Principal Component Analysis

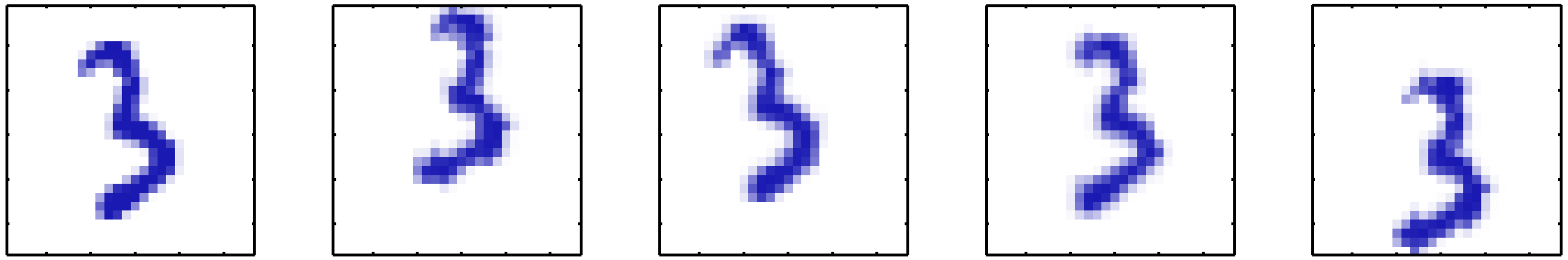
Many slides adapted from CSE 4309 in University of Texas at Arlington

The Curse of Dimensionality

- As the dimensions increase, learning probability distributions becomes more difficult.
  - More parameters must be estimated.
  - More training data is needed to reliably estimate those parameters.
- For example:
  - To learn multidimensional histograms, the number of required training data is exponential to the number of dimensions.
  - To learn multidimensional Gaussians, the number of required training data is quadratic to the number of dimensions.

Do We Need That Many Dimensions?

- Consider these five images.
  - Each of them is a 100x100 grayscale image.
  - What is the dimensionality of each image? 10,000.
- However, each image is generated by:
  - Picking an original image (like the image on the left).
  - Translating (moving) the image up or down by a certain amount t_1.
  - Translating the image left or right by a certain amount t_2.
  - Rotating the image by a certain degree θ.
- If we know the original image, to reconstruct any other image we just need three numbers: t_1, t_2, θ.



Principal Component Analysis(PCA)

- Principal component analysis is a variable(feature) reduction procedure. It is useful when you have obtained data on a number of variables (possibly a large number of variables), and believe that there is some redundancy in those variables

- Redundancy means that some of the variables are correlated with one another, possibly because they are measuring the same construct.

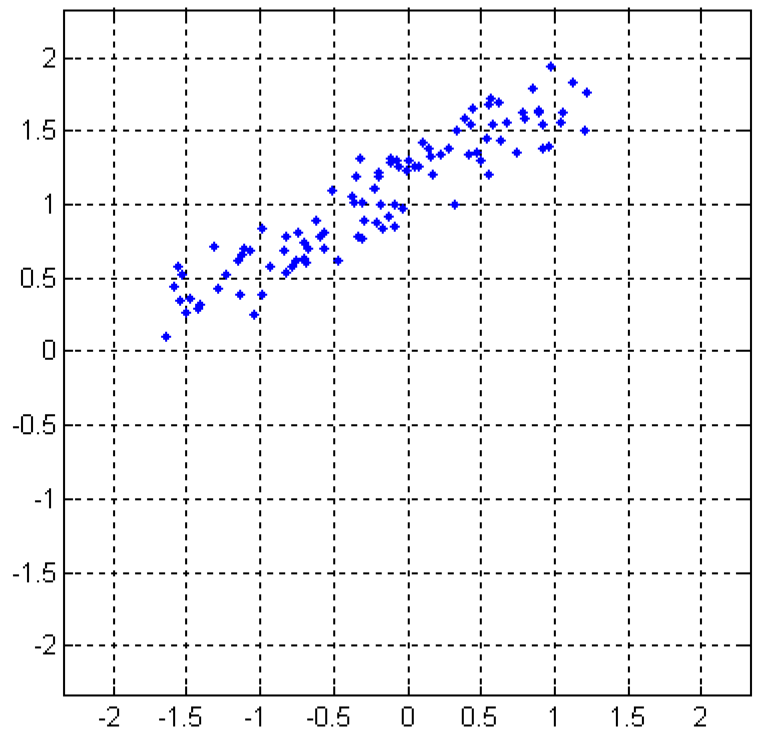
- Because of this redundancy, you believe that it should be possible to reduce the observed variables into a smaller number of principal components (artificial variables) that will account for most of the variance in the observed variables.

Principal Components Analysis

- PCA takes N-dimensional data and finds the M orthogonal directions in which the data have the most variance.
  - These M principal directions form a lower-dimensional subspace.
  - We can represent an N-dimensional datapoint by its projections onto the M principal directions.

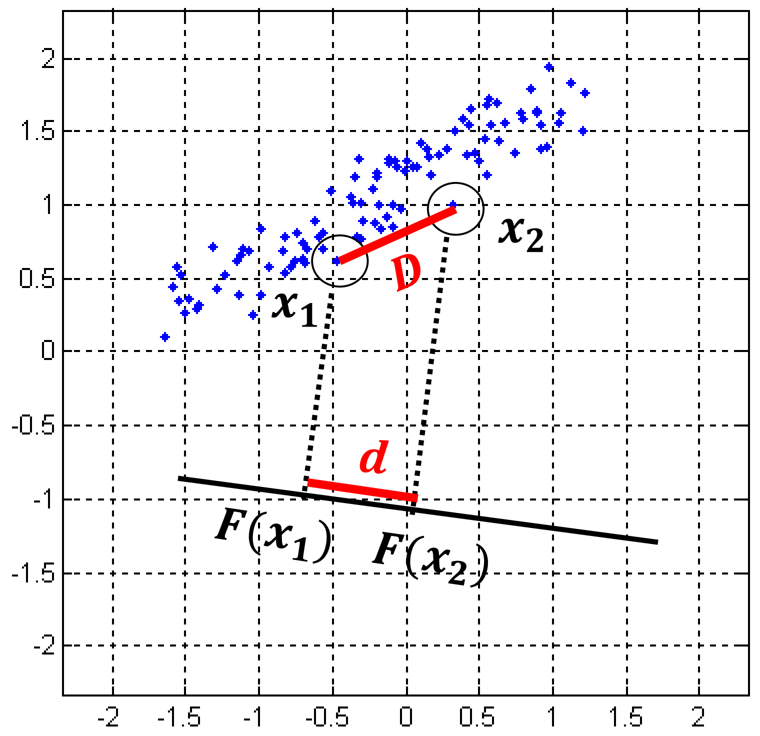
Lossy Dimensionality Reduction

- Suppose we want to project all points to a single line.
- This will be *lossy*.
- What would be the best line?
- Optimization problem.
  - The number of choices is infinite.
  - We must define an optimization criterion.



Optimization Criterion

- Consider a pair of 2-dimensional points: $\boldsymbol{x}_1, \boldsymbol{x}_2$.
- Let F map each 2D point to a point on a line.
  - So, $F\colon\ \mathbb{R}^2 \to \mathbb{R}^1$
- Define $D = \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2$.
  - Squared Euclidean distance from x_1 to x_2.
- Define $d = \|F(\boldsymbol{x}_1) - F(\boldsymbol{x}_2)\|^2$.
  - Squared Euclidean distance from F(x_1) to F(x_2).
- Define error function $\mathrm{E}(\boldsymbol{x}_1, \boldsymbol{x}_2) = D - d$.
- The closer the line F to the line connecting $(\boldsymbol{x}_1, \boldsymbol{x}_2)$, the less E$(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is
- Will $\mathrm{E}(\boldsymbol{x}_1, \boldsymbol{x}_2)$ ever be negative?
  - NO: $D \geq d$ always. Projecting to fewer dimensions can only shrink distances.



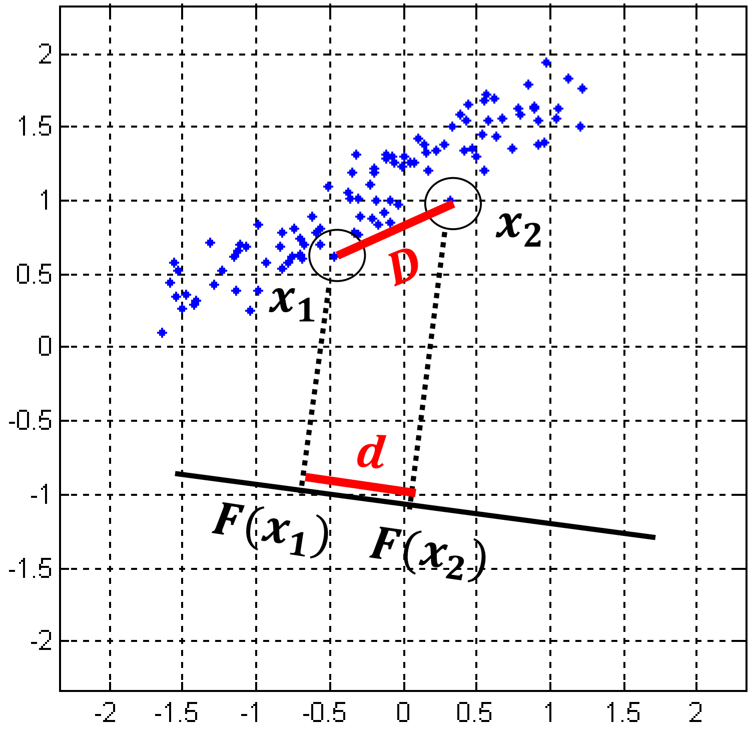
Optimization Criterion

- Now, consider all points:
  - $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$.
- Define error function E as:

$$E(F) = \sum_{m=1}^{N} \sum_{n=1}^{N} E(F, \boldsymbol{x}_m, \boldsymbol{x}_n)$$

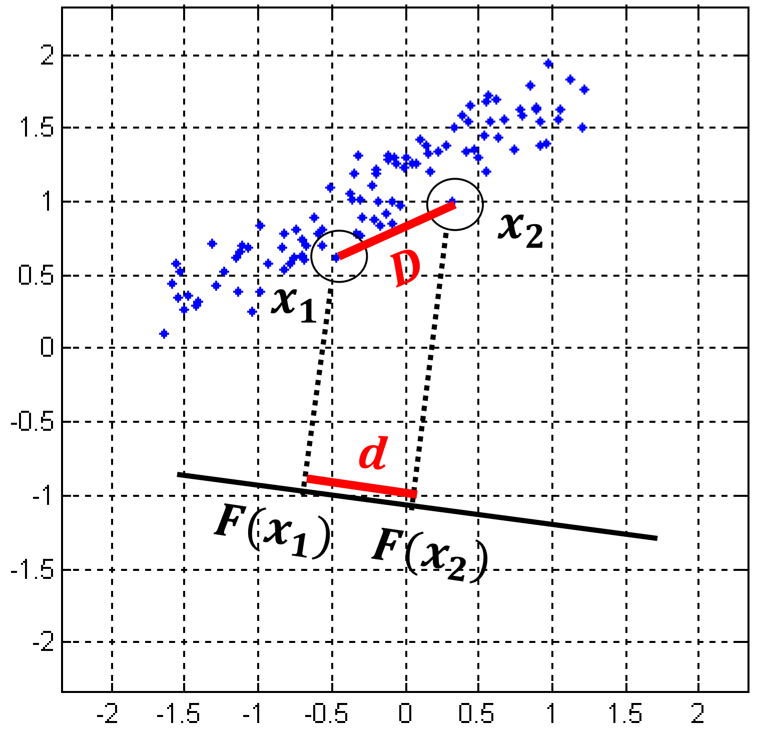
$$= \sum_{m=1}^{N} \sum_{n=1}^{N} \left[\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2 - \|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2 \right]$$

- Interpretation: Error function $E(F)$ measures how well projection F preserves distances.



Optimization Criterion: Preserving Distances

- We have defined an optimization criterion, that measures how well a projection preserves the pairwise distances of the original data.
- This goal is equivalent to the following two other criteria:
  - Maximizing the variance of the projected data $F(\boldsymbol{x}_n)$
  - Minimizing the sum of backprojection errors

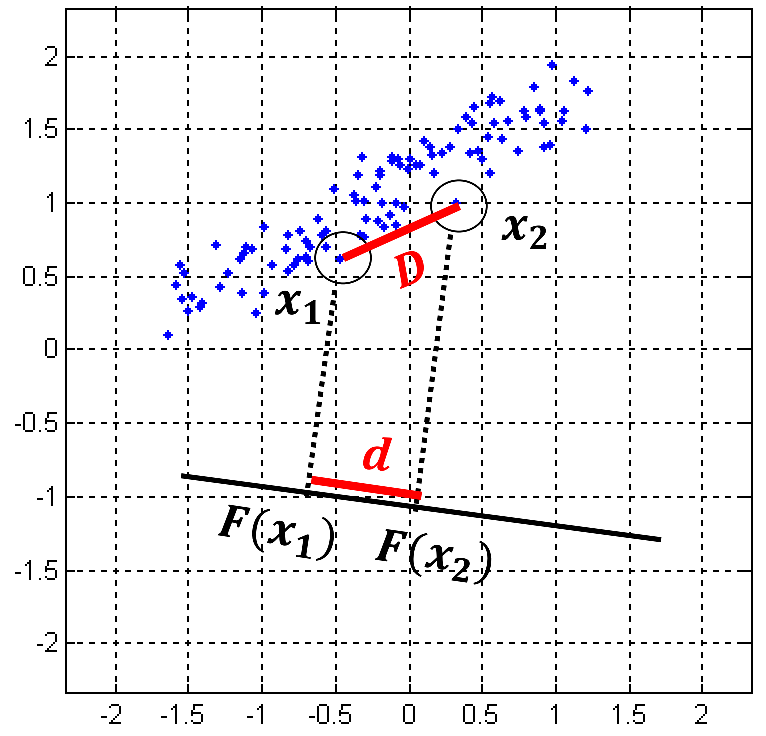


Optimization Criterion: Preserving Distances

- We have defined an error function $E(F)$ that tells us how good a linear projection is.
- Therefore, the best line projection F_{opt} is the one that minimizes $E(F)$.

$$F_{\text{opt}} = \underset{F}{\operatorname{argmin}} E(F) =$$

$$\underset{F}{\operatorname{argmin}} \left\{ \sum_{m=1}^{N} \sum_{n=1}^{N} \left[\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2 - \|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2 \right] \right\}$$



Optimization Criterion: Preserving Distances

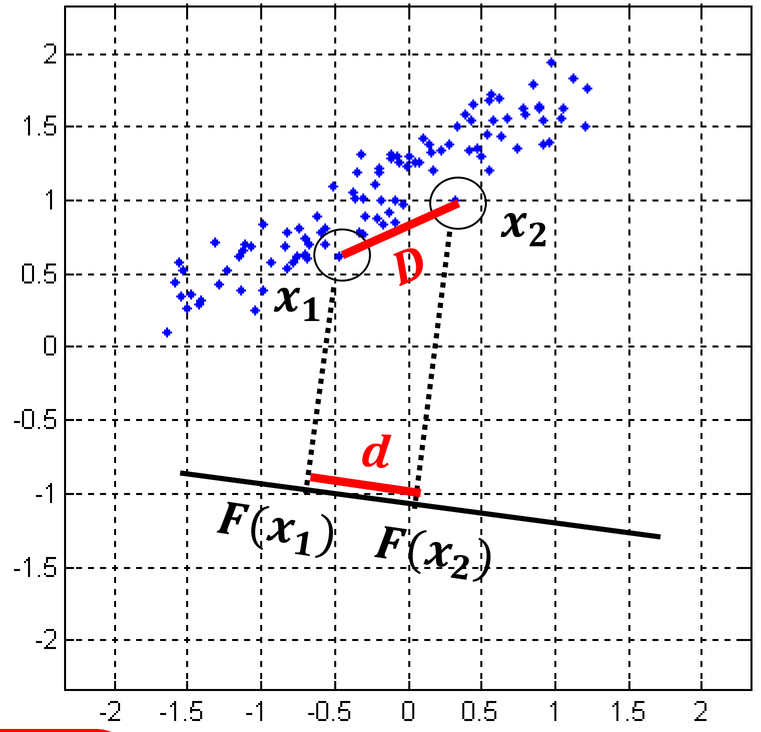
$$E(F) = \sum_{m=1}^{N}\sum_{n=1}^{N} E(F, \boldsymbol{x}_m, \boldsymbol{x}_n)$$

$$= \sum_{m=1}^{N}\sum_{n=1}^{N}\left[\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2 - \|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2\right]$$

$$= \boxed{\cancel{\sum_{m=1}^{N}\sum_{n=1}^{N}\left[\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2\right]}} - \boxed{\sum_{m=1}^{N}\sum_{n=1}^{N}\left[\|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2\right]}$$

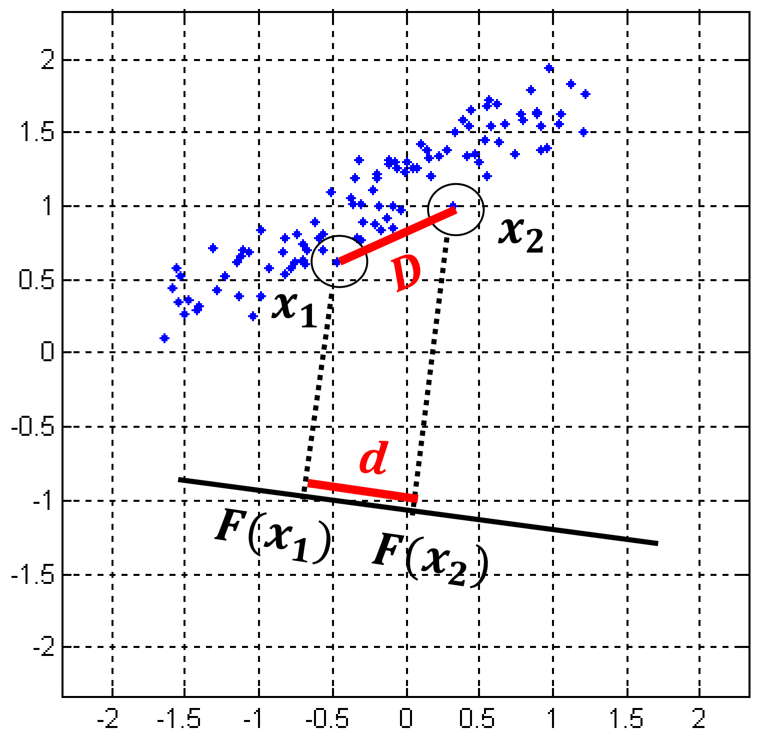
Sum of pairwise distances in original space. Independent of F.

Sum of pairwise distances in projected space. Depends on F.



Optimization Criterion: Maximizing Distances

- Therefore, our original criterion (preserving distances as much as possible) is equivalent to maximizing distances in the projected space.
- For convenience, define $y_n = F(\boldsymbol{x}_n)$.



$$F_{\text{opt}} = \underset{F}{\text{argmax}} \left\{ \sum_{m=1}^{N} \sum_{n=1}^{N} \left[\|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2 \right] \right\}$$

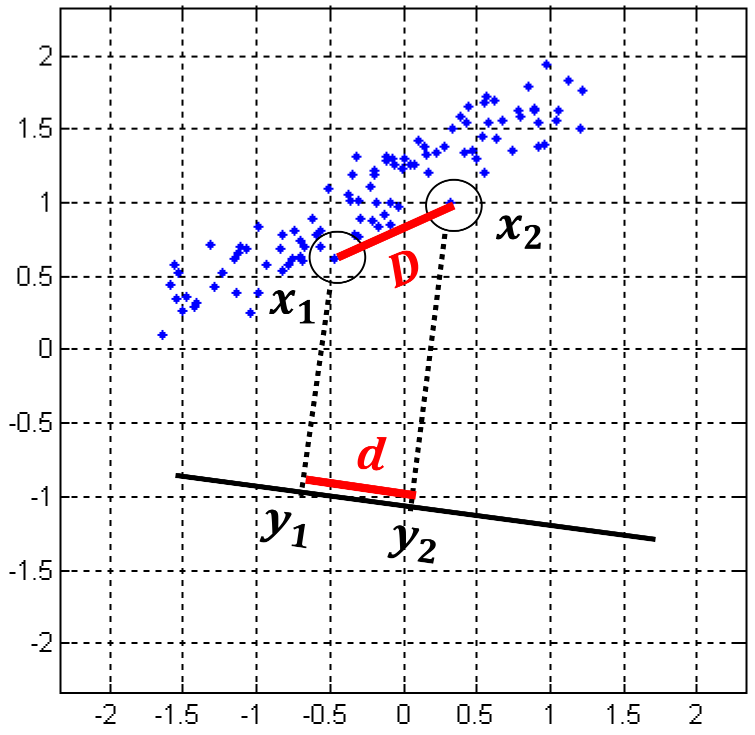
$$\sum_{m=1}^{N} \sum_{n=1}^{N} \left[\|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2 \right] = \sum_{m=1}^{N} \sum_{n=1}^{N} \left[(y_m - y_n)^2 \right] = \sum_{m=1}^{N} \sum_{n=1}^{N} \left[(y_m)^2 + (y_n)^2 - 2 y_m y_n \right]$$

Optimization Criterion: Maximizing Distances

$$\sum_{m=1}^{N}\sum_{n=1}^{N}\left[(y_m)^2 - y_m y_n + (y_n)^2 - y_m y_n\right] =$$

$$\sum_{m=1}^{N}\sum_{n=1}^{N}\left[(y_m)^2 - y_m y_n\right] + \sum_{m=1}^{N}\sum_{n=1}^{N}\left[(y_n)^2 - y_m y_n\right] =$$

$$2\sum_{m=1}^{N}\sum_{n=1}^{N}\left[(y_m)^2 - y_m y_n\right] = 2\sum_{m=1}^{N}\left\{N\cdot(y_m)^2 - y_m\sum_{n=1}^{N} y_n\right\} = 2N\sum_{m=1}^{N}\left\{(y_m)^2 - y_m\mu_y\right\}$$



Optimization Criterion: Maximizing Distances

$$2N\sum_{m=1}^{N}\{(y_m)^2 - y_m\mu_y\} = 2N\left(\sum_{m=1}^{N}\{(y_m)^2\} - N(\mu_y)^2\right)$$

$$\boxed{\sum_{m=1}^{N} y_m = N\mu_y}$$

$$= 2N\left(\sum_{m=1}^{N}\{(y_m)^2 - (\mu_y)^2\}\right) = 2N(\sigma_y)^2$$

- Note that $\sum_{m=1}^{N}\{(y_m)^2 - (\mu_y)^2\}$ is the variance of set $\{y_1, \ldots, y_N\}$.
- There are two equivalent formulas for variance:

$$(\sigma_y)^2 = \sum_{m=1}^{N}\{(y_m)^2 - (\mu_y)^2\} = \sum_{m=1}^{N}\{(y_m - \mu_y)^2\}$$

- Finally,

$$F_{\text{opt}} = \underset{F}{\operatorname{argmax}}\left\{\sum_{m=1}^{N}\sum_{n=1}^{N}\left[\|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2\right]\right\} = \underset{F}{\operatorname{argmax}}\left\{\left(\sigma(\{F(\boldsymbol{x}_n)\})\right)^2\right\}$$

Optimization Criterion

- Therefore, these optimization criteria become equivalent:
  - Finding a projection F that preserves the distances of the original data as well as possible.

$$\underset{F}{\operatorname{argmin}}\left\{\sum_{m=1}^{N}\sum_{n=1}^{N}\left[\|\boldsymbol{x}_m-\boldsymbol{x}_n\|^2-\|F(\boldsymbol{x}_m)-F(\boldsymbol{x}_n)\|^2\right]\right\}$$

  - Finding a projection F that maximizes the sum of pairwise distances of projections $F(\boldsymbol{x}_n)$.

$$\underset{F}{\operatorname{argmax}}\left\{\sum_{m=1}^{N}\sum_{n=1}^{N}\left[\|F(\boldsymbol{x}_m)-F(\boldsymbol{x}_n)\|^2\right]\right\}$$

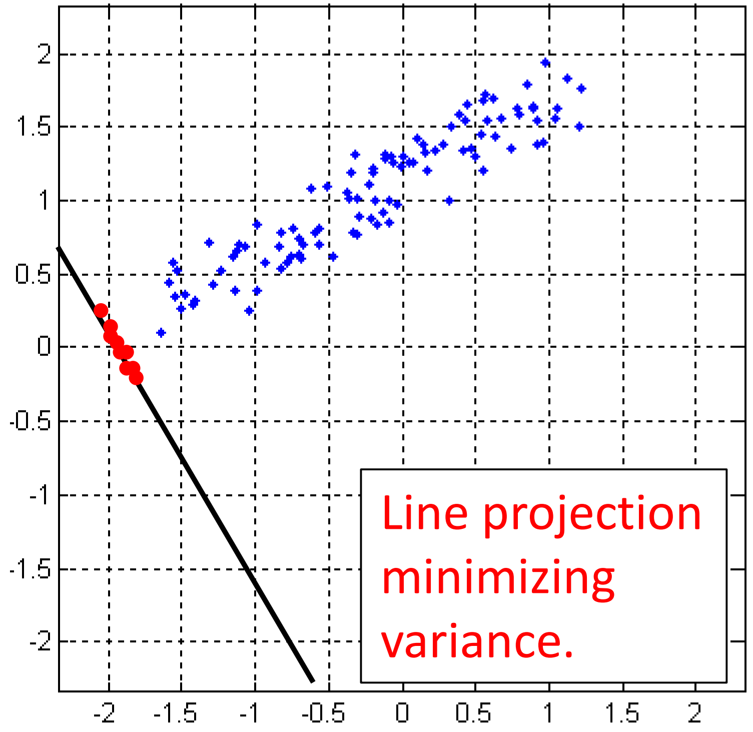
  - Finding a projection F that maximizes the variance of the projections $F(\boldsymbol{x}_n)$.

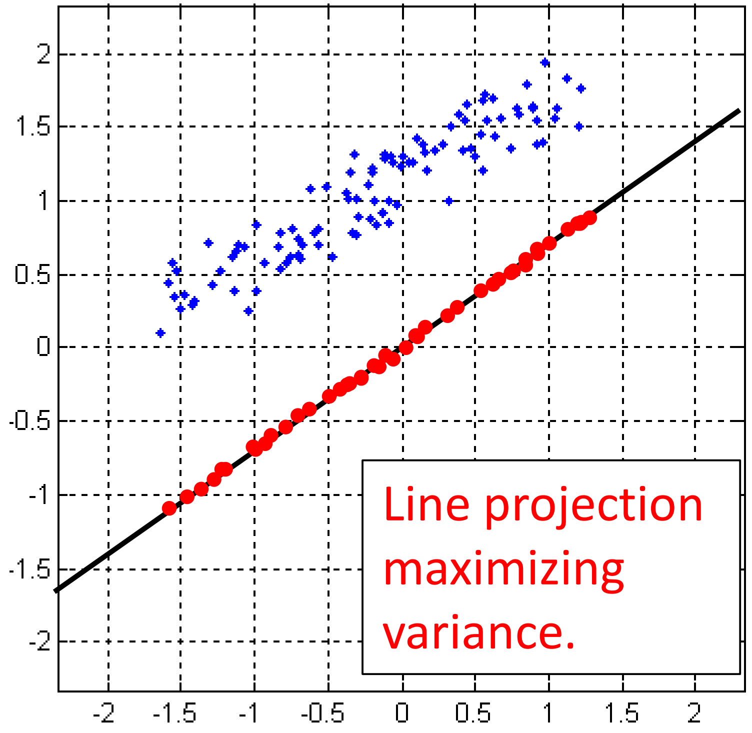
$$\underset{F}{\operatorname{argmax}}\left\{\left(\sigma_y\right)^2\right\}=\underset{F}{\operatorname{argmax}}\left\{\left(\sigma(\{F(\boldsymbol{x}_n)\})\right)^2\right\}$$

Maximizing the Variance

$$F_{\text{opt}} = \underset{F}{\operatorname{argmax}} \left\{ \left(\sigma(\{F(\boldsymbol{x}_n)\}) \right)^2 \right\}$$

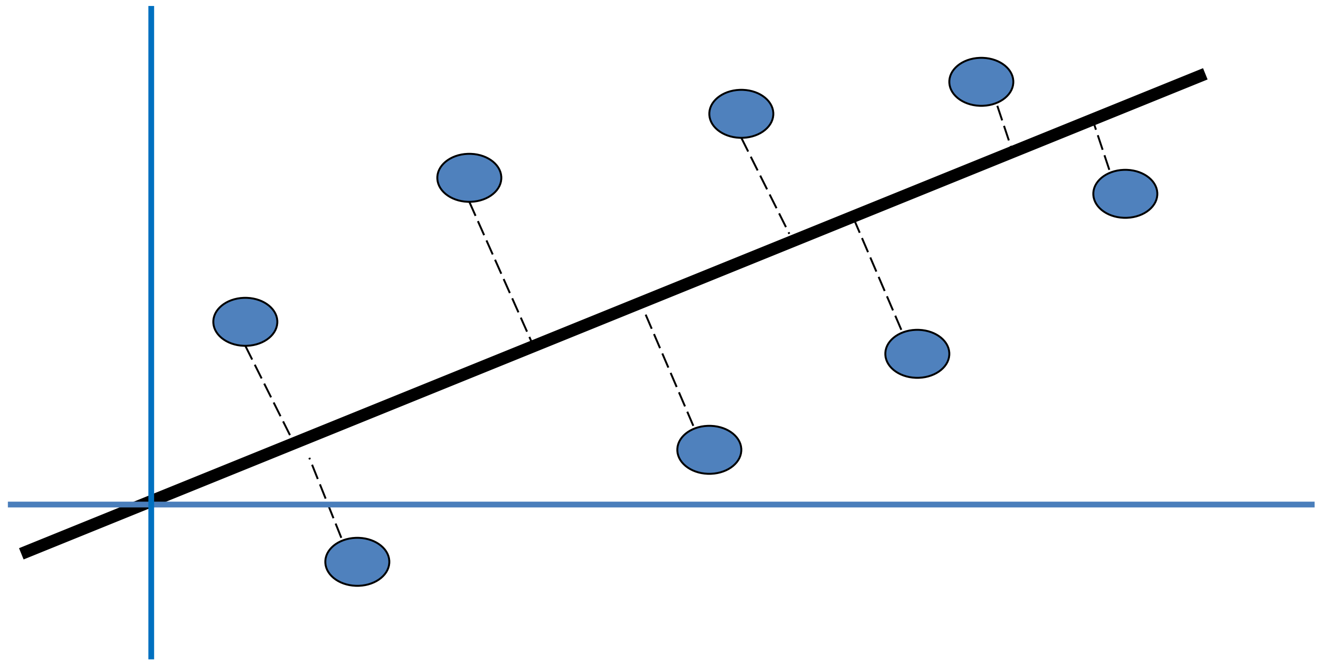
- Intuition for maximizing variance:
  - We want the data to be as spread out as possible.
  - A projection that squeezes the data loses more information (figure on left).





Another Interpretation: Maximizing the Variance

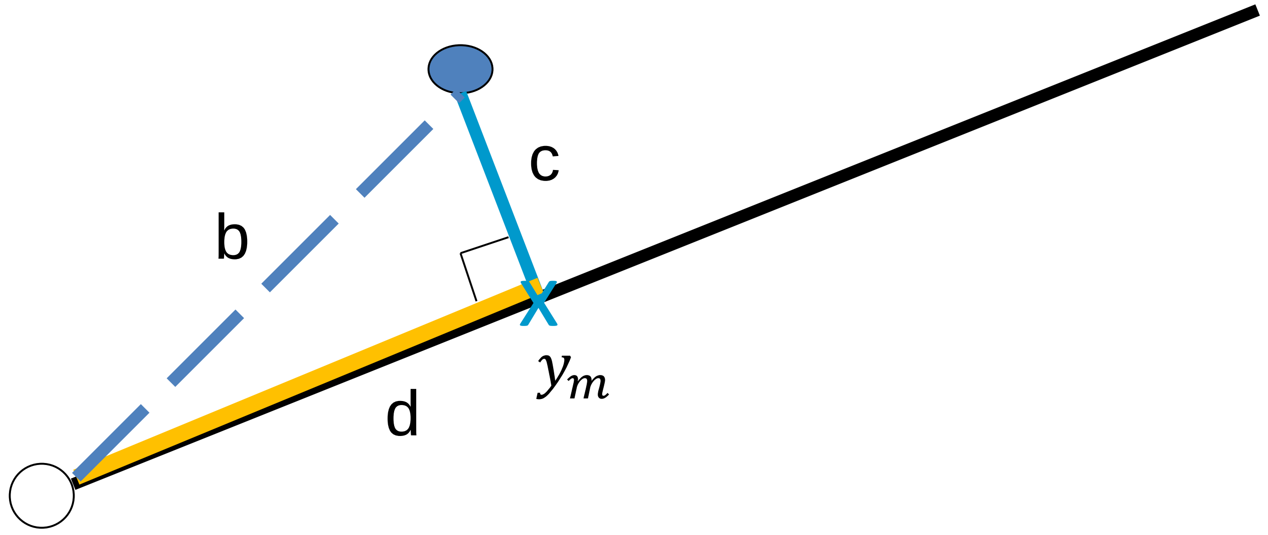
- Formally, minimize sum of squares of distances to the line.



- Why sum of squares? Because it allows fast minimization, assuming the line passes through 0

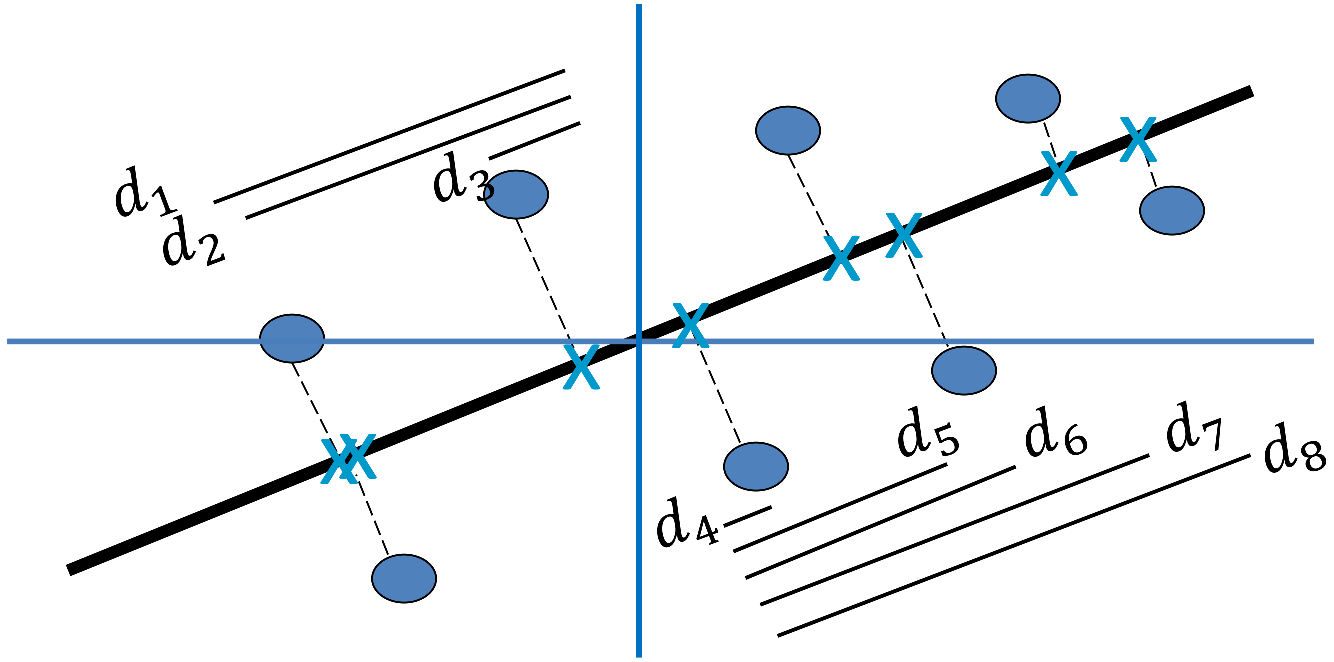
Another Interpretation: Maximizing the Variance

- Minimizing sum of squares of distances (c) to the line is the same as maximizing the sum of squares of the projections on that line, thanks to Pythagoras.
  - $b^2 = c^2 + d^2$
- Since b is fixed, minimizing c is equivalent to maximizing d



Another Interpretation: Maximizing the Variance

- Therefore, minimizing $d_1^2 + d_2^2 + \cdots + d_7^2 + d_8^2$ is same as minimizing the variance of projected data (in this figure, the variance of "**X**" data



Maximizing the Variance

- Goal: Finding a projection F that maximizes the variance of the projections $F(\boldsymbol{x}_n)$

$$F_{\text{opt}} = \underset{F}{\operatorname{argmax}} \left\{ \left(\sigma(\{F(\boldsymbol{x}_n)\}) \right)^2 \right\}$$

- Line projection F can be defined as the dot product with some unit vector $\boldsymbol{u}_1$:

$$F(\boldsymbol{x}_n) = (\boldsymbol{u}_1)^T \boldsymbol{x}_n$$

  - Projection of vector $\boldsymbol{a}$ on vector $\boldsymbol{b}$ = $\frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{b}|}$ (if $\boldsymbol{b}$ is a unit vector, $|\boldsymbol{b}| = 1$)

- Let $\bar{x}$ be the mean of the original data $\{\boldsymbol{x}_n\}$: $\overline{\boldsymbol{x}} = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}_n$.
- The mean of the projected data is the projection of the mean.

$$\frac{1}{N} \sum_{n=1}^{N} \{(\boldsymbol{u}_1)^T \boldsymbol{x}_n\} = \frac{1}{N} (\boldsymbol{u}_1)^T \sum_{n=1}^{N} \{\boldsymbol{x}_n\} = \frac{1}{N} (\boldsymbol{u}_1)^T N \overline{\boldsymbol{x}} = (\boldsymbol{u}_1)^T \overline{\boldsymbol{x}}$$

Maximizing the Variance

- The variance of the projected data is:

$$\left(\sigma(\{F(\boldsymbol{x}_n)\})\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{\left[(\boldsymbol{u}_1)^T\boldsymbol{x}_n - (\boldsymbol{u}_1)^T\overline{\boldsymbol{x}}\right]^2\right\}$$

- Let $\boldsymbol{S}$ be the covariance matrix of the original data:

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}\left\{(\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^T\right\}$$

Maximizing the Variance

- Then, it turns out that the variance of the projected data is:

$$\left(\sigma(\{F(\boldsymbol{x}_n)\})\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{\left[(\boldsymbol{u}_1)^T\boldsymbol{x}_n - (\boldsymbol{u}_1)^T\overline{\boldsymbol{x}}\right]^2\right\} = \frac{1}{N}\sum_{n=1}^{N}\left\{\left[(\boldsymbol{u}_1)^T(\boldsymbol{x}_n - \overline{\boldsymbol{x}})\right]^2\right\}$$

$$= \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{u}_1)^T(\boldsymbol{x}_n - \overline{\boldsymbol{x}})\left((\boldsymbol{u}_1)^T(\boldsymbol{x}_n - \overline{\boldsymbol{x}})\right)^{\boldsymbol{T}} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{u}_1)^T(\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^{\boldsymbol{T}}\boldsymbol{u}_1$$

$$= (\boldsymbol{u}_1)^T\frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \overline{\boldsymbol{x}})(\boldsymbol{x}_n - \overline{\boldsymbol{x}})^{\boldsymbol{T}}\boldsymbol{u}_1 = (\boldsymbol{u}_1)^T\boldsymbol{S}\boldsymbol{u}_1$$

Maximizing the Variance

- The variance of the projected data is: $(\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1$.
- We want to maximize the variance.
- We also have the constraint that $\boldsymbol{u}_1$ should be a unit vector: $(\boldsymbol{u}_1)^T \boldsymbol{u}_1 = 1$.
  - Otherwise, to maximize the variance we can just make $\boldsymbol{u}_1$ arbitrarily large.
- So, we have a constrained maximization problem.

$$\text{maximize } (\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1 \text{ subject to } (\boldsymbol{u}_1)^T \boldsymbol{u}_1 = 1$$

- As we did when discussing support vector machines, we can use **Lagrange multipliers**.
- The Lagrangian (with λ_1 as a Lagrange multiplier) is:

$$(\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1 + \lambda_1(1 - (\boldsymbol{u}_1)^T \boldsymbol{u}_1)$$

Maximizing the Variance

- The Lagrangian (with λ_1 as a Lagrange multiplier) is:

$$(\boldsymbol{u}_1)^T\boldsymbol{S}\boldsymbol{u}_1 + \lambda_1(1 - (\boldsymbol{u}_1)^T\boldsymbol{u}_1)$$

- To maximize the Lagrangian, we must set its gradient to zero.
- The gradient of the Lagrangian with respect to $\boldsymbol{u}_1$ is:

$$2\boldsymbol{S}\boldsymbol{u}_1 - 2\lambda_1\boldsymbol{u}_1$$

$$\frac{\partial(X^TAX)}{\partial X} = 2X^TA$$

- Setting the gradient to 0, we get: $\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1\boldsymbol{u}_1$.
- This means that, if $\boldsymbol{u}_1$ and λ_1 are solutions, then:
  - $\boldsymbol{u}_1$ is an **eigenvector** of $\boldsymbol{S}$.
  - λ_1 is an **eigenvalue** of $\boldsymbol{S}$.

Eigenvectors and Eigenvalues

- It is time for a quick review of eigenvectors and eigenvalues.
- Let $\boldsymbol{A}$ be a $D \times D$ square matrix.
- An **eigenvector** of $\boldsymbol{A}$ is defined to be any D-dimensional column vector $\boldsymbol{x}$, for which a real number λ exists such that:
- $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$

- If the above condition is satisfied for some eigenvector $\boldsymbol{x}$, then real number λ is called an **eigenvalue** of $\boldsymbol{A}$.
- In our case, we have found that $\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1\boldsymbol{u}_1$.
- Therefore, to maximize the variance, $\boldsymbol{u}_1$ has to be an eigenvector of $\boldsymbol{S}$, and λ_1 has to be the corresponding eigenvalue.

Maximizing the Variance

- $\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$.
- Therefore, to maximize the variance, $\boldsymbol{u}_1$ has to be an eigenvector of $\boldsymbol{S}$, and λ_1 has to be the corresponding eigenvalue.
- However, if $\boldsymbol{S}$ is a $D \times D$ matrix, it can have up to D distinct eigenvectors, and up to D distinct eigenvalues.
- Which one of those eigenvectors should we pick?

- As we saw a few slides earlier, $(\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1$ is the actual variance of the projected data.
- We get:

$$(\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1 = (\boldsymbol{u}_1)^T \lambda_1 \boldsymbol{u}_1$$
$$\Rightarrow (\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 (\boldsymbol{u}_1)^T \boldsymbol{u}_1$$
$$\Rightarrow \ (\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1 = \lambda_1$$

Maximizing the Variance

- Since $(\boldsymbol{u}_1)^T\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1$,
- to maximize the variance, we should choose $\boldsymbol{u}_1$ to be the eigenvector of $\boldsymbol{S}$ that has the largest eigenvalue.
- This eigenvector $\boldsymbol{u}_1$ is called the **principal component** of the data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$.

Finding the Eigenvector with the Largest Eigenvalue

- Finding the eigenvector with the largest eigenvalue is a general problem, and there are several computational solutions.
- Let A (covariance matrix) be a square nXn matrix and X be a non-zero vector for which

$$AX = \lambda X$$

for some scalar values λ. then λ is known as the eigenvalue of matrix A and X is known as the eigenvector of matrix A for the corresponding eigenvalue.

- How to calculate **x** and λ:

$$AX - \lambda X = 0$$

$$(A - \lambda I)X = 0$$

where I am the identity matrix of the same shape as matrix A.

1) Calculate *det*(*A*-λ*I*), yields a polynomial (degree n)
2) Determine roots to *det*(*A*-λ*I*)=0, roots are eigenvalues λ
3) Solve (*A*- λ*I*) **x**=0 for each λ to obtain eigenvectors **x**

Finding the Eigenvector with the Largest Eigenvalue

- Another method for solving this problem is the **power method**.
  - It is not the best, algorithmically, but it is pretty simple to implement.
- The power method takes as input a **<u>square</u>** matrix $\boldsymbol{A}$.
  - For PCA, $\boldsymbol{A}$ is a covariance matrix.
- Define a vector $\boldsymbol{b}_0$ to be a random D-dimensional vector.
- Define the following recurrence: $\boldsymbol{b}_{k+1} = \frac{\boldsymbol{A}\boldsymbol{b}_k}{\|\boldsymbol{A}\boldsymbol{b}_k\|}$.
- Then, the sequence $(\boldsymbol{b}_k)$ converges to the eigenvector of $\boldsymbol{A}$ with the largest eigenvalue.

- We can also use SVD(Singular Value Decomposition) method as well.

Example

- Compute the PCA of the following dataset:

(1,2),(3,3),(3,5),(5,4),(5,6),(6,5),(8,7),(9,8)

- Compute the sample covariance matrix:

$$A = \begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix}$$

- The eigenvalues can be computed by finding the roots of the polynomial:

$$Av = \lambda v$$
$$|A - \lambda I| = 0$$
$$\begin{bmatrix} 6.25 - \lambda & 4.25 \\ 4.25 & 3.5 - \lambda \end{bmatrix} = 0$$
$$\lambda_1 = 9.34; \lambda_2 = 0.41$$

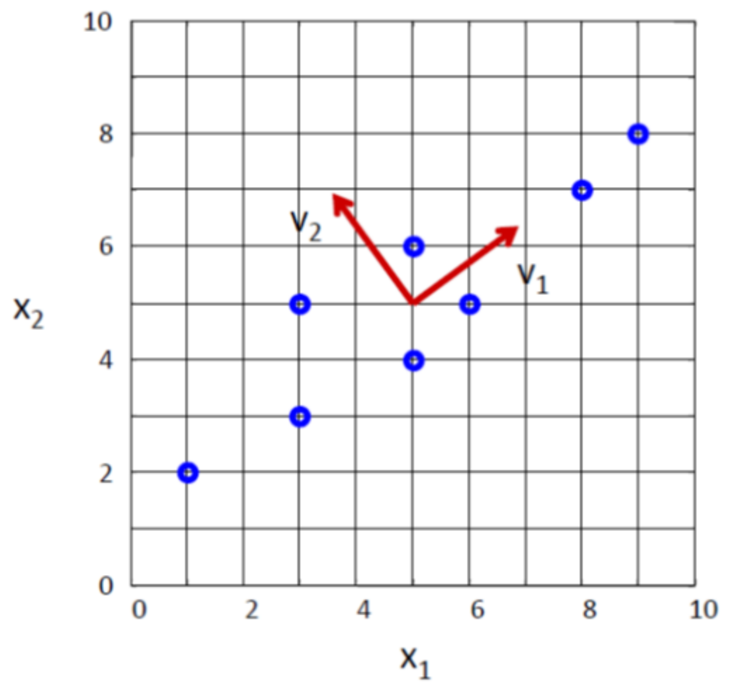
Example (cont'd)

- The eigenvectors are the solutions of:

$$Av_i = \lambda_i v_i$$

$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} \lambda_1 v_{11} \\ \lambda_1 v_{12} \end{bmatrix} \Rightarrow \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.59 \end{bmatrix}$$

$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} \lambda_2 v_{21} \\ \lambda_2 v_{22} \end{bmatrix} \Rightarrow \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} -0.59 \\ 0.81 \end{bmatrix}$$



- Eigenvectors are typically normalized to have unit-length:

$$\hat{v}_i = \frac{v_i}{\|v_i\|}$$

Computing a 2-Dimensional Projection

- At this point, we have seen how to find the vector $\boldsymbol{u}_1$ so that line projection $(\boldsymbol{u}_1)^T \boldsymbol{x}_n$ has the largest variance.
- What if we want to find a 2-dimensional projection that has the largest variance among all 2-dimensional projections?
- This projection can be decomposed into two linear projections: $(\boldsymbol{u}_1)^T \boldsymbol{x}$ and $(\boldsymbol{u}_2)^T \boldsymbol{x}$.

$$F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \end{bmatrix}$$

- We find $\boldsymbol{u}_1$ as before.

Projection to Orthogonal Subspace

- To find $\boldsymbol{u}_2$:
  - Define $\boldsymbol{x}_{n,2} = \boldsymbol{x}_n - (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1$.
  - Define $\boldsymbol{S}_2$ to be the **covariance** matrix of data $\{\boldsymbol{x}_{1,2}, \boldsymbol{x}_{2,2}, \dots, \boldsymbol{x}_{N,2}\}$.
  - Set $\boldsymbol{u}_2$ to the eigenvector of $\boldsymbol{S}_2$ having the largest eigenvalue ($\boldsymbol{u}_2$ can be computed by applying the power method on $\boldsymbol{S}_2$).
- Why are we doing this?
- More specifically, what is the meaning of:

$$\boldsymbol{x}_{n,2} = \boldsymbol{x}_n - (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1$$

- In linear algebra terms, $\boldsymbol{x}_{n,2}$ is the projection of $\boldsymbol{x}_n$ to the **subspace that is orthogonal** to vector $\boldsymbol{u}_1$.

Projection to Orthogonal Subspace

- What is the meaning of:

$$\boldsymbol{x}_{n,2} = \boldsymbol{x}_n - (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1$$

- In linear algebra terms, $\boldsymbol{x}_{n,2}$ is the projection of $\boldsymbol{x}_n$ to the subspace that is orthogonal to vector $\boldsymbol{u}_1$.
- The idea is that, given $\boldsymbol{u}_1$, any vector $\boldsymbol{x}_n$ can be decomposed into two parts:

$$\boldsymbol{x}_n = (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1 + \left(\boldsymbol{x}_n - (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1\right)$$

  - The first part (in red) is the projection of $\boldsymbol{x}_n$ to the principal component $\boldsymbol{u}_1$.
  - The second part (in blue) is what remains from $\boldsymbol{x}_n$ after we remove its projection on the principal component.

Projection to Orthogonal Subspace

- Given $\boldsymbol{u}_1$, any vector $\boldsymbol{x}_n$ can be decomposed into two parts:

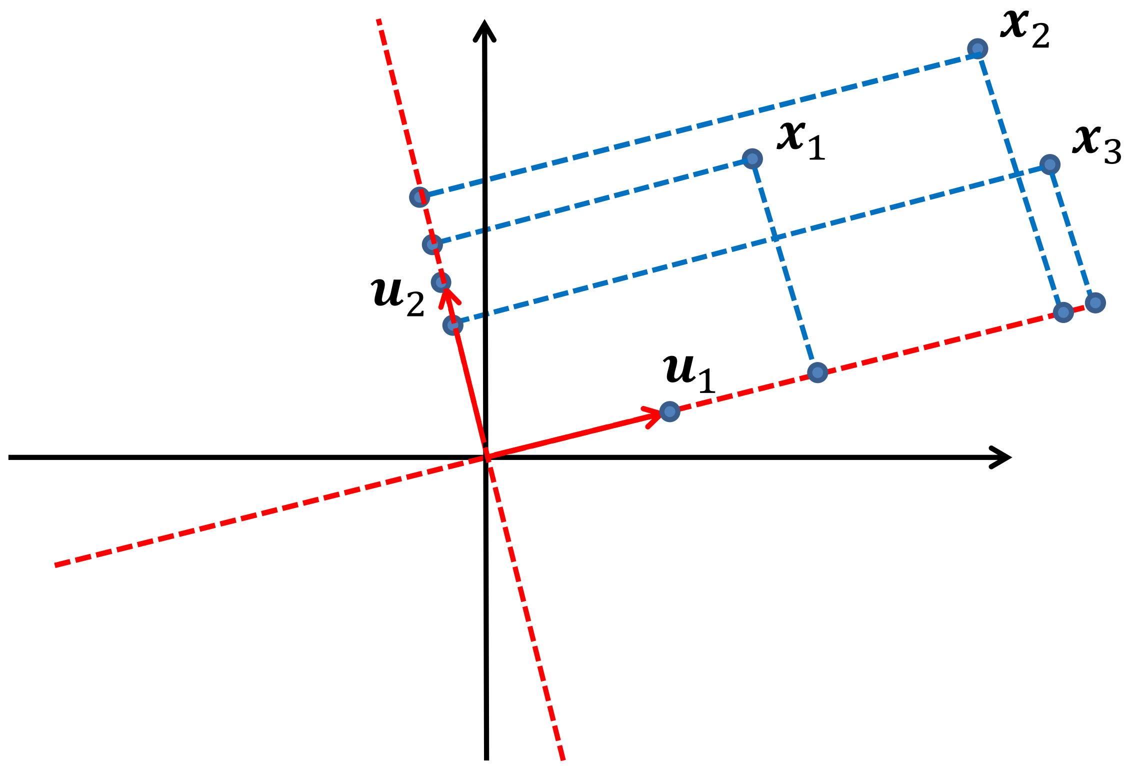
$$\boldsymbol{x}_n = (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1 + \left(\boldsymbol{x}_n - (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1\right)$$

  - The first part (in red) is the projection of $\boldsymbol{x}_n$ to the principal component.
  - The second part (in blue) is what remains from $\boldsymbol{x}_n$ after we remove its projection on the principal component.

- Dataset $\{\boldsymbol{x}_{1,2}, \boldsymbol{x}_{2,2}, \dots, \boldsymbol{x}_{N,2}\}$ is the part of the original data that is **not accounted for** by the projection to the principal component.
- Vector $\boldsymbol{u}_2$ is the principal component of $\{\boldsymbol{x}_{1,2}, \boldsymbol{x}_{2,2}, \dots, \boldsymbol{x}_{N,2}\}$.
  - It is called the **second principal component** of the original data.
- Projection to vector $\boldsymbol{u}_2$ captures as much as possible of the variance that is **not captured** by projection to $\boldsymbol{u}_1$.

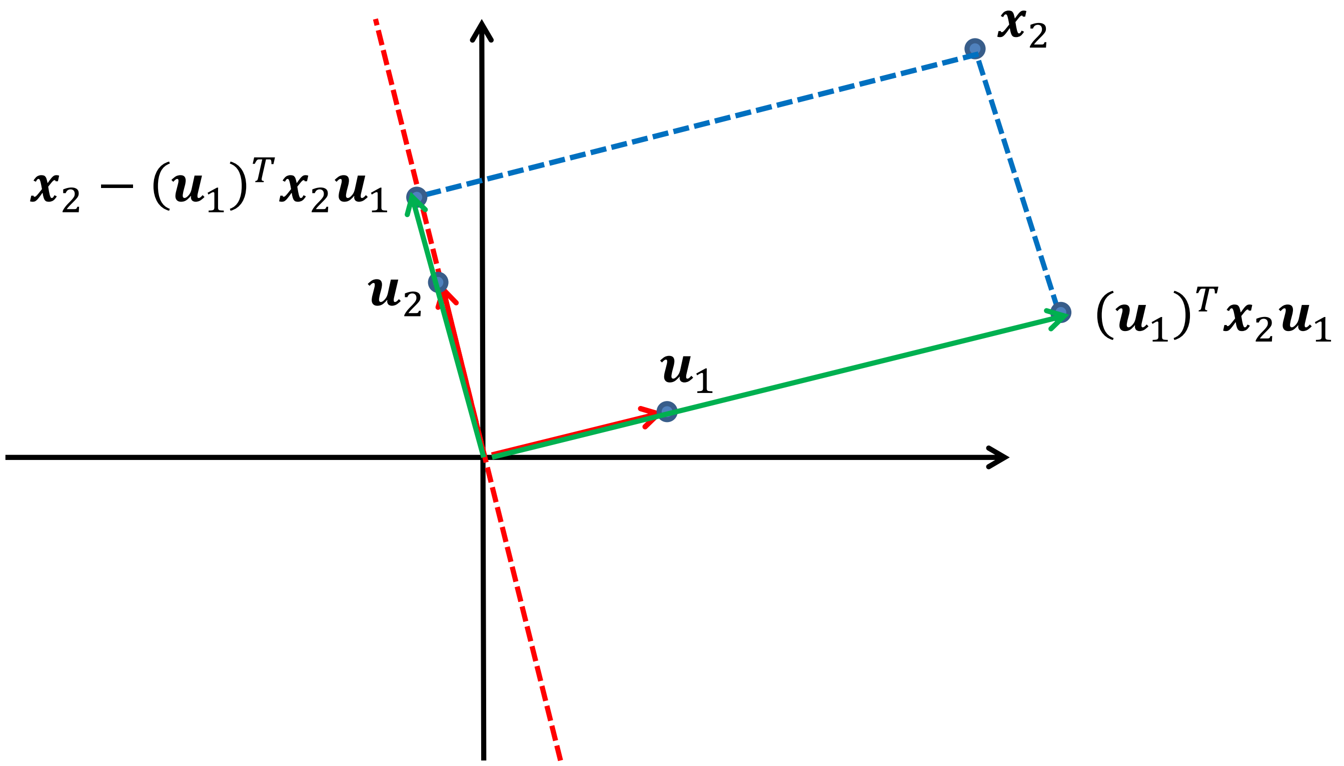
Projection to Orthogonal Subspace

- Consider this picture:
  - We see some data points $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$.
  - We see their projections on principal component $\boldsymbol{u}_1$.
  - We see their projections on the subspace orthogonal to $\boldsymbol{u}_1$.



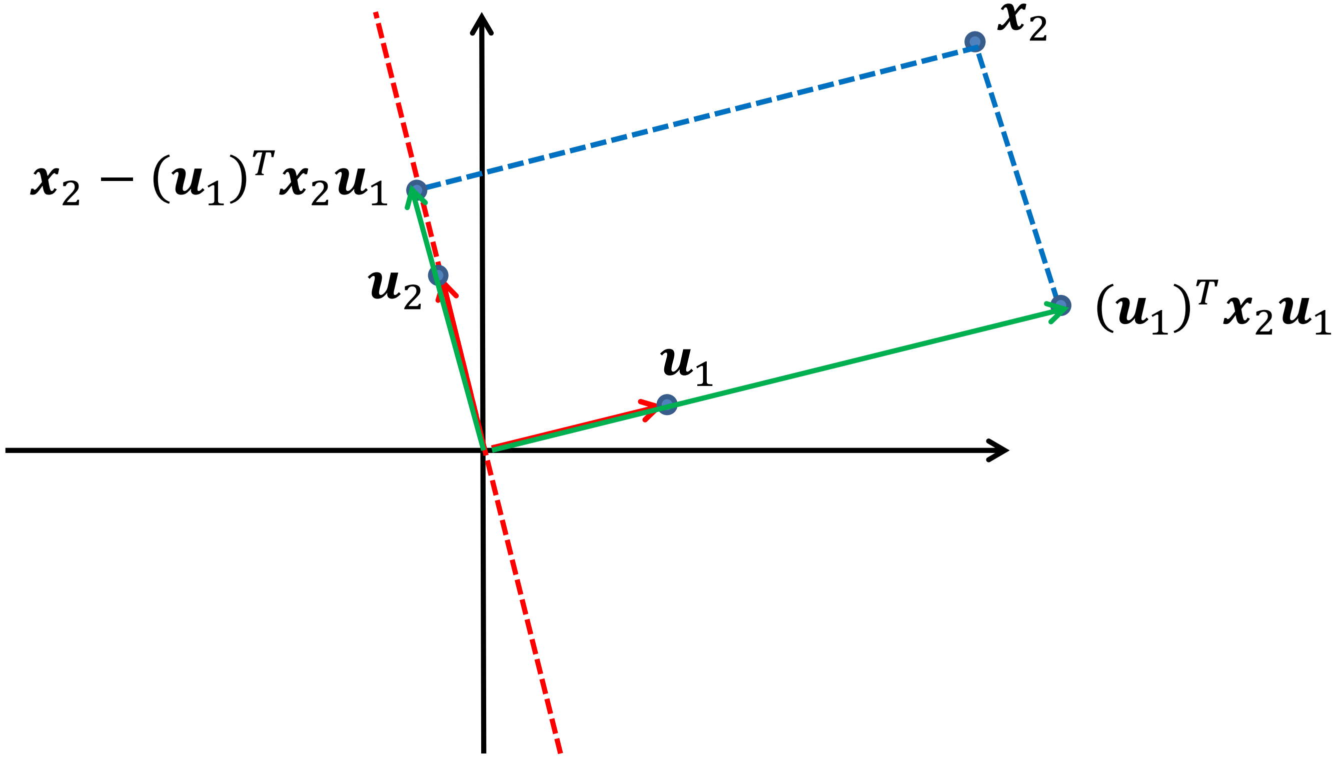
Projection to Orthogonal Subspace

- Consider vector $\boldsymbol{x}_2$.
  - $(\boldsymbol{u}_1)^T \boldsymbol{x}_2$ is just a real number.
  - $(\boldsymbol{u}_1)^T \boldsymbol{x}_2 \boldsymbol{u}_1$ is a vector, pointing in the same direction as $\boldsymbol{u}_1$.
  - $\boldsymbol{x}_2 - (\boldsymbol{u}_1)^T \boldsymbol{x}_2 \boldsymbol{u}_1$ is a vector orthogonal to $\boldsymbol{u}_1$.


$\boldsymbol{x}_2$
$\boldsymbol{x}_2 - (\boldsymbol{u}_1)^T \boldsymbol{x}_2 \boldsymbol{u}_1$
$\boldsymbol{u}_2$
$(\boldsymbol{u}_1)^T \boldsymbol{x}_2 \boldsymbol{u}_1$
$\boldsymbol{u}_1$

Projection to Orthogonal Subspace

- The second principal component $\boldsymbol{u}_2$ belongs to the subspace orthogonal to $\boldsymbol{u}_1$.
- Therefore, $\boldsymbol{u}_2$ is always orthogonal to $\boldsymbol{u}_1$.


$\boldsymbol{x}_2$
$\boldsymbol{x}_2 - (\boldsymbol{u}_1)^T \boldsymbol{x}_2 \boldsymbol{u}_1$
$\boldsymbol{u}_2$
$(\boldsymbol{u}_1)^T \boldsymbol{x}_2 \boldsymbol{u}_1$
$\boldsymbol{u}_1$

Computing an M-Dimensional Projection

- This process can be extended to compute M-dimensional projections, for any value of M.
- Here is the pseudocode:

```
// Initialization:
For $n$ = 1 to $N$, define $\boldsymbol{x}_{n,1} = \boldsymbol{x}_n$.

// Main loop:
For $d$ = 1 to $M$:
    – Define $\boldsymbol{S}_d$ to be the covariance matrix of data $\{\boldsymbol{x}_{1,d}, \boldsymbol{x}_{2,d}, \dots, \boldsymbol{x}_{N,d}\}$.
    – Set $\boldsymbol{u}_d$ to the eigenvector of $\boldsymbol{S}_d$ having the largest eigenvalue ($\boldsymbol{u}_d$ can be computed by applying
      the power method on $\boldsymbol{S}_d$).
    – For $n$ = 1 to $N$, define $\boldsymbol{x}_{n,d+1} = \boldsymbol{x}_{n,d} - (\boldsymbol{u}_d)^T \boldsymbol{x}_{n,d} \boldsymbol{u}_d$.
```

Computing an M-Dimensional Projection

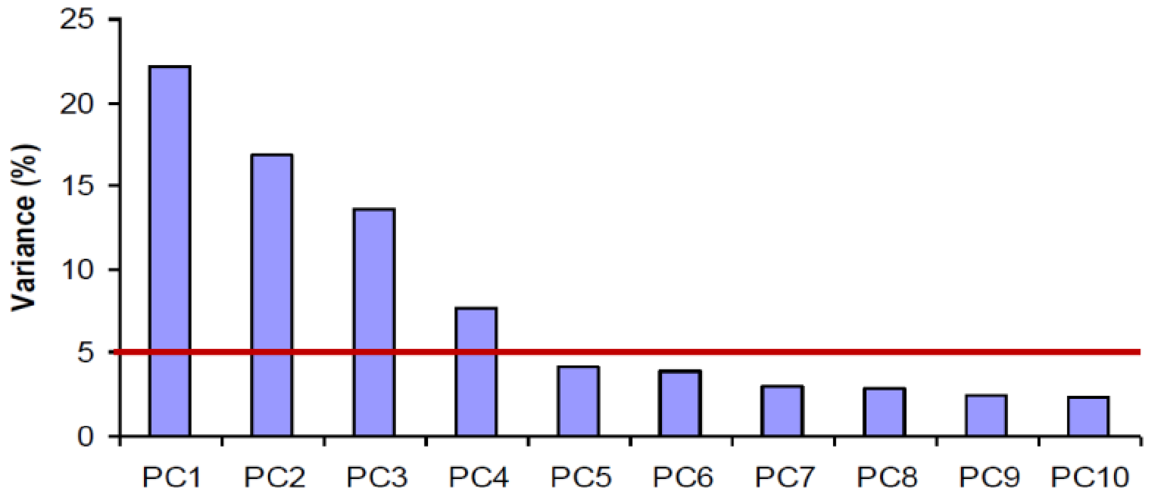
- The pseudocode we just saw computes vectors $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, …, $\boldsymbol{u}_M$.
- Then, the projection function F is defined as: $F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \\ \dots \\ (\boldsymbol{u}_M)^T \boldsymbol{x} \end{bmatrix}$
- Alternatively, we define an $M \times D$ **projection matrix** $\boldsymbol{U} = \begin{bmatrix} (\boldsymbol{u}_1)^T \\ (\boldsymbol{u}_2)^T \\ \dots \\ (\boldsymbol{u}_M)^T \end{bmatrix}$.
- Then, the projection F is defined as: $F(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{x}$.

Eigenvectors

- Vectors $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, …, $\boldsymbol{u}_M$ are the eigenvectors of covariance matrix $\boldsymbol{S}$ with the M largest eigenvalues.
- These vectors are also called the M principal components of the data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$.
- We saw before that $\boldsymbol{u}_2$ is orthogonal to $\boldsymbol{u}_1$.
- With the exact same reasoning, we can show that each $\boldsymbol{u}_i$ is orthogonal to all the other eigenvectors.
- Vectors $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, …, $\boldsymbol{u}_M$ form what is called an **orthonormal basis**, for the M-dimensional subspace that is the target space of F.
- An orthonormal basis for a vector space is a basis where:
  - Each basis vector is a unit vector.
  - Each pair of basis vectors are orthogonal to each other.

Principal Component Analysis

- Vectors $\boldsymbol{u}_1$, $\boldsymbol{u}_2$, …, $\boldsymbol{u}_M$ are the eigenvectors of covariance matrix $\boldsymbol{S}$ with the M largest eigenvalues.
- These vectors are also called the M principal components of the data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$
- Eigenvalues λ_j corresponds to variance on each component *j*
- In choosing M, our goal is to strike a balance, where, ideally:
  - M is small enough so that $F(\boldsymbol{x})$ is as low-dimensional as possible, while…
  - $F(\boldsymbol{x})$ captures almost all the information available in the original data.



Data Normalization

- The principal components depend both on the units used to measure the original variables (i.e., features) and the range of values they assume.
- If different units and/or ranges are involved, each feature x_i (e.g., length) should always be normalized before applying PCA.
- A common normalization method is to transform all the features to have zero mean and unit standard deviation:

$$\frac{x_{ij} - \mu_i}{\sigma_i}$$

where μ_i and σ_i are the mean and standard deviation of the i-th feature x_i

Variations and Alternatives to PCA

- There exist several alternatives for dimensionality reduction.
- Variations of PCA:
  - Kernel PCA (later in Kernel Methods chapter)
  - Probabilistic PCA (we will not discuss).

- Alternatives to PCA:
  - Autoencoders.
  - Isomap (we will not discuss).

PCA in sklearn

```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X, y = load_wine(return_X_y=True, as_frame=True)
scaler = StandardScaler().set_output(transform="pandas")
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
scaled_X_train = scaler.fit_transform(X_train)

pca = PCA(n_components=2).fit(X_train)
scaled_pca = PCA(n_components=2).fit(scaled_X_train)
X_train_transformed = pca.transform(X_train)
X_train_std_transformed = scaled_pca.transform(scaled_X_train)

first_pca_component = pd.DataFrame(pca.components_[0], index=X.columns, columns=["without scaling"])
first_pca_component["with scaling"] = scaled_pca.components_[0]
first_pca_component.plot.bar(title="Weights of the first principal component", figsize=(6, 8))

_ = plt.tight_layout()
```

Autoencoders

- Unlike the PCA now we can use activation functions to achieve non-linearity.
- It has been shown that an AE(Autoencoder) with linear activation functions achieves the PCA capacity.
- The autoencoder idea was a part of NN history for decades
- Traditionally an autoencoder is used for dimensionality reduction and feature learning.

- 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.

- We can simply train the model as any other Neural Network using gradient descent.

Autoencoders

- Given data x we would like to learn the functions f (encoder) and g (decoder) where

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

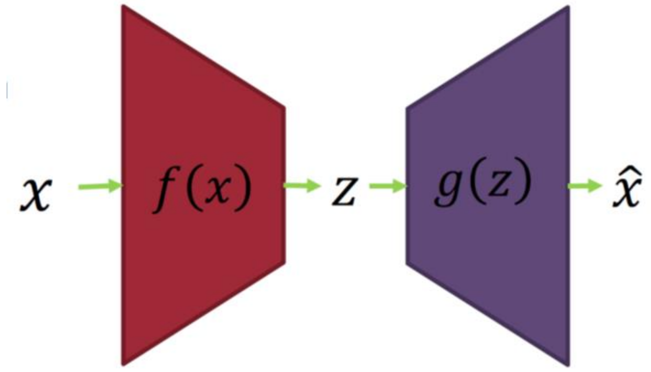
and

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

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

where h is an approximation of the identity function

- z is some latent representation and s is an activation function (usually non-linear function such as sigmoid)
- $\hat{x}$ is x's reconstruction



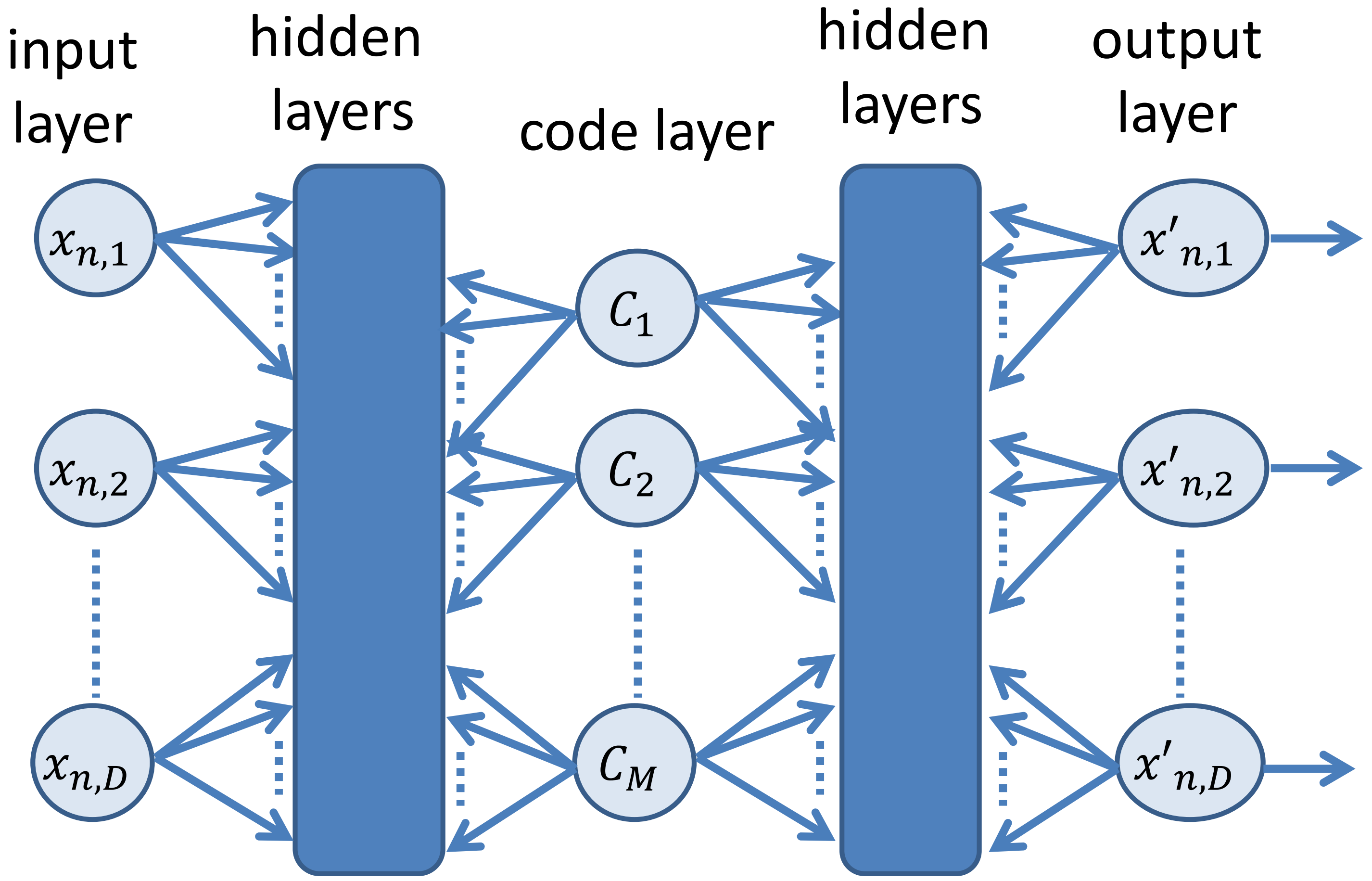
Autoencoders

- Try to make the output be the same as the input in a network with a central bottleneck.
- The activities of the hidden units in the bottleneck form an efficient code.
- If the hidden and output layers are linear, it will learn hidden units that are a linear function of the data and minimize the squared reconstruction error.
  - This is exactly what PCA does.
- The M hidden units will span the same space as the first M components found by PCA
  - Their weight vectors may not be orthogonal.
  - They will tend to have equal variances.
- With non-linear layers before and after the code, it should be possible to efficiently represent data that lies on or near a non-linear manifold.

Autoencoders

- An autoencoder is a neural network, can be trained with backpropagation or other methods.
- The target output for input $\boldsymbol{x}_n$ is $\boldsymbol{x}_n$.
- One of the layers is the **code layer**, with M units, where typically $M \ll \mathrm{D}$.

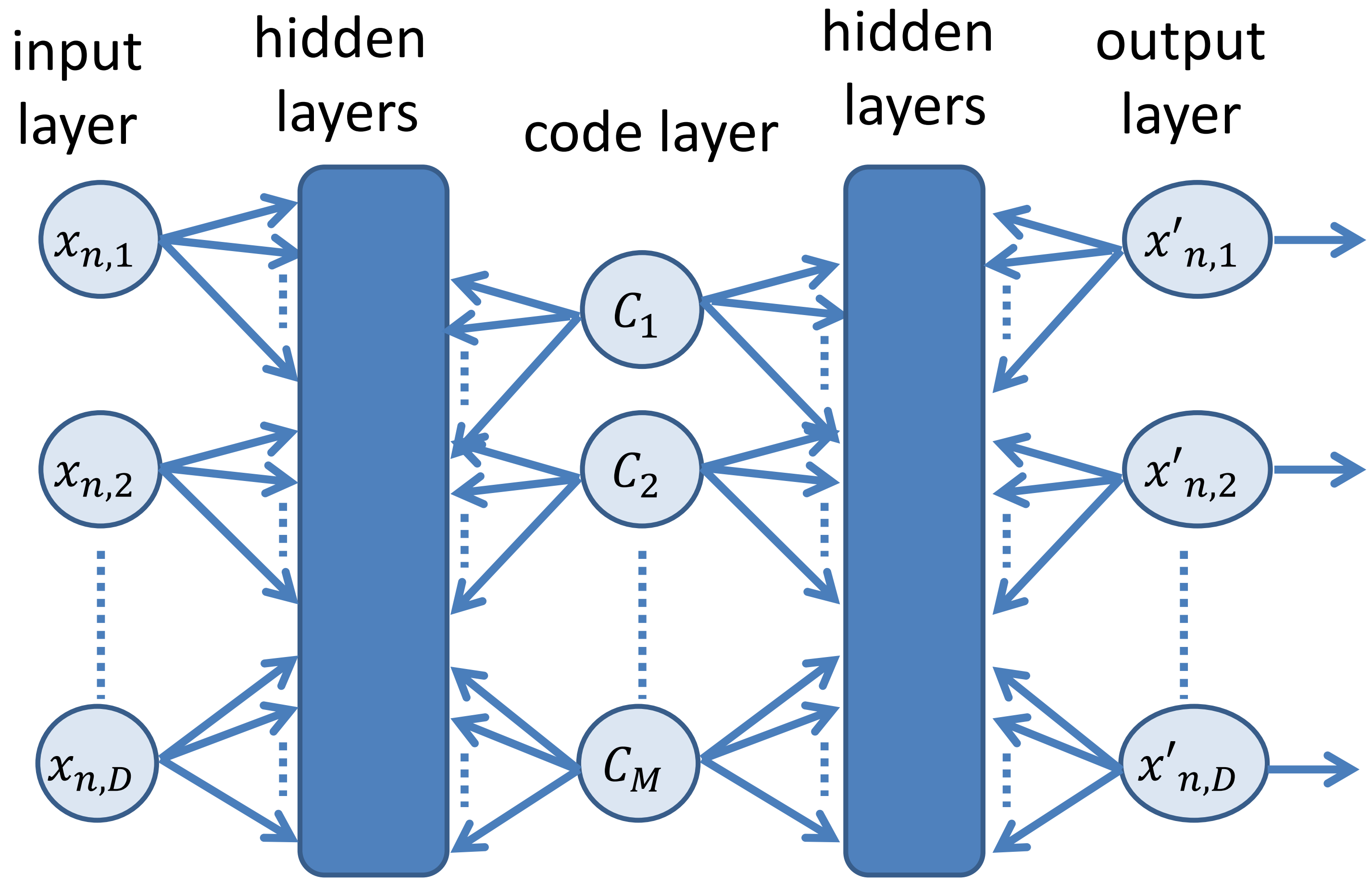
- A trained autoencoder is used to define a projection $F(\boldsymbol{x})$, mapping $\boldsymbol{x}$ to the activations (sums of weighted inputs) of the code layer units.
- F maps D-dimensional vectors to M-dimensional vectors.



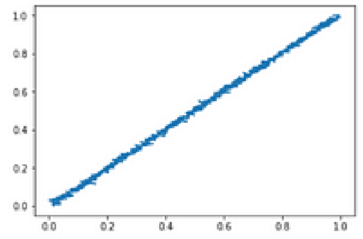
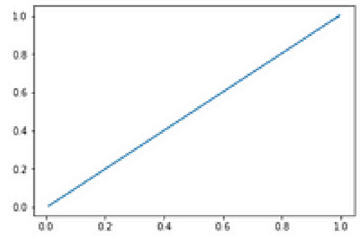
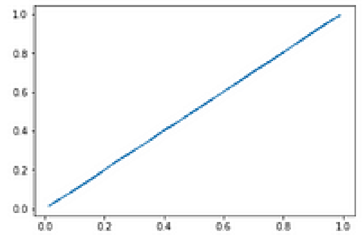
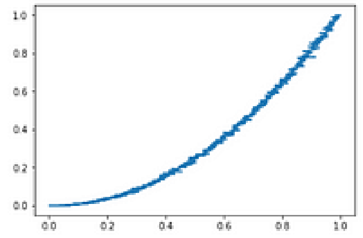
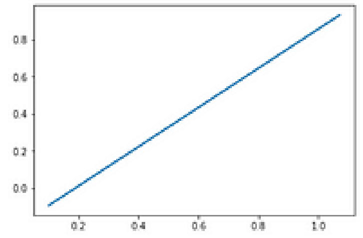
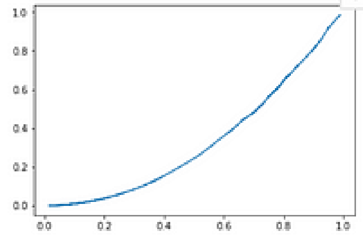
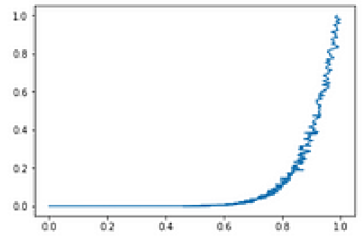
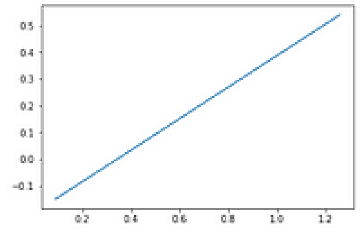
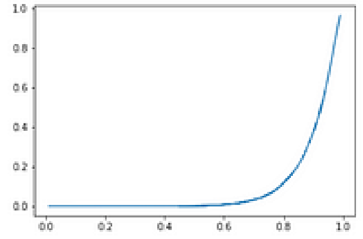
Autoencoders

- A trained autoencoder also defines a backprojection $B(\boldsymbol{z})$, mapping the activations of the code layers to the output of the output units.
- B maps M-dimensional vectors to D-dimensional vectors.

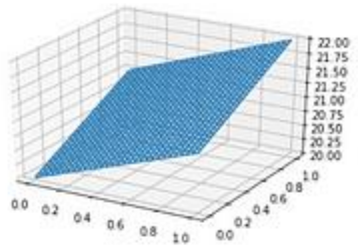
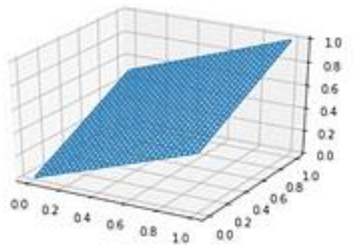
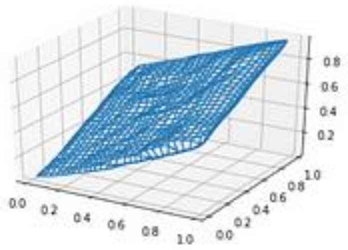
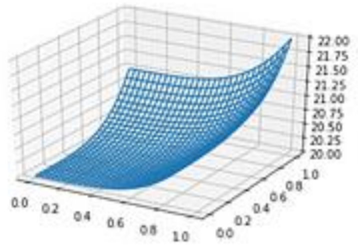
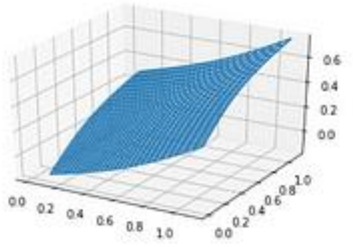
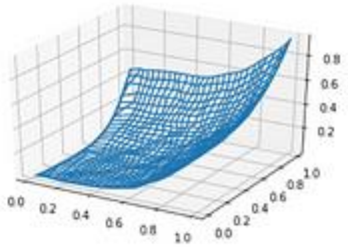
- The whole network computes the composition $F(B(\boldsymbol{x}))$.
- F is typically a nonlinear projection



PCA vs Autoencoders

Function	Feature Space	PCA Reconstruction	Auto Encoder Reconstruction
$y=mx+c$			
$y=mx^2+c$			
$y=mx^8+c$			

PCA vs Autoencoders

Function	Feature Space	PCA Reconstruction	Auto Encoder Reconstruction
Plane			
Curved Surface			

Feature Selection Using Autoencoders – 1/3

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import keras
from keras.models import Sequential
from keras.layers import Input, Dense

# load the Iris dataset
iris = load_iris()
data = iris.data
target = iris.target

# Standardize the data
scaler = StandardScaler()
data = scaler.fit_transform(data)
```

Feature Selection Using Autoencoders – 2/3

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.2, random_state=42)

# Define the autoencoder architecture
input_dim = X_train.shape[1]
encoding_dim = 2

# Set the encoding dimension
input_layer = keras.layers.Input(shape=(input_dim,))
encoder = keras.layers.Dense(encoding_dim, activation="relu")(input_layer)
decoder = keras.layers.Dense(input_dim, activation="sigmoid")(encoder)
autoencoder = keras.Model(inputs=input_layer, outputs=decoder)

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mse')

autoencoder.summary() # Summary of the autoencoder architecture

# Train the autoencoder
autoencoder.fit(X_train, X_train, epochs=100, batch_size=32, shuffle=True, validation_data=(X_test, X_test))
```

Feature Selection Using Autoencoders – 3/3

```
# Use encoder part of the autoencoder for feature selection
encoder = keras.Model(inputs=autoencoder.input, outputs=autoencoder.layers[1].output)
encoded_features_train = encoder.predict(X_train)
encoded_features_test = encoder.predict(X_test)

# Display the shape of extracted features
print("Encoded Features Shape (Train):", encoded_features_train.shape)
print("Encoded Features Shape (Test):", encoded_features_test.shape)

# Fit a logistic regression model using the selected features
model = LogisticRegression()
model.fit(encoded_features_train, y_train)

# Make predictions on the test set
y_pred = model.predict(encoded_features_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy with Selected Features:", accuracy)
```