Deep Learning Theory and Applications

Unsupervised Learning and Autoencoders

Yale

CPSC/AMTH 663

Outline

1. Unsupervised Learning
2. PCA
3. Autoencoders

Unsupervised learning

- Dataset contains only features and no labels
- Goal is to find hidden patterns in the unlabeled data
- Examples
  - Density estimation
  - Data generation
  - Clustering
  - Data denoising
  - Representation learning
- ***Semi-supervised learning*** attempts to combine information from both labeled and unlabeled data to deduce information

Representation learning

- Find the “best” representation of the data
  - I.e., find a representation of the data that preserves as much information as possible while obeying some penalty or constraint that simplifies the representation
- How do we define simple?
  - Low-dimensional
  - Sparse
  - Independent components
- Above criteria aren’t necessarily mutually exclusive
  - E.g., low-dimensional representations often have independent or weakly dependent components

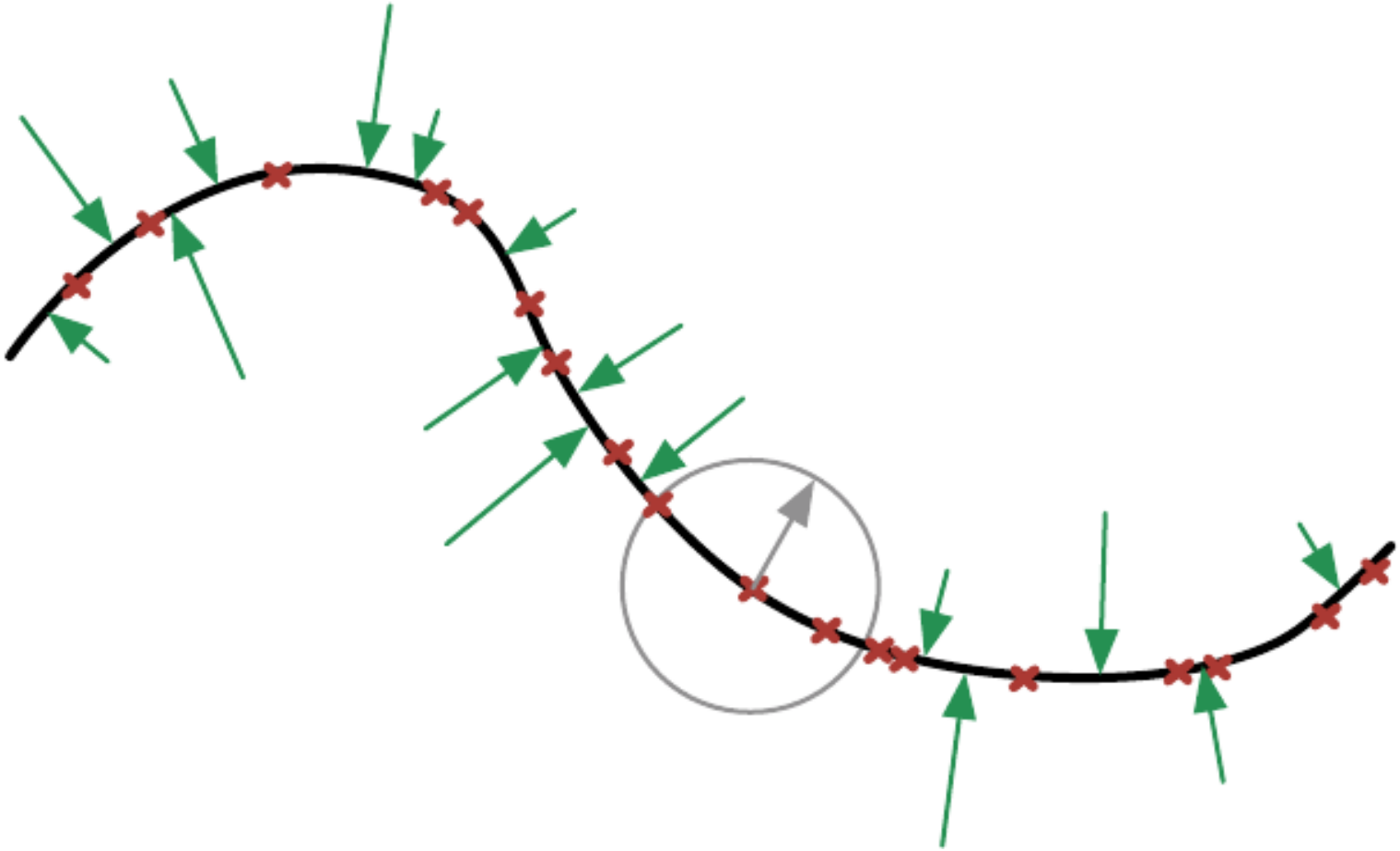
Representation learning

Why is a simple representation useful?

- Interpretability
  - E.g. visualization
- Computational cost
  - Compression
- Performance
  - Preprocessing

Manifold assumption

- Data may be modeled as lying on a low-dimensional manifold



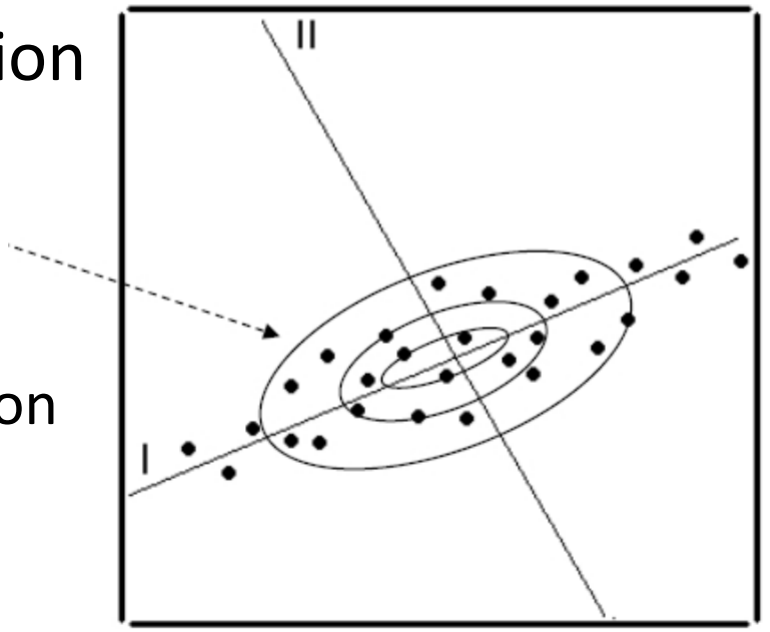
Example

- What is a good lower dimensional representation of this data?

- How can we find it?

Principal Component Analysis (PCA)

- PCA: the most popular method for dimensionality reduction
- Goal: find the directions in input space that explain the most variation
  - Data is re-represented by projecting along those directions
- Assumptions
  - The variation contains the information (not necessarily true in high noise settings)
  - Data is continuous
  - The manifold model is a linear subspace
  - Components are linearly uncorrelated with each other



PCA: Maximize Variance

- N data vectors $\boldsymbol{x}_i$ with dimension d
  - $\overline{\boldsymbol{x}}$ the sample mean
- Goal is to reduce dimensionality to $k \ll d$
- Sample covariance matrix:

$$C = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T$$

- Find direction $\boldsymbol{u}$ that maximizes the variance:

$$\arg\max_{\boldsymbol{u}} \boldsymbol{u}^T C \boldsymbol{u}$$

  - Typically, we require $\|\boldsymbol{u}\| \leq 1$
  - This is an eigendecomposition problem!
  - Choose $\boldsymbol{u}$ to be the first eigenvector of C

PCA: Maximize Variance

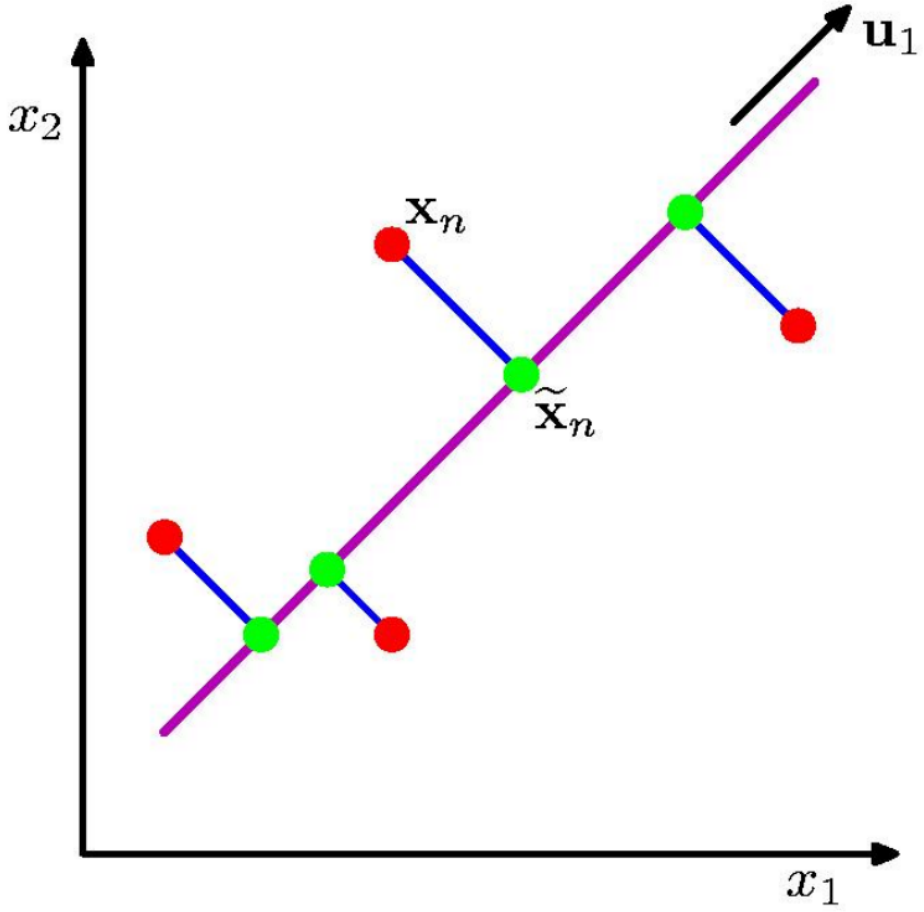
- N data vectors $\boldsymbol{x}_i$ with dimension d
  - $\overline{\boldsymbol{x}}$ the sample mean
- Goal is to reduce dimensionality to $k \ll d$
- Sample covariance matrix:

$$C = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T = \mathrm{U}\Lambda U^T$$

- U_k contains the first k eigenvectors
  - These give the directions of most explained variance
- Project the data points on this space: $\boldsymbol{z}_i = U_k^T \boldsymbol{x}_i$
  - Reconstruct with $\widehat{\boldsymbol{x}}_i = U_k U_k^T \boldsymbol{x}_i$

PCA: Two Views

1. Maximize variance (scatter of green points)
2. Minimize error (red-green distance per data point)



PCA: Minimizing reconstruction error

- PCA projects the data onto a lower-dimensional subspace
- Goal: find the projection s.t. the best linear reconstruction $\hat{X}$ minimizes the error:

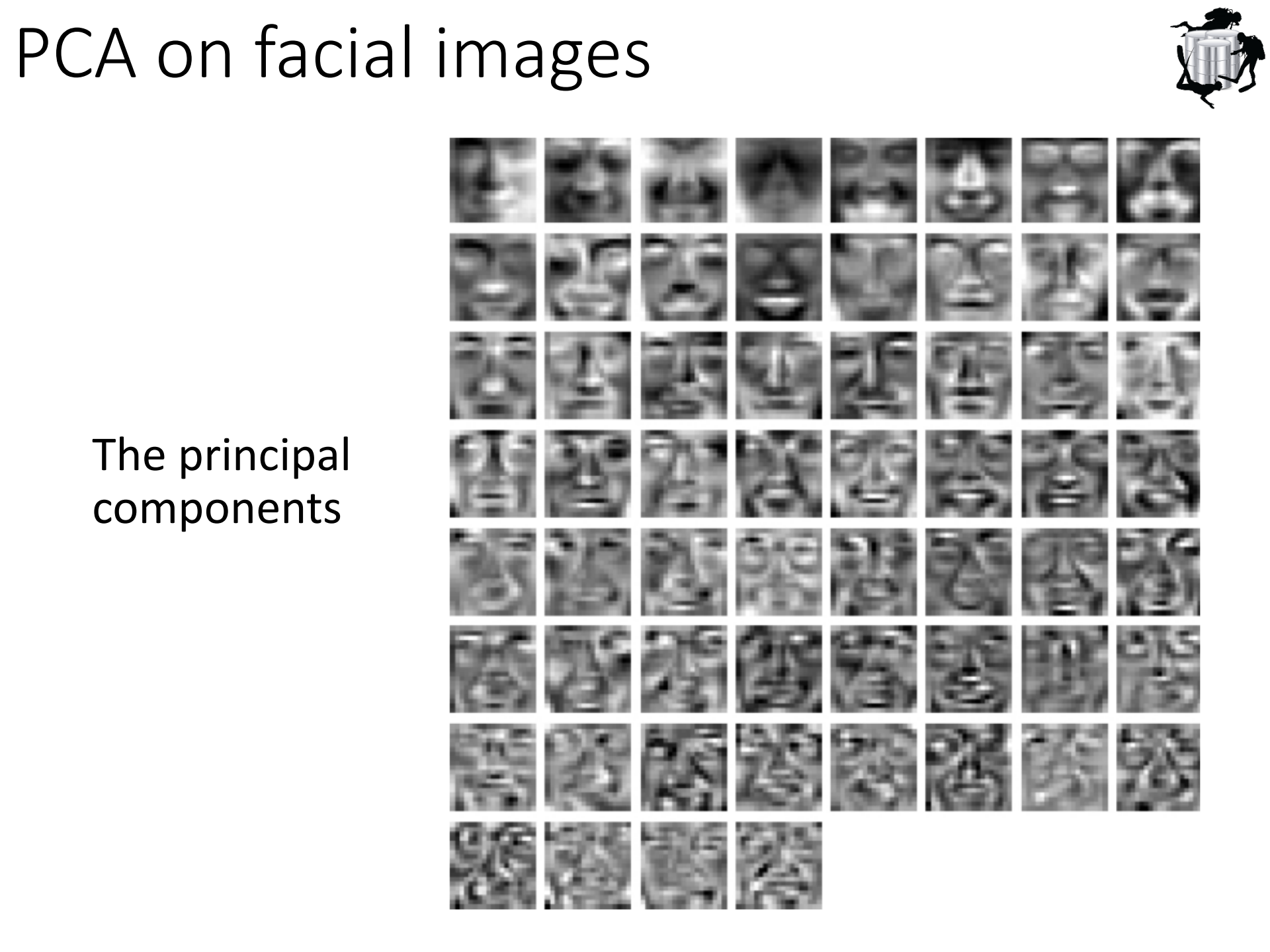
$$\left\| X - \hat{X} \right\|_2^2$$

  - X is a data matrix with data points in columns
- Corresponds to selecting the first k eigenvectors and eigenvalues of XX^T (the sample covariance matrix)

PCA on facial images

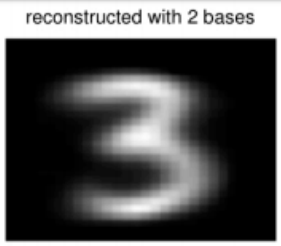
- PCA applied to 2429 19×19 images from CBCL dataset
- Reconstruction with only 3 components:

PCA on facial images

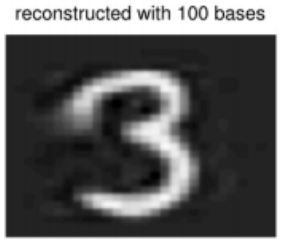


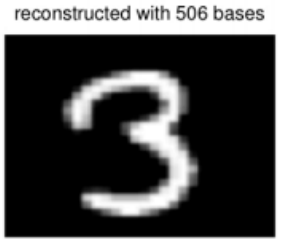
The principal components

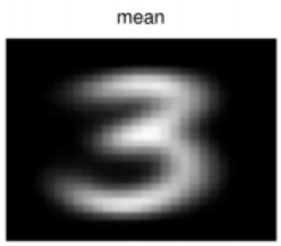
PCA on MNIST

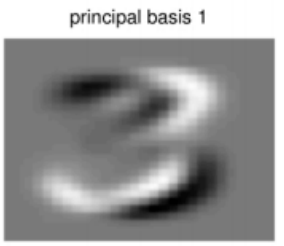

reconstructed with 2 bases

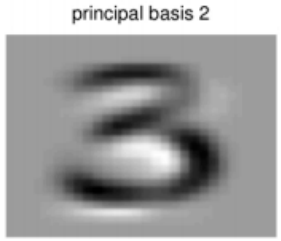

reconstructed with 10 bases

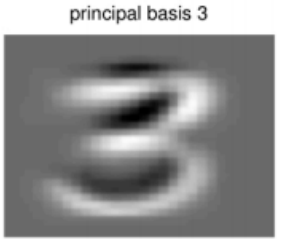

reconstructed with 100 bases


reconstructed with 506 bases


mean


principal basis 1

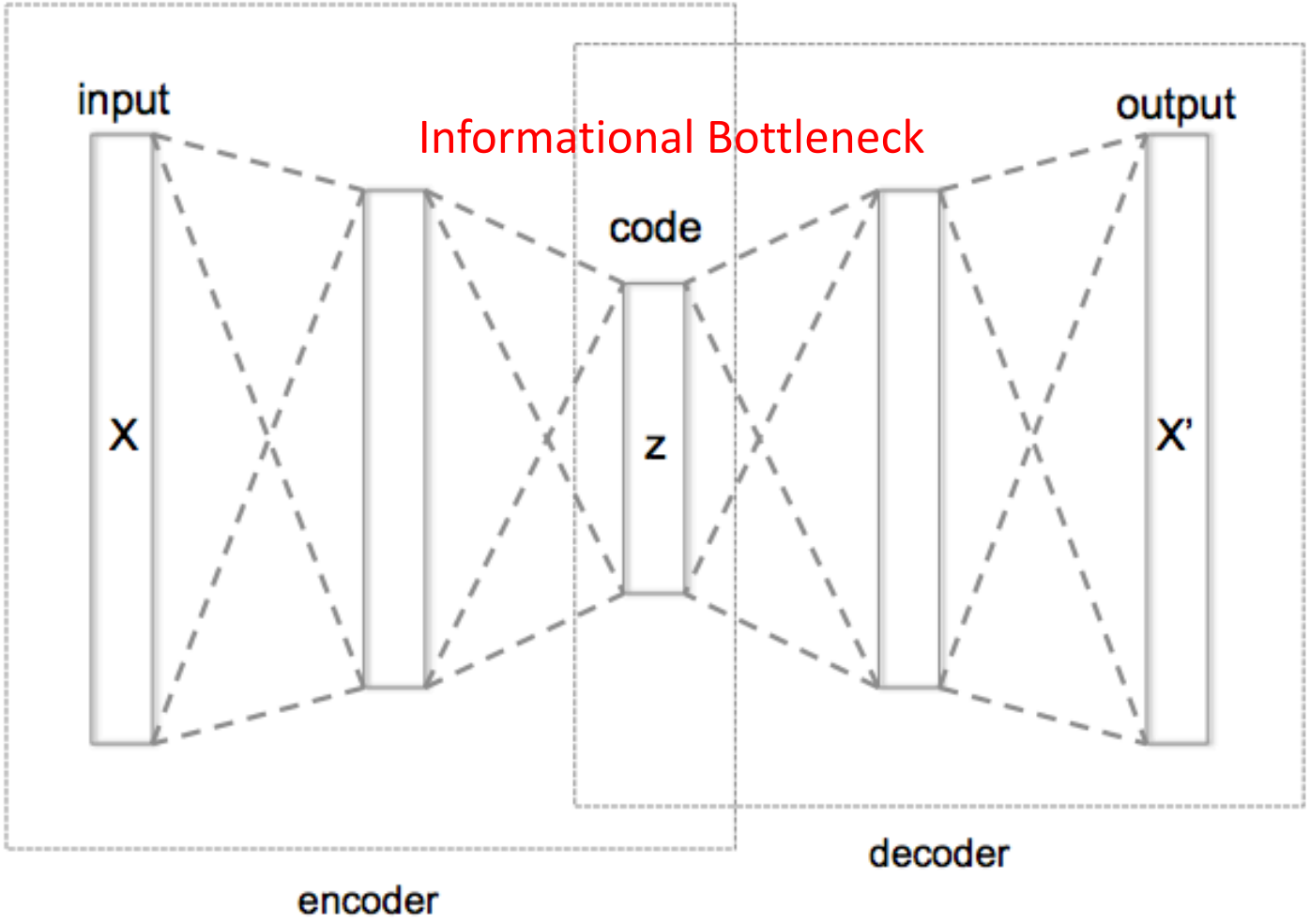

principal basis 2


principal basis 3

Autoencoder (AE)

- Neural network trained to copy its input x to its output
- Hidden layer z describes a ***code*** to represent the input
- Two parts
  - Encoder: $\mathrm{z} = f(x)$
  - Decoder: $x' = g(z)$
- Goal: minimize $L\left(x, g\left(f(x)\right)\right)$
  - L penalizes $g\left(f(x)\right)$ for being dissimilar from x
  - Example: mean squared error
- How do you train an autoencoder?
  - Backpropagation
  - Recirculation (rarely)

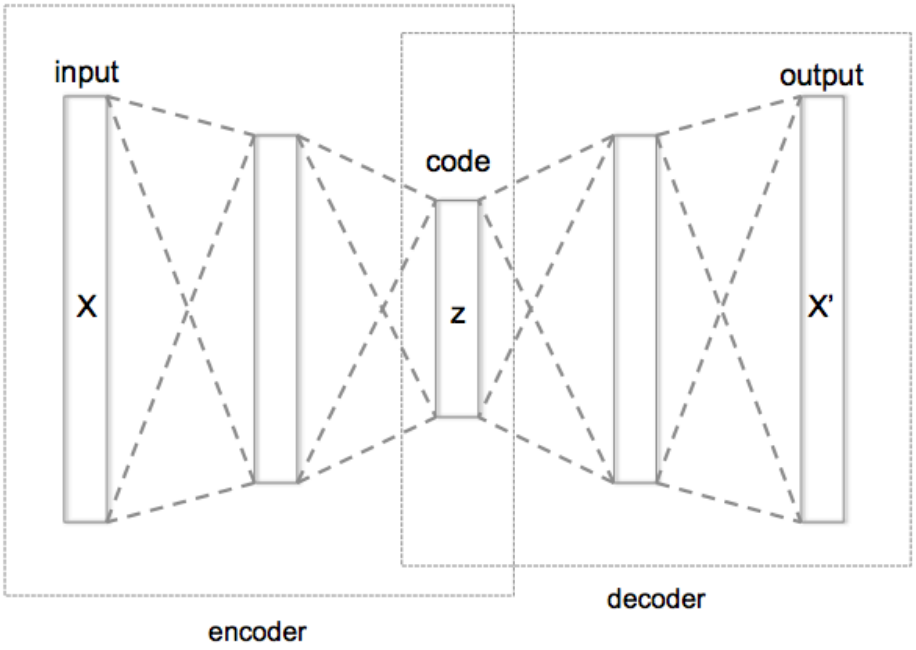
Autoencoder



[Hinton, Salakhutdinov, Science 2006]

Autoencoder Applications

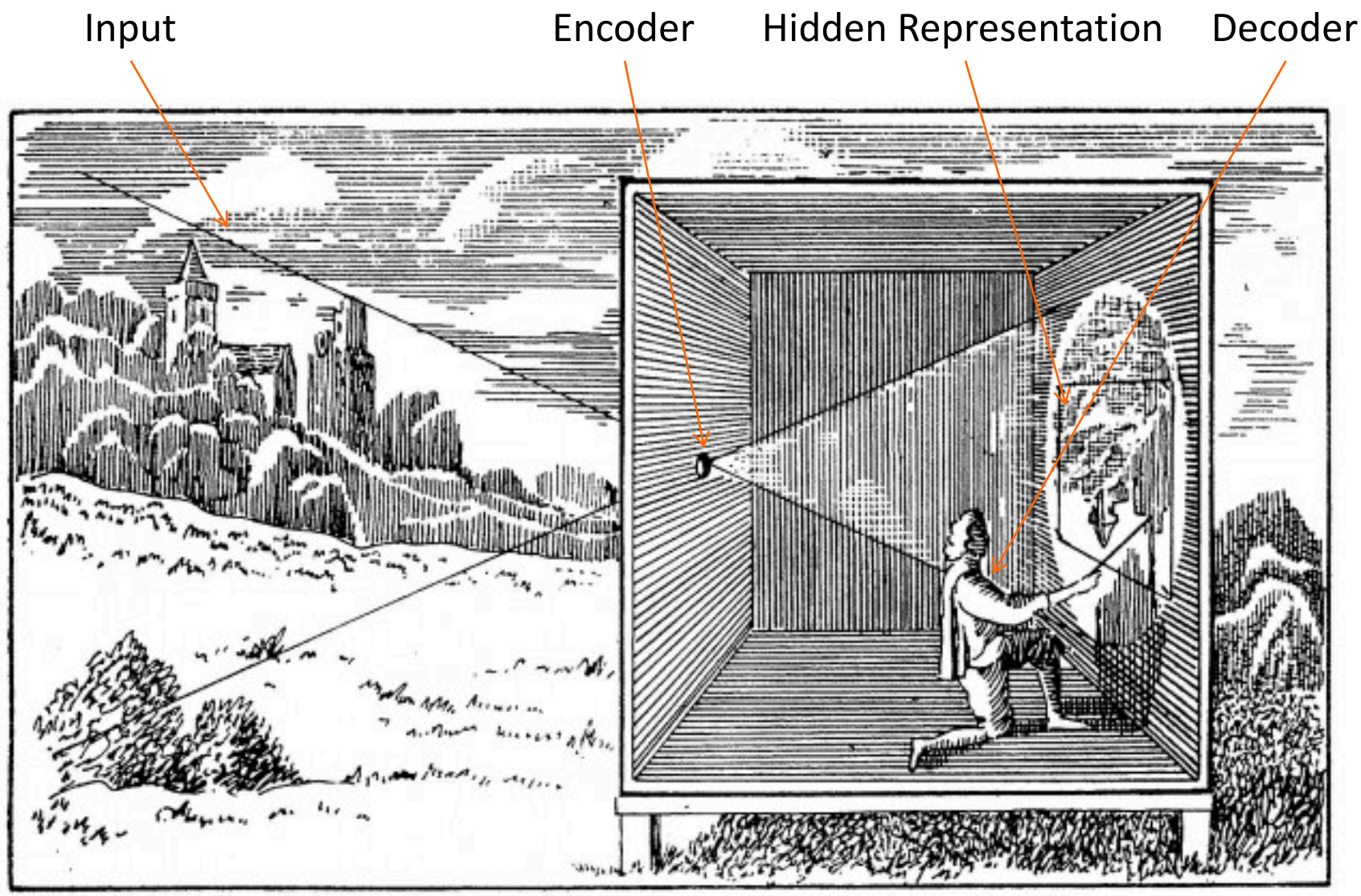
- Dimensionality reduction
  - Including visualization
- Feature learning
- Information retrieval
- Data compression
- Data denoising
- Data generation



Are Autoencoders useful?

- Can't we just set $g\big(f(x)\big) = x$ everywhere?
  - Not very useful
- Design the AE so it can't learn to copy the input perfectly
  - Restrict the AE to copy only approximately
  - Can prioritize certain aspects of the input
- Examples
  - Restrict the size of the code (i.e. add a bottleneck)
  - Add regularization

Camera Obscura



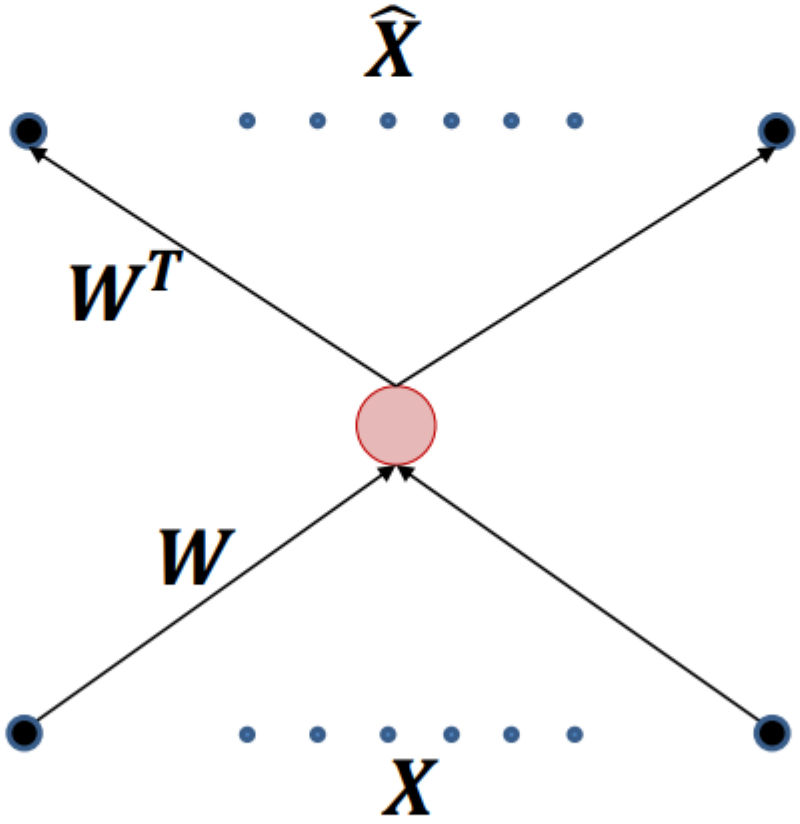
Camera Obscura, the renaissance autoencoder

Undercomplete Autoencoders

- Typically not interested in the output of the decoder x'
- Hope: training the autoencoder will result in z having useful properties
- $\text{Dim}(z) < \text{Dim}(x) \Rightarrow$ ***undercomplete*** autoencoder
  - Forces the AE to capture the most salient features of the training data
  - The AE only approximately copies the input (lossy compression)

Undercomplete Autoencoders

- Special case: Linear decoder, mean squared error loss
- Example: single hidden unit
- What will this learn?

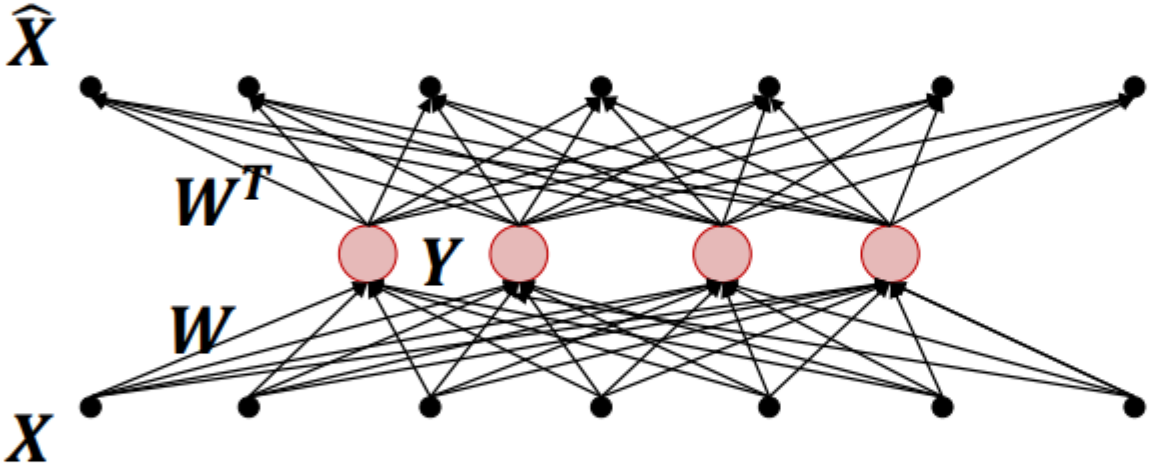


- $\hat{x} = w^T w x$
  - Recall w is a row vector
- Minimize reconstruction error:

$$\hat{w} = \arg\min_{w} \mathbb{E}[\|x - w^T w x\|^2]$$

- Equivalent to PCA!

Undercomplete Autoencoders



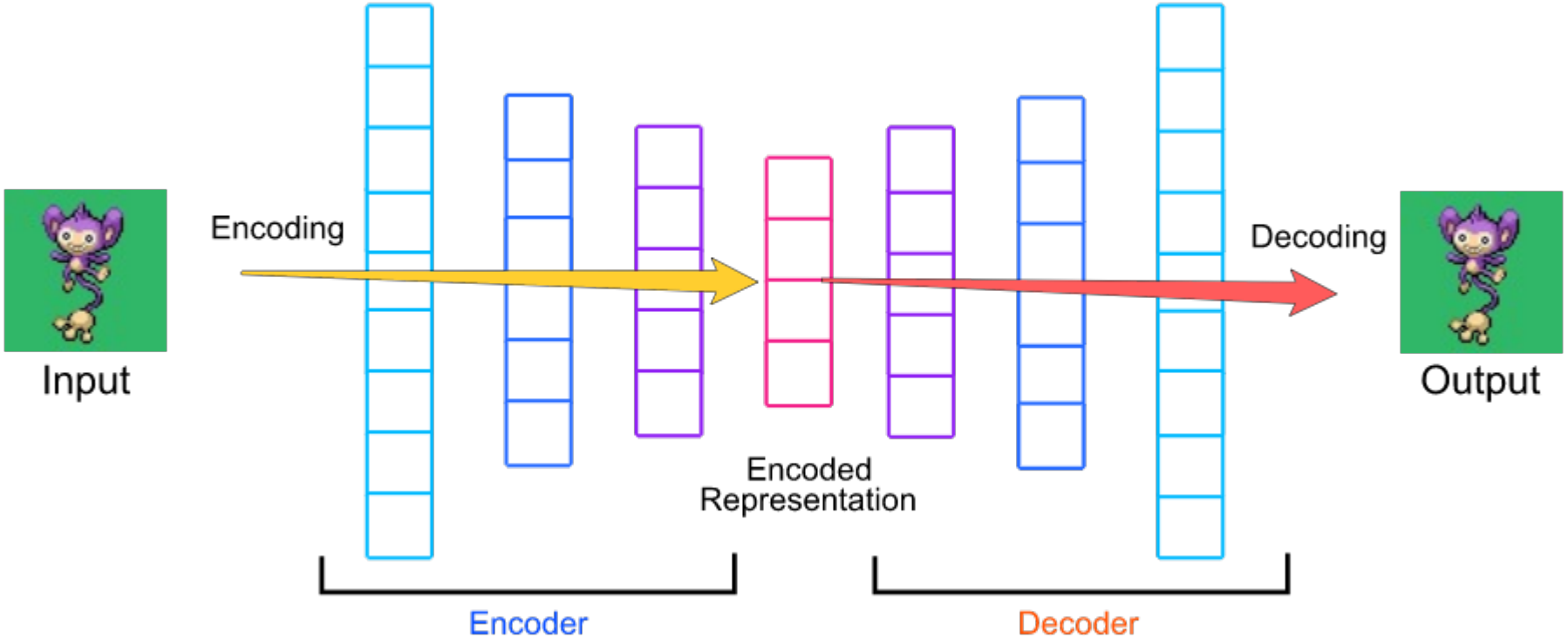
- What about more hidden nodes?
- $Y = WX, \hat{X} = W^T Y$
- Find W to minimize $\mathbb{E}[\|X - W^T WX\|^2]$
- Still PCA

Undercomplete Autoencoders

- Special case: Linear decoder, mean squared error loss
  - Undercomplete AE learns to span the same subspace as PCA
  - The AE has learned the principal subspace of the training data
- Nonlinear encoder and decoder functions can give powerful nonlinear generalization of PCA
  - Be careful with capacity
  - Too much capacity ⇒ AE learns to copy the input w/o extracting useful information
- Example: could learn a 1-dimensional code for the entire dataset
  - Map each training example x_i to the integer i
  - Nothing useful is learned in this case

Autoencoding Pokemon

- Credit: Niyas Mohammed
  - https://hackernoon.com/how-to-autoencode-your-pok%C3%A9mon-6b0f5c7b7d97



Autoencoding Pokemon

The data: Nintendo DS bitmap images

- 64×64 color images
- $\Rightarrow$ each image represented by $64 \times 64 \times 3 = 12{,}288$ values
- Images from the first 5 generations
  - 649 Pokemon?

Autoencoding Pokemon

First Generation

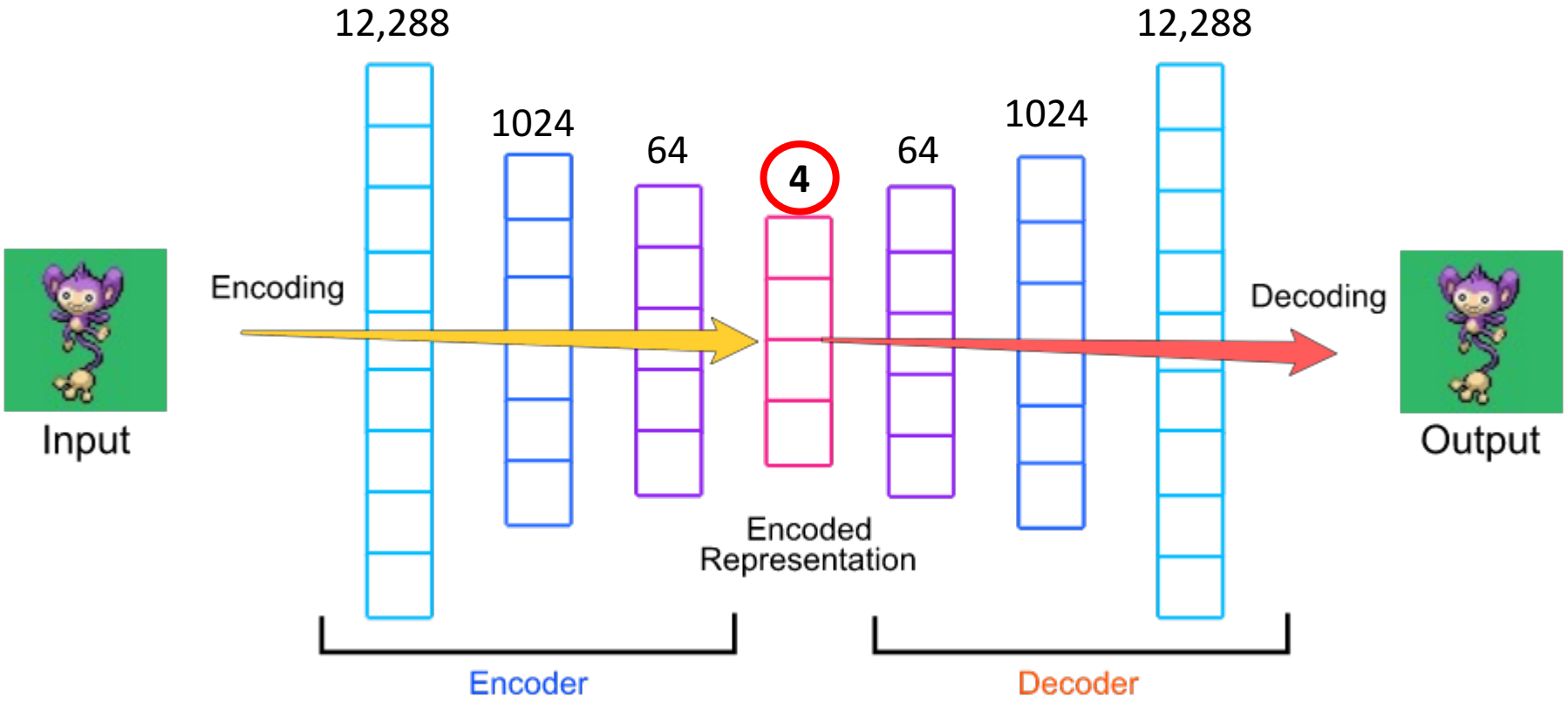
Autoencoding Pokemon

- Images from the first 5 generations
  - 649 Pokemon?
- 4 views per Pokemon

- ≈ 2600 images

Autoencoding Pokemon

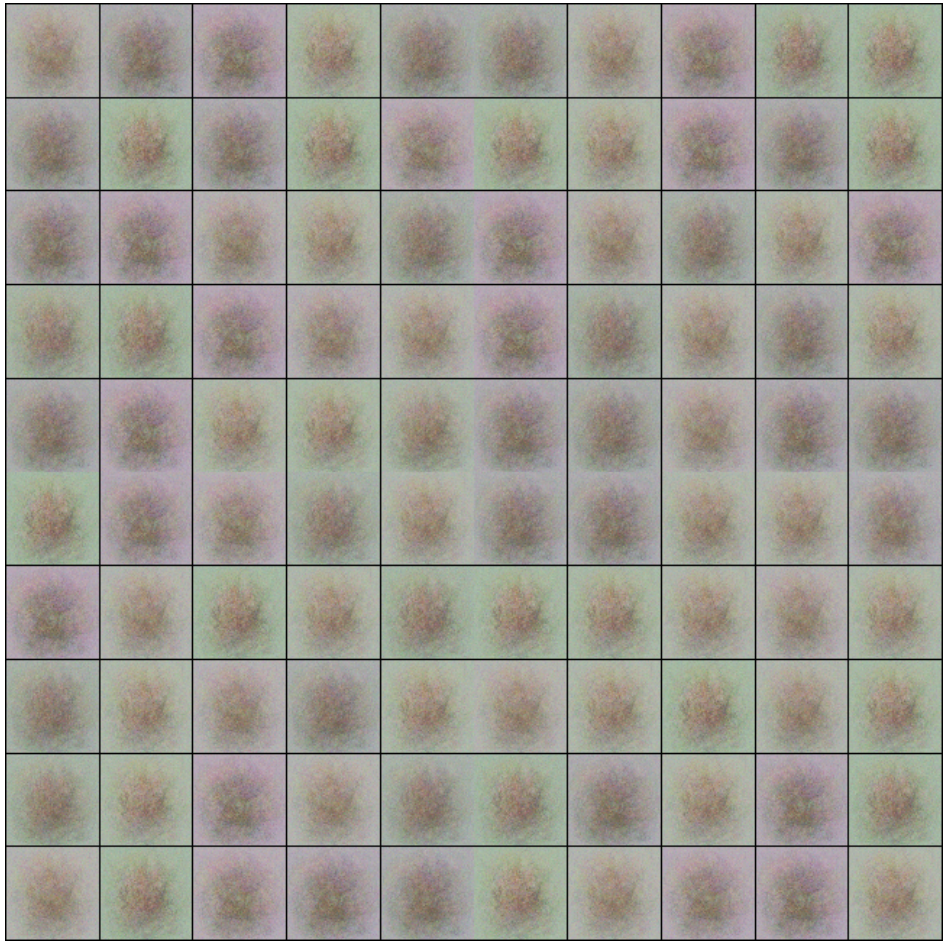
- The architecture



Autoencoding Pokemon

Autoencoding Pokemon

Autoencoding Pokemon



Regularized Autoencoders

- Undercomplete AEs can fail to learn anything useful if the capacity is too large
- Same problem if the hidden code is dimension equal to or greater than the input dimension
  - Latter case gives an ***overcomplete*** AE
  - Even linear encoder and decoder wouldn't be useful

- Regularized AEs modify the loss function to encourage desirable properties
  - E.g., sparsity, robustness to noise
  - Regularized nonlinear and overcomplete AE can still learn something useful about the data distribution

Sparse Autoencoders

- Commonly used for feature learning for another task
- Include a sparsity penalty $\Omega(z)$ on the code layer z:

$$L\left(x, g\left(f(x)\right)\right) + \Omega(z)$$

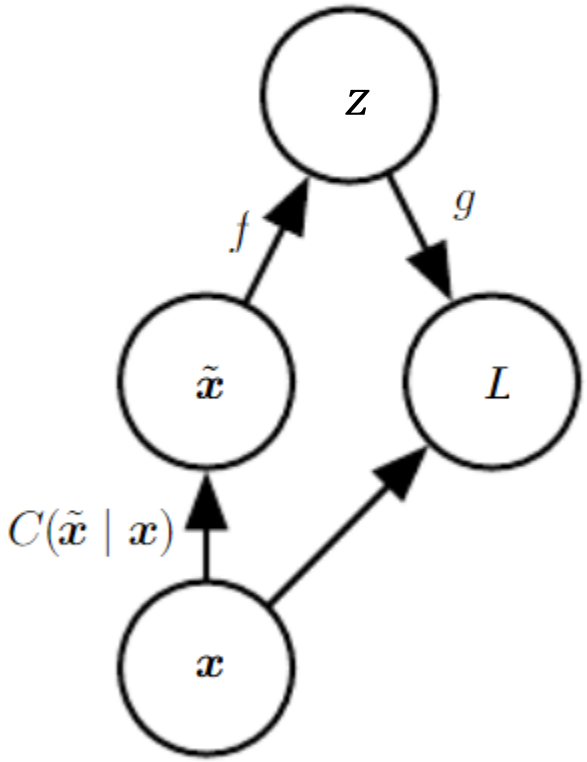
  - Recall, encoder output $z = f(x)$, decoder output $g(z)$
- Example: L1 penalty
  - $\Omega(z) = \lambda \sum_i |z_i|$
  - Alternate view: a Laplace distribution model on the code layer

$$p_{model}(z_i) = \frac{\lambda}{2} e^{-\lambda |z_i|}$$

  - Minimizing the negative log-likelihood is equivalent to the L1 penalty
- To achieve actual zeros, use ReLU activation prior to the code layer

Denoising Autoencoders

- Traditional AE minimizes $L\left(x, g\left(f\left(x\right)\right)\right)$
- ***Denoising autoencoder*** minimizes $L\left(x, g\left(f\left(\tilde{x}\right)\right)\right)$
  - $\tilde{x}$ is a corrupted version of x
  - Denoising AEs learn to undo this corruption
- Corruption process $C(\tilde{x}|x)$ modeled by conditional distribution
- The AE learns a reconstruction distribution $p_{reconstruct}(x|\tilde{x})$ from training pairs $(x, \tilde{x})$



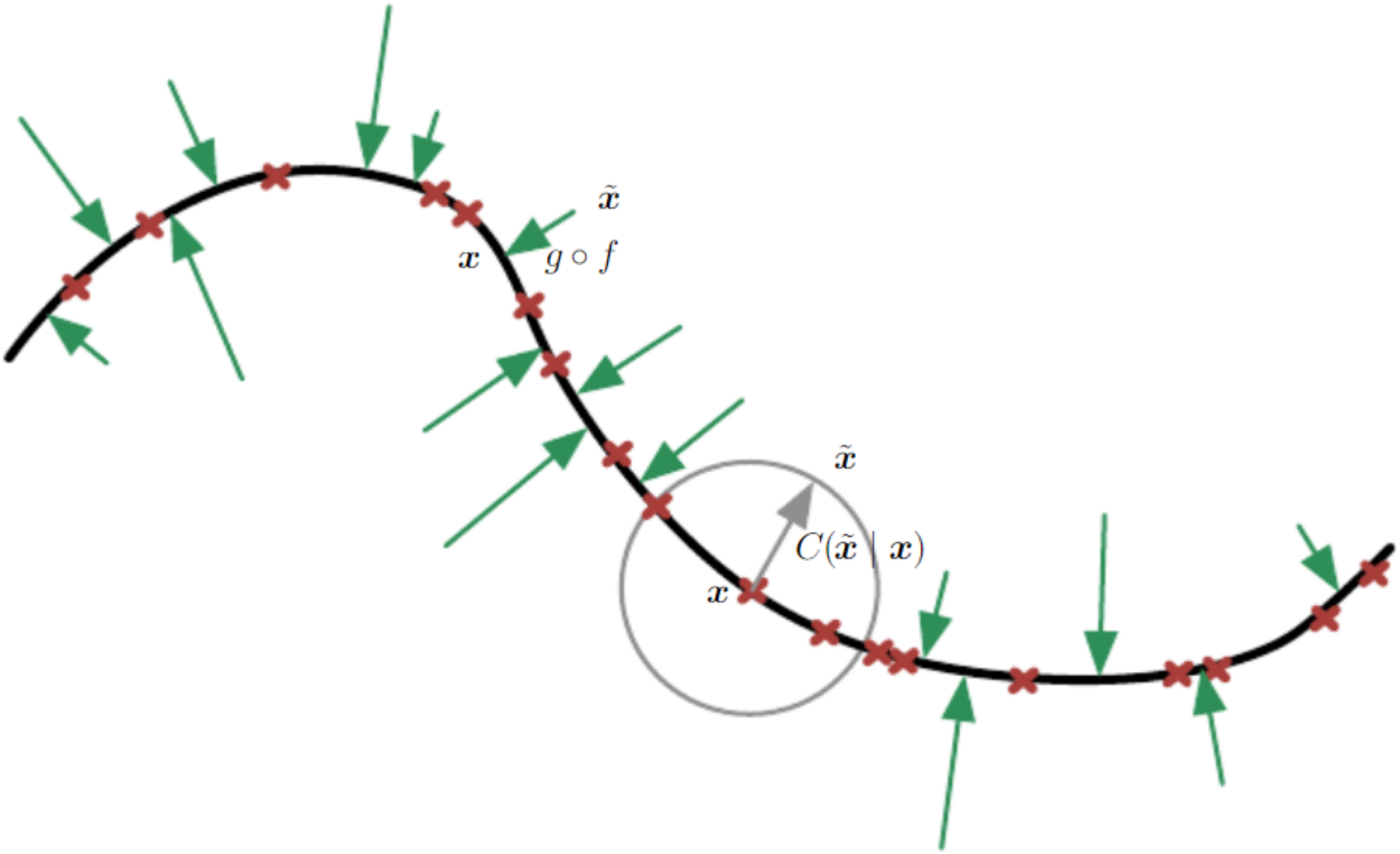
Denoising Autoencoders

The process:

1. Sample training example x from training data
2. Sample corrupted version $\tilde{x}$ from $C(\tilde{x}|x)$
3. Use $(x, \tilde{x})$ as a training example for estimating $p_{reconstruct}(x|\tilde{x}) = p_{decoder}(x|z)$
   - $z = f(\tilde{x})$ the encoder output, $p_{decoder}$ defined by decoder $g(z)$

- Can perform SGD on the negative log-likelihood $-\log p_{decoder}(x|z)$
  - Equivalent to doing SGD on
    $$-\mathbb{E}_{x\sim p_{data}(x)}\mathbb{E}_{\tilde{x}\sim C(\tilde{x}|x)}\log p_{decoder}(x|z = f(\tilde{x}))$$
    - p_{data} is the training distribution

Denoising Autoencoders



Autoencoding MNIST

- From https://blog.keras.io/building-autoencoders-in-keras.html
- Single hidden layer with 32 nodes, fully connected layers
- Cross-entropy loss per pixel, ReLU activations
- Train for 50 epochs
  - Reaches a stable train/test loss of 0.11
  - Top = original, bottom = reconstructed digits

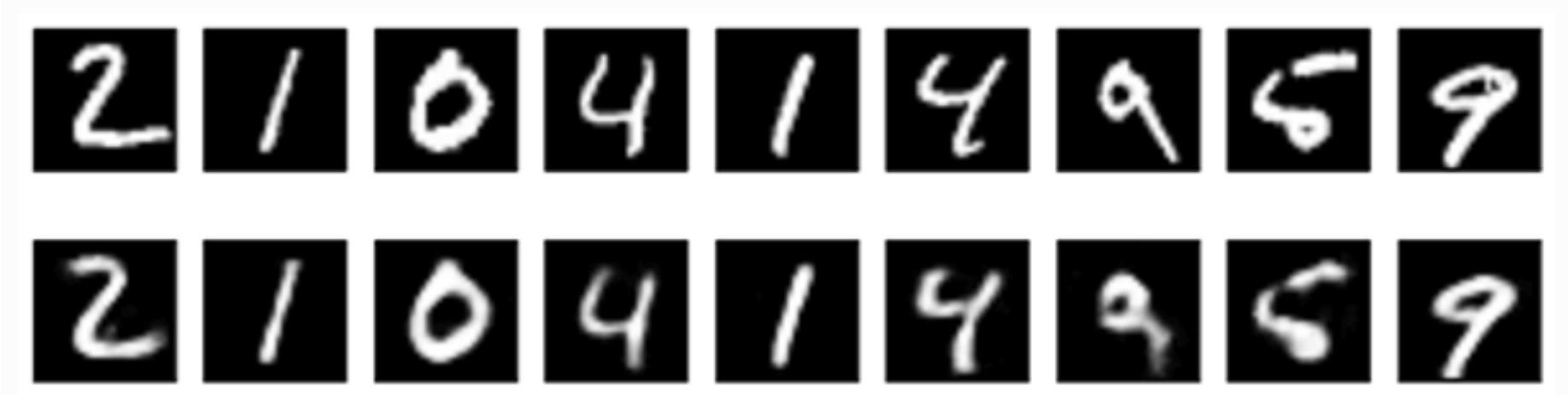
Autoencoding MNIST

- Single hidden layer with 32 nodes, fully connected layers
- Cross-entropy loss per pixel, ReLU activations
- Add L1 sparsity penalty to hidden layer
- Train for 100 epochs
  - Train loss 0.11, test loss 0.10
  - Top = original, bottom = reconstructed digits

- Look the same, but sparser code

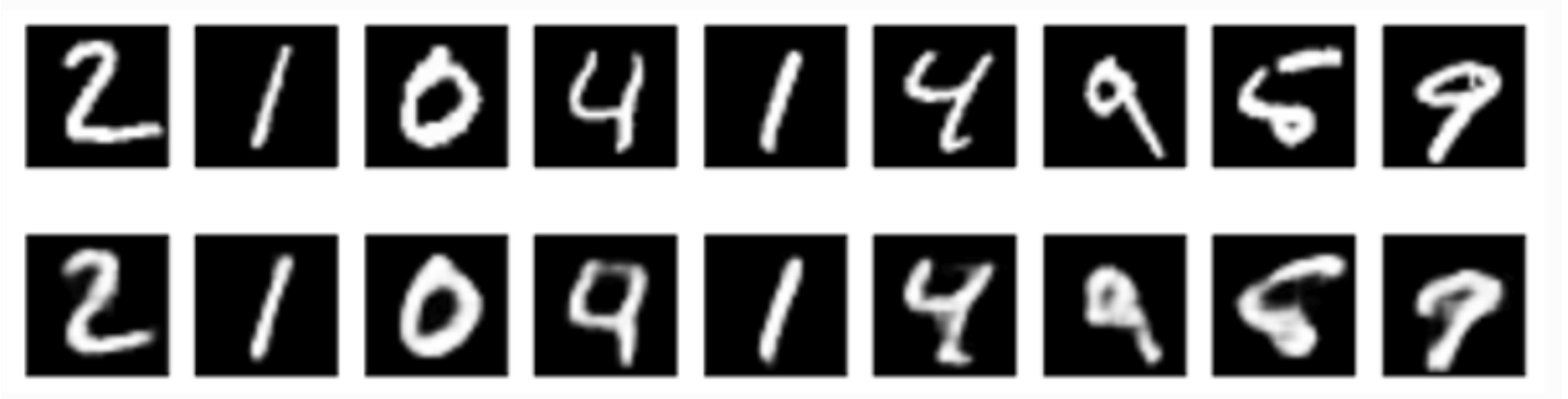
Autoencoding MNIST

- Add two other hidden layers on each side of size 128 & 64
- Cross-entropy loss per pixel, ReLU activations
- Train for 100 epochs
  - Train/test loss ≈ 0.097
  - Top = original, bottom = reconstructed digits



Autoencoding MNIST

- Convolutional autoencoder
- Loss of 0.094
- Top = original, bottom = reconstructed digits



Autoencoding MNIST

- Denoising autoencoder
  - Add Gaussian noise
- Top = noisy original, bottom = reconstructed digits

Further reading

- https://hackernoon.com/how-to-autoencode-your-pok%C3%A9mon-6b0f5c7b7d97
- U. Toronto CSC 411 Lecture 14 slides by Prof. Urtasun & Zemel
- CMU Intro to deep learning Lecture 14 slides by Prof Raj
- https://blog.keras.io/building-autoencoders-in-keras.html
- Goodfellow et al., chapter 14