Principal Component Analysis

CSE 4309 – Machine Learning
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

The Curse of Dimensionality

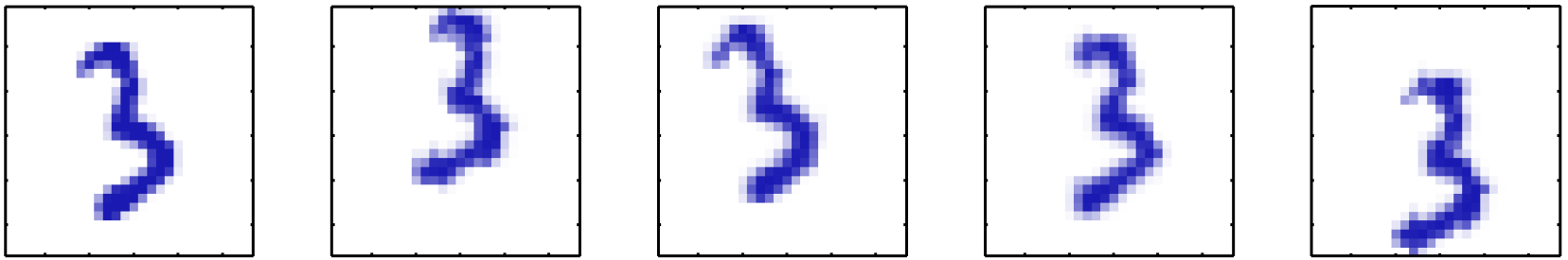
- We have seen a few aspects of this curse.
- As the dimensions increase, learning probability distributions becomes more difficult.
  - More parameters must be estimated.
  - More training data is needed to reliably estimate those parameters.
- For example:
  - To learn multidimensional histograms, 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.

The Curse of Dimensionality

- There are several other aspects of the curse.
- Running time can also be an issue.
- Running backpropagation or decision tree learning on thousands or millions of dimensions requires more time.
  - For backpropagation, running time is at least linear to the number of dimensions.
    - It can be quadratic, if the first hidden layer has as many units as the input layer, and each hidden unit is connected to each input unit.
  - For decision trees, running time is linear to the number of dimensions.
- Storage space is also linear to the number of dimensions.

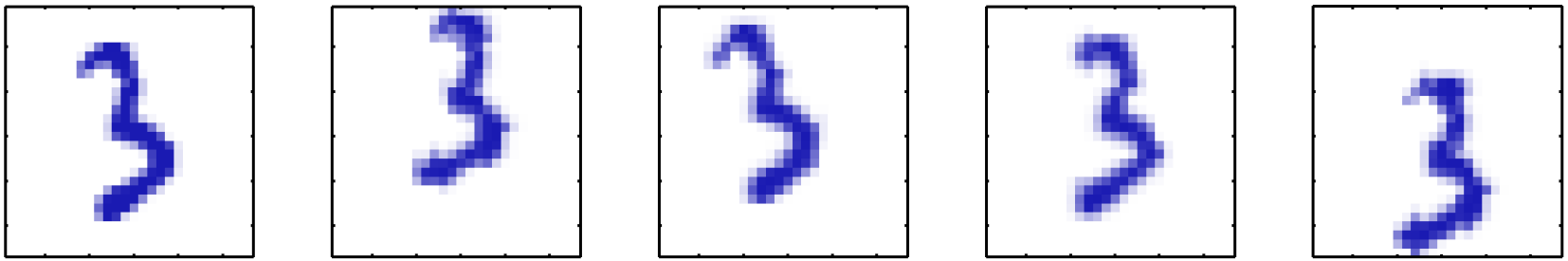
Do We Need That Many Dimensions?

- Consider these five images.
  - Each of them is a 100x100 grayscale image.
  - What is the dimensionality of each image?



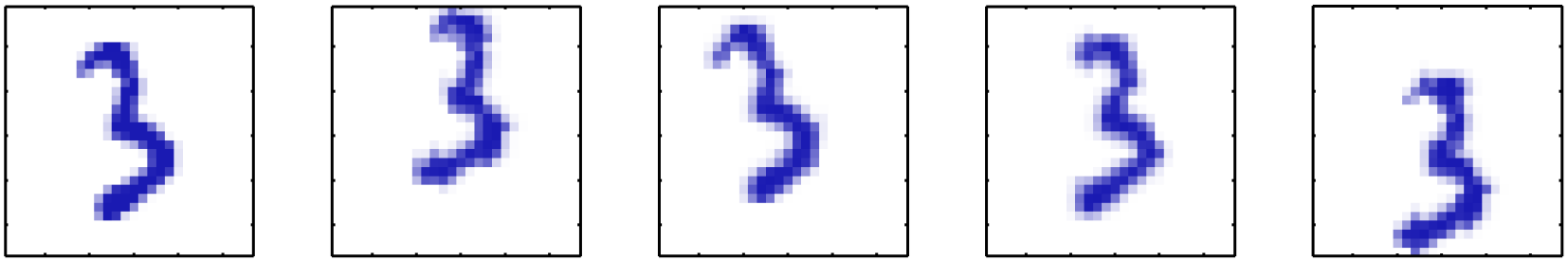
Do We Need That Many Dimensions?

- 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.



Dimensionality Reduction

- The goal of **dimensionality reduction methods** is to build models that allow representing high-dimensional vectors using a smaller number of dimensions.
  - Hopefully, a much smaller number of dimensions.
- The model is built using training data.
- In this example, the model consists of:
  - A **projection** function $\mathrm{F}(\boldsymbol{x})$: $\mathbb{R}^{10000} \rightarrow \mathbb{R}^3$. Given an input image X, F outputs the corresponding translation and rotation parameters t_1, t_2, θ.
  - A **backprojection** function $\mathrm{B}(t_1, t_2, \theta)$: $\mathbb{R}^3 \rightarrow \mathbb{R}^{10000}$. Given translation parameters t_1, t_2, and rotation parameter θ, B outputs the corresponding image.

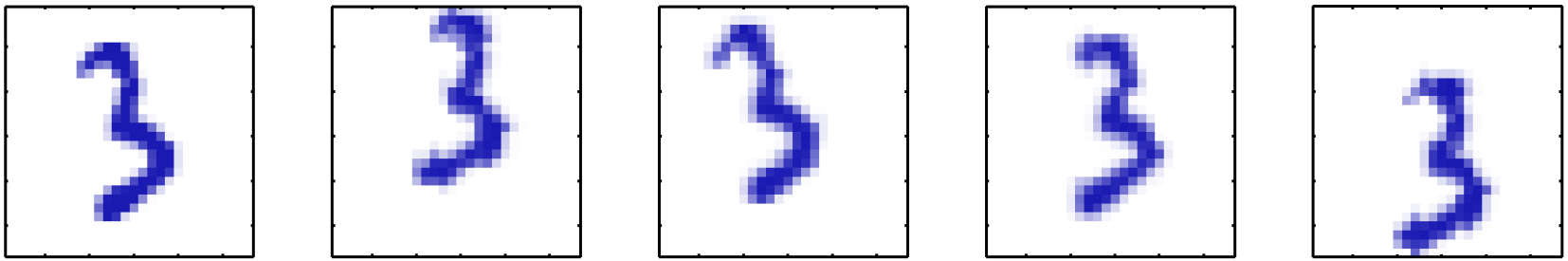


Dimensionality Reduction

- The goal of **dimensionality reduction methods** is to build models that allow representing high-dimensional vectors using a smaller number of dimensions.
  - Hopefully, a much smaller number of dimensions.
- The model is built using training data.
- In this example, the model consists of:
  - A **projection** function $\mathrm{F}(\boldsymbol{x})$: $\mathbb{R}^{10000} \rightarrow \mathbb{R}^3$. Given an input image X, F outputs the corresponding translation and rotation parameters t_1, t_2, θ.
  - A **backprojection** function $\mathrm{B}(t_1, t_2, \theta)$: $\mathbb{R}^3 \rightarrow \mathbb{R}^{10000}$. Given translation parameters t_1, t_2, and rotation parameter θ, B outputs the corresponding image.
- If we have a **lossless projection function**, then $\mathrm{B}\big(\mathrm{F}(\boldsymbol{x})\big) = \boldsymbol{x}$.
- Typically, projection functions are **lossy**.
  - We try to find F and B so that $\mathrm{B}\big(\mathrm{F}(\boldsymbol{x})\big)$ tends to be close to $\boldsymbol{x}$.

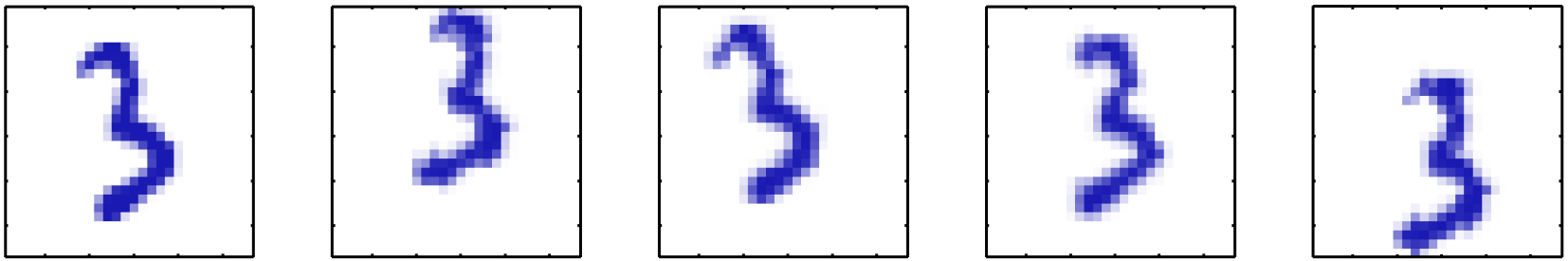
Linear Dimensionality Reduction

- **Linear dimensionality reduction methods** use linear functions for projection and backprojection.
- Projection: $\mathrm{F}(\boldsymbol{x}) = W\boldsymbol{x}$, where:
  - $\boldsymbol{x}$ is a column vector of D dimensions.
  - W is an $M \times D$ matrix, where hopefully $M \ll D$ (M is much smaller than D).
  - This way, $\mathrm{F}(\boldsymbol{x})$ projects D-dimensional vectors to M-dimensional vectors.
- Advantage of linear methods: there are well known methods for finding a good W. We will study some of those methods.
- Disadvantage: they cannot capture non-linear transformations, such as the image translations and rotations of our example.



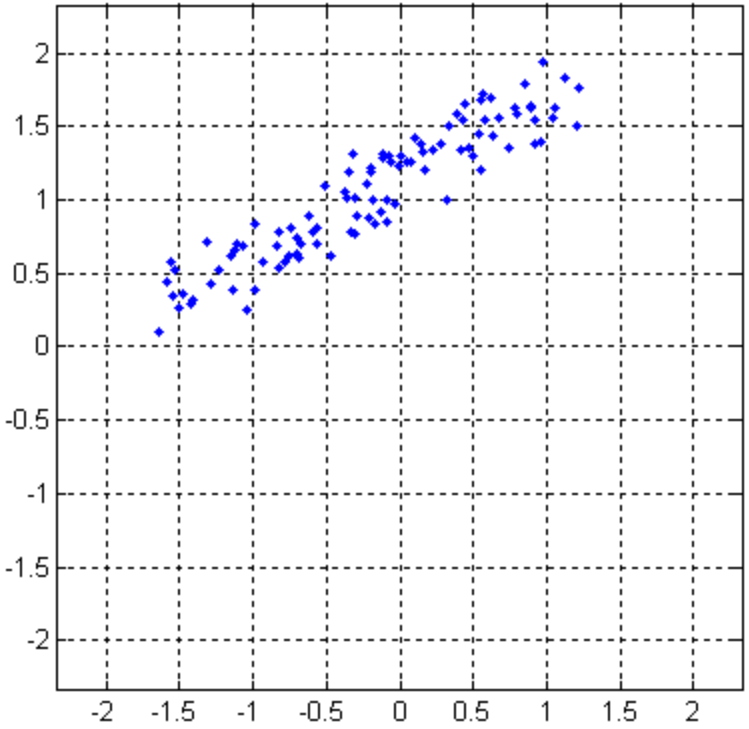
Intrinsic Dimensionality

- Sometimes, high dimensional data is generated using some process that uses only a few parameters.
  - The translated and rotated images of the digit 3 are such an example.
- In that case, the number of those few parameters is called the **intrinsic dimensionality** of the data.
- It is desirable (but oftentimes hard) to discover the intrinsic dimensionality of the data.



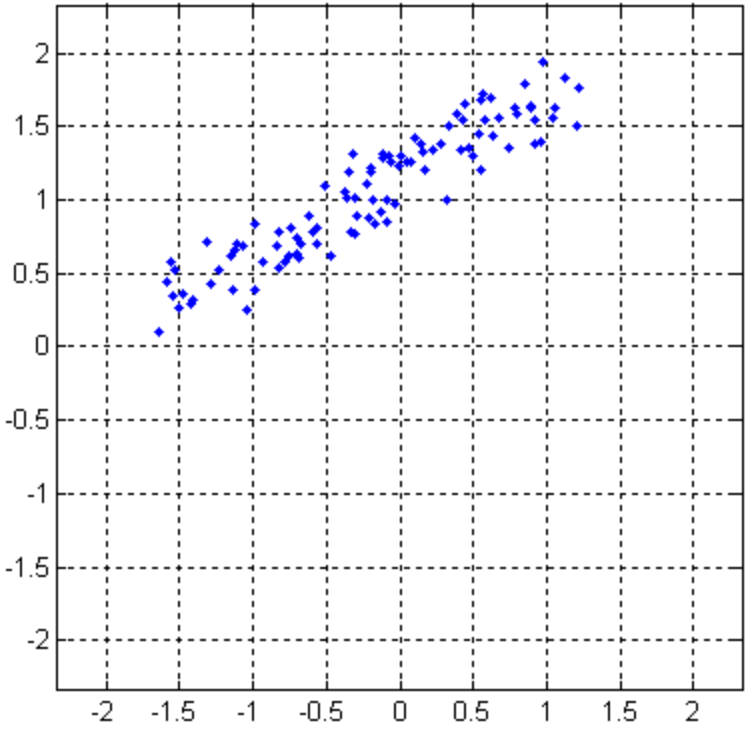
Lossy Dimensionality Reduction

- Suppose we want to project all points to a single line.
- This will be *lossy*.
- What would be the best line?



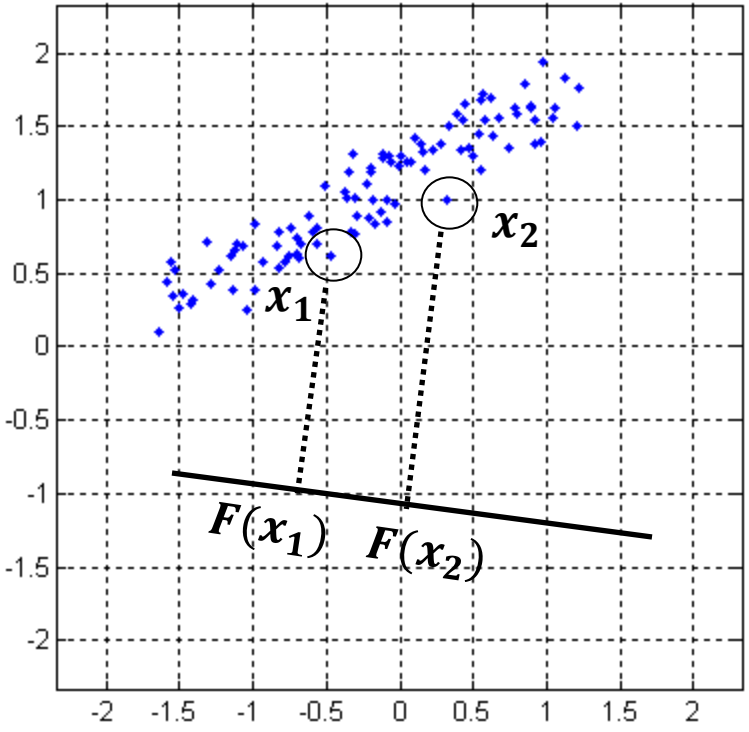
Lossy Dimensionality Reduction

- 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.



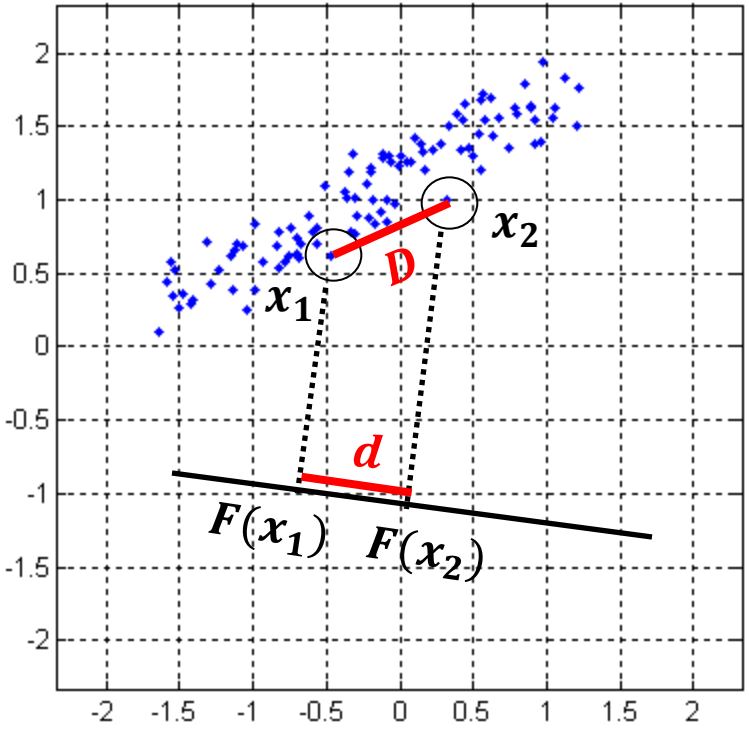
Optimization Criterion

- 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: $\mathbb{R}^2 \rightarrow \mathbb{R}^1$
- Define $D = \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2$.
  - Squared distance from x_1 to x_2.
- Define $d = \|F(\boldsymbol{x}_1) - F(\boldsymbol{x}_2)\|^2$.
- Define error function $\mathrm{E}(\boldsymbol{x}_1, \boldsymbol{x}_2) = D - d$.
- Will $\mathrm{E}(\boldsymbol{x}_1, \boldsymbol{x}_2)$ ever be negative?

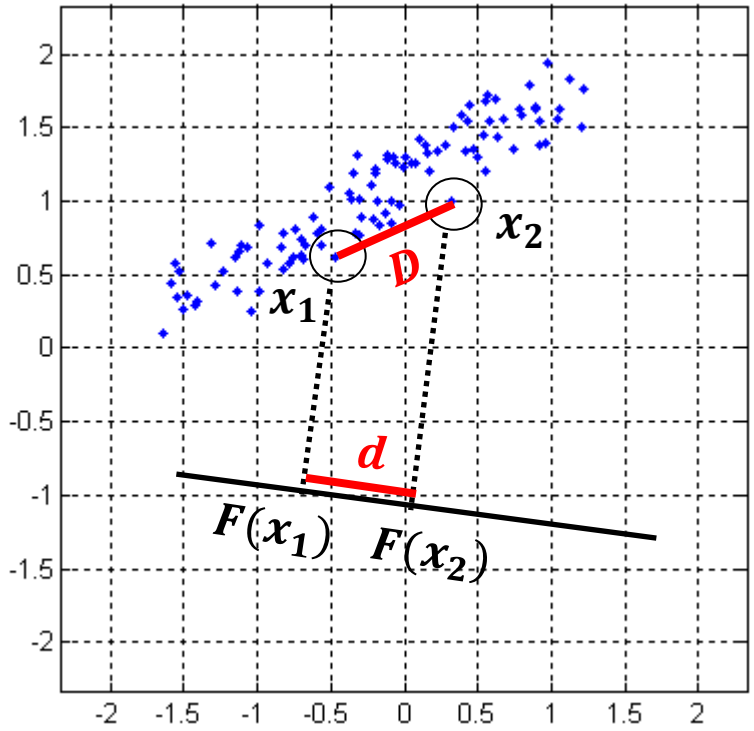


Optimization Criterion

- 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: $\mathbb{R}^2 \rightarrow \mathbb{R}^1$
- Define $D = \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2$.
  - Squared distance from x_1 to x_2.
- Define $d = \|F(\boldsymbol{x}_1) - F(\boldsymbol{x}_2)\|^2$.
- Define error function $\mathrm{E}(\boldsymbol{x}_1, \boldsymbol{x}_2) = D - d$.
- 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



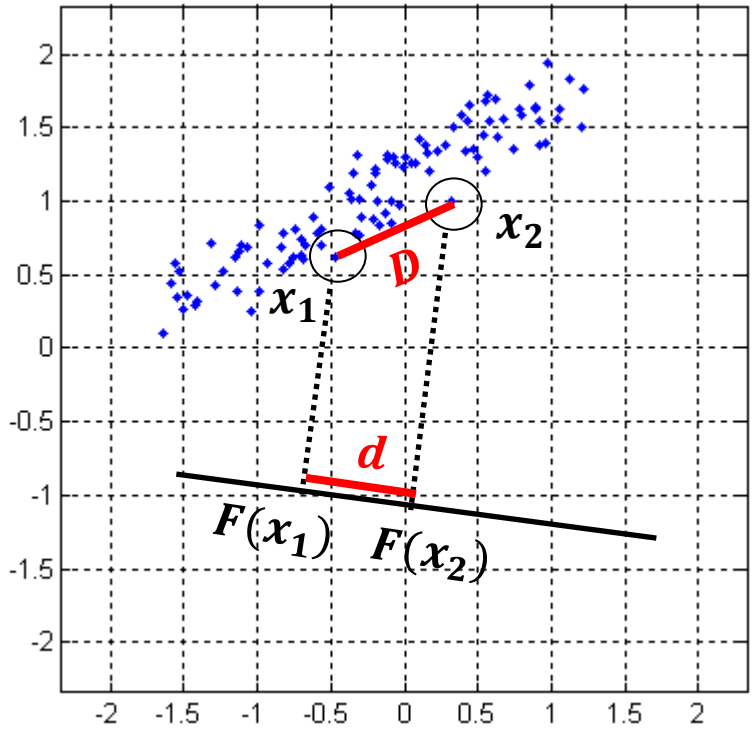
- Now, consider all points:
  - $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$.
- Define error function E as:

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

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

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

Optimization Criterion

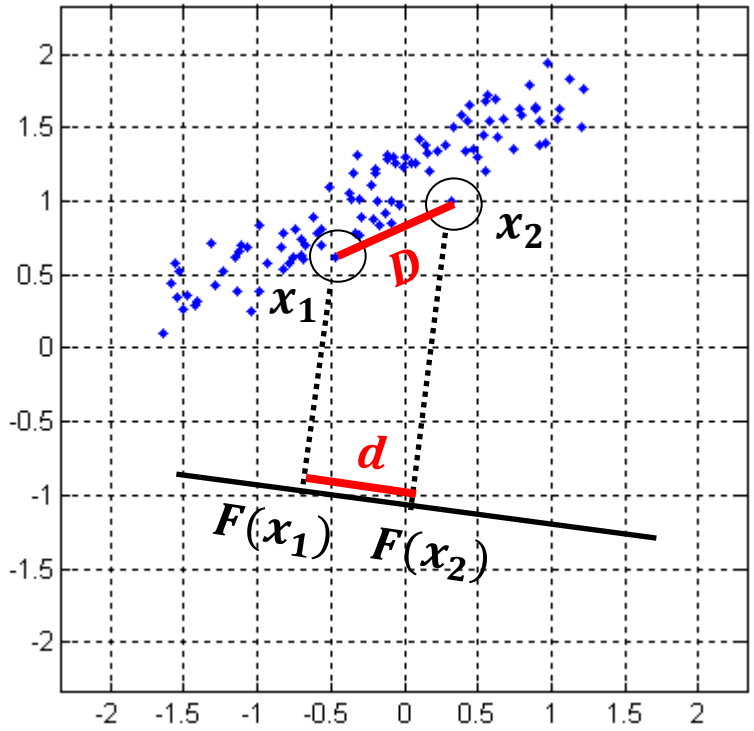


- Now, consider all points:
  - $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$.
- Define error function E as:

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

- Suppose that F perfectly preserves distances.
- Then, $\forall\, \boldsymbol{x}_m, \boldsymbol{x}_n$: $\quad \|\boldsymbol{x}_m - \boldsymbol{x}_n\| = \|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|$
- In that case, $E(F) =$???

Optimization Criterion

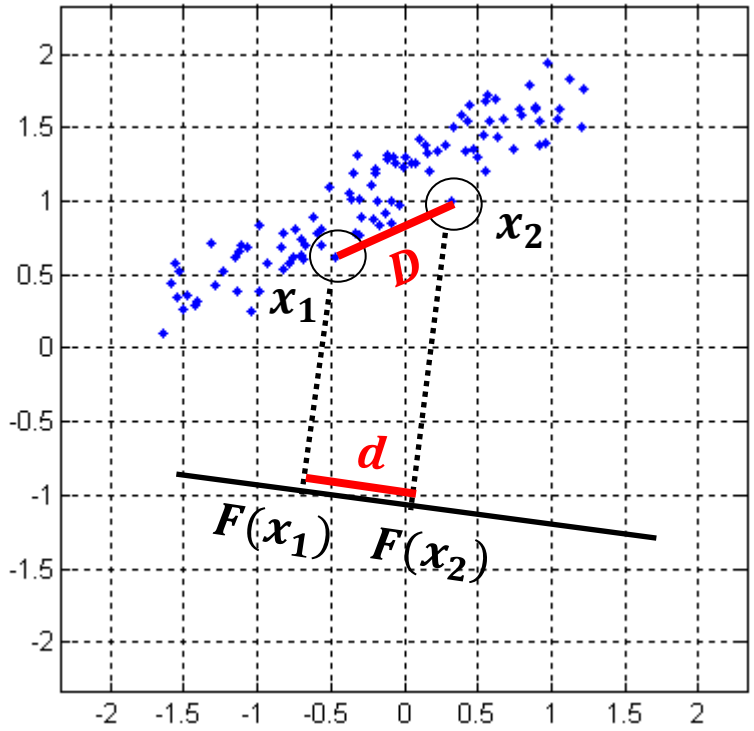


- Now, consider all points:
  - $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$.
- Define error function E as:

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

- Suppose that F perfectly preserves distances.
- Then, $\forall\, \boldsymbol{x}_m, \boldsymbol{x}_n$: $\|\boldsymbol{x}_m - \boldsymbol{x}_n\| = \|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|$
- In that case, $E(F) = 0$.
- In the example shown on the figure, obviously $E(F) > 0$.

Optimization Criterion: Preserving Distances

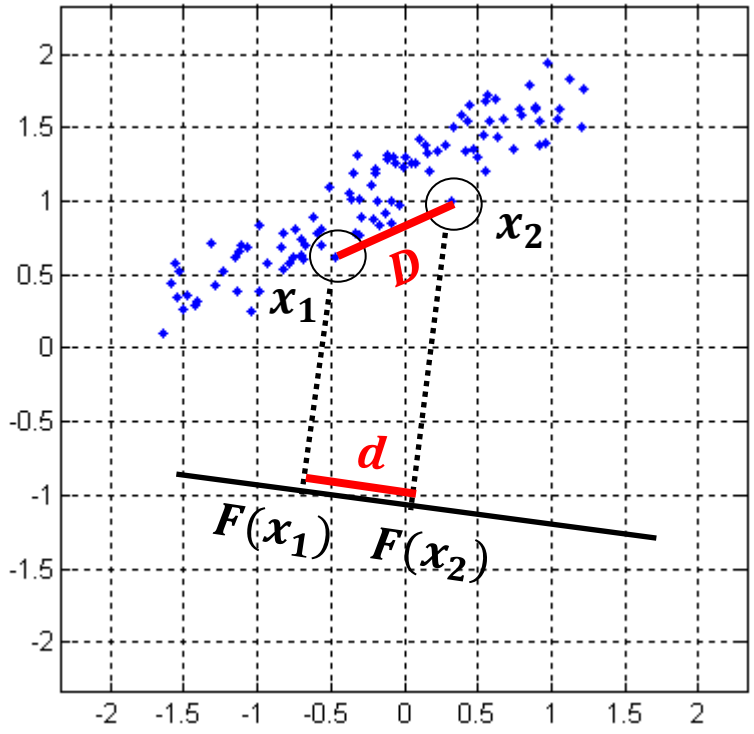


- We have defined an error function $E(F)$ that tells us how good a linear projection is.
- Therefore, the best line projection F_{opt} is the one that minimizes $E(F)$.

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

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

Optimization Criterion: Preserving Distances

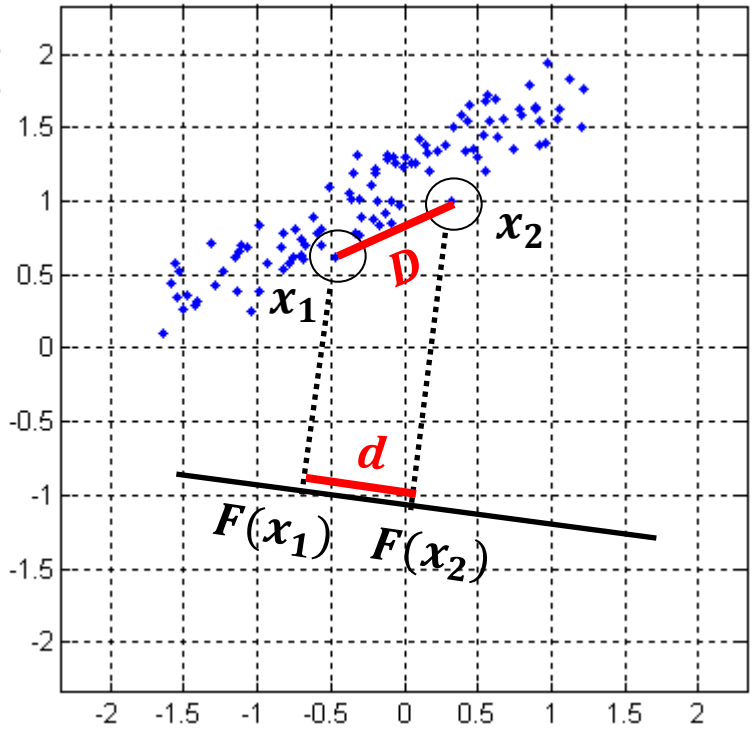


- We have defined an optimization criterion, that measures how well a projection preserves the pairwise distances of the original data.
- The textbook does not mention this criterion. Instead, the textbook mentions two other criteria:
  - Maximizing the variance of the projected data $F(\boldsymbol{x}_n)$.
  - Minimizing the sum of backprojection errors:

$$\sum_{n=1}^{N} \left\{ \left(B\left(F(\boldsymbol{x}_n)\right) - \boldsymbol{x}_n \right)^2 \right\}$$

- We will show that all three criteria are equivalent.

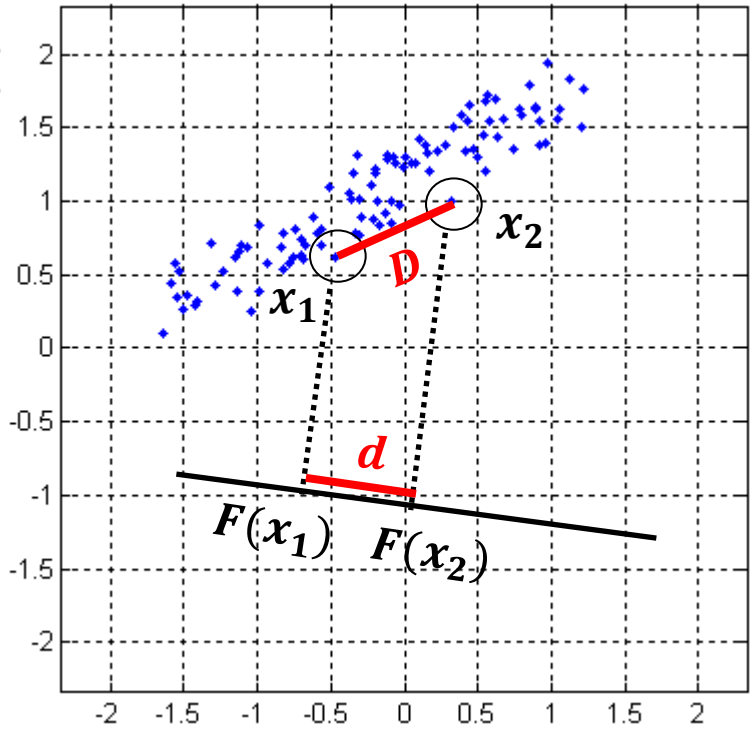
Optimization Criterion: Preserving Distances



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

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

Optimization Criterion: Preserving Distances



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

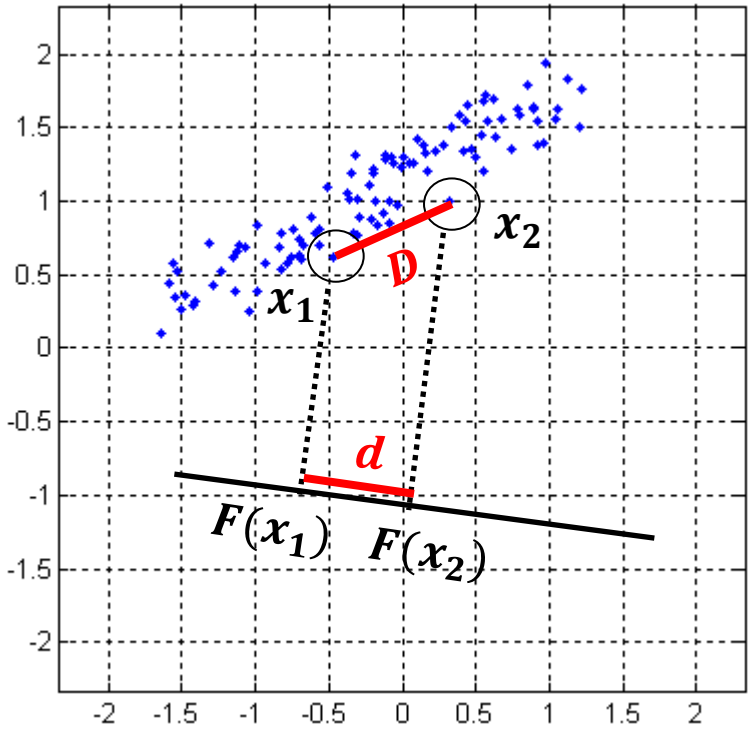
$$= \sum_{m=1}^{N}\sum_{n=1}^{N} [\|\boldsymbol{x}_m - \boldsymbol{x}_n\|^2 - \|F(\boldsymbol{x}_m) - F(\boldsymbol{x}_n)\|^2]$$

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

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

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

Optimization Criterion: Maximizing Distances

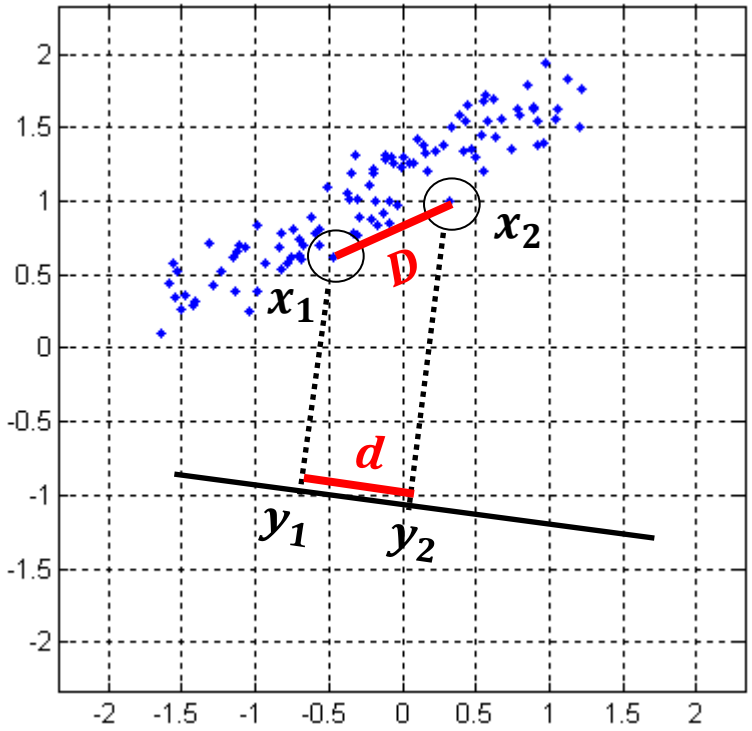


- Therefore, our original criterion (preserving distances as much as possible) is equivalent to maximizing distances in the projected space.

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

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

Optimization Criterion: Maximizing Distances

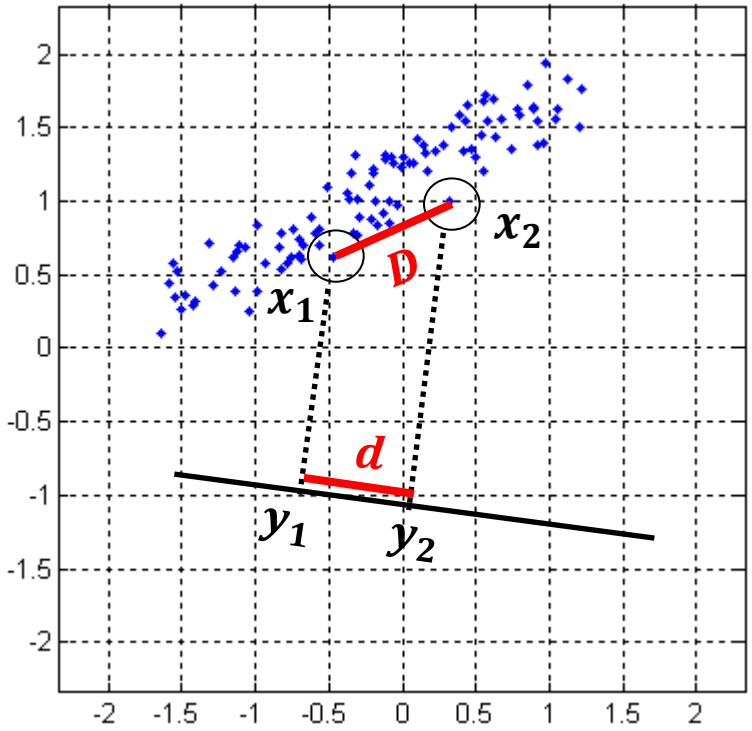


- For convenience, define $y_n = F(\boldsymbol{x}_n)$.
- Also, let's make a temporary assumption that F projects to a line.
  - Then, y_n is a real number.

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

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

Optimization Criterion: Maximizing Distances

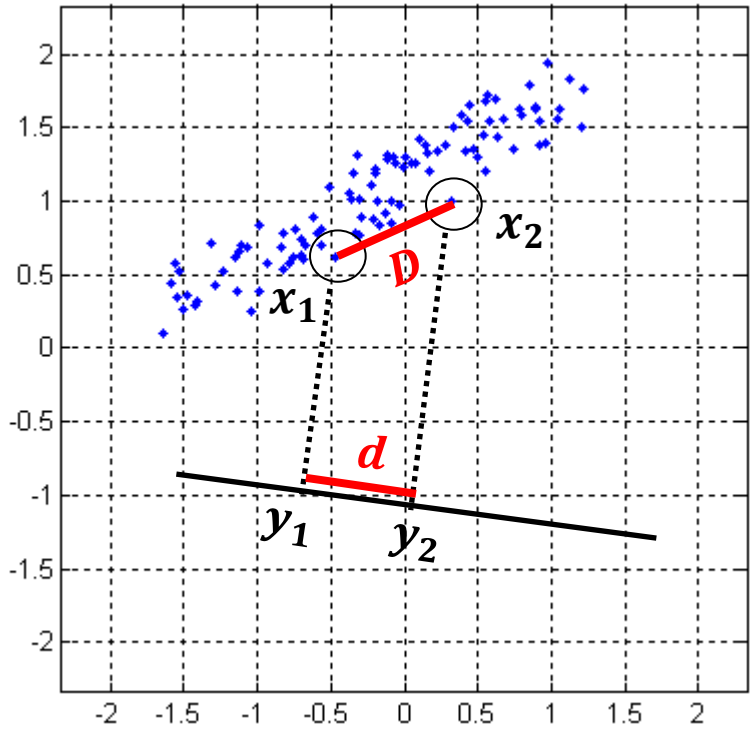


- For convenience, define $y_n = F(x_n)$.
- Also, let's make a temporary assumption that F projects to a line.
  - Then, y_n is a real number.

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

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

Optimization Criterion: Maximizing Distances

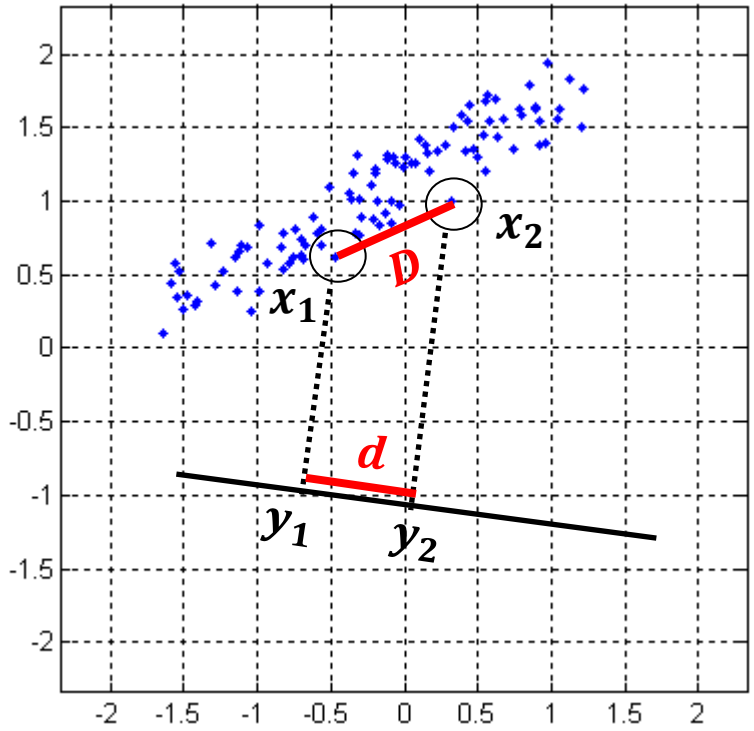


$$\sum_{m=1}^{N}\sum_{n=1}^{N}[(y_m)^2 - y_m y_n + (y_n)^2 - y_m y_n] =$$

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

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

Optimization Criterion: Maximizing Distances

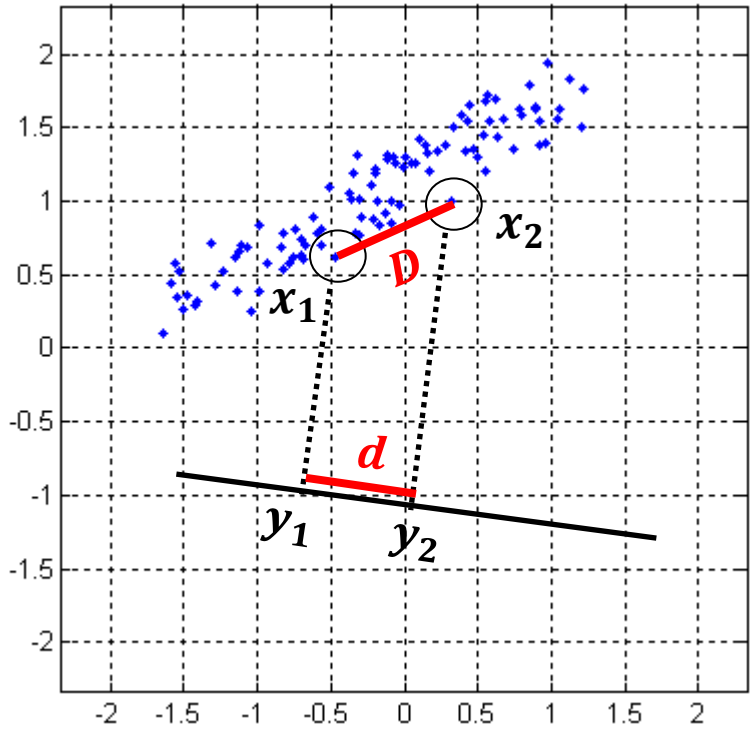


$$\sum_{m=1}^{N}\sum_{n=1}^{N}[\|F(\boldsymbol{x}_m)-F(\boldsymbol{x}_n)\|^2]=$$

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

- Note that $\frac{\sum_{n=1}^{N}y_n}{N}$ is the average of all the projections $F(\boldsymbol{x}_n)$.
- Let's denote this average as μ_y.

Optimization Criterion: Maximizing Distances

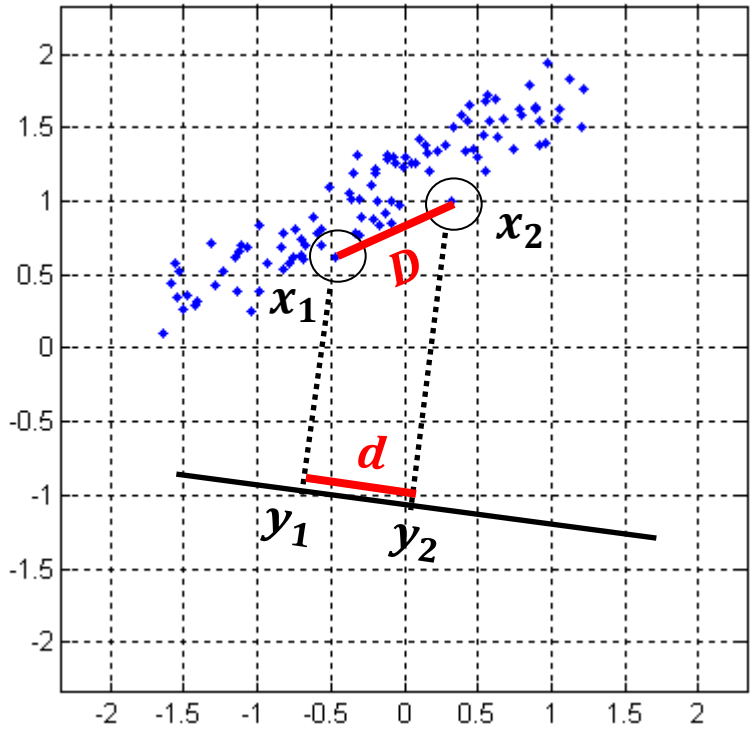


$$\sum_{m=1}^{N}\sum_{n=1}^{N}[\|F(\boldsymbol{x}_m)-F(\boldsymbol{x}_n)\|^2]=$$

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

- Note that $\frac{\sum_{n=1}^{N}y_n}{N}$ is the average of all the projections $F(\boldsymbol{x}_n)$.
- Let's denote this average as μ_y. Then, $\sum_{n=1}^{N}y_n=N\mu_y$.

Optimization Criterion: Maximizing Distances

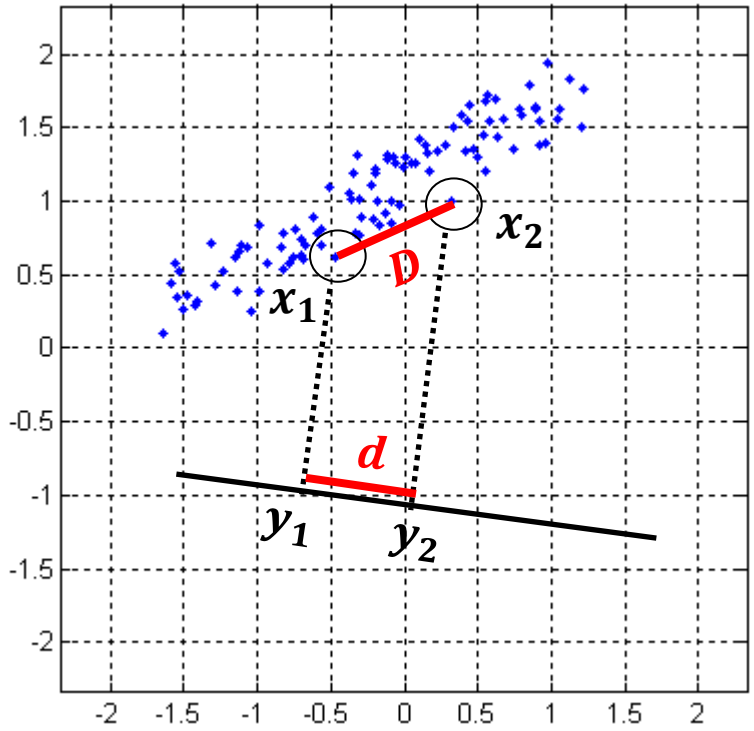


$$\sum_{m=1}^{N}\sum_{n=1}^{N}[\|F(\boldsymbol{x}_m)-F(\boldsymbol{x}_n)\|^2]=$$

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

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

Optimization Criterion: Maximizing Distances

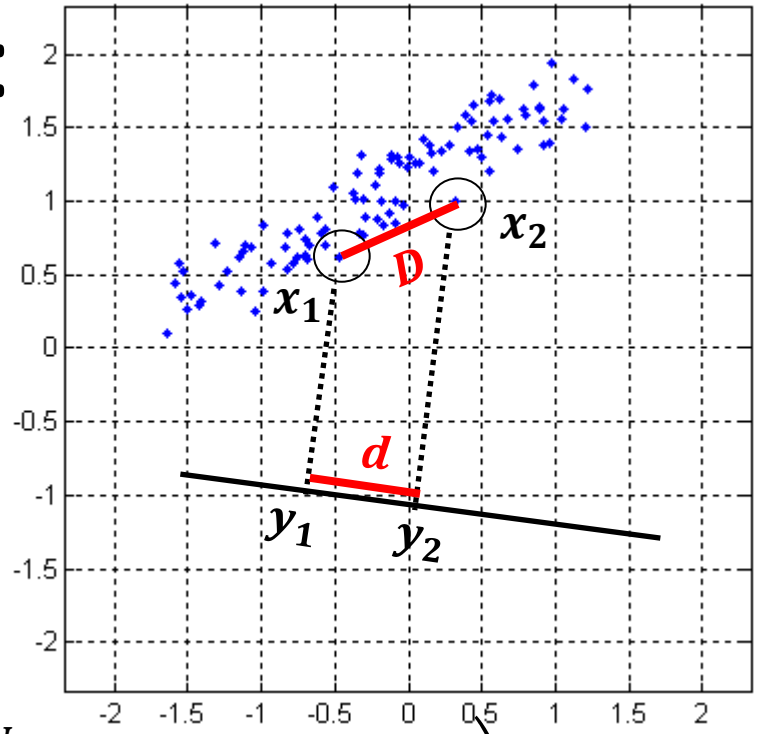


$$\sum_{m=1}^{N}\sum_{n=1}^{N}[\|F(\boldsymbol{x}_m)-F(\boldsymbol{x}_n)\|^2]=$$

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

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

Optimization Criterion: Maximizing Distances



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

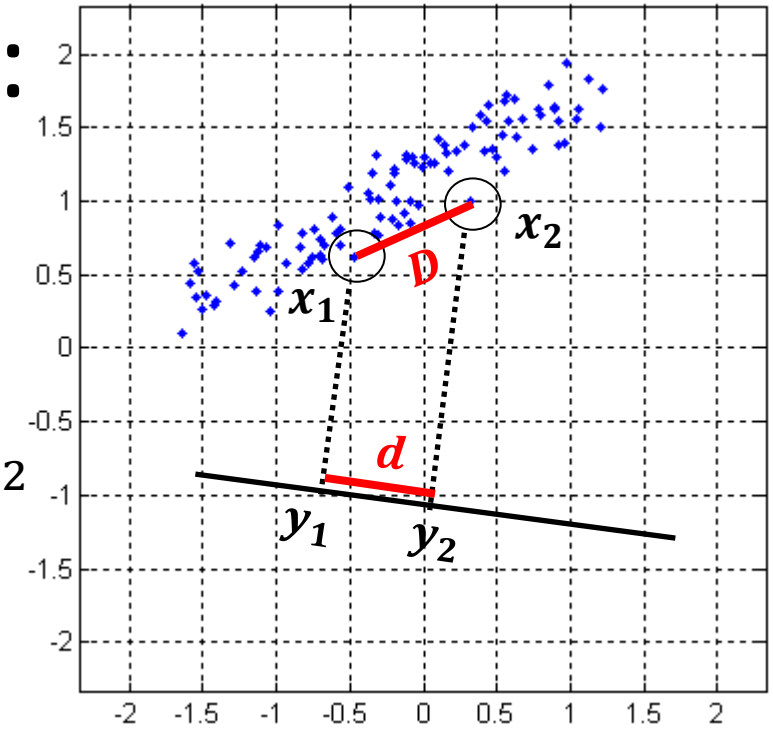
$$2N\left(\sum_{m=1}^{N}\{(y_m)^2\}-N(\mu_y)^2\right)=2N\left(\sum_{m=1}^{N}\{(y_m)^2-(\mu_y)^2\}\right)$$

- 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:

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

Optimization Criterion: Maximizing Variance

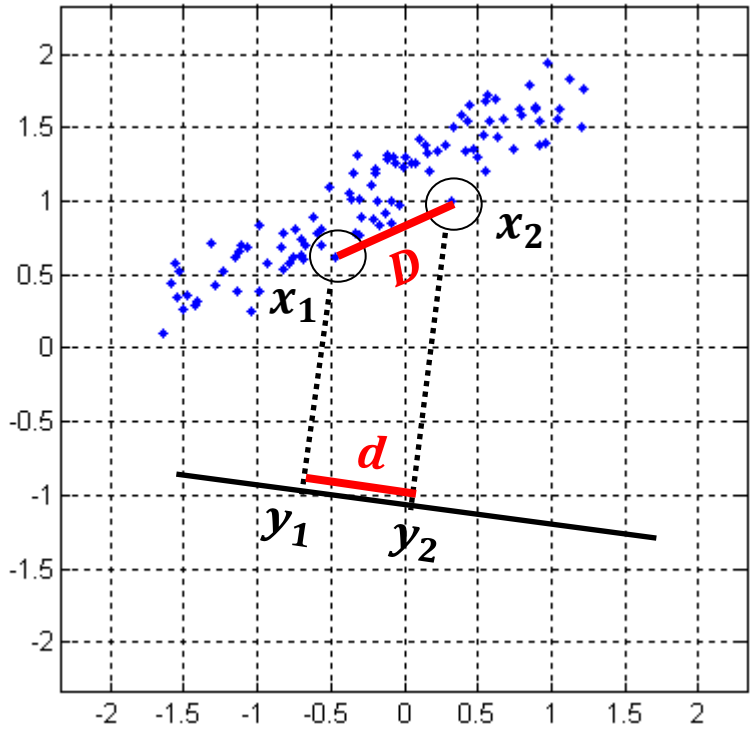
$$\sum_{m=1}^{N}\sum_{n=1}^{N}[\|F(\boldsymbol{x}_m)-F(\boldsymbol{x}_n)\|^2]=2N(\sigma_y)^2$$



- Therefore, these optimization criteria become equivalent:
  - Finding a projection F that preserves the distances of the original data as well as possible.
  - Finding a projection F that maximizes the sum of pairwise distances of projections $F(\boldsymbol{x}_n)$.
  - Finding a projection F that maximizes the variance of the projections $F(\boldsymbol{x}_n)$.

Optimization Criterion: Maximizing Variance

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

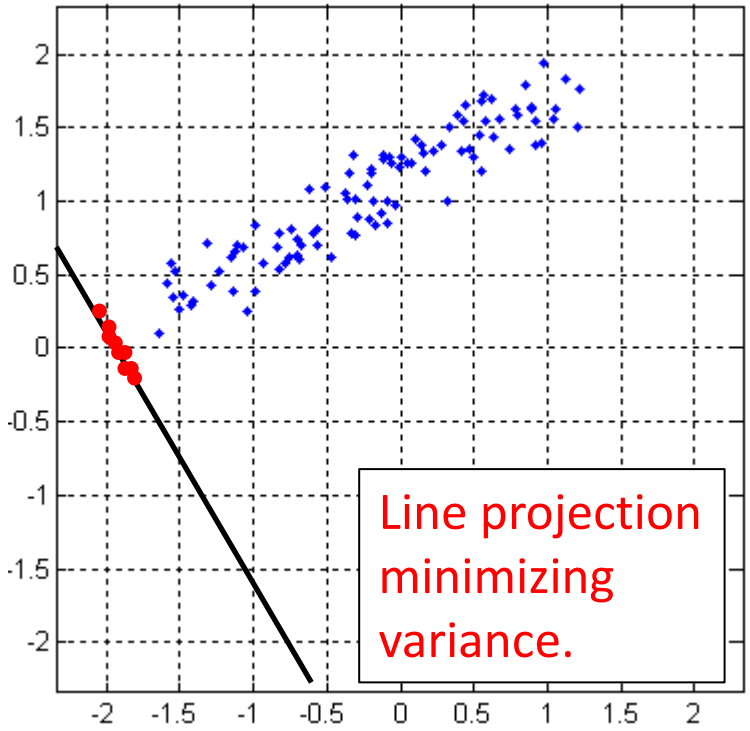


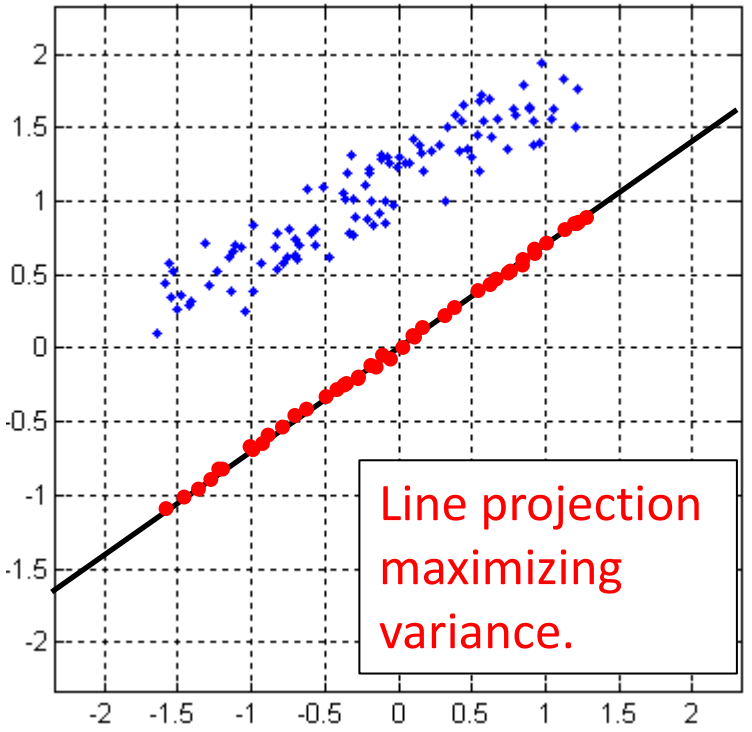
- Therefore, these optimization criteria become equivalent:
  - Finding a projection F that preserves the distances of the original data as well as possible.
  - Finding a projection F that maximizes the sum of pairwise distances of projections $F(\boldsymbol{x}_n)$.
  - Finding a projection F that maximizes the variance of the projections $F(\boldsymbol{x}_n)$.

Maximizing the Variance

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

- 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).

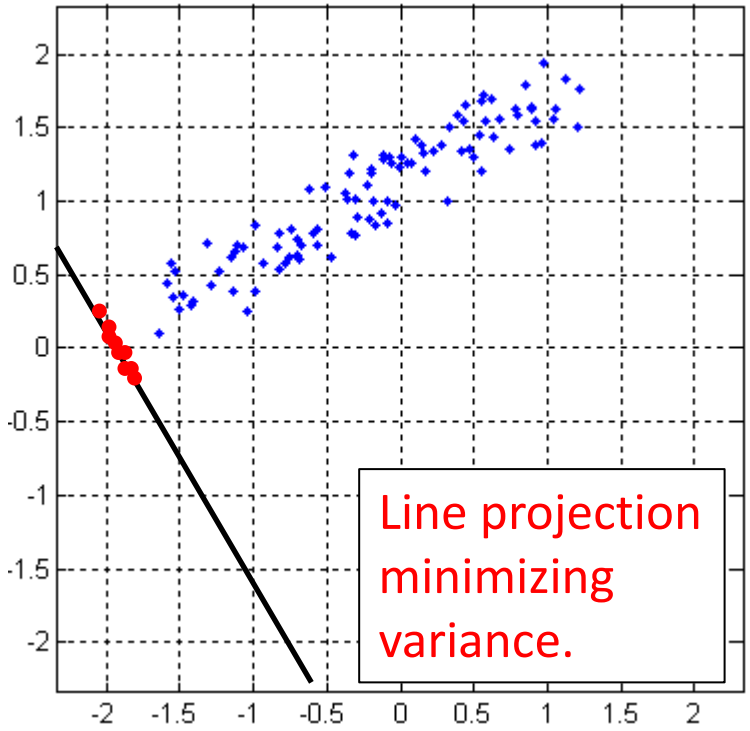


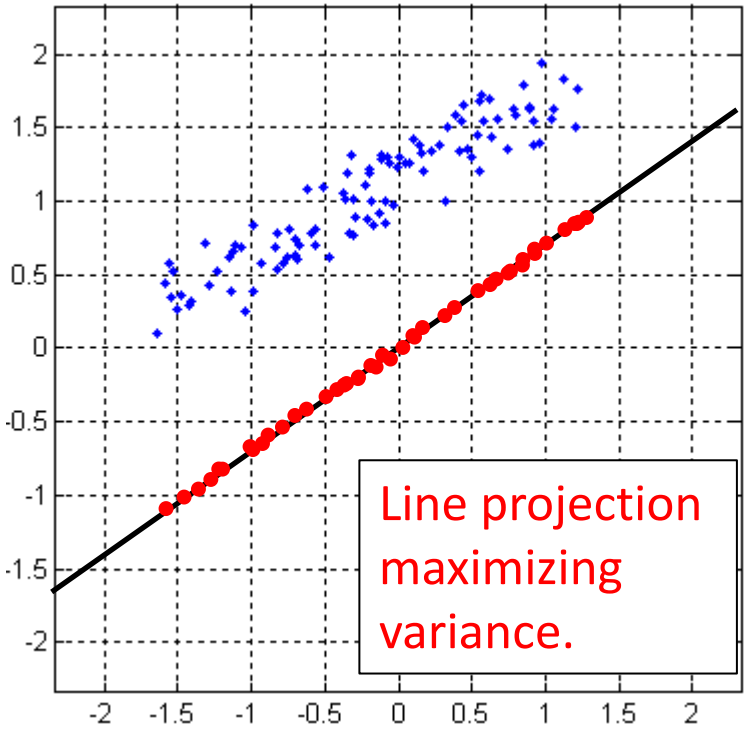


Maximizing the Variance

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

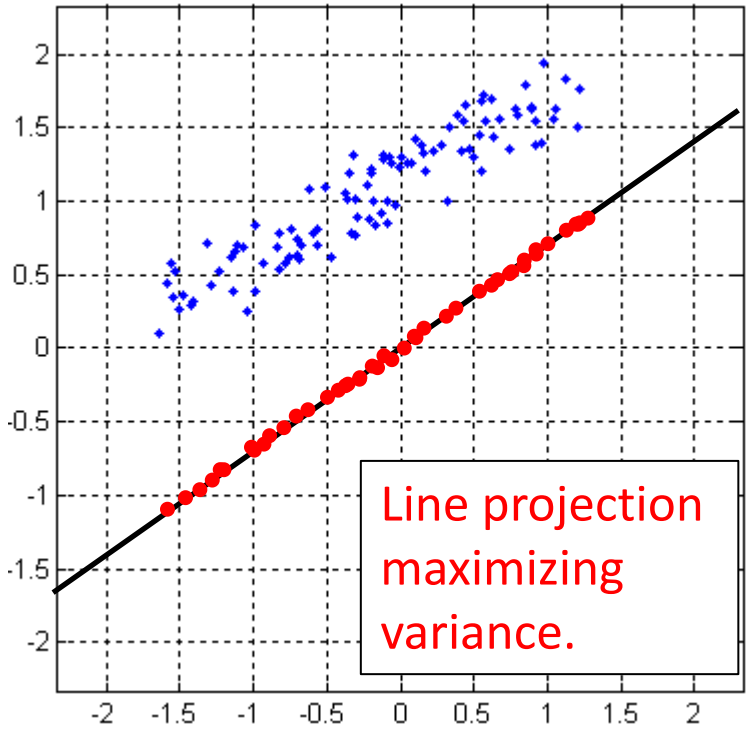
- Next step: finding F_{opt}.





Maximizing the Variance

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



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

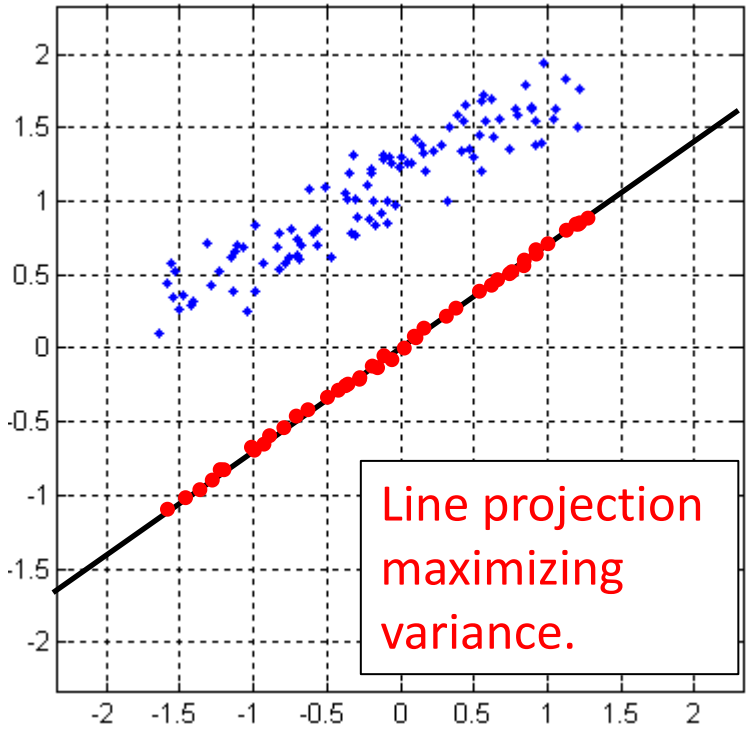
$$F(\boldsymbol{x}_n) = (\boldsymbol{u}_1)^T \boldsymbol{x}_n$$

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

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

Maximizing the Variance

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



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

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

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

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

Maximizing the Variance

- The variance of the projected data is:

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

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

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

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

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

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.
- However, we also have the constraint that $\boldsymbol{u}_1$ should be a unit vector: $(\boldsymbol{u}_1)^T \boldsymbol{u}_1 = 1$.
- Why do we need this constraint?

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.
- However, we also have the constraint that $\boldsymbol{u}_1$ should be a unit vector: $(\boldsymbol{u}_1)^T \boldsymbol{u}_1 = 1$.
- Why do we need this constraint?
  - Because otherwise, to maximize the variance we can just make $\boldsymbol{u}_1$ arbitrarily large.

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.
- However, we also have the constraint that $\boldsymbol{u}_1$ should be a unit vector: $(\boldsymbol{u}_1)^T \boldsymbol{u}_1 = 1$.
- So, we have a constrained maximization problem.
- 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:

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

- 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 $\boldsymbol{S}$.
- It is time for a quick review of eigenvectors and eigenvalues.

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?
- If we left-multiply by $(\boldsymbol{u}_1)^T$ both sides of $\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$, 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

- $\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$.
- 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 in the previous slide, $(\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1 = \lambda_1$.
- As we saw a few slides earlier, $(\boldsymbol{u}_1)^T \boldsymbol{S}\boldsymbol{u}_1$ is the actual variance of the projected data.
- Therefore, 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.
- One 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 **square** matrix $\boldsymbol{A}$.
  - For PCA, $\boldsymbol{A}$ is a covariance matrix. For other applications of eigenvectors, $\boldsymbol{A}$ just needs to be a square 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.

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

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

Computing a 2-Dimensional Projection

- A 2-dimensional 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, using the power method.
- 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 $\{\boldsymbol{x}_{1,2}, \boldsymbol{x}_{2,2}, \ldots, \boldsymbol{x}_{N,2}\}$.
  - 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 $\boldsymbol{S}_2$).

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 $\{\boldsymbol{x}_{1,2}, \boldsymbol{x}_{2,2}, \ldots, \boldsymbol{x}_{N,2}\}$.
  - 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:

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

  - 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.

Projection to Orthogonal Subspace

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

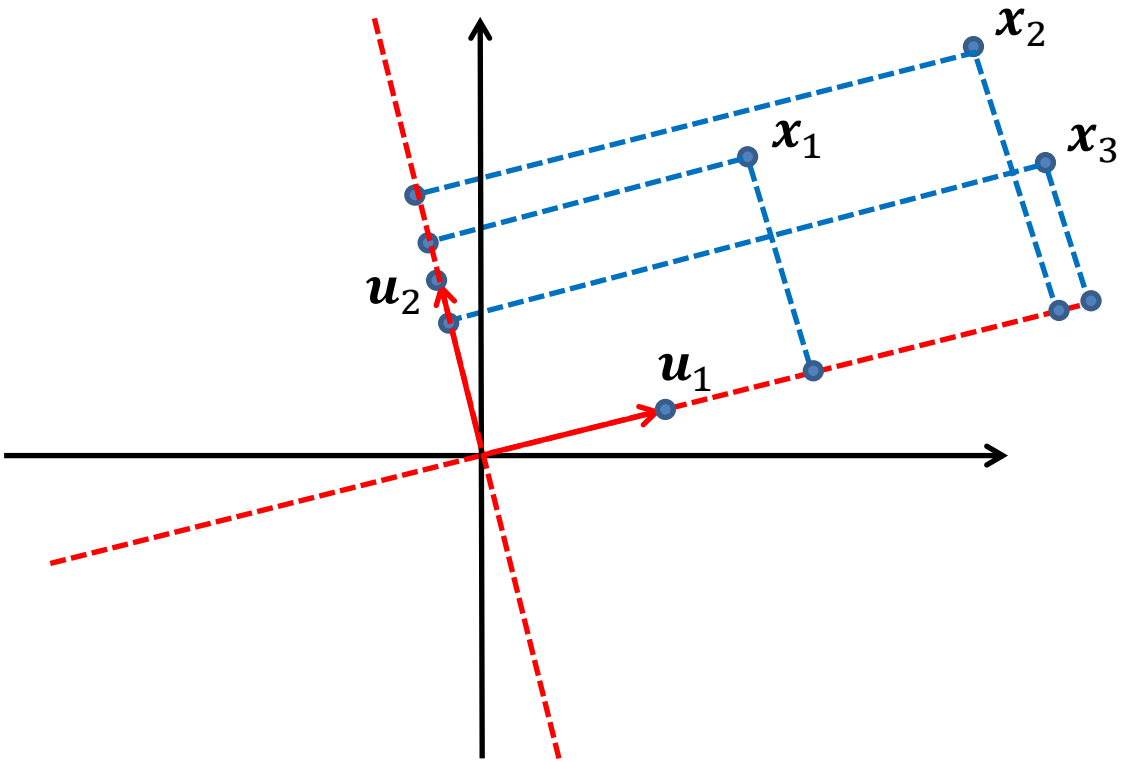
$$\boldsymbol{x}_n = (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1 + (\boldsymbol{x}_n - (\boldsymbol{u}_1)^T \boldsymbol{x}_n \boldsymbol{u}_1)$$

  - 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 $\boldsymbol{u}_1$.

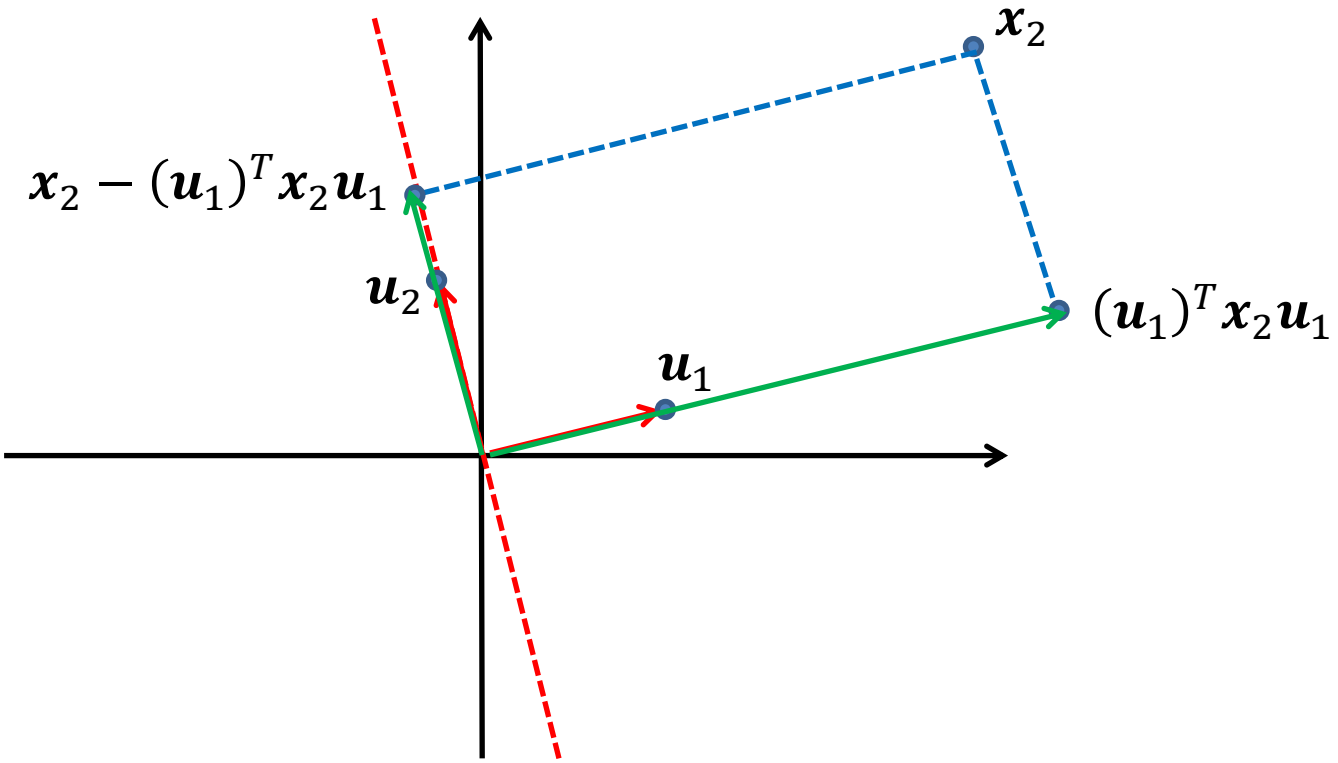
Projection to Orthogonal Subspace

- 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 $\boldsymbol{u}_1$.



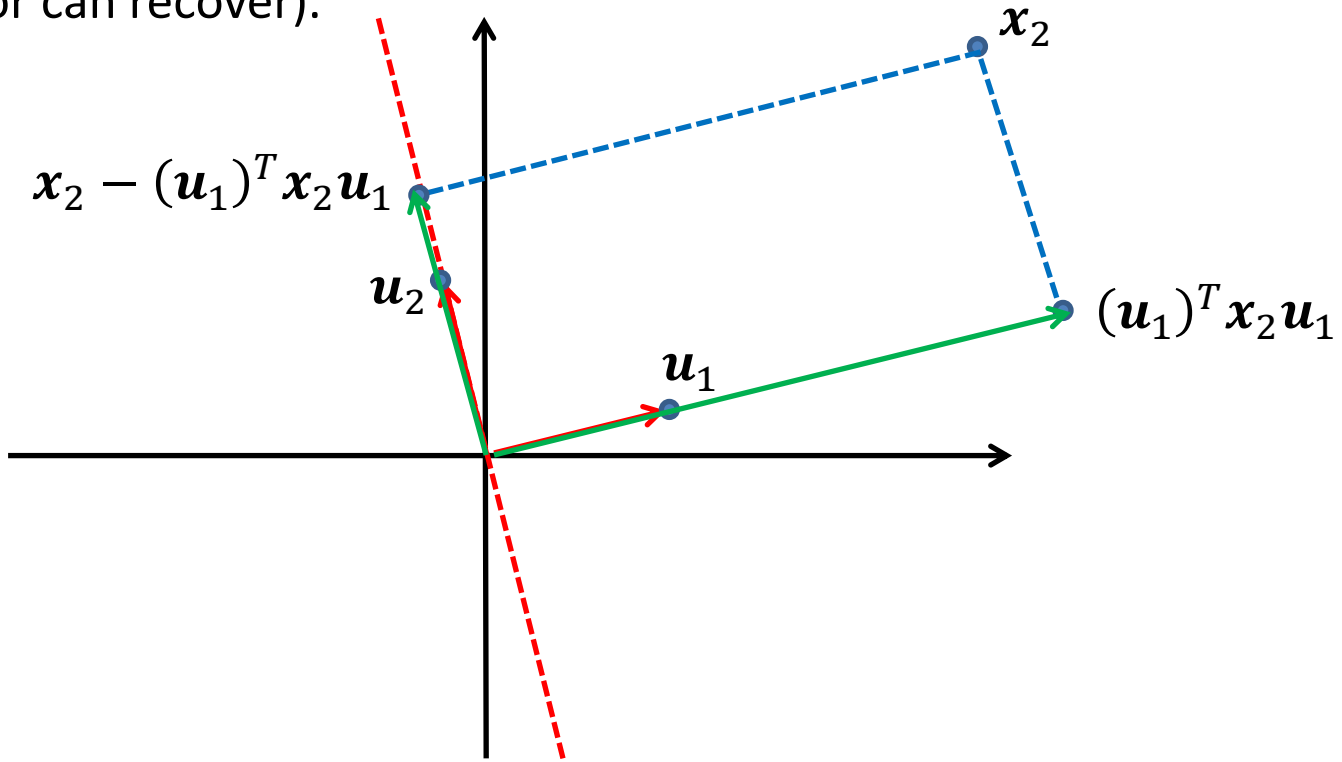
Projection to Orthogonal Subspace

- 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 $\boldsymbol{u}_1$.



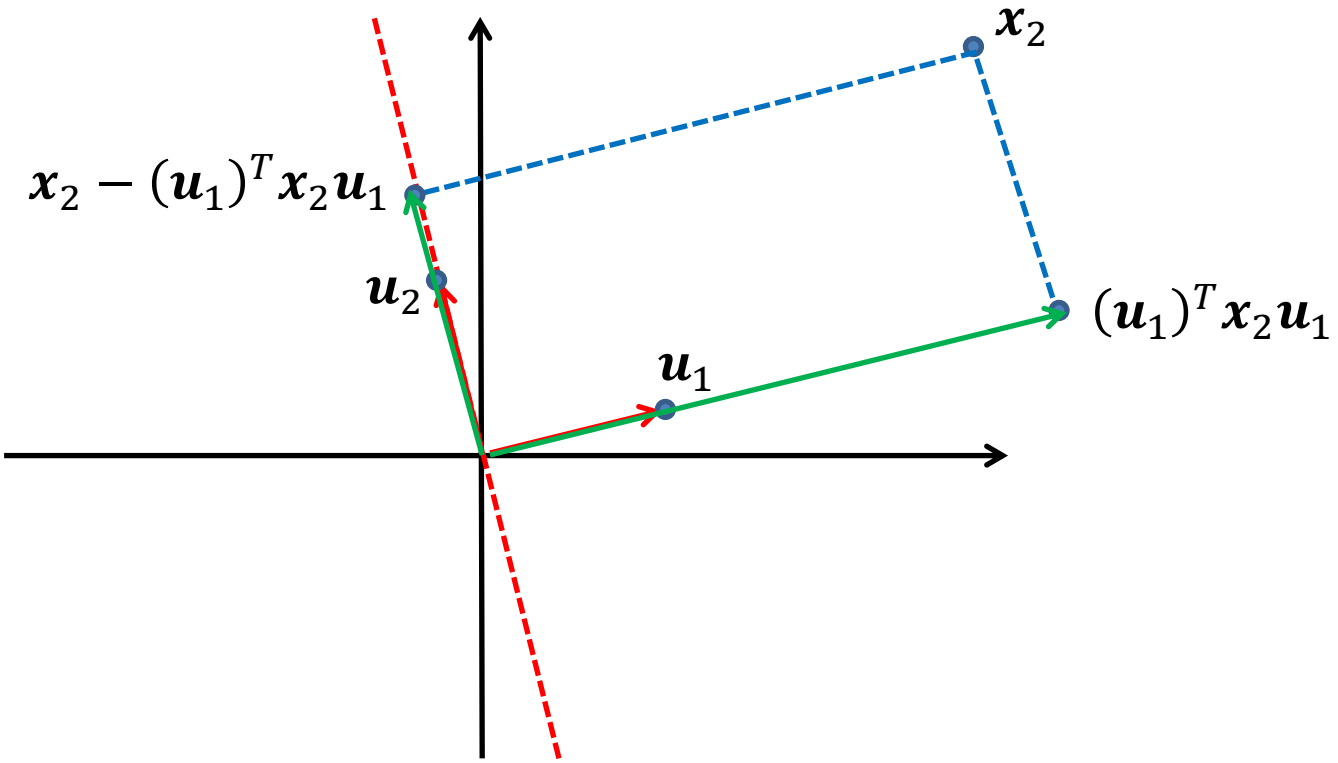
Projection to Orthogonal Subspace

- Unfortunately, visualizing more than two dimensions is harder.
- Here, the projection to the orthogonal subspace is not very interesting.
- However, suppose $D = 30$. Then, the orthogonal subspace has 29 dimensions.
- Then, projection to $\boldsymbol{u}_1$ typicall loses a lot of information from the original data.
- Projection to $\boldsymbol{u}_2$ recovers some of that information (as much as a projection to a single vector can recover).



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 $\boldsymbol{u}_1$.



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 $\{\boldsymbol{x}_{1,d}, \boldsymbol{x}_{2,d}, \ldots, \boldsymbol{x}_{N,d}\}$.
- 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 $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M$.
- Then, the projection function F is defined as: $F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \\ \ldots \\ (\boldsymbol{u}_M)^T \boldsymbol{x} \end{bmatrix}$
- Alternatively, we define an $M \times D$ **projection matrix** $\boldsymbol{U} = \begin{bmatrix} (\boldsymbol{u}_1)^T \\ (\boldsymbol{u}_2)^T \\ \ldots \\ (\boldsymbol{u}_M)^T \end{bmatrix}$.
- Then, the projection F is defined as: $F(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{x}$.

Eigenvectors

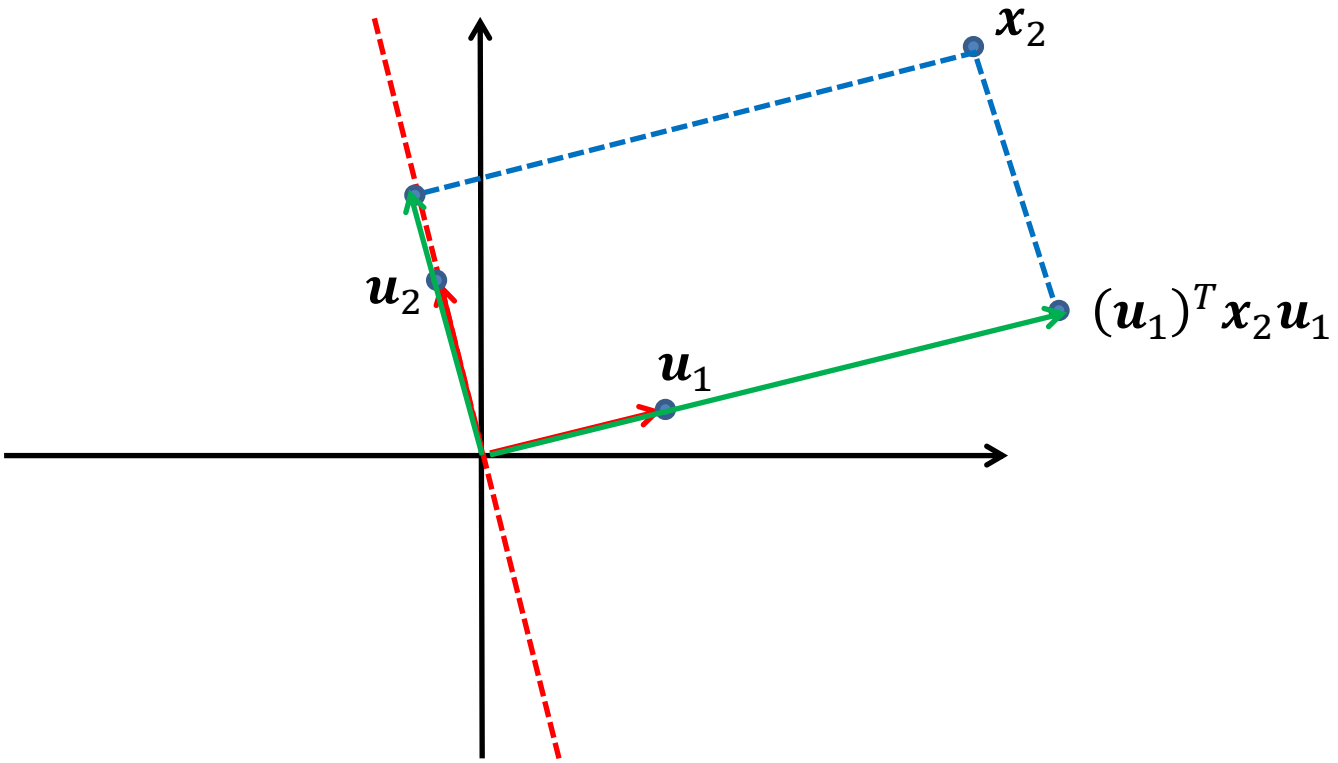
- Vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M$ are the eigenvectors of covariance matrix $\boldsymbol{S}$ with the M largest eigenvalues.
- These vectors are also called the M principal components of the data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.
- 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 $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M$ 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 $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M$ 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 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.
- The process of finding the principal components of the data is called **Principal Component Analysis** (PCA).
  - PCA is the best known and most widely used method for dimensionality reduction.
- 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.

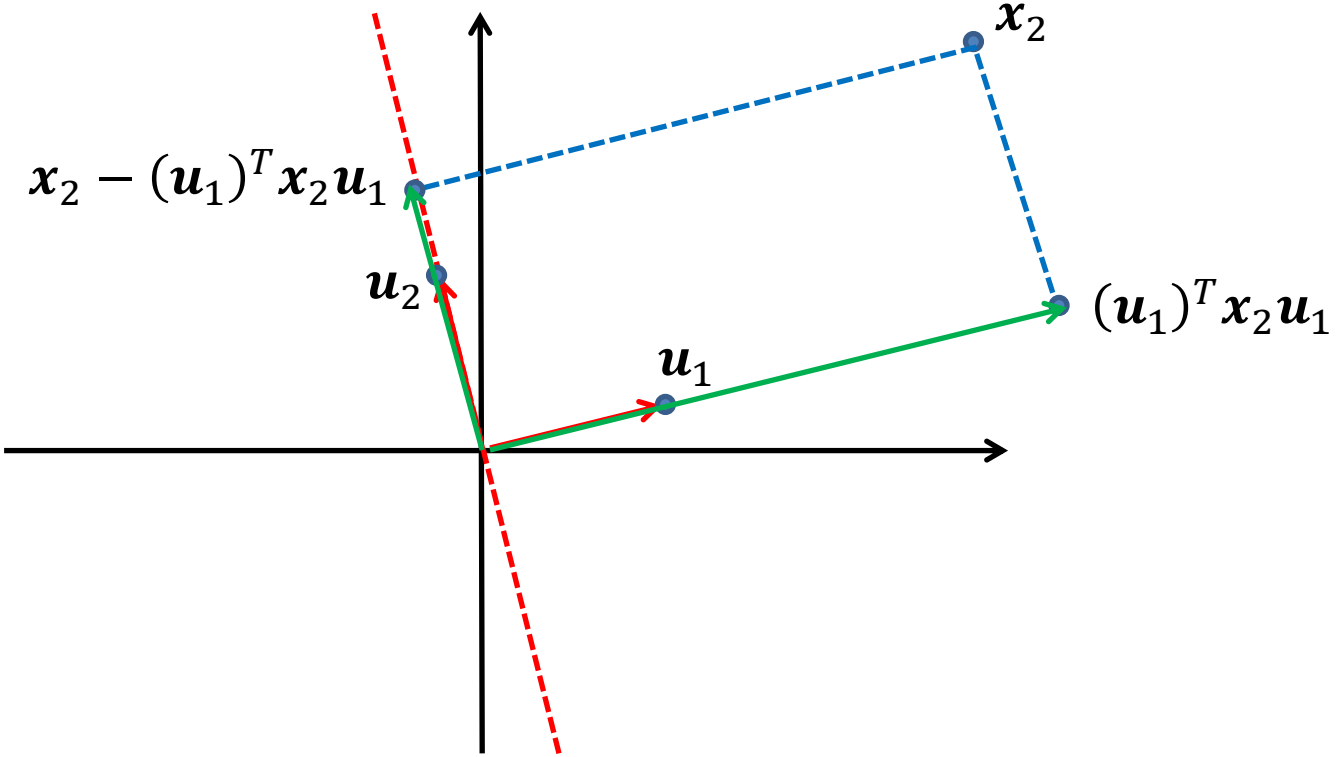
Backprojection?

- So far, we have defined a projection $F(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{x}$, that maps a D-dimensional vector $\boldsymbol{x}$ to an M-dimensional vector $F(\boldsymbol{x})$.
- We also would like to have a **backprojection function** $B(F(\boldsymbol{x}))$, that tries to estimate the original $\boldsymbol{x}$ given $F(\boldsymbol{x})$.



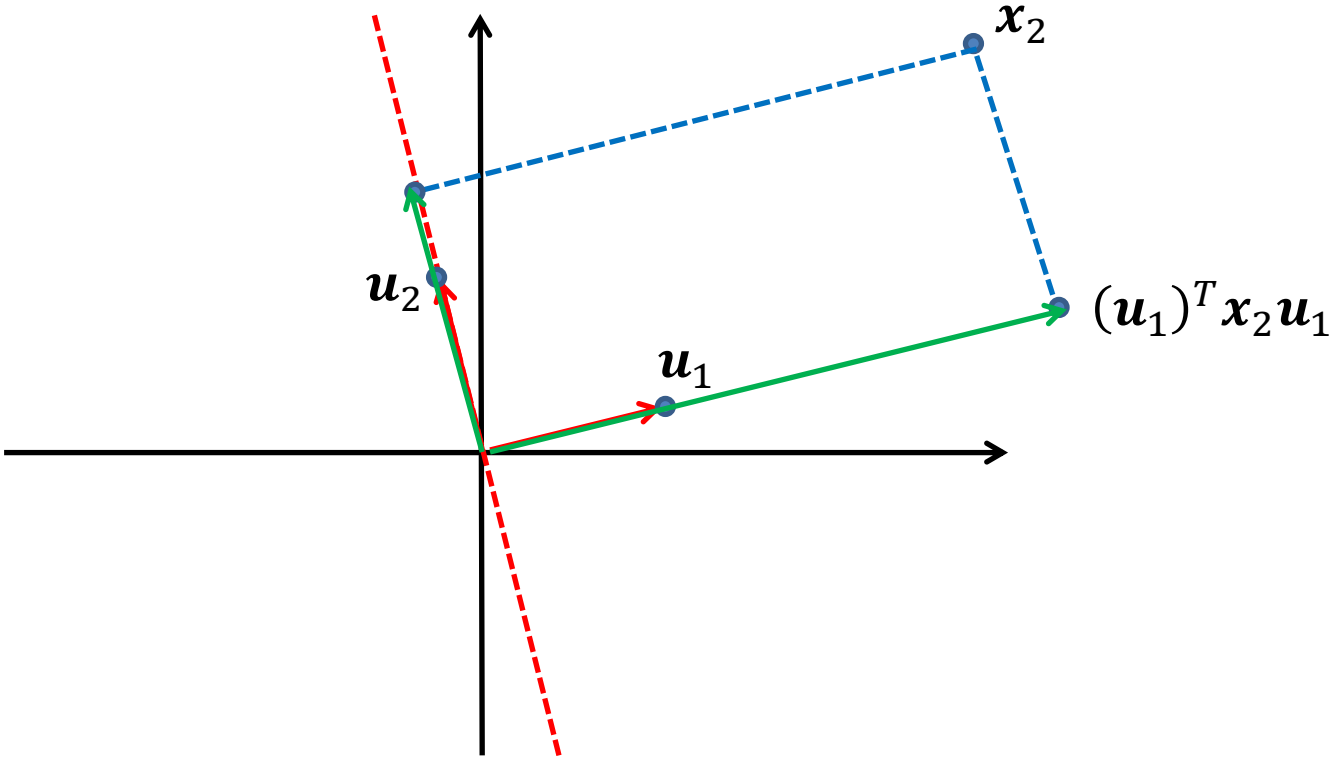
Backprojection?

- This backprojection function cannot be perfect.
  - An infinite number of D-dimensional vectors map to the same $F(\boldsymbol{x})$.
  - In the figure, all points $\boldsymbol{x}$ on the line connecting $\boldsymbol{x}_2$ to $(\boldsymbol{u}_1)^T\boldsymbol{x}_2\boldsymbol{u}_1$ have $F(\boldsymbol{x}) = F(\boldsymbol{x}_2)$.
  - $B(F(\boldsymbol{x}_2))$ can only return one of those points.



Backprojection?

- We will now see a third formulation for PCA, that uses a different optimization criterion.
  - It ends up defining the same projection as our previous formulation.
  - However, this new formulation gives us an easy way to define the backprojection function.



Minimum-Error Formulation

- As before, we have data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.
  - Each $\boldsymbol{x}_n$ is a D-dimensional vector.
- Suppose that we have an orthonormal basis $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_D\}$ for D-dimensional vector space $\mathbb{R}^D$.
- Then, each $\boldsymbol{x}_n$ can be reconstructed perfectly using its projection to the basis vectors:

$$\boldsymbol{x}_n = \sum_{i=1}^{D} [(\boldsymbol{u}_i)^T \boldsymbol{x}\, \boldsymbol{u}_i]$$

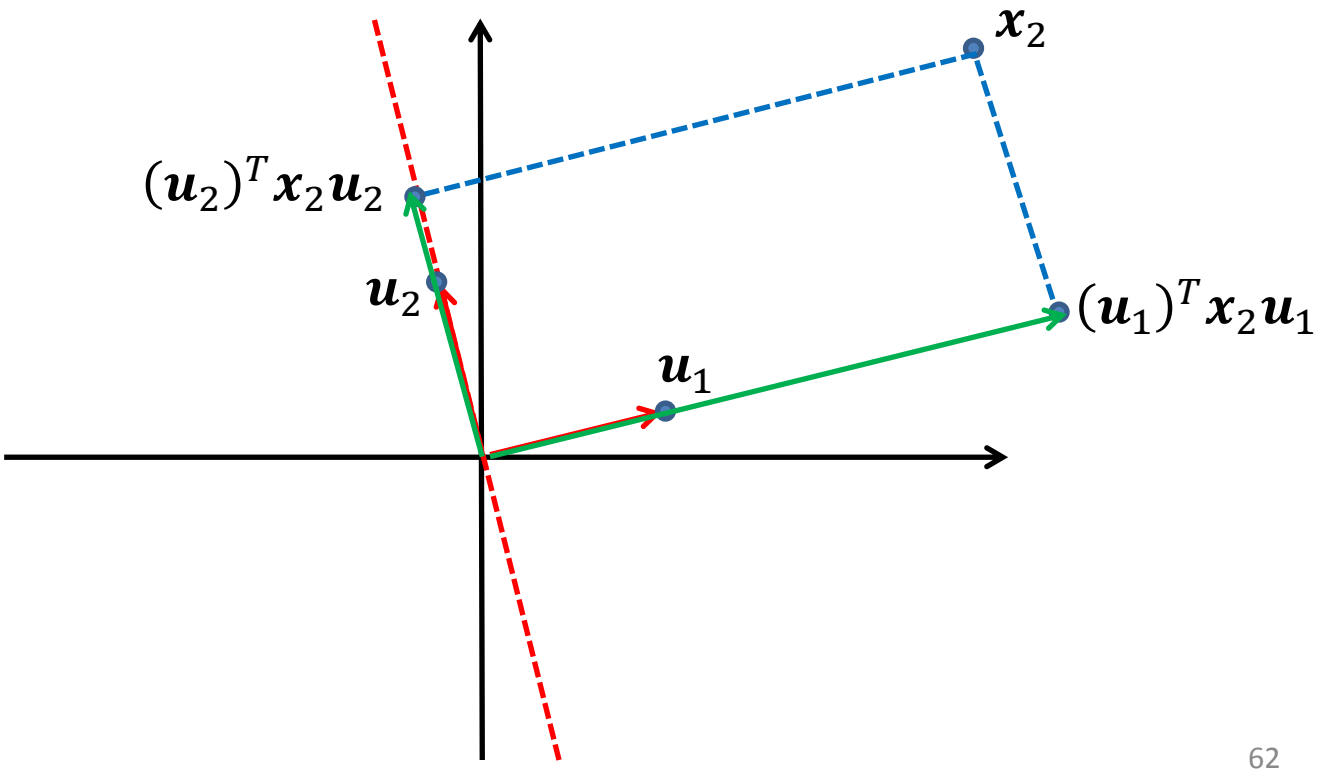
- Remember:
  - $(\boldsymbol{u}_i)^T \boldsymbol{x}$ is just a real number.
  - $(\boldsymbol{u}_i)^T \boldsymbol{x}\, \boldsymbol{u}_i$ is a vector pointing in the same direction as $\boldsymbol{u}_i$, with norm $(\boldsymbol{u}_i)^T \boldsymbol{x}$.

Minimum-Error Formulation

$$\boldsymbol{x}_n = \sum_{i=1}^{D} [(\boldsymbol{u}_i)^T \boldsymbol{x}\, \boldsymbol{u}_i]$$

In the figure, we see that:

$$\boldsymbol{x}_2 = (\boldsymbol{u}_1)^T \boldsymbol{x}\, \boldsymbol{u}_1 + (\boldsymbol{u}_2)^T \boldsymbol{x}\, \boldsymbol{u}_2$$



Minimum-Error Formulation

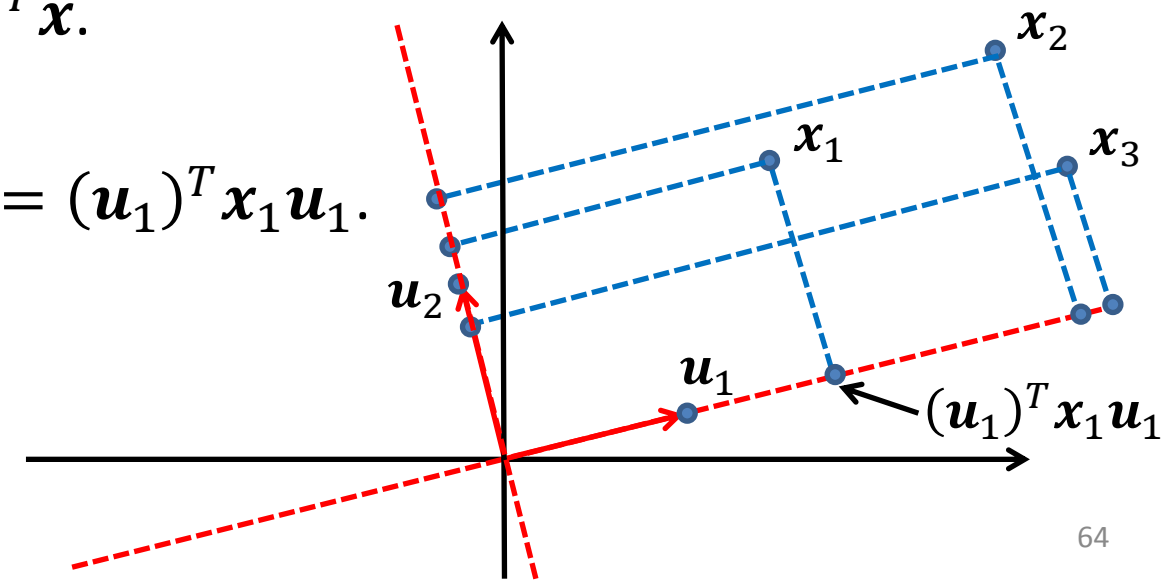
- Suppose that we have an orthonormal basis $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_D\}$ for D-dimensional vector space $\mathbb{R}^D$.
- Suppose that we pick the first M of those basis vectors, $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_M\}$, to define a projection $F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \\ \ldots \\ (\boldsymbol{u}_M)^T \boldsymbol{x} \end{bmatrix}$.
- A simple (and naïve) way to define the backprojection function B_1 is:

$$B_1\left(\begin{bmatrix} z_1 \\ z_2 \\ \ldots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i]$$

Minimum-Error Formulation

$$F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T\boldsymbol{x} \\ (\boldsymbol{u}_2)^T\boldsymbol{x} \\ \dots \\ (\boldsymbol{u}_M)^T\boldsymbol{x} \end{bmatrix}, \qquad B_1\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i\boldsymbol{u}_i]$$

- In the figure:
- Suppose $F(\boldsymbol{x}) = (\boldsymbol{u}_1)^T\boldsymbol{x}$.
- Suppose $z_1 = F(\boldsymbol{x}_1)$.
- Then, $B_1(z_1) = z_1\boldsymbol{u}_1 = (\boldsymbol{u}_1)^T\boldsymbol{x}_1\boldsymbol{u}_1$.



Minimum-Error Formulation

- The way we will actually define the backprojection function B is:

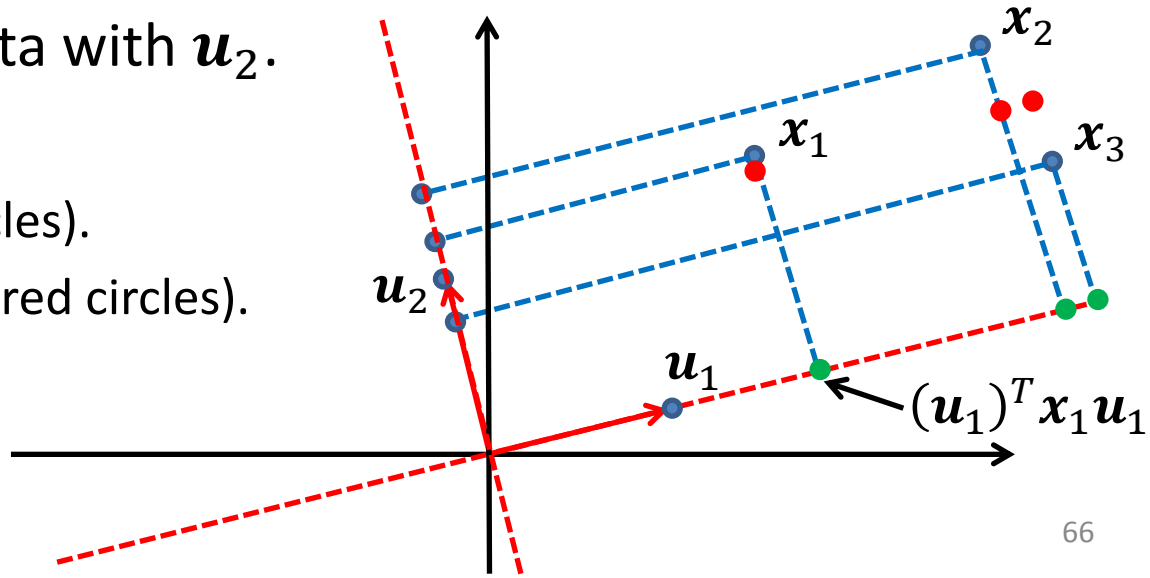
$$B\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i] + \sum_{i=M+1}^{D} [b_i \boldsymbol{u}_i]$$

- The red part is the difference with the previous definition.
- The idea is that we can improve the backprojection, by adding a linear combination of the eigenvectors that are **not** used in defining the projection.
- The b_i values used to define this linear combination are **constants**.
  - We use the same b_i values for computing any backprojection.

Why Add the $b_i \boldsymbol{u}_i$ Terms?

$$B\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i] + \sum_{i=M+1}^{D} [b_i \boldsymbol{u}_i]$$

- In the figure, we can set $b_2 = \frac{1}{N}\sum_{n=1}^{N}[(\boldsymbol{u}_2)^T \boldsymbol{x}_n]$.
- So, we add the mean of the dot products of the data with $\boldsymbol{u}_2$.
- You can compare:
  - $B_1(z_1) = z_1 \boldsymbol{u}_1$ (green circles).
  - $B(z_1) = z_1 \boldsymbol{u}_1 + b_2 \boldsymbol{u}_2$ (red circles).
- The results of B are closer to the original data.



Minimum-Error Formulation

- The way we will actually define the backprojection function B is:

$$B\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i] + \sum_{i=M+1}^{D} [b_i \boldsymbol{u}_i]$$

- In the previous figure, we set $b_2 = \frac{1}{N}\sum_{n=1}^{N}[(\boldsymbol{u}_2)^T \boldsymbol{x}_n]$ and it worked well.
- We will now show that that this formula is optimal.
- However, we first need to define what parameters we want to optimize over, and what our optimization criterion is.

Minimum-Error Formulation

$$F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \\ \dots \\ (\boldsymbol{u}_M)^T \boldsymbol{x} \end{bmatrix}, \qquad B\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i] + \sum_{i=M+1}^{D} [b_i \boldsymbol{u}_i]$$

- To fully define these functions, we need to choose:
  - The $\boldsymbol{u}_i$ vectors, for $i = 1$ to M.
  - The b_i constants , for $i = M + 1$ to D.
- Note that M itself is an input to the algorithm (the user gets to specify that).
- The parameters we need to choose are shown in red in the formula.

Backprojection Error

$$F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \\ \dots \\ (\boldsymbol{u}_M)^T \boldsymbol{x} \end{bmatrix}, \qquad B\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i] + \sum_{i=M+1}^{D} [b_i \boldsymbol{u}_i]$$

- What we want to minimize is the average squared distance between an original vector $\boldsymbol{x}$, and the backprojection of the projection of $\boldsymbol{x}$.

$$J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_D, b_{\boldsymbol{M}+1}, \dots, b_{\boldsymbol{N}}) = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - B(F(\boldsymbol{x}_n))\|^2$$

- Function J is called **distortion measure** or **backprojection error.**

Computing the b_i's

$$F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \\ \dots \\ (\boldsymbol{u}_M)^T \boldsymbol{x} \end{bmatrix}, \qquad B\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i] + \sum_{i=M+1}^{D} [b_i \boldsymbol{u}_i]$$

$$J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \dots, b_{\boldsymbol{N}}) = \frac{1}{N} \sum_{n=1}^{N} \|\boldsymbol{x}_n - B(F(\boldsymbol{x}_n))\|^2$$

- To compute the b_i's, we solve for each b_i the equation $\frac{\partial J}{\partial b_i} = 0$.
- In solving this equation, it will be useful to remember that:

$$\boldsymbol{x}_n = \sum_{i=1}^{D} [(\boldsymbol{u}_i)^T \boldsymbol{x}\, \boldsymbol{u}_i]$$

First Step: Simplifying Error J

$$J(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \ldots, b_{\boldsymbol{N}}) = \frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - B(F(\boldsymbol{x}_n))\|^2 =$$

$$\frac{1}{N}\sum_{n=1}^{N} \left\| \sum_{j=1}^{D} \left[\left(\boldsymbol{u}_j\right)^T \boldsymbol{x}_n \boldsymbol{u}_j \right] - \left(\sum_{j=1}^{M} \left[\left(\boldsymbol{u}_j\right)^T \boldsymbol{x}_n \boldsymbol{u}_j \right] + \sum_{j=M+1}^{D} \left[b_j \boldsymbol{u}_j \right] \right) \right\|^2 =$$

$$\frac{1}{N}\sum_{n=1}^{N} \left\| \sum_{j=1}^{M} \left[\left(\boldsymbol{u}_j\right)^T \boldsymbol{x}_n \boldsymbol{u}_j - \left(\boldsymbol{u}_j\right)^T \boldsymbol{x}_n \boldsymbol{u}_j \right] + \sum_{j=M+1}^{D} \left[\left(\boldsymbol{u}_j\right)^T \boldsymbol{x}_n \boldsymbol{u}_j - b_j \boldsymbol{u}_j \right] \right\|^2$$

$$= \frac{1}{N}\sum_{n=1}^{N} \left\| \sum_{j=M+1}^{D} \left[\left(\boldsymbol{u}_j\right)^T \boldsymbol{x}_n \boldsymbol{u}_j - b_j \boldsymbol{u}_j \right] \right\|^2$$

Second Step: Computing $\frac{\partial J}{\partial b_i}$

$$\frac{\partial J(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \ldots, b_{\boldsymbol{N}})}{\partial b_i} =$$

$$\frac{\partial \left(\frac{1}{N} \sum_{n=1}^{N} \left\| \sum_{j=M+1}^{D} \left[\left(\boldsymbol{u}_j \right)^T \boldsymbol{x}_n \boldsymbol{u}_j - b_j \boldsymbol{u}_j \right] \right\|^2 \right)}{\partial b_i} =$$

- In the last formula, in the $\sum_{j=M+1}^{D}$ summation, the only term that is influenced by b_i is the term we obtain when $j = i$. So, we get:

Second Step: Computing $\frac{\partial J}{\partial b_i}$

$$\frac{\partial J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \dots, b_{\boldsymbol{N}})}{\partial b_i} =$$

$$\frac{\partial \left(\frac{1}{N} \sum_{n=1}^{N} \left\| \sum_{j=M+1}^{D} \left[\left(\boldsymbol{u}_j \right)^T \boldsymbol{x}_n \boldsymbol{u}_j - b_j \boldsymbol{u}_j \right] \right\|^2 \right)}{\partial b_i} =$$

$$\frac{\partial \left(\frac{1}{N} \sum_{n=1}^{N} \| (\boldsymbol{u}_i)^T \boldsymbol{x}_n \boldsymbol{u}_i - b_i \boldsymbol{u}_i \|^2 \right)}{\partial b_i} =$$

$$\frac{\partial \left(\frac{1}{N} \sum_{n=1}^{N} \left(\left((\boldsymbol{u}_i)^T \boldsymbol{x}_n \right)^2 + {b_i}^2 - 2 (\boldsymbol{u}_i)^T \boldsymbol{x}_n b_i \right) \right)}{\partial b_i} =$$

$$\frac{1}{N} \sum_{n=1}^{N} (2 b_i - 2 (\boldsymbol{u}_i)^T \boldsymbol{x}_n)$$

Third Step: Solving $\frac{\partial J}{\partial b_i} = 0$

$$\frac{\partial J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \dots, b_{\boldsymbol{N}})}{\partial b_i} = 0 \Rightarrow$$

$$\frac{1}{N}\sum_{n=1}^{N}(2b_i - 2(\boldsymbol{u}_i)^T\boldsymbol{x}_n) = \boldsymbol{0} \Rightarrow \frac{2}{N}\sum_{n=1}^{N}(b_i - (\boldsymbol{u}_i)^T\boldsymbol{x}_n) = \boldsymbol{0} \Rightarrow$$

$$\frac{2}{N}\sum_{n=1}^{N}(b_i) - \frac{2}{N}\sum_{n=1}^{N}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_n\right) = \boldsymbol{0} \Rightarrow \frac{2Nb_i}{N} - \frac{2}{N}\sum_{n=1}^{N}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_n\right) = \boldsymbol{0} \Rightarrow$$

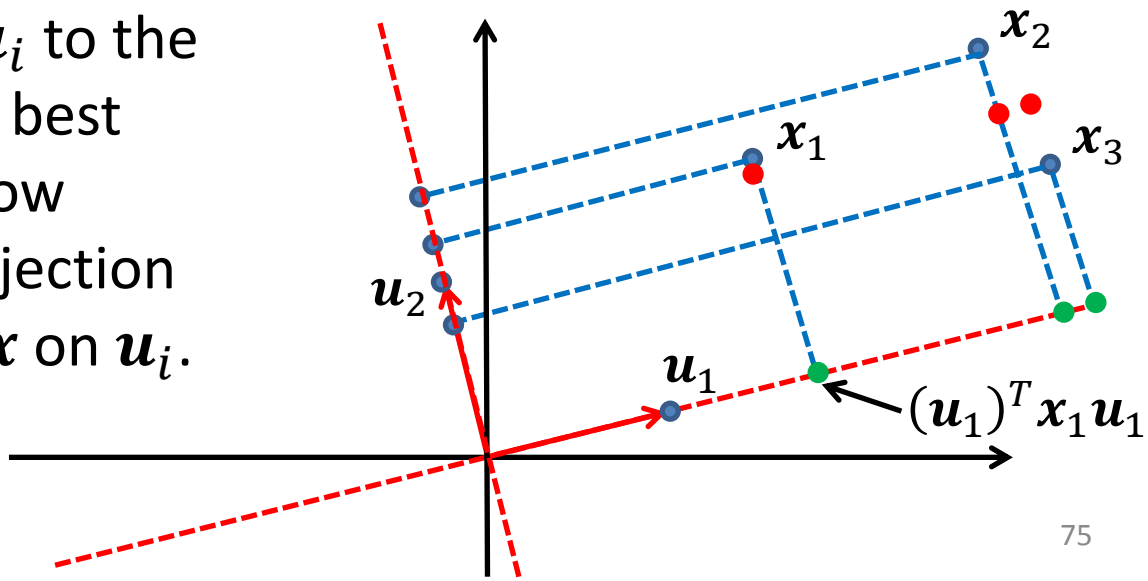
$$\frac{2Nb_i}{N} = \frac{2}{N}\sum_{n=1}^{N}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_n\right) \Rightarrow b_i = \frac{1}{N}\sum_{n=1}^{N}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_n\right)$$

Third Step: Solving $\frac{\partial J}{\partial b_i} = 0$

- We have concluded that

$$\frac{\partial J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \dots, b_{\boldsymbol{N}})}{\partial b_i} = 0 \Rightarrow b_i = \frac{1}{N} \sum_{n=1}^{N} \left((\boldsymbol{u}_i)^T \boldsymbol{x}_n \right)$$

- Thus, each b_i should be set to the mean of the projections of all data on vector $\boldsymbol{u}_i$.
- Essentially, adding $b_i \boldsymbol{u}_i$ to the backprojection is "our best guess", if we know know nothing about the projection of the original vector $\boldsymbol{x}$ on $\boldsymbol{u}_i$.



Picking Vectors $\boldsymbol{u}_i$

- We have concluded that

$$\frac{\partial J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \dots, b_{\boldsymbol{N}})}{\partial b_i} = 0 \Rightarrow b_i = \frac{1}{N} \sum_{n=1}^{N} \left((\boldsymbol{u}_i)^T \boldsymbol{x}_n \right)$$

- Thus, each b_i should be set to the mean of the projections of all data on vector $\boldsymbol{u}_i$.
- Next question: how should we pick vectors $\boldsymbol{u}_i$?
- We saw before that, under the criterion of maximizing the variance of projected data, $\boldsymbol{u}_1, \dots, \boldsymbol{u}_M$ should be the eigenvectors with the highest M eigenvalues.
- We will see now that, under the criterion of minimizing the backprojection error, we get the same result:
  - $\boldsymbol{u}_1, \dots, \boldsymbol{u}_M$ should be the eigenvectors with the highest M eigenvalues

Picking Vectors $\boldsymbol{u}_i$

- We saw earlier that:

$$J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \dots, b_{\boldsymbol{N}}) = \frac{1}{N}\sum_{n=1}^{N}\left\|\sum_{i=M+1}^{D}[(\boldsymbol{u}_i)^T\boldsymbol{x}_n\boldsymbol{u}_i - b_i\boldsymbol{u}_i]\right\|^2$$

- Also, at some earlier point we had defined $\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n$.
- Based on that, we can further simplify J, as:

$$J(\boldsymbol{u}_1, \dots, \boldsymbol{u}_M, b_{\boldsymbol{M+1}}, \dots, b_{\boldsymbol{N}}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}((\boldsymbol{u}_i)^T\boldsymbol{x}_n - b_i)^2 =$$

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_n - \frac{1}{N}\sum_{k=1}^{N}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_k\right)\right)^2$$

Picking Vectors $\boldsymbol{u}_i$

$$J(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \ldots, b_{\boldsymbol{N}}) =$$

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_n - \frac{1}{N}\sum_{k=1}^{N}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_k\right)\right)^2$$

$$=$$

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D}\left((\boldsymbol{u}_i)^T\boldsymbol{x}_n - (\boldsymbol{u}_i)^T\overline{\boldsymbol{x}}\right)^2 = \sum_{i=M+1}^{D}\left((\boldsymbol{u}_i)^T\boldsymbol{S}\boldsymbol{u}_i\right)^2$$

where $\boldsymbol{S}$ is again the covariance matrix of data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.

Picking Vectors $\boldsymbol{u}_i$

$$J(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D, b_{\boldsymbol{M}+1}, \ldots, b_{\boldsymbol{N}}) = \sum_{i=M+1}^{D} \left((\boldsymbol{u}_i)^T \boldsymbol{S} \boldsymbol{u}_i \right)^2$$

- As before, we want to minimize J with respect to each $\boldsymbol{u}_i$.
- We have several constraints:
  - Each $\boldsymbol{u}_i$ is a unit vector. Therefore, $(\boldsymbol{u}_i)^T \boldsymbol{u}_i = 1$
  - For any i, j such that i $\neq j$, $\boldsymbol{u}_i$ is orthogonal to $\boldsymbol{u}_j$. Therefore, $(\boldsymbol{u}_i)^T \boldsymbol{u}_j = 0$.
- To understand how to solve this problem, we will first consider an easy case, where $D = 2$ and $M = 1$.
  - We want to project 2-dimensional vectors onto 1-dimensional values.
- We want to choose $\boldsymbol{u}_2$ so as to minimize $(\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2$, under the constraint that $(\boldsymbol{u}_2)^T \boldsymbol{u}_2 = 1$.

Picking Vectors $\boldsymbol{u}_i$ - 2D Case

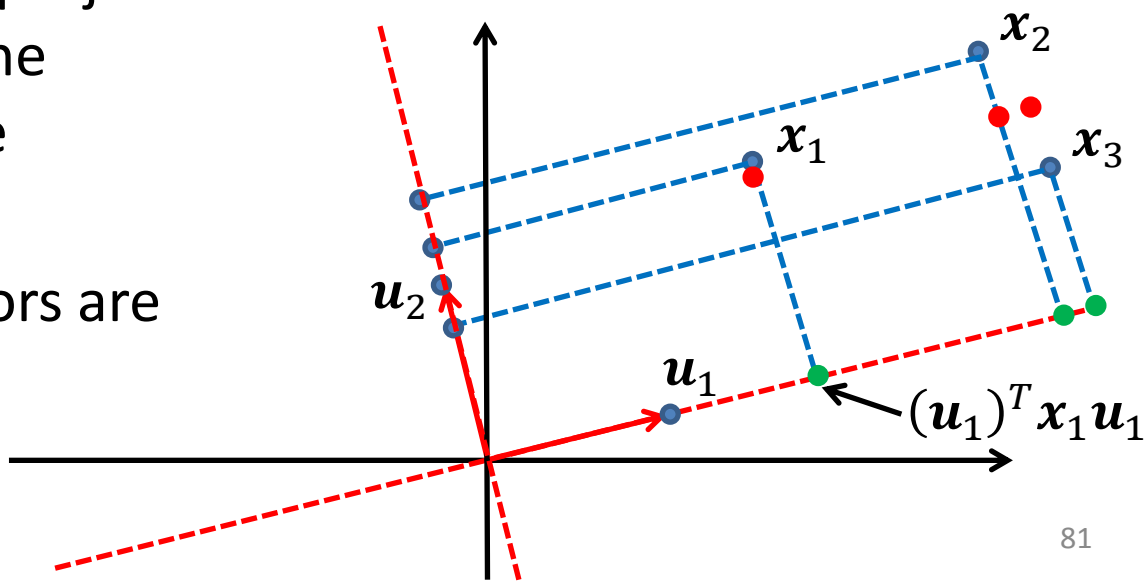
- We want to choose $\boldsymbol{u}_2$ so as to minimize $(\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2$, under the constraint that $(\boldsymbol{u}_2)^T \boldsymbol{u}_2 = 1$.
- Note that $\boldsymbol{u}_2$ is the dimension that we do NOT use for the projection.
- If we find the appropriate $\boldsymbol{u}_2$, then its orthogonal unit vector $\boldsymbol{u}_1$ will be used to define the projection.
- In general, the error J depends on the $\boldsymbol{u}_i$'s that do NOT define the projection:

$$J(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_D, b_{\boldsymbol{M+1}}, \ldots, b_{\boldsymbol{N}}) = \sum_{i=M+1}^{D} \left((\boldsymbol{u}_i)^T \boldsymbol{S} \boldsymbol{u}_i \right)^2$$

- Why is that?
  - The vectors that define the projection do not contribute to the error.

Picking Vectors $\boldsymbol{u}_i$ - 2D Case

- We want to choose $\boldsymbol{u}_2$ so as to minimize $(\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2$, under the constraint that $(\boldsymbol{u}_2)^T \boldsymbol{u}_2 = 1$.
- Note that $\boldsymbol{u}_2$ is the dimension that we do NOT use for the projection.
- If we find the appropriate $\boldsymbol{u}_2$, then its orthogonal unit vector $\boldsymbol{u}_1$ will be used to define the projection.
- In the figure, the backprojections agree perfectly with the original data along the direction of $\boldsymbol{u}_1$.
- All backprojection errors are in the direction of $\boldsymbol{u}_2$.



Picking Vectors $\boldsymbol{u_i}$ - 2D Case

- We want to choose $\boldsymbol{u}_2$ so as to minimize $J = (\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2$, under the constraint that $(\boldsymbol{u}_2)^T \boldsymbol{u}_2 = 1$.
- We can define the Lagrangian as follows:

$$\tilde{J} = (\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2 + \lambda_2 (1 - (\boldsymbol{u}_2)^T \boldsymbol{u}_2)$$

- The gradient of the Lagrangian with respect to $\boldsymbol{u}_2$ is:

$$2\boldsymbol{S}\boldsymbol{u}_2 - 2\lambda_2 \boldsymbol{u}_2$$

- Setting the gradient to 0, we get: $\boldsymbol{S}\boldsymbol{u}_2 = \lambda_2 \boldsymbol{u}_2$.
- So, we know that $\boldsymbol{u}_2$ must be one of the eigenvectors of $\boldsymbol{S}$.
- However, $\boldsymbol{S}$ is a 2×2 matrix, so it usually has two eigenvectors.
- Which eigenvector should be $\boldsymbol{u}_2$?

Picking Vectors $\boldsymbol{u}_i$ - 2D Case

- We want to choose $\boldsymbol{u}_2$ so as to minimize $J = (\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2$.
- We computed that $\boldsymbol{S}\boldsymbol{u}_2 = \lambda_2 \boldsymbol{u}_2$.
- Then, we can rewrite the backpropagation error J as:

$$J = (\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2 = (\boldsymbol{u}_2)^T \lambda_2 \boldsymbol{u}_2 = \lambda_2 (\boldsymbol{u}_2)^T \boldsymbol{u}_2 = \lambda_2$$

  - Note that, since $\boldsymbol{u}_2$ is a unit vector, $(\boldsymbol{u}_2)^T \boldsymbol{u}_2 = 1$, and $\lambda_2 (\boldsymbol{u}_2)^T \boldsymbol{u}_2 = \lambda_2$.
- So, the backpropagation error J is equal to the eigenvalue associated with eigenvector $\boldsymbol{u}_2$.
- Based on that, which of the two eigenvectors should we set $\boldsymbol{u}_2$ equal to?

Picking Vectors $\boldsymbol{u}_i$ - 2D Case

- We want to choose $\boldsymbol{u}_2$ so as to minimize $J = (\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2$.
- We computed that $\boldsymbol{S}\boldsymbol{u}_2 = \lambda_2 \boldsymbol{u}_2$.
- Then, we can rewrite the backpropagation error J as:

$$J = (\boldsymbol{u}_2)^T \boldsymbol{S} \boldsymbol{u}_2 = (\boldsymbol{u}_2)^T \lambda_2 \boldsymbol{u}_2 = \lambda_2 (\boldsymbol{u}_2)^T \boldsymbol{u}_2 = \lambda_2$$

- So, the backpropagation error J is equal to the eigenvalue associated with eigenvector $\boldsymbol{u}_2$.
- Based on that, which of the two eigenvectors should we set $\boldsymbol{u}_2$ equal to?
- Since we want to minimize J, $\boldsymbol{u}_2$ should be the eigenvector with the smallest eigenvalue.
- Thus, $\boldsymbol{u}_1$ (which defines the projection) should be the eigenvector with the largest eigenvalue.

Picking Vectors $\boldsymbol{u}_i$ - General Case

- We will not show the proof, but the same finding holds for the general case, where $D > 2$.
- Again, the $\boldsymbol{u}_i$ vectors need to be eigenvectors of covariance matrix $\boldsymbol{S}$.
- In the $D > 2$ case, it turns out that $J = \sum_{i=M+1}^{D} \lambda_i$.

- Again, since we want to minimize J, vectors $\boldsymbol{u}_{M+1}, \ldots, \boldsymbol{u}_D$ (the vectors that do NOT define the projection) should be the eigenvectors of $\boldsymbol{S}$ with the smallest eigenvalues.
- Consequently, vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_M$ that define the projection should be the eigenvectors of $\boldsymbol{S}$ with the largest eigenvalues.

PCA Recap

- Principal Component Analysis (PCA) is a linear method for dimensionality reduction.
- PCA defines a projection function F and a backprojection function B as follows:

$$F(\boldsymbol{x}) = \begin{bmatrix} (\boldsymbol{u}_1)^T \boldsymbol{x} \\ (\boldsymbol{u}_2)^T \boldsymbol{x} \\ \dots \\ (\boldsymbol{u}_M)^T \boldsymbol{x} \end{bmatrix}, B\left(\begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_M \end{bmatrix}\right) = \sum_{i=1}^{M} [z_i \boldsymbol{u}_i] + \sum_{i=M+1}^{D} [b_i \boldsymbol{u}_i]$$

- Vectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_M$, which define F, are the M eigenvectors of the data covariance matrix with the largest eigenvalues.

PCA Recap

- The inputs and constraints for PCA are:
  - Input: the data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$. Changing the data changes the results.
  - Input: the target dimensionality M.
  - Constraint: We want $F(\boldsymbol{x})$ to be linear. Therefore, $F(\boldsymbol{x}) = \boldsymbol{U}\boldsymbol{x}$ for some matrix $\boldsymbol{U}$.
- Given these inputs and constraints, PCA produces the optimal projection and backprojection functions, under all these three criteria:
  - The projection preserves pairwise distances of the original data as much as possible.
  - The projection maximizes the variance of the projected data.
  - The backprojection minimizes the average difference between original data and backprojected data.

Time Complexity of PCA

- Computing the covariance matrix takes time $O(ND^2)$.
- Computing the top M eigenvectors of the covariance matrix takes time $O(MD^2)$.
- So, the overall complexity is $O(MD^2 + ND^2)$
- There is an alternative formulation of PCA (probabilistic PCA), that allows computation of the top M eigenvectors in $O(NMD)$ time.
  - The covariance matrix is not explicitly computed.
  - The top M eigenvectors are computed using an Expectation Maximization (EM) algorithm.

Example Application: PCA on Faces

- In this example, the data $\boldsymbol{x}_i$ are face images, like:

Example Application: PCA on Faces

- Each image has size 31×25:
- Therefore, each image is represented as a 775-dimensional vector.

Visualizing Eigenfaces

- **Eigenfaces** is a computer vision term, referring to the top eigenvectors of a face dataset.
- Here are the top 4 eigenfaces.
- A face can be mapped to only 4 dimensions, by taking its dot product with these 4 eigenfaces.
- We will see in a bit how well that works.




Visualizing Eigenfaces

- **Eigenfaces** is a computer vision term, referring to the top eigenvectors of a face dataset.
- Here are the eigenfaces ranked 5 to 8.



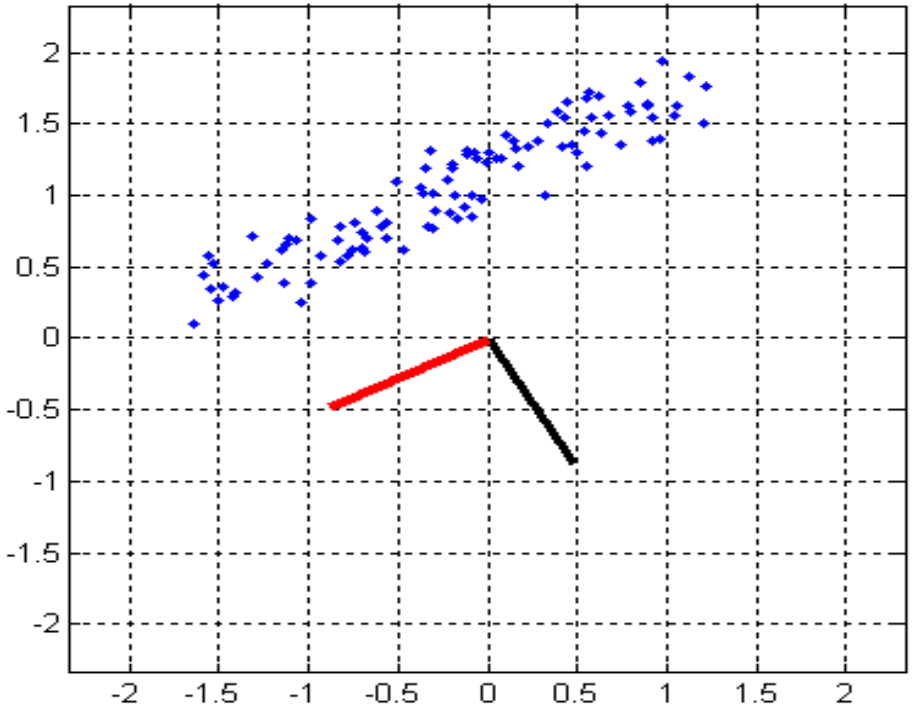

Approximating a Face With 0 Numbers

- What is the best approximation we can get for a face image, if we know nothing about the face image (except that it is a face)?

Equivalent Question in 2D

- What is our best guess for a 2D point from this point cloud, if we know nothing about that 2D point (except that it belongs to the cloud)?

Answer: the average of all points in the cloud.



Approximating a Face With 0 Numbers

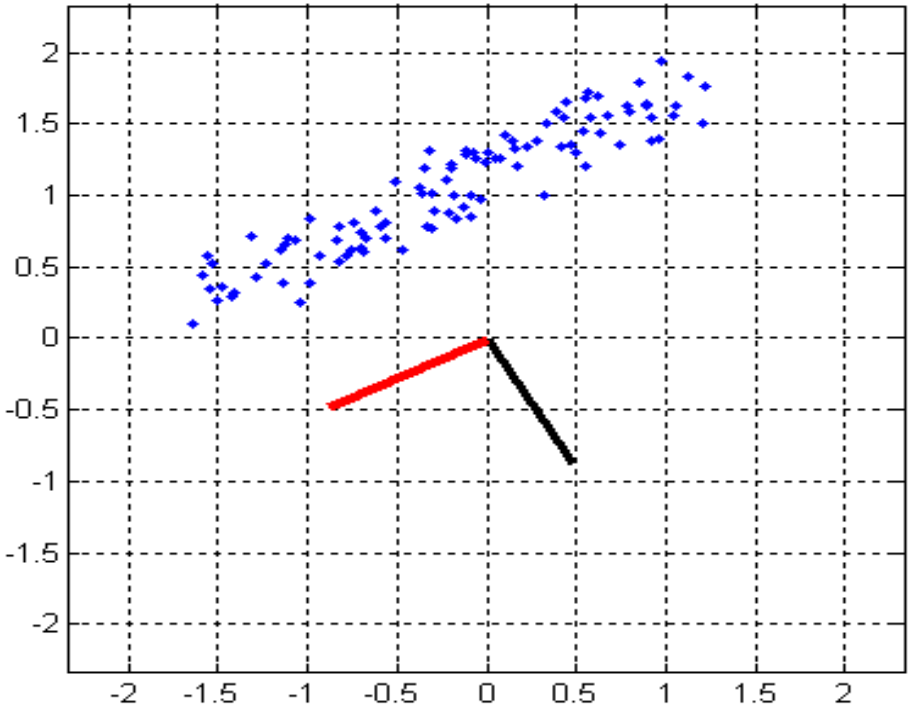
- What is the best approximation we can get for a face image, if we know nothing about the face image (except that it is a face)?

Approximating a Face With 0 Numbers

- What is the best approximation we can get for a face image, if we know nothing about the face image (except that it is a face)?
  - The average face.

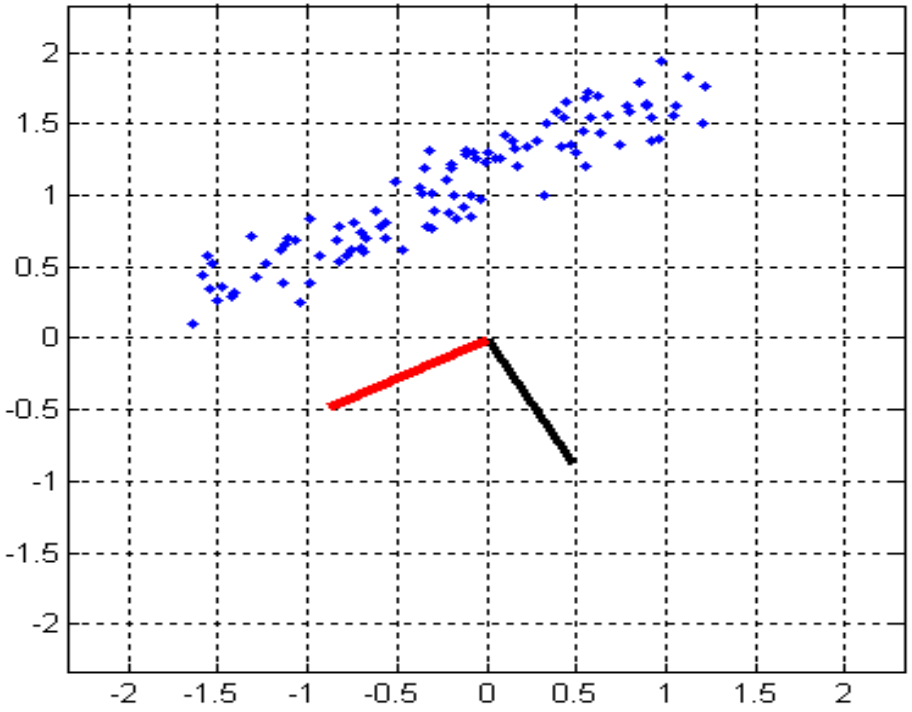
Guessing a 2D Point Given 1 Number

- What is our best guess for a 2D point from this point cloud, if we know nothing about that 2D point, except a single number?
  - What should that number be?



Guessing a 2D Point Given 1 Number

- What is our best guess for a 2D point from this point cloud, if we know nothing about that 2D point, except a single number?
  - What should that number be?
  - Answer: the projection on the first eigenvector.



Approximating a Face With 1 Number

- What is the best approximation we can get for a face image, if we can represent the face with a single number?

Average face

Approximating a Face With 1 Number

- With 0 numbers, we get the average face.
- With 1 number:
  - We map each face to its dot product with the top eigenface.
  - Here is the backprojection of that 1D projection:

Approximating a Face With 2 Numbers

- We map each face to a 2D vector, using its dot products with the top 2 eigenfaces.
- Here is the backprojection of that projection:

Approximating a Face With 3 Numbers

- We map each face to a 3D vector, using its dot products with the top 3 eigenfaces.
- Here is the backprojection of that projection:

Approximating a Face With 4 Numbers

- We map each face to a 4D vector, using its dot products with the top 4 eigenfaces.
- Here is the backprojection of that projection:

Approximating a Face With 5 Numbers

- We map each face to a 5D vector, using its dot products with the top 5 eigenfaces.
- Here is the backprojection of that projection:

Approximating a Face With 6 Numbers

- We map each face to a 6D vector, using its dot products with the top 6 eigenfaces.
- Here is the backprojection of that projection:

Approximating a Face With 7 Numbers

- We map each face to a 7D vector, using its dot products with the top 7 eigenfaces.
- Here is the backprojection of that projection:

Approximating a Face With 10 Numbers

- We map each face to a 10D vector, using its dot products with the top 10 eigenfaces.
- Here is the backprojection of that projection:

Backprojection Results

original

10 eigenfaces

40 eigenfaces

100 eigenfaces

Backprojection Results

original

10 eigenfaces

40 eigenfaces

100 eigenfaces

Backprojection Results

original

10 eigenfaces

40 eigenfaces

100 eigenfaces

Note: teeth not visible using 10 eigenfaces

Backprojection Results

original

10 eigenfaces

40 eigenfaces

100 eigenfaces

Note: using 10 eigenfaces, gaze direction is towards camera

Backprojection Results

original | 10 eigenfaces | 40 eigenfaces | 100 eigenfaces

Note: using 10 eigenfaces, glasses are removed

Projecting a Non-Face

original

10 eigenfaces

40 eigenfaces

100 eigenfaces

The backprojection looks much more like a face than the original image.

Projecting a Half Face

face with no occlusion

occluded bottom half (input to PCA)

reconstruction of picture with occluded bottom half, using 10 eigenfaces

With 10 eigenfaces, the reconstruction looks quite a bit like a regular face.

Projecting a Half Face

face with no occlusion

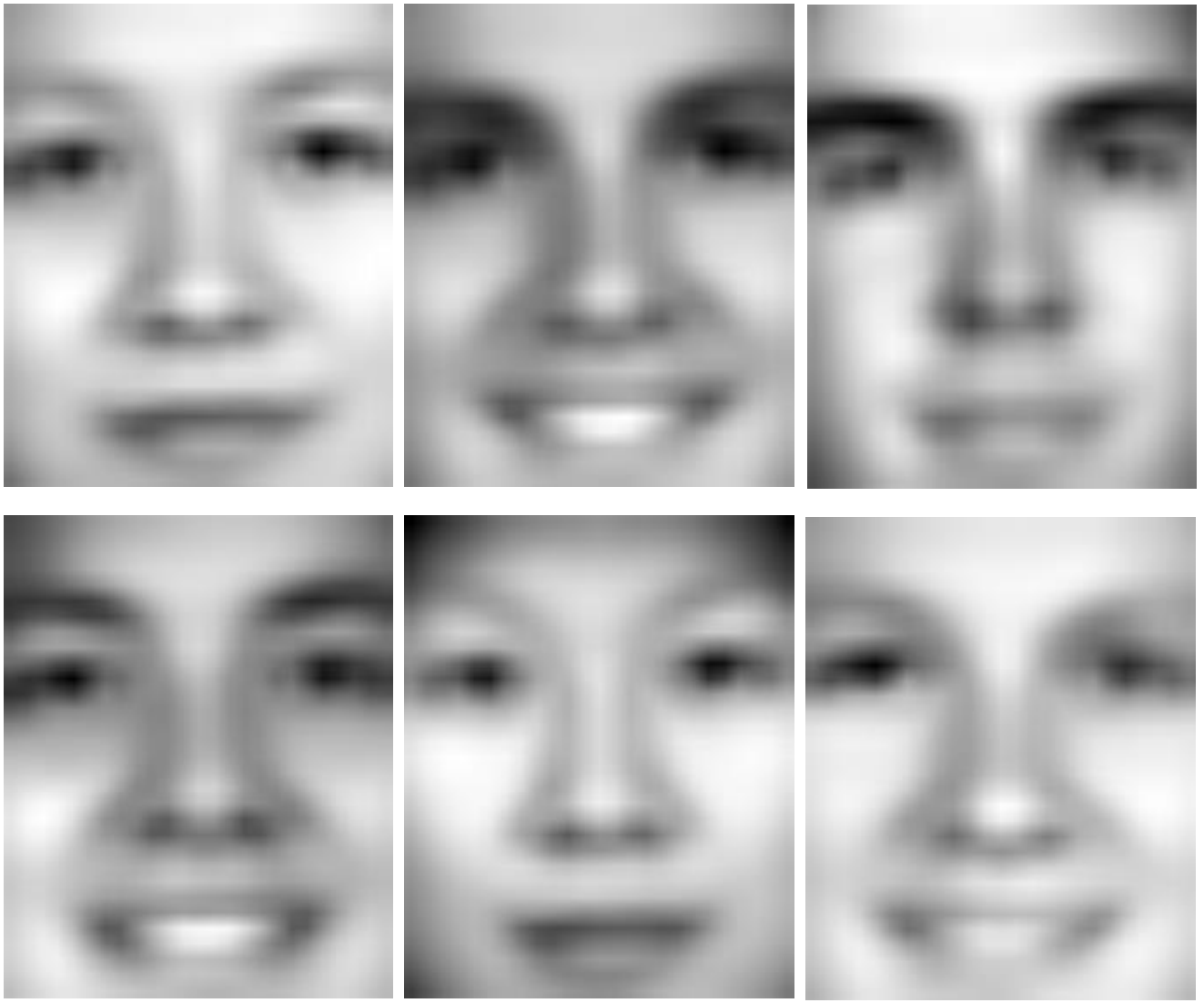
occluded bottom half
(input to PCA projection)

reconstruction of picture with occluded bottom half, using 100 eigenfaces

With 100 eigenfaces, the reconstruction starts resembling the original input to the PCA projection. The more eigenfaces we use, the more the reconstruction resembles the original input.

How Much Can 10 Numbers Tell?

original

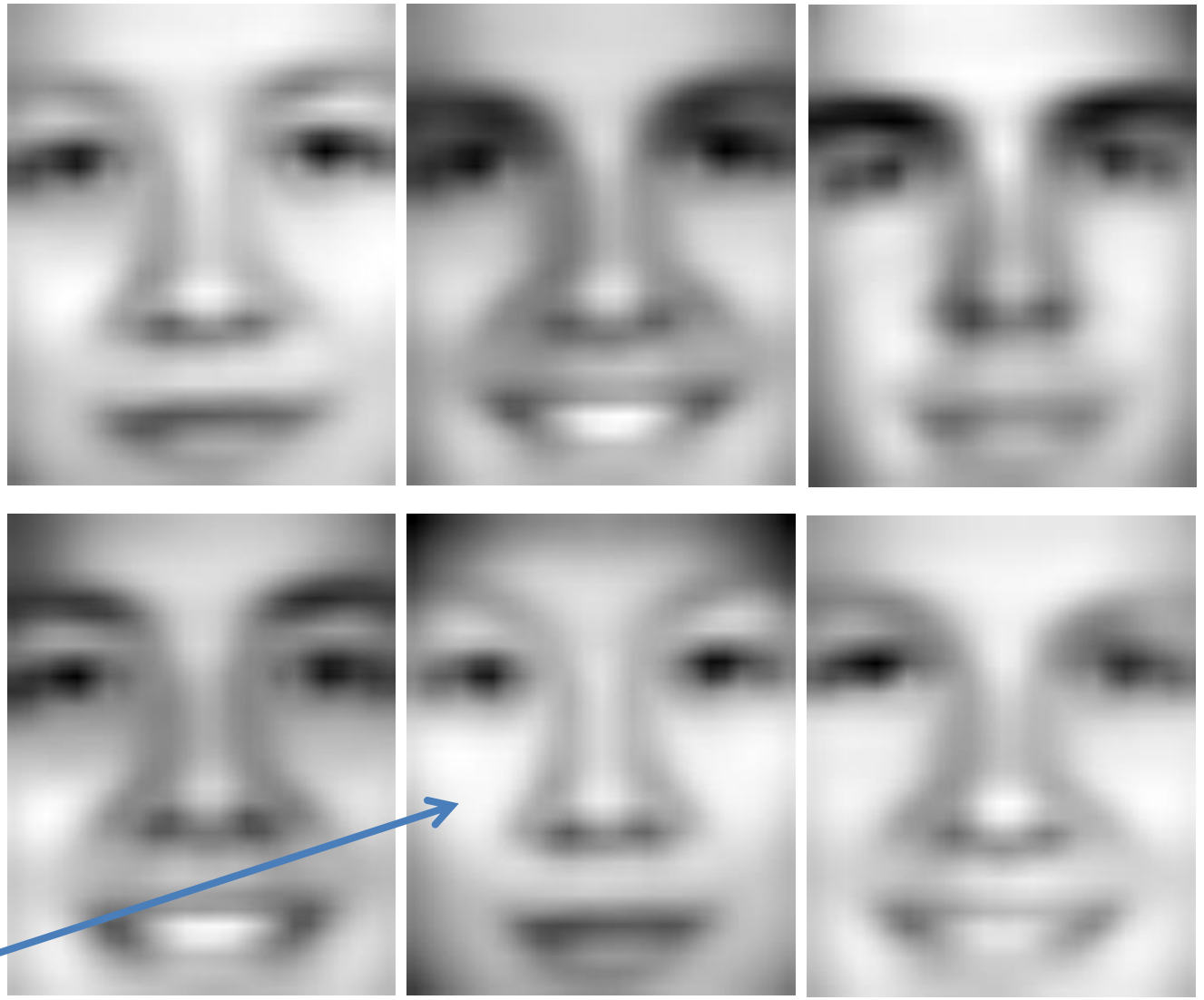


Backprojections of 10-dimensional PCA projections.

Which image on the right shows the same person as the original image?

How Much Can 10 Numbers Tell?

original



Which image on the right shows the same person as the original image?

Backprojections of 10-dimensional PCA projections.

Uses of PCA

- PCA can be a part of many different pattern recognition pipelines.
- It can be used to preprocess the data before learning probability distributions, using for example histograms, Gaussians, or mixtures of Gaussians.
- It can be used as a **basis function**, preprocessing input before applying one of the methods we have studied, like linear regression/classification, neural networks, decision trees, …

PCA Applications: Face Detection

- A common approach for detecting faces is to:
  - Build a classifier that decides, given an image window, if it is a face or not.
  - Apply this classifier on (nearly) every possible window in the image, of every possible center and scale.
  - Return as detected faces the windows where the classifier output is above a certain threshold (or below a certain threshold, depending on how the classifier is defined).

PCA Applications: Face Detection

- A common approach for detecting faces is to:
  - Build a classifier that decides, given an image window, if it is a face or not.
  - Apply this classifier on (nearly) every possible window in the image, of every possible center and scale.
  - Return as detected faces the windows where the classifier output is above a certain threshold (or below a certain threshold, depending on how the classifier is defined).

For example:

- Set scale to 20x20
- Apply the classifier on every image window of that scale.

PCA Applications: Face Detection

- A common approach for detecting faces is to:
  - Build a classifier that decides, given an image window, if it is a face or not.
  - Apply this classifier on (nearly) every possible window in the image, of every possible center and scale.
  - Return as detected faces the windows where the classifier output is above a certain threshold (or below a certain threshold, depending on how the classifier is defined).

- Set scale to 25x25
- Apply the classifier on every image window of that scale.

PCA Applications: Face Detection

- A common approach for detecting faces is to:
  - Build a classifier that decides, given an image window, if it is a face or not.
  - Apply this classifier on (nearly) every possible window in the image, of every possible center and scale.
  - Return as detected faces the windows where the classifier output is above a certain threshold (or below a certain threshold, depending on how the classifier is defined).

- Set scale to 30x30
- Apply the classifier on every image window of that scale.

PCA Applications: Face Detection

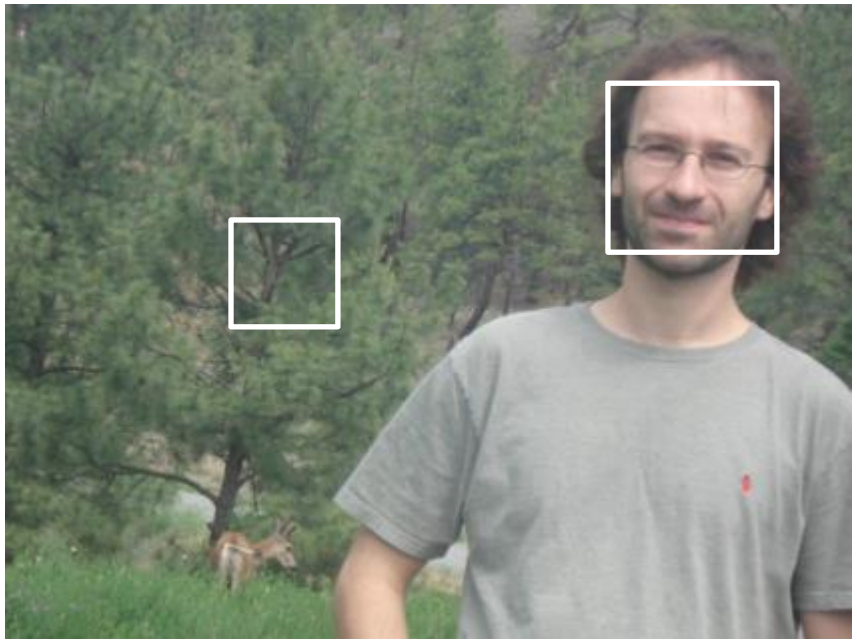
- A common approach for detecting faces is to:
  - Build a classifier that decides, given an image window, if it is a face or not.
  - Apply this classifier on (nearly) every possible window in the image, of every possible center and scale.
  - Return as detected faces the windows where the classifier output is above a certain threshold (or below a certain threshold, depending on how the classifier is defined).

- And so on, up to scales that are as large as the original image.

PCA Applications: Face Detection

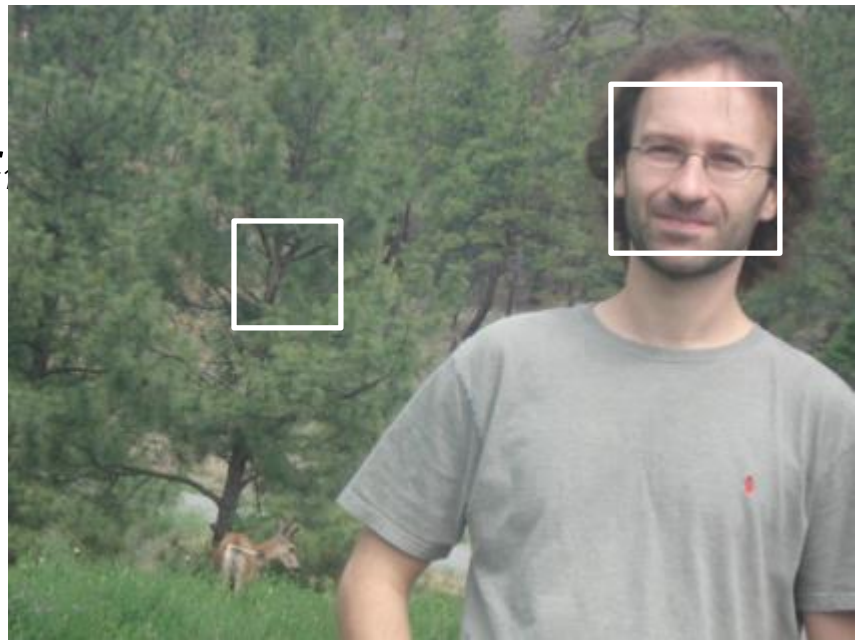
- A common approach for detecting faces is to:
  - Build a classifier that decides, given an image window, if it is a face or not.
  - Apply this classifier on (nearly) every possible window in the image, of every possible center and scale.
  - Return as detected faces the windows where the classifier output is above a certain threshold (or below a certain threshold, depending on how the classifier is defined).

- Finally, output the windows for which the classifier output was above a threshold.
  - Some may be right, some may be wrong.



PCA Applications: Face Detection

- To build this classifier, that determines if a window is a face or not, one approach is to use PCA.
- Training stage: Get a large set of faces, do PCA on them to define a projection F and backprojection B based on the top M eigenvectors, for some value of M.
- Test stage: Let $\boldsymbol{x}$ be the image data from a window:
  - Compute the backprojection error $E = \|\boldsymbol{x}$
  - If $E < t$ (where t is a threshold we can choose using validation data), output that $\boldsymbol{x}$ is a face.
  - If $E \geq t$, output that $\boldsymbol{x}$ is not a face.



PCA Applications: Pattern Classification

- Consider the **satellite** dataset.
  - Each vector in that dataset has 36 dimensions.
- A PCA projection on 2 dimensions keeps 86% of the variance of the training data.
- A PCA projection on 5 dimensions keeps 94% of the variance of the training data.
- A PCA projection on 7 dimensions keeps 97% of the variance of the training data.

PCA Applications: Pattern Classification

- Consider the **satellite** dataset.
- What would work better, from the following options?
  - Bayes classifier using 36-dimensional Gaussians.
  - Bayes classifier using 7-dimensional Gaussians computed from PCA output.
  - Bayes classifier using 5-dimensional Gaussians computed from PCA output.
  - Bayes classifier using 2-dimensional Gaussians computed from PCA output.
  - Bayes classifier using 2-dimensional histogram computed from PCA output.

PCA Applications: Pattern Classification

- We cannot really predict which one would work better, without doing experiments.
- PCA loses some information.
  - That may lead to higher classification error compared to using the original data as input to the classifier.
- Defining Gaussians and histograms on lower-dimensional spaces requires fewer parameters.
  - Those parameters can be estimated more reliably from limited training data.
  - That may lead to lower classification error, compared to using the original data as input to the classifier.

Variations and Alternatives to PCA

- There exist several alternatives for dimensionality reduction.
- We will mention a few, but in very little detail.
- Variations of PCA:
  - Probabilistic PCA.
  - Kernel PCA.
- Alternatives to PCA:
  - Autoencoders.
  - Multidimensional scaling (we will not discuss).
  - Isomap (we will not discuss).
  - Locally linear embedding (we will not discuss).

Probabilistic PCA

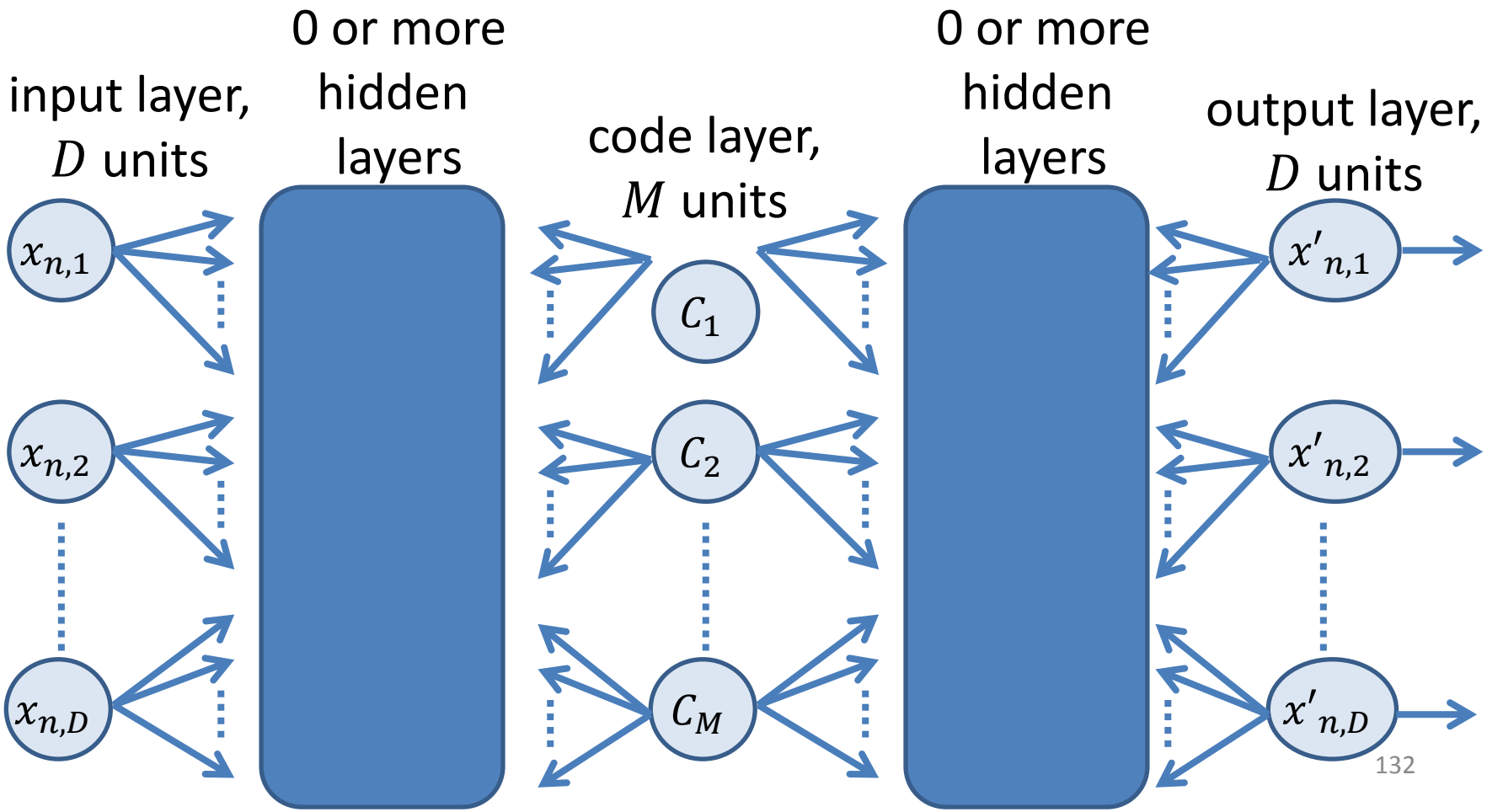
- Reformulates PCA as a problem of finding parameters that maximize the likelihood of the data.
- Leads to a more efficient algorithm (Expectation Maximization) for finding the top M eigenvectors, that takes $O(NMD)$ time.
- Can handle training data that have some **missing values** (i.e., training vectors where for some of their dimensions we do not know the value).

Kernel PCA

- Reformulates PCA so that training data $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ appear only in dot products of the form $(\boldsymbol{x}_n)^T m$.
- This allows us to substitute a kernel instead of using the standard dot product.
- Using a kernel brings similar advantages as in support vector machines.
  - The kernel corresponds to a dot product, but not in the original space.
  - The PCA-defined projection is linear in the kernel space is not necessarily linear in the original space.
  - Provides one way to do **nonlinear dimensionality reduction.**

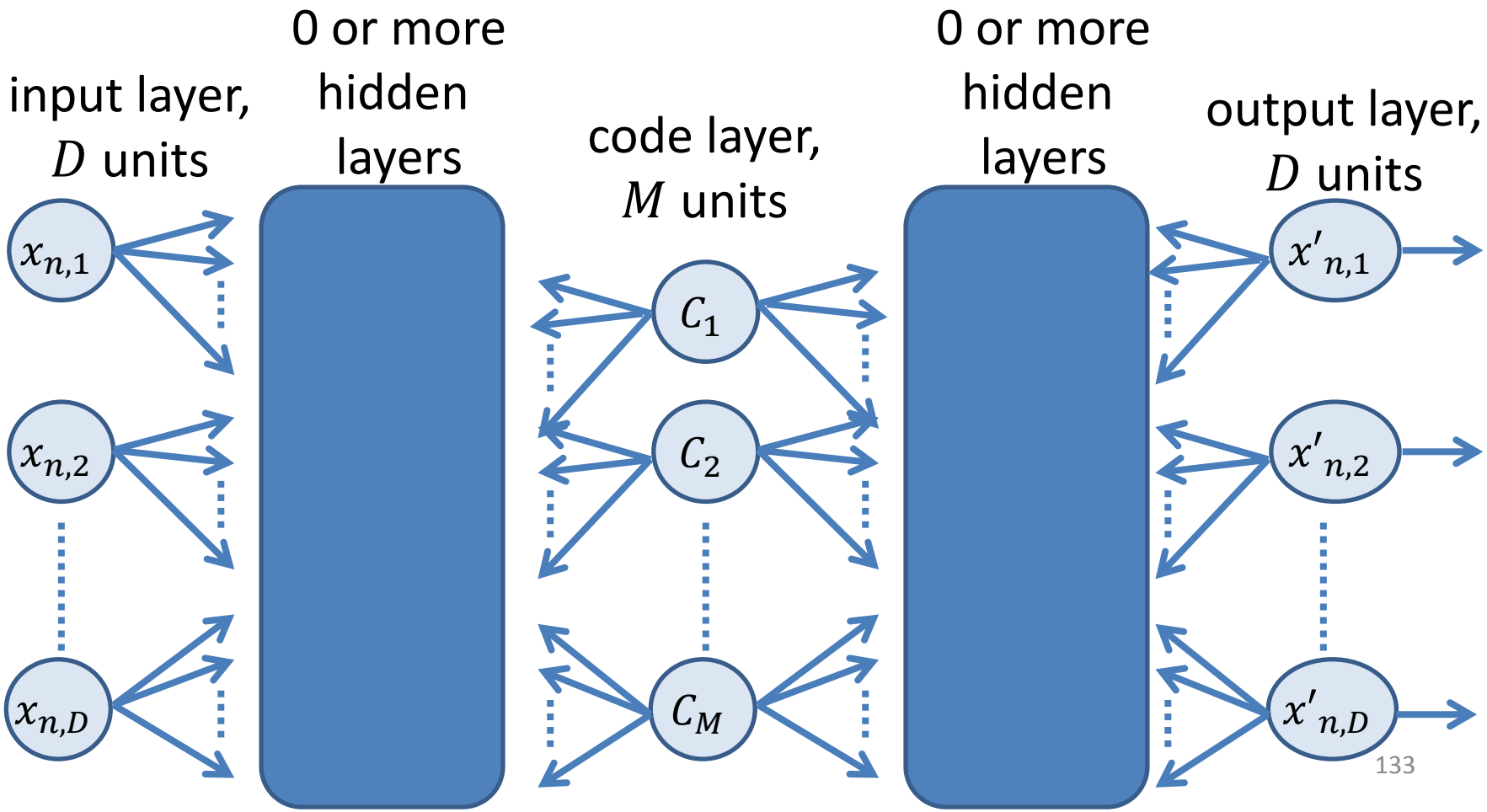
Autoencoders

- 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 $M \ll \mathrm{D}$.



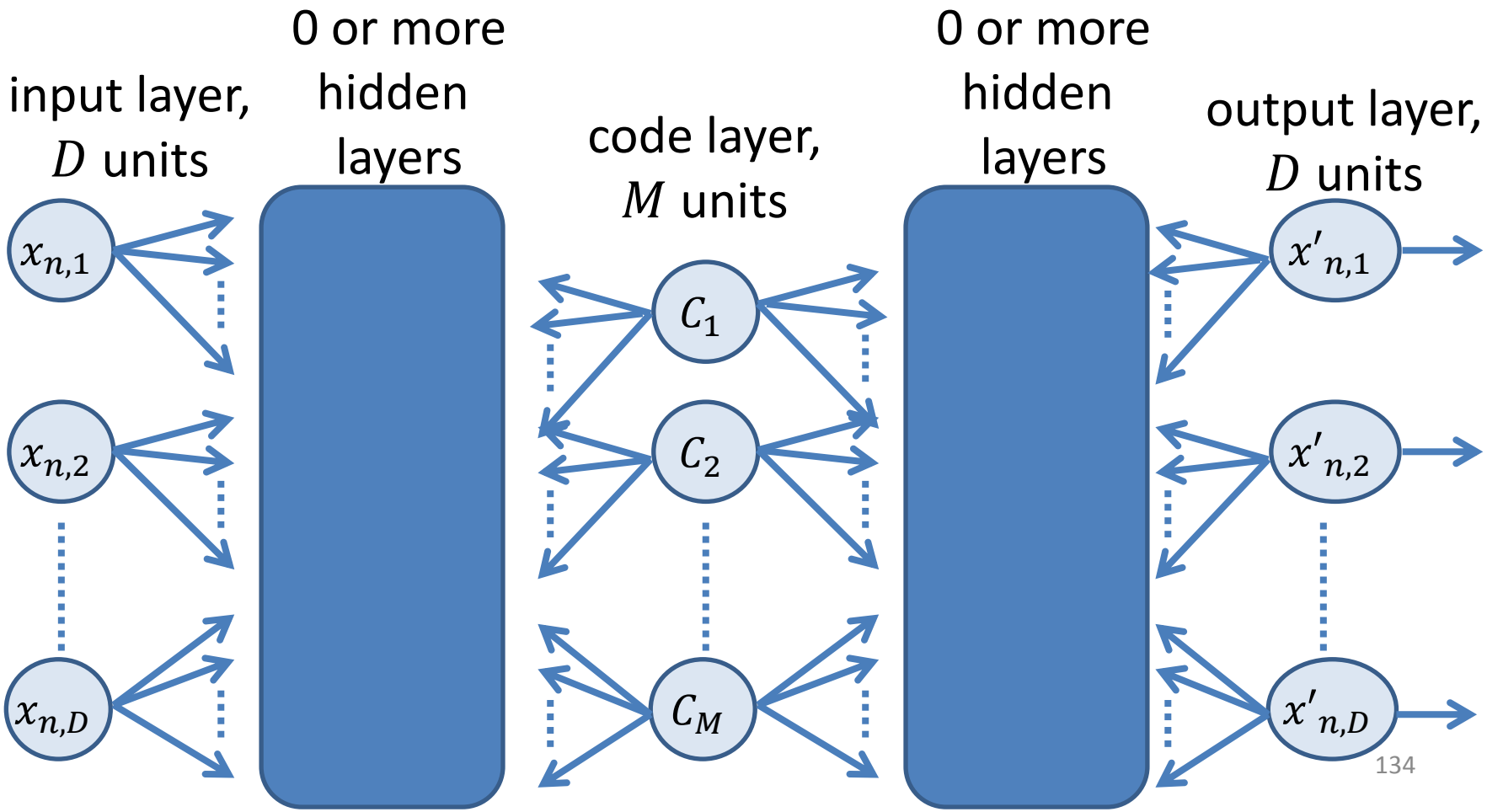
Autoencoders

- 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.



Autoencoders

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