

# 臺灣大學人工智慧中心

科技部人工智慧技術暨全幅健康照護聯合研究中心

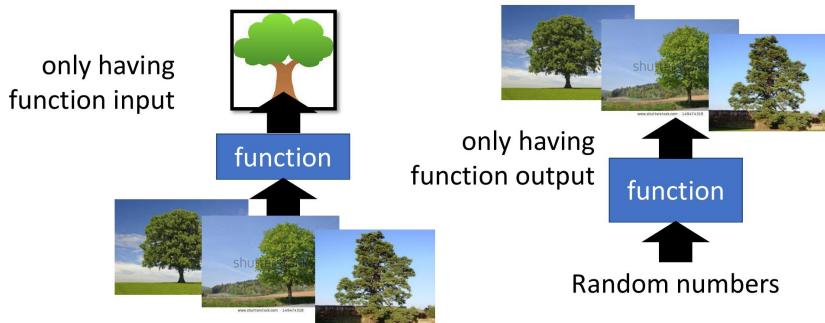
<http://aintu.tw>

## Linear Method

### Dimension Reduction

#### Unsupervised Learning

- Dimension Reduction  
(化繁為簡)
- Generation (無中生有)



Dimension Reduction 分成兩種：

- 一種做的事情，叫做化繁為簡。
- 那另外一個是 Generation，也就是無中生有。

### 化繁為簡

又可以分成兩大類，

- 一種是做 Clustering
- 一種是做 Dimension Reduction

所謂的化繁為簡的意思是說現在有很多種不同的 input，找一個 function 它可以 input 看起來像樹的東西，output 則是抽象的樹。

也就是把本來比較複雜的 input，變成比較簡單的 output。

畢竟在做 Unsupervised Learning 的時候通常只會有 function 的其中一邊，所能擁有的 training data，就只有一大堆的 image，不知道說它的 output，應該要是長什麼樣子。

### 無中生有

那另外一個 Unsupervised Learning 會做的事情呢。

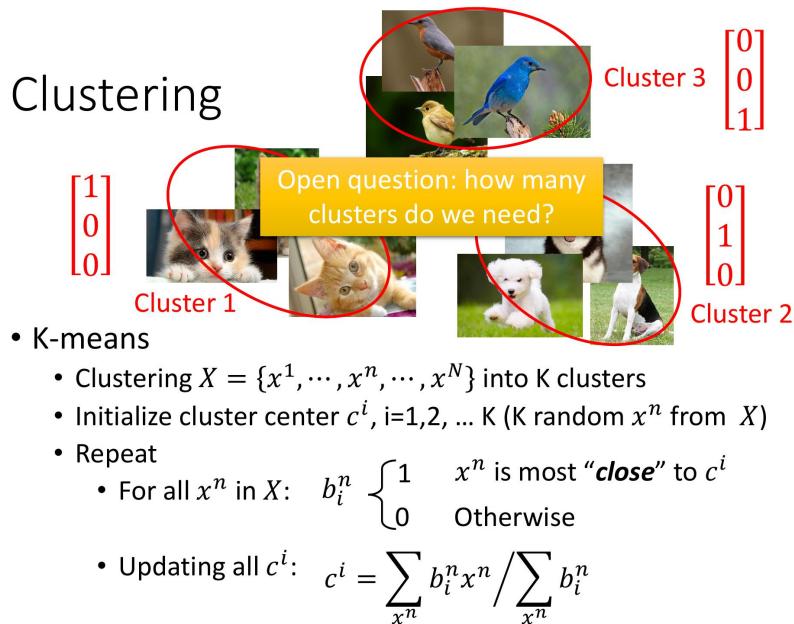
要找一個 function，隨機給它一個 input。比如說輸入一個數字 1，然後，它就會 output 這棵樹；輸入數字 2，就 output 另一棵樹；輸入數字 3，就 output 另一棵。

在這個 task 裡面，要找的是這個可以畫圖的 function，只有這一個 function 的 output，沒有這一個 function 的 input。

只有一大堆的 image，但是不知道輸入怎麼樣的 code，才可以得到這些 image。

這一份講義主要著重於在 linear 的 Dimension Reduction 上。

## Clustering



### 什麼是 clustering 呢？

clustering 就是一大堆的 image，假設現在要做 clustering，就是把它們分成一類、一類、一類的。

把本來有些不同的 image，用同一個 class 來表示就可以做到化繁為簡這一件事情。

到底應該要有幾個 cluster，就跟 neural network 要幾層一樣，是 empirical 地去決定。

### K-means

在 clustering 的方法裡面，最常用的叫做 K-means。

K-means 就是一大堆的 data，它們都是 unlabeled 的， $x^1$  一直到  $x^N$ 。

目標：要把它做成 K 個 cluster

1. 先找這些 cluster 的 center，每一個 cluster 都要先找一個 center，有 K 個 cluster，我們就需要  $c^1$  到  $c^K$  個 center。（初始的 center 可以從 training data 裡面，隨機的找 K 個 object 出來）
2. 接下來，對所有在 training data 裡面的 x，重複以下的 3. 4. 步驟。
3. 決定每一個 object 屬於 1 到 K 的哪一個 cluster（假設 object,  $x^n$  跟第 i 個 cluster 的 center 最接近，那  $x^n$  就屬於  $c^i$ ）

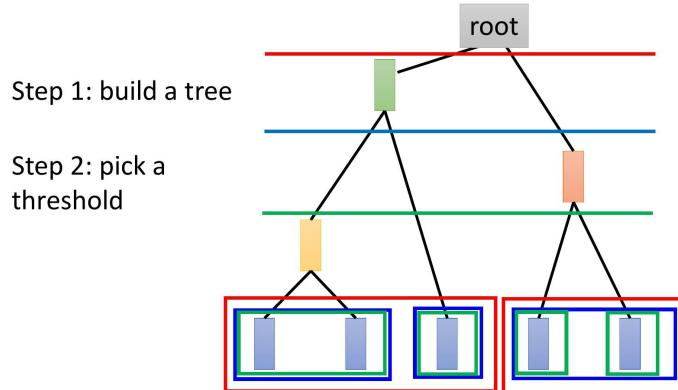
- 接下來，你update 你的 cluster。（假設要 update 第 i 個 cluster 的 center，就把所有屬於第 i 個 cluster 的 object通通拿出來，做平均得到第 i 個 cluster 的新center。）

然後，反覆的進行步驟2到4。

## Hierarchical Agglomerative Clustering (HAC)

### Clustering

- Hierarchical Agglomerative Clustering (HAC)



clustering 有另外一個方法叫做，Hierarchical Agglomerative Clustering (HAC)。

這個方法，是先建一個 tree。

假設你現在有 5 個 example：

- 把這 5 個 example 兩兩、兩兩去算他的相似度，然後挑最相似的那一個 pair 出來。（假設現在最相似的 pair，是第一個和第二個 example，那就把第一個 example 和第二個 example merge 起來。比如說，把它們平均起來，得到一個新的 vector。）
- 接下來，只剩下 4 筆 data，把這 4 筆 data，再兩兩去算它的相似度，再把它們 merge 起來，把它們平均起來得到另外一筆 data。
- 現在只剩三筆 data，再去兩兩算它的 singularity。（然後，黃色這一個和中間這一個最像，就再把它們平均起來）
- 最後只剩紅色跟綠色，只好把它平均起來。
- 就得到這個 tree 的 root，就根據這 5 筆 data，它們之間的相似度，建立出一個 tree structure。

這個 tree structure 可以告訴我們哪些 example 是比較像的，比較早分支代表比較不像，因此可以根據想要分的cluster數目，對這個tree做拆解。

### K-means VS HAC

HAC 跟 K-means最大的差別是如何決定cluster 的數目。

- 在 K-means 裡面，要事先決定你 K 的 value 是多少。
- HAC不直接決定幾個 cluster，而決定你要在樹上切幾刀。

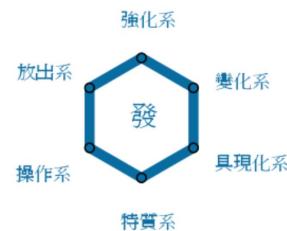
# Dimension Reduction

## Distributed Representation

- Clustering: an object must belong to one cluster  
小傑是強化系
- Distributed representation

小傑是

強化系	0.70
放出系	0.25
變化系	0.05
操作系	0.00
具現化系	0.00
特質系	0.00



在做 cluster 的時候，就是以偏概全，因為每一個 object 都必須要屬於某一個 cluster。

如果你只是把你手上所有的 object 分別 assign 到它屬於哪一個 cluster，這樣是以偏概全。

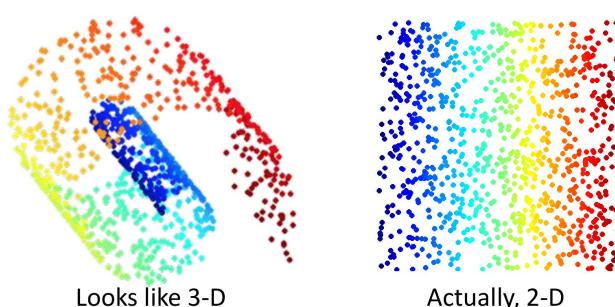
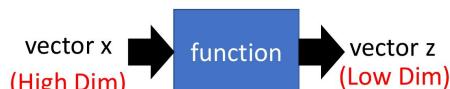
應該要用一個 vector 來表示 object，這個 vector 裡面的每一個 dimension，代表了某一種特質、某一種 attribute。這件事情就叫做 Distributed 的 representation。

如果原來的 object 是一個非常 high dimension 的東西，比如說 image。

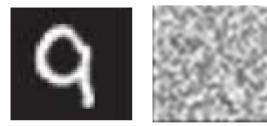
它用它的attribute，把它用它的特質來描述，就會從比較高維的空間變成比較低維的空間，這一件事情就叫做 Dimension Reduction。

實際舉例，其實只需要在 2D 的空間就可以描述這個 3D 的 information，根本不需要把這個問題放到 3D 來解，這樣是把問題複雜化。

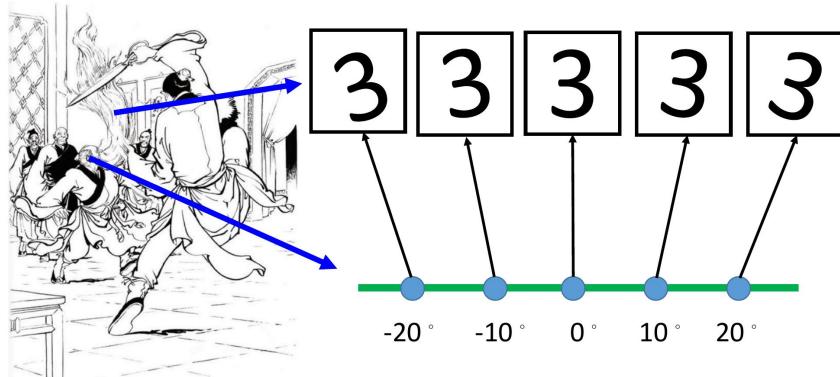
## Dimension Reduction



## Dimension Reduction



- In MNIST, a digit is  $28 \times 28$  dims.
  - Most  $28 \times 28$  dim vectors are not digits



另一個比較具體的例子，MNIST

在 MNIST 裡面，每一個 input 的 digit 都是一個 image，都用  $28 \times 28$  的 dimension 來描述。

但是，實際上多數  $28 \times 28$  的 dimension 的 vector 轉成一個 image 看起來都不像是一個數字。所以在這個  $28$  維 \*  $28$  維的空間裡面是 digit 的 vector，其實是很少的。所以，其實描述一個 digit，或許根本不需要用到  $28 \times 28$  維。

比如這邊有一堆 3，然後這一堆 3

如果你是從 pixel 來看待它的話要用  $28$  綴 \*  $28$  綴。然而，實際上這一些 3，只需要用一個維度，就可以來表示。因為這些 3 就只是說把原來的 3 放正，是中間這張 image 右轉  $10$  度轉就變成它，右轉  $20$  度就變它，左轉  $10$  度變它，左轉  $20$  度就變它。

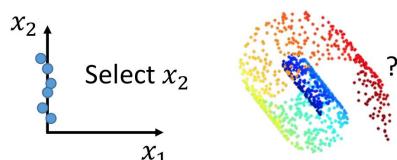
所以，唯一需要記錄的事情只有今天這張 image，它是左轉多少度、右轉多少度，就可以知道說，它在  $28$  綴的空間裡面應該長什麼樣子。

## Dimension Reduction 實作

### Distributed Representation



- Feature selection



- Principle component analysis (PCA)  
[Bishop, Chapter 12]

$$z = Wx$$

- 找一個 function，function 的 input 是一個 vector,  $x$ ，它的 output 是另外一個 vector,  $z$ 。
- 因為是 Dimension Reduction，所以 output 的這個 vector,  $z$ ，它的 dimension 要比 input 的  $x$  還要小。

## Feature Selection

最簡單的方法是 Feature Selection，如果把 data 的分布拿出來看一下，本來在二維的平面，但是，你發現都集中在  $x_2$  的 dimension 而已，所以， $x_1$  這個 dimension 沒什麼用，把它拿掉，就等於是做到 Dimension Reduction 這件事。

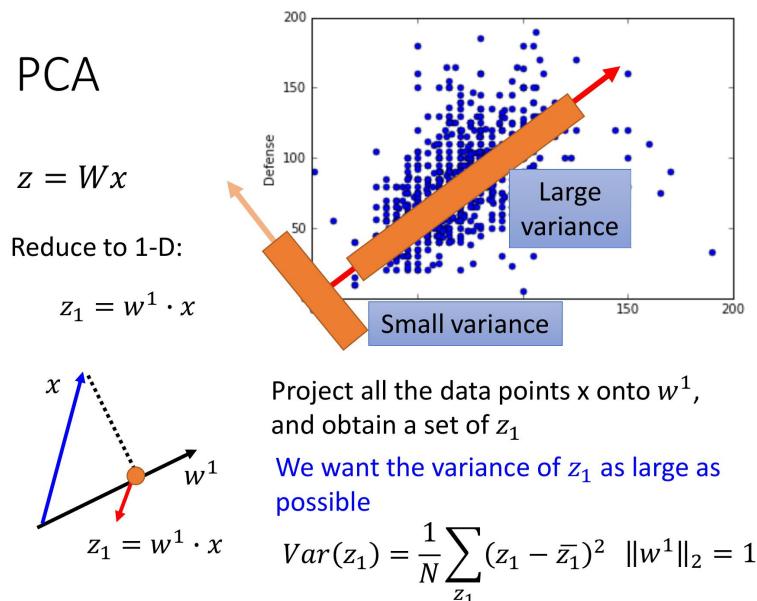
## Principle Component Analysis

另外一個常見的方法叫做 Principle Component Analysis (PCA)。

假設這個 function 是一個的 linear function，則 input,  $x$  跟這個 output,  $z$  之間的關係就是一個 linear 的 transform。也就是把這個  $x$  乘上一個 matrix,  $W$ ，就能得到它的 output,  $z$ 。

根據一大堆的  $x$ ，我們要把這個  $W$  找出來。

## PCA (Principle Component Analysis)



PCA 要做的事情就是找這一個  $W$ 。

1. 假設現在考慮一個 one dimensional 的 case，只要把 data project 到一維的空間上面，也就是  $z$  只是一個一維的 vector，也就是一個 scalar，那  $W$  實際上就是一個一個 row，用  $w^1$  來表示。
  - 把  $x$  跟  $w$  的第一個 row,  $w^1$  做 inner product，就能得到一個 scalar,  $z_1$ 。
  - 假設  $w^1$  的長度是 1， $w^1$  的 2-norm 是 1。
2.  $w$  跟  $x$  是高維空間中的一個點， $w^1$  是高維空間中的一個 vector，那所謂的  $z_1$  就是  $x$  在  $w^1$  上面的投影，這個投影的值就是  $w^1$  跟  $x$  的 inner product。現在要做的事情就是把一堆  $x$  透過  $w^1$  把它投影變成  $z_1$ 。
3. 則現在的目標是希望選一個  $w^1$  經過 projection 以後，得到的這些  $z_1$  的分布是越大越好（也就是說，我們不希望通過這個 projection 以後，所有的點通通擠在一起把本來 data point 和 data point 之間的奇異度拿掉了）。

所以，我們希望找一個 projection 的方向，它可以讓projection 後的 variance 越大越好。

如果我們要用 equation 來表示它的話，就會說現在要去 maximize 的對象是  $z_1$  的 variance， $z_1$  的 variance 就是 summation over 所有的  $z_1$  ( $z_1 - z_1\bar{}$ ) 的平方， $z_1\bar{}$  就是做  $z_1$  的平均。

## PCA

$$z = Wx$$

Reduce to 1-D:

$$z_1 = w^1 \cdot x$$

$$z_2 = w^2 \cdot x$$

$$W = \begin{bmatrix} (w^1)^T \\ (w^2)^T \\ \vdots \end{bmatrix}$$

Orthogonal matrix

Project all the data points  $x$  onto  $w^1$ , and obtain a set of  $z_1$

We want the variance of  $z_1$  as large as possible

$$Var(z_1) = \frac{1}{N} \sum_{z_1} (z_1 - \bar{z}_1)^2 \quad \|w^1\|_2 = 1$$

We want the variance of  $z_2$  as large as possible

$$Var(z_2) = \frac{1}{N} \sum_{z_2} (z_2 - \bar{z}_2)^2 \quad \|w^2\|_2 = 1$$

$$w^1 \cdot w^2 = 0$$

再來，可能不只要投影到一維，比如說投影到一個二維的平面。

1. 就把  $x$  跟另外一個  $w^2$ ，做 inner product，得到  $z_2$ 。（這個  $w^1$  跟  $w^2$  的 transpose 排起來就是， $W$  的第一個 row 跟第二個 row）
2. 基本上，延續上一維的做法，首先令  $w^2$  它的 2-norm 是 1。
3. 接下來這個  $z_2$  它的分佈也是越大越好。

這裡有唯一的constraint 就是  $w^1$  跟  $w^2$  必須是 orthogonal的，否則就只是在找一樣的  $w$ 。

更多的維度就以此類推，但記得  $W$  的每個 row 之間必須是 orthogonal的。

## Warning of Math

以下是實際的解法。

$$\begin{aligned}
z_1 &= w^1 \cdot x \\
\text{PCA} \quad \bar{z}_1 &= \frac{1}{N} \sum z_1 = \frac{1}{N} \sum w^1 \cdot x = w^1 \cdot \frac{1}{N} \sum x = w^1 \cdot \bar{x} \\
Var(z_1) &= \frac{1}{N} \sum_{z_1} (z_1 - \bar{z}_1)^2 & (a \cdot b)^2 = (a^T b)^2 = a^T b a^T b \\
&= \frac{1}{N} \sum_x (w^1 \cdot x - w^1 \cdot \bar{x})^2 & = a^T b (a^T b)^T = a^T b b^T a \\
&= \frac{1}{N} \sum (w^1 \cdot (x - \bar{x}))^2 \\
&= \frac{1}{N} \sum (w^1)^T (x - \bar{x})(x - \bar{x})^T w^1 \\
&= (w^1)^T \left[ \frac{1}{N} \sum (x - \bar{x})(x - \bar{x})^T \right] w^1 & \boxed{\begin{array}{l} \text{Find } w^1 \text{ maximizing} \\ (w^1)^T S w^1 \\ \|w^1\|_2 = (w^1)^T w^1 = 1 \end{array}} \\
&= (w^1)^T Cov(x) w^1 & S = Cov(x)
\end{aligned}$$

前半部使用Lagrange multiplier，或是Gradient Descent (preferred)。

$$\text{Find } w^1 \text{ maximizing } (w^1)^T S w^1 \quad (w^1)^T w^1 = 1$$

$$S = Cov(x) \quad \text{Symmetric} \quad \text{Positive-semidefinite} \\ (\text{non-negative eigenvalues})$$

Using Lagrange multiplier [Bishop, Appendix E]

$$g(w^1) = (w^1)^T S w^1 - \alpha((w^1)^T w^1 - 1)$$

$$\left. \begin{array}{l} \partial g(w^1)/\partial w_1^1 = 0 \\ \partial g(w^1)/\partial w_2^1 = 0 \\ \vdots \end{array} \right\} \quad \begin{array}{l} Sw^1 - \alpha w^1 = 0 \\ Sw^1 = \alpha w^1 \quad w^1 : \text{eigenvector} \\ (w^1)^T S w^1 = \alpha (w^1)^T w^1 \\ = \alpha \quad \text{Choose the maximum one} \end{array}$$

$w^1$  is the eigenvector of the covariance matrix  $S$   
Corresponding to the largest eigenvalue  $\lambda_1$

Find  $w^2$  maximizing  $(w^2)^T S w^2 \quad (w^2)^T w^2 = 1 \quad (w^2)^T w^1 = 0$

$$g(w^2) = (w^2)^T S w^2 - \alpha((w^2)^T w^2 - 1) - \beta((w^2)^T w^1 - 0)$$

$$\begin{aligned} \partial g(w^2)/\partial w_1^2 &= 0 \\ \partial g(w^2)/\partial w_2^2 &= 0 \\ \vdots & \end{aligned} \left. \begin{array}{l} Sw^2 - \alpha w^2 - \beta w^1 = 0 \\ 0 - \alpha 0 - \beta 1 = 0 \\ = ((w^1)^T S w^2)^T = (w^2)^T S^T w^1 \\ = (w^2)^T S w^1 = \lambda_1 (w^2)^T w^1 = 0 \end{array} \right\}$$

$$Sw^1 = \lambda_1 w^1$$

$$\beta = 0: \quad Sw^2 - \alpha w^2 = 0 \quad Sw^2 = \alpha w^2$$

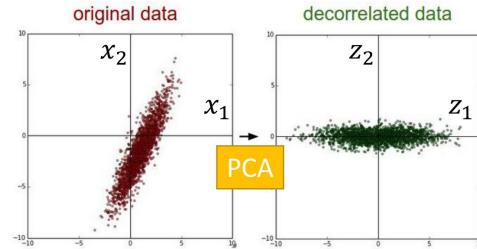
$w^2$  is the eigenvector of the covariance matrix  $S$   
Corresponding to the 2<sup>nd</sup> largest eigenvalue  $\lambda_2$

### PCA - decorrelation

$$z = Wx$$

$$Cov(z) = D$$

Diagonal matrix



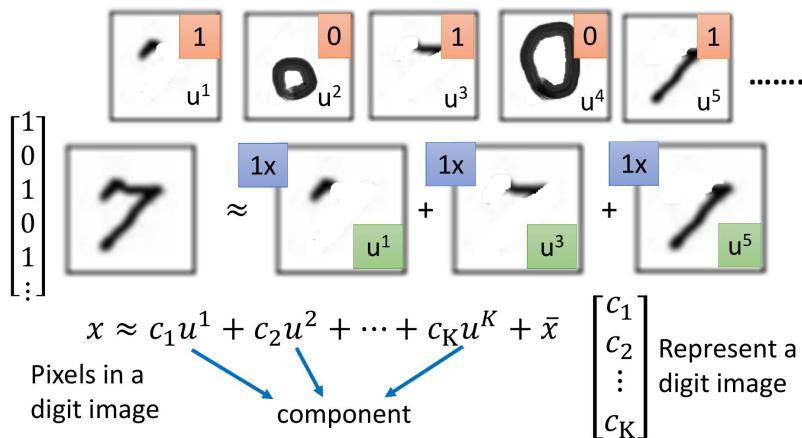
$$\begin{aligned} Cov(z) &= \frac{1}{N} \sum (z - \bar{z})(z - \bar{z})^T = W S W^T \quad S = Cov(x) \\ &= W S [w^1 \dots w^K] = W [S w^1 \dots S w^K] \\ &= W [\lambda_1 w^1 \dots \lambda_K w^K] = [\lambda_1 W w^1 \dots \lambda_K W w^K] \\ &= [\lambda_1 e_1 \dots \lambda_K e_K] = D \quad \text{Diagonal matrix} \end{aligned}$$

最後Z的covariance會是diagonal的，也就代表整個PCA會做一個decorrelation，而Z的維度可以當作是一種新的feature。

## PCA — Another Point of View

## PCA – Another Point of View

Basic Component:



PCA第一次找出來的  $w^1$  是 covariance matrix，對應到最大的 eigenvalue 的 eigenvector; 然後，第二個找出來的  $w^2$  對應到第二大的 eigenvalue 的 eigenvector，以此類推.... (上面就有一個很長的證明來說明這麼做的話，每一次投影的時候都可以讓 variance 最大)

以下是另一個比較直觀的說明：

假設現在考慮的是手寫數字。

這些數字其實是由一些basic的 component 所組成的，這些 basic 的 component 可能就代表筆劃（有斜的直線、橫的直線、比較長的直線、還有小圈、大圈等等...）這些 basic 的 component 把它加起來以後就可以得到一個數字。

這些 basic 的 component (寫作  $u^1, u^2, u^3$  等等) 實際上就是一個一個的 vector，假設現在考慮的是 MNIST 的話，MNIST 的一張 image 是 2828 pixel，那這些 component，其實也就是 2828 級的 vector。

把這些 vector 透過不同的線性組和加起來以後，所得到的總和 vector，就代表了一個 digit。

所以，每一張 image 就是有一堆 component 的 linear combination，再加上它的平均所組成的。

因此可以用這些 component 和 weight ( $c_1, c_2, c_3 \dots$ ) 來描述一張 image，如果 component 的數目比 pixel 的數目少的話，那這個描述，通常會是比較有效的

(舉例來說，7 是一倍的  $u^1$ ，一倍的  $u^3$ ，一倍的  $u^5$  所組合而成所以，7 你就可以說它是一個 vector，它第一維、第三維、第五維是 1)

## PCA – Another Point of View

$$x - \bar{x} \approx c_1 u^1 + c_2 u^2 + \cdots + c_K u^K = \hat{x}$$

Reconstruction error:

$$\| (x - \bar{x}) - \hat{x} \|_2 \quad \text{Find } \{u^1, \dots, u^K\} \text{ minimizing the error}$$

$$L = \min_{\{u^1, \dots, u^K\}} \sum \left\| (x - \bar{x}) - \left( \sum_{k=1}^K c_k u^k \right) \right\|_2$$

PCA:  $z = Wx$

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} (w_1)^T \\ (w_2)^T \\ \vdots \\ (w_K)^T \end{bmatrix} x \quad \begin{array}{l} \{w^1, w^2, \dots, w^K\} \text{ (from PCA) is the} \\ \text{component } \{u^1, u^2, \dots, u^K\} \\ \text{minimizing L} \end{array}$$

Proof in [Bishop, Chapter 12.1.2]

在 PCA 裡面，我們要找一個 matrix,  $W$ ，使原來的 vector,  $x$  乘上  $W$  以後，得到 Dimension Reduction 以後的結果， $z$ 。（可以把  $W$  的每一個 row 都寫出來  $w_1, w_2$  一直到  $w_K$ ，都是 covariance matrix 的 eigenvector。）

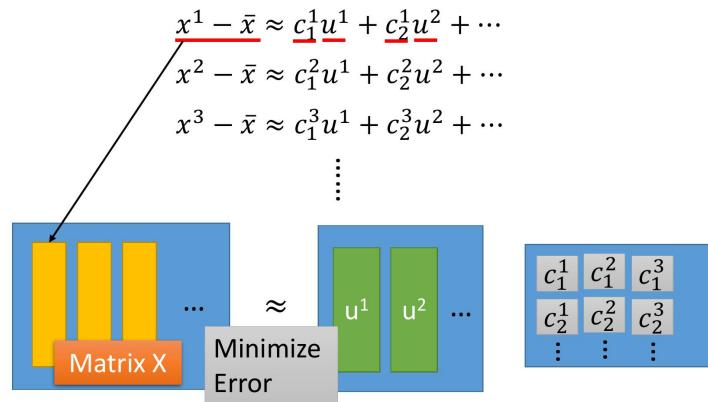
事實上，如果要解這個圖上的式子（透過把平均值移項得到的式子），找出  $u^1, u^2$  到  $u^K$ 。這個  $w^1$  到  $w^K$ ，就是由 PCA 找出來的這個解

也就是可以讓上面這一個式子最小化，就可以找到讓這個 Reconstruction error 最小的  $u^1$  到  $u^K$ 。

$$x - \bar{x} \approx c_1 u^1 + c_2 u^2 + \cdots + c_K u^K = \hat{x}$$

Reconstruction error:

$$\| (x - \bar{x}) - \hat{x} \|_2 \quad \text{Find } \{u^1, \dots, u^K\} \text{ minimizing the error}$$

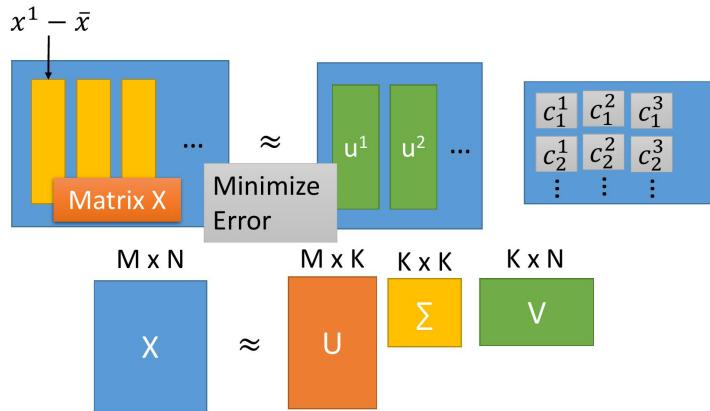


實際以data為例子，現在在 database 裡面有一大堆的  $x$ ，如果把所有的 data都用剪掉平均值後的線性組合來表示的話。

就能得到一個 matrix，在這個 matrix 的橫軸，column 的數目就是 data 的數目

現在要做的事情就是用這一個 matrix 去乘上另一個 matrix，希望越接近這左邊的matrix 越好

所以要 minimize 這兩個相乘以後得到的 matrix 跟左邊這個 matrix 之間的差距(Reconstruction error)。



K columns of U: a set of orthonormal eigen vectors corresponding to the K largest eigenvalues of  $XX^\top$

This is the solution of PCA

SVD:

[http://speech.ee.ntu.edu.tw/~tlkagk/courses/LA\\_2016/Lecture/SVD.pdf](http://speech.ee.ntu.edu.tw/~tlkagk/courses/LA_2016/Lecture/SVD.pdf)

那怎麼解這個問題呢？假如你有修過大一線代的話，就知道這個問題是怎麼解的，連結是李老師教線代的時候的投影片。

所以，根據 PCA，所找出來的那些  $w$ ，找出來的 Dimension Reduction 的 transform，就是在 minimize Reconstruction error。

PCA 裡面你得到的那些  $W$ ，其實就是 component。

## NN vs PCA

<

PCA looks like a neural network with one hidden layer (linear activation function)

Autoencoder

If  $\{w^1, w^2, \dots, w^K\}$  is the component  $\{u^1, u^2, \dots, u^K\}$

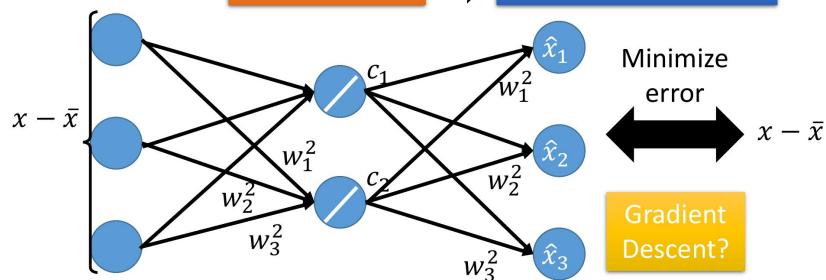
$$\hat{x} = \sum_{k=1}^K c_k w^k \quad \leftrightarrow \quad x - \bar{x}$$

To minimize reconstruction error:  
 $c_k = (x - \bar{x}) \cdot w^k$

$K = 2$ :

It can be deep.

Deep Autoencoder



用NN去解linear combination來minimize reconstruction error跟PCA做出來是會不一樣的。

因為PCA 解出來這些  $W$  它們是 orthonormal 的，它們是垂直的。

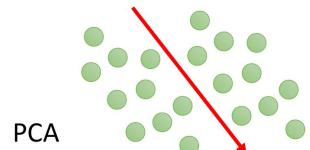
今天如果用 neural network去Gradient Descent 硬解，並不可能讓 Reconstruction error 比 PCA 找出來的還要小。所以，如果是在 linear 的情況下，直接用 PCA 來找這個  $w$ ，是比較快的。

用 neural network 的好處就是，它可以是 deep 的，可以改成很多的 hidden layer，這涉及到之後會講的 Deep Autoencoder。

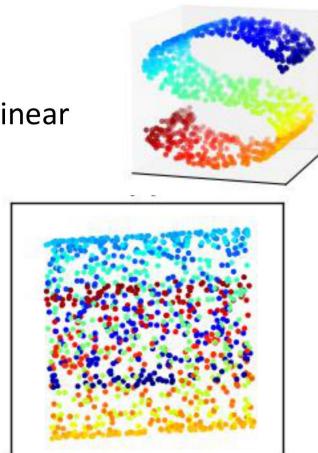
## PCA的缺點

### Weakness of PCA

- Unsupervised



- Linear



[http://www.astroml.org/book\\_figures/chapter7/fig\\_S\\_manifold\\_PCA.html](http://www.astroml.org/book_figures/chapter7/fig_S_manifold_PCA.html)

- 由於是unsupervised的，data太複雜或過多的話，PCA 可能會讓原本有區分的兩個cluster降維投影到單一平面後擠在一起。必須要用LDA(linear discriminant analysis)來引入labelled data，才能解決。
- PCA無法做到non-linear的transformation（比如說前面的3D捲軸例子），畢竟它是linear 的dimension reduction。

## PCA — Examples

以下是各種PCA的例子，以及他們所找出的pricipal components。

### 寶可夢

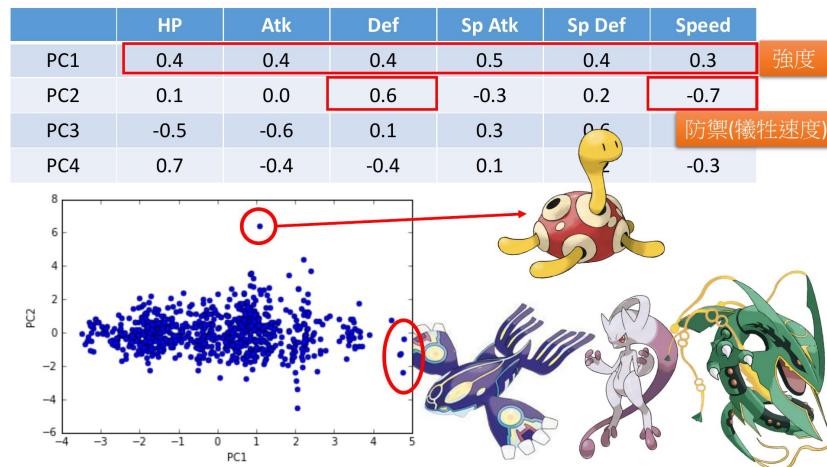
## PCA - Pokémon

- Inspired from:  
<https://www.kaggle.com/strakul5/d/abcsds/pokemon/principal-component-analysis-of-pokemon-data>
- 800 Pokemons, 6 features for each (HP, Atk, Def, Sp Atk, Sp Def, Speed)
- How many principle components?  $\frac{\lambda_i}{\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6}$

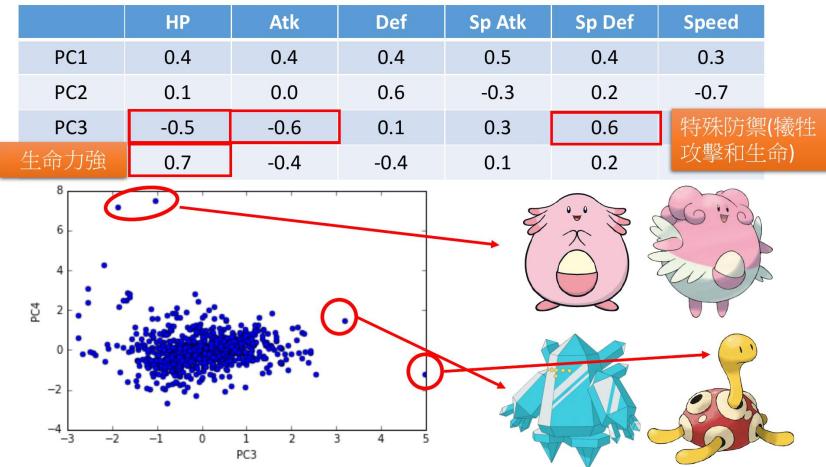
	$\lambda_1$	$\lambda_2$	$\lambda_3$	$\lambda_4$	$\lambda_5$	$\lambda_6$
ratio	0.45	0.18	0.13	0.12	0.07	0.04

Using 4 components is good enough

## PCA - Pokémon



## PCA - Pokémon

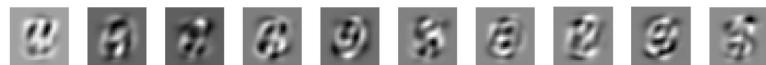


## MNIST 手寫數字

PCA - MNIST

$$= a_1 w^1 + a_2 w^2 + \dots$$

30 components:



Eigen-digits

## 人臉

## PCA - Face



30 components:



<http://www.cs.unc.edu/~lazebnik/research/spring08/assignment3.html> Eigen-face

臺灣大學人工智慧中心

科技部人工智慧技術暨全幅健康照護聯合研究中心

<http://aintu.tw>