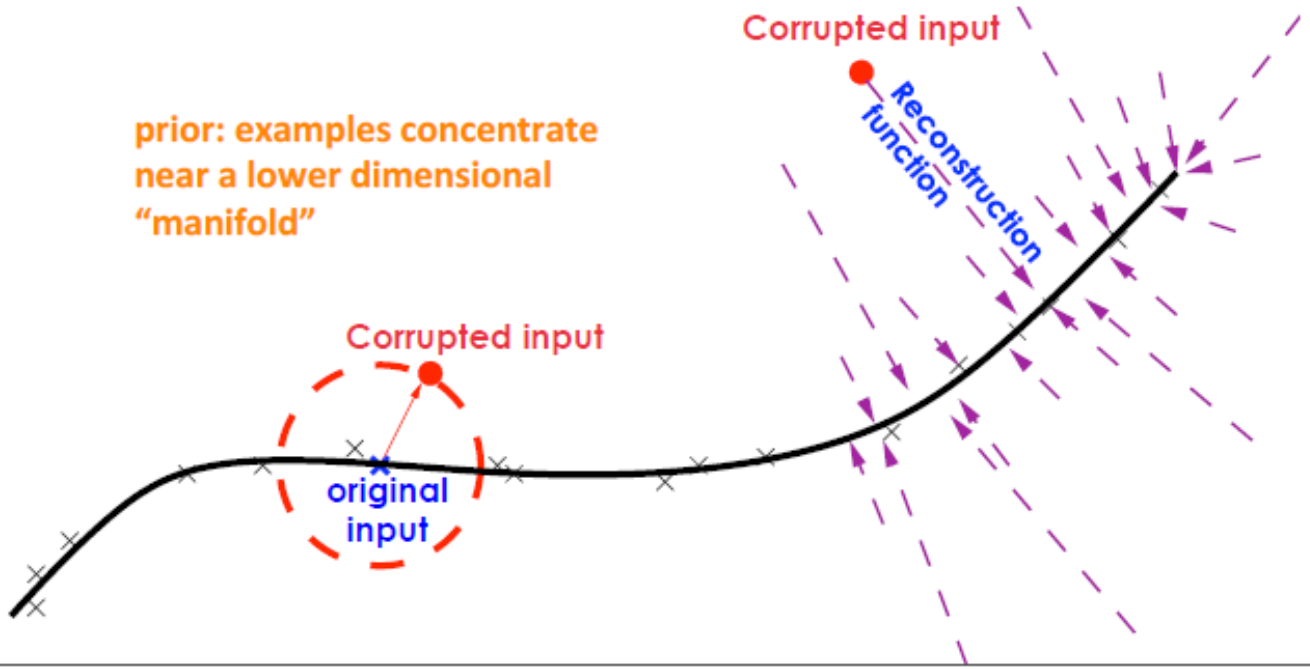
$$\begin{aligned} L_{\mathbb{H},\alpha}(\mathbf{x},\mathbf{z}) &= \alpha\left(-\sum_{j\in\mathcal{J}(\tilde{\mathbf{x}})}[\mathbf{x}_j\log\mathbf{z}_j+(1-\mathbf{x}_j)\log(1-\mathbf{z}_j)]\right) \\ &\quad +\beta\left(-\sum_{j\notin\mathcal{J}(\tilde{\mathbf{x}})}[\mathbf{x}_j\log\mathbf{z}_j+(1-\mathbf{x}_j)\log(1-\mathbf{z}_j)]\right). \end{aligned}$$

Denoising Auto-encoders

- To undo the effect of a corruption induced by the noise, the network needs to capture the statistical dependencies between the inputs.
- This can be interpreted from many perspectives (see Vincent et al., 2008):
  - the manifold learning perspective,
  - stochastic operator perspective.

Denoising auto-encoders: manifold interpretation

- DAE learns to «project back» corrupted input onto manifold.
- Representation $h \approx$ location on the manifold

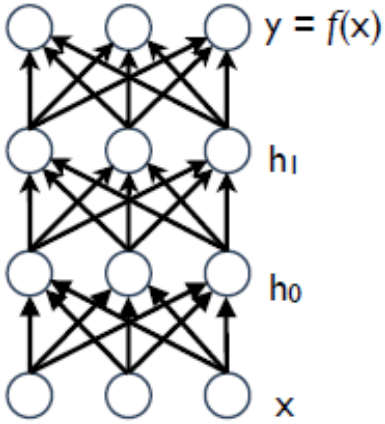


32

mercredi 5 août 2015

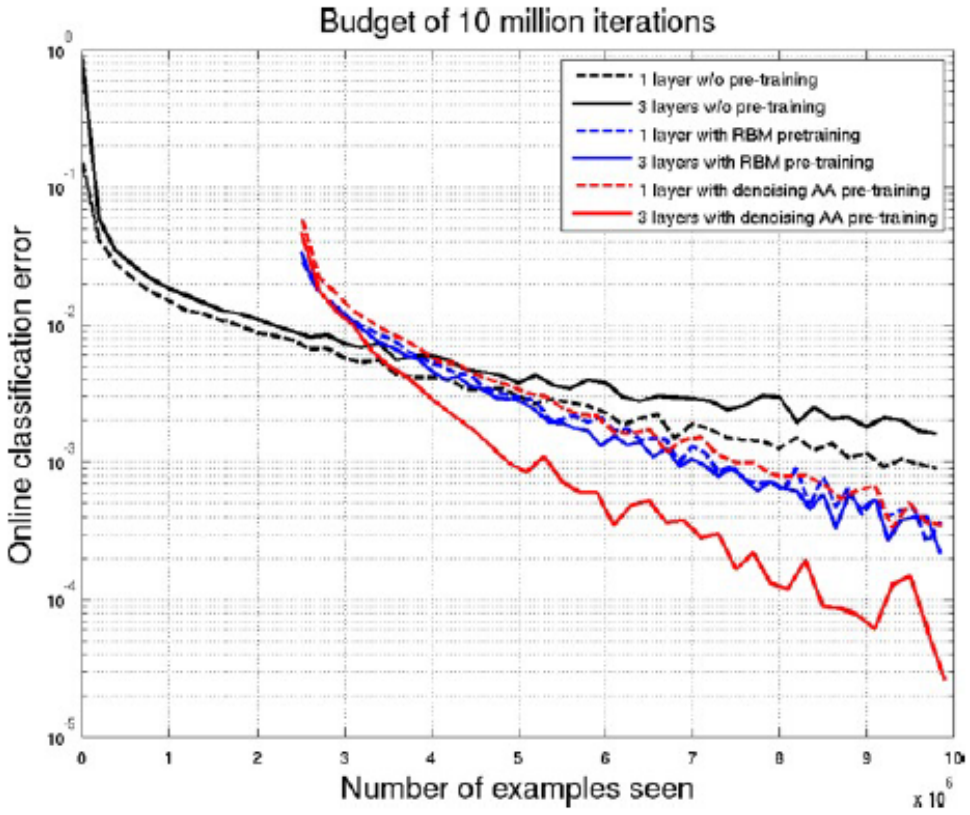
(Pascal Vincent)

Stacked Denoising Auto-Encoders (SDAE)



Advantages over stacking RBMs

- No partition function, can measure training criterion
- Very flexible: encoder & decoder can use any parametrization (more layers...)
- Performs as well or better than stacking RBMs for usupervised pre-training



Infinite MNIST

(Pascal Vincent)

Types of corruption

- Gaussian Noise (additive, isotropic)
- Masking Noise
  - Set a randomly selected subset of input to zero for each sample (the fraction ratio is constant, a parameter)
- Salt-and-pepper Noise:
  - Set a randomly selected subset of input to maximum or minimum for each sample (the fraction ratio is constant, a parameter)

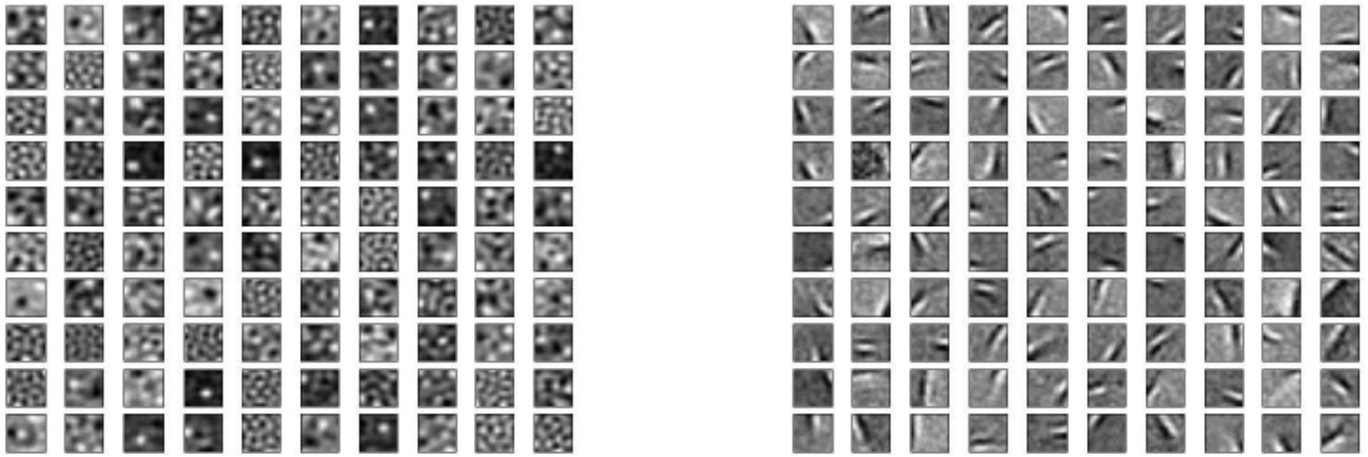


Figure 6: Weight decay vs. Gaussian noise. We show typical filters learnt from natural image patches in the over-complete case (200 hidden units). *Left:* regular autoencoder with weight decay. We tried a wide range of weight-decay values and learning rates: filters never appeared to capture a more interesting structure than what is shown here. Note that some local blob detectors are recovered compared to using no weight decay at all (Figure 5 right). *Right:* a denoising autoencoder with additive Gaussian noise ($\sigma = 0.5$) learns Gabor-like local oriented edge detectors. Clearly the filters learnt are qualitatively very different in the two cases.

Weight decay: L2 regularization.

(Vincent et al., 2010)

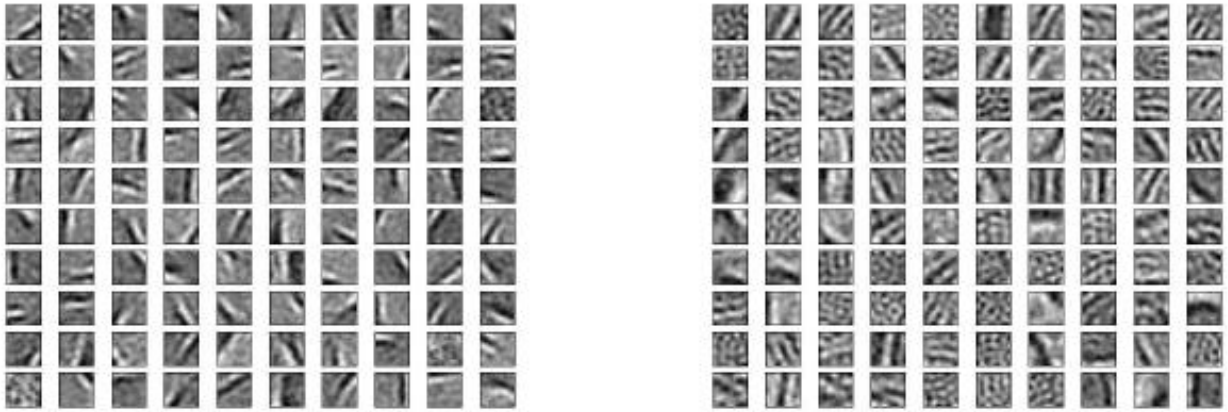


Figure 7: Filters obtained on natural image patches by denoising autoencoders using other noise types. *Left:* with 10% salt-and-pepper noise, we obtain oriented Gabor-like filters. They appear slightly less localized than when using Gaussian noise (contrast with Figure 6 right). *Right:* with 55% zero-masking noise we obtain filters that look like oriented gratings. For the three considered noise types, denoising training appears to learn filters that capture meaningful natural image statistics structure.

(Vincent et al., 2010)

Training DAE

- Training algorithm does not change
  - However, you may give different emphasis on the error of reconstruction of the corrupted input.
- SGD is a popular choice
- Sigmoid is a suitable choice unless you know what you are doing.
- Using "better" activation functions like ReLU is problematic.

Contractive Auto-encoder

Encouraging representation to be insensitive to corruption

- DAE encourages **reconstruction** to be insensitive to input corruption
- Alternative: encourage **representation** to be **insensitive**

$$\mathcal{J}_{\text{SCAE}}(\theta) = \sum_{x \in D} L(x, g(h(x))) + \lambda \mathbb{E}_{q(\tilde{x}|x)} \left[\|h(x) - h(\tilde{x})\|^2 \right]$$

Reconstruction error

stochastic regularization term

34

(Pascal Vincent)

From stochastic to analytic penalty

- SCAE stochastic regularization term: $\mathbb{E}_{q(\tilde{x}|x)}\left[\|h(x) - h(\tilde{x})\|^2\right]$
- For small additive noise $\tilde{x}|x = x + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$
- Taylor series expansion yields $h(x+\epsilon) = h(x) + \frac{\partial h}{\partial x}\epsilon + \ldots$
- It can be showed that

$$\underbrace{\mathbb{E}_{q(\tilde{x}|x)}\left[\|h(x) - h(\tilde{x})\|^2\right]}_{\text{stochastic (SCAE)}} \approx \underbrace{\sigma^2 \left\|\frac{\partial h}{\partial x}(x)\right\|_F^2}_{\text{analytic (CAE)}}$$

35

(Pascal Vincent)

Contractive Auto-Encoder (CAE)

(Rifai, Vincent, Muller, Glorot, Bengio, ICML 2011)

- Minimize $\mathcal{J}_{\text{CAE}} = \sum_{x \in D}^{n} L(x, g(h(x))) + \lambda \left\| \frac{\partial h(x)}{\partial x} \right\|^2$

Reconstruction error

analytic contractive term

- For training examples, encourages both:
  - ➡small reconstruction error
  - ➡representation insensitive to small variations around example

(Pascal Vincent)

Computational considerations
CAE for a simple encoder layer

We defined $\mathbf{h} = h(\mathbf{x}) = s(\underbrace{Wx + b}_{a})$

Further suppose: s is an elementwise non-linearity
s' its first derivative.

Let $J(x) = \frac{\partial h}{\partial x}(x)$

$J_j = s'(b + x^T W_j) W_j$ where J_j and W_j represent j[th] row

CAE penalty is: $\|J\|_F^2 = \sum_{j=1}^{d_h} s'(a_j)^2 \|W_j\|^2$

Compare to L2 weight decay: $\|W\|_F^2 = \sum_{j=1}^{d_h} \|W_j\|^2$

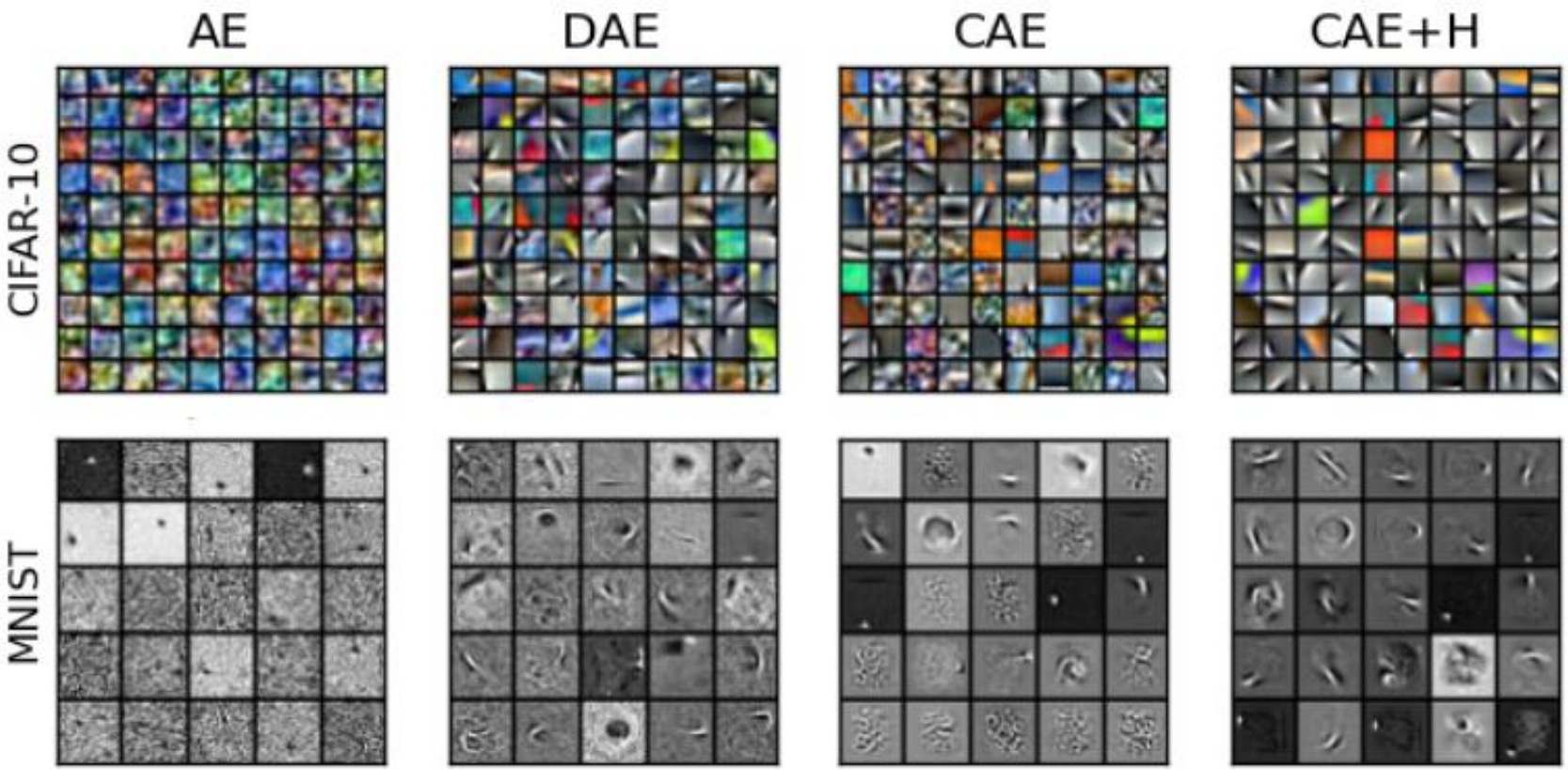
Same complexity: $O(d_h\, d)$

Gradient backprop wrt parameters: $O(d_h\, d)$

37

(Pascal Vincent)

Learned filters



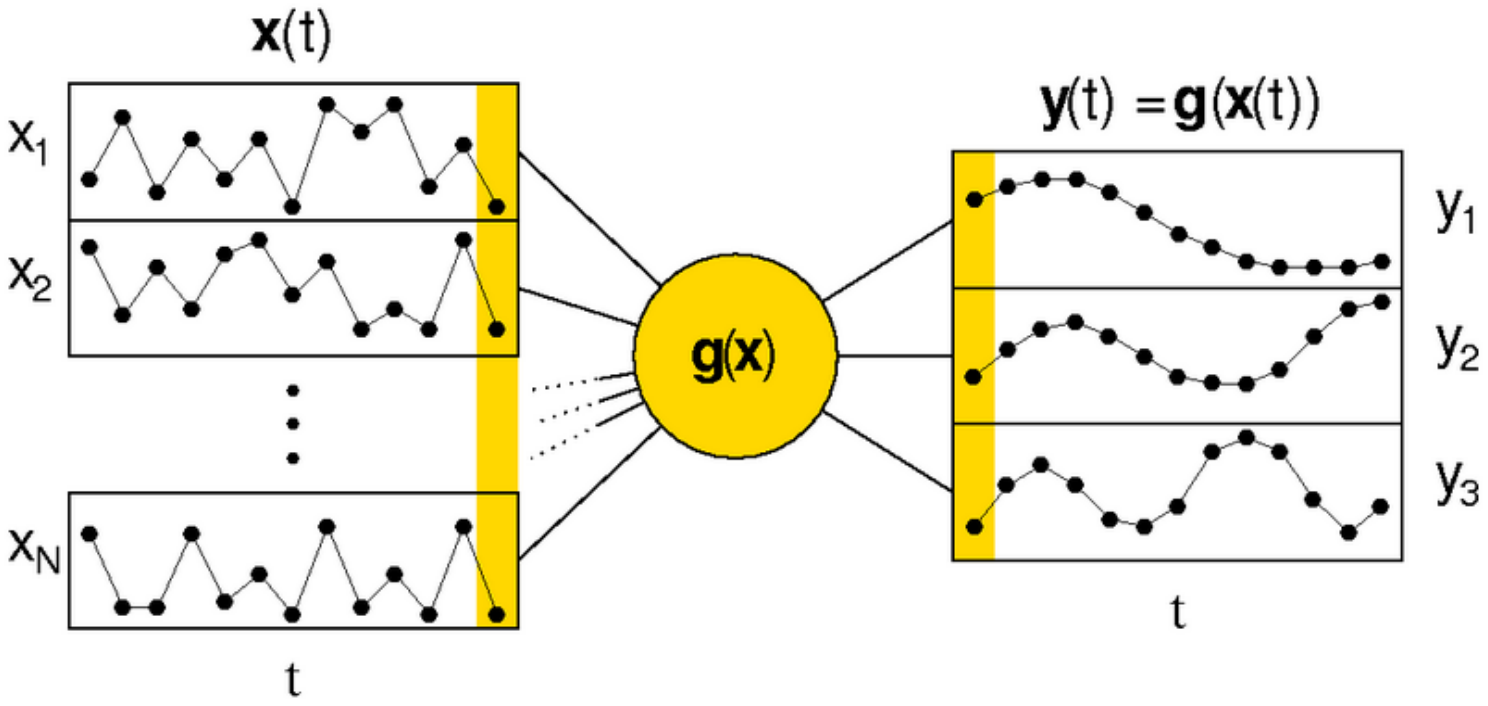
39

mercredi 5 août 2015

(Pascal Vincent)

Principles other than ‘sparsity’?

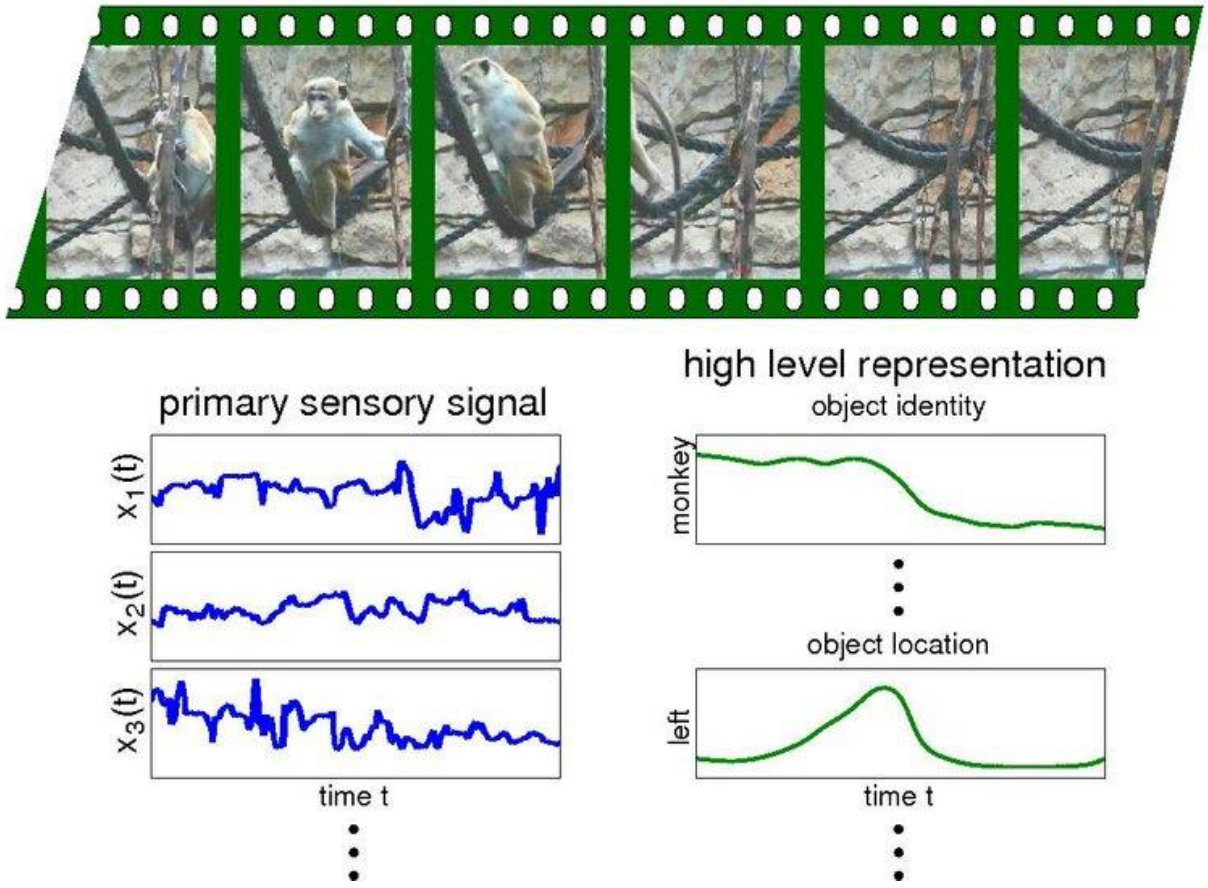
Slowness



http://www.scholarpedia.org/article/Slow_feature_analysis

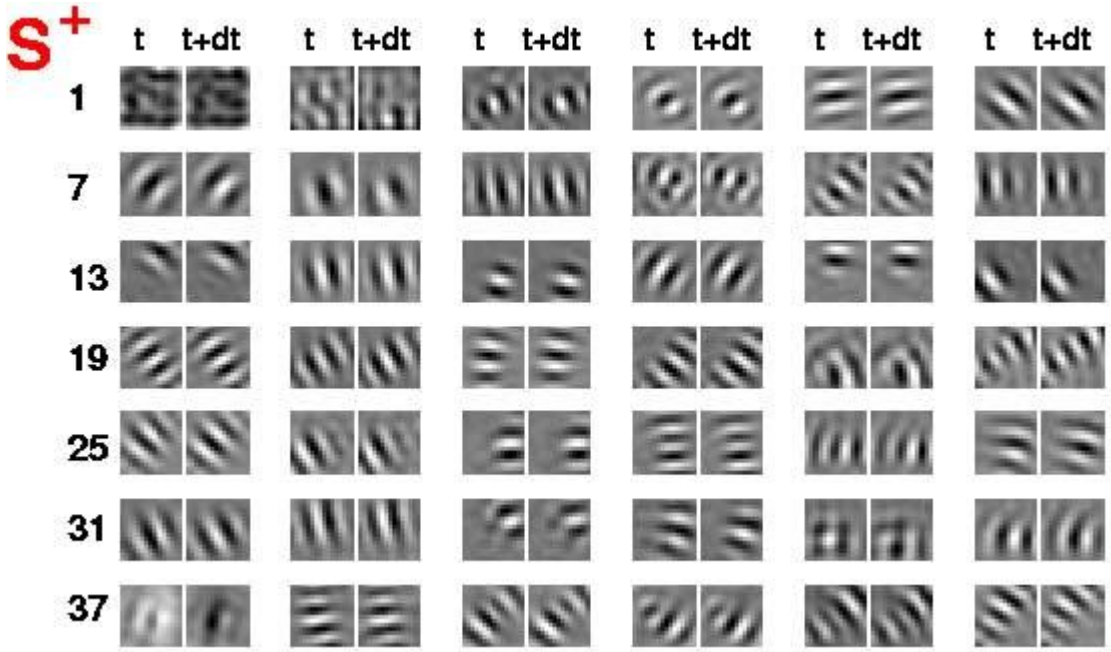
Slow Feature Analysis (SFA)

from Wiskott et al.



http://www.scholarpedia.org/article/Slow_feature_analysis

Slow Feature Analysis (SFA)



Optimal stimuli for the slowest components extracted from natural image sequences.

http://www.scholarpedia.org/article/Slow_feature_analysis

Visualizing the layers

Visualizing the layers

- Question: What is the input that activates a hidden unit h_i most?
  - i.e., we are after $\mathbf{x}^*$:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\ s.t.||\mathbf{x}||=\rho} h_i(W, \mathbf{x})$$

- For the first layer:

$$x_j = \frac{w_{ij}}{\sqrt{\sum_k (w_{ik})^2}}$$

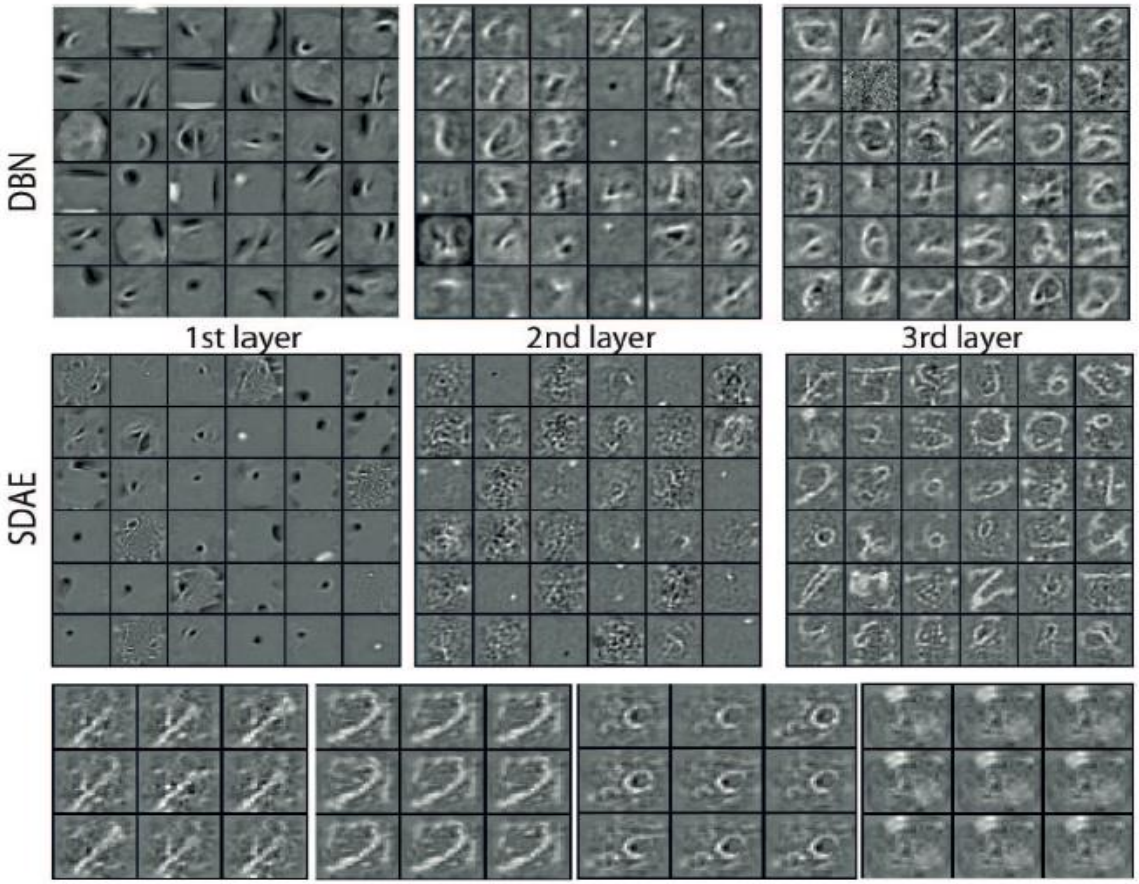
  where we assume that $\sum_i x_i^2 \leq 1$, and hence normalize the weights to match the range of the input values.

- How about the following layers?
  - Gradient ascent (not descent): find the gradient of $h_i(W, \mathbf{x})$ w.r.t $\mathbf{x}$ and move $\mathbf{x}$ in the direction of the gradient since we want to maximize $h_i(W, \mathbf{x})$.

Visualizing the layers

- Activation maximization:
  - Gradient ascent to maximize $h_i(W, \mathbf{x})$.
  - Start with randomly generated input and move towards the gradient.
  - Luckily, different random initializations yield very similar filter-like responses.
- Applicable to any network for which we can calculate the gradient $\partial h_i(W, \mathbf{x})/\partial \mathbf{x}$
- Need to tune parameters:
  - Learning rate
  - Stopping criteria
- Have the same problems of gradient descent
  - The space is non-convex
  - Local maxima etc.

Activation Maximization results



Erhan et al., "Understanding Representations Learned in Deep Architectures", 2010.

Autoencoders with the tools

The tools

- You can use “regular” network training functions/modules available in the libraries for learning the input.
- Some has nice tutorials:
  - Theano: http://deeplearning.net/tutorial/dA.html