CS307: Modeling and Learning in Data Science

Bo Li

University of Illinois at Urbana-Champaign

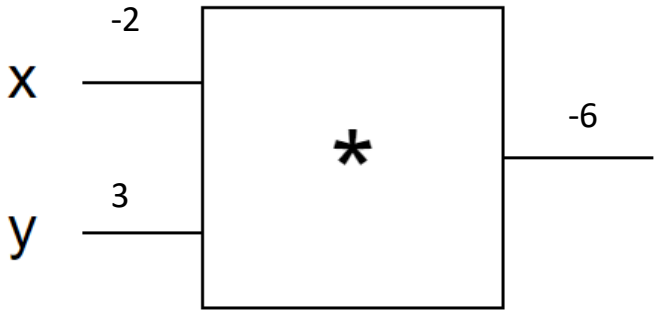
Goal of today's class

- Recall basic algebra and statistics, basic CNN design principles
- Understand the motivation and categorization of dimension reduction/mapping
- Representative Mapping algorithms (chapter. 19)
  - Sammon mapping, T-SNE
  - Autoencoders/denoising autoencoders

Recall: Deep Learning Mini Crash

- Neural Networks Background
- Convolutional Neural Networks (CNNs)

Real-Valued Circuits



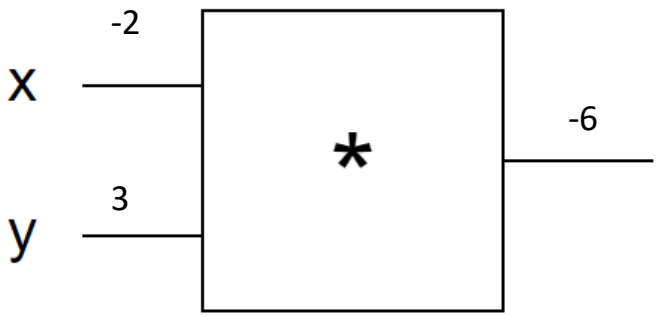
$$f(x, y) = xy$$

Goal: How do I increase the output of the circuit?

- Tweak the inputs. But how?
- Option 1. Random Search?

x = x + step_size * random_value
y = y + step_size * random_value

Real-Valued Circuits



$$f(x, y) = xy$$

Goal: How do I increase the output of the circuit?

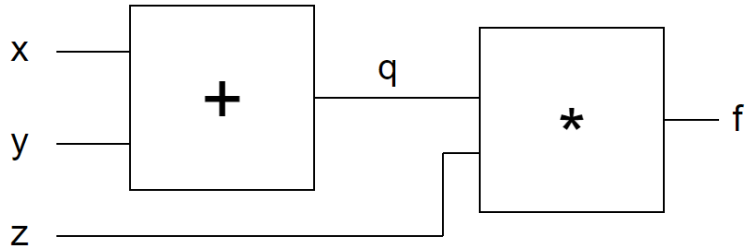
- Option 2. Analytic Gradient

$$\frac{\partial f(x,y)}{\partial x} = \frac{f(x+h,y) - f(x,y)}{h}$$

Limit as h -> 0

x = x + step_size * x_gradient
y = y + step_size * y_gradient

Composable Real-Valued Circuits



$$f(x, y, z) = (x + y)z$$

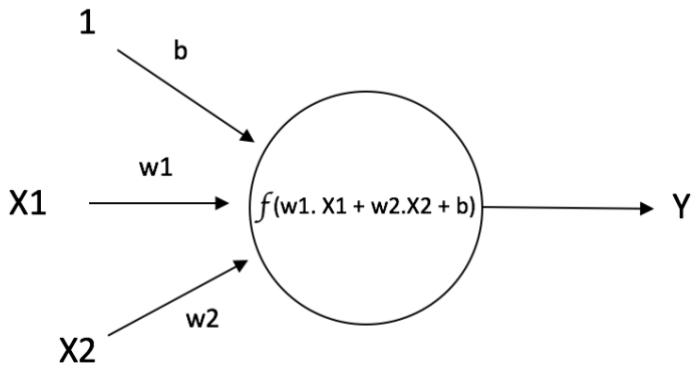
$$f(q, z) = qz \quad \Longrightarrow \quad \frac{\partial f(q,z)}{\partial q} = z, \qquad \frac{\partial f(q,z)}{\partial z} = q$$

$$q(x, y) = x + y \quad \Longrightarrow \quad \frac{\partial q(x,y)}{\partial x} = 1, \qquad \frac{\partial q(x,y)}{\partial y} = 1$$

Chain Rule $$\frac{\partial f(q,z)}{\partial x} = \frac{\partial q(x,y)}{\partial x}\frac{\partial f(q,z)}{\partial q}$$

Backpropagation!

Single Neuron



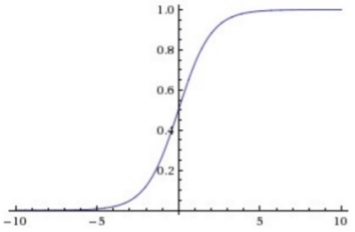
(Input 1)

(Input 2)

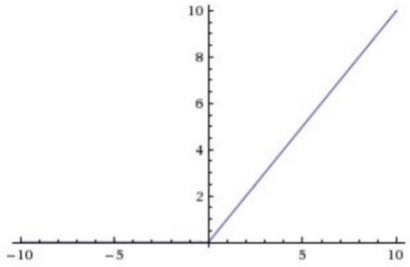
(Output)

Output of neuron = Y= f(w1. X1 + w2.X2 + b)

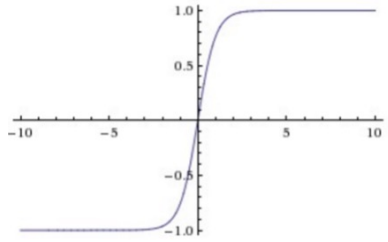
Activation function



Sigmoid

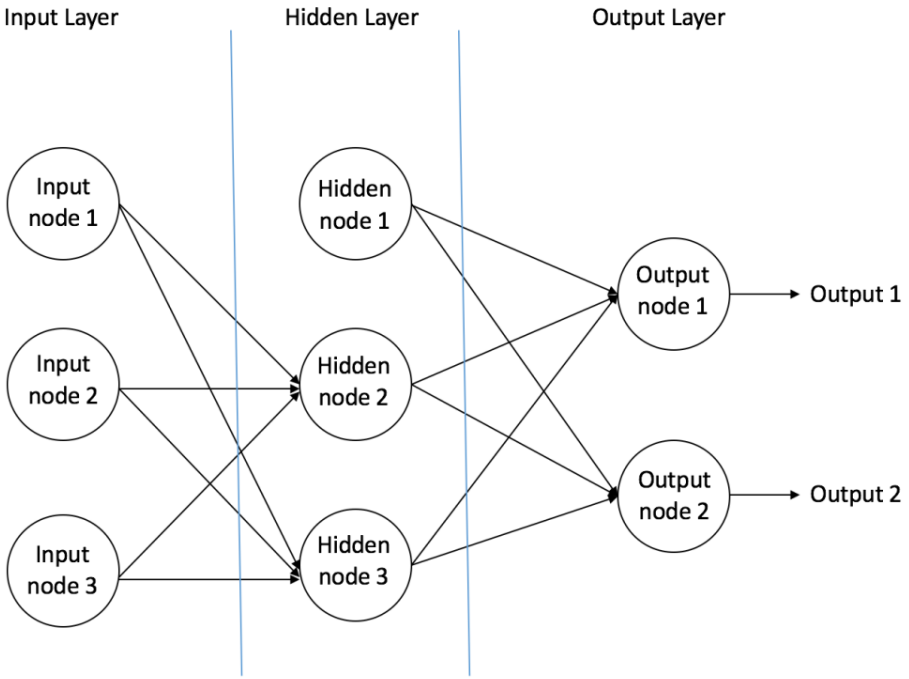


ReLU



tanh

(Deep) Neural Networks!



Organize neurons into a structure

Train (Optimize) using backpropagation

Convolutional Neural Networks (CNNs)

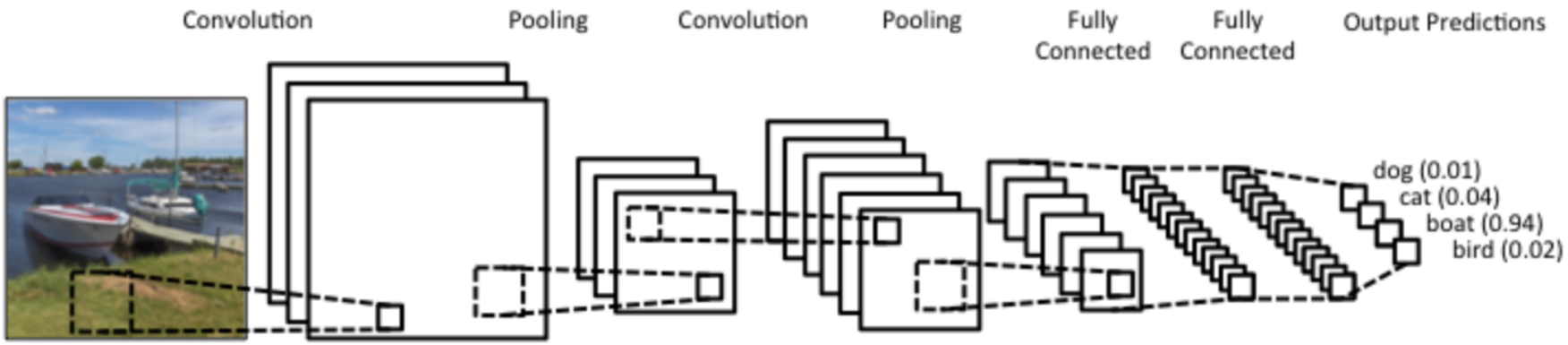
Very widely used, and very useful

a plate with a sandwich and a salad

a group of motorcycles parked in front of a building

a man riding a wave on top of a surfboard

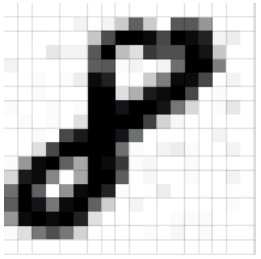
Convolutional Neural Networks (CNNs)



A CNN generally consists of 4 types of architectural units

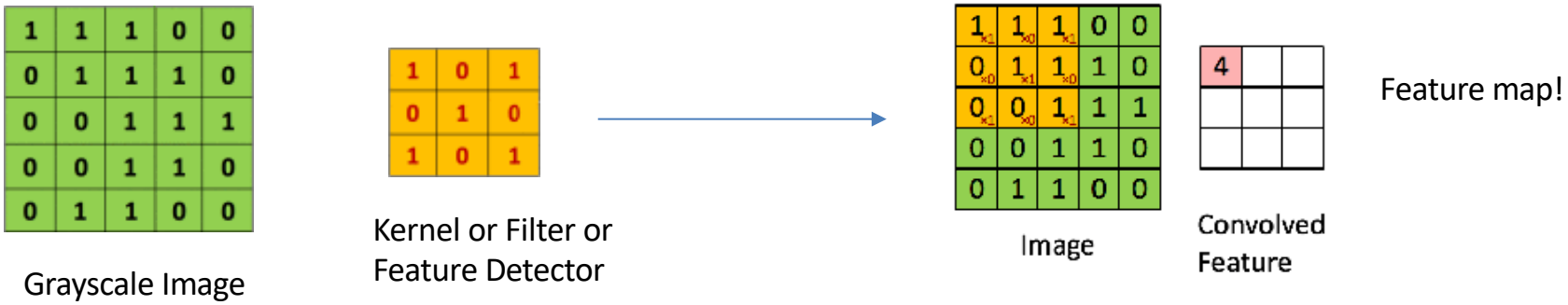
Convolution
Non Linearity (RELU)
Pooling or Subsampling
Classification (Fully Connected Layers)

How is an image represented for NNs?



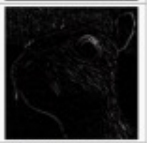
- Matrix of numbers, where each number represents pixel intensity
- If image is colored, then there are three channels per pixel, each channel representing (R, G, B) values

Convolution Operator



- Slide the kernel over the input matrix
- Compute element-wise multiplication, add results to get a single value
- Output is a feature map

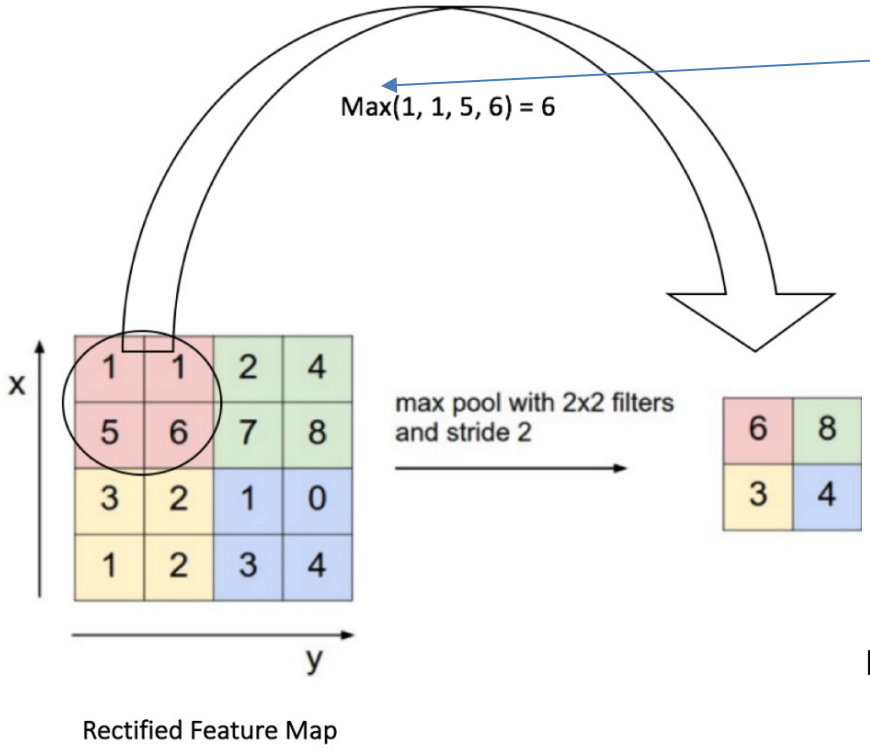
Dimension reduction!

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

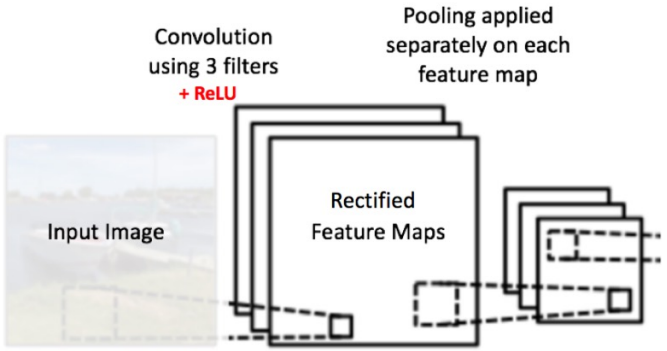
Many types of filters

A CNN learns these filters during training

Pooling

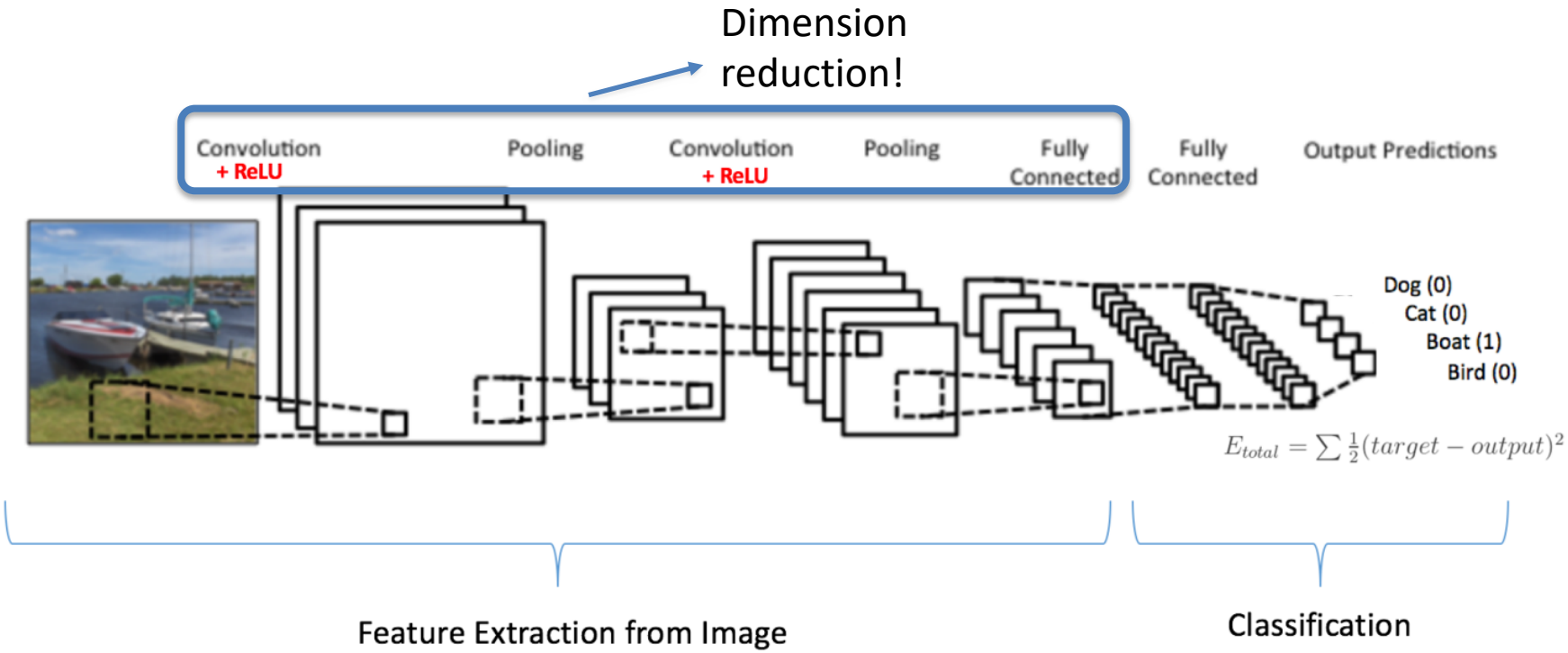


Can be Avg, sum, min, …

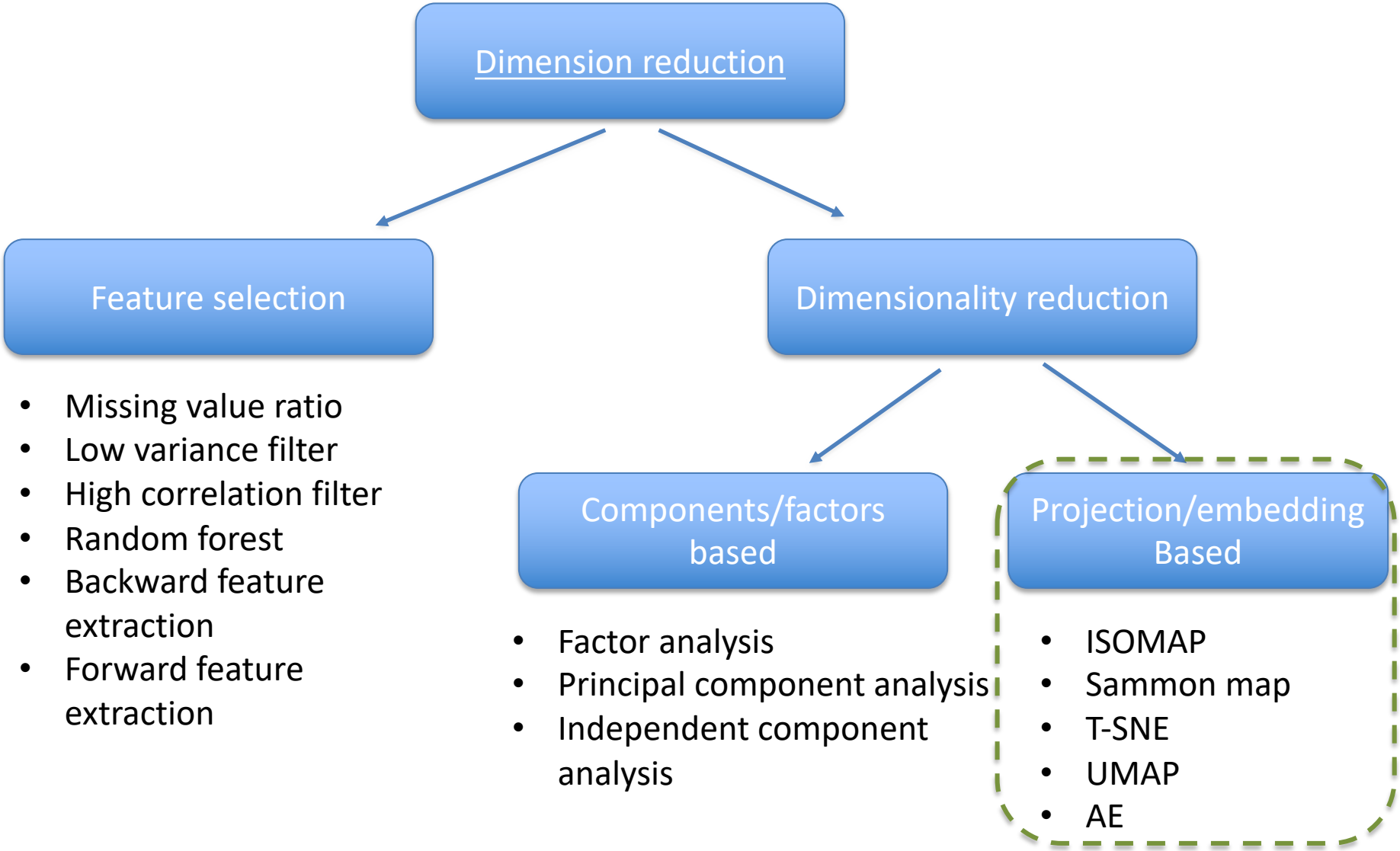


Reduce dimensionality, but retain important features

Putting Everything Together



Activation functions: provide **nonlinear** representation power
Batch norm: **normalization**
Pooling/Dropout: provide **randomness**

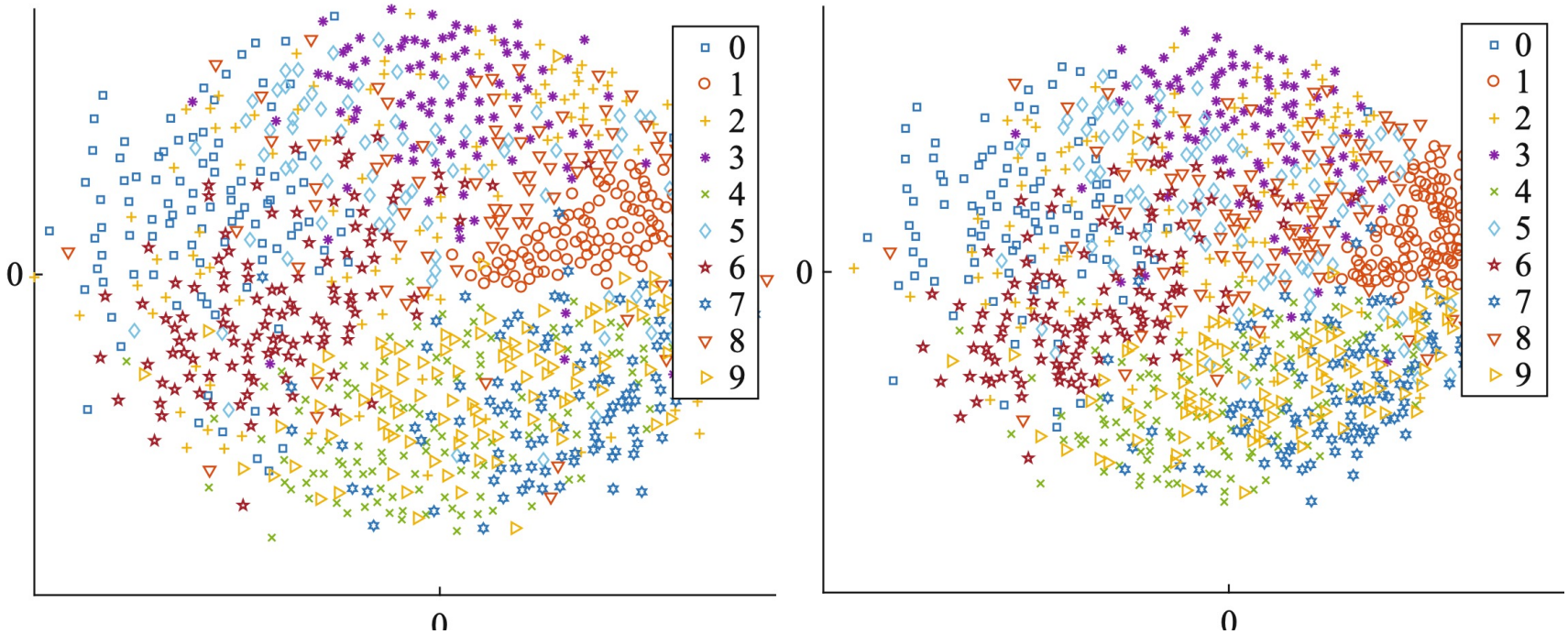

Dimension reduction
Feature selection
Dimensionality reduction
Missing value ratio
Low variance filter
High correlation filter
Random forest
Backward feature extraction
Forward feature extraction
Components/factors based
Factor analysis
Principal component analysis
Independent component analysis
Projection/embedding Based
ISOMAP
Sammon map
T-SNE
UMAP
AE

Sammon mapping

- Principled coordinate analysis: $\sum_{i,j} \left(\mathbf{y}_i^T \mathbf{y}_j - \mathbf{x}_i^T \mathbf{x}_j\right)^2$
  - limitation: the mapping will be almost determined by points that are very far away
- Sammon mapping:

$$C(\mathbf{y}_1, \ldots, \mathbf{y}_N) = \left(\frac{1}{\sum_{i<j} \|\mathbf{x}_i - \mathbf{x}_j\|}\right) \sum_{i<j} \left[\frac{(\|\mathbf{y}_i - \mathbf{y}_j\| - \|\mathbf{x}_i - \mathbf{x}_j\|)^2}{\|\mathbf{x}_i - \mathbf{x}_j\|}\right]$$

  - makes the small distances more significant
  - Limitation: a small distortion would make a big difference



Sammon mappings of 1000 samples of a 784 dimensional MNIST digits.

- On the **left**, the mapping used the whole digit vector, and on the **right**, the data was reduced to 30 dimensions using PCA, then subjected to a Sammon mapping
- The class labels were not used in training, but the plot shows class labels.
- As the legend on the side shows, the classes are moderately well separated. Reducing dimension does not appear to make much difference

Takeaways-1

- Sammon mapping produces an embedding of high dimensional data into a lower-dimensional space that reduces the emphasis that principal coordinate analysis places on large distances.
- It does so by solving an optimization problem to choose coordinates in a low dimensional space for each data point.
- Sammon mappings are often biased by very small distances, however.

T-SNE

- Goal: build a mapping model by reasoning about *probability* rather than only *distance*
- The probability that two points in the high dimensional space are neighbors: $p_{j|i} = \frac{w_{j|i}}{\sum_k w_{k|i}}$

$$w_{j|i} = \exp\left(\frac{\|\mathbf{x}_j - \mathbf{x}_i\|^2}{2\sigma_i^2}\right)$$

- The probability that two points in the *low dimensional* space are neighbors:

$$q_{ij}(\mathbf{y}_1, \ldots, \mathbf{y}_N) = \frac{\frac{1}{1+\|\mathbf{y}_i - \mathbf{y}_j\|^2}}{\sum_{k,l,k\neq l} \frac{1}{1+\|\mathbf{y}_l - \mathbf{y}_k\|^2}}$$

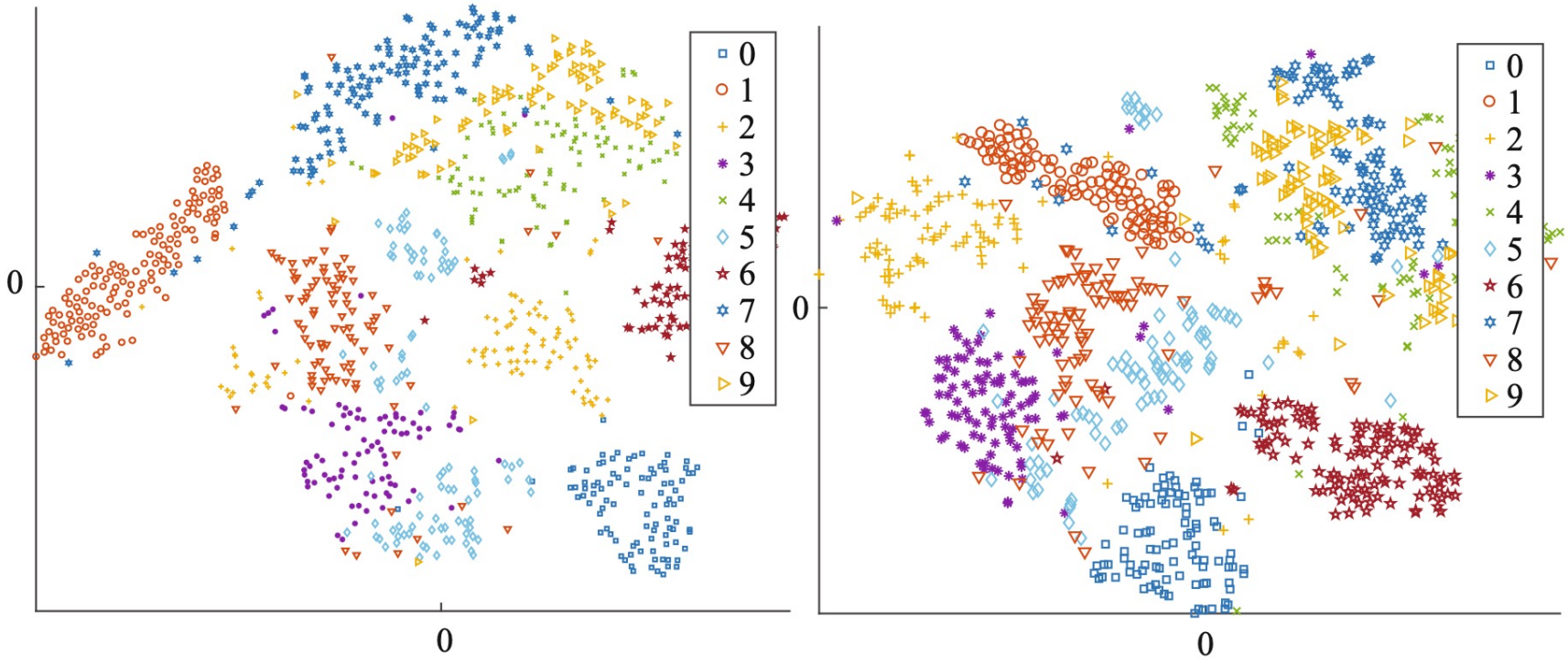
T-SNE

- Mapping from the high dimensional to low dimensional space:

$$C_{tsne}(\mathbf{y}_1, \ldots, \mathbf{y}_N) = \sum_{ij} p_{ij} \log \frac{p_{ij}}{q_{ij}(\mathbf{y}_1, \ldots, \mathbf{y}_N)}$$

- The gradient is of a simple form:

$$\nabla_{\mathbf{y}_i} C_{tsne} = 4 \sum_j \left[(p_{ij} - q_{ij}) \frac{(\mathbf{y}_i - \mathbf{y}_j)}{1 + \| \mathbf{y}_i - \mathbf{y}_j \|^2} \right]$$



A T-SNE mapping of 1000 samples of a 784 dimensional dataset.

- On the **left**, the data was reduced to 30 dimensions using PCA, then subjected to a T- SNE mapping. On the **right**, the data was reduced to 200 dimensions using PCA, then mapped.
- The class labels were not used in training, but the plot shows class labels.
- As the legend on the side shows, T-SNE separates the classes much more effectively than Sammon mapping

Takeaway-2

- T-SNE produces an embedding of high dimensional data into a lower-dimensional space.
- It does so by solving an optimization problem to choose coordinates in a low dimensional space for each data point.
- The optimization problem tries to make the *probability* a pair of points are neighbors in the low dimensional space similar to that probability in the high dimensional space.
- T-SNE appears less inclined to distort datasets than either principal coordinate analysis or Sammon mapping.

Autoencoders

- Limitations of Sammon mapping and T-SNE:
  - 1) we cannot construct y corresponding to a new x (cannot map a given low-dimensional instance to a high-dimensional one);
  - 2) we cannot tell if a representation is good or not
- Solutions?
  - Train another network to map from the low dimension to high dimension again!

Autoencoders

- An autoencoder is a neural network trained to copy its input to its output
- Network has encoder and decoder functions
- Autoencoders should not copy perfectly
  - But restricted by design to copy only approximately
  - By doing so, it learns useful properties of the data
  - Modern autoencoders use stochastic mappings
  - Autoencoders were traditionally used for
    - Dimensionality reduction as well as feature learning

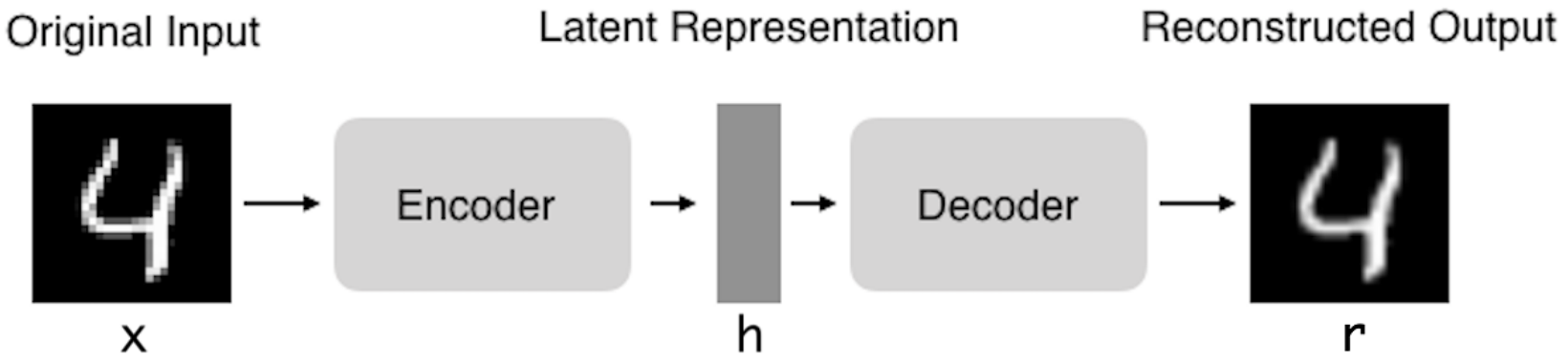
Autoencoders

- Supervised learning uses explicit labels/correct output in order to train a network.
  - E.g., classification of images.

- Unsupervised learning relies on data only.
  - Key point is to produce a useful embedding.
  - The embedding encodes structure such as word similarity and some relationships.
  - Still need to define a loss – this is an implicit supervision.

Autoencoders: Structure

- Encoder: compress input into a latent-space of usually smaller dimension. h = f(x)
- Decoder: reconstruct input from the latent space. r = g(f(x)) with r as close to x as possible



Blog: https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f

Autoencoders

- Compare PCA/SVD
  - PCA takes a collection of vectors (images) and produces a usually smaller set of vectors that can be used to approximate the input vectors via linear combination.
  - Very efficient for certain applications.
  - Fourier and wavelet compression is similar.

- Neural network autoencoders
  - Can learn nonlinear dependencies
  - Can use convolutional layers
  - Can use transfer learning

Who is more powerful?

- An autoencoder with *linear* decoder and MSE loss function learns the **same** subspace as PCA
- Nonlinear encoder/decoder functions yield more powerful *nonlinear* generalizations of PCA

Regularized autoencoders

- Use a loss model that encourages properties other than copying the input to the output
  - Sparsity of representation
    - L1 or KL regularization
  - Smallness of the derivative of the representation
    - Contractive autoencoder
  - Robustness to noise or missing inputs
    - Denoising autoencoder

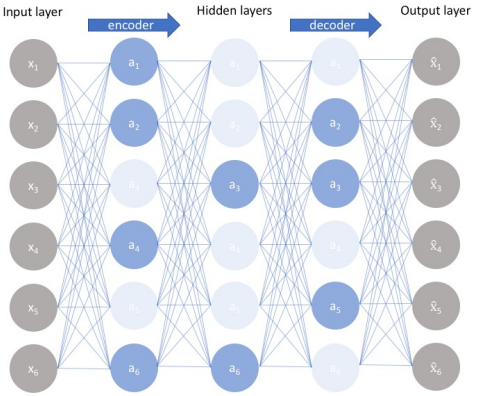
Sparse autoencoders

- Construct a loss function to penalize *activations* within a layer.
- Usually regularize the *weights* of a network, not the activations.
- Individual nodes of a trained model that activate are *data-dependent.*
  - Different inputs will result in activations of different nodes through the network.
- Selectively activate regions of the network depending on the input data.

Sparse autoencoders

- Construct a loss function to penalize *activations* the network.
  - **L1 Regularization**: Penalize the absolute value of the vector of activations *a* in layer *h* for observation $\mathcal{L}(x, \hat{x}) + \lambda \sum_i \left| a_i^{(h)} \right|$
  - **KL divergence:** Use cross-entropy between average activation and desired activation

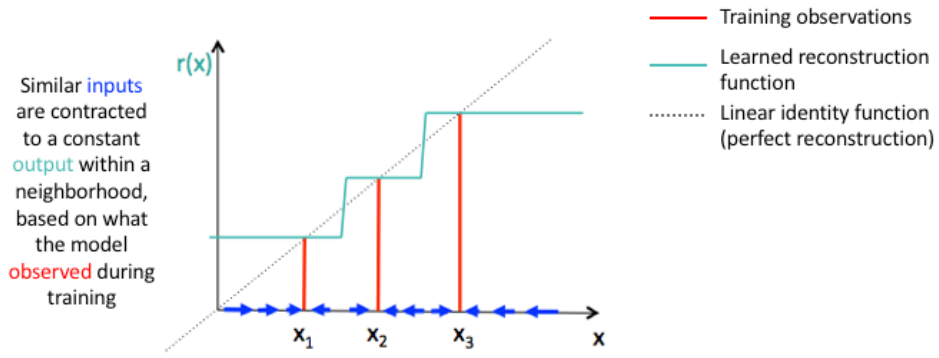
$$\mathcal{L}(x, \hat{x}) + \sum_j KL\left(\rho || \hat{\rho}_j\right)$$



Small derivatives of representation

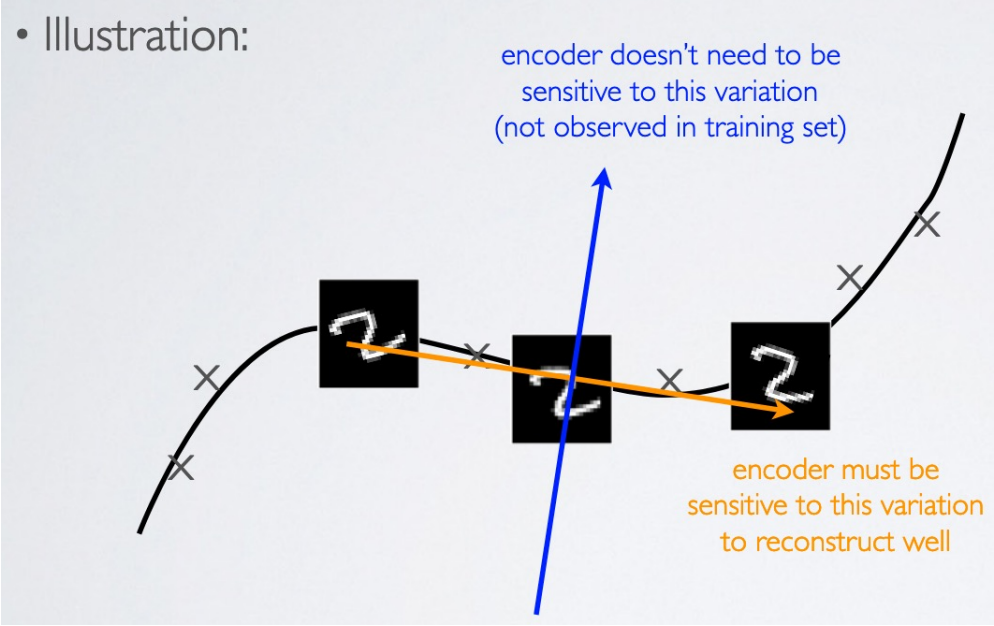
- Arrange for similar inputs to have similar activations.
  - I.e., the *derivative of the hidden layer activations are small* with respect to the input.
- Contractive autoencoders make the *feature extraction function* (ie. encoder) resist infinitesimal perturbations of the input.

$$\mathcal{L}(x, \hat{x}) + \lambda \sum_i \left\| \nabla_x a_i^{(h)}(x) \right\|^2$$



Small derivatives of representation

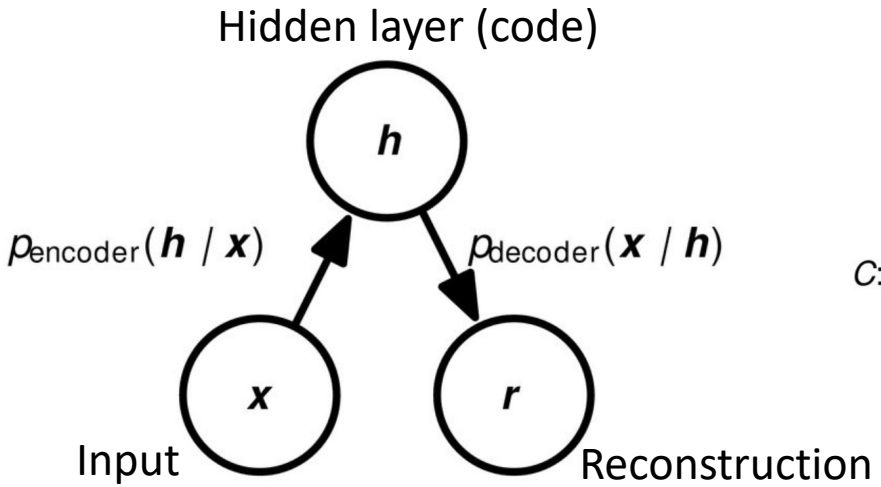
- Contractive autoencoders make the *feature extraction function* (ie. encoder) resist infinitesimal perturbations of the input.



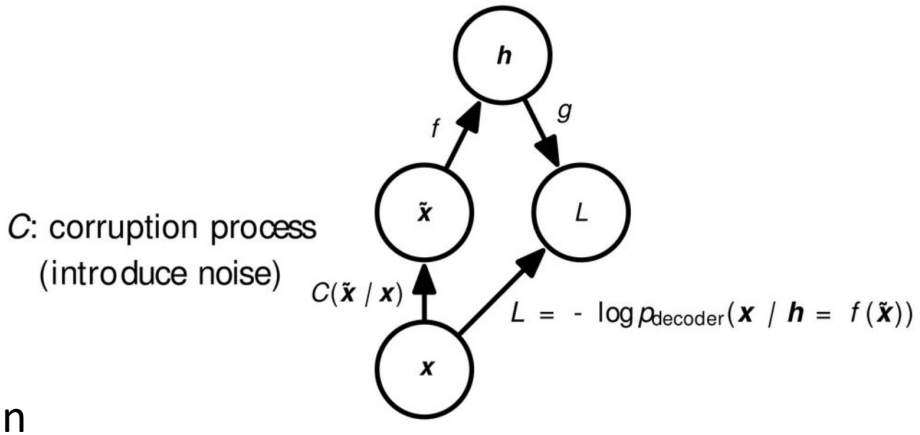
Denoising autoencoder

- A denoising autoencoder (DAE) is one that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output
- Learn the reconstructed distribution
  - Choose a training sample from the training data
  - Obtain corrupted version from corruption process
  - Use training sample pair to estimate reconstruction

Denoising autoencoder

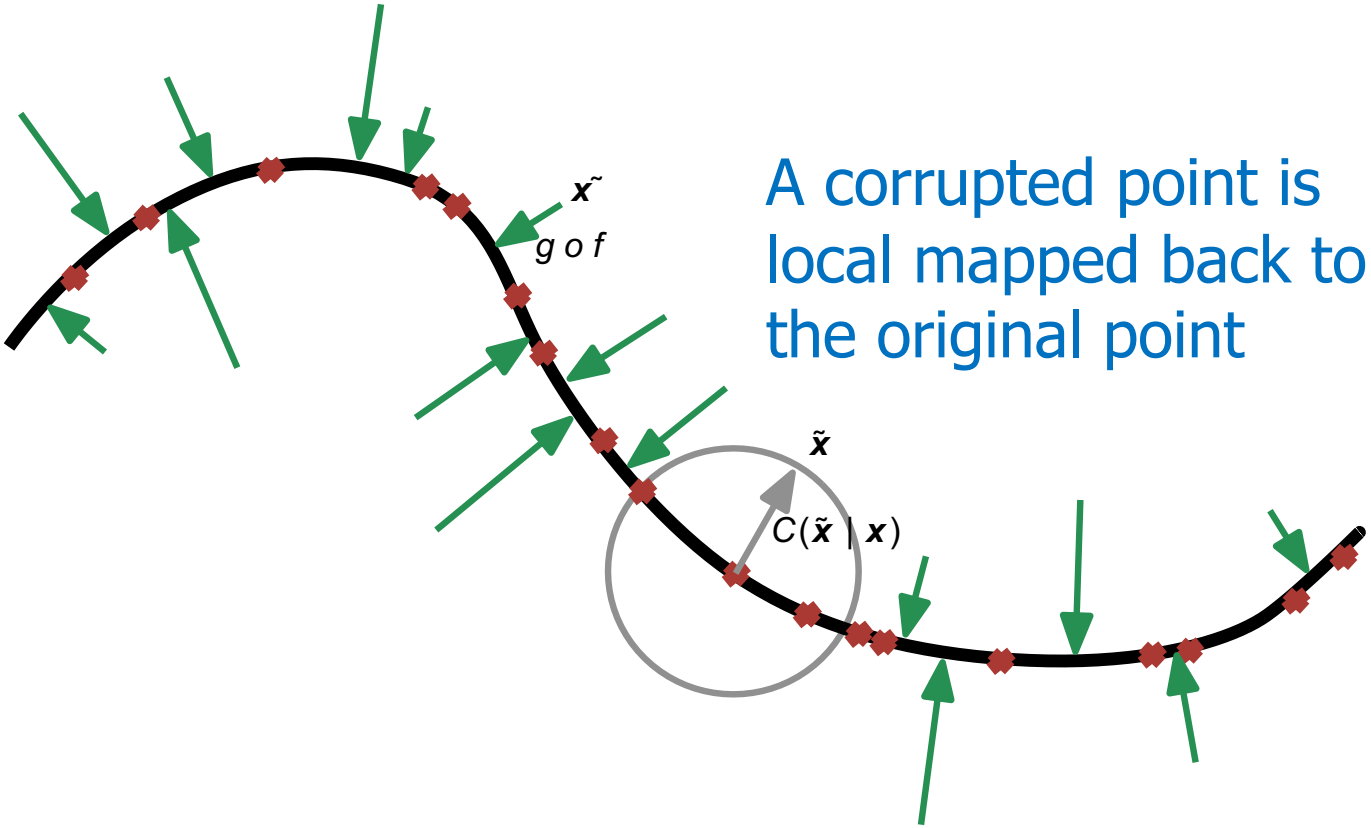


Autoencoder



Denoising Autoencoder

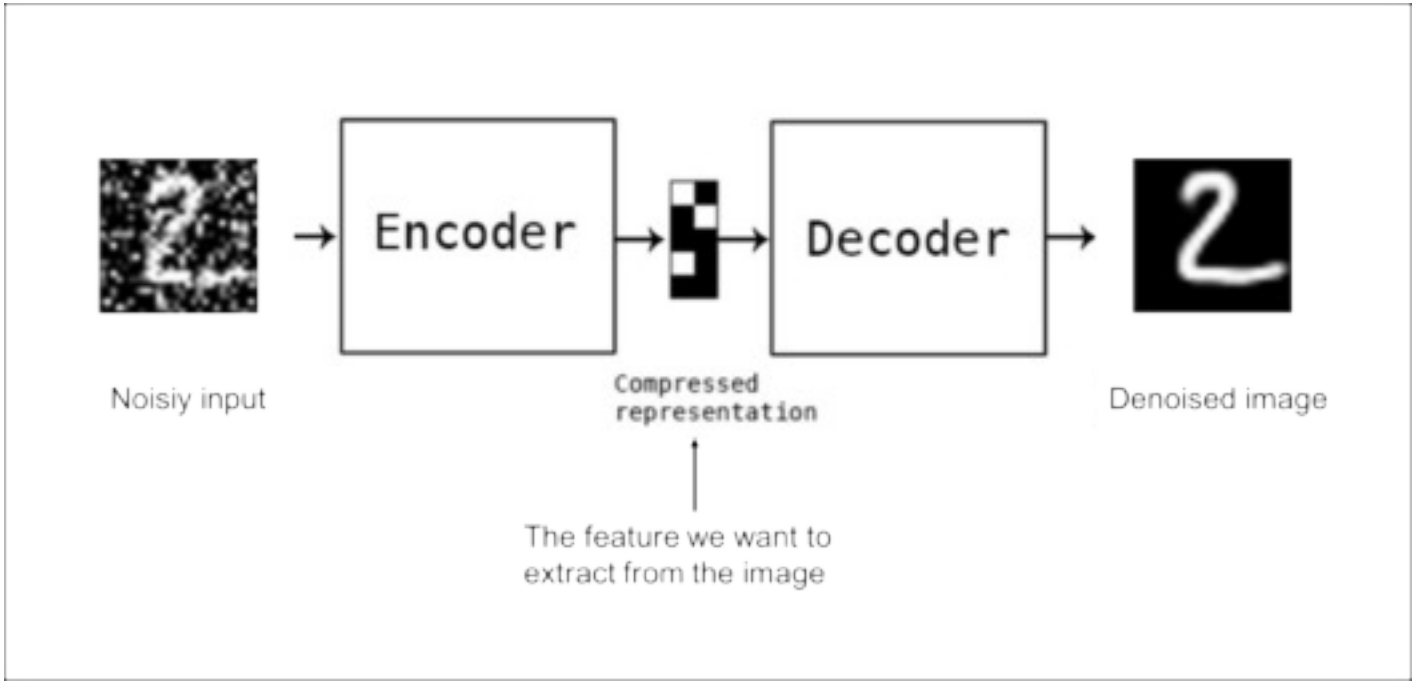
Denoising autoencoders learn a manifold



A corrupted point is local mapped back to the original point

Autoencoders: Applications

- Denoising: input clean image + noise and train to reproduce the clean image.



Autoencoders: Applications

- Image colorization: input black and white and train to produce color images

Autoencoders: Applications

- Watermark removal

Properties of Autoencoders

- **Data-specific**: Autoencoders are only able to compress data similar to what they have been trained on.
- **Lossy:** The decompressed outputs will be degraded compared to the original inputs.
- **Learned automatically from examples:** It is easy to train specialized instances of the algorithm that will perform well on a specific type of input.

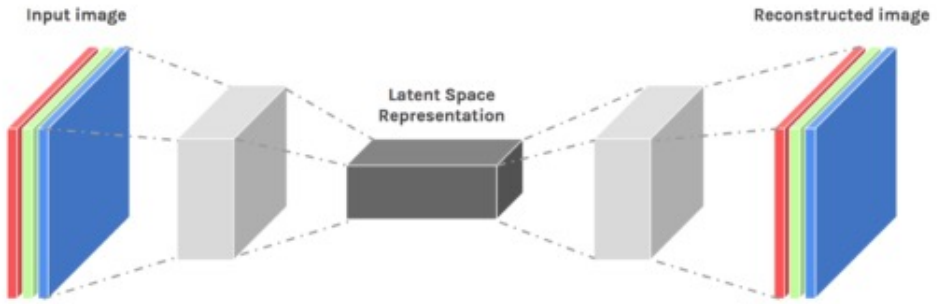
Capacity

- As with other NNs, overfitting is a problem when capacity is too large for the data.

- Autoencoders address this through some combination of:
  - Bottleneck layer – fewer degrees of freedom than in possible outputs.
  - Training to denoise.
  - Sparsity through regularization.
  - Contractive penalty.

Bottleneck layer (undercomplete)

- Suppose input images are n x n and the latent space is m < n x n.
- Then the latent space is not sufficient to reproduce all images.
- Needs to learn an encoding that captures the important features in training data, sufficient for approximate reconstruction.



Simple bottleneck layer in Keras

- input_img = Input(shape=(784,))
- encoding_dim = 32
- encoded = Dense(encoding_dim, activation='relu')(input_img)
- decoded = Dense(784, activation='sigmoid')(encoded)
- autoencoder = Model(input_img, decoded)

- Maps 28x28 images into a 32 dimensional vector.
- Can also use more layers and/or convolutions.

Denoising autoencoders

- Basic autoencoder trains to minimize the loss between x and the reconstruction g(f(x)).
- Denoising autoencoders train to minimize the loss between x and g(f(x+w)), where w is random noise.
- Same possible architectures, different training data.
- Kaggle has a dataset on damaged documents.

Takeaway-3: Autoencoders

- Autoencoders allow the nonlinear representation and learn high quality embeddings
- 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
- Various of applications of AE

Lab this week

- T-SNE plots on MNIST, ex.19.2