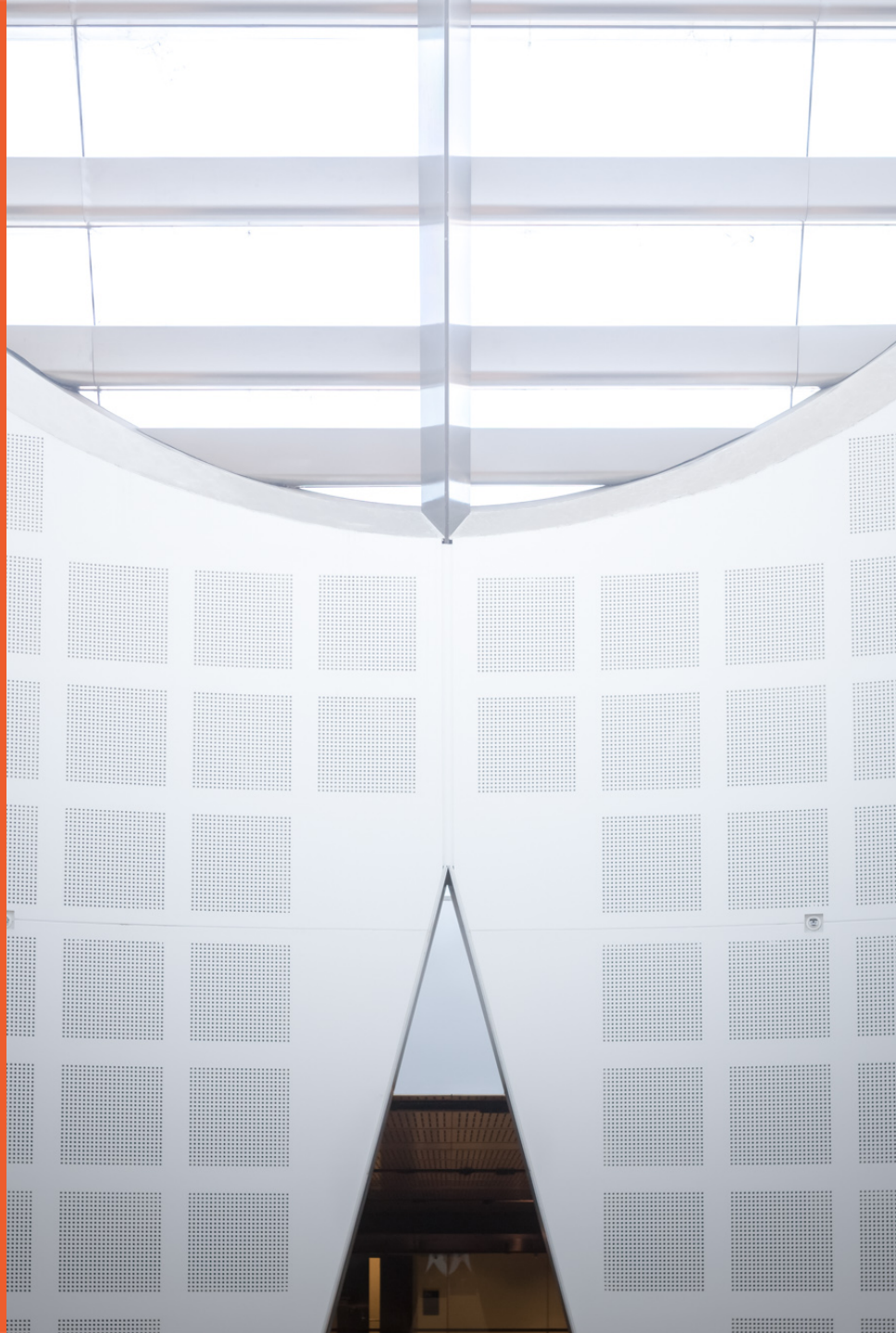
Unsupervised Feature Learning

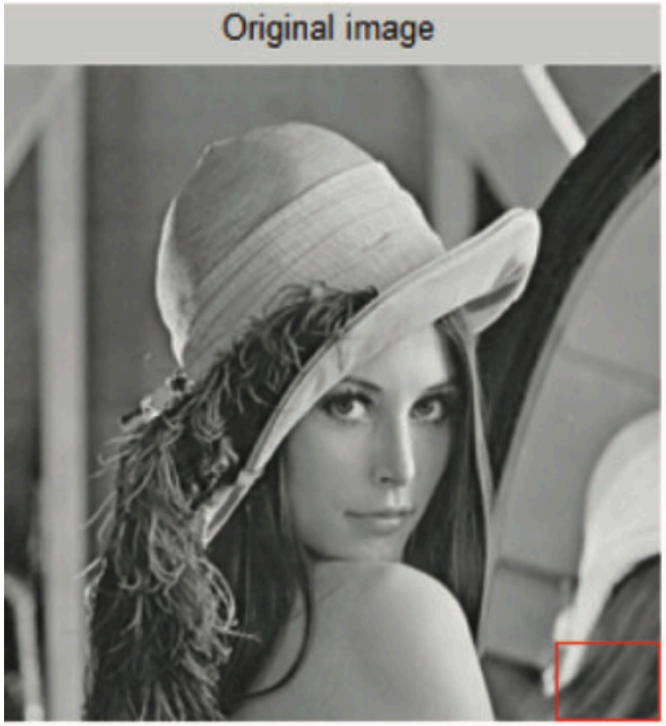
Dr Chang Xu

School of Computer Science
UBTECH Sydney AI Centre

THE UNIVERSITY OF SYDNEY



Data Representation

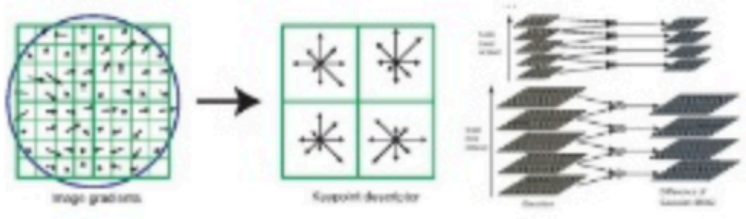


(a)

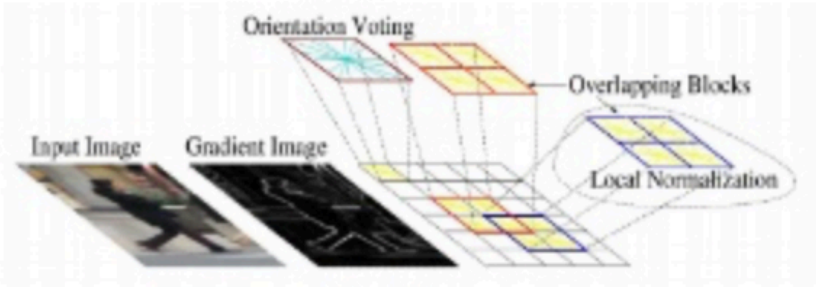
178	186	186	191	204	214	219	216	209	202	176	140	118	111	110	111	105	100	93	92	96	103	99	86	70	78	95	106	106	102	102	94	88	92	93	94	96	96	98	93
148	161	164	185	206	215	219	214	208	197	169	134	116	111	109	103	100	93	90	96	102	97	91	80	76	86	101	111	107	100	98	90	88	91	95	98	94	98	96	83
131	137	153	186	208	218	220	215	207	191	159	128	114	112	107	99	97	90	91	98	99	92	83	74	81	94	105	107	103	96	92	88	88	90	95	96	94	100	90	74
131	139	155	193	211	222	219	213	204	181	148	121	113	111	104	100	93	89	95	101	95	82	79	79	89	98	105	103	100	97	91	88	91	93	98	93	94	93	82	65
135	140	163	200	214	221	218	211	198	169	137	115	111	104	94	87	85	91	100	99	86	73	74	86	100	105	102	97	98	95	86	91	96	103	99	93	93	84	69	58
135	148	180	206	215	221	216	206	190	158	128	113	103	93	81	78	84	95	100	92	76	70	77	89	106	103	97	96	97	94	90	92	103	105	96	92	87	78	64	57
133	155	193	211	218	220	214	200	174	144	121	104	92	85	76	78	88	98	96	84	68	76	90	101	107	95	93	95	95	93	93	95	104	104	96	93	82	69	56	56
143	160	197	215	221	218	208	188	155	130	107	91	81	74	73	82	94	94	88	76	72	86	101	104	102	94	93	95	92	91	96	102	104	98	97	90	76	66	52	53
152	176	204	217	220	214	201	173	142	114	90	77	64	61	73	83	89	81	79	76	82	97	107	102	89	91	93	96	90	90	96	104	103	95	95	88	74	60	55	54
158	185	207	216	217	210	193	163	127	95	79	65	56	64	80	84	82	74	79	88	96	107	109	98	91	89	89	89	93	92	98	103	102	98	93	81	62	58	55	53
163	192	209	215	214	205	183	147	105	78	68	57	63	72	81	83	77	73	82	95	101	108	106	96	90	88	85	86	91	95	101	104	100	95	86	72	58	57	51	50
169	200	212	216	212	198	168	119	83	63	58	61	70	79	83	80	69	71	86	99	109	105	95	89	84	84	85	83	90	99	107	108	101	93	82	66	63	56	52	56
182	204	212	213	205	182	136	78	62	56	57	65	73	78	77	72	72	83	96	108	111	98	86	81	82	83	83	83	90	104	112	111	98	90	80	73	70	61	55	59
189	203	207	203	186	143	88	56	54	55	63	69	78	70	67	69	80	96	106	110	100	87	82	79	78	79	83	90	97	111	116	105	98	89	80	80	71	60	58	64
193	199	195	181	145	92	58	52	51	54	66	72	74	67	70	79	88	102	107	101	92	77	74	78	78	78	84	96	104	113	113	102	94	84	83	82	66	58	62	67
188	178	167	138	99	69	53	51	56	61	71	71	64	66	78	91	101	105	103	94	87	78	76	77	79	81	91	102	111	115	105	95	94	87	79	72	61	60	71	70
174	149	124	94	70	60	52	56	65	73	73	68	64	75	84	97	108	107	100	89	77	75	74	77	79	88	102	110	116	108	101	92	94	86	77	66	62	68	68	59
143	107	80	68	64	56	58	66	72	76	69	67	76	89	98	107	106	100	95	80	70	69	72	80	85	94	105	114	114	105	102	95	92	81	71	59	65	72	72	60
101	79	78	72	66	63	72	75	78	74	71	75	88	103	111	112	104	92	83	77	67	64	66	79	92	97	110	113	110	107	101	97	87	70	60	65	70	69	69	59
80	80	83	77	72	76	77	78	76	72	74	86	98	108	115	104	93	82	75	77	67	63	70	86	102	107	116	115	113	105	102	94	78	65	62	69	69	66	60	54
80	76	77	80	82	85	81	79	72	72	82	95	103	107	107	93	83	72	68	67	60	65	80	99	108	112	117	118	114	103	101	87	73	66	64	60	62	62	56	48
88	88	87	95	91	87	84	75	72	77	88	100	107	105	94	86	76	73	69	64	61	73	92	106	110	112	119	117	111	106	97	83	74	66	61	60	62	58	53	54
93	100	100	101	93	88	82	78	82	84	95	105	101	95	84	80	79	78	71	68	65	83	100	112	114	120	122	111	107	102	89	74	69	62	58	60	59	54	55	53
101	109	109	104	96	89	80	80	85	92	99	99	84	81	74	70	75	74	73	72	76	90	105	116	120	122	117	106	105	94	80	68	59	58	59	56	51	51	49	48
111	116	113	104	95	88	83	89	94	99	99	91	82	71	67	70	71	72	75	75	87	104	116	119	129	120	113	104	94	87	74	64	58	57	58	57	53	54	52	58
119	118	113	106	97	92	91	97	99	97	98	86	80	72	71	75	74	74	76	86	102	112	121	125	121	113	109	99	89	78	66	60	54	58	60	55	51	55	60	79
127	120	111	102	93	93	97	103	97	89	87	84	85	83	78	80	75	77	84	98	113	123	126	126	120	108	104	91	83	69	57	51	49	53	56	63	56	67	77	88
128	119	109	98	93	92	95	96	86	84	78	81	87	85	83	78	76	85	97	112	122	129	128	125	115	106	93	82	68	58	55	47	51	54	55	61	68	83	93	92
124	116	105	94	94	90	86	83	77	77	81	90	90	88	85	82	80	93	103	117	127	130	128	125	114	95	80	70	56	52	48	49	50	51	58	68	80	93	101	99
122	113	96	89	90	83	75	74	74	75	88	92	92	91	86	82	86	95	106	123	126	129	127	121	108	89	68	59	58	51	42	45	49	56	68	81	91	99	103	107

(b)

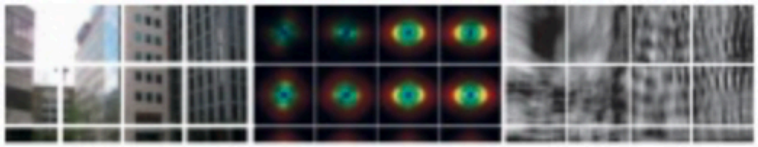
Hand-crafted features



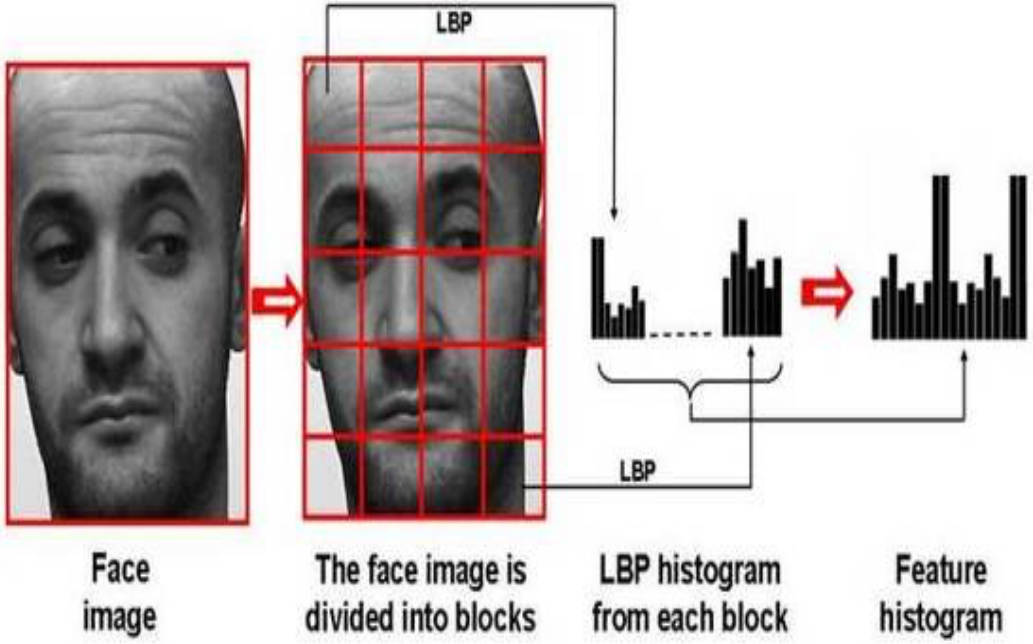
SIFT



HoG



GIST



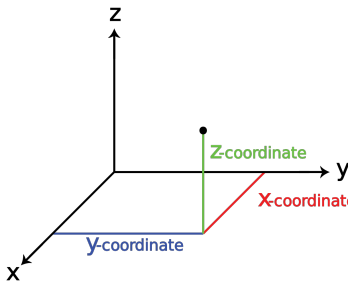
Outline

1. Background
2. Principal Component Analysis (PCA)
3. Autoencoder (AE)
4. Sparse Coding (SC)

Unsupervised feature learning

For raw data $\boldsymbol{x} \in \boldsymbol{R}^d$, we try to find an approximation in a space:

$$\hat{\boldsymbol{x}} = \sum_{i=1}^{k} \alpha_i \phi_i$$



where ϕ_i, $1 \leq i \leq k$, are basis vectors of the space; and α_i, $1 \leq i \leq k$, are features or representations in the space.

Over-complete representation: if $k > d$, i.e., The dimension of features (the number of basis vectors) is larger than that of raw data.
Under-complete representation: if $k < d$, i.e., the dimension of features is smaller than that of raw data.

Constraints such as sparsity, low-rankness, orthogonality, linear independence on the basis or features can lead to different models.

Principal Component Analysis (PCA)

Principal Component Analysis (PCA)

- A technique widely used for
  --Dimension reduction
  --Data compression
  --Feature extraction
  --Data visualization

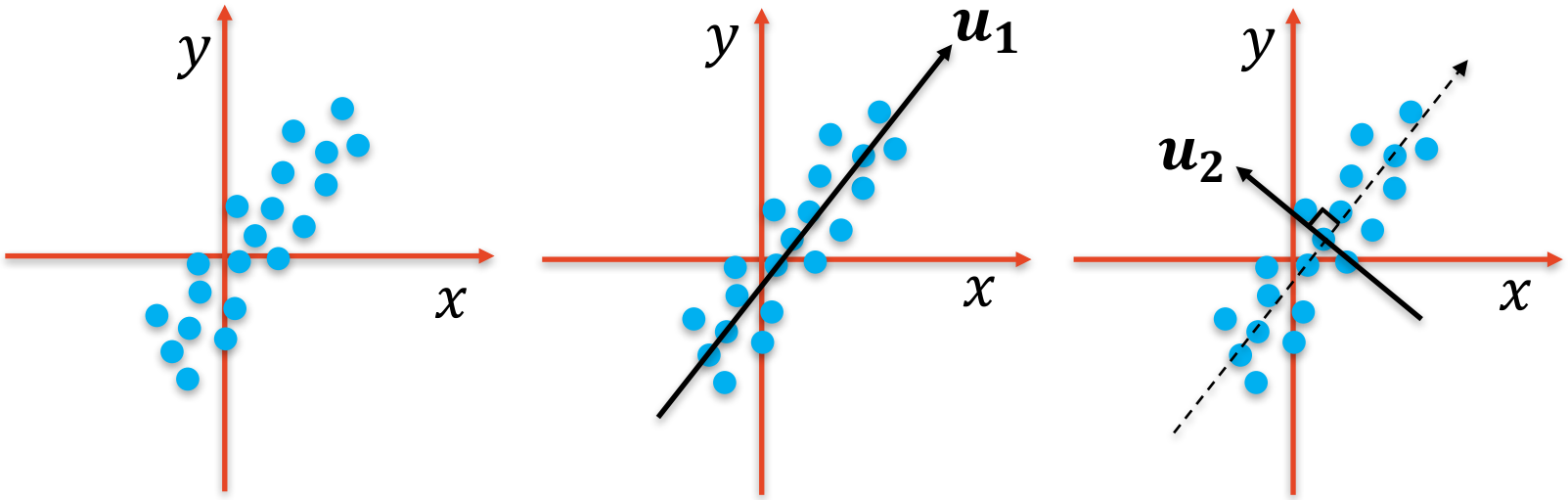
- Two equivalent definitions of PCA:

[Maximum Variance Formulation] Project the data onto a lower dimensional space such that the variance of the projected data is maximized.

[Minimum Error Formulation] Project the data onto a lower dimensional space such that the mean squared distance between data points and their projections (average projection cost) is minimized.

PCA Cont'd

Maximum Variance Formulation



data

The first principal component

The second principal component

PCA Cont'd

Let $\boldsymbol{X} = \{\boldsymbol{x}_i \in \boldsymbol{R}^d\}_{1 \le i \le n}$ be a set of observations.

PCA: project $\boldsymbol{X}$ onto a k dimensional subspace ($k < d$) such that the variance of the projected data is maximized.

We first let $k = 1$, then generalize it to the case $k > 1$.

Let $\boldsymbol{u}_1$ be the basis of the 1 dimensional subspace, and $\boldsymbol{u}_1^\top \boldsymbol{u}_1 = 1$.

$$\widehat{Var} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{u}_1^\top \boldsymbol{x}_i - \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{u}_1^\top \boldsymbol{x}_i)^2$$

PCA Cont'd

Let $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$ and $\boldsymbol{S} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top$.

$$\widehat{Var} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{u}_1^\top(\boldsymbol{x}_i - \bar{\boldsymbol{x}}))^2 = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{u}_1^\top(\boldsymbol{x}_i - \bar{\boldsymbol{x}})(\boldsymbol{x}_i - \bar{\boldsymbol{x}})^\top\boldsymbol{u}_1 = \boldsymbol{u}_1^\top\boldsymbol{S}\boldsymbol{u}_1$$

where $\boldsymbol{S}$ is the scatter matrix.

Then, the problem becomes

$$\max_{\boldsymbol{u}_1} \boldsymbol{u}_1^\top\boldsymbol{S}\boldsymbol{u}_1 \quad s.t. \quad \boldsymbol{u}_1^\top\boldsymbol{u}_1 = 1.$$

PCA Cont'd

$$\min_{\boldsymbol{u}_1} -\boldsymbol{u}_1^\top \boldsymbol{S}\boldsymbol{u}_1 \quad s.t. \quad \boldsymbol{u}_1^\top \boldsymbol{u}_1 = 1.$$

Lagrangian function:

$$L = -\boldsymbol{u}_1^\top \boldsymbol{S}\boldsymbol{u}_1 + \lambda_1(\boldsymbol{u}_1^\top \boldsymbol{u}_1 - 1)$$

where λ_1 is the Lagrangian multiplier.

According to KKT conditions,

$$\frac{\partial L}{\partial \boldsymbol{u}_1} = -\boldsymbol{S}\boldsymbol{u}_1 + \lambda_1 \boldsymbol{u}_1 = 0$$

$$\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$$

PCA Cont'd

$$\boldsymbol{S}\boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$$

Thus, $\boldsymbol{u}_1$ is a eigenvector of $\boldsymbol{S}$ and λ_1 is its eigenvalue. Note that

$$-\boldsymbol{u}_1^\top \boldsymbol{S}\boldsymbol{u}_1 = -\lambda_1 \boldsymbol{u}_1^\top \boldsymbol{u}_1 = -\lambda_1$$

λ_1 is the largest eigenvalue of $\boldsymbol{S}$.

$$\min_{\boldsymbol{u}_1} -\boldsymbol{u}_1^\top \boldsymbol{S}\boldsymbol{u}_1 \quad s.t. \quad \boldsymbol{u}_1^\top \boldsymbol{u}_1 = 1.$$

PCA Cont'd

$$\boldsymbol{S}\boldsymbol{u} = \lambda \boldsymbol{u}$$

For $k < d$ dimensions:
-- $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$ are the eigenvectors corresponding to the largest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_k$ of the scatter matrix $\boldsymbol{S}$.

Then, the i-th principal component of $\boldsymbol{X}$ is $\boldsymbol{u}_i^\top \boldsymbol{X}$ and $var\left(\boldsymbol{u}_i^\top \boldsymbol{X}\right) = \lambda_i$.

PCA Cont'd

Proof for $k = 2$ (the proof for $k \geq 3$ is similar).

$$\min_{\boldsymbol{u}_2} -\boldsymbol{u}_2^\top \boldsymbol{S}\boldsymbol{u}_2\,, \qquad s.t.\ \ \boldsymbol{u}_2^\top \boldsymbol{u}_2 = 1, \qquad \boldsymbol{u}_1^\top \boldsymbol{u}_2 = 0$$

Lagrangian function:

$$L = -\boldsymbol{u}_2^\top \boldsymbol{S}\boldsymbol{u}_2 + \lambda_2(\boldsymbol{u}_2^\top \boldsymbol{u}_2 - 1) + \theta \boldsymbol{u}_1^\top \boldsymbol{u}_2$$

According to KKT conditions,

$$\frac{\partial L}{\partial \boldsymbol{u}_2} = -\boldsymbol{S}\boldsymbol{u}_2 + \lambda_2 \boldsymbol{u}_2 + \theta \boldsymbol{u}_1 = 0$$

PCA Cont'd

$$\frac{\partial L}{\partial \boldsymbol{u}_2} = -\boldsymbol{S}\boldsymbol{u}_2 + \lambda_2 \boldsymbol{u}_2 + \theta \boldsymbol{u}_1 = 0$$

$$\Downarrow \boldsymbol{u}_1^\top \times$$

$$-\boldsymbol{u}_1^\top \boldsymbol{S}\boldsymbol{u}_2 + \lambda_2 \boldsymbol{u}_1^\top \boldsymbol{u}_2 + \theta \boldsymbol{u}_1^\top \boldsymbol{u}_1 = 0$$

$$\Downarrow \boldsymbol{u}_1^\top \boldsymbol{S}\boldsymbol{u}_2 = \lambda_1 \boldsymbol{u}_1^\top \boldsymbol{u}_2 = 0$$

$$\theta = 0$$

$$\Downarrow$$

$$-\boldsymbol{S}\boldsymbol{u}_2 + \lambda_2 \boldsymbol{u}_2 = 0$$

$$\Downarrow$$

λ_2 can not be the largest eigenvalue of $\boldsymbol{S}$. Then it is the second largest. Proof completed.

PCA Cont'd

Algorithm

- Step 1: Substract the mean data $\overline{\boldsymbol{x}}$ from original data, i.e., $\boldsymbol{z} = \boldsymbol{x} - \overline{\boldsymbol{x}}$;
- Step 2: Compute the scatter matrix $\boldsymbol{S} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{z}_i \boldsymbol{z}_i^\top$;
- Step 3: Compute the eigenvectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$ and eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_k$ by using Singular Value Decomposition (SVD) of $\boldsymbol{S}$. Then $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k]$ is the projection matrix.
- Step 4: $\boldsymbol{y}_i = \boldsymbol{U}^\top \boldsymbol{x}_i$

PCA Cont'd

Minimum Error Formulation

$\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_k]$ is the projection matrix, where $\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$.
Assume $\overline{\boldsymbol{x}} = 0$, then for any data $\boldsymbol{x}_i$, the projected data is $\boldsymbol{y}_i = \boldsymbol{U}^\top \boldsymbol{x}_i$.

The reconstructed data $\widehat{\boldsymbol{x}_i} = \boldsymbol{U}\boldsymbol{y}_i = \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}_i$.

$$\min_{\boldsymbol{U}} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}_i\|^2 \qquad s.t. \quad \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$$

$$\frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}_i\|^2 = \frac{1}{n} \sum_{i=1}^{n} (-\boldsymbol{x}_i^\top \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}_i + \boldsymbol{x}_i^\top \boldsymbol{x}_i) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{tr}(-\boldsymbol{U}^\top \boldsymbol{x}_i \boldsymbol{x}_i^\top \boldsymbol{U} + \boldsymbol{x}_i^\top \boldsymbol{x}_i)$$

$$\min_{\boldsymbol{U}} -\mathrm{tr}(\boldsymbol{U}^\top \boldsymbol{S}\boldsymbol{U}) \qquad s.t. \quad \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$$

PCA Q&A

Q: How to determine k?

A: Percentage of variance retained: $P(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \geq t$ (say, t=0.95).

Q: How about $d \gg n$?

A: Assume that data has zero mean and $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n]^\top \in \boldsymbol{R}^{n \times d}$, then $\boldsymbol{S} = \frac{1}{n}\boldsymbol{X}^\top \boldsymbol{X}$. If λ_k and $\boldsymbol{u}_k$ are the k-th eigenvalue and eigenvector of $\boldsymbol{S}$ ($k < n$), then λ_k and $\lambda_k^{-1/2}\boldsymbol{X}\boldsymbol{u}_k$ are the k-th eigenvalue and eigenvector of $\frac{1}{n}\boldsymbol{X}\boldsymbol{X}^\top$. Thus, we can decompose $\frac{1}{n}\boldsymbol{X}\boldsymbol{X}^\top$ to save computations.

PCA Cont'd

$$\hat{\boldsymbol{x}} = \sum_{i=1}^{k} \alpha_i \phi_i$$

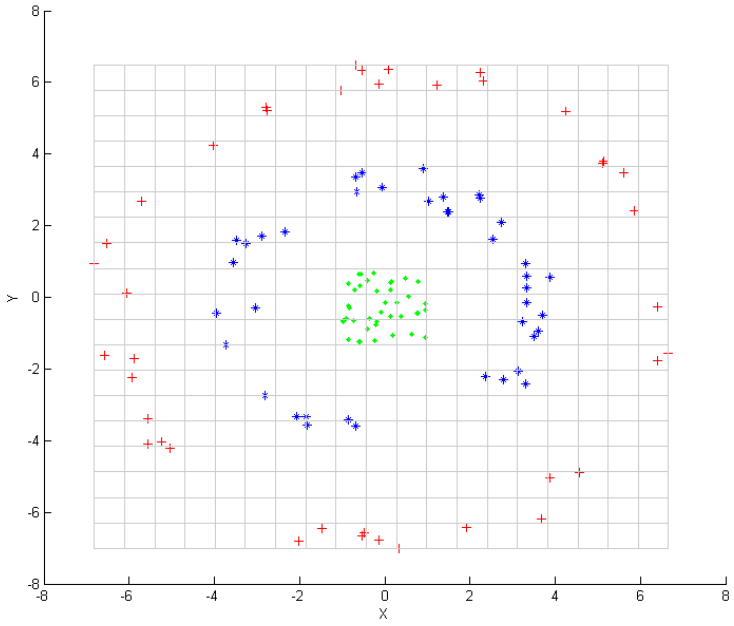
where ϕ_i, $1 \leq i \leq k$, are basis vectors of the space; and α_i, $1 \leq i \leq k$, are features or representations in the space.

For PCA:

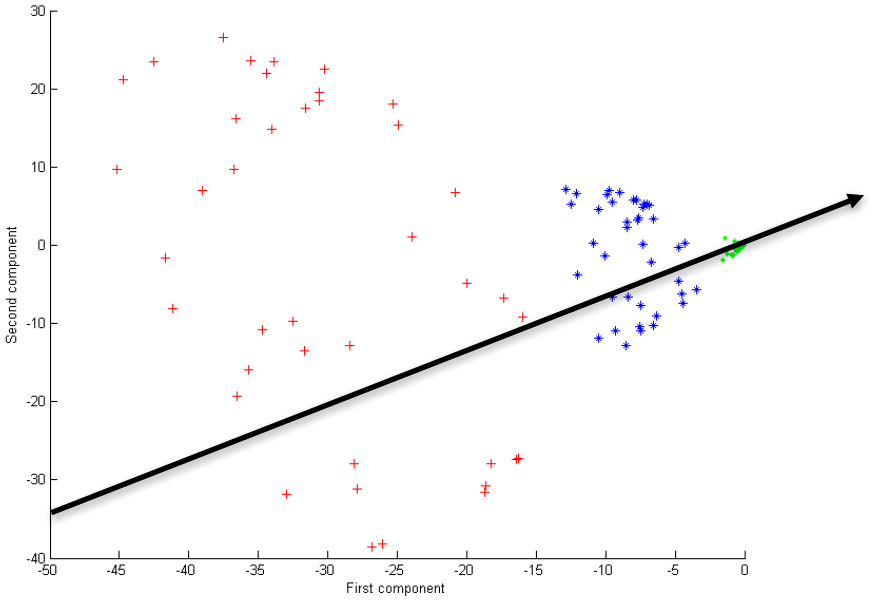
$$\hat{\boldsymbol{x}}_i = \boldsymbol{U}\boldsymbol{y}_i = \sum_{j=1}^{k} y_{ij}\boldsymbol{u}_j$$

- Under-complete representation, $k < d$
- Orthogonality constraints

Kernel PCA



Input points before kernel PCA



Output points after kernel PCA with kernel $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\boldsymbol{x}_1^\top \boldsymbol{x}_2 + 1)^2$.

Images credit: https://en.wikipedia.org/wiki/Kernel_principal_component_analysis

Kernel PCA Cont'd

We assume the data $\boldsymbol{x}_i$ is first mapped to a feature space $\psi(\boldsymbol{x}_i)$, and the mean feature has been subtracted. $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \psi(\boldsymbol{x}_1)^\top \psi(\boldsymbol{x}_2)$ is the kernel function defined in the feature space.

$$\boldsymbol{S} = \frac{1}{n}\sum_{i=1}^{n} \psi(\boldsymbol{x}_i)\psi(\boldsymbol{x}_i)^\top$$

Follow the standard PCA, the projection vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m$ statisfy:

$$\boldsymbol{S}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$$

The problem is: ψ is unknown to us!

Kernel PCA Cont'd

$$\boldsymbol{S} = \frac{1}{n}\sum_{j=1}^{n} \psi(\boldsymbol{x}_j)\psi(\boldsymbol{x}_j)^\top \qquad \boldsymbol{S}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i$$

$$\frac{1}{n}\sum_{j=1}^{n} \psi(\boldsymbol{x}_j)\{\psi(\boldsymbol{x}_j)^\top \boldsymbol{v}_i\} = \lambda_i \boldsymbol{v}_i$$

Assume that $\boldsymbol{v}_i = \sum_{j=1}^{n} \alpha_{ij}\psi(\boldsymbol{x}_j)$

$$\frac{1}{n}\sum_{j=1}^{n} \psi(\boldsymbol{x}_j)\left\{\psi(\boldsymbol{x}_j)^\top \sum_{m=1}^{n} \alpha_{im}\psi(\boldsymbol{x}_m)\right\} = \lambda_i \sum_{j=1}^{n} \alpha_{ij}\psi(\boldsymbol{x}_j)$$

Kernel PCA Cont'd

$$\frac{1}{n}\sum_{j=1}^{n}\psi(\boldsymbol{x}_j)\left\{\psi(\boldsymbol{x}_j)^\top\sum_{m=1}^{n}\alpha_{im}\psi(\boldsymbol{x}_m)\right\}=\lambda_i\sum_{j=1}^{n}\alpha_{ij}\psi(\boldsymbol{x}_j)$$

Multiplying both sides by $\psi(\boldsymbol{x}_l)^\top$,

$$\frac{1}{n}\sum_{j=1}^{n}\psi(\boldsymbol{x}_l)^\top\psi(\boldsymbol{x}_j)\left\{\sum_{m=1}^{n}\alpha_{im}\psi(\boldsymbol{x}_j)^\top\psi(\boldsymbol{x}_m)\right\}=\lambda_i\sum_{j=1}^{n}\alpha_{ij}\psi(\boldsymbol{x}_l)^\top\psi(\boldsymbol{x}_j)$$

$$\frac{1}{n}\sum_{j=1}^{n}k(\boldsymbol{x}_l,\boldsymbol{x}_j)\left\{\sum_{m=1}^{n}\alpha_{im}k(\boldsymbol{x}_j,\boldsymbol{x}_m)\right\}=\lambda_i\sum_{j=1}^{n}\alpha_{ij}k(\boldsymbol{x}_l,\boldsymbol{x}_j)$$

Kernel PCA Cont'd

$$\frac{1}{n}\sum_{j=1}^{n} k(\boldsymbol{x}_l, \boldsymbol{x}_j)\left\{\sum_{m=1}^{n} \alpha_{im} k(\boldsymbol{x}_j, \boldsymbol{x}_m)\right\} = \lambda_i \sum_{j=1}^{n} \alpha_{ij} k(\boldsymbol{x}_l, \boldsymbol{x}_j)$$

Let $\boldsymbol{K}$ be the kernel matrix, where $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{K}_{l.}$ is the l-th row of $\boldsymbol{K}$.

$$\frac{1}{n}\boldsymbol{K}_{l.}\boldsymbol{K}\boldsymbol{\alpha}_i = \lambda_i \boldsymbol{K}_{l.}\boldsymbol{\alpha}_i$$

For $l = 1, \cdots, n$, we have

$$\frac{1}{n}\boldsymbol{K}^2\boldsymbol{\alpha}_i = \lambda_i \boldsymbol{K}\boldsymbol{\alpha}_i$$

Kernel PCA Cont'd

$$\frac{1}{n}\boldsymbol{K}^2\boldsymbol{\alpha}_i = \lambda_i \boldsymbol{K}\boldsymbol{\alpha}_i$$

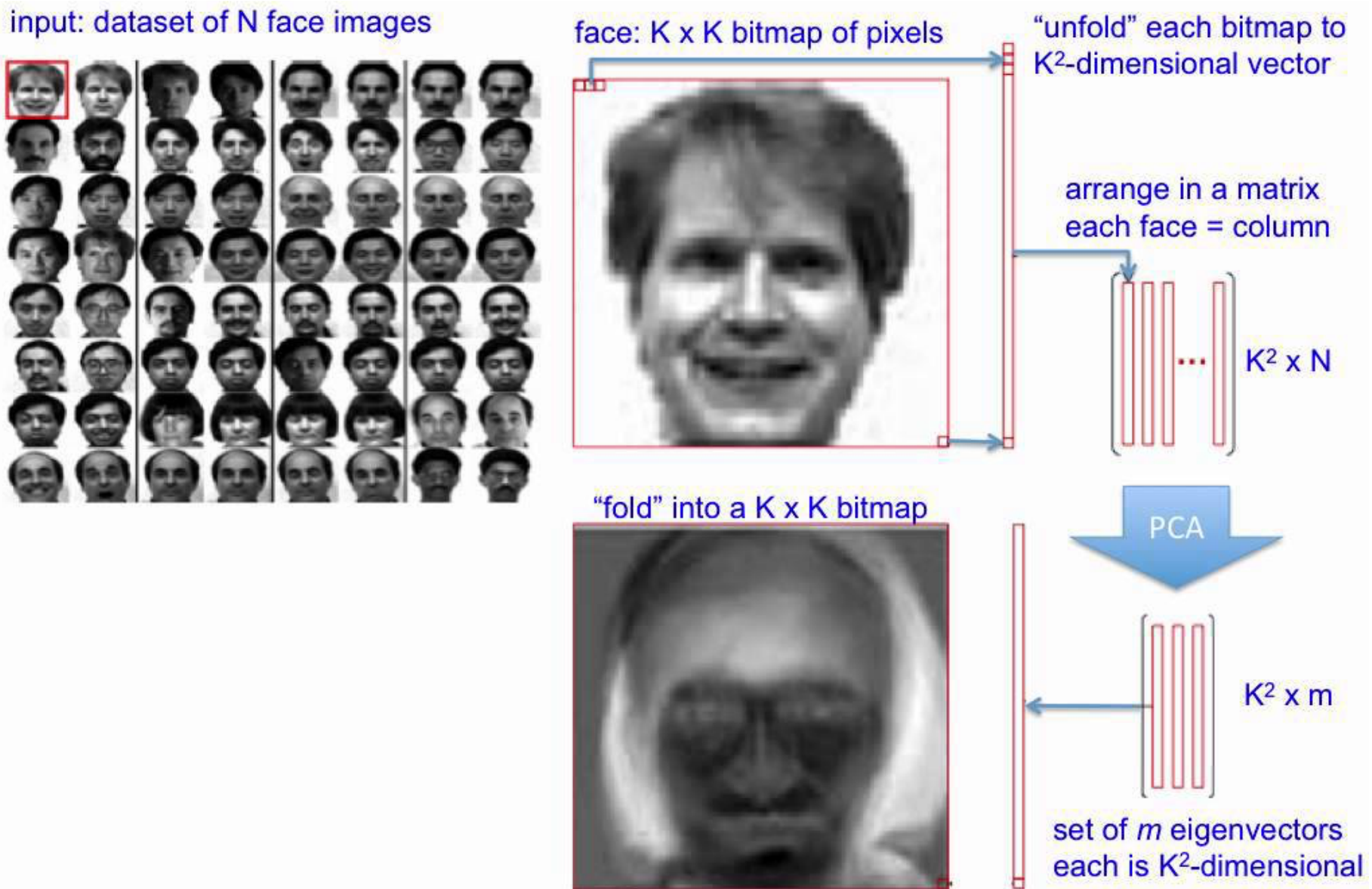
$$\boldsymbol{K}\boldsymbol{\alpha}_i = n\lambda_i \boldsymbol{\alpha}_i$$

$\boldsymbol{\alpha}_i$ is the eigenvector of $\boldsymbol{K}$.

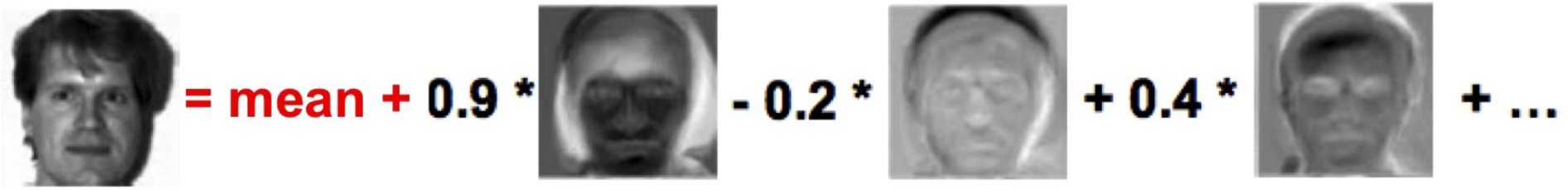
Recall that $\boldsymbol{v}_i = \sum_{j=1}^{n} \alpha_{ij}\psi(\boldsymbol{x}_j)$

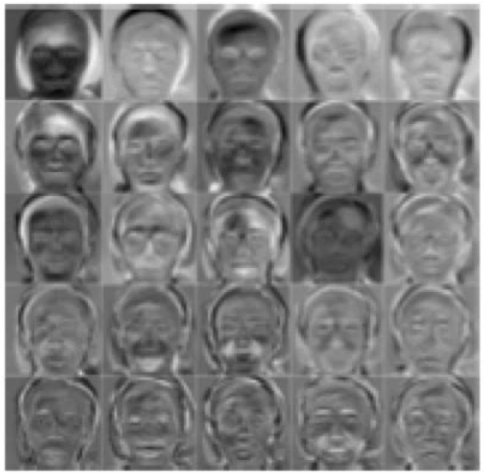
$$y = \psi(\boldsymbol{x})^\top \boldsymbol{v}_i = \sum_{j=1}^{n} \alpha_{ij}\psi(\boldsymbol{x})^\top \psi(\boldsymbol{x}_j) = \sum_{j=1}^{n} \alpha_{ij} k(\boldsymbol{x},\boldsymbol{x}_j)$$

Face Recognition

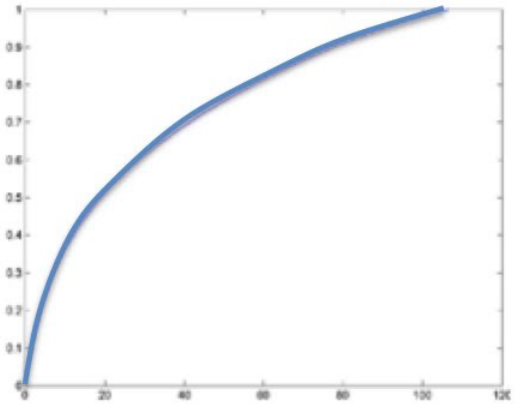


Face Recognition





- Project new face to space of eigen-faces
- Represent vector as a linear combination of principal components
- How many do we need?



Autoencoder (AE)

Autoencoder (AE)

Autoencoder aims to learn discriminative representations $\mathbf{y}$ which capture the main factors of variation in input data.

Encoder: mapping the input data $\boldsymbol{x}$ to a hidden representation $\boldsymbol{y}$ via:

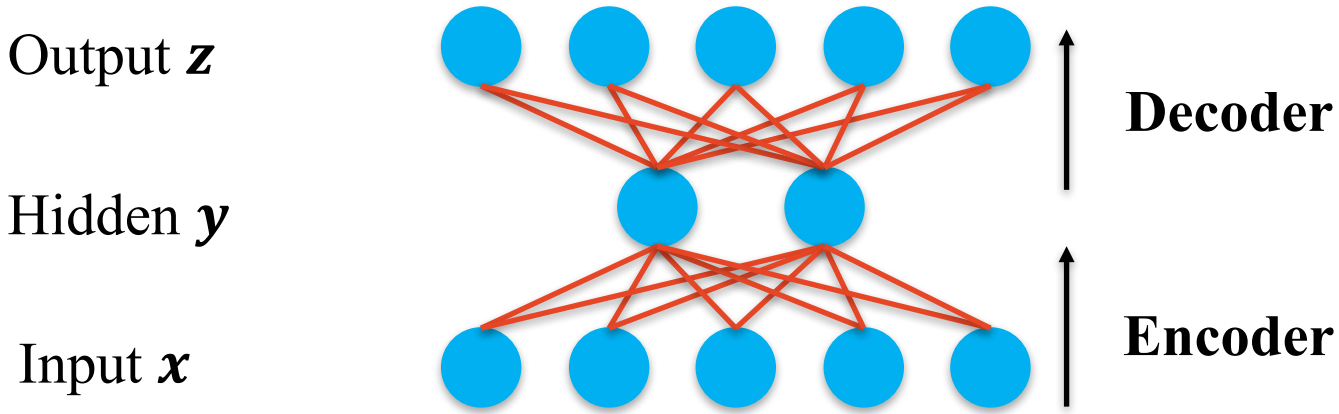
$$\boldsymbol{y} = f\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right),$$

Decoder: mapping the hidden representation back into a reconstruction $\boldsymbol{z}$ of the same shape as $\boldsymbol{x}$,

$$\boldsymbol{z} = f\left(\boldsymbol{W}^{(2)}\boldsymbol{y} + \boldsymbol{b}^{(2)}\right).$$

where f denotes the activation function, such as sigmoid.

Target: $\boldsymbol{z}$ reconstruct $\boldsymbol{x}$.



AE Cont'd

The objective function measures the reconstruction error:

- Square loss: $J(\theta) = \|\boldsymbol{x} - \boldsymbol{z}\|^2$, $\theta = \{\boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)}, \boldsymbol{W}^{(2)}, \boldsymbol{b}^{(2)}\}$
- Cross-entropy: $J(\theta) = -\sum_{i=1}^{d} x_i \log(z_i) + (1 - x_i)\log(1 - z_i)$

We can constrain the weight matrix $\boldsymbol{W}^{(2)}$ of the reverse mapping to be the transpose of the forward mapping: $\boldsymbol{W}^{(1)} = \boldsymbol{W}$, and $\boldsymbol{W}^{(2)} = \boldsymbol{W}^{\top}$, referred to as **tied weights**.

Regularization term, such as l_1 norm and l_2 norm of $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$, is often added.

AE Cont'd

Backpropagation (BP) algorithm is often applied to optimize AE. Take the objective function $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\|\boldsymbol{x}_i - \boldsymbol{z}_i\|^2 + \lambda\sum_{l=1}^{2}\left\|\boldsymbol{W}^{(l)}\right\|_F^2$ as an example:

One iteration of gradient descent updates the parameters as follows:

$$\boldsymbol{W}^{(l)} = \boldsymbol{W}^{(l)} - \alpha\frac{\partial}{\partial \boldsymbol{W}^{(l)}}J(\theta)$$

$$\boldsymbol{b}^{(l)} = \boldsymbol{b}^{(l)} - \alpha\frac{\partial}{\partial \boldsymbol{b}^{(l)}}J(\theta)$$

Where α is the learning rate. The detail of BP algorithm will be introduced in another class.

AE Cont'd

A special case: We choose f as the identity mapping, $\mathbf{y} \in \boldsymbol{R}^k$ resides in a lower dimensional space ($k < d$), and ignore the bias term. We apply the squared loss and tied weights into AE:

$$J(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{W}^\top \boldsymbol{W}\boldsymbol{x}_i\|^2$$

PCA:

$$\min_{\boldsymbol{U}} \frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x}_i\|^2 \qquad s.t. \quad \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}$$

Differences between AE and PCA are the nonlinearity in AE and the orthogonality constraint in PCA.

AE Cont'd

Without any regularization, AE tends to learn the identity function.

Thus, many regularizations have been added to AE. Beside that the code $\mathbf{y}$ has a lower dimension, there are another two popular regularizations:

- Sparse AE requires the code $\mathbf{y}$ to be sparse.
- Denoising AE: adds noises to input data, uses this corrupted data as input of AE, and then reconstructs the original input $\boldsymbol{x}$.

$$\boldsymbol{y} = f\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right),$$
$$\boldsymbol{z} = f\left(\boldsymbol{W}^{(2)}\boldsymbol{y} + \boldsymbol{b}^{(2)}\right).$$

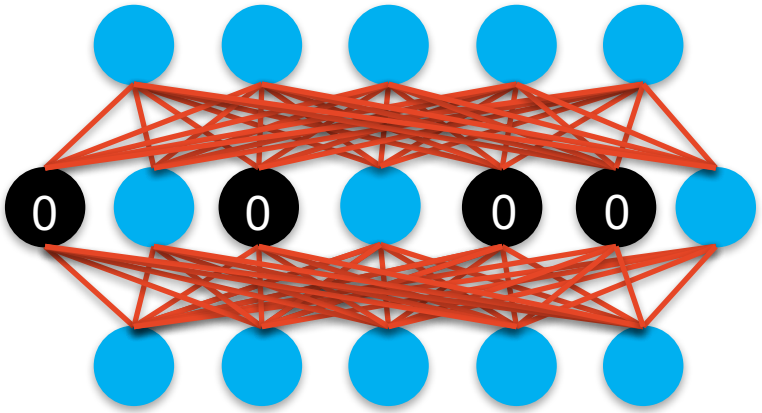
Sparse AE

An illustration of sparse autoencoder ($k > d$):

$\boldsymbol{z} \in \boldsymbol{R}^d$

$\boldsymbol{y} \in \boldsymbol{R}^k$

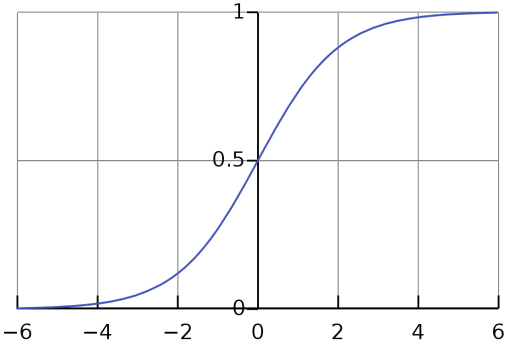
$\boldsymbol{x} \in \boldsymbol{R}^d$



$\boldsymbol{W}^{(2)} \in \boldsymbol{R}^{d \times k}$

$\boldsymbol{W}^{(1)} \in \boldsymbol{R}^{k \times \boldsymbol{d}}$

We choose sigmoid as the activation functions of sparse AE.



Sparse AE Cont'd

Compute the average activation of **each neuron** $j, \forall j \in \{1, \cdots, k\}$, on the hidden layer w.r.t. the training examples $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$:

$$\hat{\rho}_j = \frac{1}{n}\sum_{i=1}^{n} y_j(\boldsymbol{x}_i), y_j = \boldsymbol{y}[j], \boldsymbol{y} \in \boldsymbol{R}^k$$

We would like the average activation of each neuron to be low and close to a sparsity parameter ρ, say, 0.05, such that only a few neurons activate.

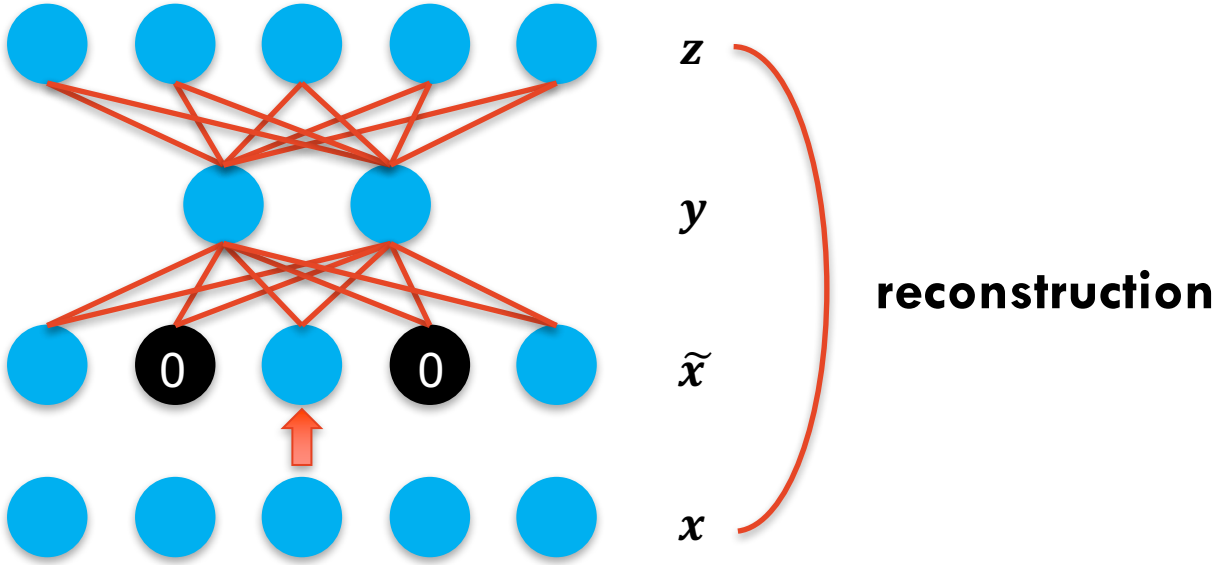
$$J_{sparse}(\theta) = J(\theta) + \beta \sum_{j=1}^{k} KL(\rho || \hat{\rho}_j),$$

where β controls the weight of sparsity penalty term and $\mathrm{KL}(\rho||\hat{\rho}_j) = \rho \log\left(\frac{\rho}{\hat{\rho}_j}\right) + (1-\rho)\log(\frac{1-\rho}{1-\hat{\rho}_j})$.

Denoising AE

To learn representations which are robust to partial destruction of the inputs. We need to add a stochastic corruption step on the input. Here is an example:

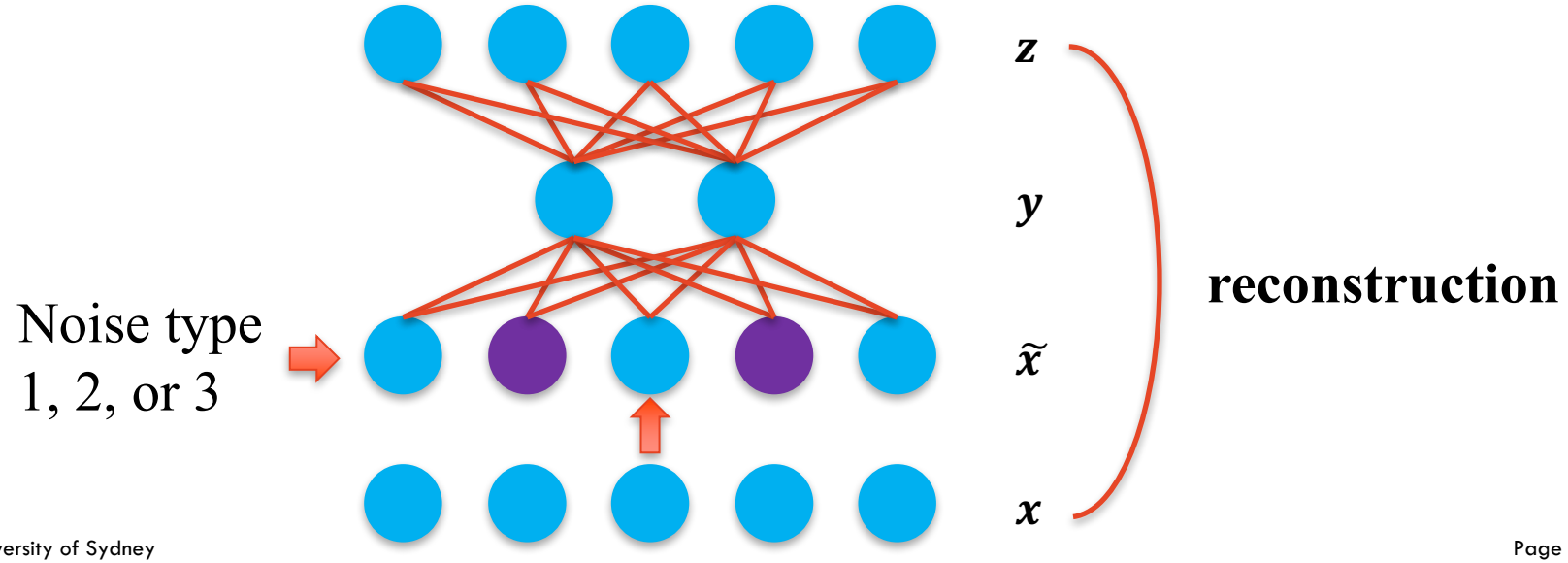
- Step 1: Choose a desired proportion ρ of destruction;
- Step 2: Select $\rho \times d$ components of $\boldsymbol{x}$ randomly and set them to 0;
- Step 3: Use the corrupted $\tilde{\boldsymbol{x}}$ as input, and train the AE.



Denoising AE

Noise types usually considered for corrupting the input data:

1. Additive Gaussian noise;
2. Multiplicative 0-1 noise or masking noise, seen in the previous slide.
3. Salt-and-pepper noise: a fraction ρ of the elements of $\boldsymbol{x}$ (chosen at random for each example) is set to their minimum or maximum possible value (typically 0 or 1) according to a fair coin flip.



AE Cont'd

$$\hat{\boldsymbol{x}} = \sum_{i=1}^{k} \alpha_i \phi_i$$

where ϕ_i, $1 \leq i \leq k$, are basis vectors of the space; and α_i, $1 \leq i \leq k$, are features or representations in the space.

In AE, assume f is identity function, and ignore the bias term, then

$$\hat{\boldsymbol{x}}_i = \boldsymbol{W}^{(2)} \boldsymbol{y}_i = \sum_{j=1}^{k} y_{ij} \boldsymbol{W}_{\cdot j}^{(2)}$$

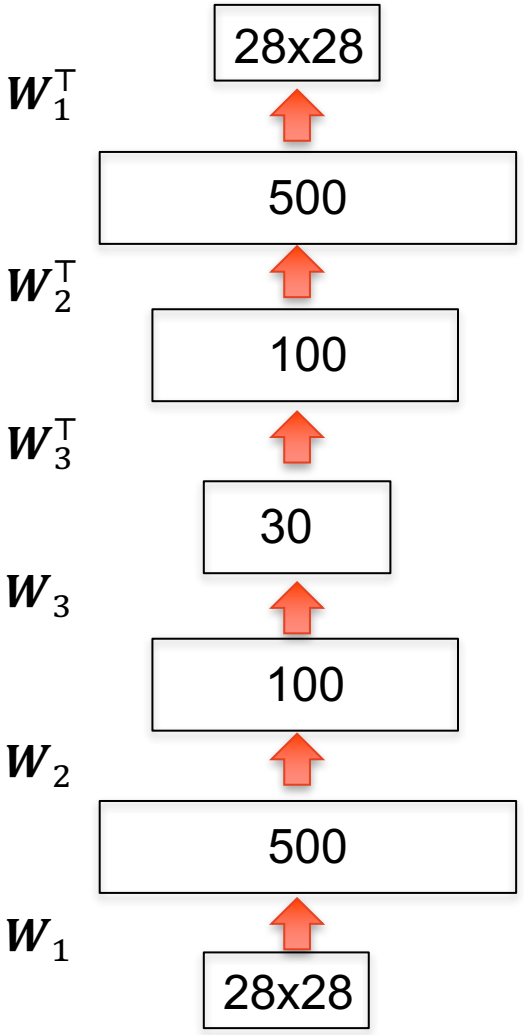
Classical AE: Under-complete representation

Sparse AE: Over-complete representation and sparsity on features

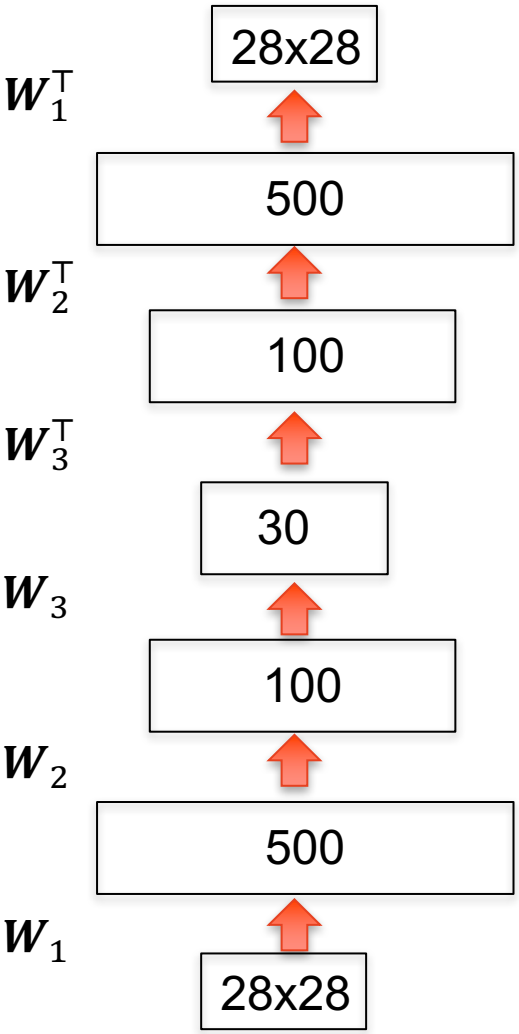
Stacked AE

A stacked autoencoder enjoys all the benefits of any deep network of greater expressive power, and tends to learn higher-order features.

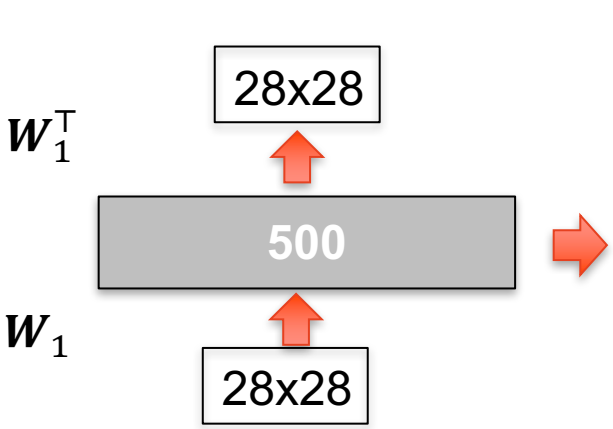
The problem is that it is very difficult to optimize directly using backpropagation (BP).

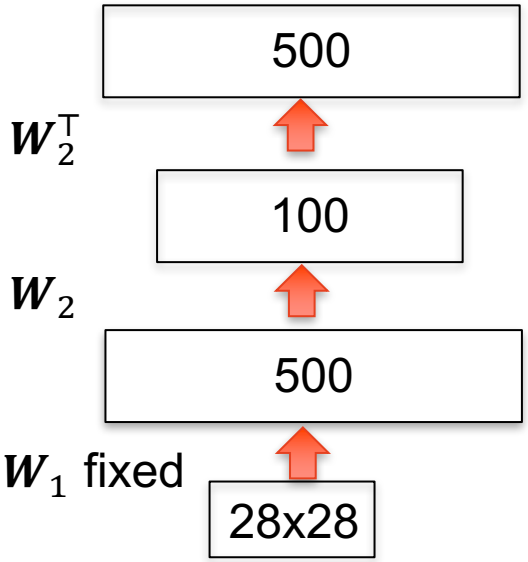


Stacked AE Cont'd



Method: Greedy layer-wise training + fine-tuning using BP.



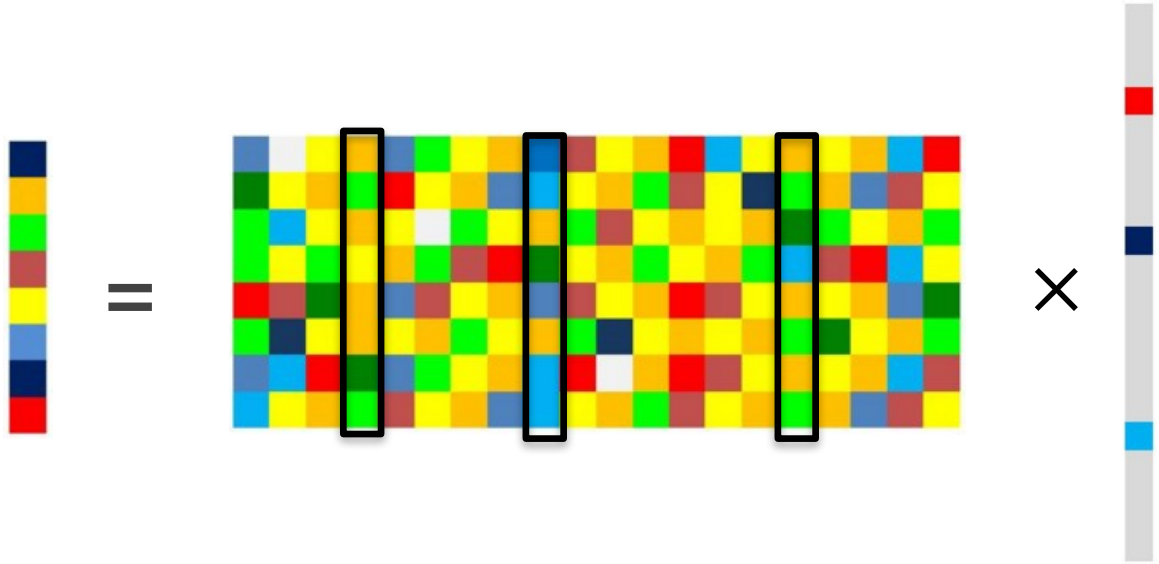


Sparse Coding (SC)

Sparse Coding (SC)

Given a set of input data, sparse coding learns a possibly over-complete set of basis vectors $\{\phi_i\}$ and features $\boldsymbol{\alpha}$.

But not all basis vectors are necessarily needed to approximate the input data $\boldsymbol{x}$. We thus add the sparsity constraint.



Data x

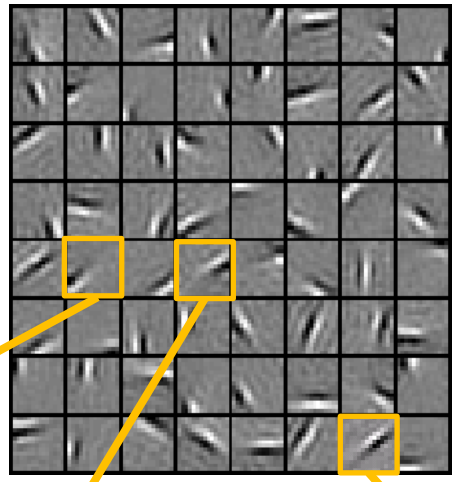
Basis vectors: Over-complete

Features: Sparse

SC Cont'd

Natural Images

Learned bases $(f_1, \ldots, f_{64})$: "Edges"



Test example

$$x = 0.8 * f_{36} + 0.3 * f_{42} + 0.5 * f_{63}$$

Feature: $[0, 0, \ldots, 0, \mathbf{0.8}, 0, \ldots, 0, \mathbf{0.3}, 0, \ldots, 0, \mathbf{0.5}, \ldots] = [\alpha_1, \ldots, \alpha_{64}]$

Slide credit: Andrew Ng

SC Cont'd

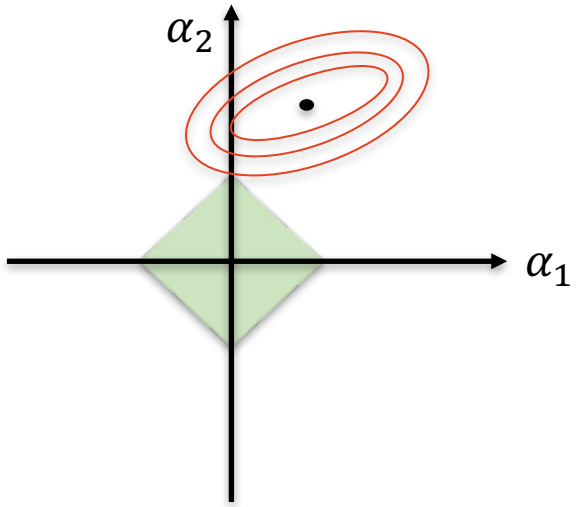
Let $\boldsymbol{X} = \{\boldsymbol{x}_i \in \boldsymbol{R}^d\}_{1 \le i \le n}$ be a set of observations. The cost function becomes the reconstruction error + sparsity term:

$$J(\theta) = \sum_{i=1}^{n} \left(\left\| \boldsymbol{x}_i - \sum_{j=1}^{k} \alpha_{ij} \phi_j \right\|^2 + \beta \sum_{j=1}^{k} s(\alpha_{ij}) \right)$$

Sparsity function $s(\alpha_j)$:

L0: $s(\alpha_j) = 1(\alpha_j \neq 0)$;

L1: $s(\alpha_j) = |\alpha_j|$;



SC Cont'd

Without regularization on the basis vector, the sparsity term can be arbitrarily small by rescaling α and ϕ.

$$\min_{\alpha,\phi} J(\theta) = \sum_{i=1}^{n} \left(\left\| \boldsymbol{x}_i - \sum_{j=1}^{k} \alpha_{ij} \phi_j \right\|^2 + \beta \sum_{j=1}^{k} s(\alpha_{ij}) \right)$$

$$\text{s.t. } \|\phi_j\|^2 \leq 1, \forall j = 1, \ldots, k$$

$$\min_{\alpha,\phi} J(\theta) = \sum_{i=1}^{n} \left(\left\| \boldsymbol{x}_i - \sum_{j=1}^{k} \alpha_{ij} \phi_j \right\|^2 + \beta \sum_{j=1}^{k} s(\alpha_{ij}) \right) + \gamma \sum_{j=1}^{k} \|\phi_j\|^2$$

SC Cont'd

$$\min_{\alpha,\phi} J(\theta) = \sum_{i=1}^{n} \left(\left\| \boldsymbol{x}_i - \sum_{j=1}^{k} \alpha_{ij} \phi_j \right\|^2 + \beta \sum_{j=1}^{k} s(\alpha_{ij}) \right) + \gamma \sum_{j=1}^{k} \|\phi_j\|^2$$

Vectorization: $\boldsymbol{\phi} = [\phi_1, \dots, \phi_k]$, $\boldsymbol{A} = [\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_n]$.

$$\min_{\alpha,\phi} J(\theta) = \|\boldsymbol{X} - \boldsymbol{\phi A}\|^2 + \beta s(\boldsymbol{A}) + \gamma \|\boldsymbol{\phi}\|^2$$

Alternating optimization:

1. Fix the basis (dictionary) $\boldsymbol{\phi}$, optimize $\boldsymbol{A}$ (a standard LASSO problem).
2. Fix $\boldsymbol{A}$, optimize $\boldsymbol{\phi}$ (a convex QP problem).

Outline

1. Background
2. Principal Component Analysis (PCA)
3. Autoencoder (AE)
4. Sparse Coding (SC)