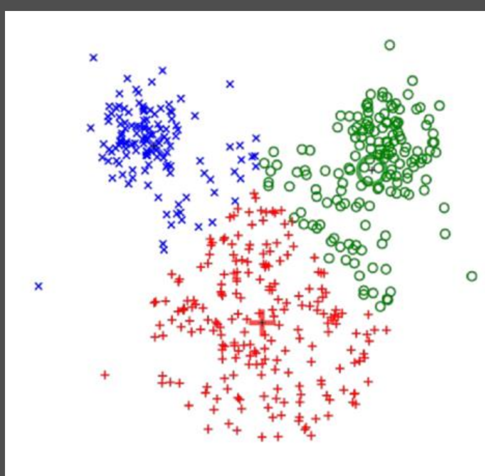
Computational Imaging Lab
Department of Electrical Engineering
IIT Madras

Autoencoders

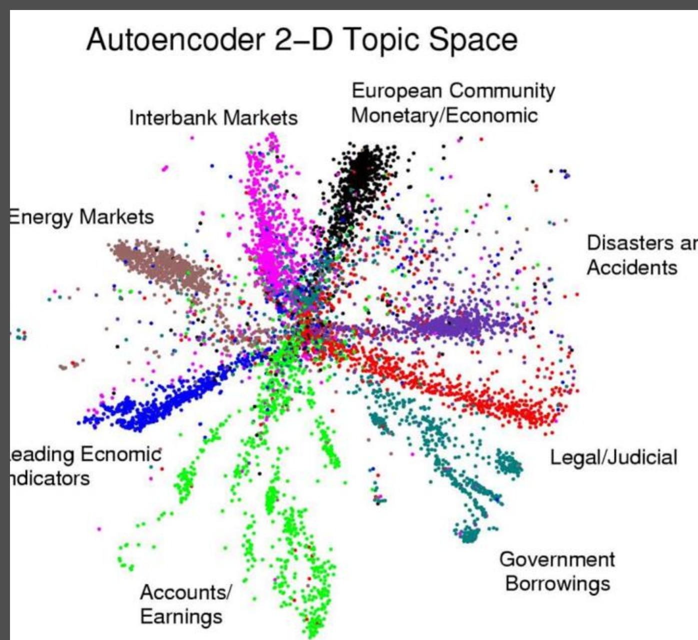
Kaushik Mitra, EE, IIT Madras

Unsupervised learning

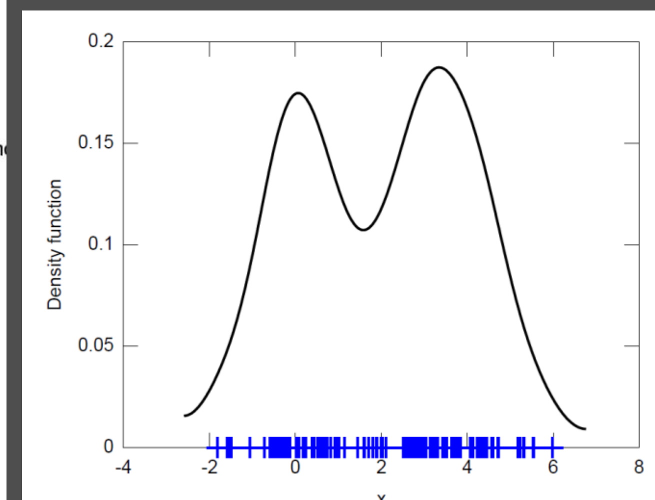
- Data: x, Just data, no labels!
- Goal: Learn some underlying hidden structure of the data
- Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



Clustering



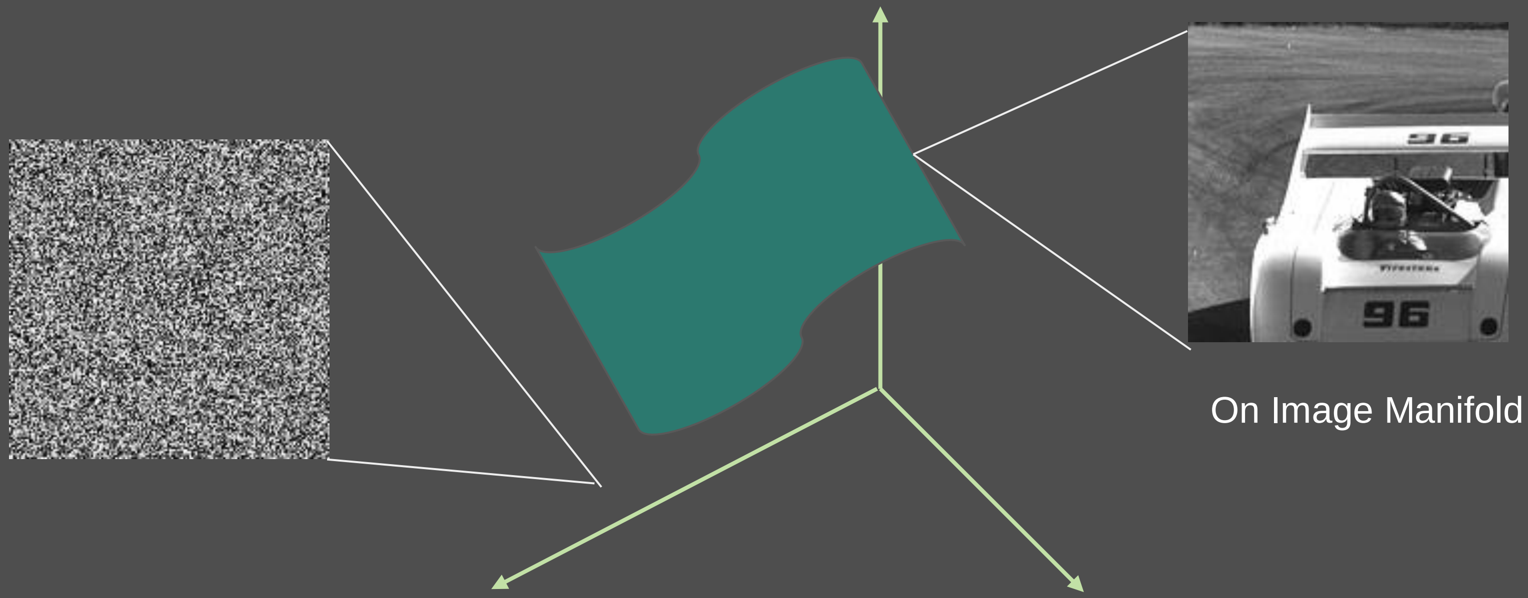
Dimensionality reduction
Document analysis, G. Hinton



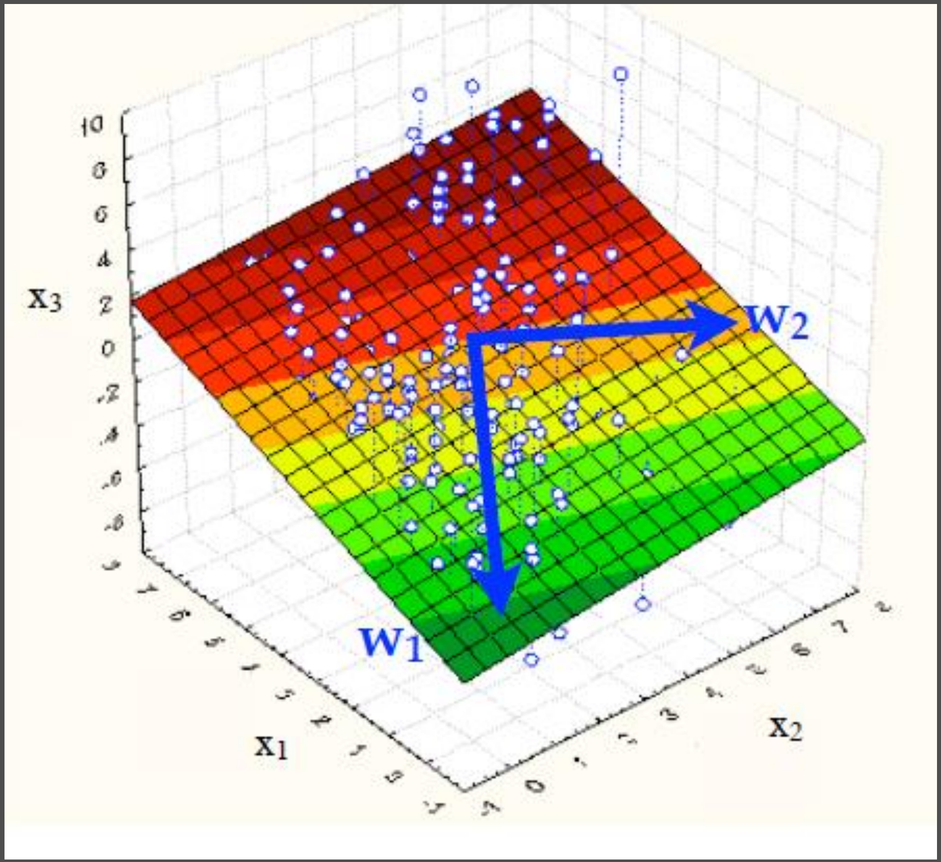
Density estimation

Why dimensionality reduction?

- High dimensional natural signal usually lie on low dimensional space (manifold)
  - Consider 64x64 binary image as a matrix
  - Very few out of 2^4096 matrices are actually natural images
- Application: compression, feature learning, sampling



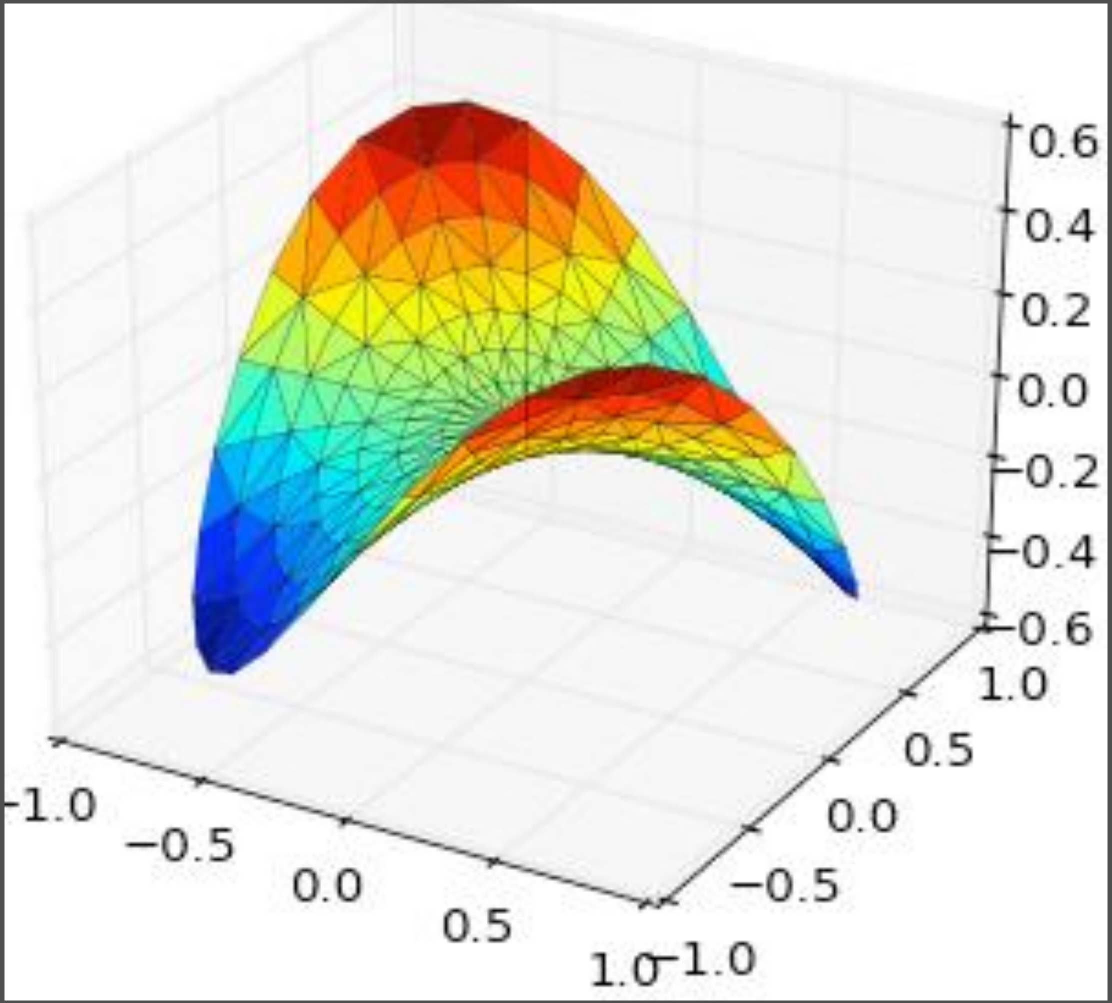
Dimensionality reduction via Principal Component Analysis (PCA)



- ❖ Learns *k* directions in which data has highest variance
- ❖ Projects the data along these directions to obtain low-dimensional representation

Slide courtesy Pascal Vincent

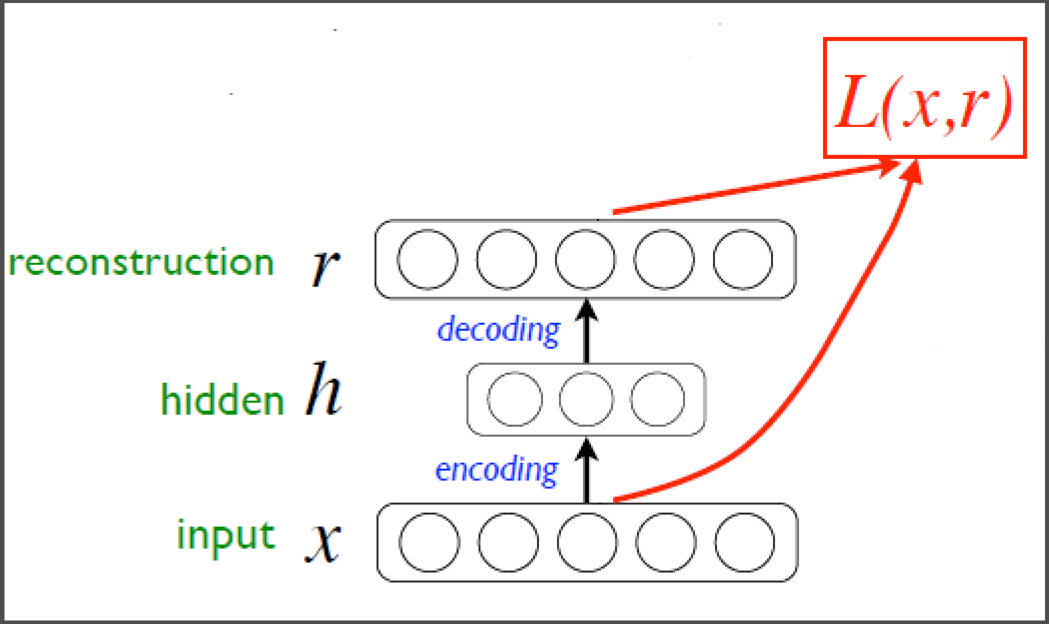
Non-linear dimensionality reduction



Slide courtesy Pascal Vincent

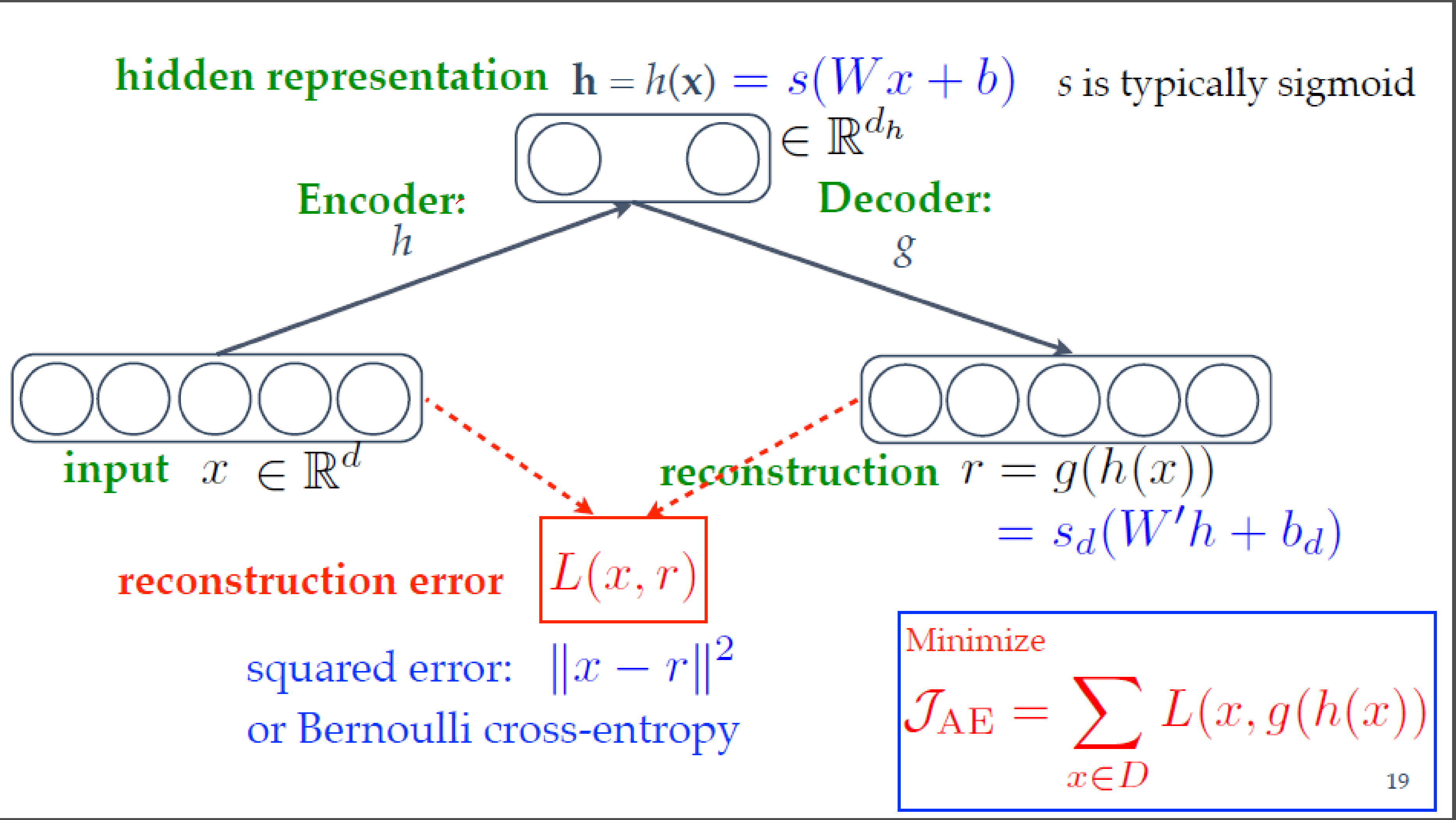
Autoencoders (AE)

- An autoencoder is a special type of feed forward neural network which does the following
  - *Encodes* its input *x* into a hidden representation *h(x)*
  - *Decodes* input from the hidden representation



Slide courtesy Pascal Vincent

Typical form of AE



Slide courtesy Pascal Vincent

Possible network Architectures

- AE can be constructed in two ways:
    - MLP based where each hidden unit is connected to all others and all layers are fully connected.
    - CNN based where the hidden units are formed by local connectivity.
        - CNN based architectures are favoured for image processing tasks since using fully connected layers for images will lead to huge networks with many layers.
        - In images information sharing is local hence CNN based architectures can utilize this better than FC.

PCA (Principal Component Analysis)

- PCA: most popular dimensionality-reduction method

- Aim: find small number of ”directions” in input space that explain variation in input data; re-represent data by projecting along those directions

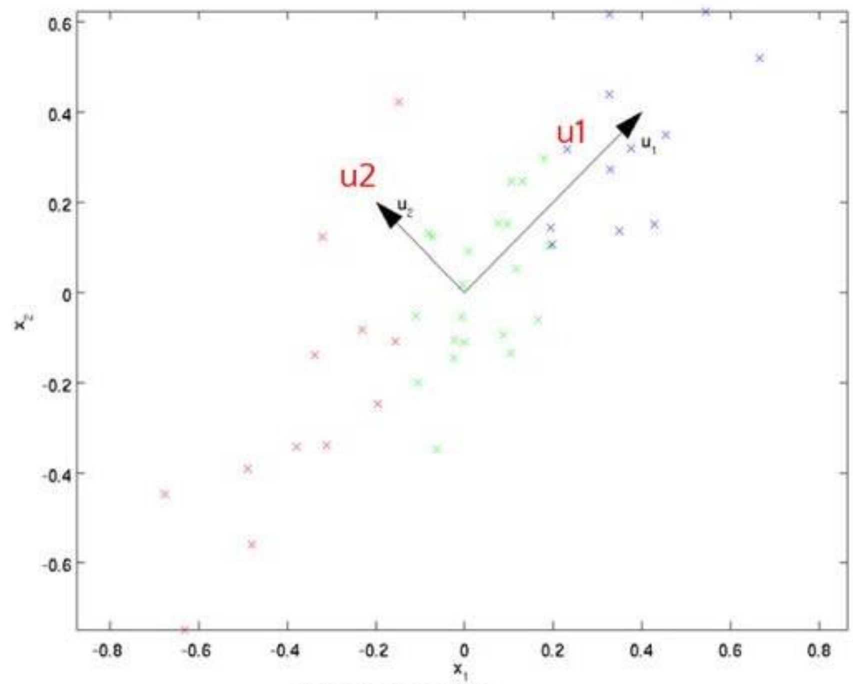
- Important assumption: variation contains information

- Handles high-dimensional data
  - if data has thousands of dimensions, can be difficult for classifier to deal with

- Often can be described by much lower dimensional representation

PCA Intuition

- Assume start with N data vectors, of dimensionality D
- Aim to reduce dimensionality:
    - linearly project (multiply by matrix) to much lower dimensional space, M << D
- Search for orthogonal directions in space w/ highest variance
    - project data onto this subspace
- Structure of data vectors is encoded in sample covariance



2D data example[8]

$Cv = \lambda v_{max}$

x_i

Mean $= E_x x = \frac{1}{N}\sum_{i=1}^{N} x_i = \bar{x}$

Covariance $= E_x\left[(x - Ex)(x - Ex)^T\right]$

$C = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$

Finding Principal Components

- To find the principal component directions, we center the data (subtract the sample mean from each variable)
- Calculate the empirical covariance matrix:

$$C = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}^{(n)} - \bar{\mathbf{x}})(\mathbf{x}^{(n)} - \bar{\mathbf{x}})^T$$

Where $\bar{X}$ is the data mean

- Find the M eigenvectors with largest eigenvalues of C: these are the principal components
- Assemble these eigenvectors into a D × M matrix U We can now express D-dimensional vectors x by projecting them to M-dimensional z

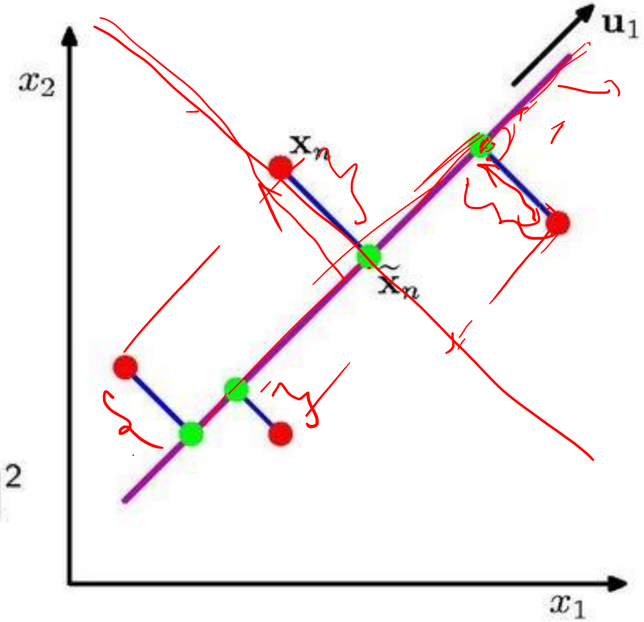
$$\mathbf{z} = U^T\mathbf{x}$$

$$U = \underbrace{\begin{bmatrix} v_1 & \cdots & v_M \end{bmatrix}}_{D \times M}$$

$$z = U^T x_i = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_M^T \end{bmatrix} x_i$$

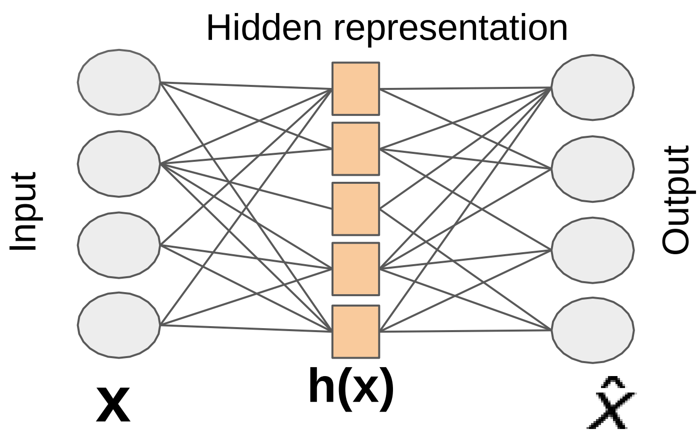
- Two views/derivations:
  - Maximize variance (scatter of green points)
  - Minimize error (red-green distance per datapoint)
- One derivation is that we want to find the projection such that the best linear reconstruction of the data is as close as possible to the original data
- The objective can be given $J = \sum ||\mathbf{x}^{(n)} - \tilde{\mathbf{x}}^{(n)}||^2$

  where $\tilde{\mathbf{x}}^{(n)} = \sum_{j=1}^{M} z_j^{(n)} \mathbf{u}_j + \sum_{j=M+1}^{D} b_j \mathbf{u}_j$
- The objective is minimised when $z_j^{(n)} = (\mathbf{x}^{(n)})^T \mathbf{u}_j; \quad b_j = \bar{\mathbf{x}}^T \mathbf{u}_j$



Autoencoders

- An autoencoder is a special type of feed forward neural network which does the following
- encodes its input x into a hidden representation h(x)
- Decodes input from the hidden representation



$$\boxed{\begin{aligned} h(x) &= f(Wx + b) \\ \hat{x} &= g(W^{*}h(x) + c) \end{aligned}}$$

$\min_{W, W^*} \|\hat{x} - x\|_2^2$

$h(x) = Wx + \cancel{b}$

$\hat{x} = W^* h(x) + \cancel{c}$

$\hat{x} = W^* W x$

$K K^{-1}$

$W^T W = I$

$U = [u_1 \; - \; u_m]_{D \times m}$ $\quad m (\le D)$

$U^T = \begin{bmatrix} u_1^T \\ u_2^T \\ u_m^T \end{bmatrix}$

$h_{m\times 1} = U^T x_{D \times 1}$

$\hat{x}_{D\times 1} = U_{D\times m} h_{m\times 1}$

$\|x - \hat{x}\|^2 = \sum_{m+1}^{D} \lambda_i$

$\hat{x} = U U^T x$

Relation between PCA and AE

- When dimension of h(x)<<x we take think of AE as a dimensionality reduction.
  Mx1 Dx1
- AE is equivalent to PCA when
  - Encoder is linear
  - Decoder is linear
  - *This means there are no non-linear activations*
  - Cost function is MSE

Objective of AE $\min_{\theta} \sum_{i=1}^{m} \sum_{j=1}^{m} (x_{ij} - \widehat{x_{ij}})^2$ = $\min_{W^* h(X)} (\|X - h(X)W^*\|_F)^2$

From SVD we know that the optimal solution for the above problem is

$$h(x)W^* = U_{.,\leq k}\Sigma_{k,k}V^T_{.,\leq k}$$

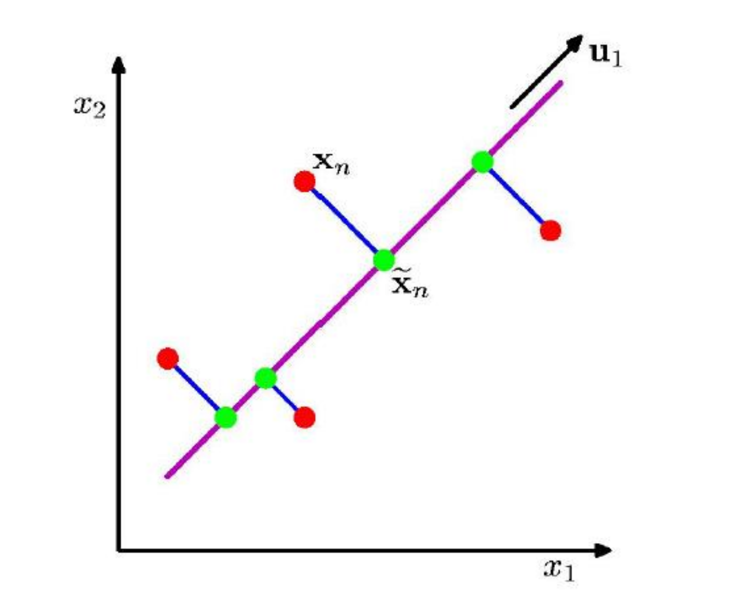
With one possible solution

$$h(x) = U_{.,\leq k}\Sigma_{k,k}$$
$$W^* = V^T_{.,\leq k}$$

$$
\begin{aligned}
h(x) &= U_{.,\leq k}\Sigma_{k,k} \\
&= (XX^T)(XX^T)^{-1}U_{.,\leq K}\Sigma_{k,k} && \textit{(pre-multiplying } (XX^T)(XX^T)^{-1} = I \\
&= (XV\Sigma^T U^T)(U\Sigma V^T V\Sigma^T U^T)^{-1}U_{.,\leq k}\Sigma_{k,k} && \textit{(using } X = V\Sigma U^T \\
&= XV\Sigma^T U^T (U\Sigma\Sigma^T U^T)^{-1}U_{.,\leq k}\Sigma_{k,k} && (V^T V = I \\
&= XV\Sigma^T U^T U(\Sigma\Sigma^T)^{-1}U^T U_{.,\leq k}\Sigma_{k,k} && ((ABC)^{-1} = C^{-1}B^{-1}A^{-1} \\
&= XV\Sigma^T(\Sigma\Sigma^T)^{-1}U^T U_{.,\leq k}\Sigma_{k,k} && (U^T U = I \\
&= XV\Sigma^T\Sigma^{T^{-1}}\Sigma^{-1}U^T U_{.,\leq k}\Sigma_{k,k} && ((AB)^{-1} = B^{-1}A^{-1} \\
&= XV\Sigma^{-1}I_{.,\leq k}\Sigma_{k,k} && (U^T U_{.,\leq k} = I_{.,\leq k} \\
&= XVI_{.,\leq k} && (\Sigma^{-1}I_{.,\leq k}\Sigma_{k,k}^{-1} \\
h(x) &= XV_{.,\leq k}
\end{aligned}
$$

Thus $h(x)$ is a linear transformation of X and $W = V_{.,\leq k}$

Linear AE and PCA



- AE same as PCA for
  - Linear encoder: $h(x) = Wx$
  - linear decoder: $g(h) = W'x$
  - Squared loss:

$$\sum_{n=1}^{N} ||x_n - W'Wx_n||^2$$

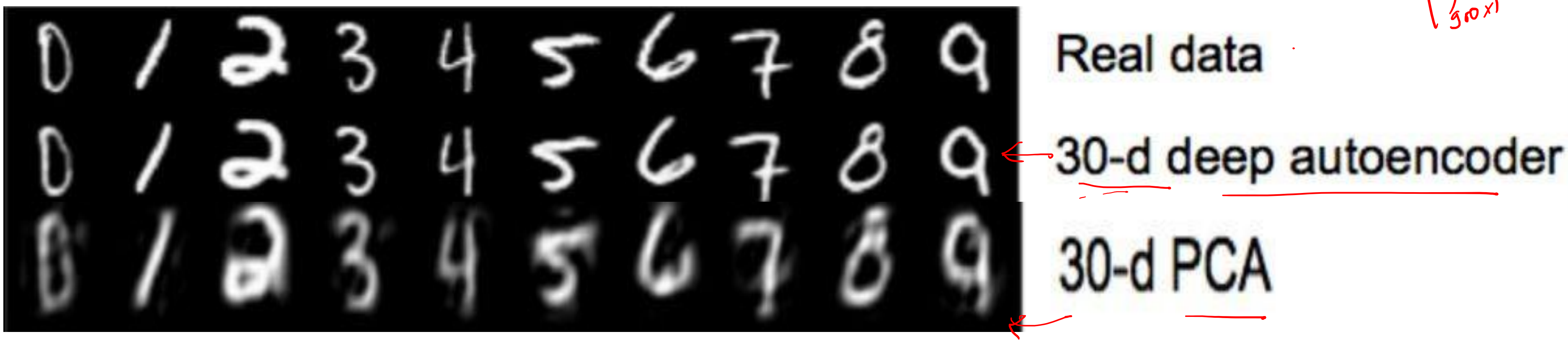
- Non-linear AE is similar to non-linear manifold techniques

MNIST dataset reconstruction comparison

30
×30
900

900×1

30×1



Choice of activation and loss functions

- Design choices
  - Choice of activation functions, h(x) and g(x)
  - Choice of loss function, *L*
- Suppose all our inputs are binary
  - Which activation function to use in the decoder?
    - linear , sigmoid or tanh
  - Sigmoid as it naturally restricts all outputs to be between 0 and 1
  - Loss function:

$$\min_{W,W',b,c}\left\{-\sum_{n=1}^{N}(x_n \log \hat{x}_n + (1-x_n)\log(1-\hat{x}_n))\right\}$$

- Suppose input are real valued
  - Linear activation for decoder
  - Loss function:

$$\min_{W,W',b,c}\left\{\sum_{n=1}^{N}||x_n - \hat{x}_n||^2\right\}$$

Training - back propagation

$(x-\hat{x})^2 \sim \max \langle x, \hat{x} \rangle$

$\min \; x^2 + \hat{x}^2 - 2x\hat{x} \rightarrow \max \; x\hat{x}$

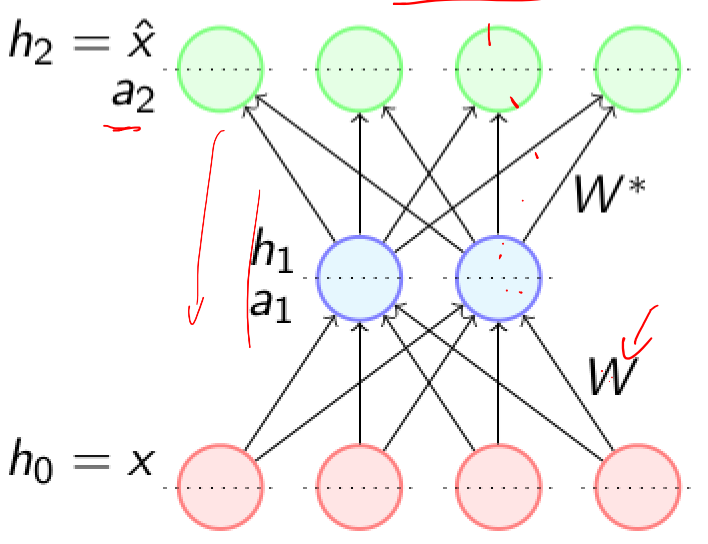
$a_2 = W^* h_1$

$h_2 = f(a_2)$

$\frac{\partial L}{\partial W^*} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial a_2}\frac{\partial a_2}{\partial W^*}$

$$\min_{W, W^*, c, b} \left\{ \frac{1}{2}(\hat{x} - x)^2 \right\}$$

$$\mathscr{L} = \frac{1}{2}(\hat{x}_i - x)^2$$



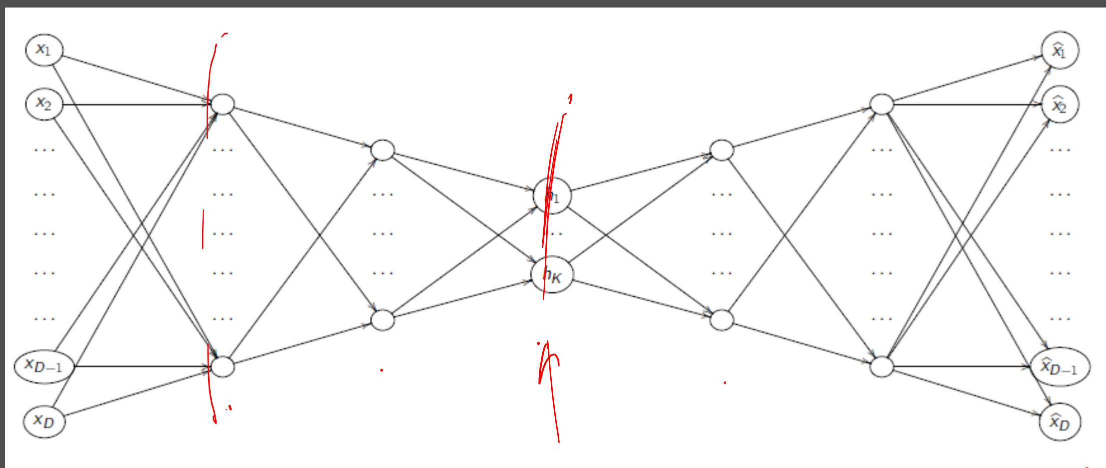
$$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} \qquad \frac{\partial \mathscr{L}(\theta)}{\partial W}$$

$$\frac{\partial \mathscr{L}(\theta)}{\partial W^*} = \frac{\partial \mathscr{L}(\theta)}{\partial h_2} \boxed{\frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial W^*}}$$

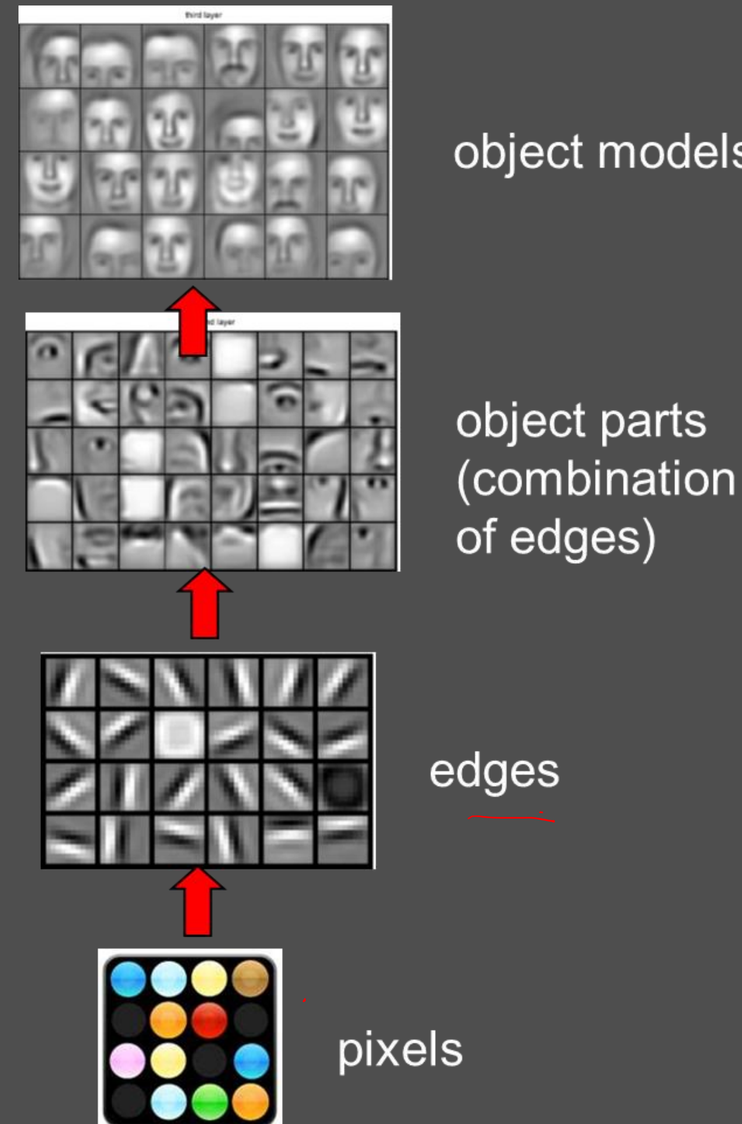
$$\frac{\partial \mathscr{L}(\theta)}{\partial W} = \frac{\partial \mathscr{L}(\theta)}{\partial h_2} \boxed{\frac{\partial h_2}{\partial a_2} \frac{\partial a_2}{\partial h_1} \frac{\partial h_1}{\partial a_1} \frac{\partial a_1}{\partial W}}$$

$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial h_2}\frac{\partial h_2}{\partial a_2}\frac{\partial a_2}{\partial h_1}\frac{\partial h_1}{\partial a_1}\frac{\partial a_1}{\partial W}$

Stacked autoencoder



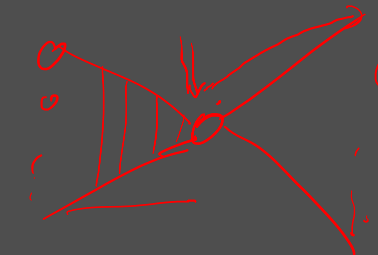
❖ Learns to abstract feature hierarchies



Pic courtesy, Andrew Ng

$x_i, i=1,...,10$

Cases when Autoencoder learning fails



- ❖ Capacity of encoder/decoder is too high
  - ➢ Autoencoder with a one-dimensional code and a very powerful nonlinear encoder can learn to map x_i to code i.
  - ➢ The decoder can learn to map these integer indices back to the values of specific training examples

$W^* = I$

sparse autoencoder

denoising autoen

- ❖ Overcomplete case: where hidden code *h* has dimension greater than input *x*
  - ➢ Even a linear encoder/decoder can learn to copy input to output without learning anything useful about data distribution

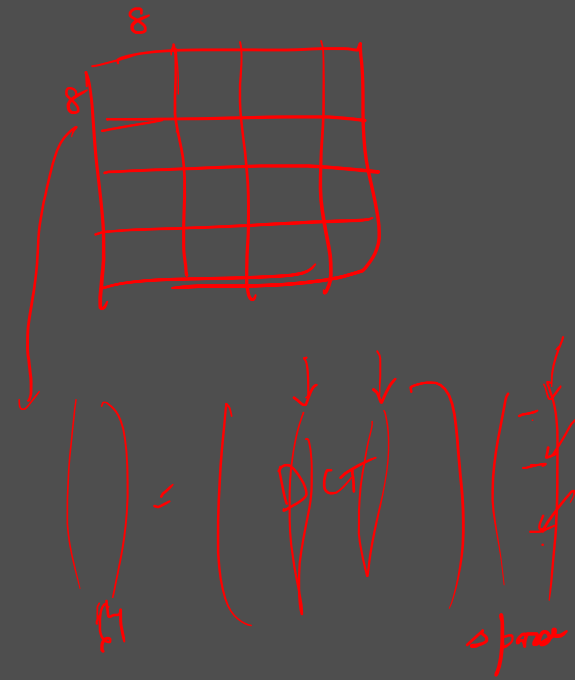
Slide courtesy Sargur Sriharit

Correct autoencoder design

- Choose code size and network capacity based on the complexity of the underlying manifold
- Use regularization
  - Denoising autoencoder
  - Sparse autoencoder
  - Contractive autoencoder

$(x - \hat{x})$

Regularized autoencoders



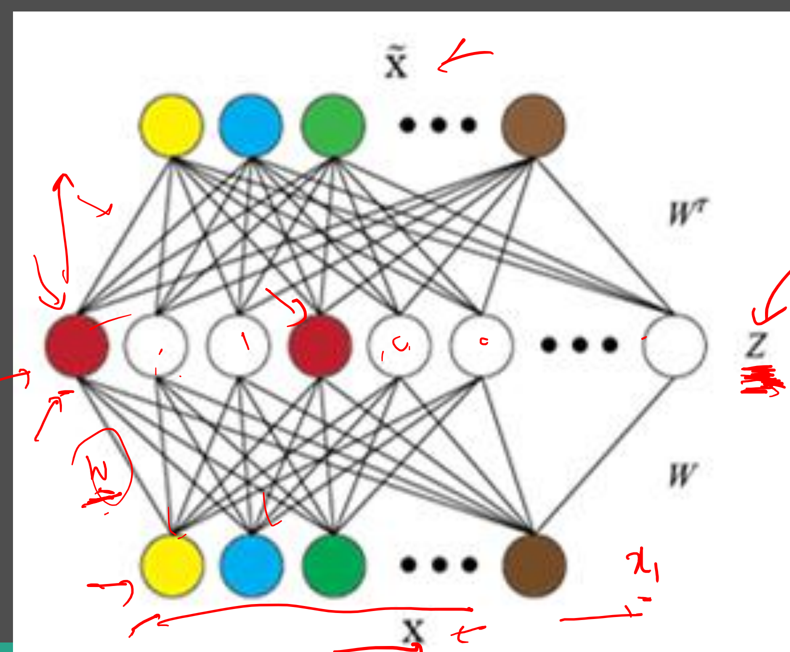
- ❖ Prevents learning *identity* mappings
- ❖ Other properties include:
  - ➢ *Sparsity* of hidden representation
  - ➢ Robustness to *noise*
  - ➢ Robustness to *missing* inputs
  - ➢ Insensitive to *minor variations* in data
- ❖ Can be *overcomplete*
  - ➢ learns something useful about data manifold even if the model capacity is great enough to learn trivial identity mapping

Sparse Autoencoder

- ❖ Inspired from sparse coding approaches e.g. *dictionary learning*
- ❖ Enforce sparsity over hidden unit activation
  - ➢ Only a subset of the hidden units are active.
  - ➢ Learns a good representation even for an overcomplete case
- ❖ Sparse encoders are typically used to learn features for tasks such as classification

k-sparse if only k entries are non zero

k-sparse k=10



10

100

(100,10)

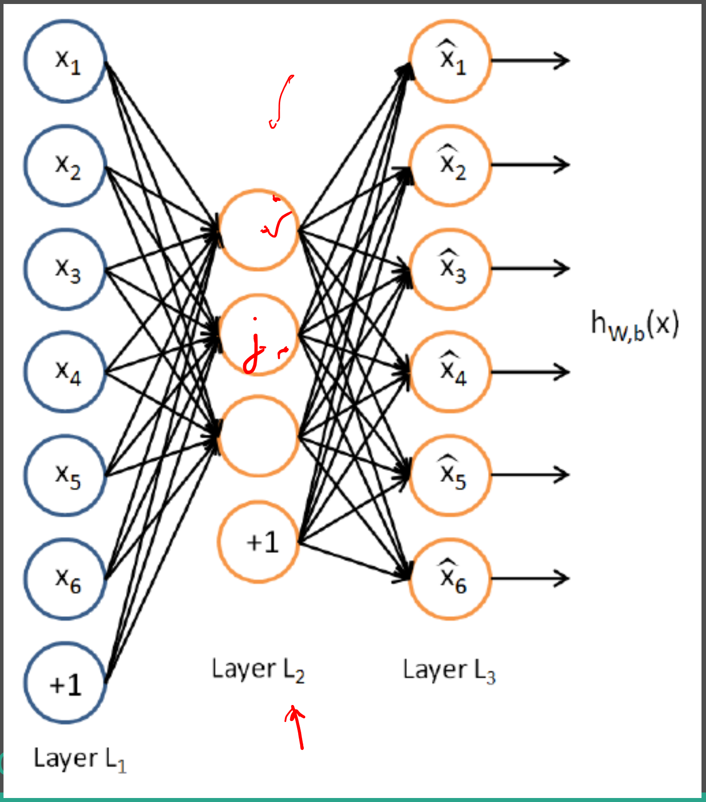
Sparsity as regularizer

- A sparse AE constrain the neurons to be inactive most of the time.
- The average activation of j-th hidden unit is given by

$$\hat{\rho}_j = \frac{1}{N}\sum_{n=1}^{N} a_j^{(2)}(x^{(n)})$$

$\frac{\partial \hat{\rho}}{\partial W}$

  - averaging is over the training dataset



- We would like to (approx) enforce $\boxed{\hat{\rho}_j = \rho}$
  - small value of ρ (say, 0.05)

$L(x, \hat{x}) + (\hat{\rho}_j - \rho)^2 \rightarrow KL(\hat{\rho}, \rho)$

Need extra penalty term

$\sum KL(\rho, \hat{\rho}_j)$

$\|x - \hat{x}\|^2 + \Omega(\theta)$

- To enforce $\hat{\rho}_j = \rho$
  - We need to add an extra penalty term is added to the objective
  - One possible choice:

$$\Omega(\theta) = \sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

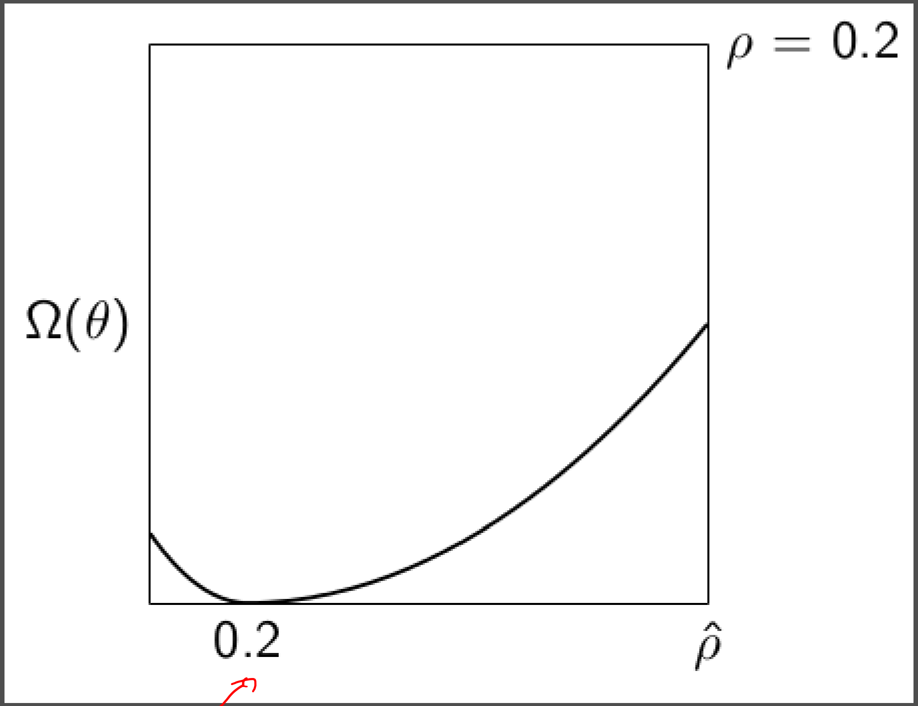
  - This term is equivalent to KL divergence

min $$\sum_{j=1}^{s_2} KL(\rho || \hat{\rho}_j)$$

When will this term reach its minimum and what is the minimum value?

Sparsity as regularizer

- $KL(\rho||\hat{\rho}_j) = 0$ If $\hat{\rho}_j = \rho$, with $\rho = 0.2$
- Increases monotonically as $\hat{\rho}_j$ diverges from ρ



Training

$\|x-\hat{x}\|^2$

The new cost function is $\hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta)$

- For updating the weights, we require $$\frac{\partial \hat{\mathcal{L}}(\theta)}{\partial W} = \frac{\partial \mathcal{L}(\theta)}{\partial W} + \frac{\partial \Omega(\theta)}{\partial W}$$
- The first term is known. To estimate the second term expand the constraint function $$\Omega(\theta) = \sum_{i=1}^{n} \rho \log \rho - \rho \log \hat{\rho}_i + (1-\rho)\log(1-\rho) - (1-\rho)\log(1-\hat{\rho}_i)$$

$\hat{\rho}(W)$

- Now the second term can be expressed in chain rule as $$\frac{\partial \Omega(\theta)}{\partial W} = \frac{\partial \Omega(\theta)}{\partial \hat{\rho}} \frac{\partial \hat{\rho}}{\partial W}$$ where $$\frac{\partial \Omega(\theta)}{\partial \hat{\rho}} = -\frac{\rho}{\hat{\rho}} + \frac{1-\rho}{1-\hat{\rho}}$$

Visualization of learned weights

We can think of each neuron as a filter which will fire (or get maximally) activated for a certain input configuration *x*.

Suppose, we want to find the input that activated **the neuron in blue** here. The pre trained weights from inputs to that neuron be W_1. Suppose our inputs are normalized $||x||_2=1$.

$\min_x \; -w^T x + \lambda(x^T x - 1)$

Objective can be stated as,

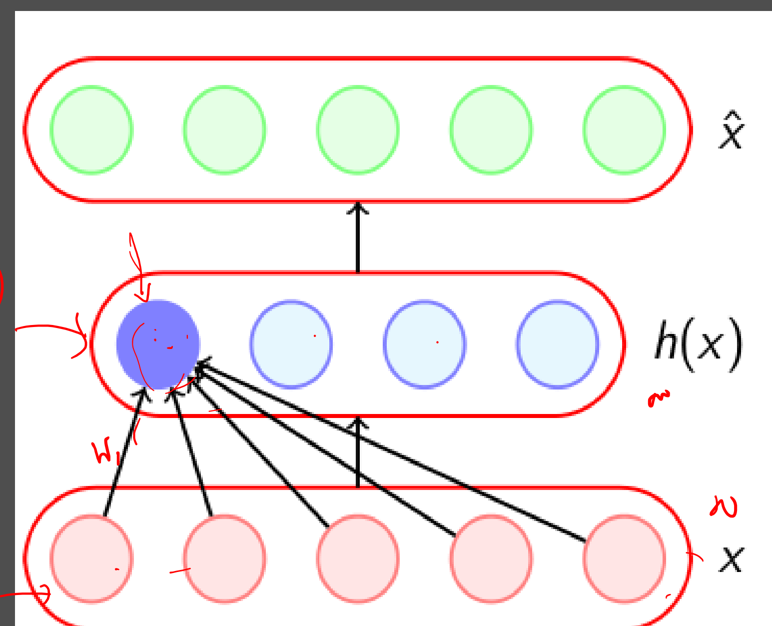
$$\max_x \; \{w_1^T x\}$$

$$s.t. \;\; ||x||^2 = x^T x = 1$$

$$\text{Solution:} \quad x = \frac{w_1}{\sqrt{w_1^T w_1}}$$

$\max \sigma(w^T x)$

$\max w^T x$



ras

Visualization of learned weights

- Different hidden units have learned to detect edges at different position and orientation
- Useful for tasks such as object recognition

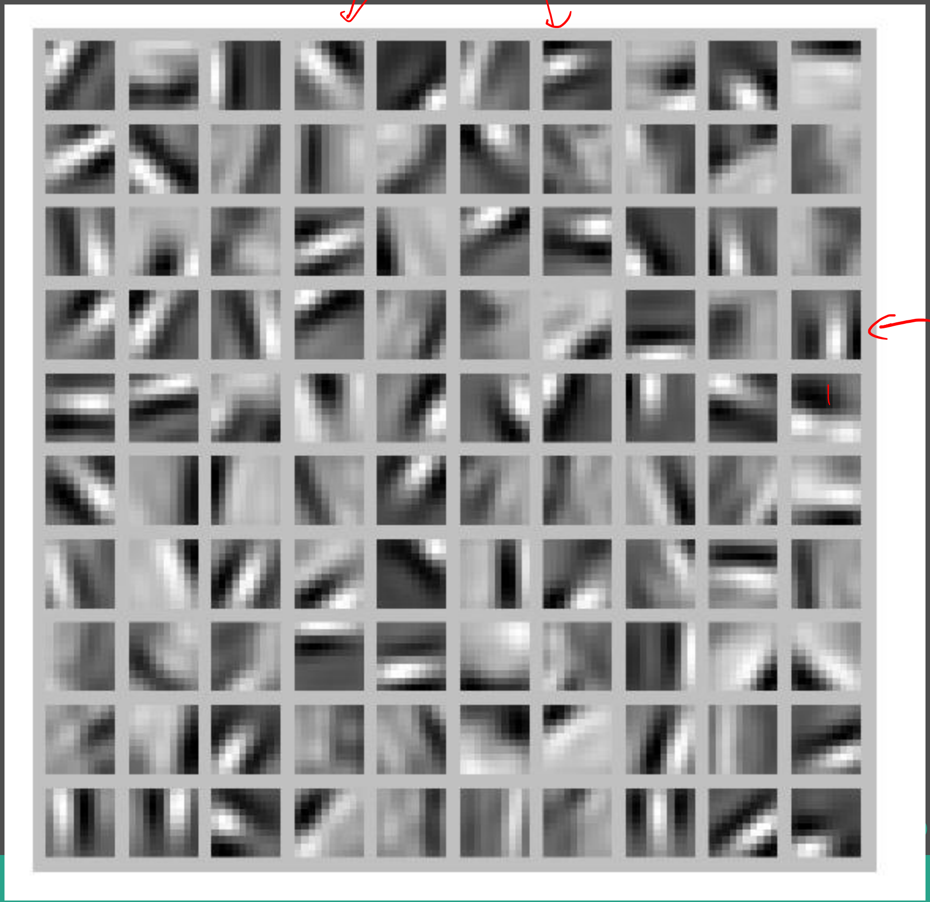
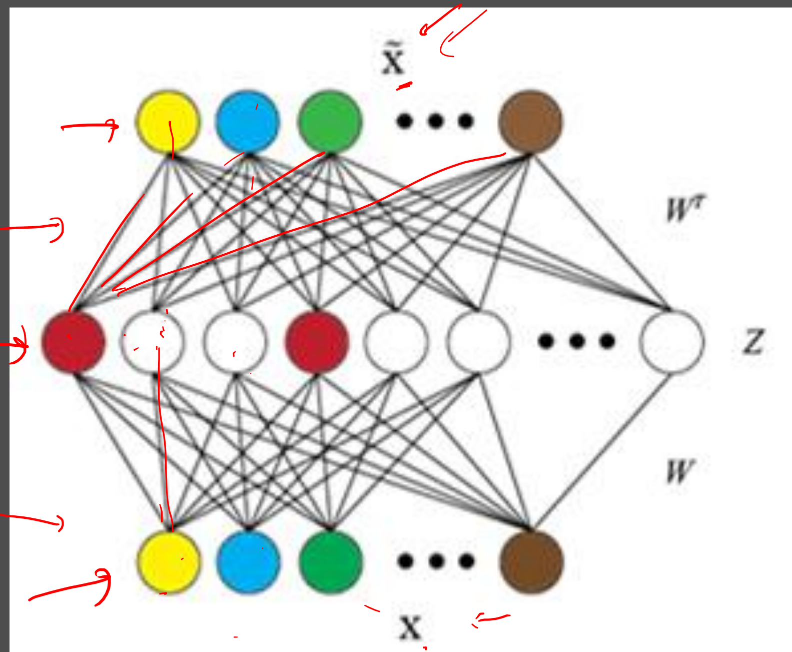


Figure courtesy Andrew Ng

K-sparse autoencoder

Makhzani et al., 2013

❖ Finds the *K* highest activations in *h(x)* and zeros out the rest.



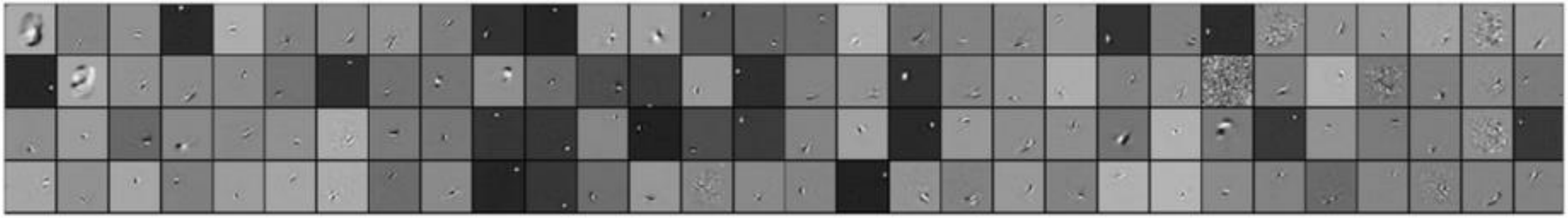
Dictionary learning

❖ The error is then back-propagated only through the *K* active nodes in *h(x)*

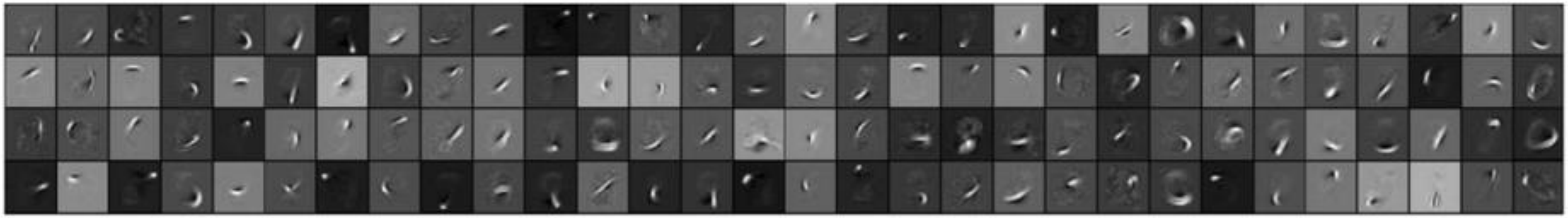
K-sparse autoencoder

❖ With 1000 hidden units and different values of K

784x1 784x1000



(a) $k = 70$

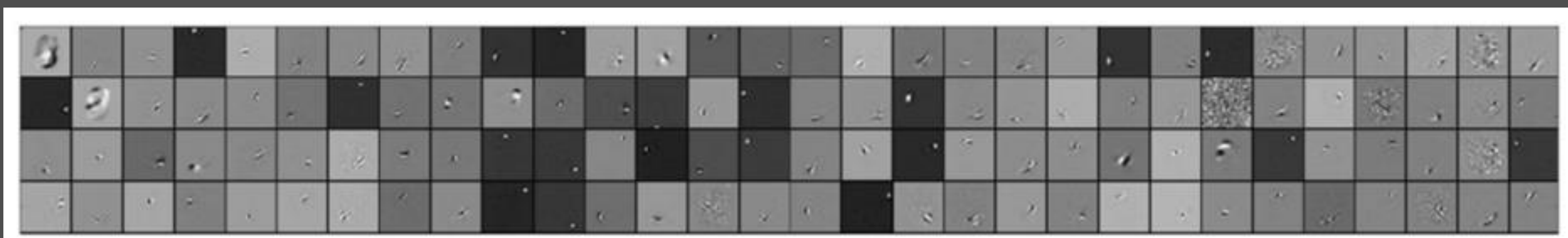


(b) $k = 40$

(d) $k = 10$

K-sparse autoencoder

- As the value of k decreases the network is forced to learn increasingly complete representations of each individual digit.
- K = 70, over-complete representation, network learns highly local features involving small stroke and blob detectors.



- K = 10, highly sparse, each node learns to represent a digit
  - To reconstruct each handwritten digit from only 10 basis functions, each basis must closely resemble the full image.

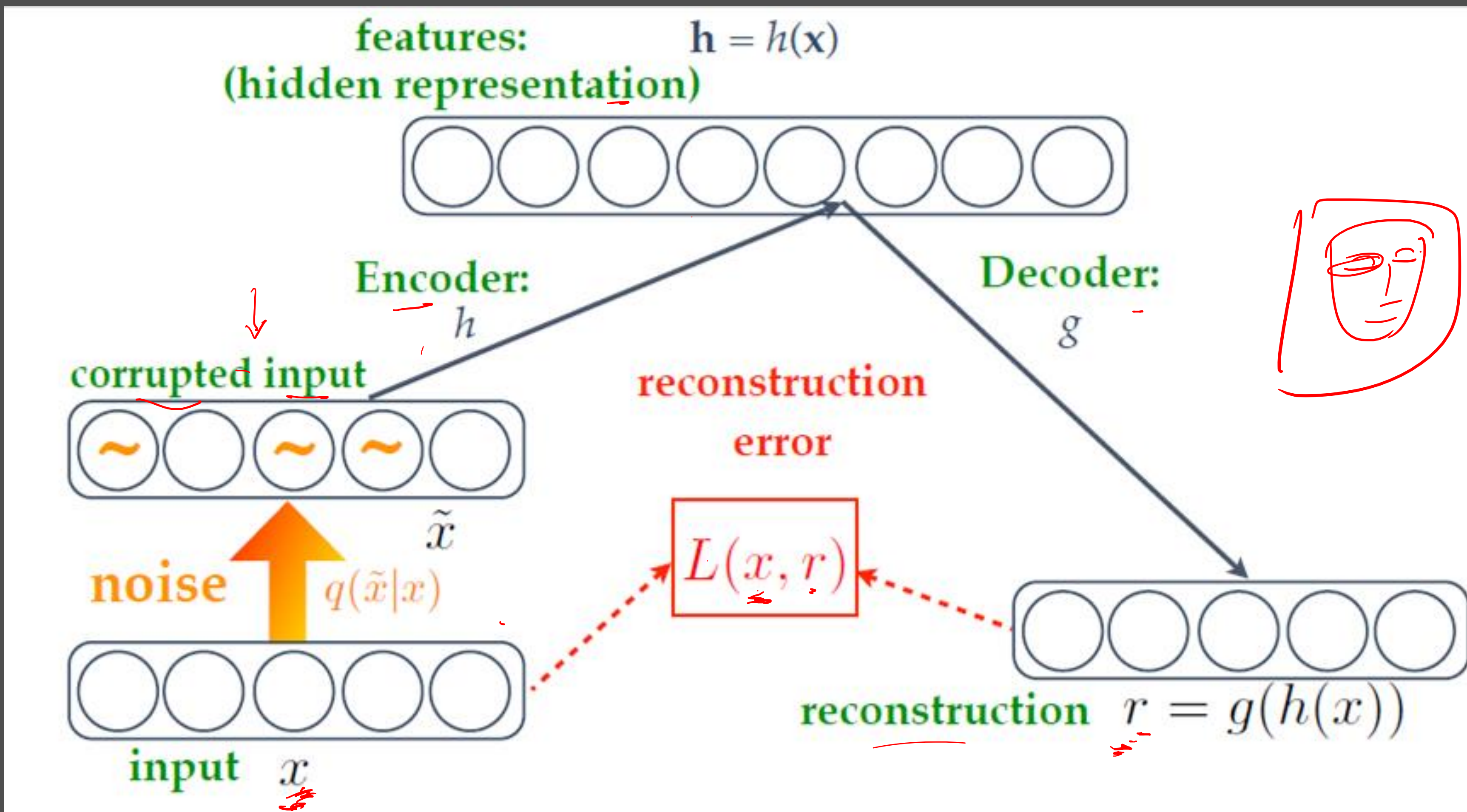
Denoising autoencoder

(Vincent et al, ICML 2008)

- Rather than adding a penalty Ω to the cost function, the reconstruction error term of the cost function is changed
- Traditional autoencoders minimize $L(x, g(f(x)))$
- A DAE minimizes $L(x, g(f(\tilde{x})))$, where $\tilde{x}$ is noise corrupted version of x
  - The autoencoder must undo this corruption rather than simply copying their input
- Denoising training forces f and g to implicitly learn the structure of p_data(x)

Denoising AE

(Vincent et al, ICML 2008)



Ways of adding noise

- Adding noise can be done in two ways
  - flips a fraction q of the inputs to zero as
  - Add gaussian noise as

$$\tilde{x} = x + \mathcal{N}(0,1)$$

$$p(\tilde{x} = 0|x) = q$$
$$p(\tilde{x} = 1|x) = 1 - q$$

x 0 0 0 0 0

- How adding noise helps?
  - The network can no longer copy the input as such to the output node.
  - The network thus learns the underlying data space and projects the noisy input to this space.

Denoising AE

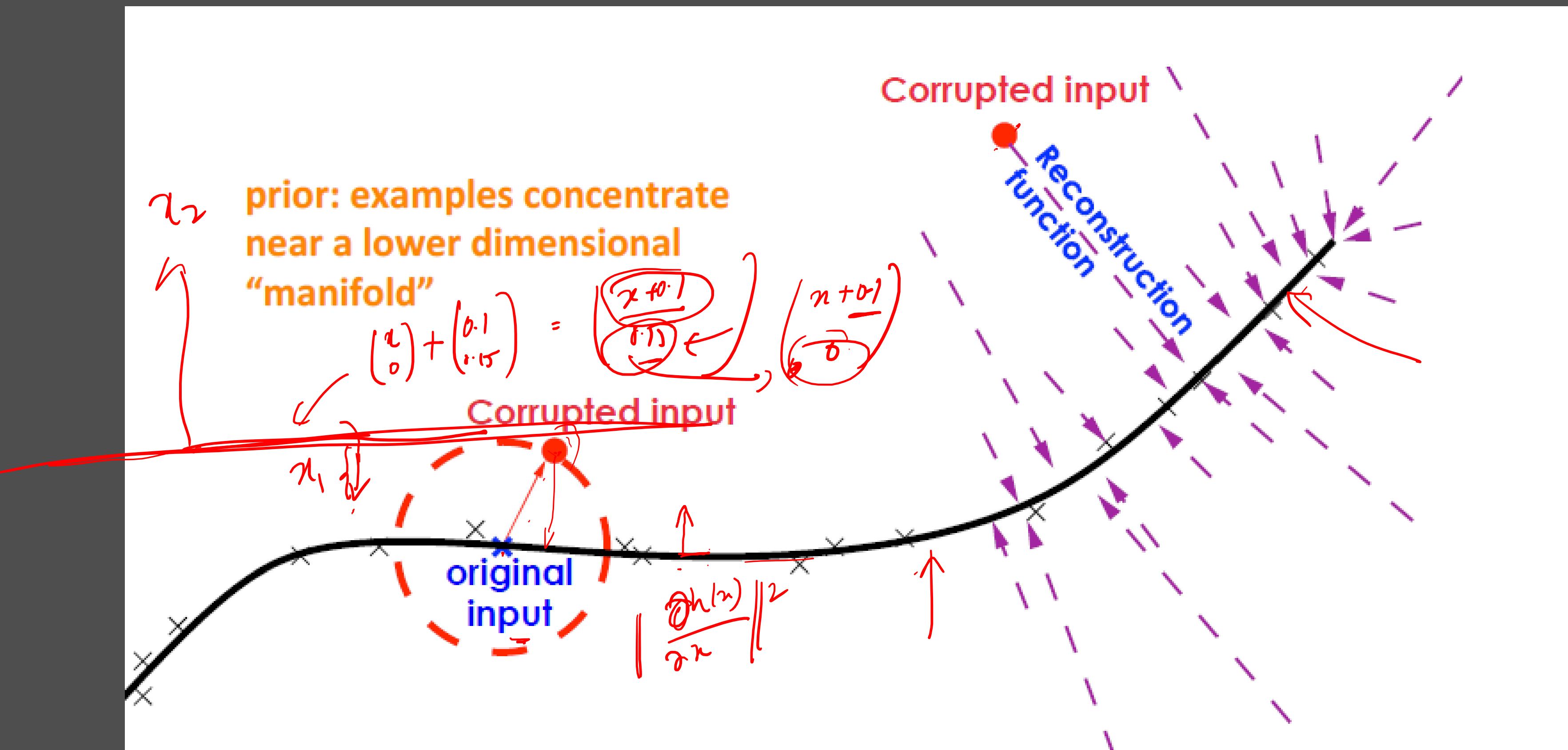
❖ Autoencoder training minimizes:

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

❖ Denoising autoencoder training minimizes:

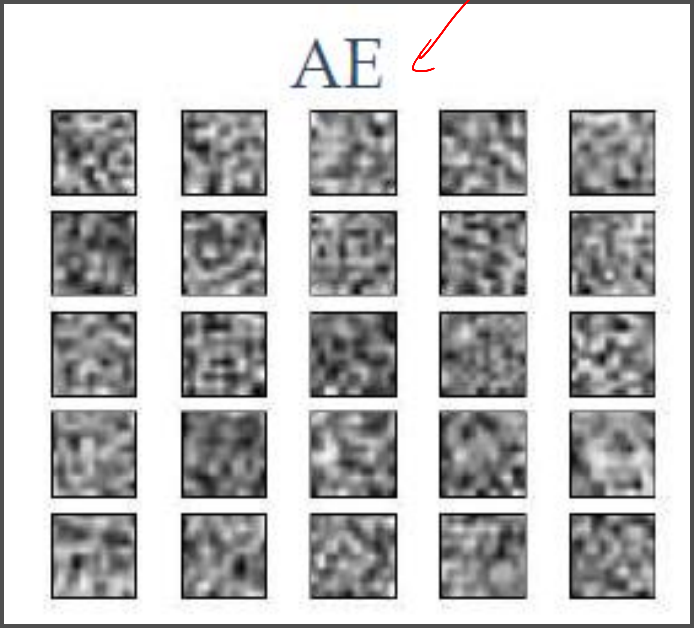
$$\mathcal{L}_{DAE}(\theta) = \sum_{x \in D} \mathbb{E}_{q(\tilde{x}|x)}[\mathcal{L}(x, g(h(\tilde{x})))]$$

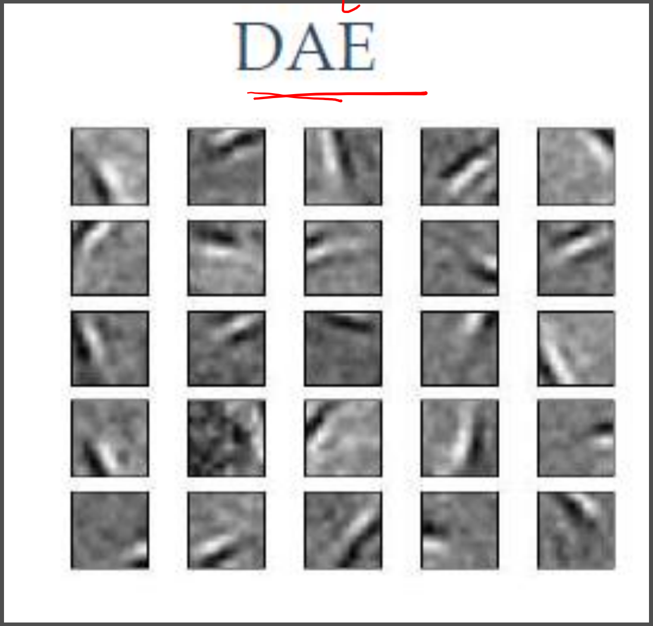
DAE learns to project back corrupted input back to manifold



Slide courtesy Pascal Vincent

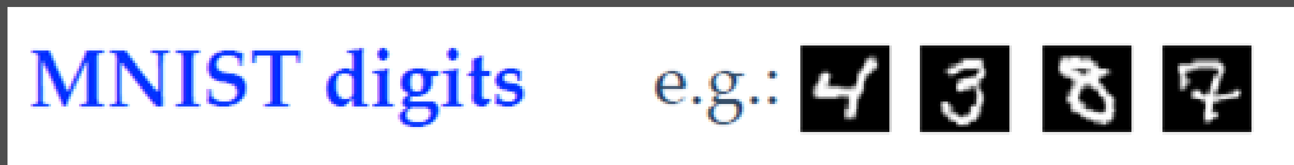
Learned weights

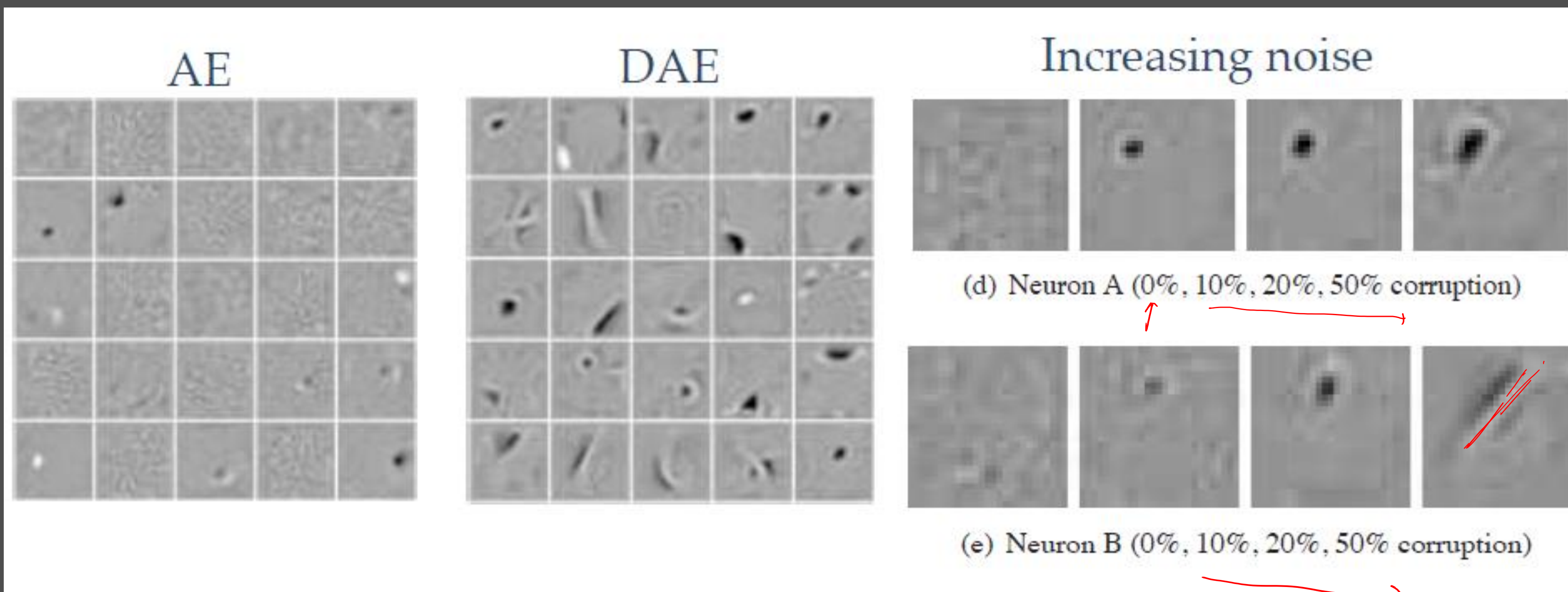




Slide courtesy Pascal Vincent

Learned weights





Slide courtesy Pascal Vincent

Observations

- The vanilla AE does not learn many meaningful patterns
- The hidden neurons of the denoising AEs seem to act like pen-stroke detectors
- As the noise increases the filters become more wide because the neuron has to rely on more adjacent pixels to feel confident about a stroke

Making representation insensitive to noise

❖ DAE encourages reconstruction *x* to be insensitive to noise

$x \to \tilde{x} = x + n$

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

❖ Alternative: encourage representation *h(x)* to be insensitive

$N(0, \sigma^2)$

$\tilde{x} = x + n$

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

$x_i \to \tilde{x}_i^1, \tilde{x}_i^2, \tilde{x}_i^3, \tilde{x}_i^4$

Reconstruction error

Stochastic regularization term

Slide courtesy Pascal Vincent

From stochastic to analytic regularization

stochastic contractive AE

- SCAE stochastic regularization term $\mathbb{E}_{q(\tilde{x}|x)}||h(x) - h(\tilde{x})||^2$
- For small additive noise

$$\tilde{x}|x = x + \epsilon \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

$E||\epsilon||^2 \rightarrow E[\epsilon_1^2 + \dots \epsilon_n^2] = n\sigma^2$

$x_{n\times 1}$ $h_{k\times 1}$

- Taylor series expansion gives

matrix

vector

$$h(\tilde{x}) = h(x + \epsilon) = h(x) + \frac{\partial h}{\partial x}\epsilon + \cdots \quad E||h(\tilde{x}) - h(x)||^2$$

vector

$$= E||\frac{\partial h}{\partial x}\epsilon||^2 \leq E||\frac{\partial h}{\partial x}||^2 ||\epsilon||^2$$

- It can be shown that

$$\mathbb{E}_{q(\tilde{x}|x)}||h(x) - h(\tilde{x})||^2 \approx \sigma^2 \parallel \frac{\partial h(x)}{\partial x} \parallel_F^2$$

stochastic (SCAE) **analytic (CAE)**

Contractive Auto-Encoder (CAE)

(Rafai et al, ICML, 2011)

- Minimize

$$\mathcal{L}_{CAE}(\theta) = \sum_{x \epsilon D} \mathcal{L}\left(x, g\big(h(x)\big)\right) + \lambda \parallel \frac{\partial h(x)}{\partial x} \parallel_F^2$$

Reconstruction error

Analytic contractive term

- For training examples, encourages
  - Small reconstruction error
  - Representation h(x) insensitive to small changes around example

Slide courtesy Pascal Vincent

Role of regularizer

$\frac{\partial h}{\partial x}$

- The regularization term is $\Omega(\theta) = \| J_x(h(x)) \|_F^2$
  - where $J_x(h(x))$ is the Jacobian of the encoder.
- For an *n* dimensional input *x* with *k* dimensional hidden unit the matrix equivalent is given as

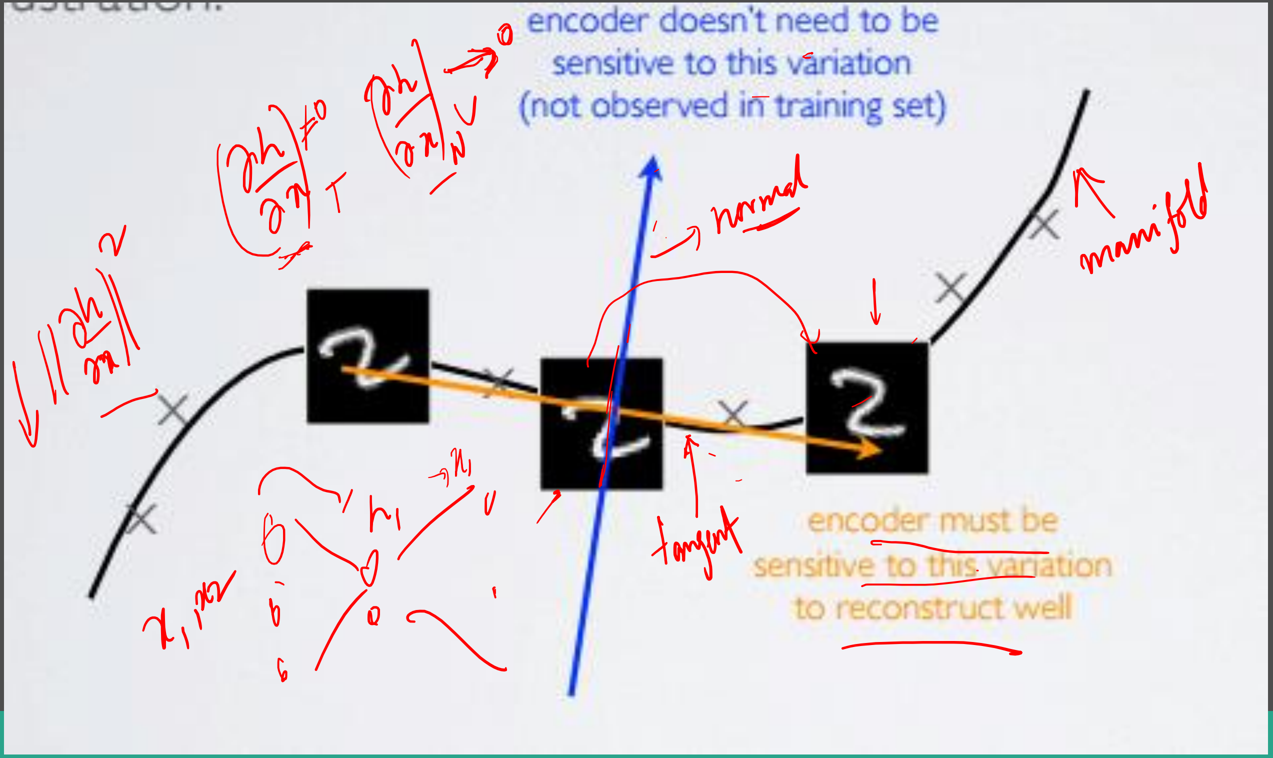
$$\begin{bmatrix} \frac{\partial h(x)_1}{\partial x_1} & \cdots & \cdots & \cdots & \frac{\partial h(x)_k}{\partial x_1} \\ \frac{\partial h(x)_1}{\partial x_2} & \cdots & \cdots & \cdots & \frac{\partial h(x)_k}{\partial x_2} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h(x)_1}{\partial x_n} & \cdots & \cdots & \cdots & \frac{\partial h(x)_k}{\partial x_n} \end{bmatrix}$$

$$\| J_x(h(x)) \|_F^2 = \sum_{i=1}^{N} \sum_{j=1}^{K} \left(\frac{\partial h(x)_j}{\partial x_i}\right)^2$$

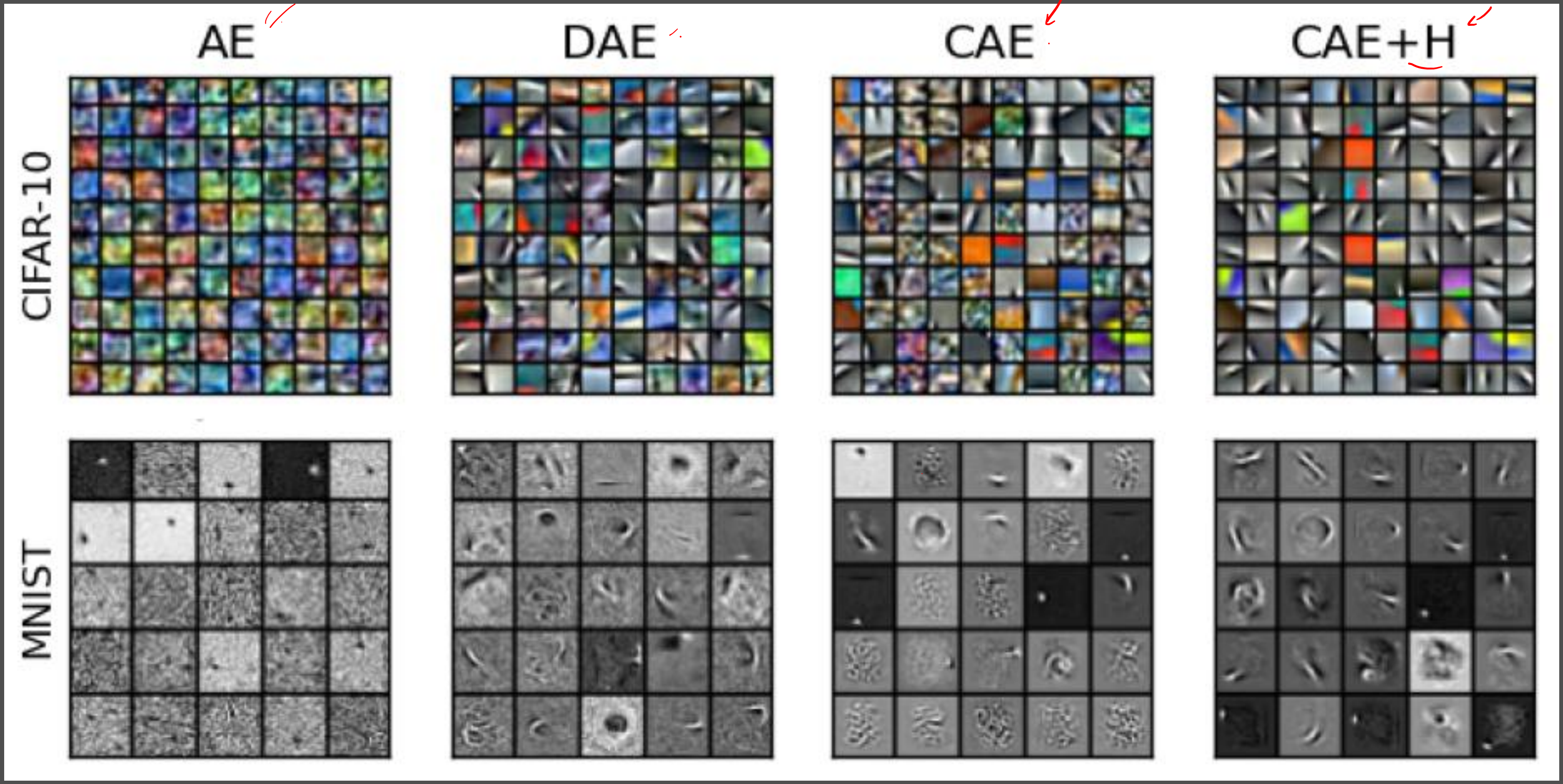
- The (i,j) entry of the Jacobian captures the variation in the output of the j-th neuron with a small variation in the i-th input.
- What does the regularizer do?

Learns tangent space of manifold

- Tradeoff between
  - Reconstruction error : capture all variations in data
  - Regularization : Do not capture variations in data
  - Net effect: capture important variations in data only (tangent space of manifold)



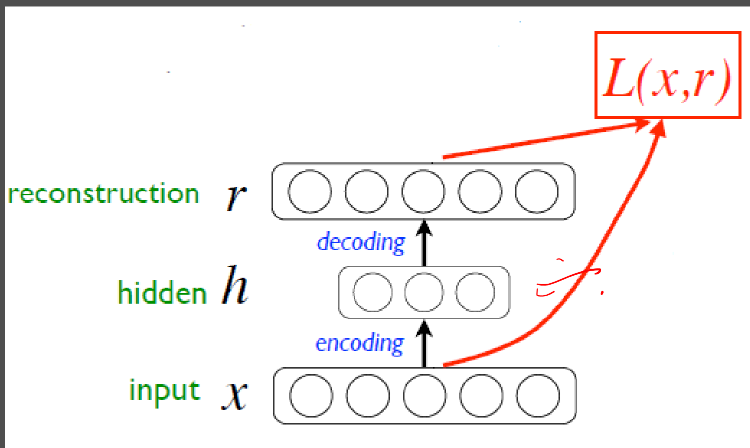
Learned filters



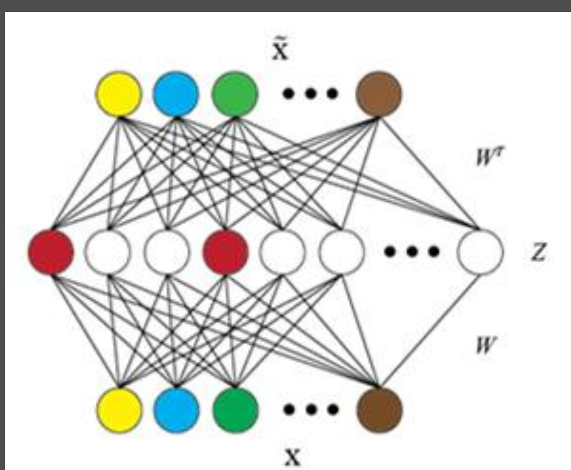
Summary of autoencoders

$x_1 \xrightarrow{\text{encoder}} 1 \xrightarrow{\text{decoder}} x_1$

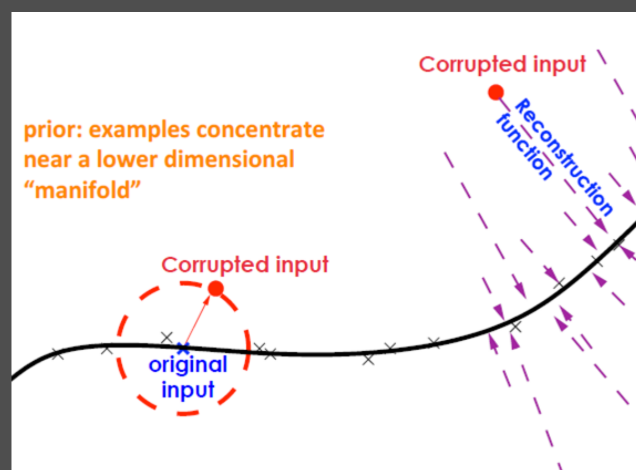
$x_2 \longrightarrow 2 \longrightarrow x_2$



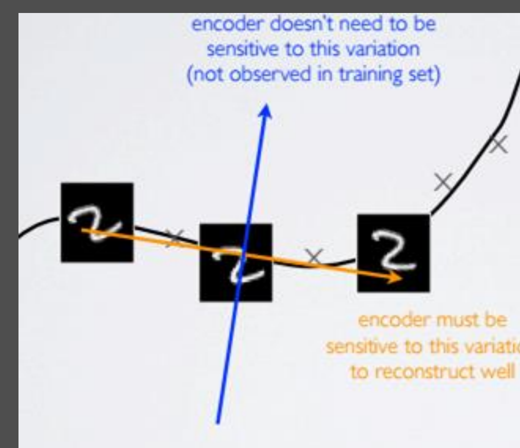
- AE learns low-dimensional representation
- Linear AE same as PCA
- Non-linear AE may learn manifold
- Problem: identity mapping



Sparse autoencoder



Denoising autoencoder



Contractive autoencoder

Applications of AE

- Low-dimensional/manifold representation
- Compression of data
- Use this as feature for classification/regression
- Image restoration such as denoising, inpainting