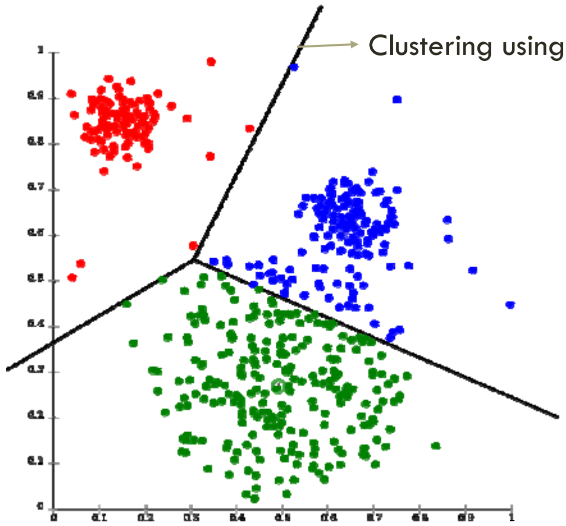
AUTOENCODER AND SEMISUPERVISED LEARNING

Most of this material is from Guy Golan's slides.

AGENDA

- Unsupervised Learning
- Autoencoder (AE)
- Convolutional AE (with code)
- Regularization: Sparse
- Denoising AE
- Stacked AE
- Contractive AE

INTRODUCTION TO UNSUPERVISED LEARNING



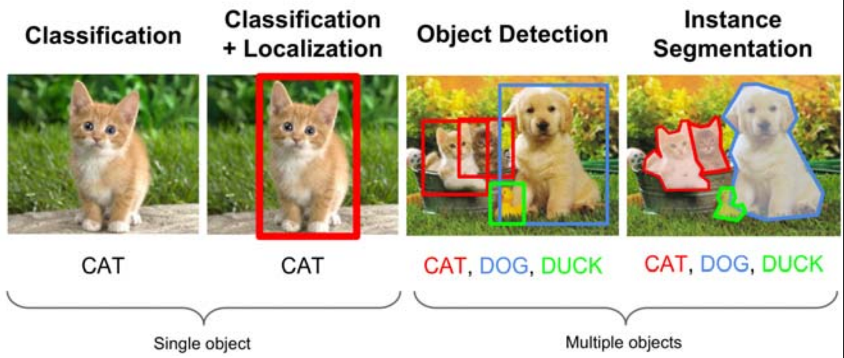
SUPERVISED LEARNING

Supervised Learning

Data: (X,Y)

Goal: Learn a Mapping Function f where:

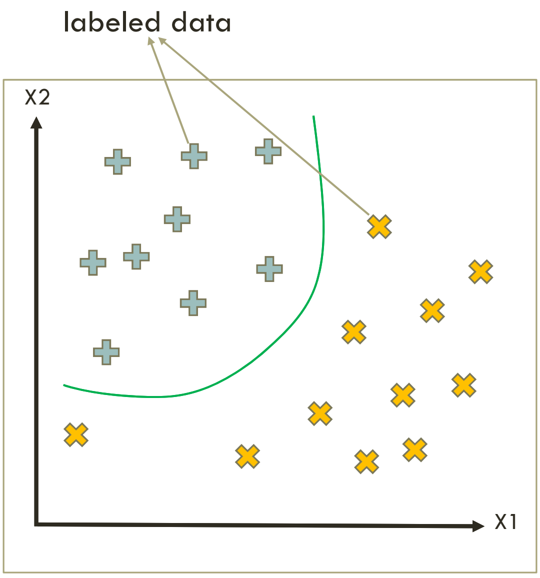
$$f(X) = Y$$



SUPERVISED LEARNING

Examples: Classification.

Decision Trees	Naïve Bayes
kNN (k-Nearest Neighbor)	SVM (Support Vector Machine)
Perceptron (logistic regression)	Multi Layer Perceptron (neural network)



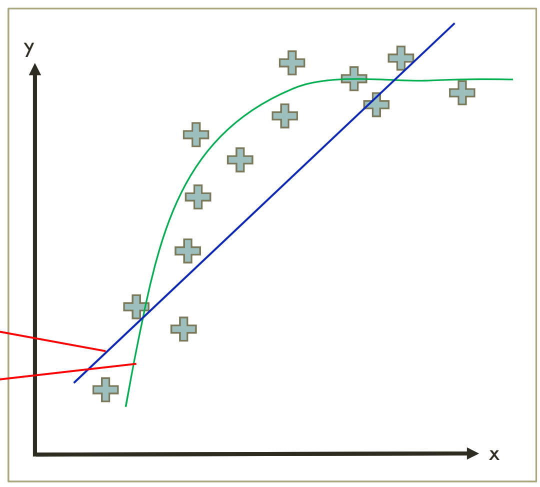
Classification

SUPERVISED LEARNING

Examples: Regression.

Linear Regression

Logistic Regression



Regression

SUPERVISED LEARNING VS UNSUPERVISED LEARNING

01

What happens when our labels are noisy?

- Missing values.
- Labeled incorrectly.

02

What happens where we don't have labels for training **at all**?

SUPERVISED LEARNING VS UNSUPERVISED LEARNING

Up until now we have encountered in this lecture mostly **Supervised Deep Learning** problems and algorithms.

Lets talk about **Unsupervised Deep Learning**

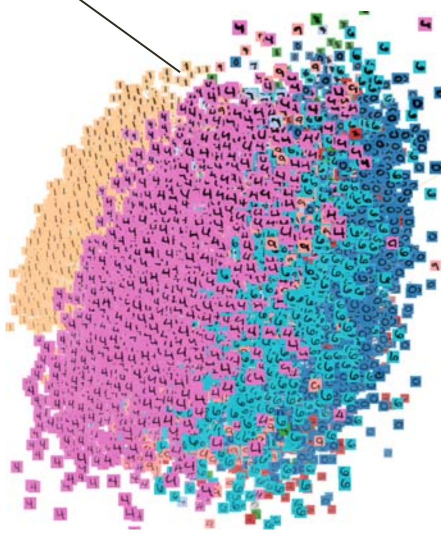
UNSUPERVISED LEARNING

Unsupervised Learning

Data: X (no labels!)

Goal: Learn the structure of the data (learn correlations between features)

Clustering using unlabeled data



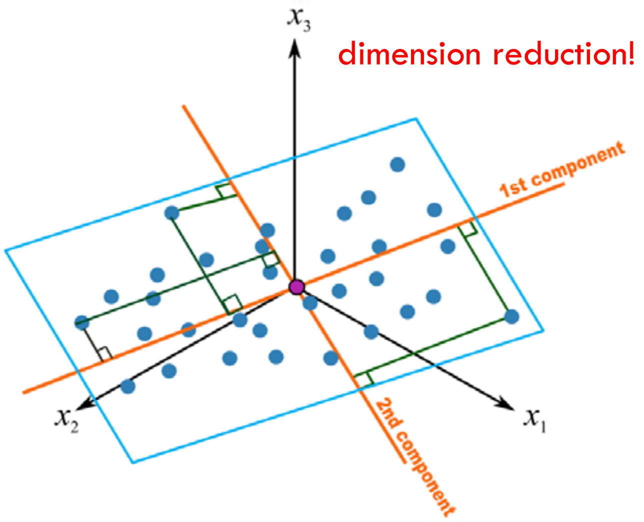
UNSUPERVISED LEARNING

Examples: Clustering, **Compression, Feature & Representation learning, Dimensionality reduction,** Generative models ,etc.

PCA – PRINCIPAL COMPONENT ANALYSIS

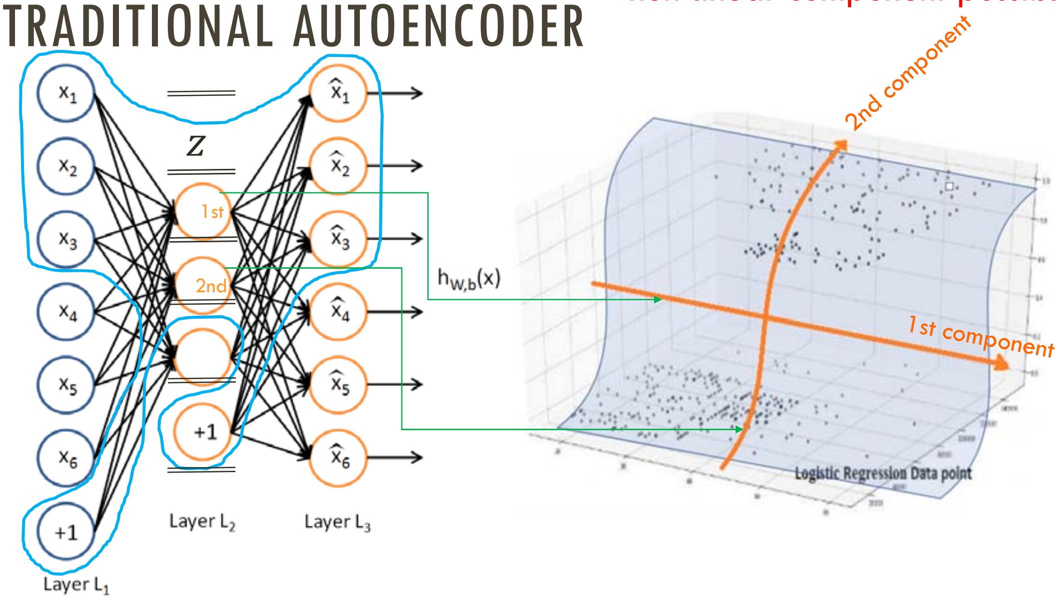
- Statistical approach for data compression and visualization
- Invented by Karl Pearson in 1901
- Weakness: linear components only
- Refer to Statistical Data Analysis for more details !

dimension reduction!



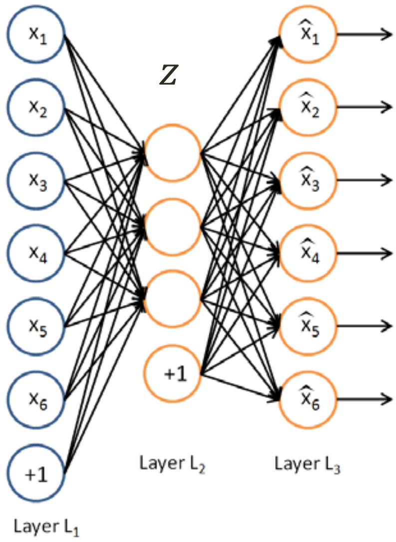
TRADITIONAL AUTOENCODER

- dimension reduction
- non-linear component possible!



TRADITIONAL AUTOENCODER

- Unlike the **PCA,** now we can use activation functions to achieve non-linearity.
- It has been shown that an AE without activation functions shows the **PCA** capability.



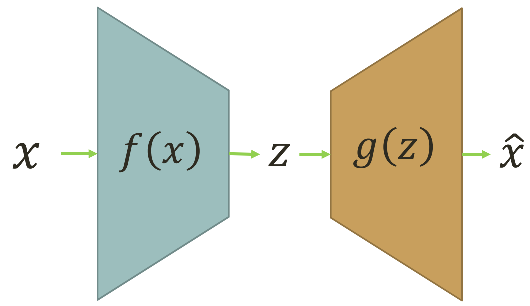
USES

- The autoencoder idea was a part of NN history for decades (LeCun et al, 1987).
- Traditionally, an autoencoder is used for dimensionality reduction and feature learning.
- Recently, the connection between autoencoders and latent space modeling has brought autoencoders to the front of generative modeling.

- **Not used for lossless compression.**
  -Data specific compression.
  -Lossy.

SIMPLE IDEA

- Given data x (no labels), we would like to learn the functions f (encoder) and g (decoder) where:



$$f(x) = s(wx + b) = z$$

and

$$g(z) = s(w'z + b') = \hat{x}$$

s.t $h(x) = g\big(f(x)\big) = \hat{x}$

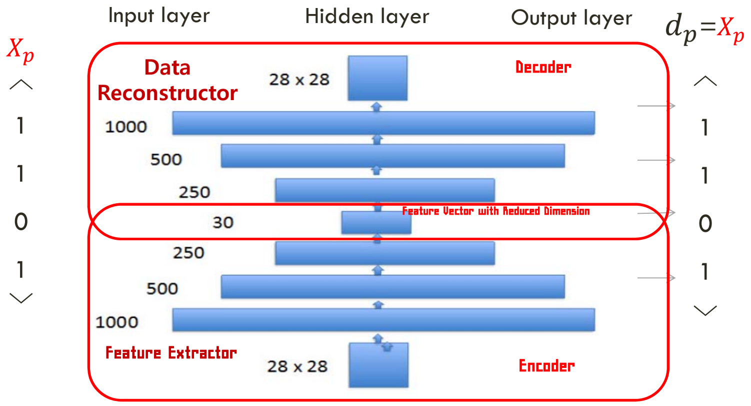
where h is an **approximation** of the identity function.

(z is some **latent** representation or **code** and s is a non-linearity such as the sigmoid)

($\hat{x}$ is x's reconstruction)

Autoencoder

- **Learning? Gradient Decent Method**



SIMPLE IDEA

Learning the identity function seems trivial, but with added constraints on the network (such as limiting the number of hidden neurons or regularization), we can learn information about the structure of the data.

Trying to capture the distribution of the data (data specific!)

TRAINING THE AE

Using **Gradient Descent**, we can simply train the model as any other FC NN with:

- Traditionally with *squared error loss* function

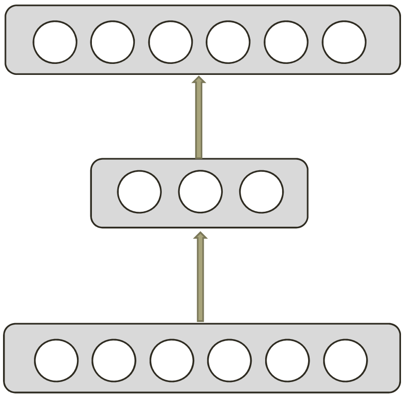
$$L(x, \hat{x}) = \|x - \hat{x}\|^2$$

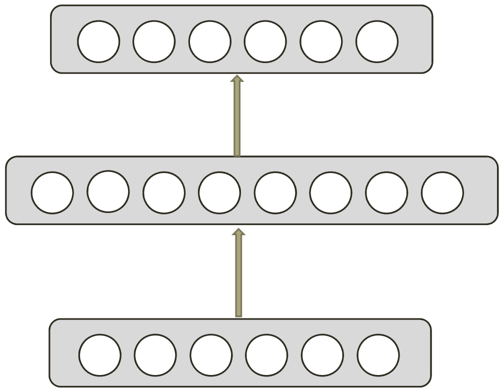
- If our input is interpreted as bit vectors or vectors of bit probabilities, the *cross entropy* can be used

$$H(p, q) = -\sum_x p \log \hat{p}$$

UNDERCOMPLETE AE VS OVERCOMPLETE AE

We distinguish between two types of AE structures:

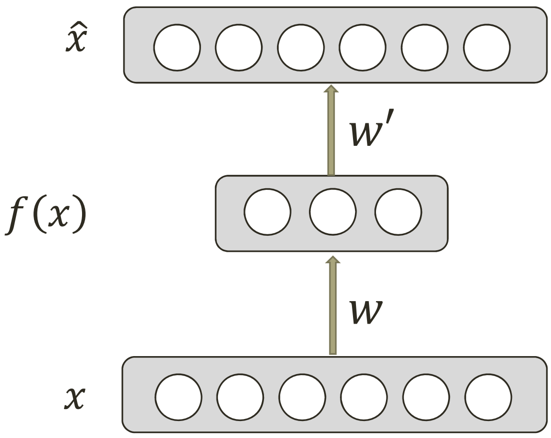




UNDERCOMPLETE AE

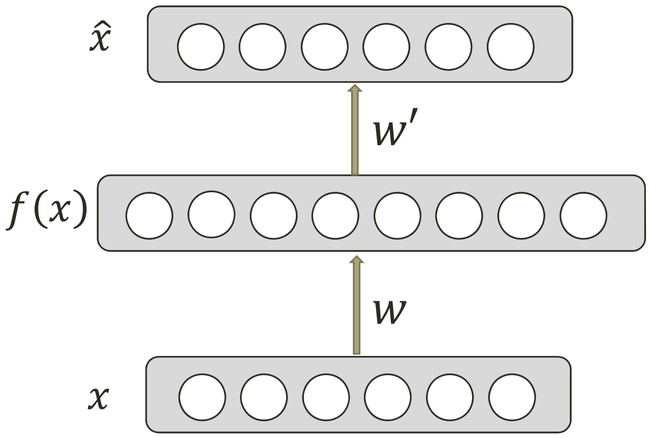
- Hidden layer is **Undercomplete** if it is smaller than the input layer
  - ❑Compresses the input
  - ❑Compresses well only for the training dist.

- Hidden nodes will be
  - ❑Good features for the training distribution.
  - ❑Bad for other types on input

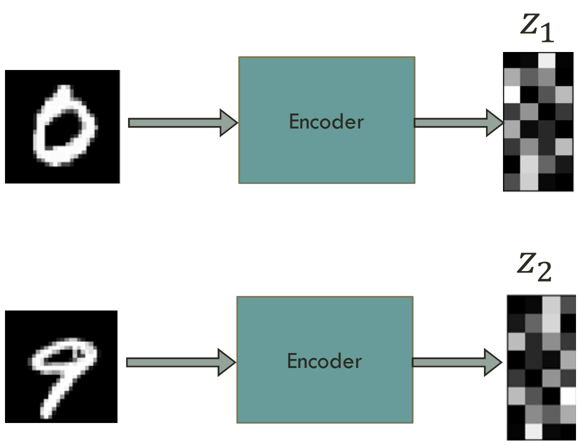


OVERCOMPLETE AE

- Hidden layer is **Overcomplete** if it is greater than the input layer
  - ❑No compression in hidden layer.
  - ❑In a extreme case, each hidden unit could copy a different input.
- No guarantee that the hidden units will extract meaningful structure.
- A higher dimension code helps model a more complex distribution.

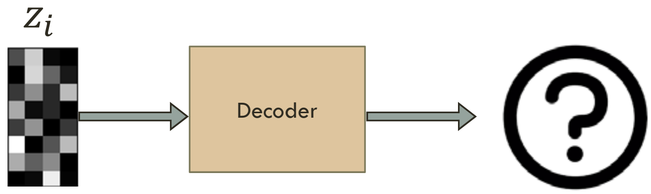


SIMPLE LATENT SPACE INTERPOLATION

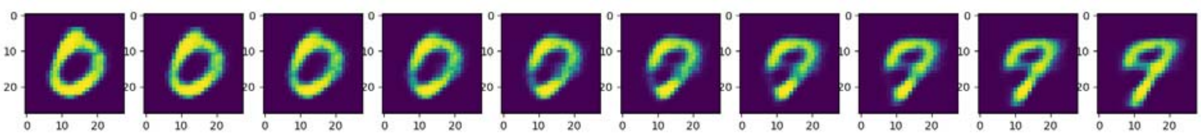


SIMPLE LATENT SPACE INTERPOLATION

$$z_i = \alpha \, z_1 + (1 - \alpha) \, z_2$$

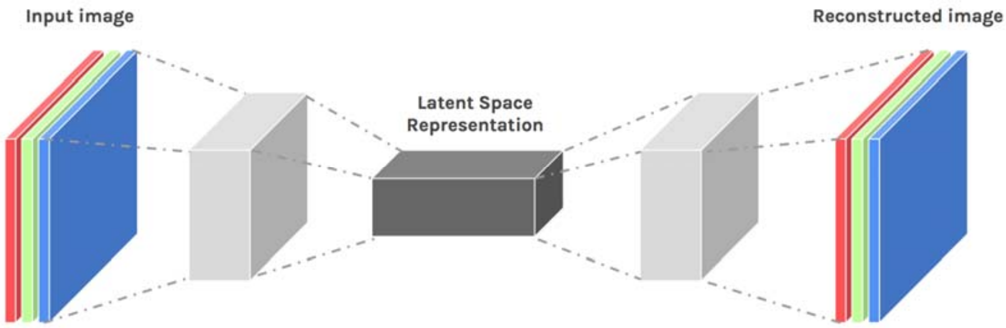


SIMPLE LATENT SPACE – INTERPOLATION



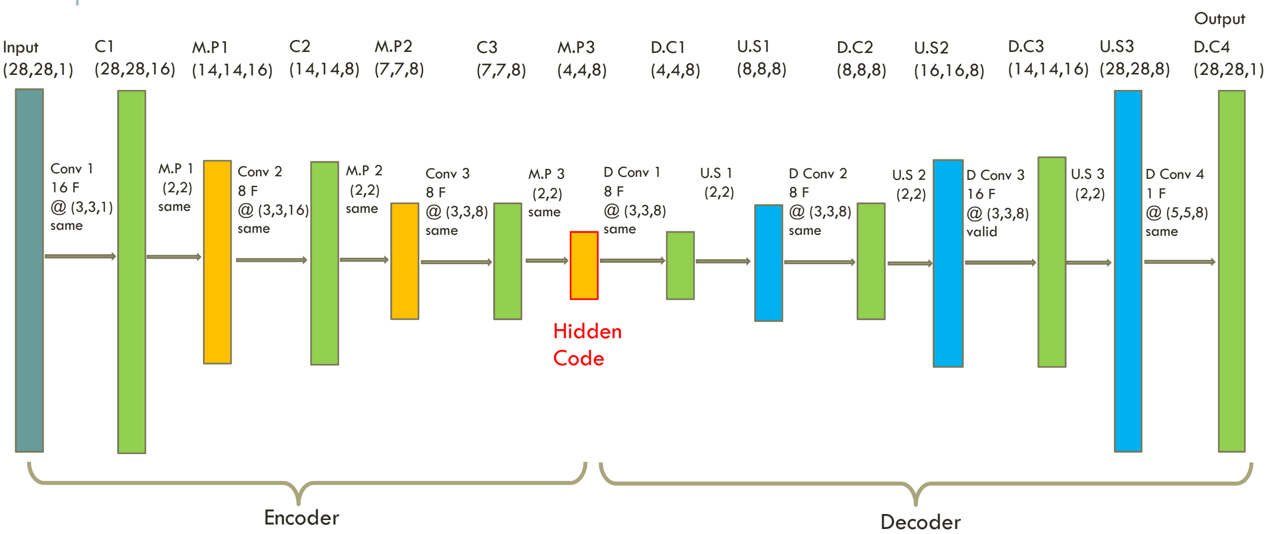
$\alpha = 1$ $\alpha = 0$

CONVOLUTIONAL AE



* Input values are normalized
* All of the conv layers activation functions are relu except for the last conv which is sigm

CONVOLUTIONAL AE

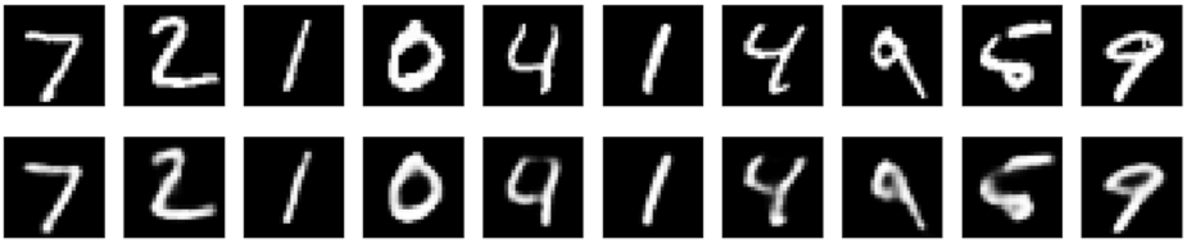


CONVOLUTIONAL AE

- 50 epochs.
- 88% accuracy on validation set.

x

$\hat{x}$



REGULARIZATION

Motivation:

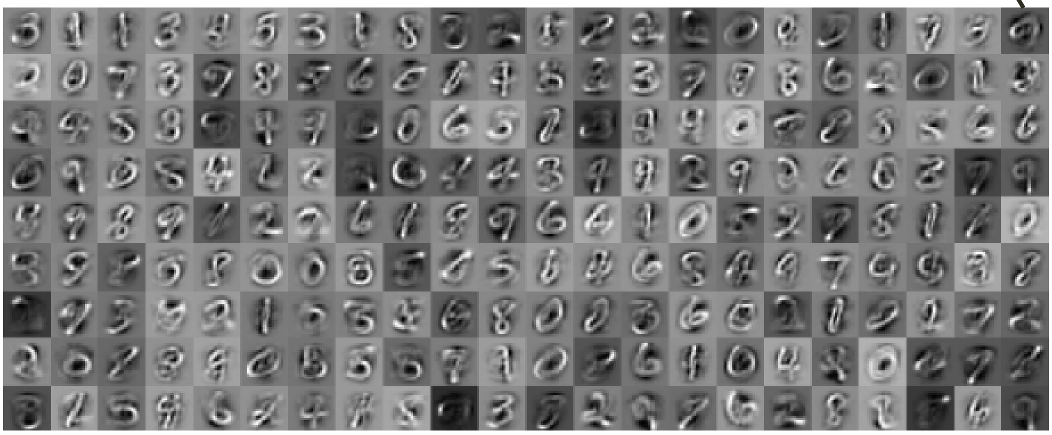
- Assume that we would like to learn meaningful features **without** altering the code's dimensions (Overcomplete or Undercomplete).

The solution: imposing other constraints on the network.

SPARSELY REGULATED AUTOENCODERS

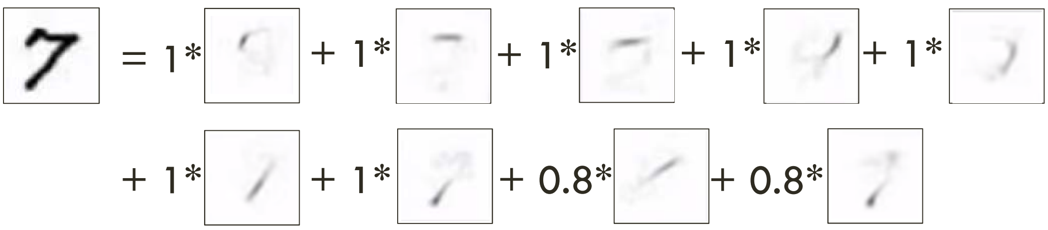
Activation Maps
= Hidden Codes

A bad (non-generalized) example:



SPARSELY REGULATED AUTOENCODERS

- We want our learned features to be as **sparse** as possible.
- With sparse features, we can generalize better.



SPARSELY REGULATED AUTOENCODERS

Recall:

a_j is defined to be the activation of the jth hidden unit (bottleneck) of the autoencoder.

Let $a_j\ (x)$ be the activation of this specific node on a given input x.

SPARSELY REGULATED AUTOENCODERS

Further let,

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j\ \left(x^{(i)}\right)\right]$$

be the average activation of hidden unit j (over the training set).

Thus, we would like to force the constraint:

$$\hat{\rho}_j = \rho$$

where ρ is a "sparsity parameter", typically small. In other words, we want the average activation of each neuron j to be close to ρ.

SPARSELY REGULATED AUTOENCODERS

- We need to penalize $\hat{\rho}_j$ for deviating from ρ.
- Many choices of the penalty term will give reasonable results.

- For example: $$\sum_{j=1}^{n} D_{\mathrm{KL}}(\rho || \hat{\rho}_j)$$

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$
$$= -\sum_i P(i) \log \frac{Q(i)}{P(i)}$$
$$= \underbrace{-\textstyle\sum_i P(i) \log Q(i)}_{\text{Cross Entropy}} - \underbrace{(-\textstyle\sum_i P(i) \log P(i))}_{\text{Self-information Entropy}}$$

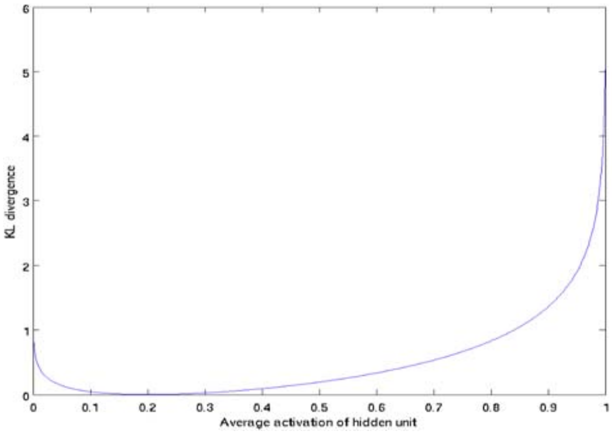
where $KL(\rho|\hat{\rho}_j)$ is a Kullback-Leibler divergence function.

SPARSELY REGULATED AUTOENCODERS

- A reminder:
  - KL is a standard function for measuring how different two distributions are, which has the properties:

$$D_{\mathrm{KL}}(\rho || \hat{\rho}_j) = 0 \text{ if } \hat{\rho}_j = \rho$$

otherwise it is increased monotonically.



$\rho = 0.2$

SPARSELY REGULATED AUTOENCODERS

- Our overall cost functions is now:

$$J(W,b) = J(W,b) + \beta \sum_{j=1}^{n} D_{\mathrm{KL}}\left(\rho || \hat{\rho}_j\right)$$

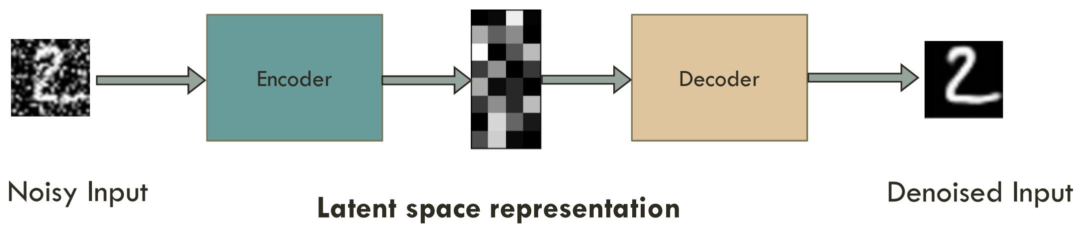
*Note: We need to know $\hat{\rho}_j$ before hand,
so we have to compute a forward pass on all the training set.

DENOISING AUTOENCODERS

Intuition:

- We still aim to encode the input and to NOT mimic the identity function.
- We try to undo the effect of *corruption* process stochastically applied to the input.

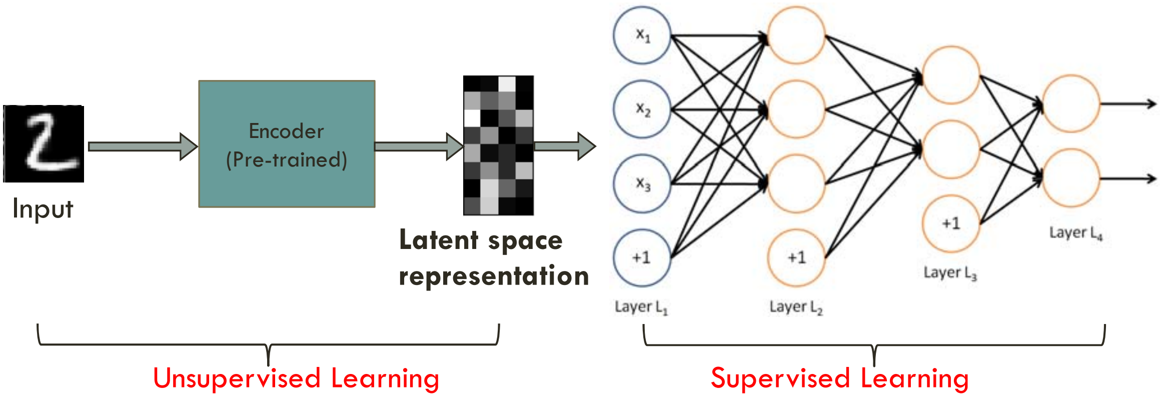
A more robust model



SEMISUPERVISED LEARNING USING (DENOISING) AUTOENCODERS

Use Case:

- Extract robust representation for a NN classifier.



DENOISING AUTOENCODERS

Instead of trying to mimic the identity function by minimizing:

$$L\left(x, g\left(f(x)\right)\right)$$

where L is some loss function

A **DAE** instead minimizes:

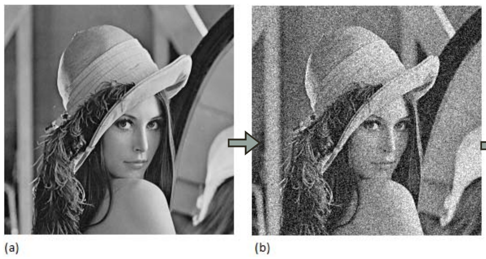
$$L\left(x, g\left(f(\tilde{x})\right)\right)$$

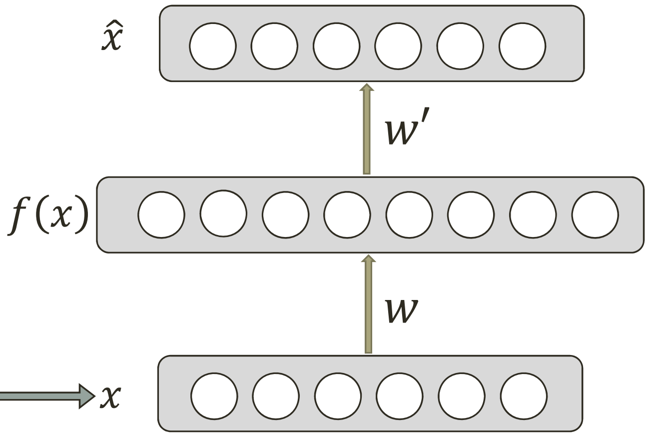
where $\tilde{x}$ is a copy of x that has been corrupted by some form of noise.

DENOISING AUTOENCODERS

Idea: A robust representation against noise:

- Random assignment of subset of inputs to 0, with probability v.
- or Gaussian additive noise.





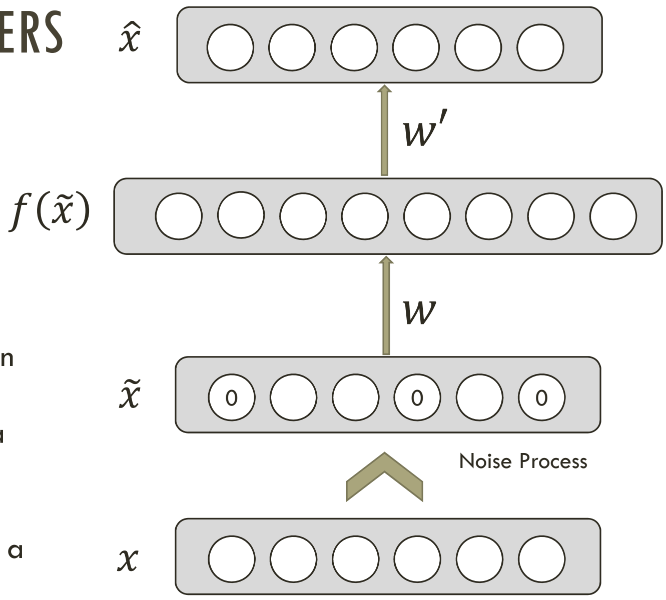
DENOISING AUTOENCODERS

- Reconstruction $\hat{x}$ computed from the corrupted input $\tilde{x}$.
- Loss function compares $\hat{x}$ reconstruction with the noiseless x.

❖ The autoencoder cannot fully trust each feature of x independently so it must learn the correlations of x's features.

❖ Based on those relations, we can predict a more 'not prune to changes' model.

➢ We are forcing the hidden layer to learn a generalized structure of the data.



DENOISING AUTOENCODERS - PROCESS

Taken some input x

Apply Noise

$\tilde{x}$

DENOISING AUTOENCODERS - PROCESS

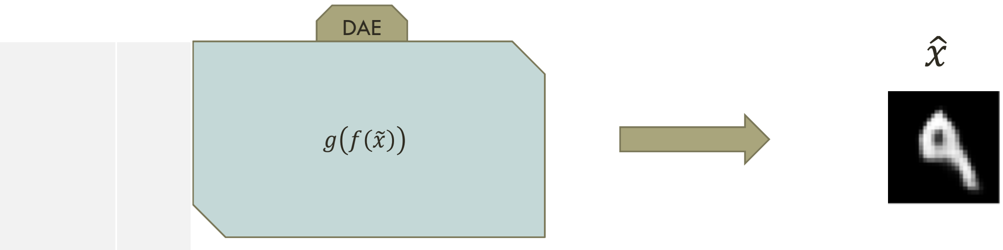
$\tilde{x}$

Encode And Decode

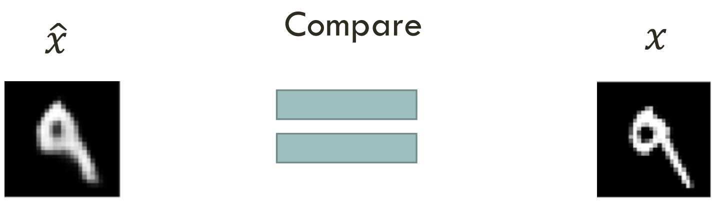
DAE

$g(f(\tilde{x}))$

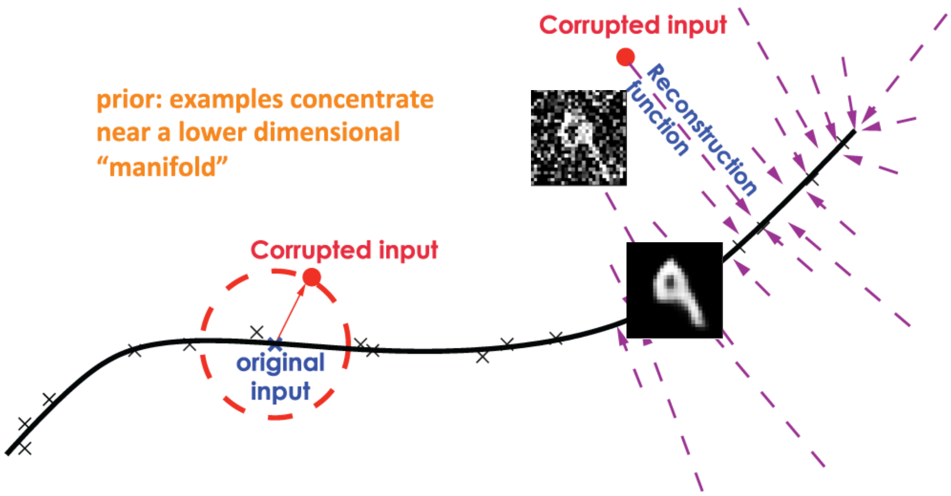
DENOISING AUTOENCODERS - PROCESS



DENOISING AUTOENCODERS - PROCESS

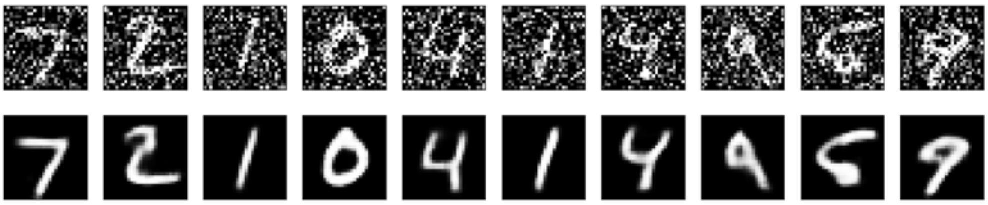


DENOISING AUTOENCODERS



DENOISING CONVOLUTIONAL AE

- 50 epochs.
- Noise factor 0.5
- 92% accuracy on validation set.



STACKED AE

- Motivation:

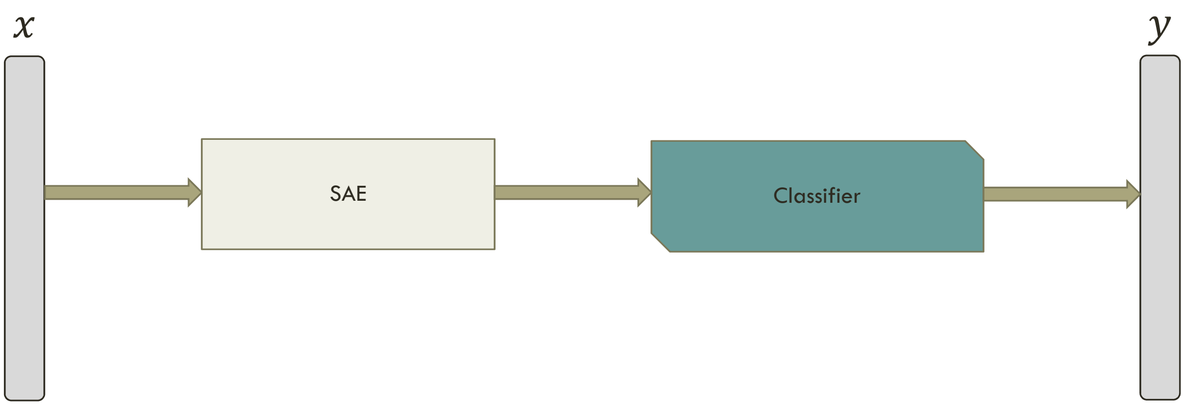
- We want to harness the feature extraction quality of a AE for more enhancement of performance.
- For example: we can build a deep supervised classifier where it's input is the output of a SAE.
- The benefit: our deep model's SAE part of W are not randomly initialized but are rather "smartly pre-selected"
- Also, using this semi-supervised technique, lets us have a larger unlabeled dataset trained.

STACKED AE

- Building a SAE consists of two phases:

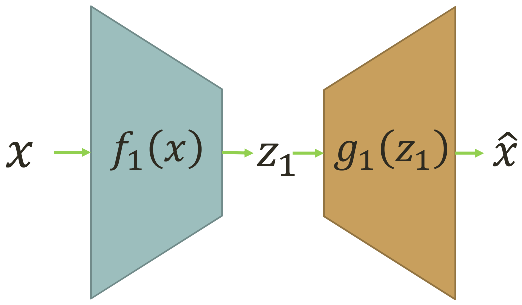
- 1. Train each AE layer one after the other.
- 2. Connect any classifier (SVM / FC NN layer etc.)

STACKED AE



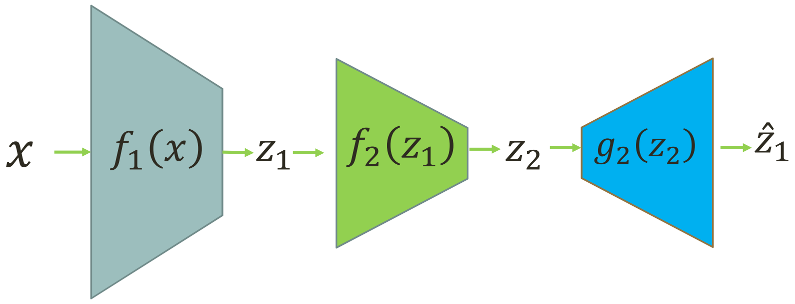
STACKED AE – TRAIN PROCESS

First Layer Training (AE 1)



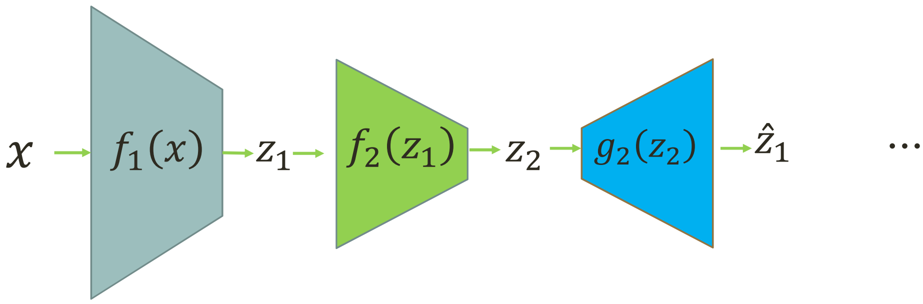
STACKED AE – TRAIN PROCESS

Second Layer Training (AE 2)



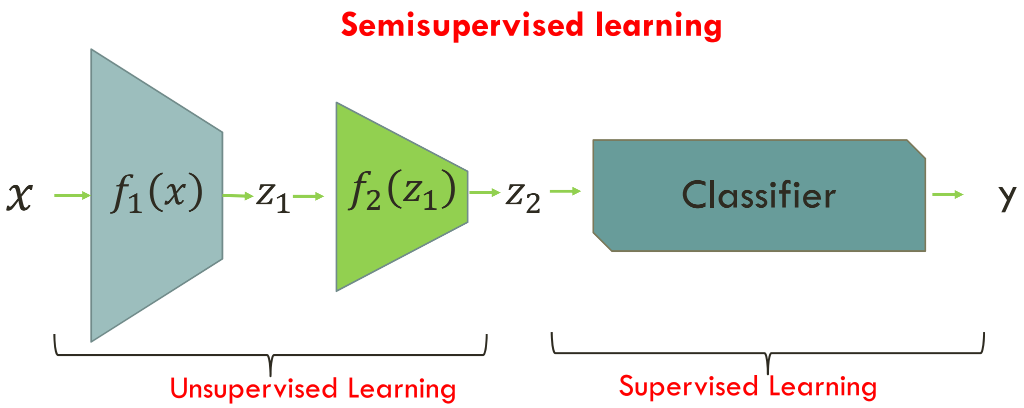
STACKED AE – TRAIN PROCESS

Second Layer Training (AE …)



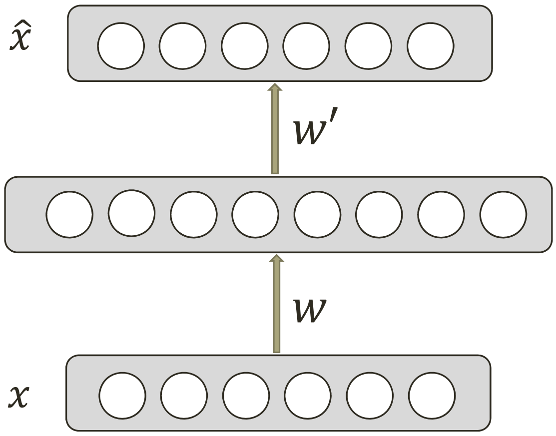
STACKED AE – TRAIN PROCESS

Add any classifier and train it



CONTRACTIVE AUTOENCODERS

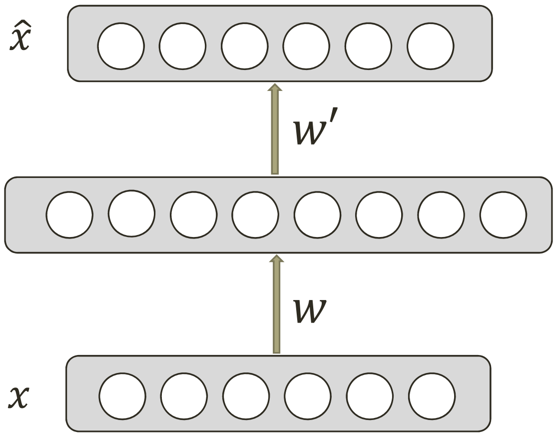
- We are still trying to avoid uninteresting features.
- Here we add a regularization term $\Omega(x)$ to our loss function to limit the hidden layer.



CONTRACTIVE AUTOENCODERS

- Idea: We wish to extract features that **only** reflect variations observed in the training set. We would like to be invariant to the other variations.

- Points close to each other in the input space maintain that property in the latent space.



CONTRACTIVE AUTOENCODERS

Definitions and reminders:

- Frobenius norm (L2): $\|J_f(x)\|_F = \sqrt{\Sigma_{i,j} {w_{ij}}^2}$

Gradient of Hidden layer Vector f at Input Vector x

- Jacobian Matrix: $J_f(x) = \frac{\partial f(x)}{\partial x} = \boxed{\begin{bmatrix} \frac{\partial f(x)_1}{\partial x_1} & \cdots & \frac{\partial f(x)_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(x)_m}{\partial x_1} & \cdots & \frac{\partial f(x)m}{\partial x_n} \end{bmatrix}} = [w_{ij}]$

assuming activation function is identify function.

CONTRACTIVE AUTOENCODERS

Our new loss function would be:

$$L^*(x) = L(x) + \lambda \Omega(x)$$

where $\Omega(x) = \left\| J_f(x) \right\|_F^2$ or simply: $\sum_{i,j} \left(\frac{\partial f(x)_j}{\partial x_i} \right)^2$

and where λ controls the balance of our reconstruction objective and the hidden layer “flatness”.

CONTRACTIVE AUTOENCODERS

Our new loss function would be:

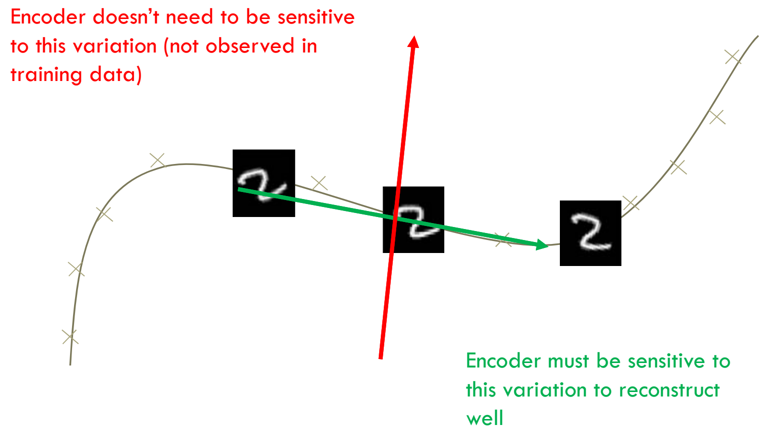
$$L^*(x) = L(x) + \lambda \Omega(x)$$

$L(x)$ - would be an encoder that keeps as much good information as possible ($\lambda \rightarrow 0$)

$\Omega(x)$ - would be an encoder that throws away all information ($\lambda \rightarrow \infty$)

Combination would be an encoder that keeps **only** good information.

CONTRACTIVE AUTOENCODERS



WHICH AUTOENCODER?

- DAE make the **reconstruction function** resist small, finite sized perturbations in input.
- CAE make the **feature encoding function** resist small, infinitesimal perturbations in input.

- Both denoising AE and contractive AE perform well!