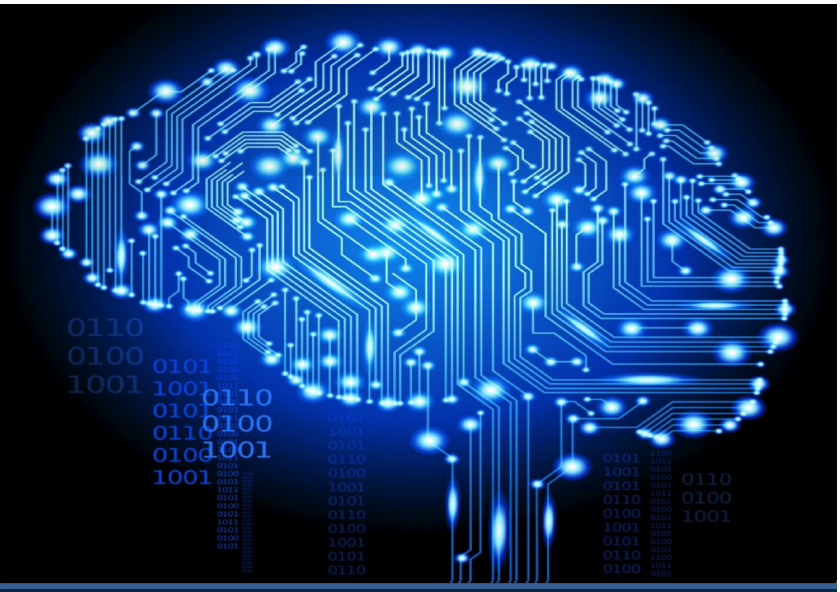
Prof. Saerom Park

Department of Convergence Security Engineering
psr6275@sungshin.ac.kr



Autoencoders

Reference

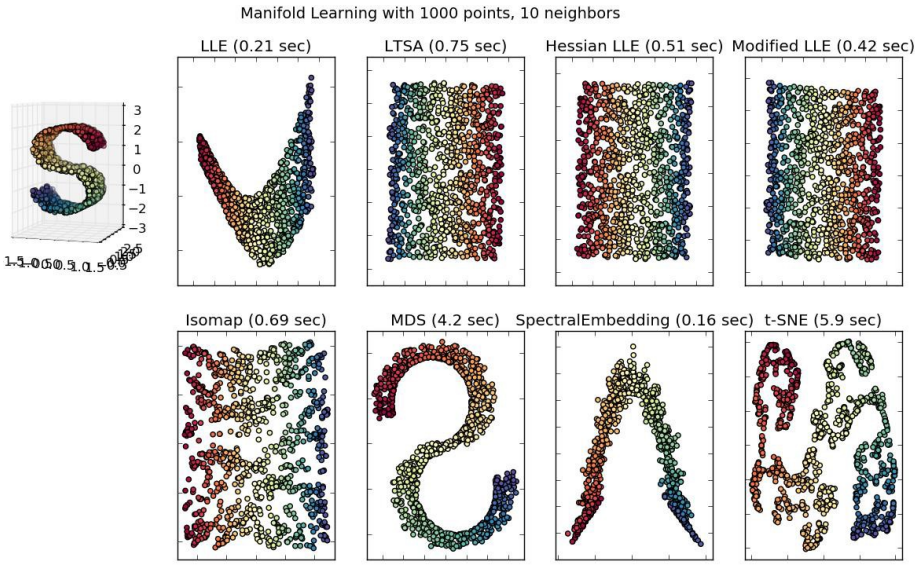
- [Goodfellow et al. (2016), CH14] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning, The MIT Press, 2016.
- [Hinton. (2015)] G. Hinton, Online course on Neural Networks for Machine Learning.

  https://www.coursera.org/learn/neural-networks
- [Larochelle. (2014)] H. Larochelle, Online course on Neural Networks.

  http://info.usherbrooke.ca/hlarochelle/neural_networks/content.html
- [Gavves. (2017)] E. Gavves, UvA Deep Learning Course.

  http://uvadlc.github.io/
- Borrows slides (some modified) from Larochelle & Gavves

The manifold hypothesis



Prof. Saerom Park

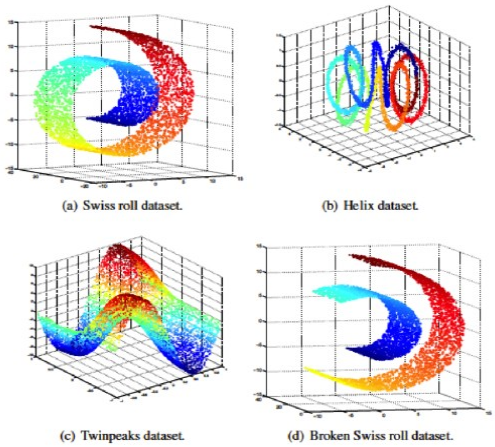
Data in theory

- One image: $256^{256*256*3}$
- 256 height, 256 width, 3 RGB channels, 256 pixel intensities
- Each of these images is like the one in the background
- For text the equivalent would be generating random letter sequences
  - qgkhlkjijxskmbisuwephrhudskneyaeajdzhowieyqwhfnago

Source: http://karpathy.github.io/

High dimensional spaces and the manifold hypothesis

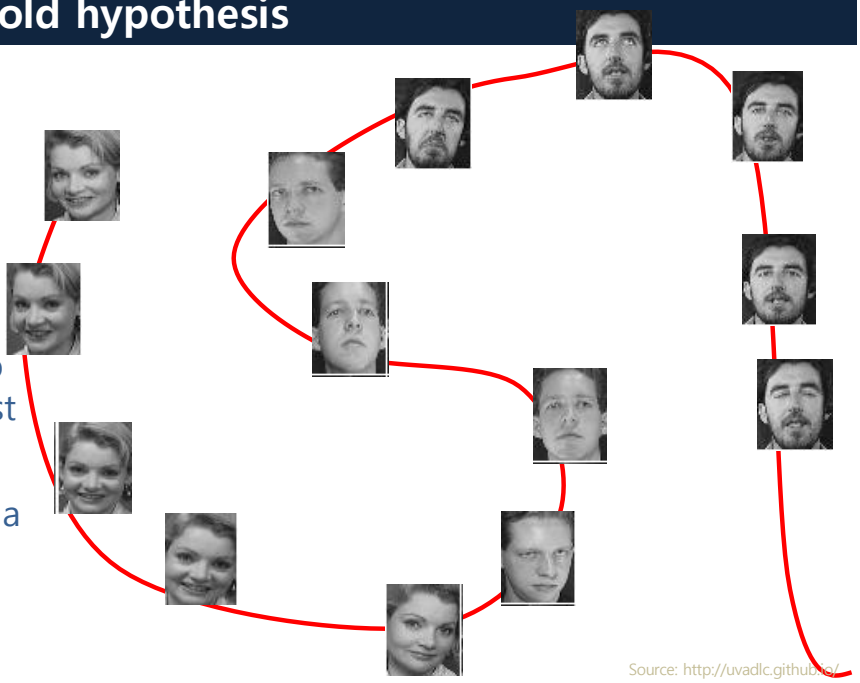
- Manifold hypothesis: natural data lives in a low-dimensional non-linear manifold
- Or equivalently, data is concentrated with high probability in a small non-linear region of the high-dimensional space
- See Goodfellow's 5.11.3



Source: http://uvadlc.github.io/

Data in reality: The manifold hypothesis

- Data live in manifolds
- A manifold is a latent structure of much lower dimensionality
- $\dim_{manifold} \ll \dim_{data}$
- Nobody "forces" the data to be on the manifold, they just are
- The manifold occupies only a tiny fraction of the possible space



Source: http://uvadlc.github.io/

Data in reality: The manifold hypothesis

- Each image is a either a set of coordinates in the full data space

$$\text{Image} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.8 \\ 0.3 \\ -0.5 \\ -0.3 \\ 0.8 \\ 0.1 \\ -0.4 \end{bmatrix}$$

Source: http://uvadlc.github.io/

Data in reality: The manifold hypothesis

- Or a smaller set of intrinsic manifold coordinates

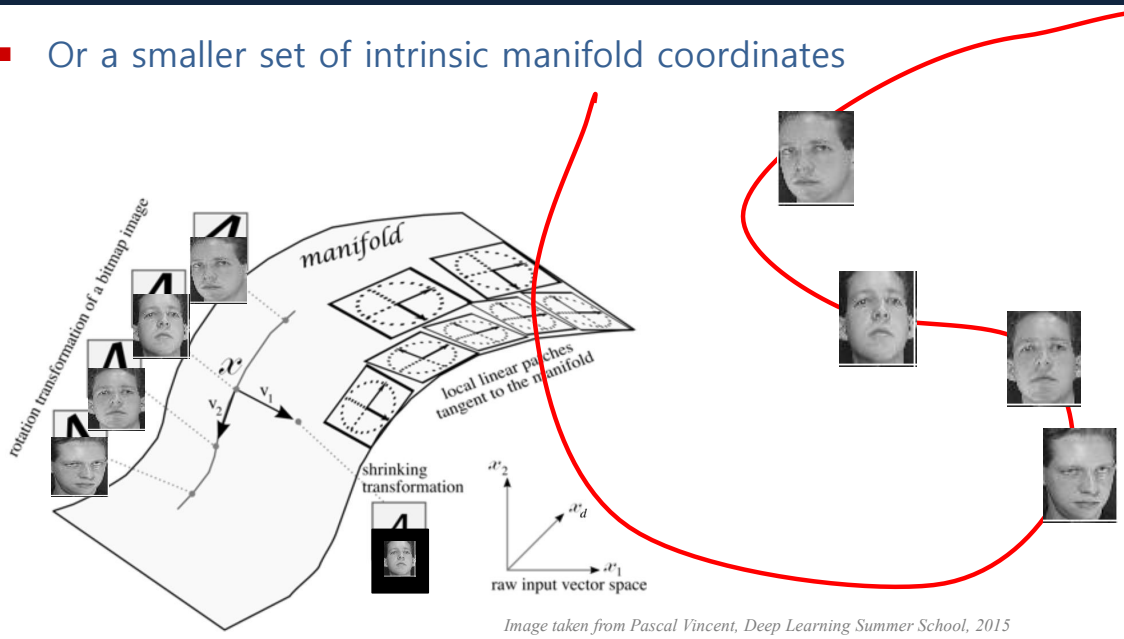


Image taken from Pascal Vincent, Deep Learning Summer School, 2015

Source: http://uvadlc.github.io/

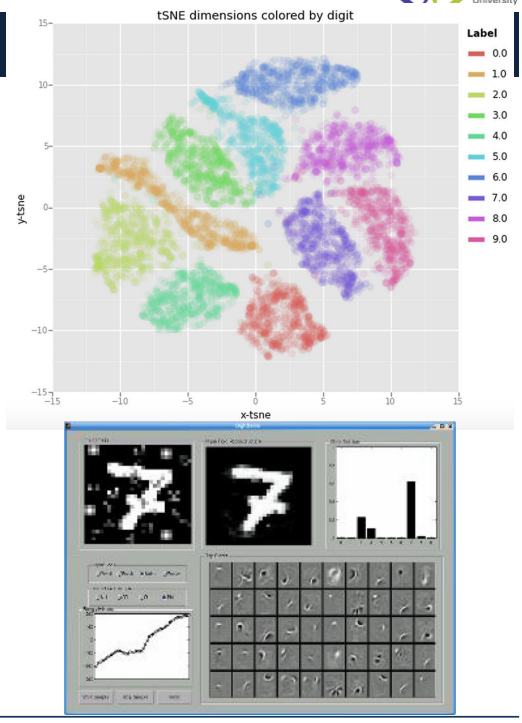
So what?

- Manifold → Data distribution
- Learning the manifold → Learning data distribution and data variances
- How to learn the these variances automatically?
- Unsupervised and/or generative learning

Source: http://uvadlc.github.io/

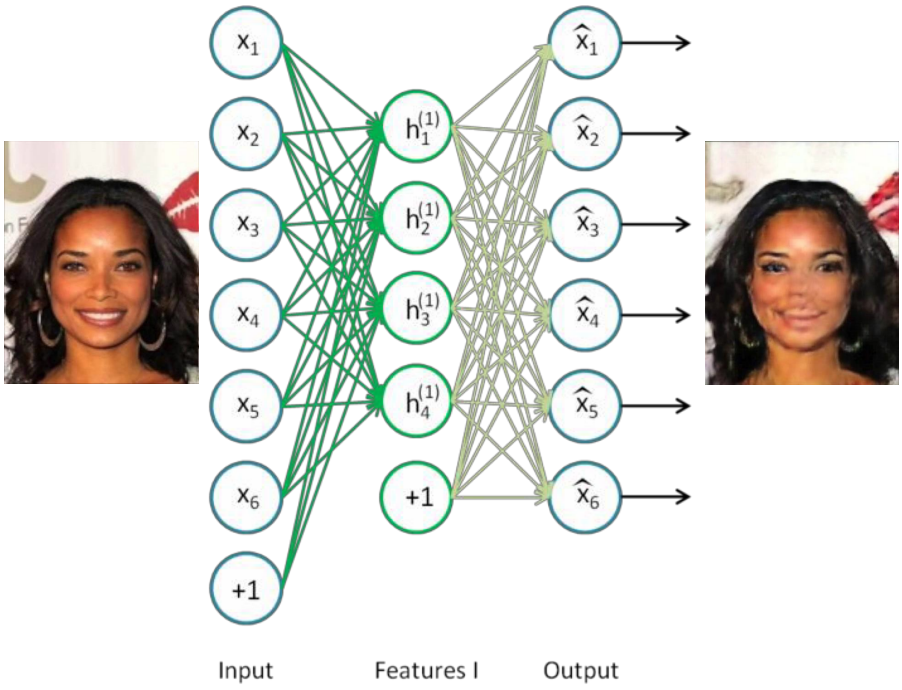
Why unsupervised learning?

- Much more unlabeled than labeled data
  - Large data → better models
  - Ideally not pay for annotations
- What is even the correct annotation for learning data distribution and/or data variances
- Discovering structure
  - What are the important features in the dataset



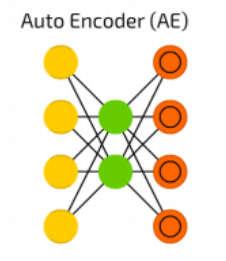
Autoencoders

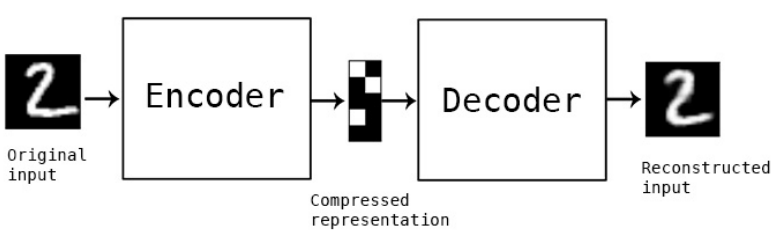
Prof. Saerom Park

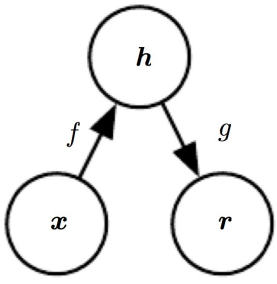


Autoencoders

- Autoencoders
  - Encoder/decoder $\boldsymbol{h} = f(\boldsymbol{x})$ and $\boldsymbol{r} = g(\boldsymbol{h})$ and where $\boldsymbol{h}$ is the low-dimensional representation of $\boldsymbol{x}$ and $\boldsymbol{r}$ is its reconstruction
  - $\boldsymbol{h}$ names: features, representation, code, embedding, latent variables
  - Encoder and decoder are both neural nets



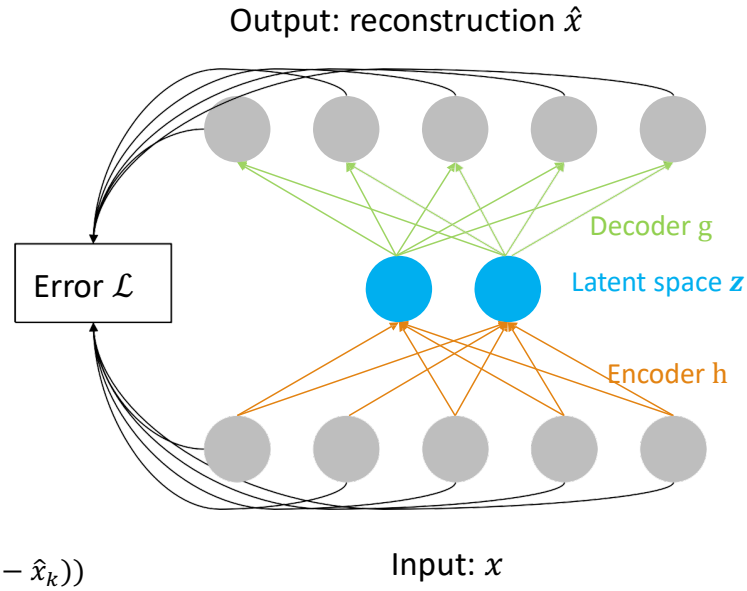




Source: http://uvadlc.github.io/

Canonical Autoencoder

- z is usually after a non-linearity
  - $z = f(x) = \text{ReLU}(Wx + b)$,
  - $g(x) = \sigma(Vf(x) + c)$
- Sometimes, weights are **tied**:
  - $W^T = V$
- Reconstruction error $\mathcal{L}$
  - $\mathcal{L} = \sum_t \ell(x, g(h(x)))$
- For binary inputs: cross-entropy
  - $l = -\sum_k (x_k \log(\hat{x}_k) + (1 - x_k)\log(1 - \hat{x}_k))$



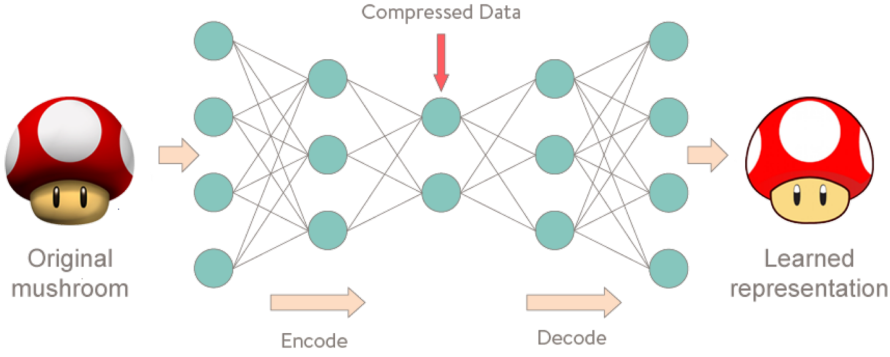
Source: http://uvadlc.github.io/

Autoencoders

- Autoencoders
  - Minimize a loss function (=dissimilarity) between input and reconstruction.

$$MSE = \frac{1}{N}\sum_{i=1}(\boldsymbol{x}_i - \boldsymbol{r}_i)^2 = \frac{1}{N}\sum_{i=1}(\boldsymbol{x}_i - g(f(\boldsymbol{x}_i)))^2$$

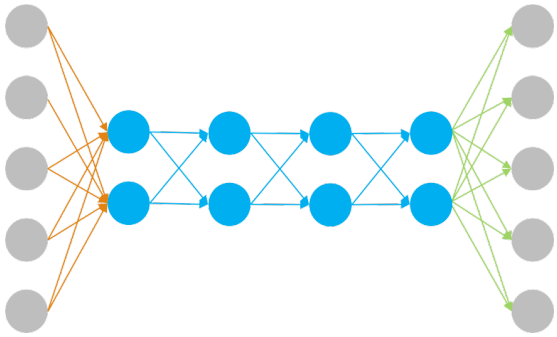
  - Encoder과 Decoder의 층을 쌓아서 **Stacked Auto Encoder**로 확장가능



Source: http://uvadlc.github.io/

Stacking Autoencoders

- Stacking layers on top of each other
- Bigger depth → higher level abstractions
- Slower training



Source: http://uvadlc.github.io/

Auto-encoders maximize the mutual information

- Goal of auto-encoder: learn a good representation* $p(\boldsymbol{h}|\boldsymbol{x})$ (encoder) of the manifold
  - Find encoder with parameter θ to maximize mutual information ($I(x;h) \geq 0$) where the entropy can be defined as $H(x) = \mathbb{E}[\log p(x)] = -\int p(x) \log p(x)\, dx$

$$\text{argmax}_\theta\, I(x;\, h) = \int p(x,h) \log\left(\frac{p(x,h)}{p(x)p(h)}\right) = \int p(x,h) \log\left(\frac{p(x|h)}{p(x)}\right)$$

$$= \text{argmax}_\theta H(x) - H(x|h) = \text{argmax}_\theta - H(x|h) = \text{argmax}_\theta\, \mathbf{E}_{p(x,h)}[\log p(x|h)]$$

$$= \text{argmax}_\theta\, \mathbb{E}_{p(x)p(h|x)}[\log p(x|h)]$$

  - Approximate $p(x|h)$ with a parametric decoder and use a deterministic encoder. The log-likelihood is:

$$\text{argmax}_{\theta,\theta'}\, \mathbf{E}_{p(x)}[\log p_{decoder}(x|h = f_\theta(x); \theta')]$$

Source: http://uvadlc.github.io/

Gaussian <-> squared Euclidean norm

- Assume that the likelihood has Gaussian density:
  - $p_{decoder}(x|h = f_\theta(x); \theta') = N(g_{\theta'}(f_\theta(x)), \sigma^2 I)$
  - Then

$$\mathbf{E}_{p(x)}[\log p_{decoder}(x|h = f_\theta(x); \theta')] = const. - \frac{1}{2\sigma^2}\mathbf{E}_{p(x)}[\|x - g_{\theta'}(f_\theta(x))\|_2^2]$$

  - Therefore, connecting the mutual information,

$$\max_\theta I(x;\ h) \approx \max_{\theta,\theta'} -\mathbf{E}_{p(x)}\ [\|x - g_{\theta'}(f_\theta(x))\|_2^2]$$

Source: http://uvadlc.github.io/

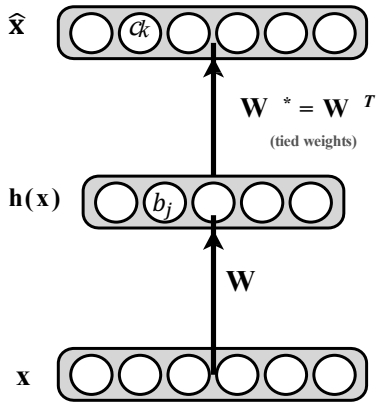
Standard Autoencoder

- The latent space should have fewer dimensions than input
  - Undercomplete representation
  - Bottleneck architecture
- Otherwise (overcomplete) autoencoder might learn the identity function
  - $W \propto I \Longrightarrow \tilde{x} = x \Longrightarrow \mathcal{L} = 0$
  - Assuming no regularization
  - Often in practice still works though
- Also, if $z = Wx + b$ (linear) autoencoder learns same subspace as PCA

Source: http://uvadlc.github.io/

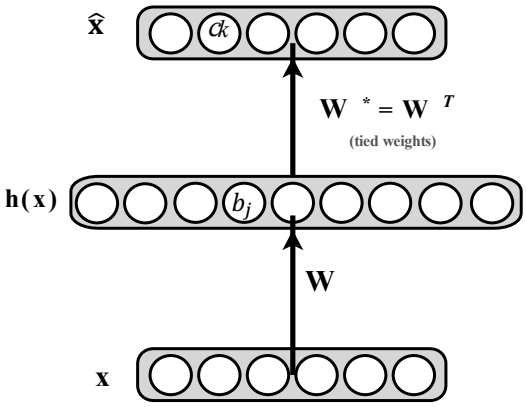
Undercomplete Hidden Layer

- Undercomplete representation
  - Hidden layer is under-complete if smaller than the input layer
    - hidden layer ''compresses'' the input
    - will compress well only for the training distribution
    - Tied weight can regularize it in some extent
  - Hidden units will be
    - good features for the training distribution 
    - but bad for other types of input 



Overcomplete Hidden Layer

- Over-complete representation
  - Hidden layer is over-complete if greater than the input layer
    - no compression in hidden layer
    - each hidden unit could copy a different input component
    - Tied weight can regularize it in some extent
  - No guarantee that the hidden units will extract meaningful structure



Learned filters: over vs. undercomplete

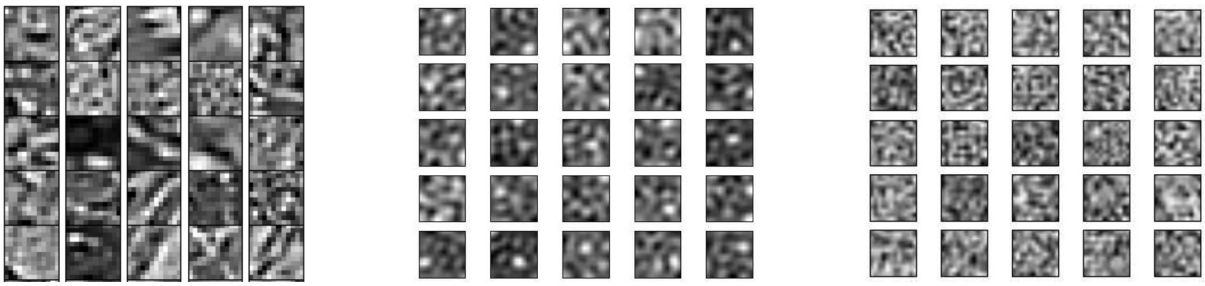


Figure 5: Regular autoencoder trained on natural image patches. *Left:* some of the 12×12 image patches used for training. *Middle:* filters learnt by a regular *under-complete* autoencoder (50 hidden units) using tied weights and L2 reconstruction error. *Right:* filters learnt by a regular *over-complete* autoencoder (200 hidden units). The under-complete autoencoder appears to learn rather uninteresting local blob detectors. Filters obtained in the over-complete case have no recognizable structure, looking entirely random.

[Vincent et al.'10]

Source: http://uvadlc.github.io/

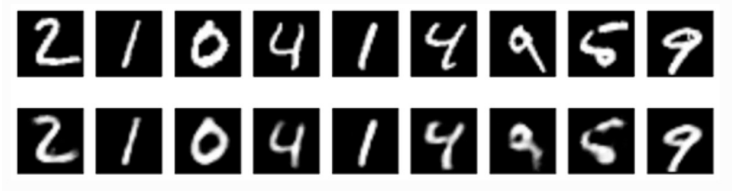
Example: deep auto-encoder in Keras

- 3-layer encoder and 3-layer decoder, under complete.
- The output is sigmoid because we want to get black& white images (on MNIST). Other activations are ReLU.
- Dense (=fully connected) layers. But they can be CNN.

```
input_img = Input(shape=(784,))
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)

decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
decoded = Dense(784, activation='sigmoid')(decoded)
```

[https://goo.gl/9kCxqz Chollet]



Source: http://uvadlc.github.io/

Regularized auto-encoders

- Alternative to undercomplete:
  - use a regularizer to constraint the objective
  - $\frac{1}{N}\sum_{i=1} L(\boldsymbol{x}_i, g(f(\boldsymbol{x}_i)))^2 + \Omega(\boldsymbol{h})$
- In supervised learning,
  - regularizers **reduce the capacity** of the model **to overfit** the training set
- In unsupervised learning,
  - we need them to be **invariant to nuisance** factors (=irrelevant noise) in the data…
  - it is actually the same thing! (overfitting)
- Interpretation:
  - those are **bottlenecks** that allows us to **compress** the data into a useful representation, robust to irrelevant variations of the training data

Source: http://uvadlc.github.io/

Sparse auto-encoder

- Sparsity-inducing regularizer:

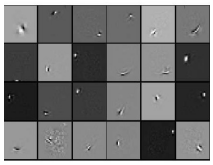
$$\frac{1}{N}\sum_{i=1} L(\boldsymbol{x}_i, g(f(\boldsymbol{x}_i))) \quad + \lambda\|\boldsymbol{h}\|_1$$

- Analogue to use the L1-norm in supervised learning.
  - Effect: pushes many components to **exact 0**
- Probabilistic interpretation:
  - train the auto-encoder with maximum likelihood with a Laplace prior on the code (the latent variable):

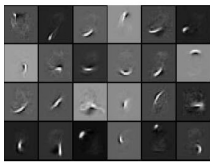
$$p(h) = \frac{\lambda}{2} e^{-\lambda\|\boldsymbol{h}\|_1}$$

Source: http://uvadlc.github.io/

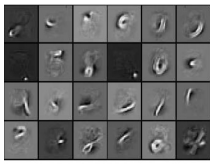
Filters of a sparse auto-encoder



(a) $k = 70$



(b) $k = 40$

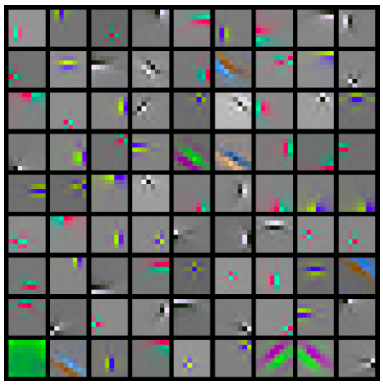


(c) $k = 25$

more sparsity

On MNIST

$$\hat{z}_i = \underset{z}{\operatorname{argmin}} \|x_i - Wz\|_2^2 \quad s.t. \quad \|z\|_0 < k \qquad (1)$$



On CIFAR10

[Makhazani&Frey'14]

Source: http://uvadlc.github.io/

Application: dimensionality reduction

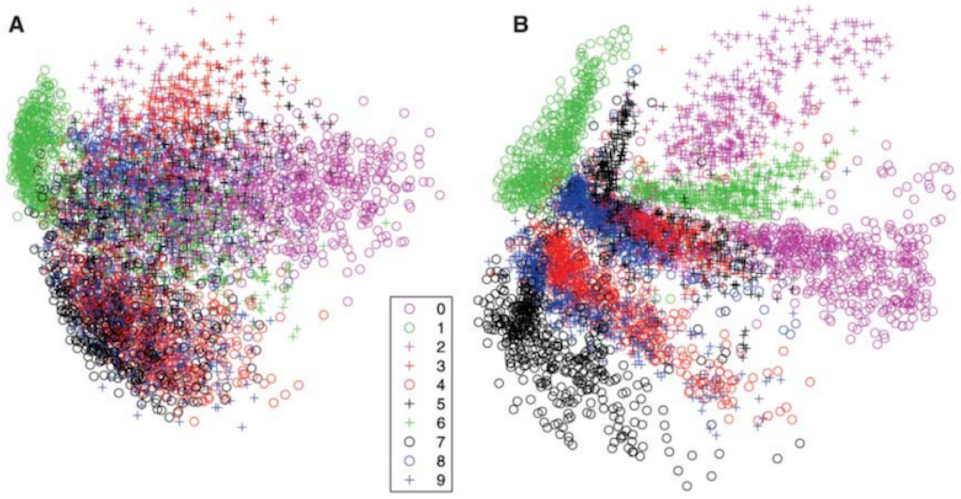
- Fix the representation size to M.
  - Then we can reduce the dimensionality of the data to M.
- Advantages:
  - Less memory
  - Less time consumption for any following algorithm (e.g. supervised learning)
- With respect to PCA:
  - Pro: more meaningful representation, less information discarded
  - Cons: harder and slower to train

Source: http://uvadlc.github.io/

Application: visualization

- Dimensionality reduction onto 2D or 3D for visualization

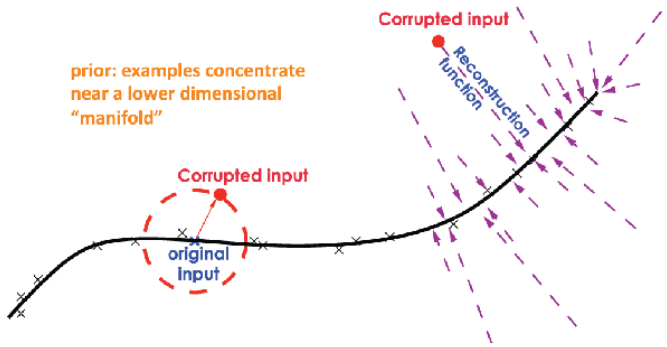
Fig. 3. (**A**) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (**B**) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (*8*).



[Hinton&Salakhutdinov'06]

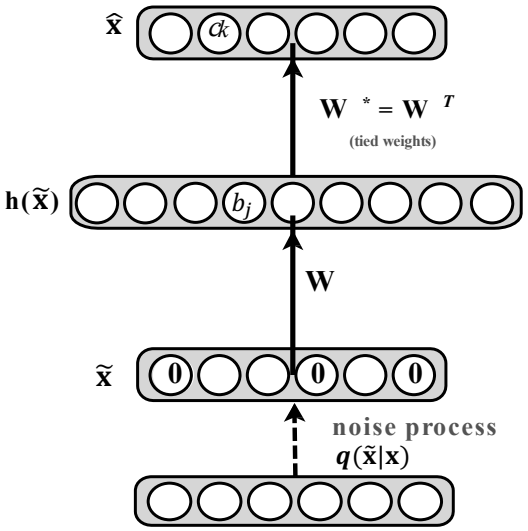
Source: http://uvadlc.github.io/

Other Autoencoders



Denoising autoencoder

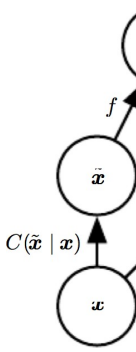
- Idea: representation should be robust to introduction of noise:
  - Masking noise: random assignment of subset of inputs to 0, with probability v
  - Gaussian additive noise: noise follows Gaussian distribution
- Reconstruction $\hat{\mathbf{x}}$ from the corrupted input $\tilde{\mathbf{x}}$
  - Loss function compares $\hat{\mathbf{x}}$ reconstruction with the noiseless input x
  - $\frac{1}{N}\sum_{i=1} L(\boldsymbol{x}_i, g(f(\tilde{\boldsymbol{x}}_i)))$

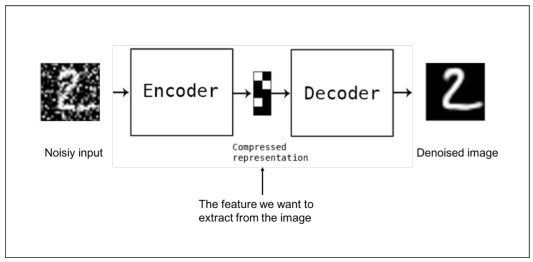


Denoising auto-encoder revisited

- Introduce the noisy transition as a stochastic operation in the computational graph
  1. Sample $\boldsymbol{x}$ from the data
  2. Sample a corrupted version by $C(\tilde{\boldsymbol{x}}|\boldsymbol{x})$
  3. Train the auto-encoder $g(f(\tilde{x}))$ to reconstruct $\boldsymbol{x}$
- The loss function as negative log likelihood is:

$$-\mathbf{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\,\mathbf{E}_{\tilde{\boldsymbol{x}}\sim C(\tilde{\boldsymbol{x}}|\boldsymbol{x})}[\log p_{decoder}(\boldsymbol{x}|\boldsymbol{h}=f(\tilde{\boldsymbol{x}}))]$$

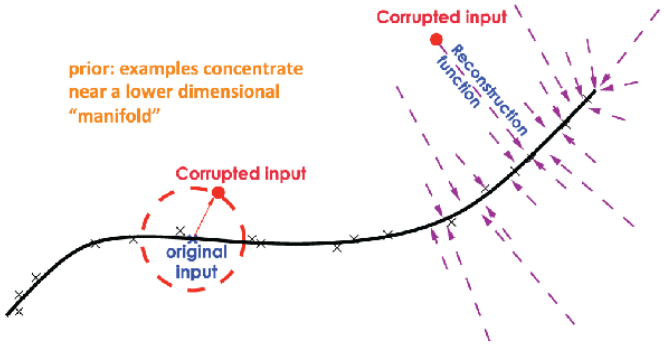




Source: http://uvadlc.github.io/

Denoising Autoencoders

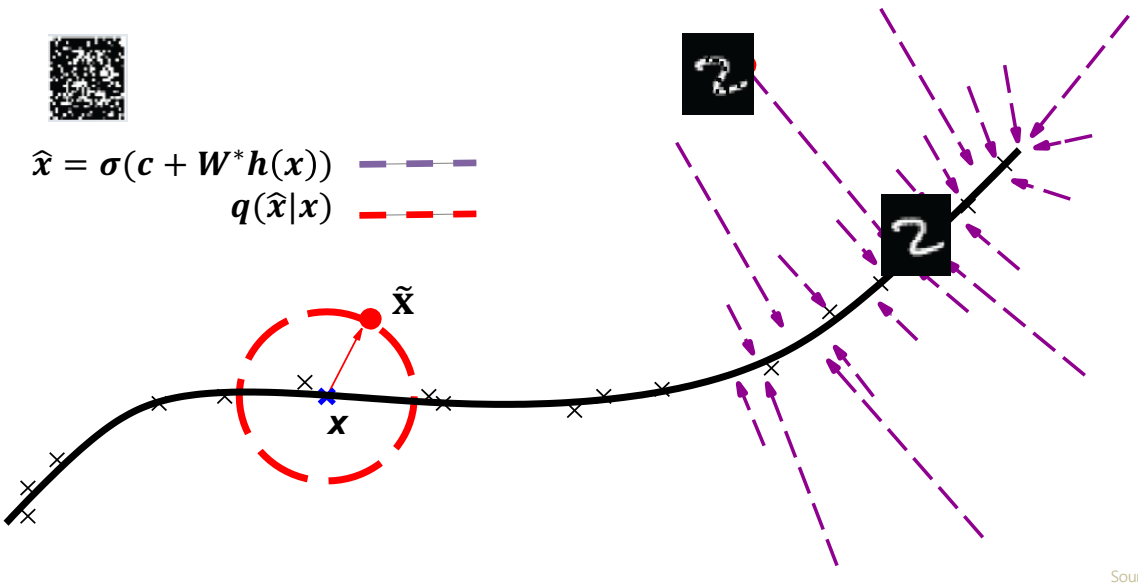
- Denoising Autoencoders
  - 이렇게 학습을 진행 할 경우 noise에 강한(robust) 네트워크가 생성됨.
  - 아래 그림의 오른쪽 예시처럼, 실제 manifold에서 벗어난 input이 들어오더라도 manifold상으로 움직여 주는 효과가 있음.



Manifold = https://en.wikipedia.org/wiki/Manifold

Source: http://uvadlc.github.io/

Denoising Autoencoders



Source: http://uvadlc.github.io/

Application of denoising auto-encoder: image denoising

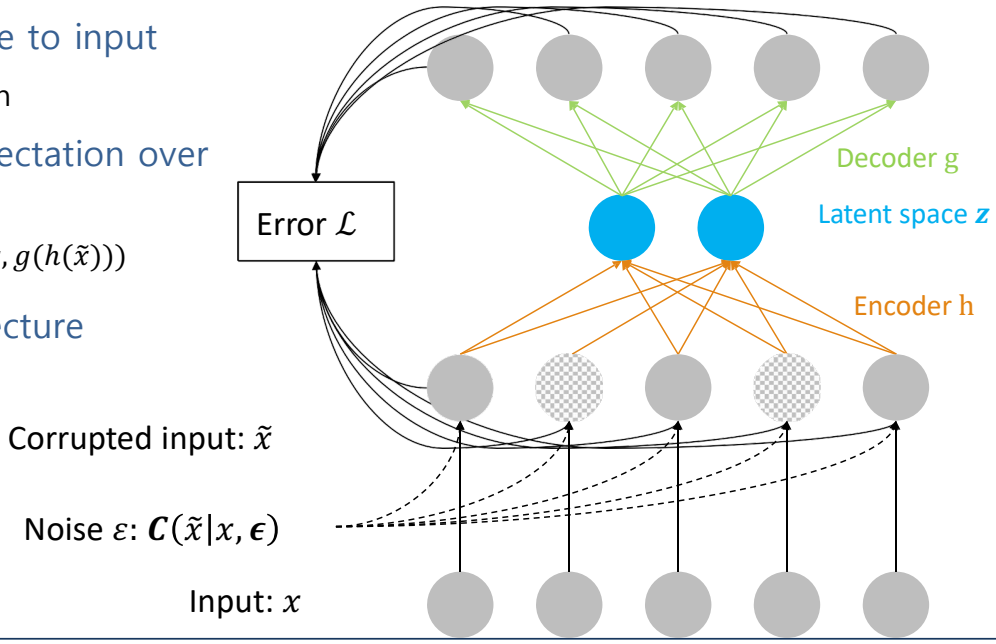
- Apply the **whole auto-encoder** to real images that are affected by noise => output denoised images.
- To work well, the real noise has to be similar to Gaussian though

[hbps://goo.gl/9kCxqz Chollet]

Source: http://uvadlc.github.io/

Denoising Autoencoder (undercomplete)

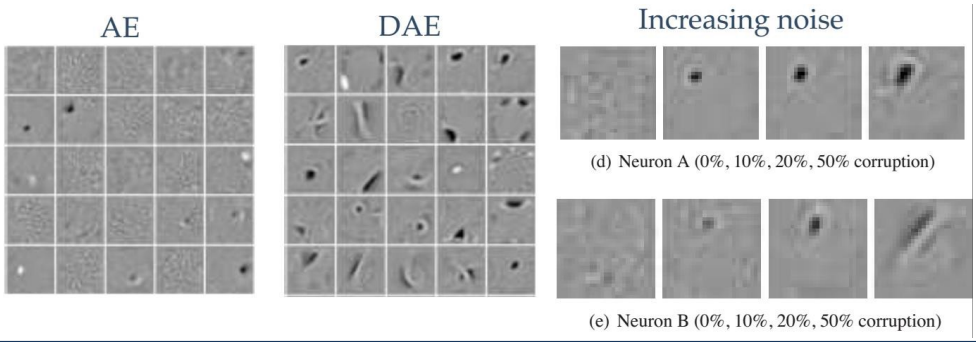
- Add random noise to input
  - Dropout, gaussian
- Loss includes expectation over noise distribution
  - $\mathcal{L} = \sum_t \mathbb{E}_{c(\tilde{x}|x)} \ell(x, g(h(\tilde{x})))$
- Bottleneck architecture



Source: http://uvadlc.github.io/

Denoising Autoencoder (Overcompelete)

- The network does not overlearn the data because of noises
  - Can even use over-complete latent spaces
- Model forced to learn more intelligent, robust
  - Learn to ignore noise or trivial solutions(identity)
  - Focus on "underlying" data generation process



Source: http://uvadlc.github.io/

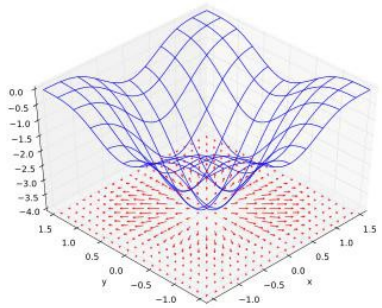
Contractive auto-encoder

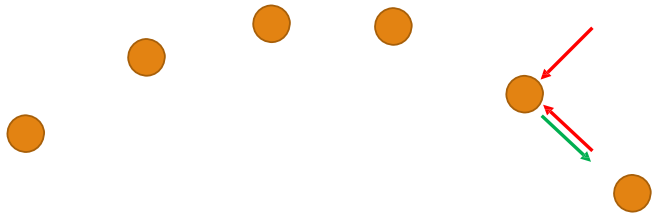
- The compromise here:
  - Contractive: resist to local perturbations of the input **by squashing** their representation through the encoder.
  - But at the same time minimize the reconstruction error
  - => Therefore: only the irrelevant directions of variation of the input will be contracted by the encoder
- Measures for resistance to local perturbations
  - Jacobian $\frac{\partial h}{\partial x}$ can be a solution
  - Jacobian is expensive, but we can compute it by auto-diff tools as usual. A finite difference approximation works too.

Source: http://uvadlc.github.io/

Some geometric intuition

- Gradients show sensitivity of function around a point
  - Combined Loss:
  - $\mathcal{L} = \sum_d \ell(x, g(h(x))) + \lambda \left\| \frac{\partial h}{\partial x} \right\|^2$
  - Regularization term: penalizes sensitivity to all directions
  - Reconstruction term: enforces sensitivity only to direction of manifold

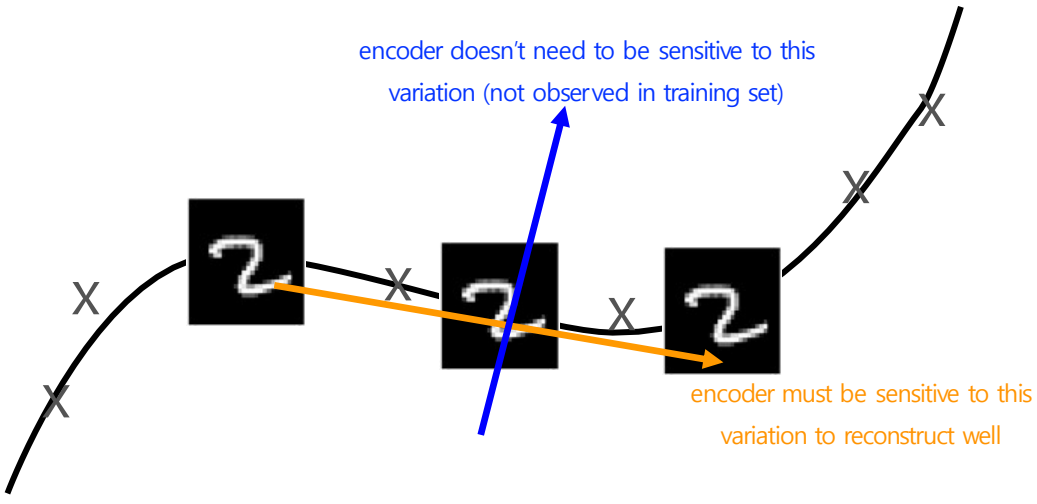




Source: http://uvadlc.github.io/

Contractive autoencoder

- Illustration:



Source: http://uvadlc.github.io/

Some geometric intuition: let's get real

- Let's try to check where the gradients are sensitive around our manifold
  - Gradients → Jacobian
- Strongest directions of Jacobian → Compute SVD of Jacobian
  - $\frac{\partial \mathrm{h}}{\partial \mathrm{x}} = USV^T$
- Take strongest singular values in S and respective vectors from U
  - These are our strongest tangents → directions of Jacobian of most sensitivity
- Hopefully,
  - the tangents sensitive to direction of the manifold, not random
- How to check?
  - Visualize!!

Source: http://uvadlc.github.io/

Visualization for Singular values

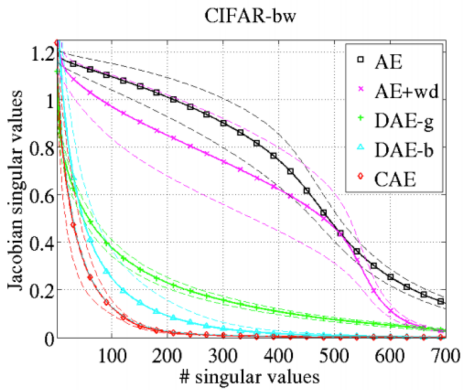


Figure 2. Average spectrum of the encoder's Jacobian, for the CIFAR-bw dataset. Large singular values correspond to the local directions of "allowed" variation learnt from the dataset. The CAE having fewer large singular values and a sharper decreasing spectrum, it suggests that it does a better job of characterizing a *low-dimensional manifold* near the training examples.

Which Autoencoder? Denoising vs. Contractive

- Both the denoising and contractive autoencoder perform well
  - Advantage of denoising autoencoder: **simpler to implement**
    - requires 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
  - They penalise different things:
    - Denoising: reconstruction (g + f) robust to noise
    - Contractive: representation (f) robust to noise

Source: http://uvadlc.github.io/

Reading material & references

- http://www.deeplearningbook.org/
  - Part II: Chapter 13,14
- Hinton & Salakhutdinov, Semantic Hashing 2006
- Vincent et al., Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion, JMLR 2010
- Rifai et al., Contractive Auto-Encoders: Explicit Invariance During Feature Extraction, ICML 11
- LeCun & Ranzato, Deep learning tutorial, ICML 13 hbps://goo.gl/37GbPS
- Makhezani & Frey, k-sparse autoencoders, ICLR14