PCA and Autoencoders

Mesut Erhan Unal

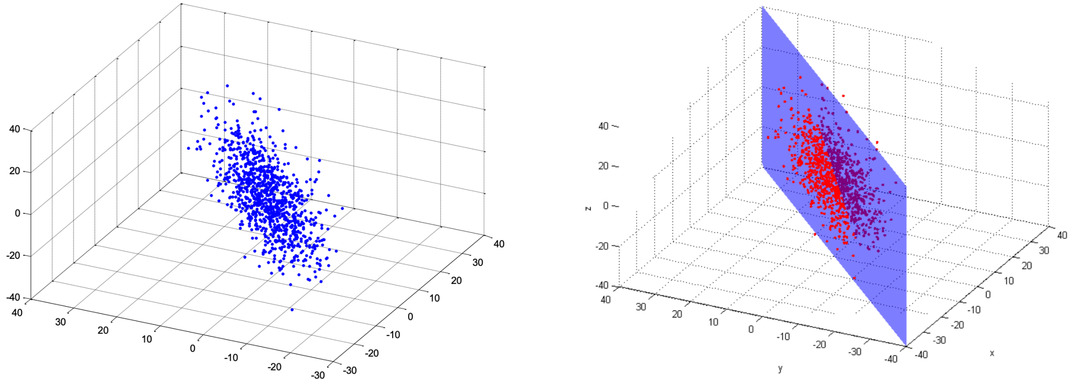
Roadmap

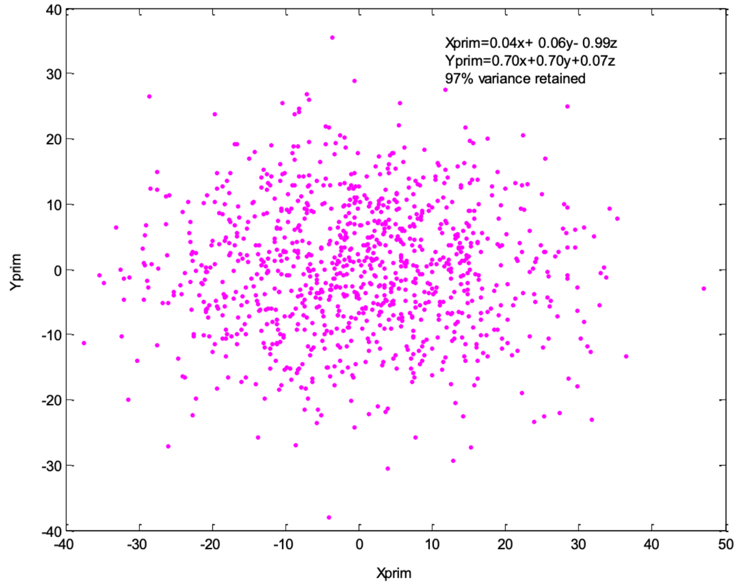
1. PCA
   - Definition and introduction
   - Optimization perspective
   - Pros and cons
   - Example: Eigenfaces
   - Computational aspect
2. Autoencoders
   - Definition and introduction
   - Sub-types
     - Sparse Autoencoders
     - Denoising Autoencoders
     - Contractive Autoencoders
   - Example
3. Q&A

PCA: Definition and introduction

Principal Component Analysis (PCA) is a linear transformation that projects data vectors from d-dimensional space to k-dimensional space where $k < d$, while retaining as much as possible variance present in the dataset.

PCA assumes data is generated by a few hidden causes or factors, so each data point can be described compactly by how much each factor contributes to generate it.





Why we use it?

- To visualize data more easily
- To remove noise present in data
- To have lower resource requirements to store / process data
- and many more.

Credit: CS2750 Spring 2019 Lecture Slides (22) by Milos Hauskrecht

PCA: Change of basis

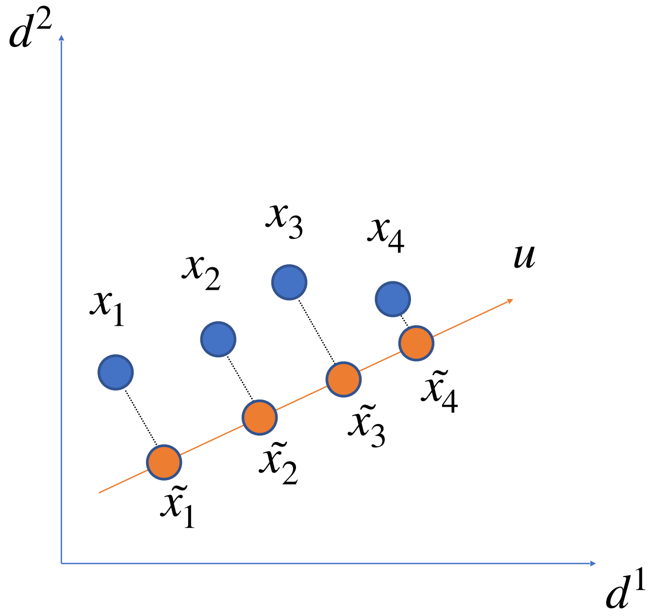
Let us assume we have a data matrix $M^{d \times n}$, in which n data points reside as column vectors, it can be written as a linear combination of orthonormal basis vectors as $UZ = M$ where $U^{d \times d}$ is an orthogonal matrix.

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 4\sqrt{2} \\ \sqrt{2} \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \end{bmatrix}$$

Since $U^T U$ is I, we also have $Z = U^T M$.

Goal: Find the best k basis vectors (*principal components*) to re-express M while keeping as much variance as possible.

PCA: How to choose basis vectors?



We want to find best u that maximizes the variance of among $\tilde{x}$ (*make $\tilde{x}$s as scattered as possible*).

$$\tilde{X} = u^T X$$

$$\bar{\tilde{X}} = \frac{\tilde{x}_1 + \tilde{x}_2 + \tilde{x}_3 + \tilde{x}_4}{4}$$

$$\bar{\tilde{X}} = \frac{u^T(x_1 + x_2 + x_3 + x_4)}{4}$$

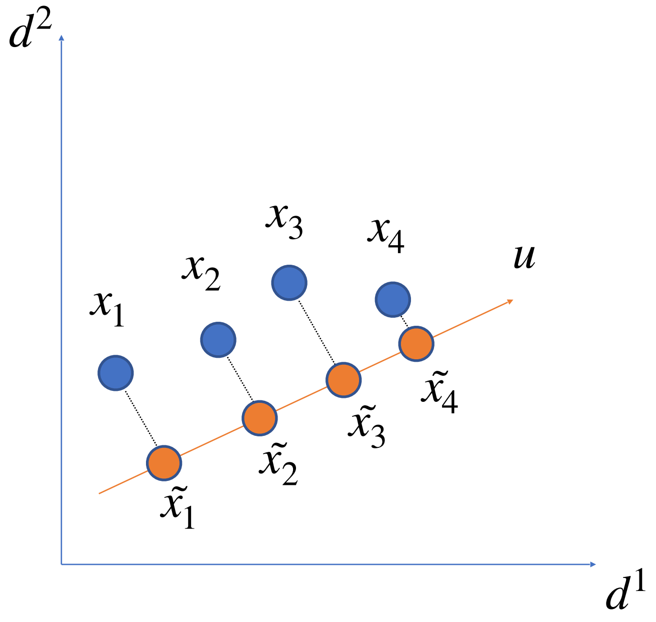
$$\bar{\tilde{X}} = u^T \bar{X}$$

So, the variance among projected data will be

$$\frac{(\tilde{x}_1 - \bar{\tilde{X}})^2 + (\tilde{x}_2 - \bar{\tilde{X}})^2 + (\tilde{x}_3 - \bar{\tilde{X}})^2 + (\tilde{x}_4 - \bar{\tilde{X}})^2}{4}$$

We want to maximize this

PCA: How to choose basis vectors?



$$J(u) = \frac{1}{N}\sum_{n=1}^{N}(u^T x_n - u^T \bar{X})^2$$

$$= \frac{1}{N}\sum_{n=1}^{N} u^T (x_n - \bar{X})(x_n - \bar{X})^T u$$

$$= u^T \left[\frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{X})(x_n - \bar{X})^T\right] u$$

Covariance matrix

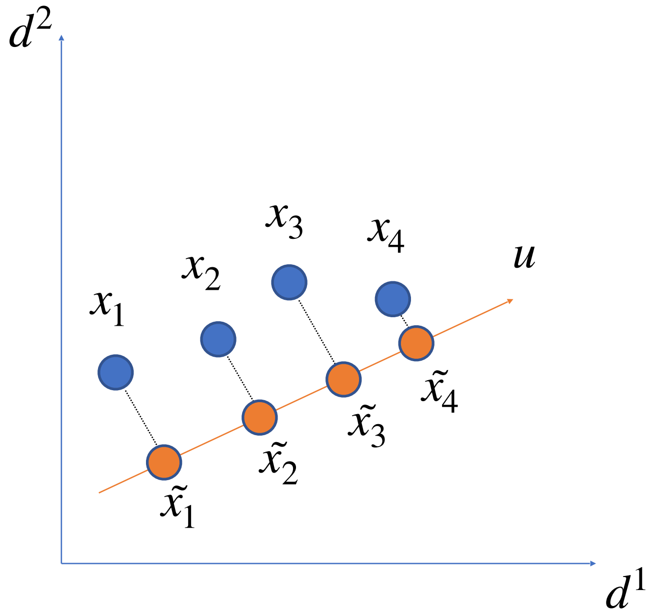
$$= u^T \Sigma u$$

Our optimization problem turns out to be

$$\max J(u) = u^T \Sigma u$$

$$s.t: u^T u = 1$$

PCA: How to choose basis vectors?



After bringing the constraint with its Lagrange multiplier into the original equation we have:

$$\min J(u, \lambda) = u^T \Sigma u - \lambda(u^T u - 1)$$

$$\frac{\partial J}{\partial u} = 2\Sigma u - 2\lambda u = 0$$

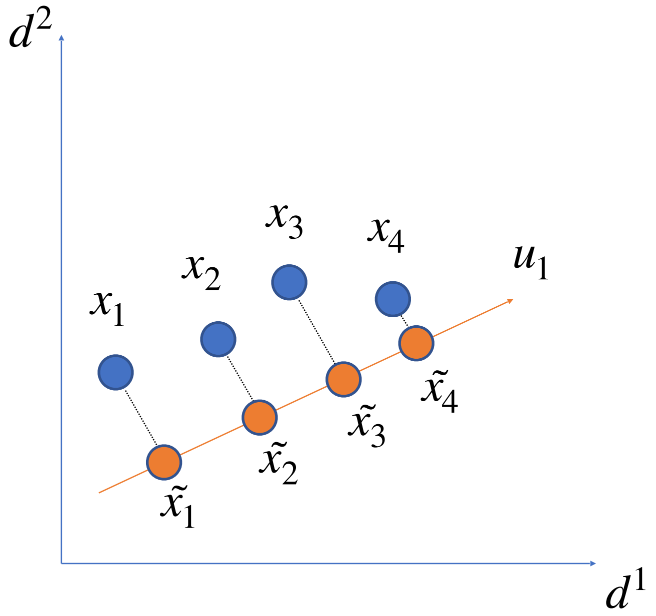
$$\Sigma u = \lambda u$$

Now we know u must be an eigenvector of X's covariance matrix, and λ is corresponding eigenvalue. But which (u, λ) pair we need to pick? Remember out initial objective was maximizing $u^T \Sigma u$.

$$\max J(u) = u^T \Sigma u$$
$$= u^T \lambda u$$
$$= \lambda u^T u$$
$$= \lambda$$

We want λ to be as large as possible.

PCA: How to choose basis vectors?



There is also another way to prove this using reconstruction error. For the sake of the simplicity, let us assume data is mean normalized and we are projecting our data from $\mathbb{R}^2$ to $\mathbb{R}$.

$$[z_1 \quad z_2 \quad z_3 \quad z_4] = \begin{bmatrix} \dots u_1^T \dots \\ \dots u_2^T \dots \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x_1 & x_2 & x_3 & x_4 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

We are dropping this orthonormal basis

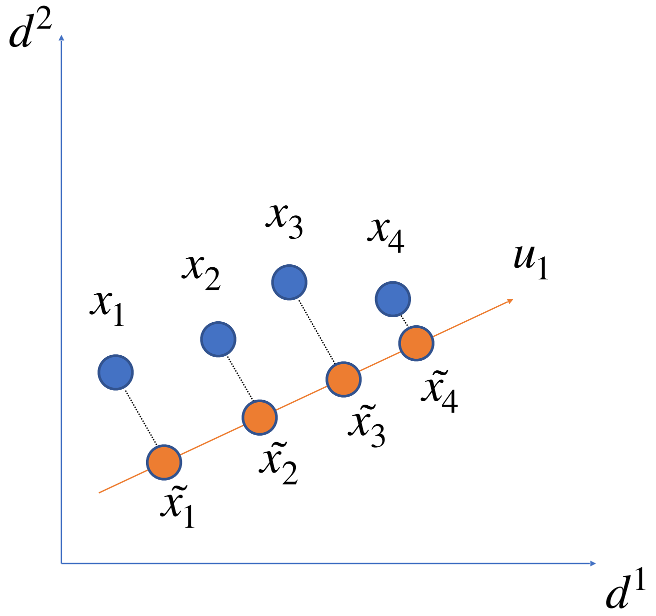
So $x \rightarrow z \rightarrow \hat{x}$ transformations can be defined as

$$z_i = u_1^T x_i$$

$$\hat{x}_i = u_1 z_i$$

$$= u_1 u_1^T x_i$$

PCA: How to choose basis vectors?



Then we can write mean reconstruction error as follows

$$J = \frac{1}{N}\sum_{i=1}^{N} \| x_i - \tilde{x}_i \|^2$$

$$= \frac{1}{N}\sum_{i=1}^{N} \| x_i - u_1 u_1^T x_i \|^2$$

$$= \frac{1}{N}\sum_{i=1}^{N} \| u_2 u_2^T x_i \|^2$$

Reconstruction error can be thought as how the basis we dropped would project x_i and reconstruct it back.

$$= \frac{1}{N}\sum_{i=1}^{N} (u_2^T x_i)^2 \| u_2 \|^2$$

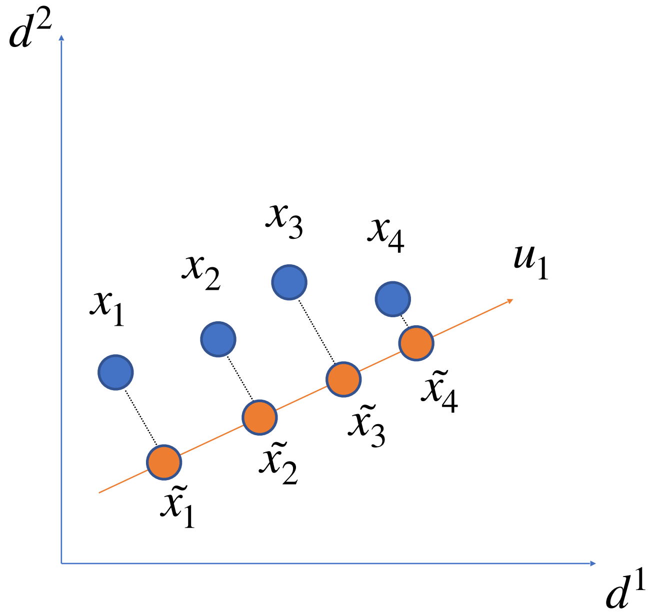
Since u_2 is an orthonormal basis, this equals to 1.

$$= \frac{1}{N}\sum_{i=1}^{N} u_2^T x_i x_i^T u_2$$

$$= u_2^T \left[\frac{1}{N}\sum_{i=1}^{N} x_i x_i^T \right] u_2$$

Covariance matrix

PCA: How to choose basis vectors?



Our optimization problem turns to be:

$$\min J(u_2) = u_2^T \Sigma u_2$$
$$s.t: \ u_2^T u_2 = 1$$

After plugging the constraint into the equation with its Lagrange multiplier:

$$\min J(u_2, \lambda) = u_2^T \Sigma u_2 - \lambda(u_2^T u_2 - 1)$$
$$\frac{\partial J}{\partial u_2} = 2\Sigma u_2 - 2\lambda u_2 = 0$$
$$\Sigma u_2 = \lambda u_2$$

We know u_2 is an eigenvector of X's covariance matrix, and λ is its corresponding eigenvalue. But which (u_2, λ) pair to drop? Remember our initial objective was minimizing $u_2^T \Sigma u_2$.

$$\min J(u) = u_2^T \Sigma u_2$$
$$= u_2^T \lambda u_2$$
$$= \lambda u_2^T u_2$$
$$= \lambda$$

We want λ to be as small as possible.

PCA: Pros and cons

Pros

- Deterministic.
- Relative differences in data points tend to be preserved.
- Easy to implement.

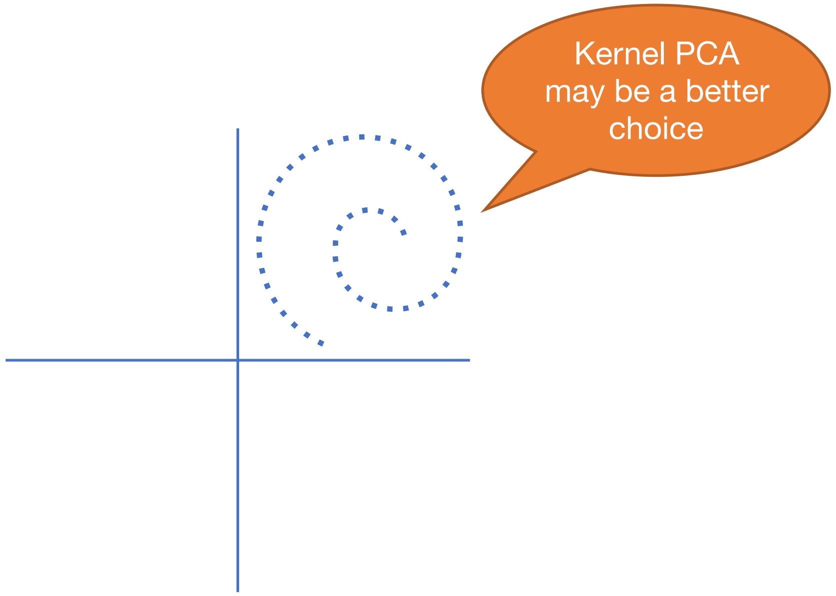
PCA: Pros and cons

Pros

- Deterministic.
- Relative differences in data points tend to be preserved.
- Easy to implement.

Cons

- Relies on linearity assumption.



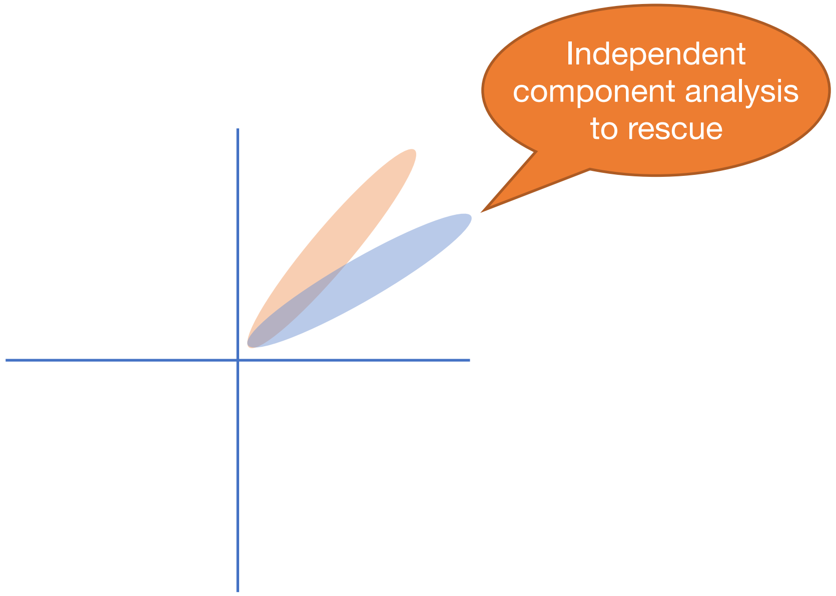
PCA: Pros and cons

Pros

- Deterministic.
- Relative differences in data points tend to be preserved.
- Easy to implement.

Cons

- Relies on linearity assumption.
- Relies on orthogonal transformations.



PCA: Pros and cons

Pros

- Deterministic.
- Relative differences in data points tend to be preserved.
- Easy to implement.

Cons

- Relies on linearity assumption.
- Relies on orthogonal transformations.
- Assumes mean and covariance can describe the distribution.

Eigenfaces

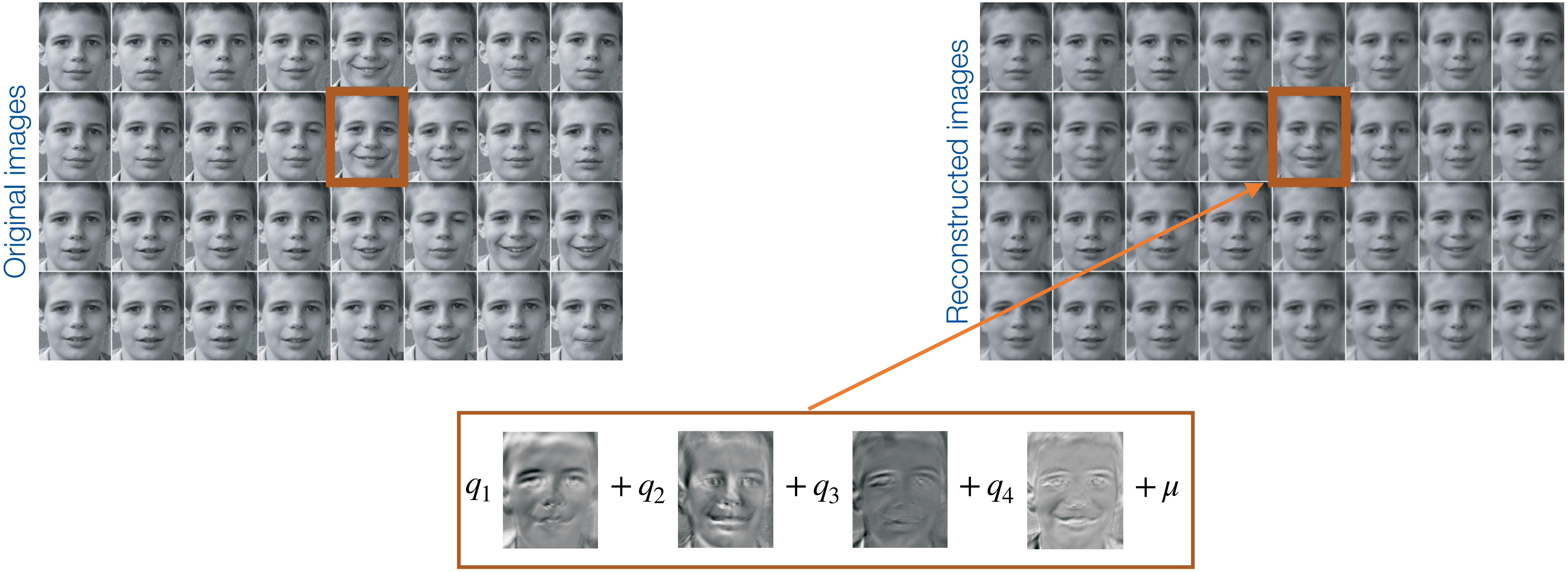
(Turk & Pentland, 1991) applied PCA to produce low dimensional representations of faces.

Approach

- Flatten every image as a vector.
- Calculate mean vector (*face*) over dataset.
- Subtract mean from each face.
- Calculate covariance matrix on mean-normalized data.
- Perform eigendecomposition and select k eigenvectors (*PCs*) to define bases for k -dimensional projection space.
- Basis vectors are called ***Eigenfaces***.
- During reconstruction, add mean face vector onto reconstructed face.

Eigenfaces

Reconstruction using 4 basis vectors (*principal components*).



PCA: Computational aspect

- **Eigendecomposition of covariance matrix**

  Let $M^{d \times n}$ be the mean-normalized data matrix, then calculating covariance matrix Σ as MM^T yields $O(nd^2)$. Eigendecomposition on Σ yields $O(d^3)$.

  If $d \gg n$, we can use a trick to perform eigendecomposition on $M^T M$ instead and this gives $O(dn^2)$ for covariance matrix calculation and $O(n^3)$ for eigendecomposition.

  $$M^T Mu = \lambda u \rightarrow MM^T(Mu) = \lambda(Mu)$$

- **Covariance-free methods**
  - Iterative computation of PCs with power iteration
  - The NIPALS method
  - and more

Autoencoders: Definition and introduction

Autoencoder is an artificial neural network that tries to encode data efficiently in an unsupervised manner by being trained to reconstruct its input on its output.

It consists of two functions,

- An encoder function that maps input to a hidden representation
  $f : x \rightarrow h$
- A decoder function that performs reconstruction from hidden representation
  $g : h \rightarrow \hat{x}$

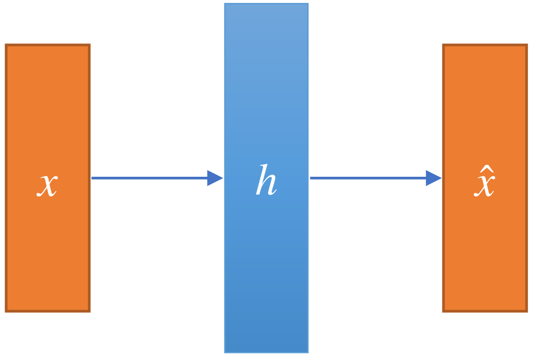
If they are trained to minimize mean squared reconstruction loss

$$L(\xi_f, \xi_g) = \frac{1}{N} \sum_{i=0}^{N} \left\| x_i - g(f(x_i; \xi_f); \xi_g) \right\|^2$$

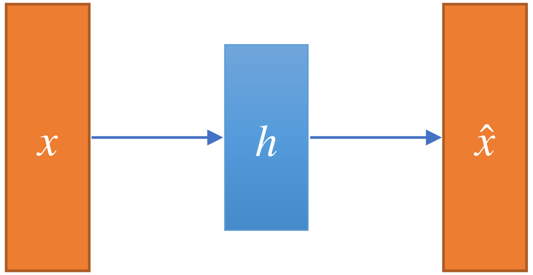
they span the same subspace as PCA if no non-linearity involved in both f and g.

Autoencoders: Definition and introduction

If the projection space is larger than input space in terms of dimensions ($d' \geq d$ where $x \in \mathbb{R}^d, f(x; \xi_f) \in \mathbb{R}^{d'}$), autoencoders tend to learn $f \circ g$ as an identity function.



If the projection space is smaller than input space, an autoencoder is said to be *undercomplete*.



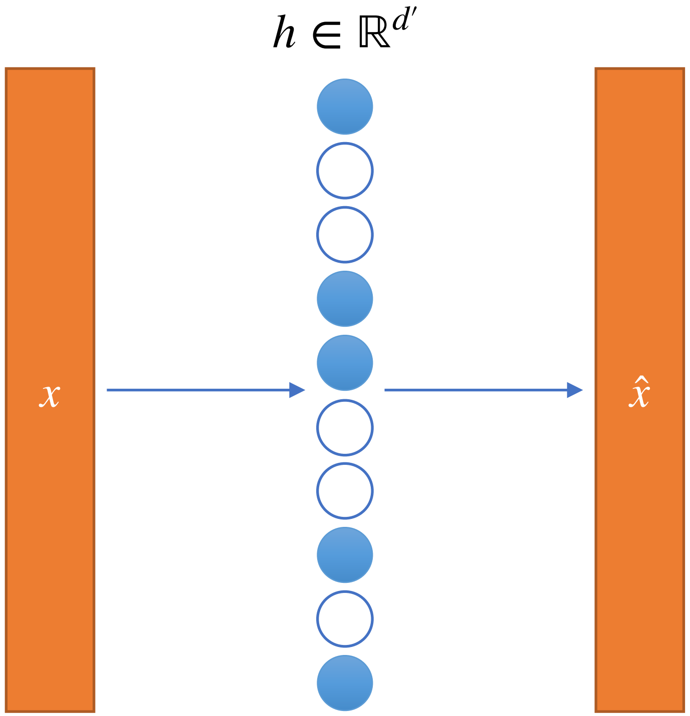
Sparse Autoencoders

Even with a large projection space, autoencoders can be forced to extract useful representations by imposing a sparsity penalty on code layer, h. We can interpret this penalty as we want only a small subset of hidden units to be active at once. For sparse autoencoders, total loss can be written as:

$$J = L(x, \hat{x}) + \Omega(h)$$

Sparsity penalty

Reconstruction loss



Sparse Autoencoders

There are different ways to define a sparsity penalty on code layer, h. Some of them includes:

- To penalize $L1$ norm of h with a scalar λ.

$$J = L(x, \hat{x}) + \lambda \sum_{i=1}^{d'} |h_i|$$

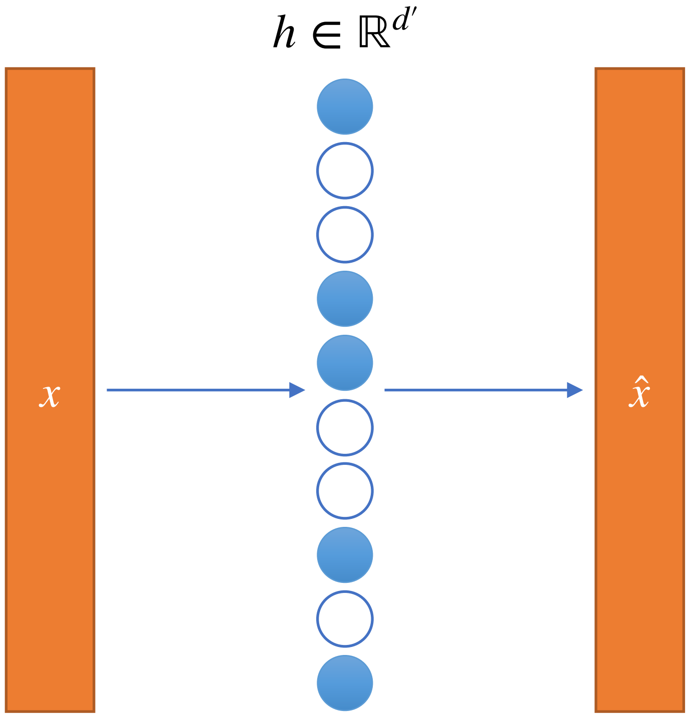
- To introduce a Bernoulli random variable with mean p, and force every hidden unit activations to follow this distribution.

$$J = L(x, \hat{x}) + \beta \sum_{j=1}^{d'} KL(p || \hat{p}_j)$$

$$\hat{p}_j = \frac{1}{N} \sum_{i=1}^{N} [a_j(x_i)]$$

Activation of j^{th} hidden unit averaged over entire dataset.

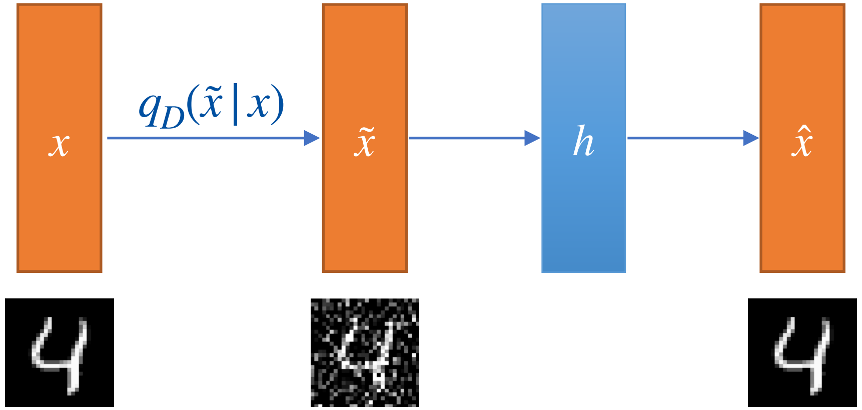
- To zero-out all hidden unit activations but top K.



University of Pittsburgh | School of Computing and Information

Denoising Autoencoders

Rather than adding a penalty term Ω to the cost function, another way to make an autoencoder to learn useful representations is changing its reconstruction criteria.



A *denoising autoencoder* takes inputs that are partially corrupted through a stochastic mapping $\tilde{x} \sim q_D(\tilde{x} \mid x)$, and tries to reconstruct uncorrupted versions.

$$L(\xi_f, \xi_g) = \frac{1}{T} \sum_{i=1}^{T} \left\| x_i - g(f(\tilde{x}_i; \xi_f); \xi_g)) \right\|^2$$

Contractive Autoencoders

One other way to regularize an autoencoder is to add a penalty term Ω to penalize derivatives of hidden units with respect to model input.

$$J = L(x, \hat{x}) + \lambda \left\| \frac{\partial h}{\partial x} \right\|_F^2$$

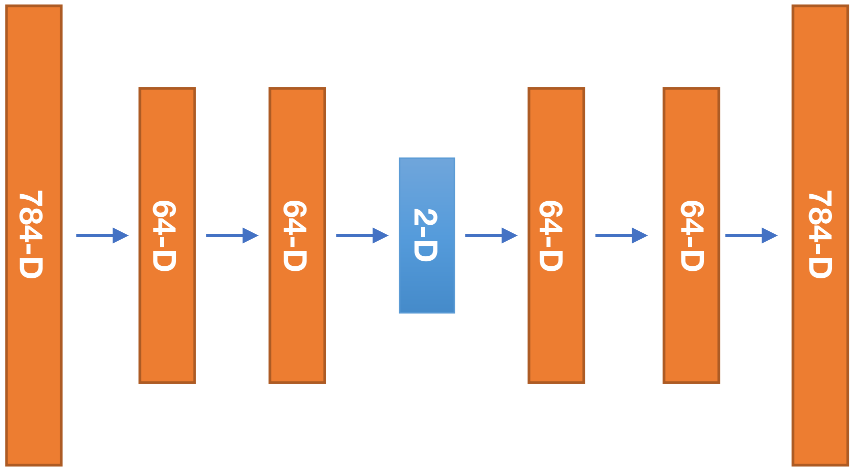
Squared Frobenius norm of the Jacobian matrix.

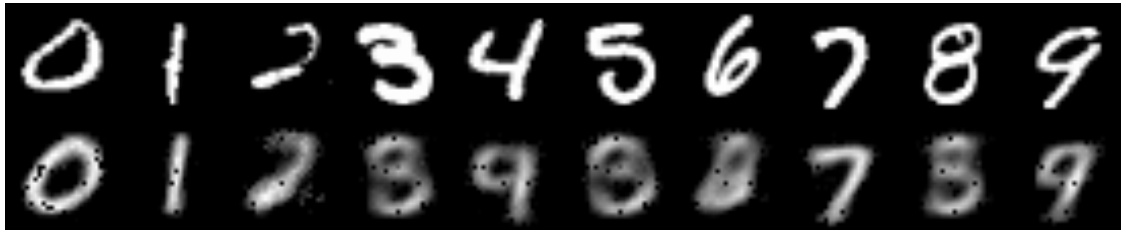
This penalty forces the model to learn a function that does not change much when x is subject to small perturbations. As we can see, both denoising autoencoders and contractive autoencoders want to achieve robustness but there are some differences:

- Contractive autoencoders explicitly encourage robustness on encoder $f(x)$, whereas denoising autoencoders encourage it on reconstruction $(f \circ g)(x)$.
- Denoising autoencoders' robustness is achieved stochastically, while contractive autoencoders achieve it analytically.

Example

- Implemented a deep autoencoder that maps 784-D (*flattened*) MNIST images to 2-D and reconstructs back.
- **MNIST** dataset contains 70,000 (*60,000 train + 10,000 test*) 28x28 handwritten digits.
- The model has been trained for 100 epochs.





https://github.com/meunal/AutoencoderExample

Off-the-shelf implementations

- **PCA**
  - Comes built-in in Matlab and R.
  - Available within 3rd party libraries for Python (*sklearn, statsmodels, mlxtend, etc.*).
- **AutoEncoders**
  - Can be implemented using modern machine learning frameworks (*or using just NumPy, if you want to write backpropagation*).
  - TensorFlow
  - PyTorch
  - MXNet
  - CNTK
  - Theano
  - and more

University of Pittsburgh | School of Computing and Information

References

- Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.A., 2008, July. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning (pp. 1096-1103).
- Jiang, P., Maghrebi, M., Crosky, A. and Saydam, S., 2017. Unsupervised deep learning for data-driven reliability and risk analysis of engineered systems. In Handbook of Neural Computation (pp. 417-431). Academic Press.
- Rifai, S., Vincent, P., Muller, X., Glorot, X. and Bengio, Y., 2011. Contractive auto-encoders: Explicit invariance during feature extraction.
- Wold, S., Esbensen, K. and Geladi, P., 1987. Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3), pp.37-52.
- Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.
- Hauskrecht, M., 2019. CS2750 Lecture notes, 22 (2019)
- Ng, A., 2011. Sparse autoencoder. CS294A Lecture notes, 72(2011), pp.1-19.
- Tong, H., 2018. CSE535 Lecture notes, 4 (2018)
- Kontorovich, A. and Sabato, S., 2017. Introduction to learning and analysis of big data lecture notes, 14 (2017)