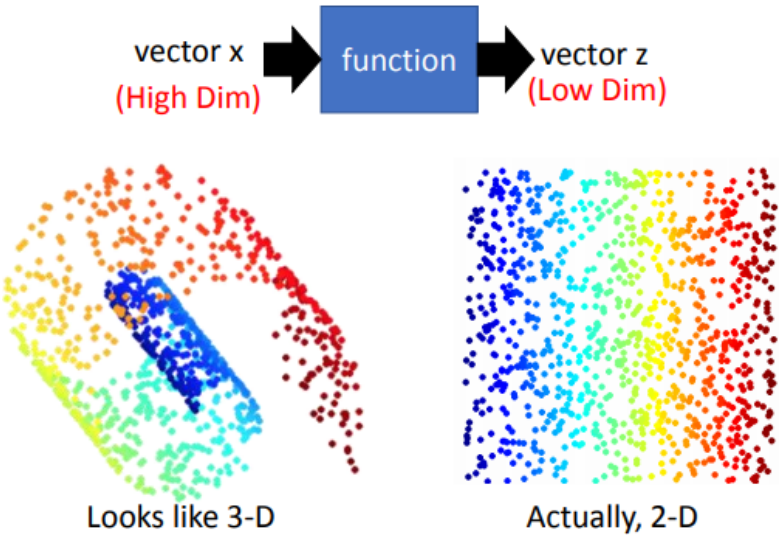
INDEX

Unsupervised Learning

Data 皆不具 label，且訓練時僅有 input 而無法直接獲得 output 的學習模式。主要可分作兩類，Dimension Reduction 與 Generation。

Dimension Reduction

基本精神為「化簡為繁」，意即把本來比較複雜的 input 變成比較簡單的 output。



一、Clustering

假設現在要做 image 的 clustering，那就是把一大堆的 image 分成好幾類。將本來有些不同的 image 都用同一個 class 來表示。

1. K-means

將一大堆的 unlabeled data 把他們分作 K 個 cluster。

首先就是找這些 cluster 的 center，從 training data 裡面隨機找 K 個 object 出來，當成 K 個 cluster 的 center。

接下來決定每一個 object 屬於 1 到 K 的哪一個 cluster。假設現在的 object x^n，跟第 i 個 cluster 的 center 最接近的話，那 x^n 就屬於 c^i。
簡而言之用一個 binary 的 value b （上標 n，下標 i）來代表第 n 個 object 有沒有屬於第 i 個 class，如果第 n 個 object 屬於第 i 個 class 的話，那這一個 binary 的 value 就是 1，反之就是 0。

接下來，就是 update cluster，把所有屬於第 i 個 cluster 的 object 做平均，得到第 i 個 cluster 的 center c^i。

最後就重複上述步驟即可。

- K-means
  - Clustering $X = \{x^1, \cdots, x^n, \cdots, x^N\}$ into K clusters
  - Initialize cluster center c^i, i=1,2, … K (K random x^n from X)
  - Repeat
    - For all x^n in X: $b_i^n \begin{cases} 1 & x^n \text{ is most "close" to } c^i \\ 0 & \text{Otherwise} \end{cases}$
    - Updating all c^i: $c^i = \sum_{x^n} b_i^n x^n \Big/ \sum_{x^n} b_i^n$

2. Hierarchical Agglomerative Clustering (HAC)

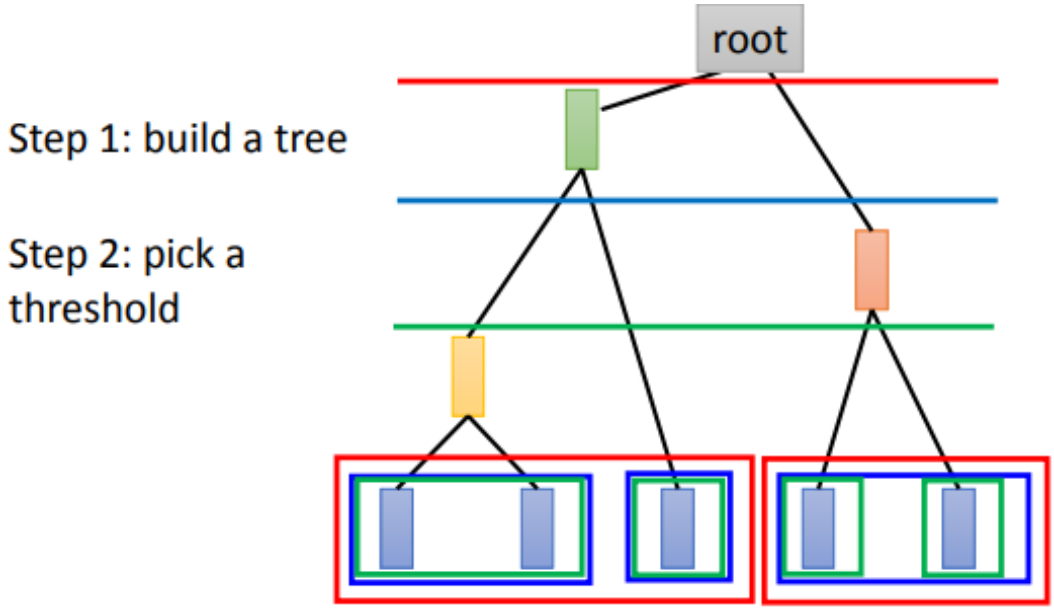
該方法是先建一個 tree，假設現在有 5 個 example，兩兩去算他的相似度，然後挑最相似的那一個 pair 出來。

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，像是對他們取平均得到一個新的 vector （下圖黃色方塊），同時代表第一個和第二個 example。

接下來變成有四個 example，再對這 4 筆 data 兩兩去計算他們的相似度，假設是第三筆和第四筆最像，那就再把他們 merge 起來，得到另外一筆 data（下圖紅色方塊）。

最終得到這個 tree 的 root，建立出一個 tree structure

接下來要決定在這個 tree structure 上面哪地方切一刀，就可將 example 分成好幾個 cluster。



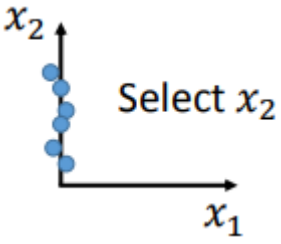
HAC 跟 K-means 最大的差別就是，如何決定 cluster 的數目。在 K-means 裡面要需要決定那個 K 的 value 是多少，而到底有多少個 cluster 是不容易決定的；HAC 的好處就是不直接決定幾個 cluster，而是決定要切在這個樹的 structure 的哪裡。

二、Distributed Representation (Dimension Reduction)

然而在做 cluster 的時候比較以偏概全，因為每一個 object 最後都必須要屬於某一個 cluster。實際上來說應該用一個 vector 來表示各個 object，那這個 vector 裡面的每一個 dimension 就代表了某一種 attribute。
該方式就稱 distributed representation。

1. Feature Selection

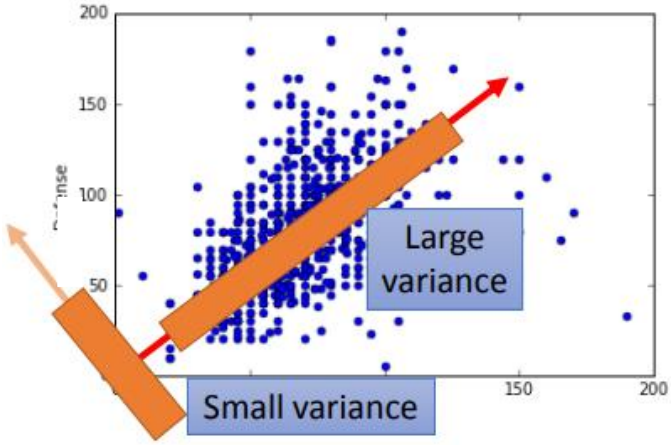
假設 data 的分布本來在二維的平面上，然後發現幾乎都集中在 x_2 的 dimension 而已，如此就可以拿掉 x_1 這個 dimension。
然而這個方法不見得總是有用，因為有很多時候處理的 case 是任何一個 dimension 都不能拿掉的。



2. Principle Component Analysis (PCA)

假設這個 function 是一個很簡單的 linear function，這個 input x 跟這個 output z 之間的關係就是 linear 的 transform，也就是把這個 x 乘上一個 matrix W 可得到 output z。

那現在要做的事情就是根據一大堆的 x 把 W 找出來。可理解成將 x 投影到 W 上，使他們在 W 有較大的 variance，而投影後在 W 上的點就是 z。



假設把 x 投影到一維，我們希望選一個 w^1，他經過 projection 以後，得到的這些 z_1 的分布越大越好。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（見上圖）。

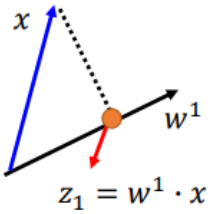
所以我們希望找一個 projection 的方向，它可以讓 projection 後的 variance 越大越好。因此現在要去 maximize 的對象是 z_1 的 variance，就是 summation over 所有的(z_1 - z_1\bar) 的平方；而 z_1\bar 就是做 z_1 的平均值。因此只要找到一個 w^1 讓 z_1 的 variance 最大就結束了。

再來可能不只要投影到一維，推廣至二維的情況大略相同。同樣是找一個 w^2 讓 z_2 的 variance 最大。

只是在一維時我們必須限制 w^1 的 2-norm，使 w^1 跟 x 做內積能直接得到 z_1；但是為避免 w^2 與 w^1 同值，必須再限制兩者的內積值為 1。

Reduce to 1-D:

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



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

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

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

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

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

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

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

Orthogonal matrix

We want the variance of z_2 as large as possible

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

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

(1) Lagrange Multiplier

前面提到 z_1 等於 w^1 跟 x 的內積值，那 z_1 的平均值就是 summation over 所有 w^1 跟 x 的內積再除以總數。可進一步簡化成 w^1 與 x\bar 的內積，即：

$$\bar{z_1} = w^1 \cdot \frac{1}{N}\sum x = w^1 \cdot \bar{x}$$

而 z_1 的 variance 可整理為 w^1 的 transpose 乘上 x 的 covariance 再乘上 w^1。
而此處用 S 來描述 x 的 covariance matrix。

所以現在要解的問題是找出一個 w^1 可以 maximize 該式。但這個 optimization 的對象是有 constraint 的，如果沒有 constraint 的話，這裡的 w^1 每一個值都變無窮大就結束了。所以這裡的 constraint 是說 w^1 的 2-norm 要等於 1。

$$Var(z_1) = \frac{1}{N}\sum_{z_1}(z_1 - \bar{z_1})^2$$

$$= \frac{1}{N}\sum_{x}(w^1 \cdot x - w^1 \cdot \bar{x})^2$$

$$= \frac{1}{N}\sum\left(w^1 \cdot (x - \bar{x})\right)^2$$

$$= \frac{1}{N}\sum (w^1)^T (x - \bar{x})(x - \bar{x})^T w^1$$

$$= (w^1)^T \boxed{\frac{1}{N}\sum (x - \bar{x})(x - \bar{x})^T} w^1$$

$$= (w^1)^T Cov(x) w^1 \qquad S = Cov(x)$$

$$(a \cdot b)^2 = (a^T b)^2 = a^T b a^T b$$

$$= a^T b (a^T b)^T = a^T b b^T a$$

Find w^1 maximizing

$$(w^1)^T S w^1$$

$$\|w^1\|_2 = (w^1)^T w^1 = 1$$

那有了這些以後，我們就要解這一個 optimization 的 problem。
由於 S 是 symmetric 又是 positive-semidefinite 的關係，他所有的 eigenvalue 都是 non-negative 的。

接著用 Lagrange multiplier（開頭如下式假設），

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

把這個 function 對 w 的第一個 element 做偏微分，再對第二個 element 做偏微分，依此類推。然後令這些式子通通等於 0，整理完後得到，會得到一個式子帶入 w^1 後使其為 0。

$$S(w^1) - \alpha(w^1) = 0$$

而 w^1 就是 S 的 eigenvector，接下來看哪一個 eigenvector 代到下式，可以 maximize 該式。

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

所以問題變成找一個 w^1 使 α 最大。而當 α 最大時，這個 α 就是最大的 eigenvalues λ_1；w^1 是對應到最大的 eigenvalue 的 eigenvector。

Find w^1 maximizing $(w^1)^T S w^1$ $(w^1)^T w^1 = 1$

$S = Cov(x)$ Symmetric Positive-semidefinite (non-negative eigenvalues)

Using Lagrange multiplier [Bishop, Appendix E]

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

$\partial g(w^1)/\partial w_1^1 = 0$
$\partial g(w^1)/\partial w_2^1 = 0$
$\vdots$

$Sw^1 - \alpha w^1 = 0$
$Sw^1 = \alpha w^1$ w^1 : eigenvector

$(w^1)^T S w^1 = \alpha(w^1)^T w^1$
$= \alpha$ Choose the maximum one

w^1 is the eigenvector of the covariance matrix S
Corresponding to the largest eigenvalue λ_1

同理，如果要找 w^2 的話，就要 maximize 根據 w^2 投影以後的 variance：

$$(\mathrm{w}^2)^{\mathrm{T}}\mathrm{S}(\mathrm{w}^2)$$

同樣假設 function g 裡面包含了你要 maximize 的對象，還有兩個 constraint（w^1 跟 w^2 他們是 orthogonal 的），然後分別乘上 α 跟 β。

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

接下來對所有的參數做偏微分得到這個值：

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

接著式子左邊同乘 w^1 的 transpose 變為：

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

紅字部分為一個 scalar (vector* matrix* vector)，而 scalar 在做 transpose 以後還是他自己，所以 transpose 結果是一樣的，得到：

$$(w^1)^T S(w^2) = (w^2)^T(S^T)(w^1) = (w^2)^T S(w^1)$$

（因為 S 是 symmetric 的，所以 transpose 以後還是他自己）

接下來我們已經知道 w^1 是 S 的 eigenvector，而且它對應到最大的 eigenvalue λ^1，所以寫為下式：

$$\because S(w^1) = (\lambda^1)(w^1)$$

$$\therefore (w^2)^T S(w^1) = (w^2)^T(\lambda^1)(w^1) = (\lambda^1)(w^1)(w^2)^T$$

因為$(w^1)*(w^2)^T$ 又等於 0 (orthogonal)，所以得到的結論是如果 β 等於 0 的話，剩下的 $S*(w^2)$會等於 $\alpha*(w^2)$。

所以 w^2 也是一個 eigenvector 且必須跟 w^1 orthogonal，故選第二大的 w^2，然後他對應到第二大的 eigenvalue λ^2。
其餘維度依此類推。

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

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

$\partial g(w^2)/\partial w_1^2 = 0$

$\partial g(w^2)/\partial w_2^2 = 0$

$\vdots$

$$Sw^2 - \alpha w^2 - \beta w^1 = 0$$

$$0 - \alpha\, 0 - \beta\, 1 = 0$$

$$= \left((w^1)^T S w^2\right)^T = (w^2)^T S^T w^1$$

$$= (w^2)^T S w^1 = \lambda_1 (w^2)^T w^1 = 0$$

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

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

w^2 is the eigenvector of the covariance matrix S
Corresponding to the 2nd largest eigenvalue λ_2

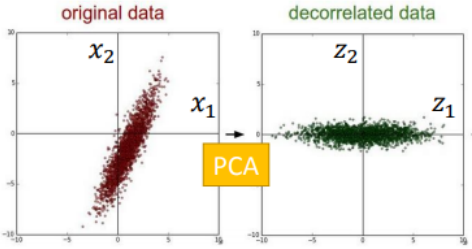
另外 PCA 中 z 的 covariance 會是一個 diagonal matrix。
也就是說，假設 PCA 所得到的新的 feature z 給其他的 model 描述某一個 class 的 distribution （假設為 generative model）。
那在做這個 Gaussian 的假設的時候，假設說 input data 的 covariance 就是 diagonal，且不同的 dimension 之間沒有 correlation，這樣一來減少參數量。
所以他就可以用比較簡單的 model 來處理 input data，避免 overfitting 的情形

PCA - decorrelation

$z = Wx$

$Cov(z) = D$

Diagonal matrix



$$Cov(z) = \frac{1}{N}\sum (z - \bar{z})(z - \bar{z})^T = WSW^T \quad S = Cov(x)$$

$$= WS[w^1 \quad \cdots \quad w^K] = W[Sw^1 \quad \cdots \quad Sw^K]$$

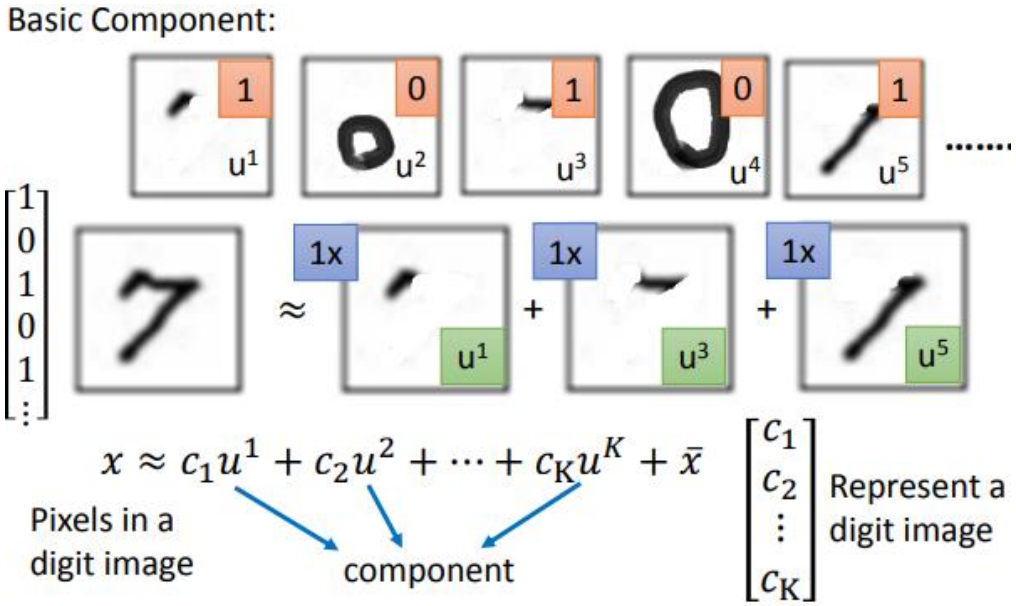
$$= W[\lambda_1 w^1 \quad \cdots \quad \lambda_K w^K] = [\lambda_1 W w^1 \quad \cdots \quad \lambda_K W w^K]$$

$$= [\lambda_1 e_1 \quad \cdots \quad \lambda_K e_K] = D$$ Diagonal matrix

(2) SVD

假設在考慮的是 MNIST，這些數字其實是由一些 basic 的 component（筆畫）所組成的。那這些 component 寫作 u^1, u^2, u^3 等等。則 input x 會等於 u^1 這個 component 乘上 c^1 加上 u^2 這個 component 乘上 c^2，以此類推，然後再加上 x\bar 代表所有的 image 的平均。

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



接著這一些 linear combination 的結果減去 x\bar，該值必須與目標值 x\head 越近越好，即：

$$x - \bar{x} \approx c_1u^1 + c_2u^2 + \cdots + c_Ku^K = \hat{x}$$

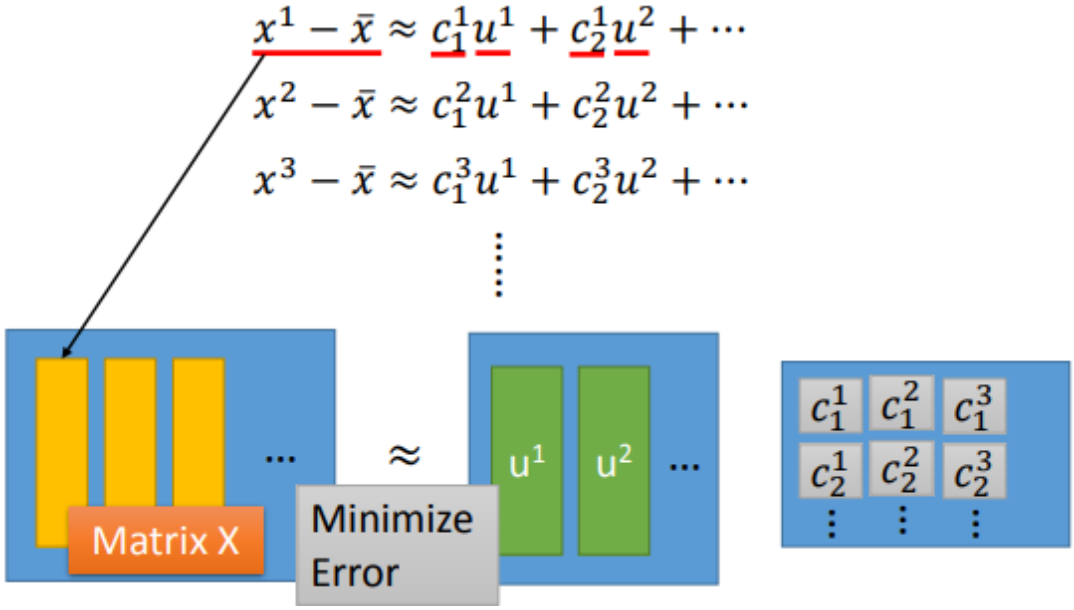
因此就必須找 K 個 vector 去 minimize 他們的距離 (reconstruction error)

Reconstruction error：$\|(x - \bar{x}) - \hat{x}\|_2$

$$L = \min_{\{u^1,\ldots,u^K\}} \sum \|(x - \bar{x}) - \hat{x}\|_2$$

$$\hat{x} = \sum_{k=1}^{K} c_k u^k$$

接下來，可進一步將 reconstruction error 表示為 matrix 的乘積。



接著可以用 SVD 把 matrix X 拆成 U，Σ 與 V 三個 matrix 的乘積，U 就是代表 matrix u^K；Σ*V 就是代表 matrix c_K。

然後 U 這個 matrix，他的 k 個 column，其實就是一組 orthonormal vector，對應到的就是 $X*X^T$ 最大的 k 個的 eigenvector。

而這個 $X*(X^T)$就是 covariance matrix，也就是 PCA 找出來的那一些 w（covariance matrix 的 eigenvector）等同於解出來的 U 的每一個 column 的 vector。

換句話說，根據 PCA 找出來的那些 w 其實就是在 minimize 這個 reconstruction error；那 Dimension Reduction 的結果就是這些 vector。

(3) Neural Network

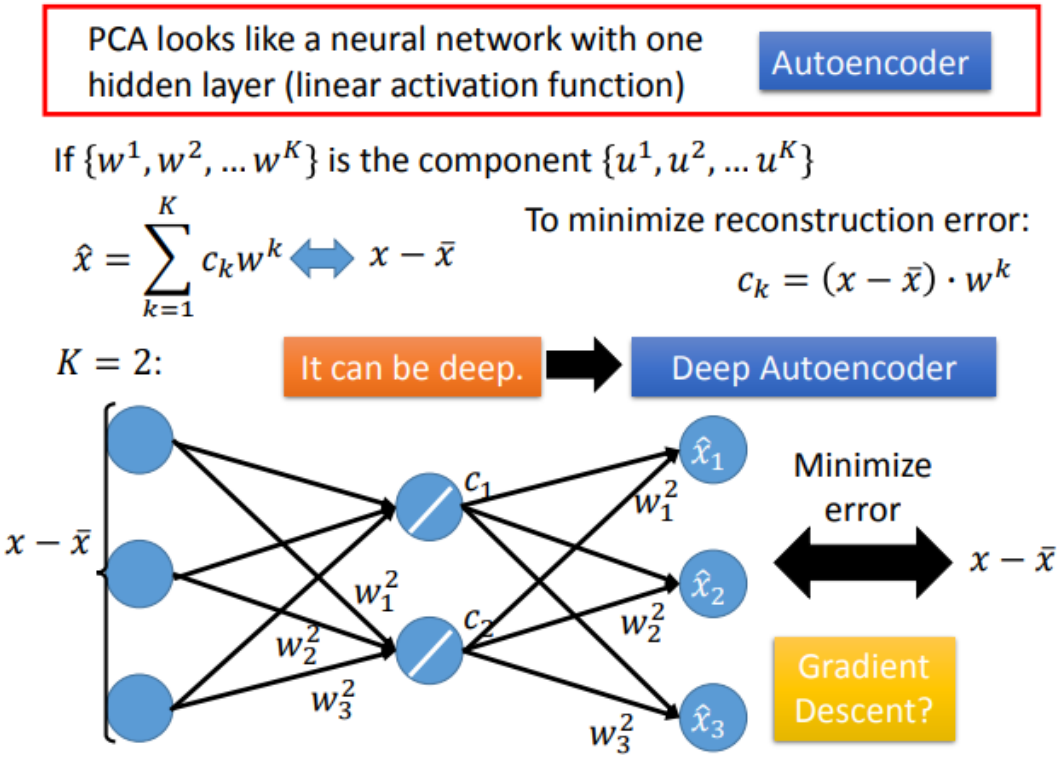
已知從用 PCA 找出來的 w^1 到 w^K 就是 K 個 component，u^1, u^2 到 u^K，再根據 component linear combination 得到的結果叫做 x\head，也就是(w^K)*c_k 做 linear combination 的結果。

接著我們會希望這個 x\head 跟(x - x\bar)他的距離越近越好，也就是要 minimize 這個 reconstruction error。

由於 W 已經找出來了，所以接下來只需要找的 c_k 的值。由於這些 K 個 vector w^K 是 orthonormal 的，因此只要把(x-x\bar)跟 w^k 做內積，就可找出 c_k。

而該內積的過程可以想成用 neural network 來表示，最終我們要使 NN 的 output 與(x - x\bar)的距離越近越好；

換句話說就是讓 input 等於 output，而這個東西就叫作 Autoencoder。



Word Embedding

一、1-of-N Encoding

將文字描述為 vector 的一種方式，這個 vector 的 dimension，就是這個世界上可能有的詞彙數目，而每一個文字，皆對應到其中一維。

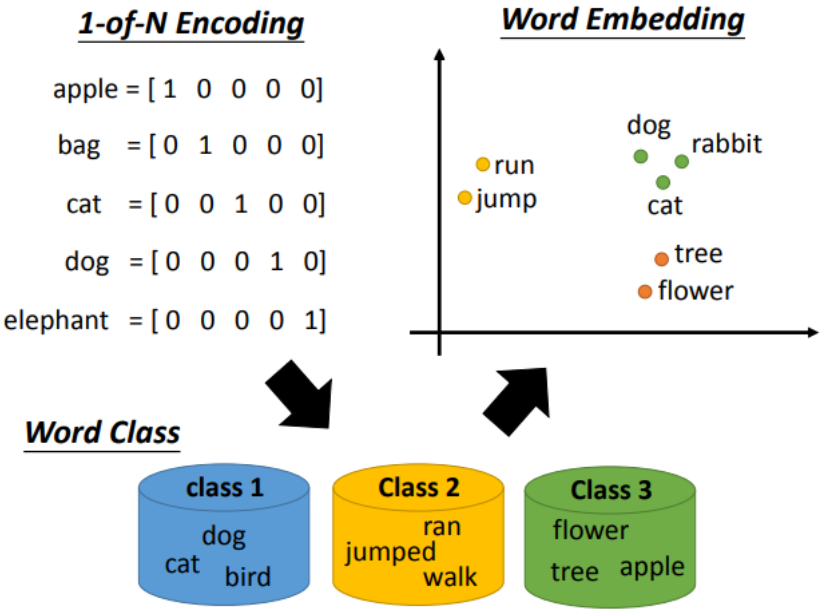
二、Word Class

然而上述方式無法從 vector 獲得任何資訊，因此可進一步使用 Word Class 將同性質的文字歸做一類。等同於在做 Dimension Reduction 的時候做 clustering 的概念。

三、Word Embedding

考量到光用一個 feature 無法將所有文字完全分開；或是 class 之間的相似程度無法區別，例如下圖 class 1 與 class 3 雖然分別是動物及植物，但是他們皆屬生物比起 class 2 的相近度較高。

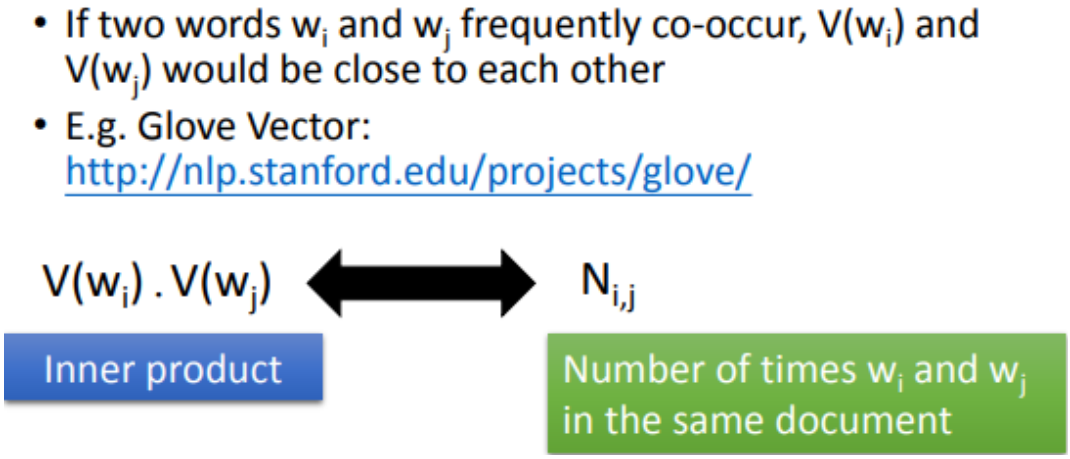
使用 Word Embedding 的時候，就是把每一個 word 都 project 到一個 high dimensional 的 space 上面，而 project 後的 vector 又稱 feature vector。



1. Count Based

如果現在有兩個詞彙，w_i 與 w_j 常常在同一個文章中出現，那他們的 word vector 就分別用 $V(w_i)$以及 $V(w_j)$來代表。而這種方法有一個很代表性的例子，叫做 Glove vector。

這個方法的原則是計算 $V(w_i)$以及 $V(w_j)$的內積；而假設 N_{ij} 是 w_i 跟 w_j 他們 co-occur 在同樣的 document 裡面的次數。最後我們希望找一組 w_i 與 w_j 的 vector 使其內積與 N_{ij} 越近越好。

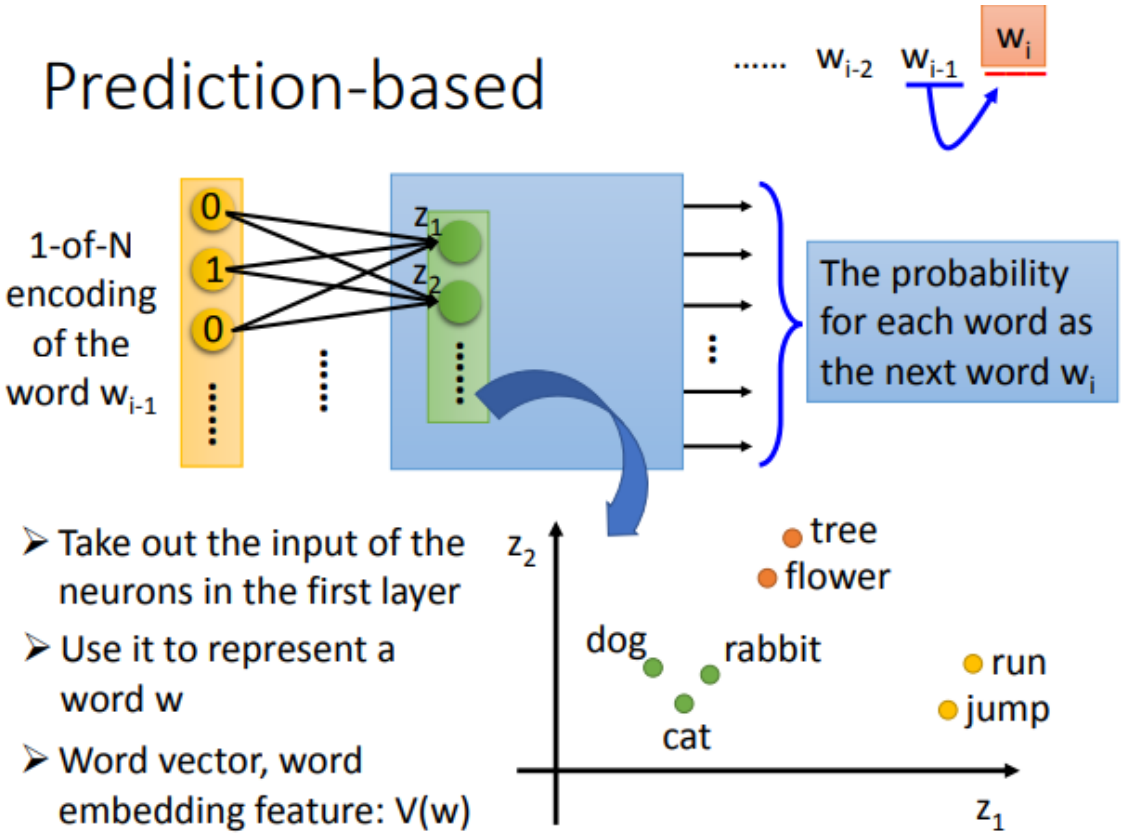


2. Perdition based

Prediction based 的方法主要是 learn 一個 neural network，他做的事情是 given 前一個 word 然後 predict 下一個可能出現的 word 是什麼。

假設給一個 sentence，這邊的每一個 w 代表一個 word；而這個 neural network 的 input w_{i-1} 就是 1-of-N encoding 的 vector；output 就是下一個 word w_i 是某一個 word 的機率，也就是說 output 的 dimension 就是 vector 的 size，假設現在世界上有 10 萬個 word，這個 model 的 output 就是 10 萬維。

至於將 input feature vector 丟進去 NN 的時候，他會通過很多 hidden layer。接下來把第一個 hidden layer 的 input 拿出來，寫作他的第一個 dimension 是 Z_1，第二個 dimension 是 Z_2，以此類推。這一個 input 1-of-N encoding 得到 Z 的這個 vector 就可代表一個 word 的 embedding。



(1) Sharing Parameters

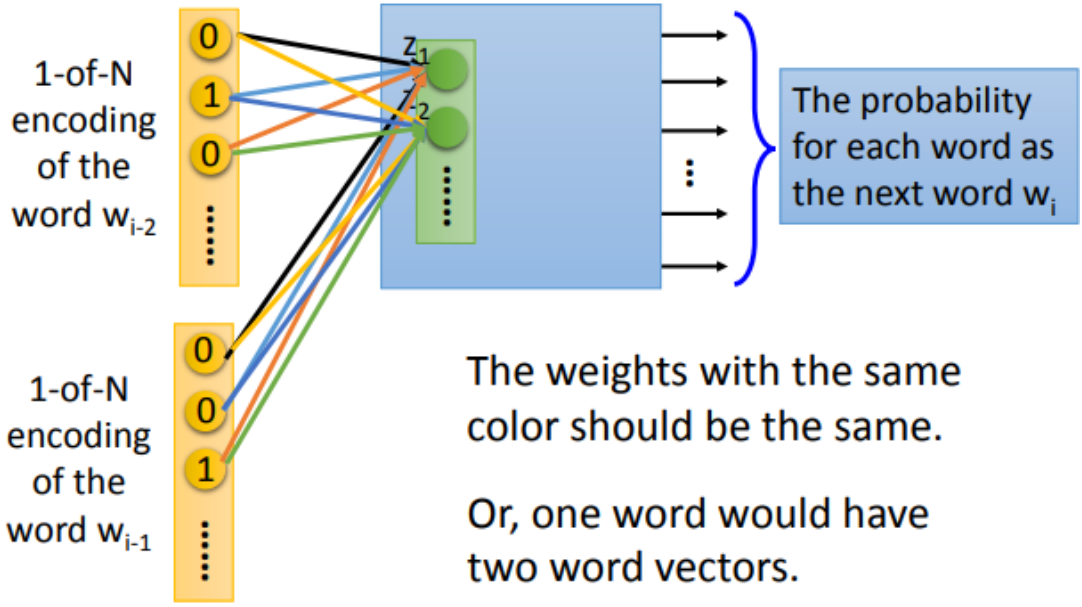
如果只看一個詞彙，他下一個連接的詞彙有非常多種組合可能性。因此可以拓展這個問題，希望 machine learn 的是 input 前面兩個詞彙 w_{i-2} 跟 w_{i-1} 並且 predict 下一個 word w_i。

因此可以輕易地把這個 model 拓展到 N 個詞彙。一般而言，如果要 learn 這樣的 word vector 的話至少需要 10 個詞彙才能夠 learn 出比較 reasonable 的結果。

這邊用 input 兩個 word 當作例子，值得注意的是一般的 neural network，就把 input w_{i-2} 跟 w_{i-1} 的 1-of-N encoding 的 vector 接在一起變成一個很長的 vector。接著直接丟到 neural network 裡面當作 input 就可以了。

但實際上會希望 w_{i-2} 相連的 weight 跟 w_{i-1} 相連的 weight 是被 tight 在一起的，意思就是 w_{i-2} 的第一個 dimension 跟第一個 hidden layer 的第一個 neuron 中間連的 weight；以及 w_{i-1} 的第一個 dimension 跟第一個 hidden layer 的第一個 neuron，他們之間連的 weight。這兩個 weight 必須是一樣的，以此類推。

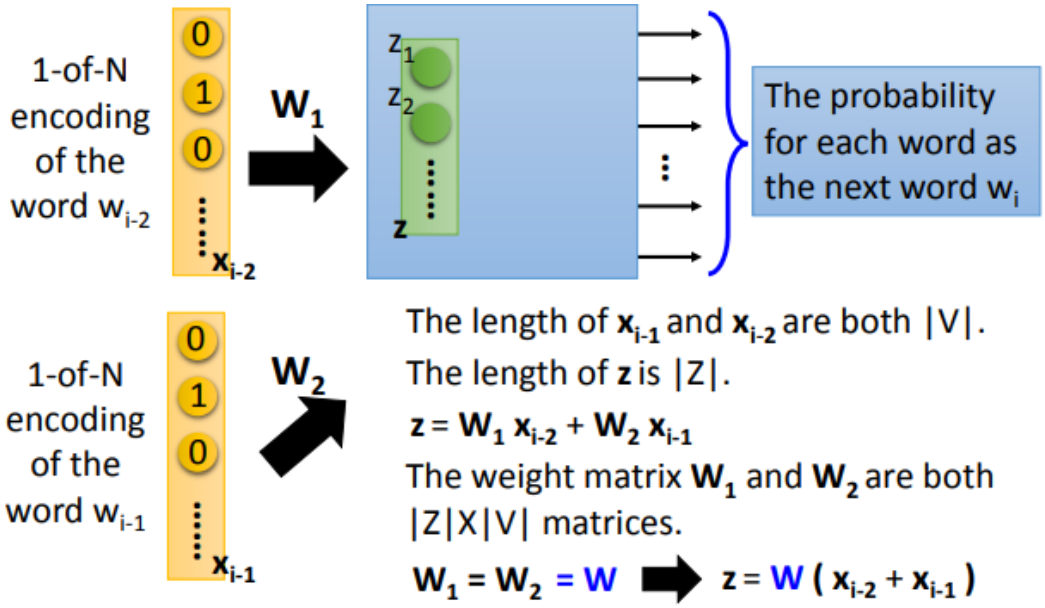
如果不這麼做的話，把同一個 word 放在 w_{i-2} 的位置跟放在 w_{i-1} 的位置，通過這個 transform 以後得到的 embedding 就會不一樣；另外一個理由在於可以減少參數量，因為 input 這個 dimension 很大，所以就算這個 feature vector 是 50 維，也是一個非常大的 matrix。如果強迫讓所有的 1-of-N encoding，他後面接的 weight 是一樣的，那就不會隨著 contest 的增長，而需要這個更多的參數。



進一步用 formulation 來表示該想法。假設 w_{i-2} 的 1-of-N encoding 就是 X_{i-2}；w_{i-1} 的 1-of-N encoding 就是 X_{i-1}，那他們的長度都是 V 的絕對值。而這個 hidden layer 的 input 寫做一個 vector Z，長度為 Z 的絕對值。

而 Z 等於 $X_{i-2} * W_1 + X_{i-1} * W_2$。現在這個 W_1 跟 W_2 都是一個 Z 乘上一個 V dimension 的 weight matrix。接著我們強制讓 W_1 跟 W_2 相等，等於一個一模一樣的 matrix W。

也就是說在處理這個問題的時候，可以把 X_{i-2} 跟 X_{i-1} 直接先加起來，再乘上 W 的這個 transform 就會得到 z。



事實上在 train CNN 的時候也有讓某一些參數必須是相同的需求。因此採用相同作法，假設我們希望 w_i 跟 w_j 他們的 weight 是一樣的，因此要給他們一樣的 initialization。

接下來計算 w_i 對 cost function 的偏微分，然後 update w_i；同理計算 w_j 對 cost function 的偏微分，然後 update w_j。

$$w_i \leftarrow w_i - \eta \frac{\partial C}{\partial w_i}$$

$$w_j \leftarrow w_j - \eta \frac{\partial C}{\partial w_j}$$

然而如果他們對 C 的偏微分是不一樣的，那就必須把 w_i 進一步減掉 w_j 對 C 的偏微分；同時把 w_j 減掉 w_i 對 C 的偏微分。也就是確保 w_i 跟 w_j 在訓練的過程中，他們的 weight 永遠都是被 tight 在一起的。

$$w_i \leftarrow w_i - \eta \frac{\partial C}{\partial w_i} - \eta \frac{\partial C}{\partial w_j}$$

$$w_j \leftarrow w_j - \eta \frac{\partial C}{\partial w_j} - \eta \frac{\partial C}{\partial w_i}$$

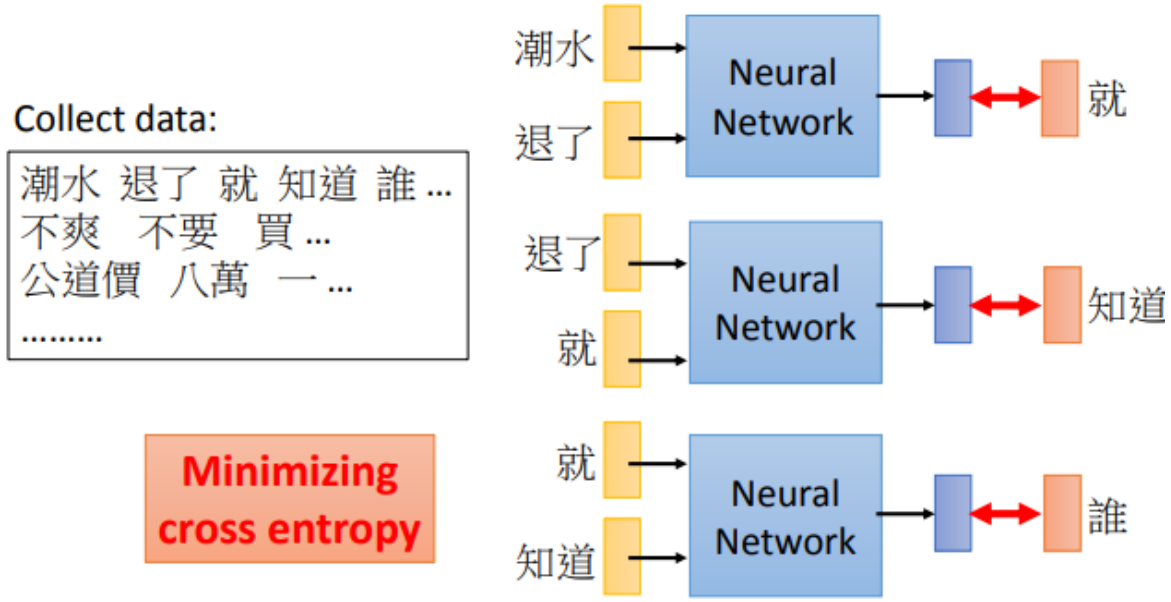
(2) Training（以下截自影片字幕檔）

那要怎麼訓練這個 network 呢？這個 network 的訓練，完全是 unsupervised 的。也就是說，你只要 collect 一大堆文字的 data，然後接下來就可以 train 你的 model。

比如說這邊有一個句子就是：潮水退了，就知道誰沒穿褲子。
那就讓你的 neural network input "潮水" 跟 "退了"，希望他的 output 是 "就" 。

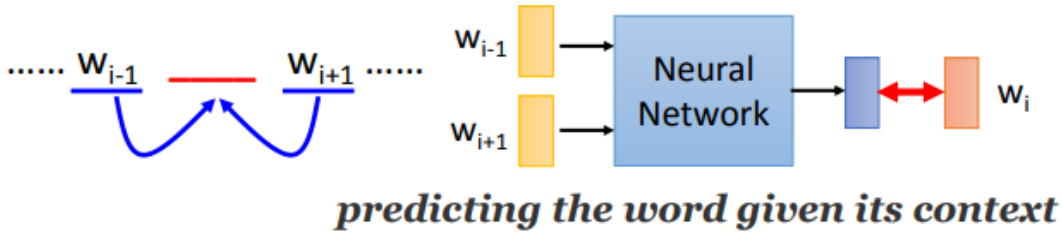
所以你希望 network 的 output 跟 "就" 的 1-of-N encoding，是 minimize cross entropy。然後再來就 input "退了 " 跟 "就"，希望他的 output 跟 "知道" 越接近越好。

最後 output "就" 跟 "知道"，希望他跟 "誰" 越接近越好。



(3) Continuous bag of word (CBOW)

拿某一個詞彙的 context 去 predict 中間這個詞彙，也就是拿 W_{i-1} 跟 W_{i+1} 去 predict W_i。



predicting the word given its context

(4) Skip-gram

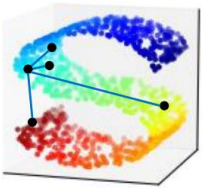
拿中間的詞彙去 predict 接下來的 context，也就是拿 W_i 去 predict W_{i-1} 跟 W_{i+1}。

predicting the context given a word

此外，上述 Perdition based 的各種變形，皆不需要 deep 就能做得起來，因此可以大幅減少運算量。

Neighbor Embedding

與前述的 Dimension Reduction 不同，雖然皆是處理降維的問題，但 Neighbor Embedding 可以處理 Manifold 形式的 data point。由於在該例中兩點距離很遠的時候 Euclidean distance 不一定會成立，因此 PCA 無法有效分析兩點的距離。



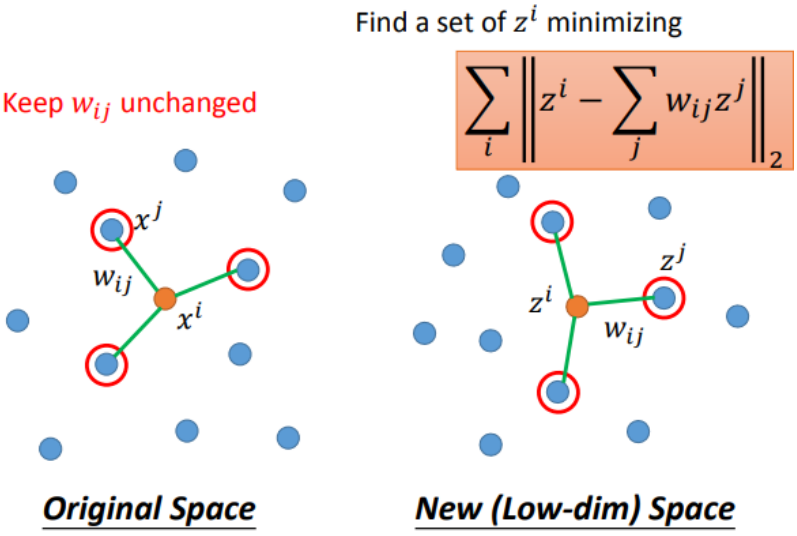
一、Locally Linear Embedding (LLE)

在原來的空間裡面有某一個點，x^i，接著選出這個 x^i 的 neighbor x^j。
接下來要找 x^i 跟 x^j 的關係，寫作 w_{ij}。

假設每一個 x^i，都可以用他的 neighbor 做 linear combination，而 w_{ij} 就是拿 x^j 去組合 x^i 的 weight。

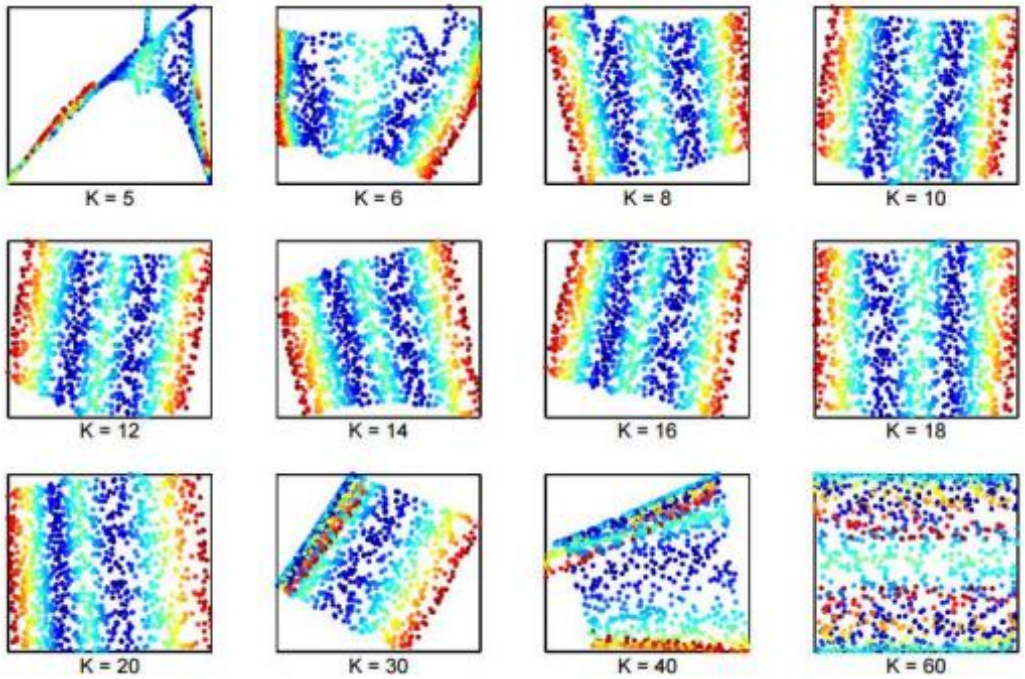
接著我們希望這組 w_{ij} 對 x^i 的所有 neighbor x^j 做 weighted sum 的時候，他可以跟 x^i 越接近越好。也就是 x^i 減掉 summation over 所有的 w_{ij} 乘以 x^j，他的 two norm 是越小越好的。

然後做 dimension reduction 把原來所有的 x^i 跟 x^j 轉成 z^i 和 z^j，然而他們中間的關係 w_{ij}，是不變的。



原來這些 x^j 可以做 linear combination 產生 x^i，而這些 z^j 也可以用同樣的 linear combination 產生 z^i。所以現在在這個式子裡面 w_{ij} 變成是已知的，接著要找一組 z，讓 z^j 透過 w_{ij} 做 weighted sum 以後，他可以跟 z^i 越接近越好。

此外該方法中，neighbor 的數量 K 選太小的時候，無法考慮距離較遠的點的情況；而 K 太大時，則會納入一些 transform 後關係太弱的 neighbor。



二、Laplacian Eigenmaps

然而比較兩點之間的距離，只算 Euclidean distance 是不足夠的，要看的是他們在這個 high density 的 region 之間的 distance。如果兩個點之間有 high density 的 connection，才算是真正的接近。

這件事情可以用一個 graph 來描述這件事情，也就是把 data point construct 成一個 graph，計算 data point 兩兩之間的相似度，如果相似度超過 threshold，就把他們 connect 起來。

此處可考慮 smoothness 的距離來建立 graph。如果 x^1 跟 x^2 在 high density 的 region 是 close 的，那我們就會希望，z^1 跟 z^2 也是相近的。

上述描述可用 smoothness 的式子寫出來，解法就同 semi-supervised learning：

$$S = \frac{1}{2}\sum_{i,j} w_{i,j}\sqrt{(z^i)^2 + (z^j)^2} \quad (2-norm)$$

而 $w_{i,j}$ 代表如果今天兩個 data point 在圖上是相連的，那 $w_{i,j}$ 就是他們的相似程度；若不相連，即為 0。最後找出 z^i 跟 z^j minimize S。

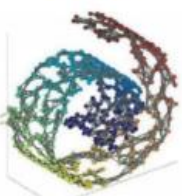
(Review: semi-supervised learning)

$$w_{i,j} = \begin{cases} \text{similarity} & \text{If connected} \\ 0 & \text{otherwise} \end{cases}$$

- *Review in semi-supervised learning*: If x^1 and x^2 are close in a high density region, $\hat{y}^1$ and $\hat{y}^2$ are probably the same.

$$L = \sum_{x^r} C(y^r, \hat{y}^r) + \lambda S$$

As a regularization term



$$S = \frac{1}{2}\sum_{i,j} w_{i,j}\left(y^i - y^j\right)^2 = \boldsymbol{y}^T L \boldsymbol{y}$$

S evaluates how smooth your label is

L: (R+U) x (R+U) matrix

Graph Laplacian

$$L = D - W$$

然而為防止 z^i 跟 z^j 為相同的值，必須再加上 constrain。如果 z 降維以後的空間是 M 維的空間則希望 z^1 到 z^N 做 span 以後會等於 R^M。也就是說 z 會佔據整個 M 維的空間。

而 z 與前述的 graph Laplacian L 是有關係的，他其實就是 graph Laplacian 的對應到比較小的 eigenvalue 的那些 eigenvector。

- *Dimension Reduction*: If x^1 and x^2 are close in a high density region, z^1 and z^2 are close to each other.

$$S = \frac{1}{2}\sum_{i,j} w_{i,j}\left(z^i - z^j\right)^2$$

Any problem? How about $z^i = z^j = \mathbf{0}$?

Giving some constraints to z:

If the dim of z is M, Span{z^1, z^2, ... z^N} = R^M

Spectral clustering: clustering on z

Belkin, M., Niyogi, P. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems* . 2002

至於如果先找出 z 之後，再用 K-means 做 clustering 稱 spectral clustering。

三、T-distributed Stochastic Neighbor Embedding (t-SNE)

前述的方法有一個最大的問題就是，他只假設相近的點應該要是接近的，但沒有假設不相近的點要盡量分開。

比如用 LLE 在 MNIST 上時，他確實會把同個 class 的點都聚集在一起，但沒有防止不同 class 的點不要疊成一團。

而做 t-SNE 時一樣是做降維，把原來的 data point x 變成比較低維的 vector z。

那在原來的 x 的 space 上計算所有的點的 pair，x^i 和 x^j 之間的 similarity，寫作 $S(x^i, x^j)$。

接下來做 normalization，計算 $P(x^j \mid x^i)$。在分子的地方是 x^i 跟 x^j 的 similarity，然後分母的地方就是 summation over 除了 x^i 以外，所有其他的點和 x^i 之間所算出來的距離。

$$P(x^j|x^i) = \frac{S(x^i, x^j)}{\sum_{k \neq i} S(x^i, x^k)}$$

另外假設已經找出了一個 low dimension 的 representation z^i 跟 z^j，同樣也可以計算 similarity S'。

同理可定義 $Q(z^j \mid z^i)$，他的分子的地方就是 $S'(z^i, z^j)$，分母的地方就是 summation over z^i 跟所有 database 裡面的 data point z^k 之間的距離。

此處做 normalization 是必要的，因為不知道在高維空間中算出來的距離 $S(x^i, x^j)$ 跟 $S'(z^i, z^j)$，他們的 scale 是不是一樣的。
如果有做 normalization，就可以把他們都變成機率，此時他們的值都會介於 0 到 1 之間，他們 scale 會是一樣的。

x ⟶ z

Compute similarity between all pairs of x: $S(x^i, x^j)$

$$P(x^j|x^i) = \frac{S(x^i, x^j)}{\sum_{k \neq i} S(x^i, x^k)}$$

Compute similarity between all pairs of z: $S'(z^i, z^j)$

$$Q(z^j|z^i) = \frac{S'(z^i, z^j)}{\sum_{k \neq i} S'(z^i, z^k)}$$

接下來我們希望找一組 z^i 跟 z^j，讓這兩個 distribution P 與 Q 越接近越好。
此處便是用 KL divergence 衡量兩個 distribution 之間的相似度，也就是使這兩個 distribution 之間的 KL divergence 越小越好，最後 summation over 所有的 data point minimize L (gradient descent)。

Find a set of z making the two distributions as close as possible

$$L = \sum_i KL\left(P(*|x^i)||Q(*|z^i)\right)$$

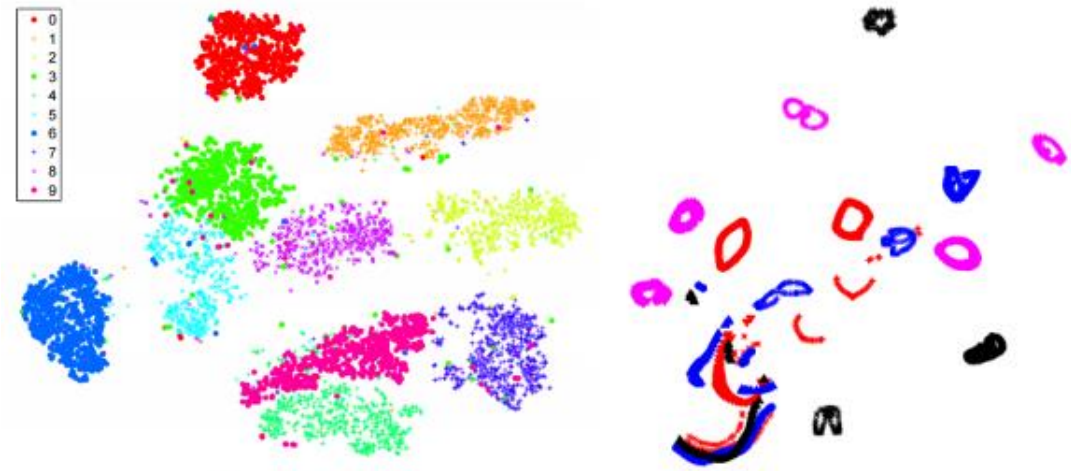
$$= \sum_i \sum_j P(x^j|x^i) log \frac{P(x^j|x^i)}{Q(z^j|z^i)}$$

此外在做 t-SNE 的時候，他會計算所有 data point 之間的 similarity，所以其運算量有點大。
因此常見的做法是先用比較快的方法做降維 (PCA)，最後再使用 t-SNE。

然而如果給 t-SNE 一個新的 data point 他會無法處理，他只能夠先給他一大堆 x，再把每一個 x 的 z 都找出來，接著繼續給他一個新的 x。
因此會需要重新跑一遍這一整套演算法，過程會變得相當麻煩。
所以一般 t-SNE 的作用比較不是用在這種 training testing 的這種 base 上面，而是拿來做 visualization。

也就是說如果已經有一大堆的 x，他是 high dimensional 的，那想要 visualize 他們在二維空間的分佈上是什麼樣子，就可以使用 t-SNE。

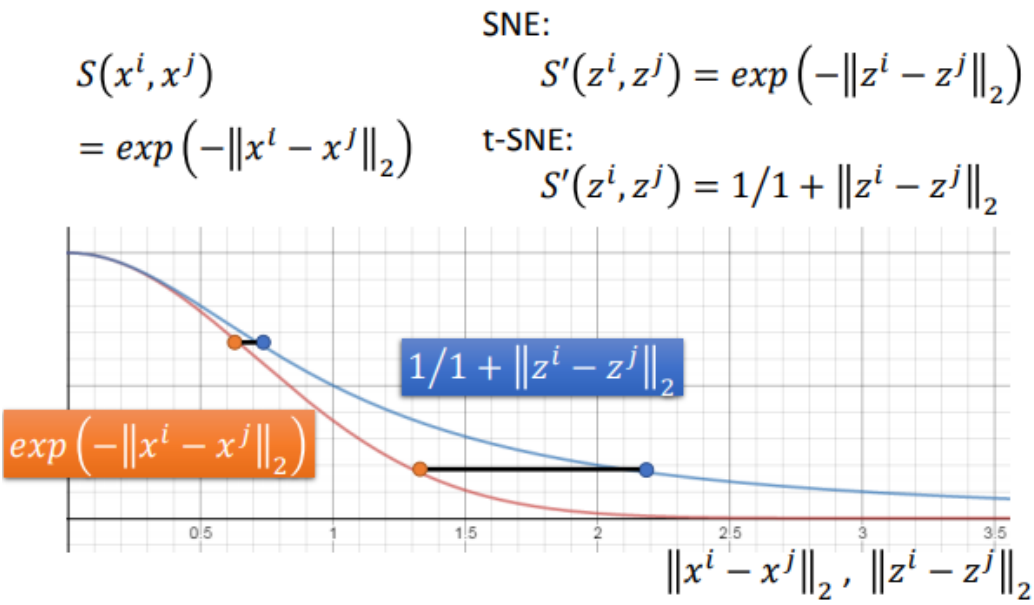
- Good at visualization



t-SNE on MNIST　　t-SNE on COIL-20

1. t-SNE–Similarity Measure

下圖橫軸代表了在原來 space 上的 Euclidean distance 或是做 dimension reduction 以後的 Euclidean distance；而紅線是 RBF function，藍線是 t-distribution。

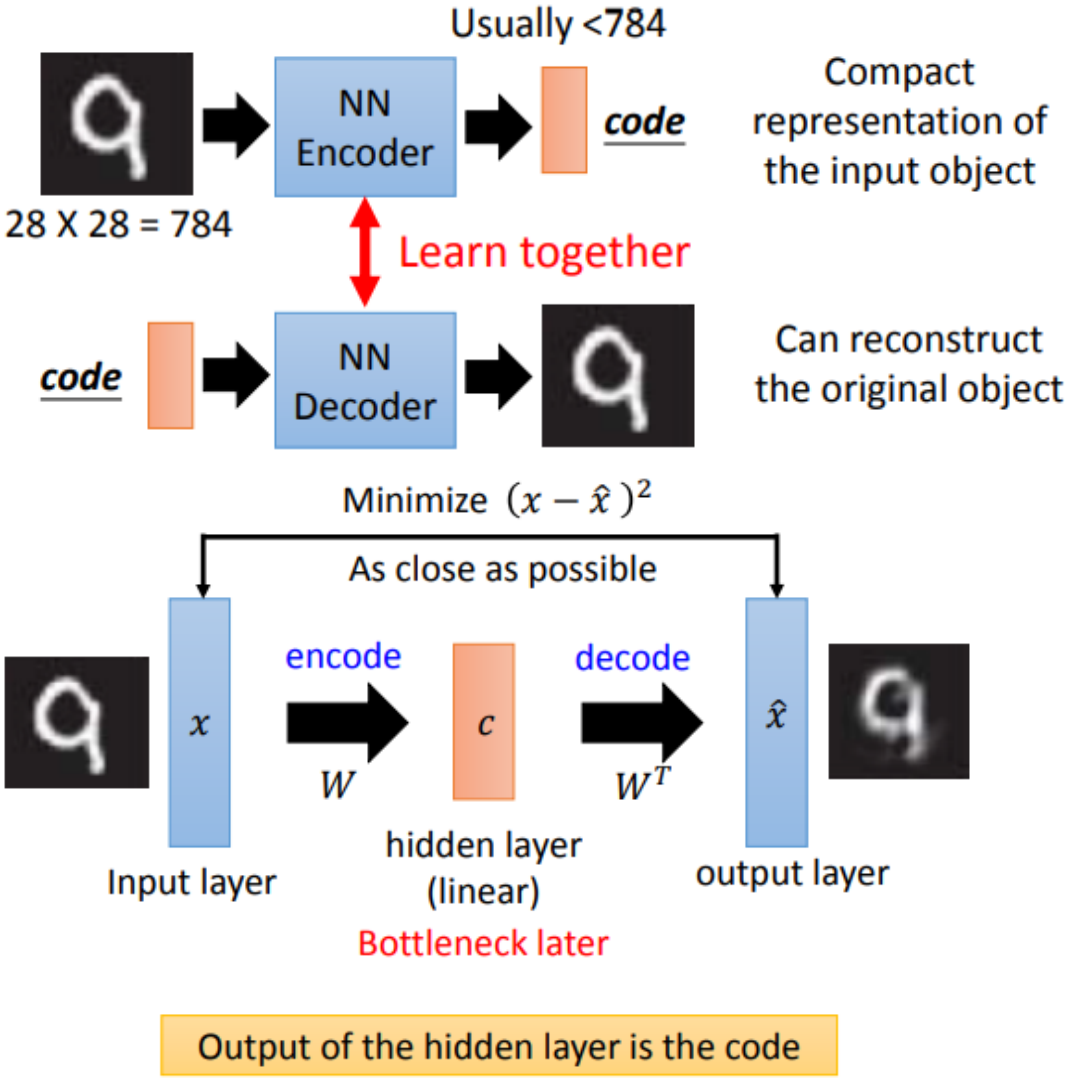


原來橘點做 dimension reduction 以後，為維持他們原來之間的距離，會轉換為藍點。此時就可得知，變到 t-distribution 以後，原來在高維空間裡面如果距離很近，做完 transform 以後他還是很近；如果原來就已經有一段距離，那做完 transform 以後他就會被拉得很遠。

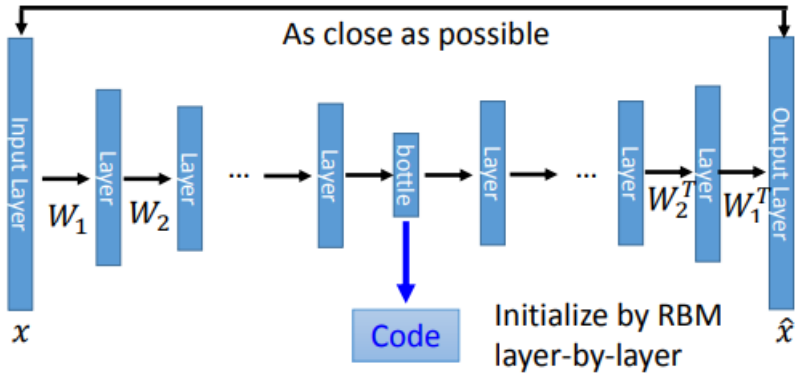
Auto-encoder

一、Deep Auto-encoder

假設該做的是 MNIST 的話，input 就是一張 digit，他是 784 維的 vector；output 就是一個遠比 784 維還要小的 code。
也就是我們需要 learn 一個 encoder，讓 input 通過後變成 code。為了做到這件事必須同時 learn 一個 decoder，使 code 還原回原本的圖片。
接著就是將他們接起來變成一個 neural network，並希望他的 input 與 output 距離越近越好。



當成 neural network 來看的話，input 的 x 就是 input layer，output 的 x\head 就是 output layer，而中間 component 的 weight 就是 hidden layer，又稱 bottleneck layer，他是個特別窄的 layer，代表的是一組 code。
從 input layer 到 bottleneck layer 就是 encoder；而從 bottleneck layer 的 output 到 output layer 就是 decoder。
而 training 的方法與 neural network 的方法相同，就是 back propagation。



Reference: Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507

另外他還可以 learn 到把雜訊濾掉這件事，就是 de-noising auto-encoder，也就是加了 noise 以後，還要 reconstruct 回原來沒有 noise 的結果。

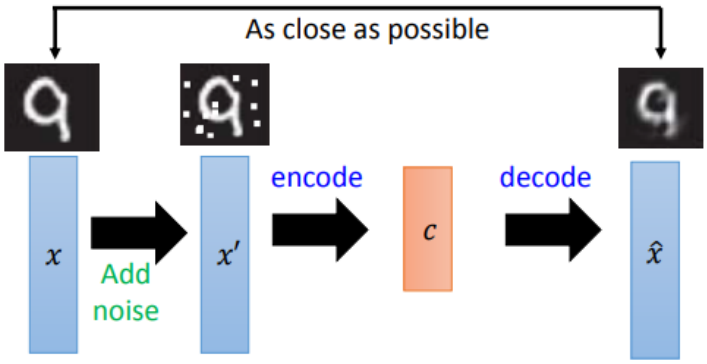
與他類似的方法叫做 constrictive auto-encoder，我們會希望說，在 learn 這個 code 的時候，我們加上一個 constrain，表示當 input 有變化的時候對這個 code 的影響是被 minimize 的。也就是說希望說當 input 變了（加了 noise）以後，對這個 code 的影響是小的

More: Contractive auto-encoder

Auto-encoder

Ref: Rifai, Salah, et al. "Contractive auto-encoders: Explicit invariance during feature extraction." *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011.

• De-noising auto-encoder



Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." *ICML*, 2008.

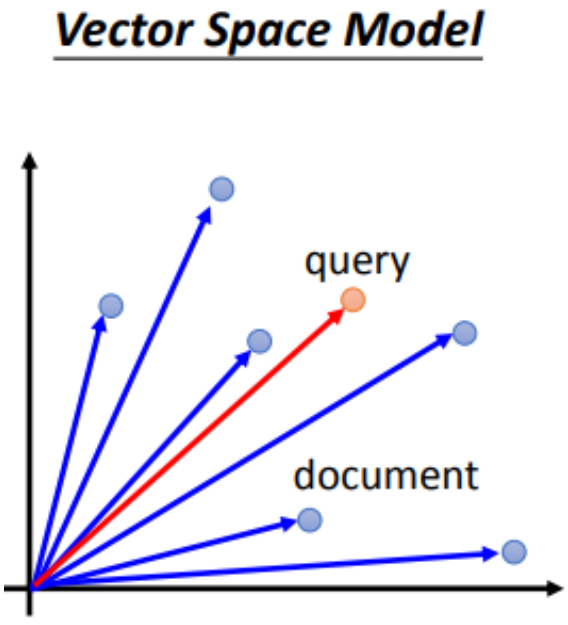
二、Applications

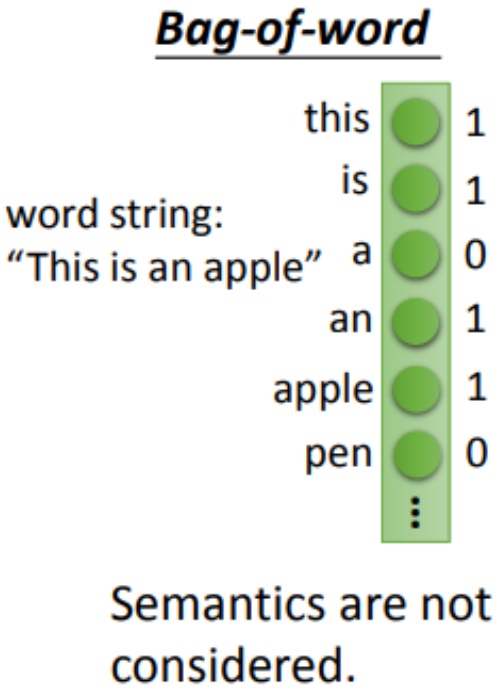
1. Encoder for Text Retrieval

auto-encoder 也可以把他用在文字的搜尋處理上。
常見的方法為 vector space model。也就是把每一篇文章都表示成空間中的一個 vector，然後把查詢的詞彙也變成空間中的一個點。

接下來就是計算這個輸入的查詢詞彙，跟每一篇 document 之間的 inner product 或是 cosine similarity 等等，後者會有 normalize 的效果，可能會得到比較好的結果。
這個模型的表現好壞取決於把一個 document 變成 vector 表示的好還是不好。

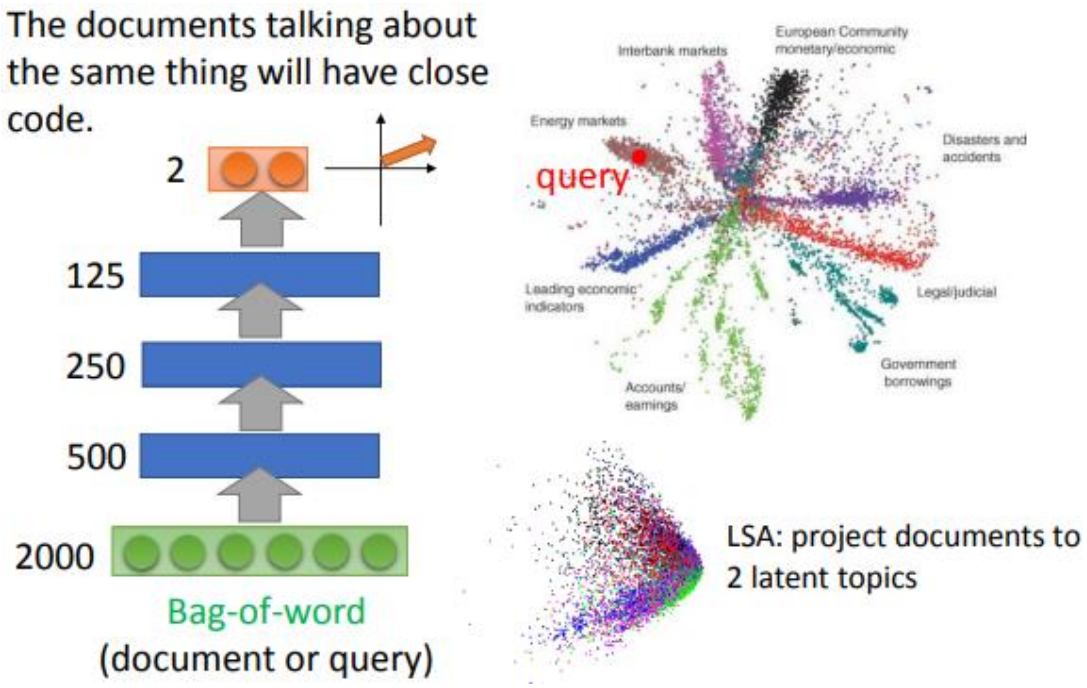
最常用的方法叫做 bag-of-word。如果想把他做得更好，可以乘上 inverse document frequency，也就是每一維不只會用詞彙在 document 出現的次數，還會再乘上一個 weight，代表那個詞彙的重要性。
（在每個 document 出現次數越高，重要性越低，如 " is " ）





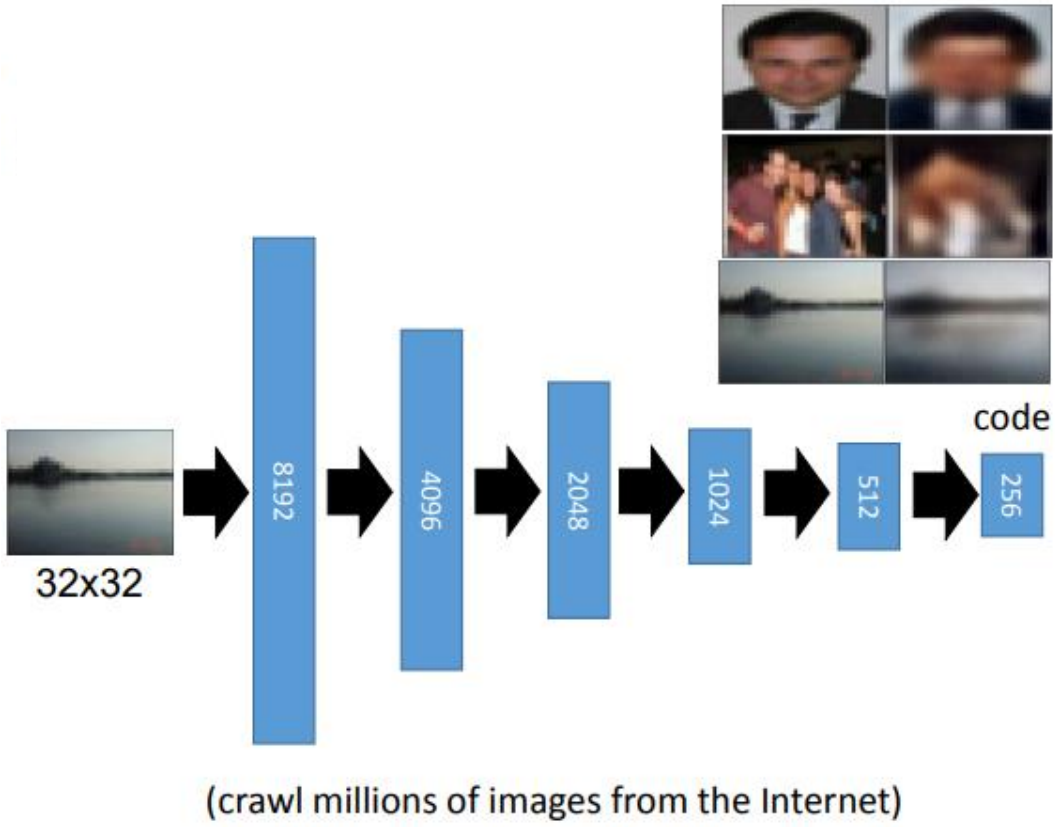
但是用這個模型沒辦法考慮任何語意相關的東西，因此可以用 auto-encoder 讓語意這件事情被考慮進來。

舉例來說，learn 一個 auto-encoder，他的 input 就是一個 document，透過 encoder 壓成二維的 vector 後可以 visualize 為右圖。
接著如果要查詢詞彙的話，也同樣通過這個 encoder，把他變成一個二維的 vector。最後看他落在哪個區域，就可以得知與何種類別有關。



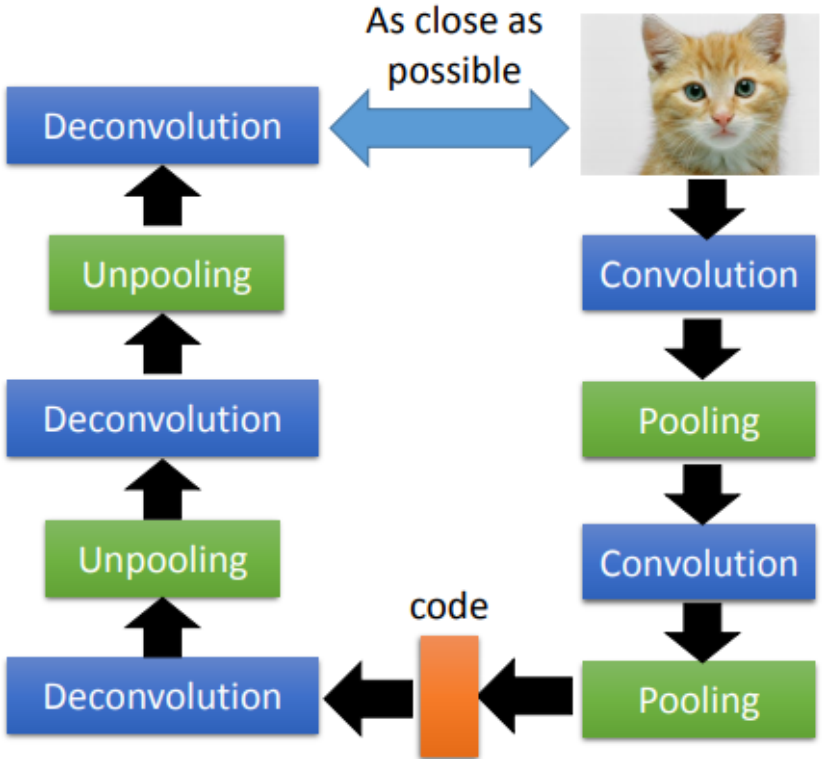
2. Encoder for Similar Image Search

如果只在 pixel wise 上做比較圖片的相似度的話，通常無法找到好的結果。
此時就可以用 deep auto-encoder 把每一張 image 變成一個 code，然後在 code 上面再去做搜尋。



3. Auto-encoder for CNN

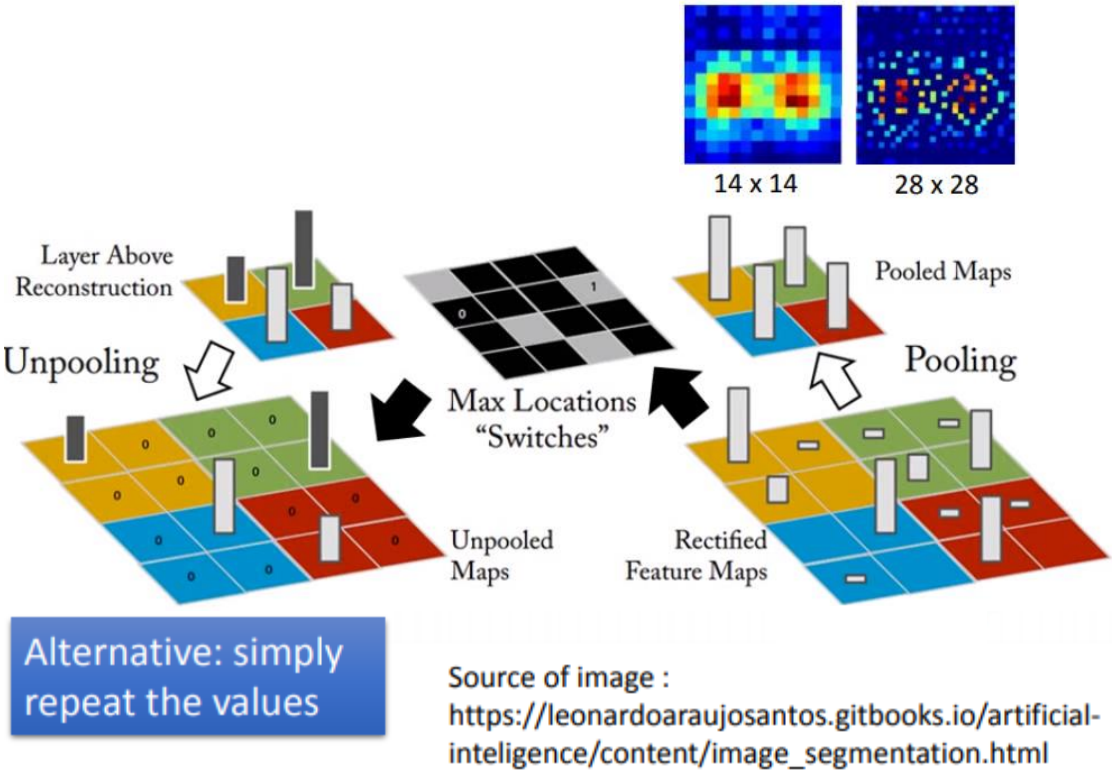
那如果是做 Auto-encoder 改善 CNN 的表現的話，不只要有個 encoder 還要有個 decoder。前者的部分就是做 convolution 再做 pooling，而後者理論上應該就是做跟 encode 相反的事情，也就是 unpooling 與 deconvolution。
最後 training 的 criteria 一樣就是讓 input 和 output 越接近越好。



(1) Unpooling

假設圖片在做 pooling 的時候，是把一個 4*4 的 matrix 裡面的 pixel 分成四個一組，接下來從每一組裡面挑一個最大的，使 image 變成原來的四分之一。

做 unpooling 的時候，必須記得做 pooling 的時候是從哪裡取值，然後把原來比較小的 matrix 擴大，也就是將剩餘的部分補 0 或是複製剛剛取 pooling 的值。

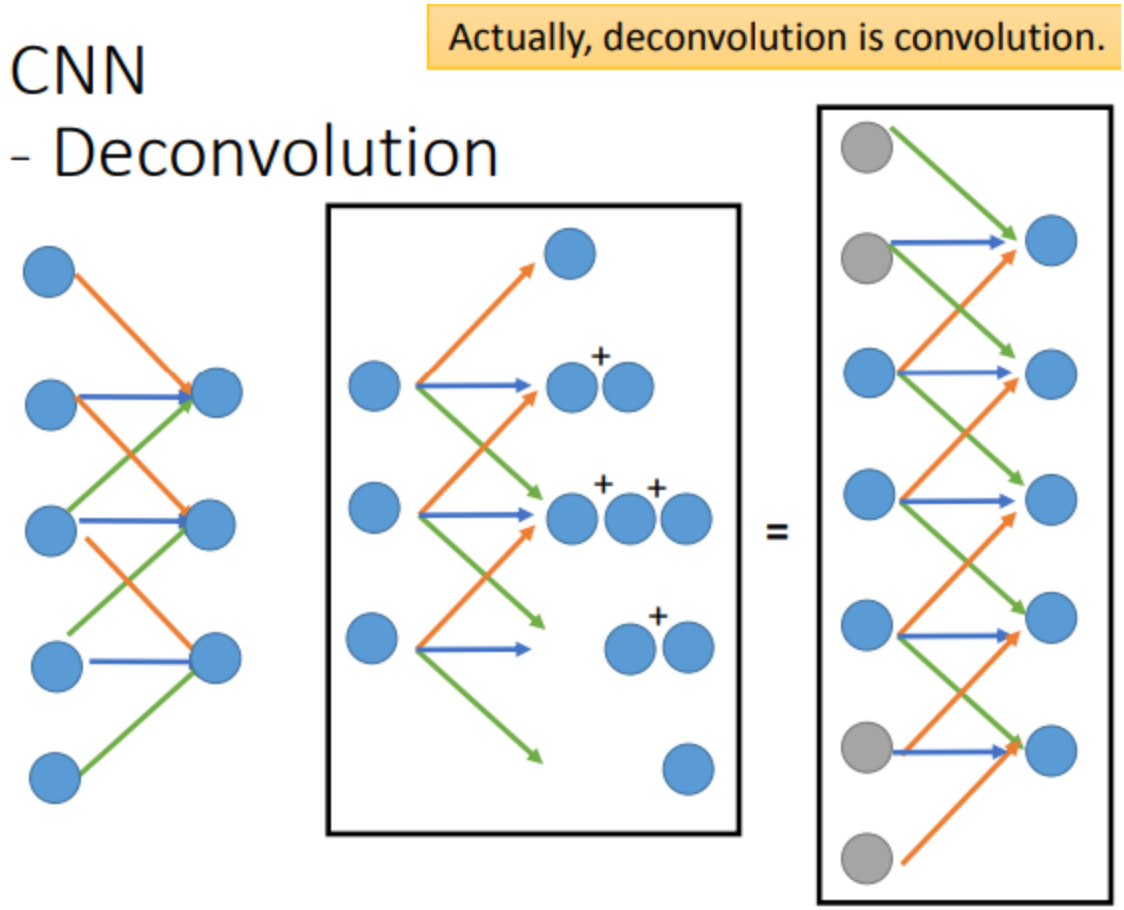


(2) Deconvolution

事實上 deconvolution 就是 convolution。
假設 input 有 5 個 dimension 且 filter size 為 3，然後就把 input 的這 3 個 value 分別乘上紅色藍色綠色的 weight，得到一個 output。
再把這個 filter shift 一格，3 個 value 分別乘上 weight 得到下一個 output。

而 deconvolution 可想成把一個值變成三個值，也就是一個值分別乘上紅色綠色藍色的 weight 變成三個值
shift 一格後也乘上 weight 變成三個值再跟前者做疊加。

這件事等同於把 input 做 padding，在旁邊補零，接下來一樣做 convolution。
三個 input 乘上綠色藍色紅色的 weight 得到一個值，再 shift 一格，以此類推。
也就是框框裡面做的事情是一模一樣的，不同點是在他們的 weight 是相反。



4. Auto-encoder for Pre-training DNN

在 train 一個 neural network 的時候，可以先用 auto-encoder 找到一組比較好的 initialization，該流程就叫做 pre-training。

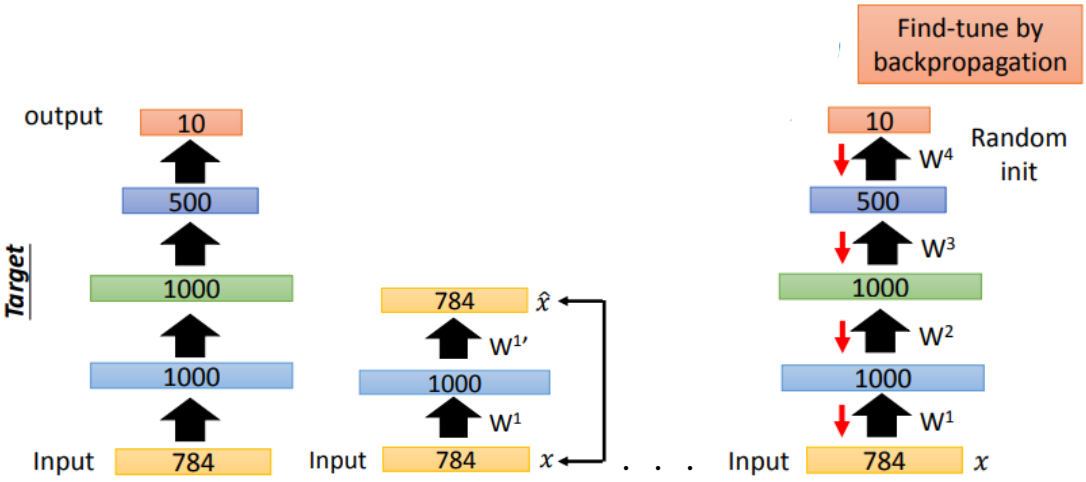
假設為 MNIST 的 recognition，network 的 input 是 784 維，第一個 hidden layer 是 1000 維，第二個 hidden layer 是 1000 維，然後 500 到 10 維。

在做 pre-train 的時候就是先 train 一個 auto-encoder，input 是 784 維，然後中間有個 1000 維的 vector，再把他變回 784 維，希望 input 跟 output 越接近越好。如此就可得到參數 W^1。

接著重複上述步驟得到 W^2 與 W^3，再給 W^4 一個 random 的初始值，再用 back propagation 去調一遍找參數 W，我們稱之為 fine tune。

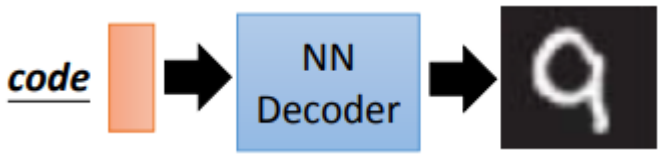
然而當 code 比 dimension 還要大時，auto-encoder 可能會直接把 input 硬背起來再輸出，因此要在這 1000 維的 output 加一個很強的 regularization (L1)，也就是希望這 1000 維的 output 是 sparse，裡面可能只有某幾維是可以有值的，其他維都必須要是零。

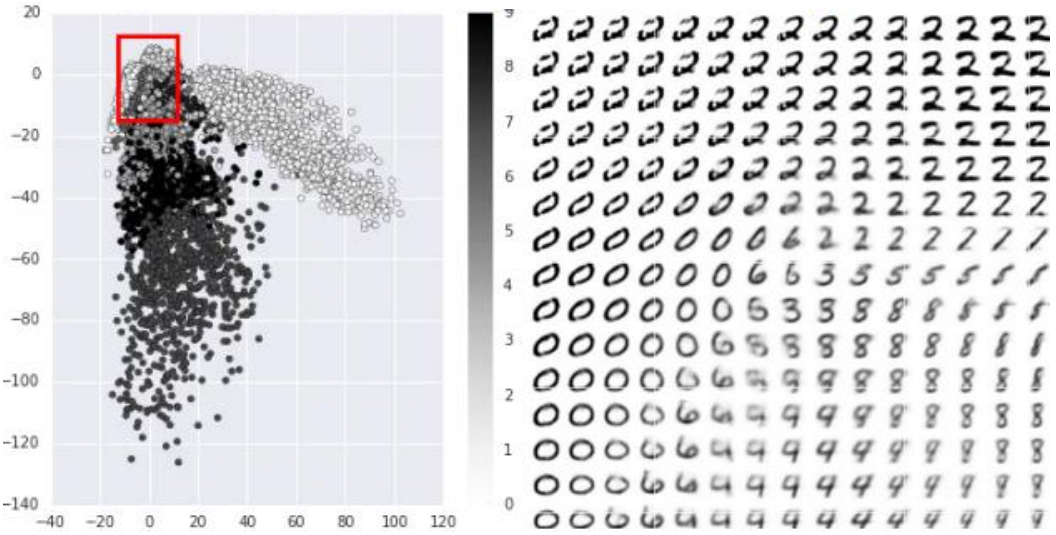
pre-training 的好處在於，如果有很多 unlabeled data 與少量的 labeled data，可以用大量的 unlabeled data 去把 W^1，W^2，W^3 先 initialize 好，那最後的 labeled data 就只需要稍微調整 weight 就好。



5. Decoder for Drawing

理論上可以給 decoder input random number，而 output 希望就是一張圖。
以 MNIST 為例，把 code 取出來後可以 visualize 為左邊這張圖。接著隨機 sample vector 丟進 decoder，產生的結果就為右圖。





然而 sample 在紅框其他地方，得到東西看起來就不像是 image 了，因此我們希望 region 裡面都是 image。
比較簡單的做法就是在 code 上面加上 L2 的 regularization，讓所有的 code 都比較接近零。
接下來就在零附近 sample，而 sample 出來的 vector 就都可以對應到數字。