Linear Dimensionality Reduction:
Principal Component Analysis

Piyush Rai

Machine Learning (CS771A)

Sept 2, 2016
Dimensionality Reduction
Dimensionality Reduction

- Usually considered an **unsupervised learning** method
Dimensionality Reduction

- Usually considered an **unsupervised learning** method
- Used for learning the **low-dimensional structures** in the data
Dimensionality Reduction

- Usually considered an *unsupervised learning* method
- Used for learning the *low-dimensional structures* in the data

Also useful for "feature learning" or "representation learning" (learning a better, often smaller-dimensional, representation of the data), e.g.,
Dimensionality Reduction

- Usually considered an **unsupervised learning** method
- Used for learning the **low-dimensional structures** in the data

![Diagram showing dimensionality reduction](image)

- Also useful for **“feature learning”** or **“representation learning”** (learning a better, often smaller-dimensional, representation of the data), e.g.,
  - Documents using topic vectors instead of bag-of-words vectors
Dimensionality Reduction

- Usually considered an *unsupervised learning* method
- Used for learning the *low-dimensional structures* in the data

Also useful for "feature learning" or "representation learning" (learning a better, often smaller-dimensional, representation of the data), e.g.,

- Documents using using topic vectors instead of bag-of-words vectors
- Images using their constituent parts (faces - eigenfaces)
Dimensionality Reduction

- Usually considered an *unsupervised learning* method
- Used for learning the *low-dimensional structures* in the data

Also useful for "feature learning" or "representation learning" (learning a better, often smaller-dimensional, representation of the data), e.g.,
- Documents using using topic vectors instead of bag-of-words vectors
- Images using their constituent parts (faces - eigenfaces)

Can be used for *speeding up* learning algorithms
Dimensionality Reduction

- Usually considered an **unsupervised learning** method
- Used for learning the **low-dimensional structures** in the data

Also useful for **“feature learning”** or **“representation learning”** (learning a better, often smaller-dimensional, representation of the data), e.g.,
- Documents using using topic vectors instead of bag-of-words vectors
- Images using their constituent parts (faces - eigenfaces)

- Can be used for **speeding up** learning algorithms
- Can be used for **data compression**
Curse of Dimensionality

- Exponentially large # of examples required to “fill up” high-dim spaces

\[ D = 1 \quad D = 2 \quad D = 3 \]
Curse of Dimensionality

- Exponentially large # of examples required to “fill up” high-dim spaces

- Fewer dimensions ⇒ Less chances of overfitting ⇒ Better generalization
Curse of Dimensionality

- Exponentially large number of examples required to “fill up” high-dim spaces

- Fewer dimensions $\Rightarrow$ Less chances of overfitting $\Rightarrow$ Better generalization

- Dimensionality reduction is a way to beat the curse of dimensionality
A projection matrix $U = [u_1, u_2, \ldots, u_K]$ of size $D \times K$ defines $K$ linear projection directions, each $u_k \in \mathbb{R}^D$, for the $D$ dim. data (assume $K < D$).
Linear Dimensionality Reduction

- A projection matrix $\mathbf{U} = [\mathbf{u}_1 \ \mathbf{u}_2 \ \ldots \ \mathbf{u}_K]$ of size $D \times K$ defines $K$ linear projection directions, each $\mathbf{u}_k \in \mathbb{R}^D$, for the $D$ dim. data (assume $K < D$)

- Can use $\mathbf{U}$ to transform $\mathbf{x}_n \in \mathbb{R}^D$ into $\mathbf{z}_n \in \mathbb{R}^K$ as shown below

$$
\mathbf{z}_n = \mathbf{U}^\top \mathbf{x}_n = \begin{bmatrix}
\mathbf{u}_1^\top \\
\mathbf{u}_2^\top \\
\vdots \\
\mathbf{u}_K^\top
\end{bmatrix}
\begin{bmatrix}
\mathbf{x}_n
\end{bmatrix}
$$

$\mathbf{z}_n$ is also called low-dimensional "embedding" of $\mathbf{x}_n \in \mathbb{R}^D$.
Linear Dimensionality Reduction

- A projection matrix $U = [u_1 \ u_2 \ \ldots \ u_K]$ of size $D \times K$ defines $K$ linear projection directions, each $u_k \in \mathbb{R}^D$, for the $D$ dim. data (assume $K < D$)

- Can use $U$ to transform $x_n \in \mathbb{R}^D$ into $z_n \in \mathbb{R}^K$ as shown below

$$z_n = U^T x_n = [u_1^T x_n \ u_2^T x_n \ \ldots \ u_K^T x_n]$$

is a $K$-dim projection of $x_n$
Linear Dimensionality Reduction

- A projection matrix $\mathbf{U} = [\mathbf{u}_1 \mathbf{u}_2 \ldots \mathbf{u}_K]$ of size $D \times K$ defines $K$ linear projection directions, each $\mathbf{u}_k \in \mathbb{R}^D$, for the $D$ dim. data (assume $K < D$).

- Can use $\mathbf{U}$ to transform $\mathbf{x}_n \in \mathbb{R}^D$ into $\mathbf{z}_n \in \mathbb{R}^K$ as shown below.

$$\begin{align*}
\mathbf{z}_n &= \mathbf{U}^\top \mathbf{x}_n \\
&= \begin{bmatrix}
\mathbf{u}_1^\top \\
\mathbf{u}_2^\top \\
\vdots \\
\mathbf{u}_K^\top
\end{bmatrix}
\begin{bmatrix}
\mathbf{x}_n \\
\mathbf{x}_n \\
\vdots \\
\mathbf{x}_n
\end{bmatrix}
\end{align*}$$

- Note that $\mathbf{z}_n = \mathbf{U}^\top \mathbf{x}_n = [\mathbf{u}_1^\top \mathbf{x}_n \mathbf{u}_2^\top \mathbf{x}_n \ldots \mathbf{u}_K^\top \mathbf{x}_n]$ is a $K$-dim projection of $\mathbf{x}_n$.

- $\mathbf{z}_n \in \mathbb{R}^K$ is also called low-dimensional “embedding” of $\mathbf{x}_n \in \mathbb{R}^D$. 

Machine Learning (CS771A)
Linear Dimensionality Reduction

- $X = [x_1 \ x_2 \ \ldots \ x_N]$ is $D \times N$ matrix denoting all the $N$ data points.

$Z = [z_1 \ z_2 \ \ldots \ z_N]$ is $K \times N$ matrix denoting embeddings of data points.

With this notation, the figure on the previous slide can be re-drawn as below.

How do we learn the "best" projection matrix $U$?

What criteria should we optimize for when learning $U$?

Principal Component Analysis (PCA) is an algorithm for doing this.
Linear Dimensionality Reduction

- $X = [x_1 \ x_2 \ \ldots \ x_N]$ is $D \times N$ matrix denoting all the $N$ data points
- $Z = [z_1 \ z_2 \ \ldots \ z_N]$ is $K \times N$ matrix denoting embeddings of data points

With this notation, the figure on previous slide can be re-drawn as below.

How do we learn the “best” projection matrix $U$?

What criteria should we optimize for when learning $U$?

Principal Component Analysis (PCA) is an algorithm for doing this.
Linear Dimensionality Reduction

- $X = [x_1 \ x_2 \ \ldots \ x_N]$ is $D \times N$ matrix denoting all the $N$ data points
- $Z = [z_1 \ z_2 \ \ldots \ z_N]$ is $K \times N$ matrix denoting embeddings of data points
- With this notation, the figure on previous slide can be re-drawn as below

How do we learn the "best" projection matrix $U$?

What criteria should we optimize for when learning $U$?

Principal Component Analysis (PCA) is an algorithm for doing this.
Linear Dimensionality Reduction

- $X = [x_1 \ x_2 \ \ldots \ x_N]$ is $D \times N$ matrix denoting all the $N$ data points
- $Z = [z_1 \ z_2 \ \ldots \ z_N]$ is $K \times N$ matrix denoting embeddings of data points
- With this notation, the figure on previous slide can be re-drawn as below

How do we learn the “best” projection matrix $U$?
Linear Dimensionality Reduction

- $X = [x_1 \ x_2 \ \ldots \ x_N]$ is $D \times N$ matrix denoting all the $N$ data points
- $Z = [z_1 \ z_2 \ \ldots \ z_N]$ is $K \times N$ matrix denoting embeddings of data points
- With this notation, the figure on previous slide can be re-drawn as below

- How do we learn the “best” projection matrix $U$?
- What criteria should we optimize for when learning $U$?
Linear Dimensionality Reduction

- \( X = [x_1 \ x_2 \ \ldots \ x_N] \) is a \( D \times N \) matrix denoting all the \( N \) data points.
- \( Z = [z_1 \ z_2 \ \ldots \ z_N] \) is a \( K \times N \) matrix denoting embeddings of data points.

With this notation, the figure on the previous slide can be re-drawn as below:

- How do we learn the “best” projection matrix \( U \)?
- What criteria should we optimize for when learning \( U \)?
- Principal Component Analysis (PCA) is an algorithm for doing this.
Principal Component Analysis
Principal Component Analysis (PCA)

- A classic linear dim. reduction method (Pearson, 1901; Hotelling, 1930)
Principal Component Analysis (PCA)

- A classic linear dim. reduction method (Pearson, 1901; Hotelling, 1930)
- Can be seen as
Principal Component Analysis (PCA)

- A classic linear dim. reduction method (Pearson, 1901; Hotelling, 1930)
- Can be seen as
  - Learning projection directions that capture \textit{maximum variance} in data
- Also related to other classic methods, e.g., Factor Analysis (Spearman, 1904)
Principal Component Analysis (PCA)

- A classic linear dim. reduction method (Pearson, 1901; Hotelling, 1930)
- Can be seen as
  - Learning projection directions that capture maximum variance in data
  - Learning projection directions that result in smallest reconstruction error
- Also related to other classic methods, e.g., Factor Analysis (Spearman, 1904)
A classic linear dim. reduction method (Pearson, 1901; Hotelling, 1930)

Can be seen as

- Learning projection directions that capture maximum variance in data
- Learning projection directions that result in smallest reconstruction error

Can also be seen as changing the basis in which the data is represented (and transforming the features such that new features become decorrelated)
Principal Component Analysis (PCA)

- A classic linear dim. reduction method (Pearson, 1901; Hotelling, 1930)
- Can be seen as
  - Learning projection directions that capture maximum variance in data
  - Learning projection directions that result in smallest reconstruction error
- Can also be seen as changing the basis in which the data is represented (and transforming the features such that new features become decorrelated)

- Also related to other classic methods, e.g., Factor Analysis (Spearman, 1904)
PCA as Maximizing Variance
Variance Captured by Projections

- Consider projecting $x_n \in \mathbb{R}^D$ on a one-dim subspace defined by $u_1 \in \mathbb{R}^D$
- Projection/embedding of $x_n$ along a one-dim subspace $u_1 = u_1^T x_n$ (location of the green point along the purple line representing $u_1$)
Variance Captured by Projections

- Consider projecting $x_n \in \mathbb{R}^D$ on a one-dim subspace defined by $u_1 \in \mathbb{R}^D$

- Projection/embedding of $x_n$ along a one-dim subspace $u_1 = u_1^\top x_n$ (location of the green point along the purple line representing $u_1$)

- Mean of projections of all the data: $\frac{1}{N} \sum_{n=1}^{N} u_1^\top x_n = u_1^\top (\frac{1}{N} \sum_{n=1}^{N} x_n) = u_1^\top \mu$

- $S$ is the $D \times D$ data covariance matrix: $S = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu)(x_n - \mu)^\top$.

  If data already centered ($\mu = 0$) then $S = \frac{1}{N} X^\top X$. 

Machine Learning (CS771A)
Consider projecting \( x_n \in \mathbb{R}^D \) on a one-dim subspace defined by \( u_1 \in \mathbb{R}^D \). Projection/embedding of \( x_n \) along a one-dim subspace \( u_1 = u_1^T x_n \) (location of the green point along the purple line representing \( u_1 \)).

Mean of projections of all the data:

\[
\frac{1}{N} \sum_{n=1}^{N} u_1^T x_n = u_1^T \left( \frac{1}{N} \sum_{n=1}^{N} x_n \right) = u_1^T \mu
\]

Variance of the projected data (“spread” of the green points)

\[
\frac{1}{N} \sum_{n=1}^{N} \left( u_1^T x_n - u_1^T \mu \right)^2 = \frac{1}{N} \sum_{n=1}^{N} \left( u_1^T (x_n - \mu) \right)^2 = u_1^T S u_1
\]

\( S \) is the \( D \times D \) data covariance matrix:

\[
S = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu)(x_n - \mu)^T.
\]

If data already centered (\( \mu = 0 \)) then

\[
S = \frac{1}{N} X^T X = \frac{1}{N} X^2
\]
### Variance Captured by Projections

- Consider projecting $x_n \in \mathbb{R}^D$ on a one-dim subspace defined by $u_1 \in \mathbb{R}^D$
- Projection/embedding of $x_n$ along a one-dim subspace $u_1 = u_1^\top x_n$ (location of the green point along the purple line representing $u_1$)

![Diagram showing projection of data points]

- Mean of projections of all the data: $\frac{1}{N} \sum_{n=1}^{N} u_1^\top x_n = u_1^\top (\frac{1}{N} \sum_{n=1}^{N} x_n) = u_1^\top \mu$
- Variance of the projected data (“spread” of the green points)

$$\frac{1}{N} \sum_{n=1}^{N} (u_1^\top x_n - u_1^\top \mu)^2 = \frac{1}{N} \sum_{n=1}^{N} (u_1^\top (x_n - \mu))^2 = u_1^\top S u_1$$

- $S$ is the $D \times D$ data covariance matrix: $S = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu)(x_n - \mu)^\top$. If data already centered ($\mu = 0$) then $S = \frac{1}{N} \sum_{n=1}^{N} x_n x_n^\top = \frac{1}{N} X^\top X$
Direction of Maximum Variance

We want $u_1$ s.t. the variance of the projected data is maximized

$$\arg \max_{u_1} u_1^T S u_1$$

To prevent trivial solution (max var. = infinite), assume $||u_1|| = 1 = u_1^T u_1$

We will find $u_1$ by solving the following constrained opt. problem

$$\arg \max_{u_1} u_1^T S u_1 + \lambda_1 (1 - u_1^T u_1)$$

where $\lambda_1$ is a Lagrange multiplier
We want $u_1$ s.t. the variance of the projected data is maximized

$$\arg \max_{u_1} u_1^T S u_1$$

To prevent trivial solution (max var. = infinite), assume $||u_1|| = 1 = u_1^T u_1$
Direction of Maximum Variance

- We want \( u_1 \) s.t. the variance of the projected data is maximized
  \[
  \arg \max_{u_1} u_1^\top Su_1
  \]

- To prevent trivial solution (max var. = infinite), assume \( ||u_1|| = 1 = u_1^\top u_1 \)

- We will find \( u_1 \) by solving the following constrained opt. problem
  \[
  \arg \max_{u_1} u_1^\top Su_1 + \lambda_1(1 - u_1^\top u_1)
  \]
where \( \lambda_1 \) is a Lagrange multiplier
Direction of Maximum Variance

- The objective function: \( \arg \max_{u_1} u_1^\top S u_1 + \lambda_1 (1 - u_1^\top u_1) \)

- Taking the derivative w.r.t. \( u_1 \) and setting to zero gives
  \[ Su_1 = \lambda_1 u_1 \]
Direction of Maximum Variance

- The objective function: \(\text{arg max}_{u_1} u_1^\top Su_1 + \lambda_1(1 - u_1^\top u_1)\)

- Taking the derivative w.r.t. \(u_1\) and setting to zero gives
  \[Su_1 = \lambda_1 u_1\]

- Thus \(u_1\) is an eigenvector of \(S\) (with corresponding eigenvalue \(\lambda_1\))
Direction of Maximum Variance

- The objective function: \( \arg \max_{\mathbf{u}_1} \; \mathbf{u}_1^\top \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^\top \mathbf{u}_1) \)

- Taking the derivative w.r.t. \( \mathbf{u}_1 \) and setting to zero gives
  \[
  \mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1
  \]

- Thus \( \mathbf{u}_1 \) is an eigenvector of \( \mathbf{S} \) (with corresponding eigenvalue \( \lambda_1 \))

- But which of \( \mathbf{S}'s \) (\( D \) possible) eigenvectors it is?
Direction of Maximum Variance

The objective function: \( \text{arg max}_{u_1} \ u_1^\top Su_1 + \lambda_1 (1 - u_1^\top u_1) \)

Taking the derivative w.r.t. \( u_1 \) and setting to zero gives
\[
Su_1 = \lambda_1 u_1
\]

Thus \( u_1 \) is an eigenvector of \( S \) (with corresponding eigenvalue \( \lambda_1 \))

But which of \( S \)'s (\( D \) possible) eigenvectors it is?

Note that since \( u_1^\top u_1 = 1 \), the variance of projected data is
\[
u_1^\top Su_1 = \lambda_1
\]
Direction of Maximum Variance

- The objective function: \( \text{arg max}_{u_1} \ u_1^\top S u_1 + \lambda_1 (1 - u_1^\top u_1) \)
- Taking the derivative w.r.t. \( u_1 \) and setting to zero gives
  \[ S u_1 = \lambda_1 u_1 \]
- Thus \( u_1 \) is an eigenvector of \( S \) (with corresponding eigenvalue \( \lambda_1 \))
- But which of \( S \)'s (\( D \) possible) eigenvectors it is?
- Note that since \( u_1^\top u_1 = 1 \), the variance of projected data is
  \[ u_1^\top S u_1 = \lambda_1 \]
- Var. is maximized when \( u_1 \) is the (top) eigenvector with largest eigenvalue
The objective function: \( \arg \max_{\mathbf{u}_1} \; \mathbf{u}_1^\top \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^\top \mathbf{u}_1) \)

Taking the derivative w.r.t. \( \mathbf{u}_1 \) and setting to zero gives

\[
\mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1
\]

Thus \( \mathbf{u}_1 \) is an eigenvector of \( \mathbf{S} \) (with corresponding eigenvalue \( \lambda_1 \))

But which of \( \mathbf{S} \)'s (\( D \) possible) eigenvectors it is?

Note that since \( \mathbf{u}_1^\top \mathbf{u}_1 = 1 \), the variance of projected data is

\[
\mathbf{u}_1^\top \mathbf{S} \mathbf{u}_1 = \lambda_1
\]

Var. is maximized when \( \mathbf{u}_1 \) is the (top) eigenvector with largest eigenvalue

The top eigenvector \( \mathbf{u}_1 \) is also known as the first Principal Component (PC)
Direction of Maximum Variance

- The objective function: \( \arg \max_{\mathbf{u}_1} \mathbf{u}_1^\top \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^\top \mathbf{u}_1) \)

- Taking the derivative w.r.t. \( \mathbf{u}_1 \) and setting to zero gives
  \[
  \mathbf{S} \mathbf{u}_1 = \lambda_1 \mathbf{u}_1
  \]

- Thus \( \mathbf{u}_1 \) is an eigenvector of \( \mathbf{S} \) (with corresponding eigenvalue \( \lambda_1 \))

- But which of \( \mathbf{S}'s \) (\( D \) possible) eigenvectors it is?

- Note that since \( \mathbf{u}_1^\top \mathbf{u}_1 = 1 \), the variance of projected data is
  \[
  \mathbf{u}_1^\top \mathbf{S} \mathbf{u}_1 = \lambda_1
  \]

- Var. is maximized when \( \mathbf{u}_1 \) is the (top) eigenvector with largest eigenvalue

- The top eigenvector \( \mathbf{u}_1 \) is also known as the first Principal Component (PC)

- Other directions can also be found likewise (with each being orthogonal to all previous ones) using the eigendecomposition of \( \mathbf{S} \) (this is PCA)
Principal Component Analysis

Steps in Principal Component Analysis

- Center the data (subtract the mean $\mu = \frac{1}{N} \sum_{n=1}^{N} x_n$ from each data point)
- Compute the covariance matrix $S$ using the centered data as $S = \frac{1}{N} X X^\top$ (note: $X$ assumed $D \times N$ here)
- Do an eigendecomposition of the covariance matrix $S$
- Take first $K$ leading eigenvectors $\{u_k\}_{k=1}^{K}$ with eigenvalues $\{\lambda_k\}_{k=1}^{K}$
- The final $K$ dim. projection/embedding of data is given by $Z = U^\top X$ where $U = [u_1 \ldots u_K]$ is $D \times K$ and embedding matrix $Z$ is $K \times N$

A word about notation: If $X$ is $N \times D$, then $S = \frac{1}{N} X X^\top$ (needs to be $D \times D$) and the embedding will be computed as $Z = XU$ where $Z$ is $N \times K$.
Principal Component Analysis

**Steps in Principal Component Analysis**

- Center the data (subtract the mean $\mu = \frac{1}{N} \sum_{n=1}^{N} x_n$ from each data point)

- Compute the covariance matrix $S$ using the centered data as $S = \frac{1}{N} XX^\top$ (note: $X$ assumed $D \times N$ here)

- Do an eigendecomposition of the covariance matrix $S$

- Take first $K$ leading eigenvectors $\{u_k\}_{k=1}^{K}$ with eigenvalues $\{\lambda_k\}_{k=1}^{K}$

- The final $K$ dim. projection/embedding of data is given by $Z = U^\top X$ where $U = [u_1 \ldots u_K]$ is $D \times K$ and embedding matrix $Z$ is $K \times N$

A word about notation: If $X$ is $N \times D$, then $S = \frac{1}{N} XX^\top$ (needs to be $D \times D$) and the embedding will be computed as $Z = XU$ where $Z$ is $N \times K$.
Steps in Principal Component Analysis

- Center the data (subtract the mean \( \mu = \frac{1}{N} \sum_{n=1}^{N} x_n \) from each data point)
- Compute the covariance matrix \( S \) using the centered data as

\[
S = \frac{1}{N} XX^T
\]

(note: \( X \) assumed \( D \times N \) here)
Principal Component Analysis

**Steps in Principal Component Analysis**

- Center the data (subtract the mean \( \mu = \frac{1}{N} \sum_{n=1}^{N} x_n \) from each data point)
- Compute the covariance matrix \( S \) using the centered data as
  \[
  S = \frac{1}{N} X X^\top
  \]
  (note: \( X \) assumed \( D \times N \) here)
- Do an eigendecomposition of the covariance matrix \( S \)
Principal Component Analysis

Steps in Principal Component Analysis

- Center the data (subtract the mean $\mu = \frac{1}{N} \sum_{n=1}^{N} x_n$ from each data point)
- Compute the covariance matrix $S$ using the centered data as
  \[ S = \frac{1}{N} XX^\top \]  
  (note: $X$ assumed $D \times N$ here)
- Do an eigendecomposition of the covariance matrix $S$
- Take first $K$ leading eigenvectors $\{u_k\}_{k=1}^{K}$ with eigenvalues $\{\lambda_k\}_{k=1}^{K}$

A word about notation: If $X$ is $N \times D$, then $S = \frac{1}{N} XX^\top$ (needs to be $D \times D$) and the embedding will be computed as $Z = XU$ where $Z$ is $N \times K$. 
Principal Component Analysis

**Steps in Principal Component Analysis**

- Center the data (subtract the mean $\mu = \frac{1}{N} \sum_{n=1}^{N} x_n$ from each data point)
- Compute the covariance matrix $S$ using the centered data as
  \[ S = \frac{1}{N} X X^T \] (note: $X$ assumed $D \times N$ here)
- Do an eigendecomposition of the covariance matrix $S$
- Take first $K$ leading eigenvectors $\{u_k\}_{k=1}^{K}$ with eigenvalues $\{\lambda_k\}_{k=1}^{K}$
- The final $K$ dim. projection/embedding of data is given by
  \[ Z = U^T X \]
  where $U = [u_1 \ldots u_K]$ is $D \times K$ and embedding matrix $Z$ is $K \times N$
Principal Component Analysis

**Steps in Principal Component Analysis**

- Center the data (subtract the mean \( \mu = \frac{1}{N} \sum_{n=1}^{N} x_n \) from each data point)
- Compute the covariance matrix \( S \) using the centered data as
  \[
  S = \frac{1}{N} XX^T
  \]
  (note: \( X \) assumed \( D \times N \) here)
- Do an eigendecomposition of the covariance matrix \( S \)
- Take first \( K \) leading eigenvectors \( \{ u_k \}_{k=1}^{K} \) with eigenvalues \( \{ \lambda_k \}_{k=1}^{K} \)
- The final \( K \) dim. projection/embedding of data is given by
  \[
  Z = U^T X
  \]
  where \( U = [u_1 \ldots u_K] \) is \( D \times K \) and embedding matrix \( Z \) is \( K \times N \)

**A word about notation:** If \( X \) is \( N \times D \), then \( S = \frac{1}{N} X^T X \) (needs to be \( D \times D \)) and the embedding will be computed as \( Z = XU \) where \( Z \) is \( N \times K \)
PCA as Minimizing the Reconstruction Error
Assume *complete* orthonormal basis vectors $u_1, u_2, \ldots, u_D$, each $u_d \in \mathbb{R}^D$. We can represent each data point $x_n \in \mathbb{R}^D$ exactly using this new basis $x_n = \sum_{k=1}^{D} z_{nk} u_k$.

Denoting $z_n = [z_{n1}, z_{n2}, \ldots, z_{nD}]^\top$, $U = [u_1, u_2, \ldots, u_D]$, and using $U^\top U = I_D$, $x_n = U z_n$ and $z_n = U^\top x_n$.

Also note that each component of vector $z_n$ is $z_{nk} = u_k^\top x_n$. 

---

**Machine Learning (CS771A)**

Linear Dimensionality Reduction: Principal Component Analysis
Data as Combination of Basis Vectors

- Assume *complete* orthonormal basis vectors $u_1, u_2, \ldots, u_D$, each $u_d \in \mathbb{R}^D$
- We can represent each data point $x_n \in \mathbb{R}^D$ **exactly** using this new basis

$$x_n = \sum_{k=1}^{D} z_{nk} u_k$$

Denoting $z_n = [z_{n1} \ z_{n2} \ldots z_{nD}]^\top$, $U = [u_1 \ u_2 \ldots u_D]$, and using $U^\top U = I_D$ $x_n = U z_n$ and $z_n = U^\top x_n$ Also note that each component of vector $z_n$ is $z_{nk} = u_k^\top x_n$
Data as Combination of Basis Vectors

- Assume complete orthonormal basis vectors $u_1, u_2, \ldots, u_D$, each $u_d \in \mathbb{R}^D$
- We can represent each data point $x_n \in \mathbb{R}^D$ exactly using this new basis

\[ x_n = \sum_{k=1}^{D} z_{nk} u_k \]

Also note that each component of vector $z_n$ is $z_{nk} = u^\top_k x_n$.
Data as Combination of Basis Vectors

- Assume complete orthonormal basis vectors $u_1, u_2, \ldots, u_D$, each $u_d \in \mathbb{R}^D$
- We can represent each data point $x_n \in \mathbb{R}^D$ exactly using this new basis
  \[ x_n = \sum_{k=1}^{D} z_{nk} u_k \]

  ![Diagram](image.png)

- Denoting $z_n = [z_{n1} \ z_{n2} \ldots z_{nD}]^\top$, $U = [u_1 \ u_2 \ldots u_D]$, and using $U^\top U = I_D$
  \[
  x_n = U z_n \quad \text{and} \quad z_n = U^\top x_n
  \]
Data as Combination of Basis Vectors

- Assume complete orthonormal basis vectors $u_1, u_2, \ldots, u_D$, each $u_d \in \mathbb{R}^D$
- We can represent each data point $x_n \in \mathbb{R}^D$ exactly using this new basis

$$x_n = \sum_{k=1}^{D} z_{nk} u_k$$

Denoting $z_n = [z_{n1} \ z_{n2} \ldots z_{nD}]^\top$, $U = [u_1 \ u_2 \ldots u_D]$, and using $U^\top U = I_D$

$$x_n = U z_n \quad \text{and} \quad z_n = U^\top x_n$$

Also note that each component of vector $z_n$ is $z_{nk} = u_k^\top x_n$
Reconstruction of Data from Projections

- Reconstruction of $x_n$ from $z_n$ will be exact if we use all the $D$ basis vectors.

Let's use $K = 1$ basis vector. Then the one-dim embedding of $x_n$ is $z_n = u_1^\top x_n$ (note: this will just be a scalar).

We can now try "reconstructing" $x_n$ from its embedding $z_n$ as follows:

$\tilde{x}_n = u_1 z_n = u_1 u_1^\top x_n$

Total error or "loss" in reconstructing all the data points:

$L(u_1) = \sum_{n=1}^{N} ||x_n - \tilde{x}_n||^2$

$= \sum_{n=1}^{N} ||x_n - u_1 u_1^\top x_n||^2$
Reconstruction of Data from Projections

- Reconstruction of \( x_n \) from \( z_n \) will be exact if we use all the \( D \) basis vectors.
- Will be approximate if we only use \( K < D \) basis vectors: \( x_n \approx \sum_{k=1}^{K} z_{nk} u_k \).
Reconstruction of Data from Projections

- Reconstruction of $x_n$ from $z_n$ will be exact if we use all the $D$ basis vectors.
- Will be approximate if we only use $K < D$ basis vectors: $x_n \approx \sum_{k=1}^{K} z_{nk} u_k$
- Let’s use $K = 1$ basis vector. Then the one-dim embedding of $x_n$ is
  \[ z_n = u_1^\top x_n \] (note: this will just be a scalar)
Reconstruction of Data from Projections

- Reconstruction of $\mathbf{x}_n$ from $\mathbf{z}_n$ will be exact if we use all the $D$ basis vectors.
- Will be approximate if we only use $K < D$ basis vectors: $\mathbf{x}_n \approx \sum_{k=1}^{K} z_{nk} \mathbf{u}_k$
- Let’s use $K = 1$ basis vector. Then the one-dim embedding of $\mathbf{x}_n$ is:

$$\mathbf{z}_n = \mathbf{u}_1^\top \mathbf{x}_n$$

(note: this will just be a scalar)

- We can now try “reconstructing” $\mathbf{x}_n$ from its embedding $\mathbf{z}_n$ as follows:

$$\tilde{\mathbf{x}}_n = \mathbf{u}_1 \mathbf{z}_n = \mathbf{u}_1 \mathbf{u}_1^\top \mathbf{x}_n$$
Reconstruction of Data from Projections

- Reconstruction of $x_n$ from $z_n$ will be exact if we use all the $D$ basis vectors.
- Will be approximate if we only use $K < D$ basis vectors: $x_n \approx \sum_{k=1}^{K} z_{nk}u_k$
- Let’s use $K = 1$ basis vector. Then the one-dim embedding of $x_n$ is $z_n = u_1^\top x_n$ (note: this will just be a scalar).

We can now try “reconstructing” $x_n$ from its embedding $z_n$ as follows $\tilde{x}_n = u_1z_n = u_1u_1^\top x_n$

Total error or “loss” in reconstructing all the data points

$L(u_1) = \sum_{n=1}^{N} ||x_n - \tilde{x}_n||^2 = \sum_{n=1}^{N} ||x_n - u_1u_1^\top x_n||^2$
Direction with Best Reconstruction

- We want to find $u_1$ that minimizes the reconstruction error

$$L(u_1) = \sum_{n=1}^{N} \left| |x_n - u_1 u_1^T x_n| \right|^2$$

Thus the problem is equivalent to the following maximization

$$\text{arg max } u_1: ||u_1||^2 = 1 \text{ u}_1^T \left( \frac{1}{N} \sum_{n=1}^{N} x_n x_n^T \right) u_1 = \text{arg max } u_1: ||u_1||^2 = 1 \text{ u}_1^T S u_1$$

where $S$ is the covariance matrix of the data (data assumed centered)
Direction with Best Reconstruction

- We want to find $u_1$ that minimizes the reconstruction error

$$L(u_1) = \sum_{n=1}^{N} ||x_n - u_1 u_1^T x_n||^2$$

$$= \sum_{n=1}^{N} \{ x_n^T x_n + (u_1 u_1^T x_n)^T (u_1 u_1^T x_n) - 2x_n^T u_1 u_1^T x_n \}$$

Thus the problem is equivalent to the following maximization

$$\arg \max_{u_1} ||u_1||^2 = 1 \quad u_1 = \arg \max_{u_1} u_1^T S u_1$$

where $S$ is the covariance matrix of the data (data assumed centered)

It's the same objective that we had when we maximized the variance
Direction with Best Reconstruction

- We want to find \( u_1 \) that minimizes the reconstruction error

\[
L(u_1) = \sum_{n=1}^{N} \| x_n - u_1 u_1^T x_n \|^2
\]

\[
= \sum_{n=1}^{N} \{ x_n^T x_n + (u_1 u_1^T x_n)^T (u_1 u_1^T x_n) - 2 x_n^T u_1 u_1^T x_n \}
\]

\[
= \sum_{n=1}^{N} -u_1^T x_n x_n^T u_1 \quad \text{(using } u_1^T u_1 = 1 \text{ and ignoring constants w.r.t. } u_1 )
\]
Direction with Best Reconstruction

- We want to find \( u_1 \) that minimizes the reconstruction error

\[
L(u_1) = \sum_{n=1}^{N} ||x_n - u_1 u_1^T x_n||^2 
\]

\[
= \sum_{n=1}^{N} \{x_n^T x_n + (u_1 u_1^T x_n)^T (u_1 u_1^T x_n) - 2x_n^T u_1 u_1^T x_n\}
\]

\[
= \sum_{n=1}^{N} -u_1^T x_n x_n^T u_1 \quad \text{(using } u_1^T u_1 = 1 \text{ and ignoring constants w.r.t. } u_1)\]

- Thus the problem is equivalent to the following maximization
Direction with Best Reconstruction

- We want to find $u_1$ that minimizes the reconstruction error

$$L(u_1) = \sum_{n=1}^{N} \left| \left| x_n - u_1 u_1^T x_n \right| \right|^2$$

$$= \sum_{n=1}^{N} \left\{ x_n^T x_n + (u_1 u_1^T x_n)^T (u_1 u_1^T x_n) - 2x_n^T u_1 u_1^T x_n \right\}$$

$$= \sum_{n=1}^{N} -u_1^T x_n x_n^T u_1 \quad \text{(using } u_1^T u_1 = 1 \text{ and ignoring constants w.r.t. } u_1)$$

- Thus the problem is equivalent to the following maximization

$$\arg \max_{u_1 : \left| u_1 \right| ^2 = 1} u_1^T \left( \frac{1}{N} \sum_{n=1}^{N} x_n x_n^T \right) u_1 = \arg \max_{u_1 : \left| u_1 \right| ^2 = 1} u_1^T S u_1$$

where $S$ is the covariance matrix of the data (data assumed centered)
Direction with Best Reconstruction

- We want to find $u_1$ that minimizes the reconstruction error

$$L(u_1) = \sum_{n=1}^{N} ||x_n - u_1 u_1^T x_n||^2$$

$$= \sum_{n=1}^{N} \{x_n^T x_n + (u_1 u_1^T x_n)^T (u_1 u_1^T x_n) - 2x_n^T u_1 u_1^T x_n\}$$

$$= \sum_{n=1}^{N} -u_1^T x_n x_n^T u_1 \quad \text{(using } u_1^T u_1 = 1 \text{ and ignoring constants w.r.t. } u_1)$$

- Thus the problem is equivalent to the following maximization

$$\arg \max_{u_1:||u_1||^2=1} u_1^T \left( \frac{1}{N} \sum_{n=1}^{N} x_n x_n^T \right) u_1 = \arg \max_{u_1:||u_1||^2=1} u_1^T S u_1$$

where $S$ is the covariance matrix of the data (data assumed centered)

- It's the same objective that we had when we maximized the variance
How many Principal Components to Use?

- Eigenvalue $\lambda_k$ measures the variance captured by the corresponding PC $u_k$
How many Principal Components to Use?

- Eigenvalue $\lambda_k$ measures the variance captured by the corresponding PC $u_k$
- The “left-over” variance will therefore be

$$\sum_{k=K+1}^{D} \lambda_k$$
How many Principal Components to Use?

- Eigenvalue $\lambda_k$ measures the variance captured by the corresponding PC $u_k$
- The “left-over” variance will therefore be

$$\sum_{k=K+1}^{D} \lambda_k$$

- Can choose $K$ by looking at what fraction of variance is captured by the first $K$ PCs

Another direct way is to look at the spectrum of the eigenvalues plot, or the plot of reconstruction error vs number of PC. Can also use other criteria such as AIC/BIC (or more advanced probabilistic approaches to PCA using nonparametric Bayesian methods).
How many Principal Components to Use?

- Eigenvalue $\lambda_k$ measures the variance captured by the corresponding PC $u_k$
- The “left-over” variance will therefore be
  $$\sum_{k=K+1}^{D} \lambda_k$$
- Can choose $K$ by looking at what fraction of variance is captured by the first $K$ PCs
- Another direct way is to look at the spectrum of the eigenvalues plot, or the plot of reconstruction error vs number of PC
How many Principal Components to Use?

- Eigenvalue $\lambda_k$ measures the variance captured by the corresponding PC $u_k$
- The “left-over” variance will therefore be
  $$\sum_{k=K+1}^{D} \lambda_k$$
- Can choose $K$ by looking at what fraction of variance is captured by the first $K$ PCs
- Another direct way is to look at the spectrum of the eigenvalues plot, or the plot of reconstruction error vs number of PC
- Can also use other criteria such as AIC/BIC (or more advanced probabilistic approaches to PCA using nonparametric Bayesian methods)
PCA as Matrix Factorization

- Note that PCA represents each $x_n$ as $x_n = U z_n$
PCA as Matrix Factorization

- Note that PCA represents each $x_n$ as $x_n = Uz_n$
- When using only $K < D$ components, $x_n \approx Uz_n$
PCA as Matrix Factorization

- Note that PCA represents each $x_n$ as $x_n = Uz_n$
- When using only $K < D$ components, $x_n \approx Uz_n$
- For all the $N$ data points, we can write the same as

$$X \approx UZ$$

where $X$ is $D \times N$, $U$ is $D \times K$ and $Z$ is $K \times N$
PCA as Matrix Factorization

- Note that PCA represents each \( x_n \) as \( x_n = Uz_n \)
- When using only \( K < D \) components, \( x_n \approx Uz_n \)
- For all the \( N \) data points, we can write the same as
  \[ X \approx UZ \]
  where \( X \) is \( D \times N \), \( U \) is \( D \times K \) and \( Z \) is \( K \times N \)

- The above approx. is equivalent to a low-rank matrix factorization of \( X \)
PCA as Matrix Factorization

- Note that PCA represents each $x_n$ as $x_n = Uz_n$
- When using only $K < D$ components, $x_n \approx Uz_n$
- For all the $N$ data points, we can write the same as $X \approx UZ$

where $X$ is $D \times N$, $U$ is $D \times K$ and $Z$ is $K \times N$

- The above approx. is equivalent to a low-rank matrix factorization of $X$
- Also closely related to Singular Value Decomposition (SVD); see next slide
A rank-$K$ SVD approximates a data matrix $X$ as follows: $X \approx U \Lambda V^T$
PCA and SVD

- A rank-$K$ SVD approximates a data matrix $X$ as follows: $X \approx U\Lambda V^T$

U is $D \times K$ matrix with top $K$ left singular vectors of $X$
PCA and SVD

A rank-$K$ SVD approximates a data matrix $X$ as follows: $X \approx U\Lambda V^T$

- $U$ is $D \times K$ matrix with top $K$ left singular vectors of $X$
- $\Lambda$ is a $K \times K$ diagonal matrix (with top $K$ singular values)
PCA and SVD

- A rank-\(K\) SVD approximates a data matrix \(X\) as follows: \(X \approx U \Lambda V^T\)

- \(U\) is a \(D \times K\) matrix with top \(K\) left singular vectors of \(X\)
- \(\Lambda\) is a \(K \times K\) diagonal matrix (with top \(K\) singular values)
- \(V\) is a \(N \times K\) matrix with top \(K\) right singular vectors of \(X\)
PCA and SVD

- A rank-\(K\) SVD approximates a data matrix \(X\) as follows: \(X \approx U\Lambda V^T\)

- \(U\) is \(D \times K\) matrix with top \(K\) left singular vectors of \(X\)
- \(\Lambda\) is a \(K \times K\) diagonal matrix (with top \(K\) singular values)
- \(V\) is \(N \times K\) matrix with top \(K\) right singular vectors of \(X\)
- Rank-\(K\) SVD is based on minimizing the reconstruction error
A rank-$K$ SVD approximates a data matrix $X$ as follows: $X \approx UV^T$

- $U$ is a $D \times K$ matrix with top $K$ left singular vectors of $X$
- $\Lambda$ is a $K \times K$ diagonal matrix (with top $K$ singular values)
- $V$ is a $N \times K$ matrix with top $K$ right singular vectors of $X$
- Rank-$K$ SVD is based on minimizing the reconstruction error $||X - UV^T||$
PCA and SVD

A rank-$K$ SVD approximates a data matrix $X$ as follows: $X \approx U\Lambda V^T$

- $U$ is $D \times K$ matrix with top $K$ left singular vectors of $X$
- $\Lambda$ is a $K \times K$ diagonal matrix (with top $K$ singular values)
- $V$ is $N \times K$ matrix with top $K$ right singular vectors of $X$
- Rank-$K$ SVD is based on minimizing the reconstruction error $\|X - U\Lambda V^T\|$
- PCA is equivalent to the best rank-$K$ SVD after centering the data
PCA: Some Comments

- The idea of approximating each data point as a combination of basis vectors
  \[ x_n \approx \sum_{k=1}^{K} z_{nk} u_k \quad \text{or} \quad X \approx UZ \]

  is also popularly known as “Dictionary Learning” in signal/image processing; the learned basis vectors represent the “Dictionary”
PCA: Some Comments

- The idea of approximating each data point as a combination of basis vectors

\[ x_n \approx \sum_{k=1}^{K} z_{nk} u_k \quad \text{or} \quad X \approx UZ \]

is also popularly known as “Dictionary Learning” in signal/image processing; the learned basis vectors represent the “Dictionary”

- Some examples:
  - Each face in a collection as a combination of a small no of “eigenfaces”
PCA: Some Comments

- The idea of approximating each data point as a combination of basis vectors
  \[ x_n \approx \sum_{k=1}^{K} z_{nk} u_k \quad \text{or} \quad X \approx UZ \]
  is also popularly known as “Dictionary Learning” in signal/image processing; the learned basis vectors represent the “Dictionary”

- Some examples:
  - Each face in a collection as a combination of a small no of “eigenfaces”
    \[ X \ (DxN) \ \overset{\mathbb{R}}{\rightarrow} \ U \ (DxK) \ \overset{Z\ (KxN)}{\Rightarrow} \ \begin{pmatrix} Z_1 & \cdots & Z_N \end{pmatrix} \]
  - Each document in a collection as a comb. of a small no of “topics”
The idea of approximating each data point as a combination of basis vectors

\[ x_n \approx \sum_{k=1}^{K} z_{nk} u_k \quad \text{or} \quad X \approx UZ \]

is also popularly known as “Dictionary Learning” in signal/image processing; the learned basis vectors represent the “Dictionary”

Some examples:

- Each face in a collection as a combination of a small no of “eigenfaces”

\[
\begin{pmatrix}
X (DxN)
\end{pmatrix}
\rightarrow
\begin{pmatrix}
U (DxK)
\end{pmatrix}
\begin{pmatrix}
Z (KxN)
\end{pmatrix}
\]

- Each document in a collection as a comb. of a small no of “topics”

- Each gene-expression sample as a comb. of a small no of “genetic pathways”
PCA: Some Comments

- The idea of approximating each data point as a combination of basis vectors
  \[ x_n \approx \sum_{k=1}^{K} z_{nk} u_k \quad \text{or} \quad X \approx UZ \]

is also popularly known as “Dictionary Learning” in signal/image processing; the learned basis vectors represent the “Dictionary”

- Some examples:
  - Each face in a collection as a combination of a small no of “eigenfaces”
    \[
    \begin{pmatrix}
    \vdots & \vdots & \vdots \\
    X \ (DxN) & U \ (DxK) & Z \ (KxN)
    \end{pmatrix}
    \]
  - Each document in a collection as a comb. of a small no of “topics”
  - Each gene-expression sample as a comb. of a small no of “genetic pathways”
  - The “eigenfaces”, “topics”, “genetic pathways”, etc. are the “basis vectors”, which can be learned from data using PCA/SVD or other similar methods
PCA: Example

Original Collection of Images

K=49 Eigenvectors ("eigenfaces") learned by PCA on this data

Each image's reconstructed version
PCA: Example

16 × 16 pixel images of handwritten 3s (as vectors in $\mathbb{R}^{256}$)

Mean $\mu$ and eigenvectors $v_1, v_2, v_3, v_4$

<table>
<thead>
<tr>
<th>Mean</th>
<th>$\lambda_1 = 3.4 \cdot 10^5$</th>
<th>$\lambda_2 = 2.8 \cdot 10^5$</th>
<th>$\lambda_3 = 2.4 \cdot 10^5$</th>
<th>$\lambda_4 = 1.6 \cdot 10^5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

Reconstructions:

$x$ $k = 1$ $k = 10$ $k = 50$ $k = 200$

Each input image now represented by just $k$ numbers (combination weights of each of the $k$ eigenvectors)
PCA: Limitations and Extensions

- A linear projection method

- Won't work well if data can't be approximated by a linear subspace
- But PCA can be kernelized easily (Kernel PCA)
- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)
- PCA relies on eigendecomposition of an $D \times D$ covariance matrix
- Can be slow if done naively. Takes $O(D^3)$ time
- Many faster methods exist (e.g., Power Method)
PCA: Limitations and Extensions

- A linear projection method
  - Won’t work well if data can’t be approximated by a linear subspace

- But PCA can be kernelized easily (Kernel PCA)
- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)

- PCA relies on eigendecomposition of an $D \times D$ covariance matrix
- Can be slow if done naively. Takes $O(D^3)$ time
- Many faster methods exist (e.g., Power Method)
PCA: Limitations and Extensions

- A linear projection method
  - Won’t work well if data can’t be approximated by a linear subspace
  - But PCA can be kernelized easily (Kernel PCA)
PCA: Limitations and Extensions

- A linear projection method
  - Won’t work well if data can’t be approximated by a linear subspace
  - But PCA can be kernelized easily (Kernel PCA)

- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)
PCA: Limitations and Extensions

- A linear projection method
  - Won’t work well if data can’t be approximated by a linear subspace
  - But PCA can be kernelized easily (Kernel PCA)
- Variance based projection directions can sometimes be suboptimal (e.g., if we want to preserve class separation, e.g., when doing classification)

- PCA relies on eigendecomposition of an $D \times D$ covariance matrix
  - Can be slow if done naïvely. Takes $O(D^3)$ time
  - Many faster methods exist (e.g., Power Method)