Reducción de dimensionalidad

Pablo Huijse Heise

Reducción de dimensionalidad

- Sea una matriz de datos $X \in \mathbb{R}^{NxM}$
- Se denomina característica a cada una de las M columnas de X
- La dimensionalidad M puede ser muy grande

Reducción de dimensionalidad

- Sea una matriz de datos $X \in \mathbb{R}^{NxM}$
- Se denomina característica a cada una de las M columnas de X
- La dimensionalidad M puede ser muy grande

Maldición de la dimensionalidad

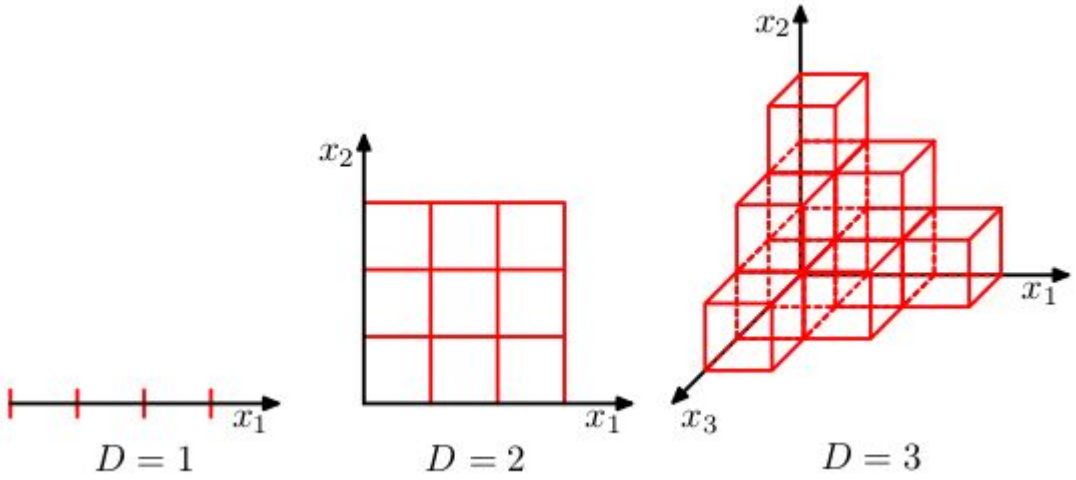
Mientras más dimensiones se tengan:

- Más datos de entrenamiento se necesitan para optimizar nuestros modelos
- Más costoso es el cómputo de distancias y métricas
- Más problemas asociados a generalización y sobre-ajuste

Reducción de dimensionalidad

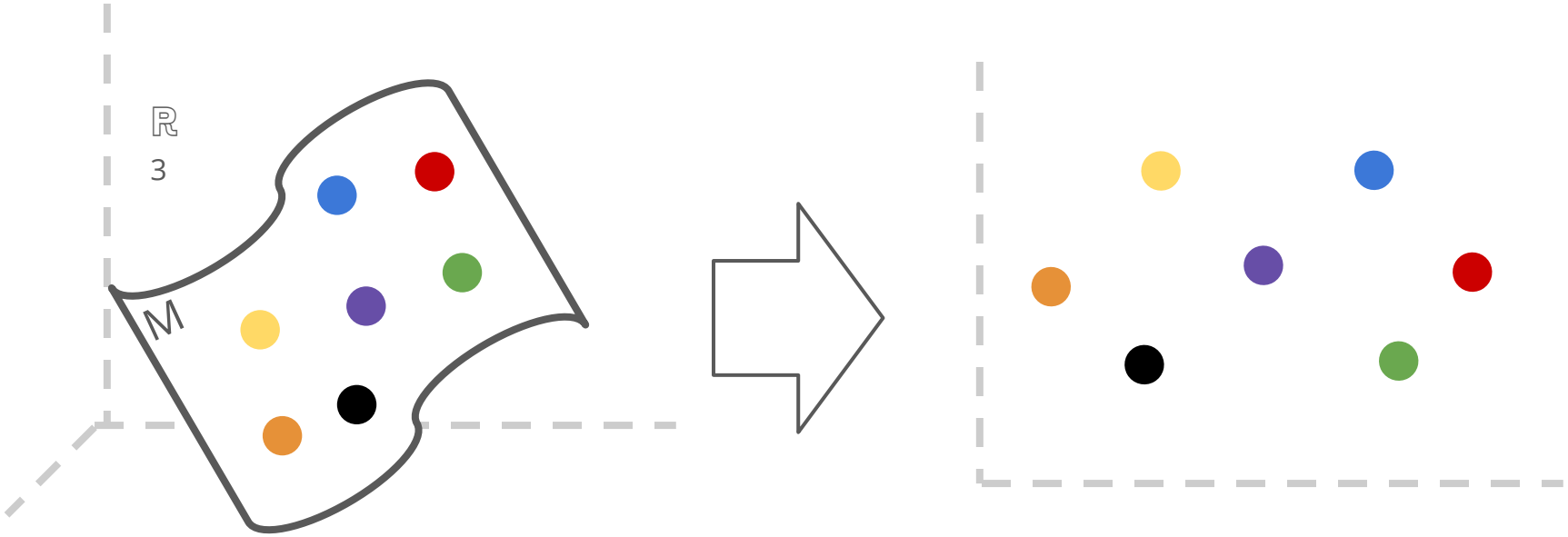
- Sea una matriz de datos $X \in \mathbb{R}^{NxM}$
- Se denomina característica a cada una de las M columnas de X
- La dimensionalidad M puede ser muy grande

Maldición de la dimensionalidad



Reducción de dimensionalidad

- Sea una matriz de datos $X \in \mathbb{R}^{NxM}$
- Se denomina característica a cada una de las M columnas de X
- La dimensionalidad M puede ser muy grande
- Suponemos que los datos habitan en un sub-espacio (manifold) de menor dimensión que el espacio de entrada



Reducción de dimensionalidad

- Sea una matriz de datos $X \in \mathbb{R}^{NxM}$
- Se denomina característica a cada una de las M columnas de X
- La dimensionalidad M puede ser muy grande
- Suponemos que los datos habitan en un sub-espacio (manifold) de menor dimensión que el espacio de entrada

Deseamos reducir la cantidad de características sin perder “información relevante“

Principal Component Analysis (PCA)

Sean

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad C = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$$

PCA busca un conjunto de vectores U normalizados que resulten en una proyección de los datos con varianza máxima

Intuición: Los ejes de máxima varianza son los de mayor interés

Principal Component Analysis (PCA)

Sean

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad C = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$$

PCA busca un conjunto de vectores U normalizados que resulten en una proyección de los datos con varianza máxima

$$\max_{U} \; U^T C U$$

$$\text{s.a.} \;\; U^T U = I$$

Principal Component Analysis (PCA)

PCA busca un conjunto de vectores U que resulten en una proyección de los datos con varianza máxima

Incorporamos la restricción usando multiplicadores de Lagrange

$$U^T C U + \Lambda(I - U^T U)$$

De la derivada en función de U se tiene que

$$CU = \Lambda U \qquad \Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_N \end{pmatrix}$$

Principal Component Analysis (PCA)

La solución

$$CU = \Lambda U \qquad \Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_N \end{pmatrix}$$

Es equivalente al problema de valores propios

Reconocemos entonces a

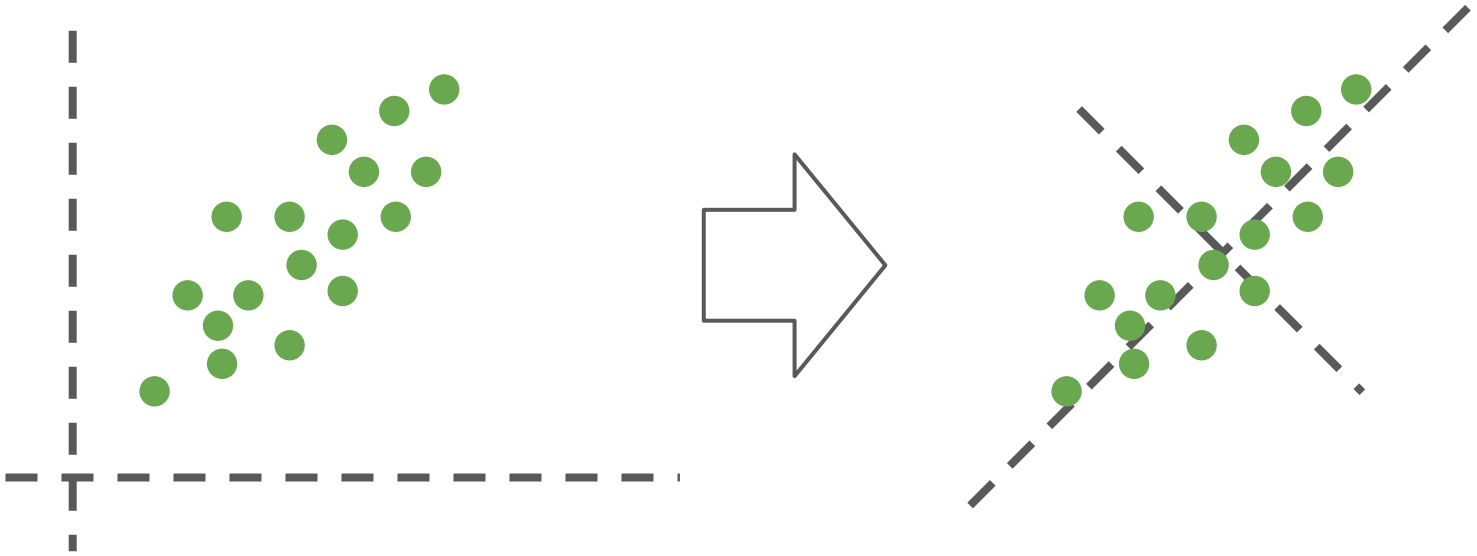
- U: vectores propios
- L: valores propios

$$|C - \Lambda| = 0$$

$$(C - \lambda_k I)U_{[:k]} = 0 \quad \forall k$$

Principal Component Analysis (PCA)

PCA encuentra un base ortogonal rotada con respecto al espacio original y cuyos ejes maximizan la varianza de los datos



Principal Component Analysis (PCA)

- La Matriz U es de MxM
- ¿Cómo reducimos dimensionalidad con PCA?
- Seleccionamos un subconjunto de los valores propios

Notemos que los valores propios nos indican la varianza proyectada

$$U^T C U = \Lambda$$

Criterio:

- Ordenamos los valores propios de mayor a menor
- Buscamos el primer D < M tal que:

$$J_D = \frac{\sum_{i=1}^{D<M} \lambda_i}{\sum_{i=1}^{M} \lambda_i} > 0.9$$

Principal Component Analysis (PCA)

Una vez que encontramos un valor para D creamos podemos crear nuestra matriz de proyección de menor dimensión a partir de las columnas de U que acumulan la mayor varianza

$$W = [u_{\lambda_1}, u_{\lambda_2}, \ldots, u_{\lambda_D}]$$

Finalmente nuestros datos proyectados

$$Z = (X - \bar{X})W$$

$$Z \in \mathbb{R}^{N \times D} \quad X \in \mathbb{R}^{N \times M} \quad W \in \mathbb{R}^{M \times D}$$

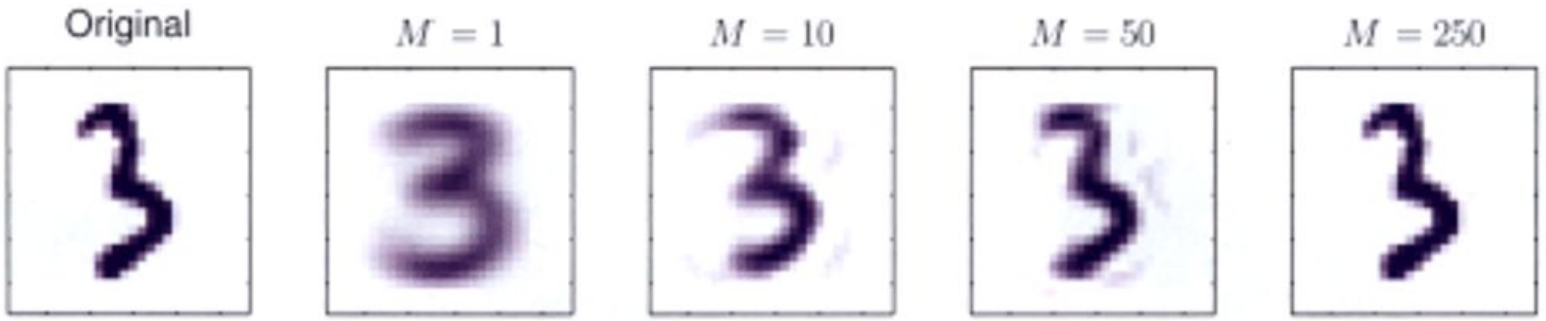
Donde X - X barra son los datos centrados (media cero)

Principal Component Analysis (PCA)

También es de interés estudiar la información perdida al reconstruir cuando se usan distintos valores de D

Una reconstrucción se obtiene reproyectando al espacio original luego de haber reducido dimensionalidad

En el siguiente ejemplo se aprecian reconstrucciones de un dígito manuscrito (MNIST, M=748) usando distinta cantidad de vectores propios



Principal Component Analysis (PCA)

Consideraciones adicionales

- Los datos deben estar centrados (media cero) antes de ser proyectados
- No olvidar restar la media cuando se calcula la covarianza de lo contrario la media de los datos dominará la proyección
- Si las características no están en la misma escala (unidades) es conveniente estandarizar (dividir columnas por su desviación estándar)
- PCA se puede usar para obtener una proyección esférica (blanqueada)

$$Z = (X - \bar{X})WL^{-1/2}$$

- Donde L es una matriz diagonal con los valores propios asociados a W. Se dice proyección esférica pues ahora Z tiene covarianza identidad

PCA probabilístico

Podemos expresar PCA como la solución de máxima verosimilitud para un modelo probabilístico de variables latentes continuas

Sea X ϵ $\mathbb{R}^{NxM}$ nuestros datos y Z ϵ $\mathbb{R}^{NxD}$ la variable latente

Definimos un prior Gaussiano isotrópico para los componentes principales

$$p(z) = \mathcal{N}(z|0, I)$$

Y para la probabilidad condicional x|z

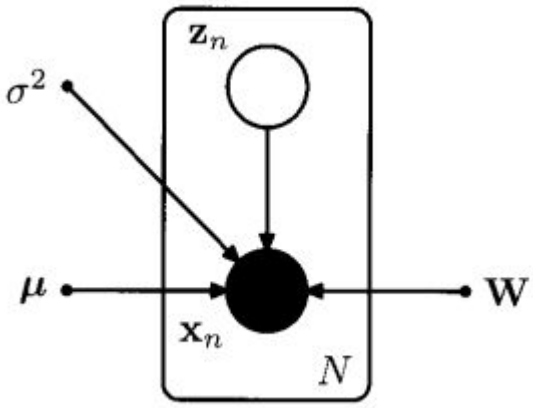
$$p(x|z) = \mathcal{N}(x|Wz + \mu, \sigma^2 I)$$

Donde W ϵ $\mathbb{R}^{MxD}$ y μ ϵ $\mathbb{R}^{D}$

PCA probabilístico

Escribimos la verosimilitud como

$$p(x) = \int p(x|z)p(z)dz$$



p(x) es Gaussiana pues es una convolución de dos gaussianas

Para encontrar sus parámetros reparametrizamos x

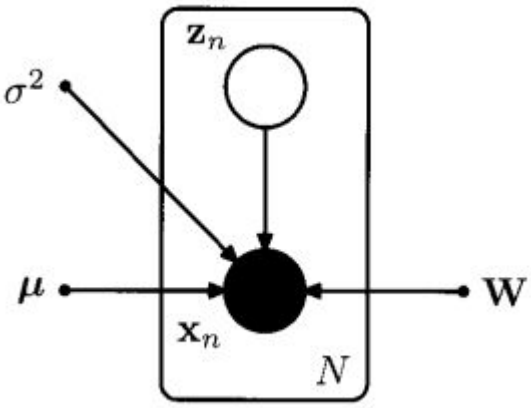
$$x = Wz + \mu + \varepsilon$$

$$\varepsilon \sim \mathcal{N}(0, \sigma^2 I)$$

PCA probabilístico

Escribimos la verosimilitud como

$$p(x) = \int p(x|z)p(z)dz$$



p(x) es Gaussiana pues es una convolución de dos gaussianas

Luego

$$\mathbb{E}[x] = W\mathbb{E}[z] + \mu + \mathbb{E}[\varepsilon] = \mu$$

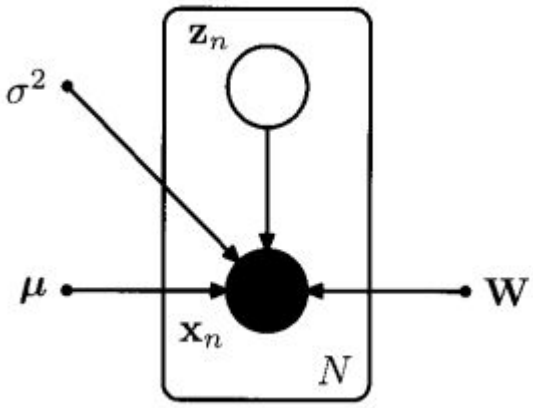
$$\begin{aligned}\mathrm{cov}[x] &= \mathbb{E}[(Wz + \varepsilon)(Wz + \varepsilon)^T] \\ &= W\mathbb{E}[zz^T]W^T + \mathbb{E}[\varepsilon\varepsilon^T] = WW^T + \sigma^2 I\end{aligned}$$

PCA probabilístico

Escribimos la verosimilitud como

$$p(x) = \int p(x|z)p(z)dz$$

p(x) es Gaussiana pues es una convolución de dos gaussianas



Finalmente

$$p(x) = \mathcal{N}(x|\mu, W^T W + \sigma^2 I)$$

PCA probabilístico

Haciendo un proceso similar para el posterior obtenemos que

$$p(z|x) = \mathcal{N}(z|M^{-1}W^T(x-\mu), \sigma^{-2}M)$$

Donde

$$M = W^TW + \sigma^2 I$$

Es decir sólo la media de z depende de X

PCA probabilístico

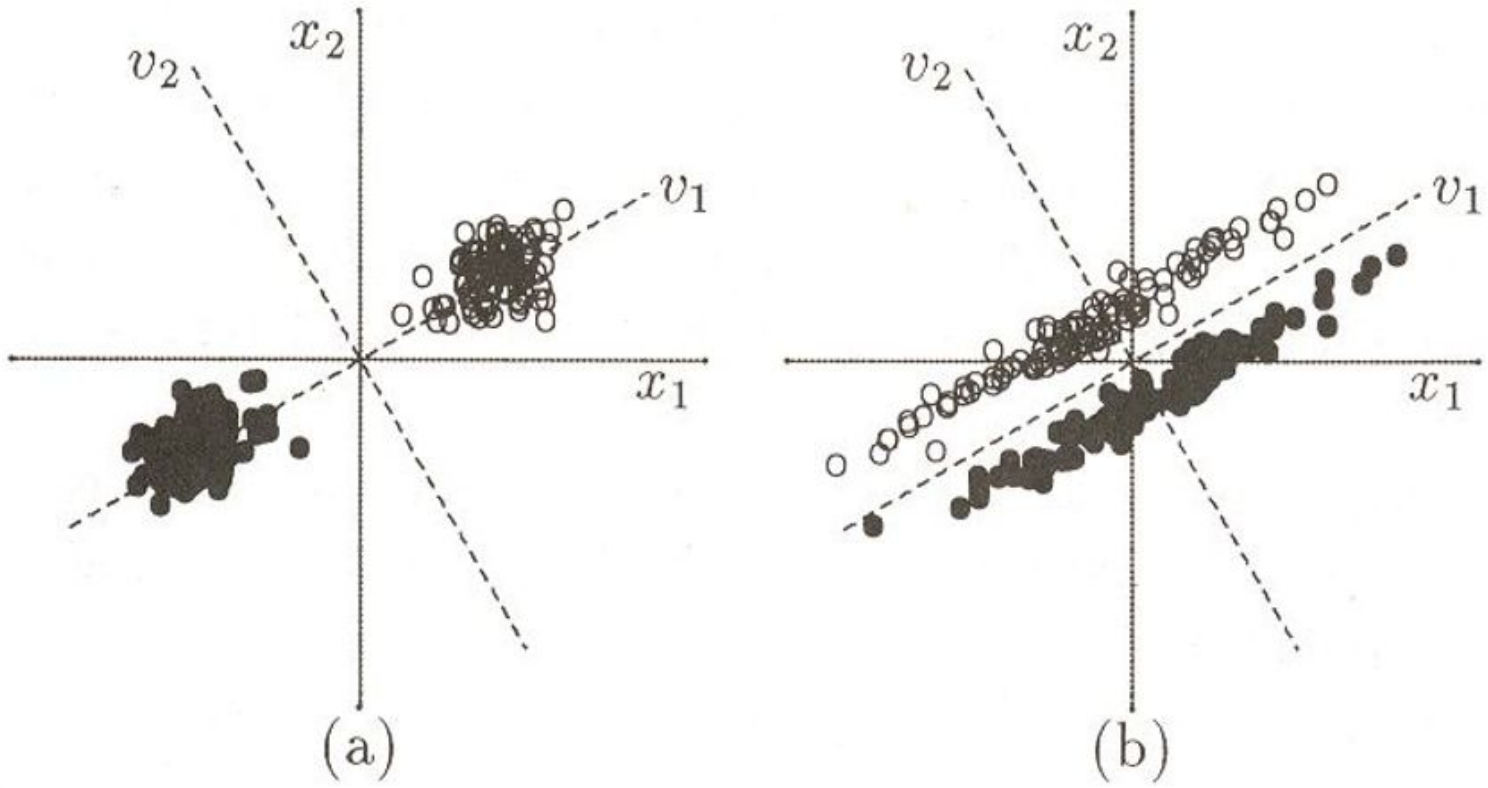
Para la solución de máxima verosimilitud resolvemos

$$\log p(X|\mu, W, \sigma^2) = \sum_{i=1}^{N} \log p(x_i|\mu, W, \sigma^2)$$

- De las derivadas se obtienen formas cerradas para μ, W y σ
- Si los datos están centrados se puede omitir μ
- Otra opción es asignar priors para los parámetros y usar un tratamiento Bayesiano completo
- Es particularmente interesante asignar un prior para W. Así es posible aprender el número óptimo de dimensiones para Z

Linear Discriminant Analysis (LDA)

- La proyección de máxima varianza puede no ser la de mayor interés



Linear Discriminant Analysis (LDA)

- La proyección de máxima varianza puede no ser la de mayor interés
- Técnica de reducción de dimensionalidad supervisada
- También se conoce como Discriminante Lineal de Fisher (FLD)
- La proyección encontrada es lineal (como en PCA)
- Busca una rotación de los ejes de características que maximice la **dispersión entre-clases** y minimice la **dispersión intra-clase**

Linear Discriminant Analysis (LDA)

- Sea un conjunto de datos M dimensionales con etiqueta categórica separados en dos clases

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

- Podemos definir las medias de los conjuntos como

$$m_1 = \frac{1}{N_1} \sum_{x_i \in C_1} x_i \qquad m_2 = \frac{1}{N_2} \sum_{x_i \in C_2} x_i$$

- Lo que buscamos es un vector de proyección w tal que

$$z = w^T x = \langle w, x \rangle$$

Linear Discriminant Analysis (LDA)

- Sea la matriz de dispersión entre-clases

$$S_B = (m_2 - m_1)(m_2 - m_1)^T$$

- Sea la matriz de dispersión intra-clases

$$S_W = \sum_{x_i \in C_1} (x_i - m_1)(x_i - m_1)^T + \sum_{x_i \in C_2} (x_i - m_2)(x_i - m_2)^T$$

- La función de costo que buscamos maximizar es

$$\max_w J(w) = \frac{w^T S_B w}{w^T S_W w}$$

Linear Discriminant Analysis (LDA)

- Derivando en función de w e igualando a cero se llega a

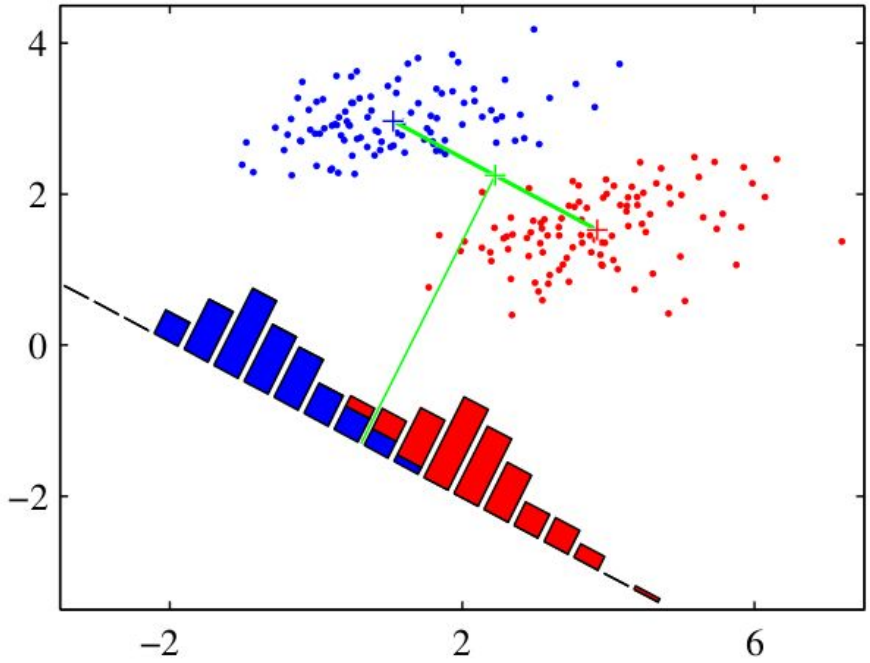
$$S_B w = \lambda S_W w$$
$$S_W^{-1} S_B w = \lambda w$$

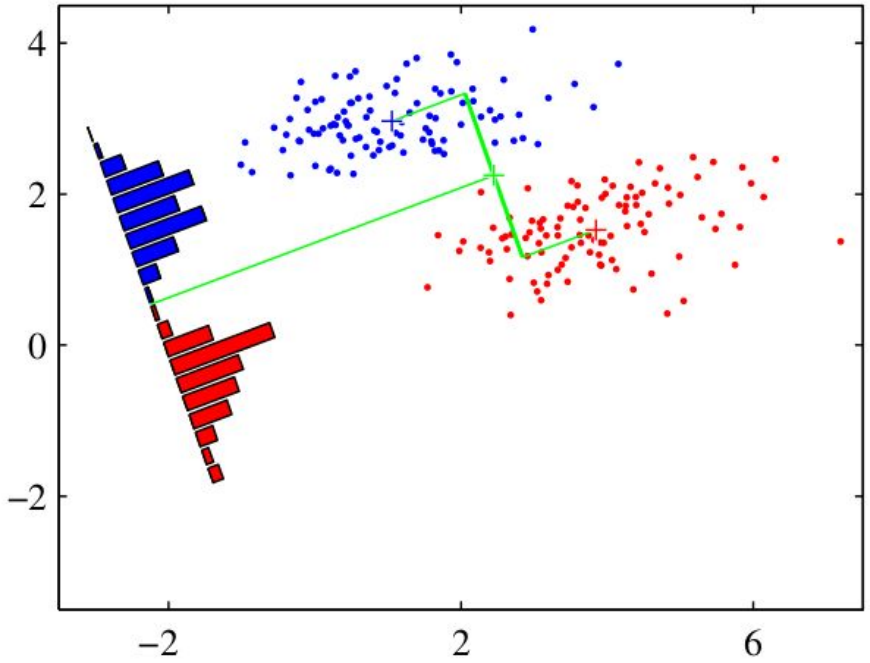
- Que corresponde al problema de autovalores
- Dado que $S_B w$ está en la dirección (m1 - m2)

$$w = S_W^{-1}(m_2 - m_1)$$

- Función discriminante: Si <w, x> supera un cierto umbral decimos que x pertenece a C1
- Esta regla de decisión es óptima si (a) las densidades condicionales son Gaussianas multivariadas y (b) las covarianzas son iguales.

Linear Discriminant Analysis (LDA)





Sección 4.1.4, Bishop

Multiple Discriminant Analysis (MDA)

- Se hacen una proyección a C-1 dimensiones de C es el número de clases
- Sólo funciona para datos con dimensionalidad M > C
- Se tiene ahora una matriz de proyección de M x (C-1)

$$z = W^T x$$

- La matriz de dispersión entre-clases con m la media total es

$$S_B = \sum_{c=1}^{C} N_c (m_c - m)(m_c - m)^T$$

- La matriz de dispersión intra-clases suma una covarianza por clases
- El vector óptimo de proyección se resuelve mediante autovalores

Independent Component Analysis (ICA)

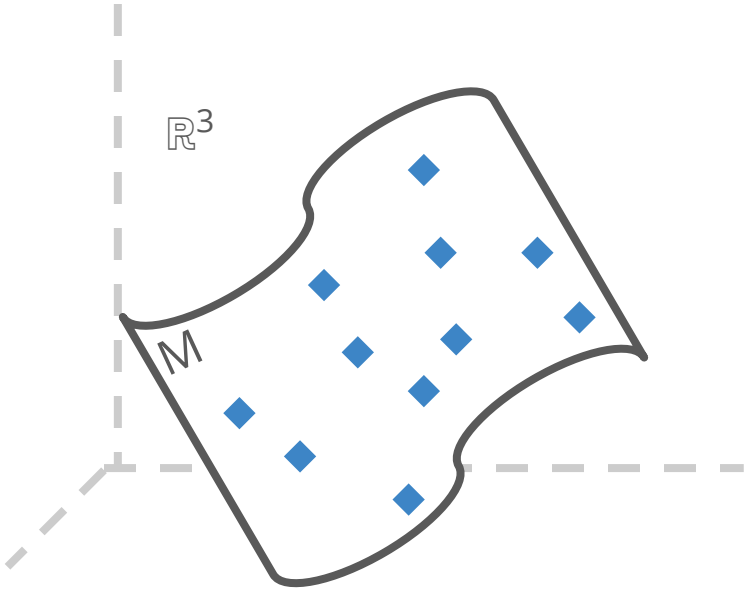
Pendiente

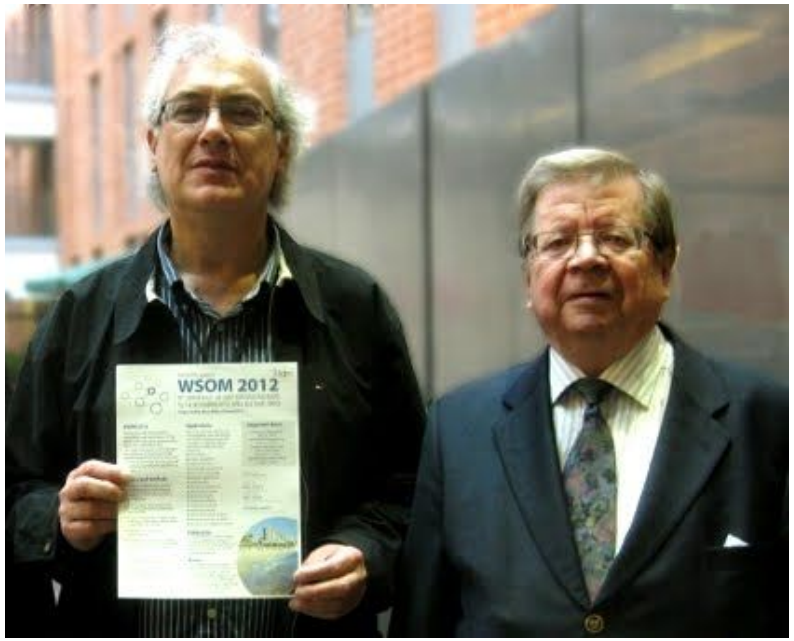
Non-negative matrix factorization (NMF)

Pendiente

Mapas auto-organizativos

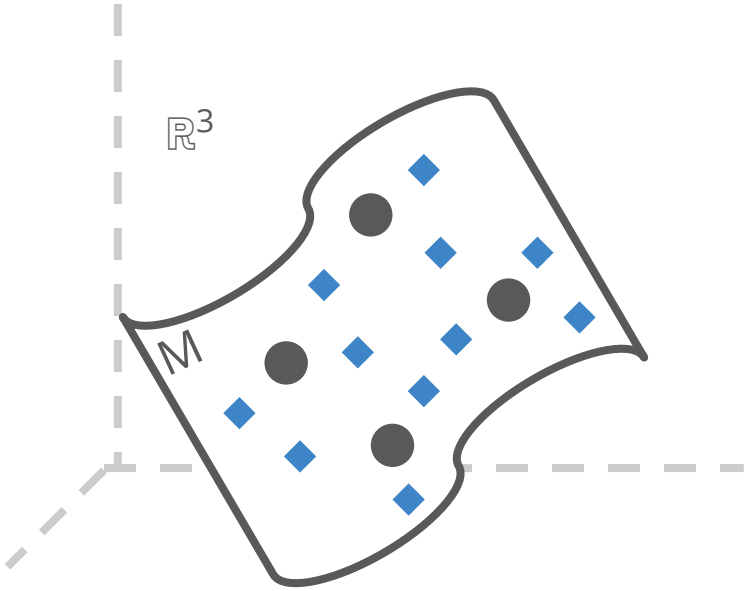
- Los mapas auto-organizativos o **mapas de Kohonen** son un tipo de red neuronal no-supervisada para hacer proyección y visualización de datos





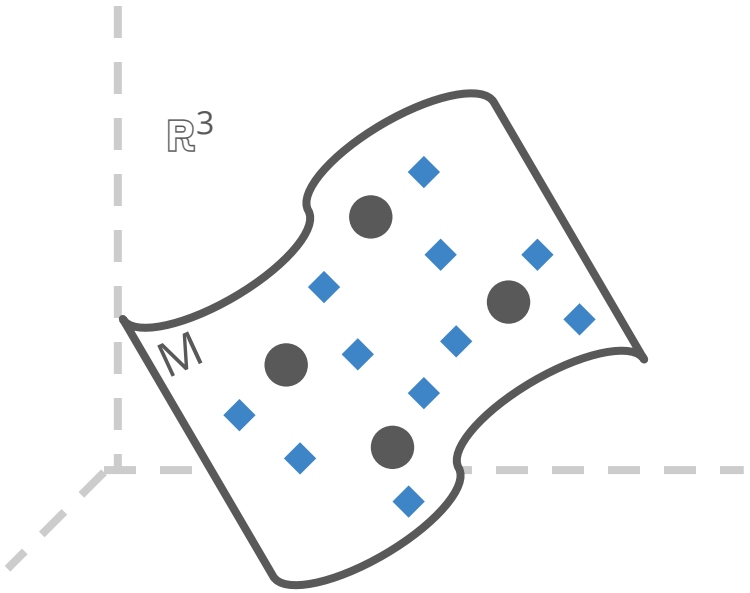
Mapas auto-organizativos

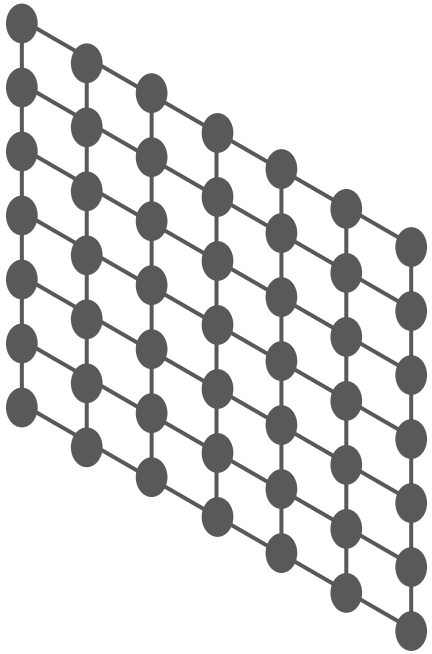
- Los mapas auto-organizativos o **mapas de Kohonen** son un tipo de red neuronal no-supervisada para hacer proyección y visualización de datos
- Los datos son representados por un conjunto de vectores prototipo



Mapas auto-organizativos

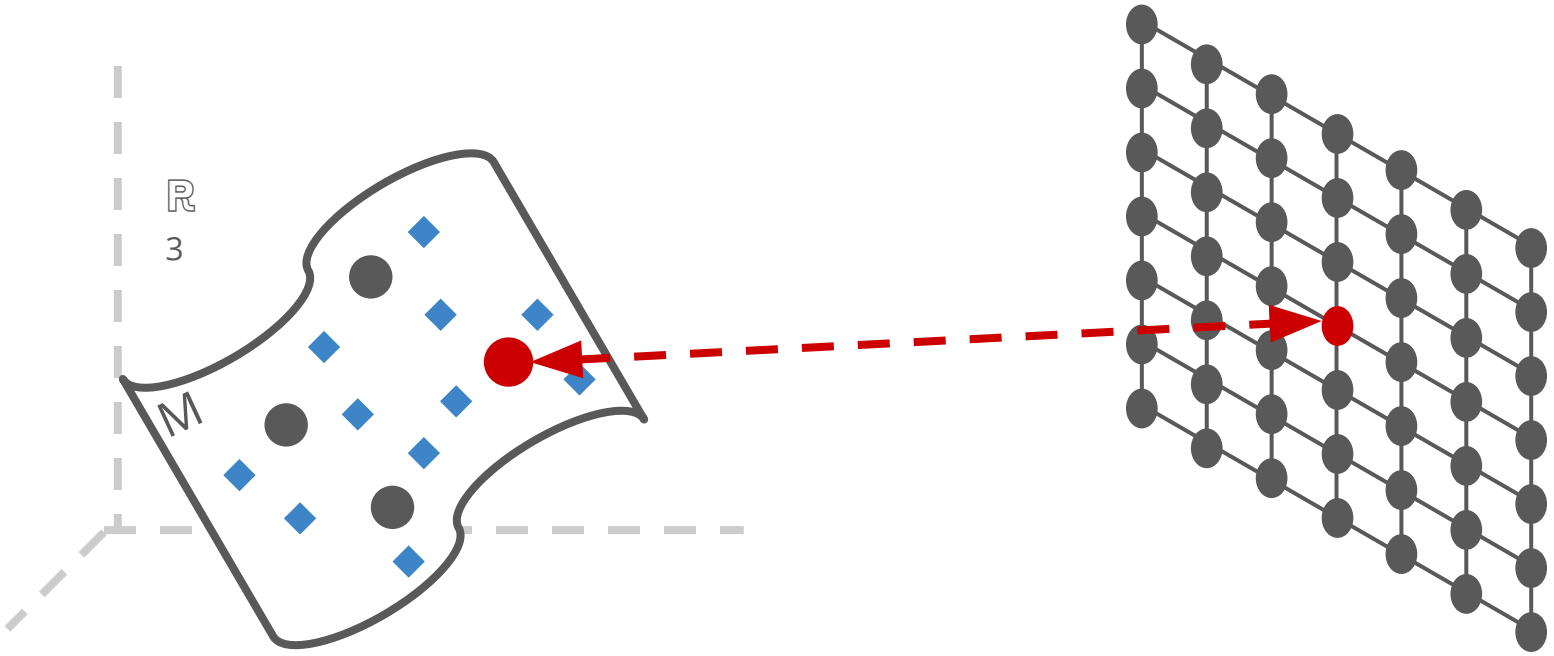
- Los mapas auto-organizativos o **mapas de Kohonen** son un tipo de red neuronal no-supervisada para hacer proyección y visualización de datos
- Los datos son representados por un conjunto de vectores prototipo
- Los prototipos se ordenan topológicamente en una grilla (bidimensional)





Mapas auto-organizativos

- Los mapas auto-organizativos o **mapas de Kohonen** son un tipo de red neuronal no-supervisada para hacer proyección y visualización de datos
- Los datos son representados por un conjunto de vectores prototipo
- Los prototipos se ordenan topológicamente en una grilla (bidimensional)
- Los prototipos viven en el espacio de entrada y en la grilla



Cuantización vectorial (VQ)

- Es una técnica para hacer compresión de datos
- VQ encuentro un conjunto de vectores prototipo (*codebook*) que representa a los datos
- Sea un conjunto de N datos D-dimensionales

$$X = \{x_1, x_2, \ldots, x_N\}$$

- Se busca un codebook con K prototipos

$$M = \{m_1, m_2, \ldots, m_K\} \text{ con } K < N$$

- Donde una muestra cualquiera es aproximada como

$$x \approx m_c \qquad c = \arg\min_i \|w - m_i\|$$

Cuantización vectorial (VQ)

- Es una técnica para hacer compresión de datos
- VQ encuentro un conjunto de vectores prototipo (*codebook*) que representa a los datos
- ¿Cómo se seleccionan los prototipos?
- Se usa la siguiente función de error

$$E = \int \|x - m_c\|_2^2 \, p(x) \, dx$$

- Si optimizamos esta función estamos aproximando $p(x)^{N/(N+2)}$

Cuantización vectorial (VQ)

- ¿Cómo se seleccionan los prototipos?
- Se usa la siguiente función de error

$$E = \int \|x - m_c\|_2^2 \, p(x) \, dx$$

$$x \approx m_c \qquad\qquad c = \arg\min_i \|x - m_i\|$$

$$\min_i a_i = \lim_{r \to -\infty} \left(\sum_i a_i^r \right)^{\frac{1}{r}}$$

$$\|x - m_c\|^2 = \lim_{r \to -\infty} \left(\sum_i \|x - m_i\|^r \right)^{\frac{2}{r}}$$

Cuantización vectorial (VQ)

- ¿Cómo se seleccionan los prototipos?
- Se usa la siguiente función de error

$$E = \int \|x - m_c\|_2^2 \, p(x) \, dx$$

$$x \approx m_c \qquad\qquad c = \arg\min_i \|x - m_i\|$$

$$\nabla_{m_j} E = \int \lim_{r \to -\infty} \nabla_{m_j} \left(\sum_i \|x - m_i\|^r \right)^{\frac{2}{r}} p(x) \, dx$$

Cuantización vectorial (VQ)

- ¿Cómo se seleccionan los prototipos?
- Se usa la siguiente función de error

$$E = \int \|x - m_c\|_2^2 \, p(x) \, dx$$

$$x \approx m_c \qquad\qquad c = \arg\min_i \|x - m_i\|$$

$$\nabla_{m_j} \left(\sum_i \|x - m_i\|^r\right)^{\frac{2}{r}} = -2 \left(\sum_i \|x - m_i\|^r\right)^{\frac{2}{r}} \frac{\|x-m_j\|^{r-2}}{\sum_i \|x-m_i\|^r}(x - m_j)$$

Al aplicar el limite: $= -2\|x - m_c\|^2 \|x - m_c\|^{-2} \delta_{cj}(x - m_j)$

$$= -2\delta_{cj}(x - m_j) \qquad \delta_{cj} = \begin{cases} 1 & c = j \\ 0 & c \neq j \end{cases}$$

Cuantización vectorial (VQ)

- ¿Cómo se seleccionan los prototipos?
- Se usa el siguiente algoritmo de aprendizaje secuencial

Para una nueva muestra x(t)

$$c = \arg\min_i \|x(t) - m_i\|$$

$$m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$$

$$m_i(t+1) = m_i(t) \quad \forall i \neq c$$

$$0 \leq \alpha(t) \leq 1$$

Mapas auto-organizativos (SOM)

- El mapa auto-organizativo conecta topológicamente los prototipos de VQ en una grilla bidimensional
- Noción de vecindad. Una muestra modifica el prototipo más cercano y sus vecinos topológicos

Algoritmo SOM

1. Encontrar BMU

$$c = \arg\min_i \|x - m_i\|$$

Mapas auto-organizativos (SOM)

- El mapa auto-organizativo conecta topológicamente los prototipos de VQ en una grilla bidimensional
- Noción de vecindad. Una muestra modifica el prototipo más cercano y sus vecinos topológicos

Algoritmo SOM

1. Encontrar BMU
2. Actualizar prototipos según BMU y vecindad

$$m_i(t+1) = \begin{cases} m_i(t) + \alpha(t)[x(t) - m_i(t)] & i \in N_c \\ m_i(t) & i \notin N_c \end{cases}$$

Mapas auto-organizativos (SOM)

- El mapa auto-organizativo conecta topológicamente los prototipos de VQ en una grilla bidimensional
- Noción de vecindad. Una muestra modifica el prototipo más cercano y sus vecinos topológicos

Algoritmo SOM

1. Encontrar BMU
2. Actualizar prototipos según BMU y vecindad
3. Disminuir la vecindad y la tasa de aprendizaje

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_f}{\alpha_0} \right)^{\frac{t}{t_f}}$$

Mapas auto-organizativos (SOM)

- El mapa auto-organizativo conecta topológicamente los prototipos de VQ en una grilla bidimensional
- Noción de vecindad. Una muestra modifica el prototipo más cercano y sus vecinos topológicos

Algoritmo SOM

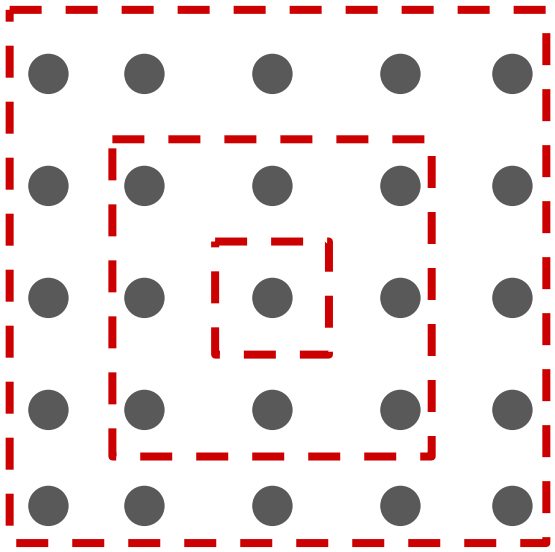
1. Encontrar BMU
2. Actualizar prototipos según BMU y vecindad
3. Disminuir la vecindad y la tasa de aprendizaje
4. Si no hay cambios, terminar

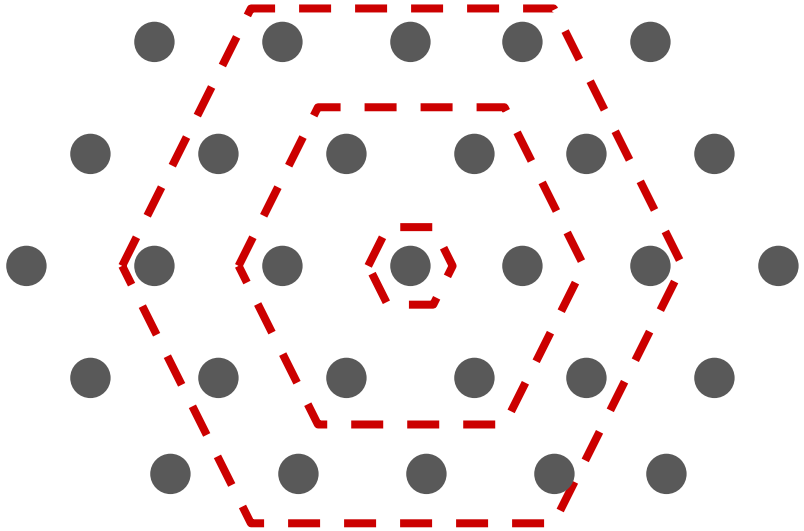
Mapas auto-organizativos (SOM)

- El mapa auto-organizativo conecta topológicamente los prototipos de VQ en una grilla bidimensional
- Noción de vecindad. Una muestra modifica el prototipo más cercano y sus vecinos topológicos

Algoritmo SOM

- Vecindad





Mapas auto-organizativos (SOM)

- El mapa auto-organizativo conecta topológicamente los prototipos de VQ en una grilla bidimensional
- Noción de vecindad. Una muestra modifica el prototipo más cercano y sus vecinos topológicos

Algoritmo SOM

- Vecindad: También se puede definir como

$$h_{ci}(t) = \exp\left(-\frac{\|r_c - r_i\|_2^2}{2\sigma(t)^2}\right)$$

Nota: r es la posición del prototipo en la grilla

Mapas auto-organizativos (SOM)

- El mapa auto-organizativo conecta topológicamente los prototipos de VQ en una grilla bidimensional
- Noción de vecindad. Una muestra modifica el prototipo más cercano y sus vecinos topológicos

Relación con k-means: En el caso de fijar la vecindad Nc = c, es decir sólo actualizar el BMU, se recupera una versión on-line de k-means

$$c = \arg\min_i \|x - m_i\| \qquad \alpha(t) = \alpha_0 \left(\frac{\alpha_f}{\alpha_0}\right)^{\frac{t}{t_f}}$$

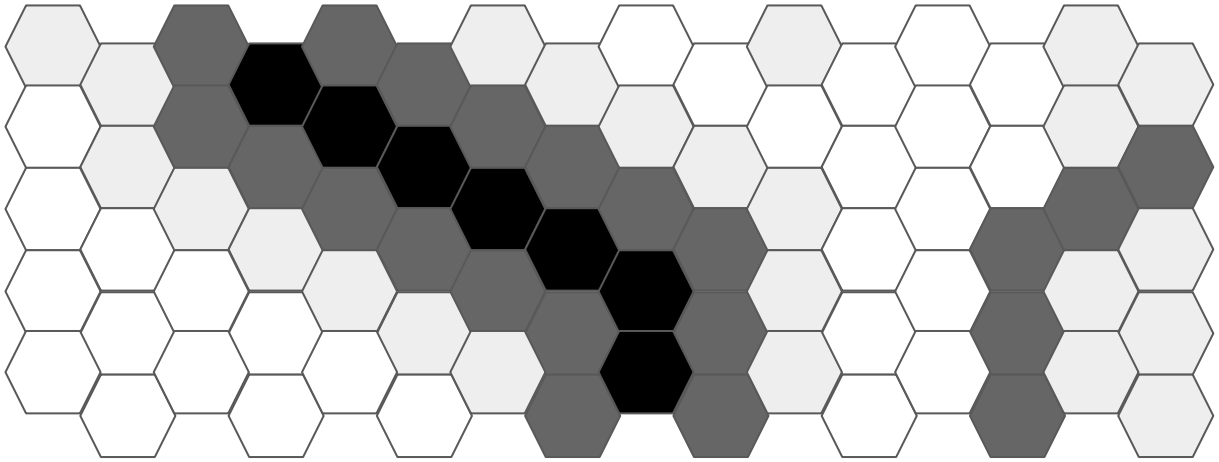
$$\min_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$$

Mapas auto-organizativos (SOM)

- Es posible usar SOM para hacer clustering
- Se pueden visualizar las distancias en la grilla usando técnicas de post-procesamiento
- Un ejemplo es la U-matrix (Unified-distance matrix)

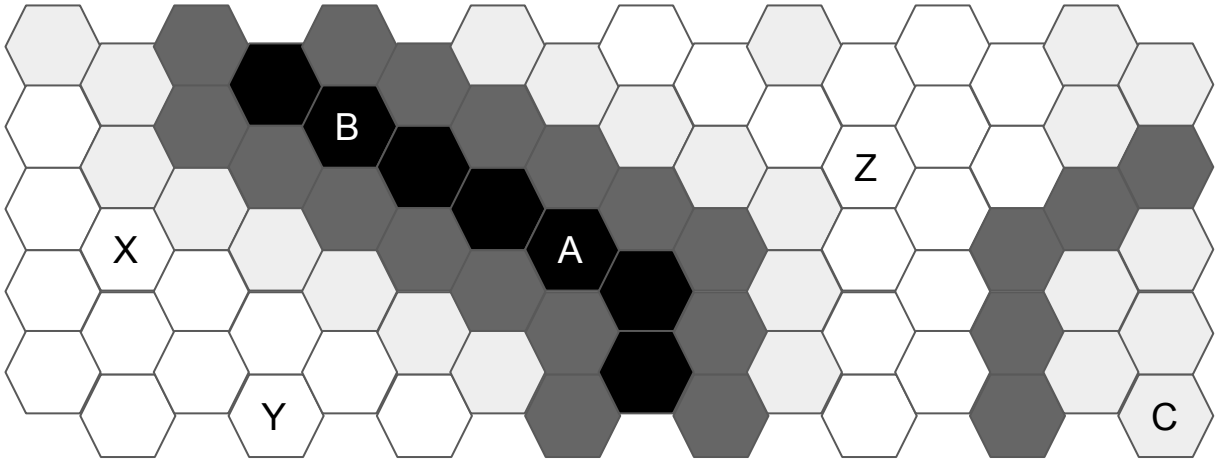
Mapas auto-organizativos (SOM)

- Es posible usar SOM para hacer clustering
- Se pueden visualizar las distancias en la grilla usando técnicas de post-procesamiento
- **U-matrix**: Suma de distancias a sus vecinos más cercanos. A cada prototipo se le asigna un color correspondiente a su lejanía relativa



Mapas auto-organizativos (SOM)

- Es posible usar SOM para hacer clustering
- Se pueden visualizar las distancias en la grilla usando técnicas de post-procesamiento
- **U-matrix**: Suma de distancias a sus vecinos más cercanos. A cada prototipo se le asigna un color correspondiente a su lejanía relativa
- ¿Cuáles ejemplos son similares?



Mapas auto-organizativos (SOM)

Recomendaciones de Teuvo Kohonen

- Condiciones de convergencia: Decaer la tasa de aprendizaje y la vecindad exponencialmente
- Las vecindades hexagonales son preferibles a las rectangulares por ser visualmente más ilustrativas y precisas
- Nx > Ny o viceversa para darle orientación al mapa y facilitar la convergencia
- El número de neuronas es la resolución del mapa. Más neuronas revelarán estructuras más finas pero incrementan el tiempo de cómputo. Se escoge mediante prueba-y-error
- Para evitar efectos de bordes se puede usar una grilla toroidal (cíclica)
- Para una convergencia más estable conviene separar el entrenamiento en etapa de ajuste grueso y etapa de ajuste fino

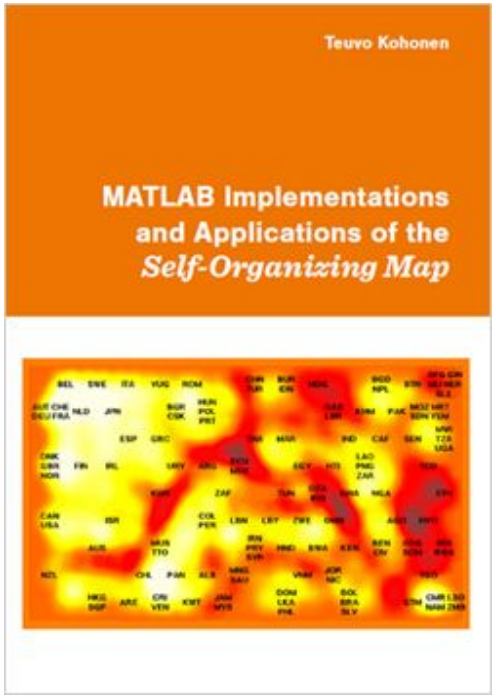
Mapas auto-organizativos (SOM)

Implementaciones

- Python: github.com/peterwittek/somoclu
- Matlab: http://www.cis.hut.fi/somtoolbox

Libro de Teuvo Kohonen con implementaciones en MATLAB disponible libremente en

docs.unigrafia.fi/publications/kohonen_teuvo/



Neural Gas (NG)

- Es una variante de SOM que no usa grilla
- NG es máx flexible ya que no está restringido por la forma de la grilla

Algoritmo NG

1. Encontrar BMU

$$c = \arg\min_i \|x - m_i\|$$

Neural Gas (NG)

- Es una variante de SOM que no usa grilla
- NG es máx flexible ya que no está restringido por la forma de la grilla

Algoritmo NG

1. Encontrar BMU
2. Actualizar prototipos según BMU y **ranking**

$$m_i(t+1) = m_i(t) + \alpha(t) h_j(t) [x(t) - m_i(t)]$$

$$h_j(t) = \exp\left(-\frac{r(x(t), w_j(t))}{\lambda(t)}\right)$$

Neural Gas (NG)

- Es una variante de SOM que no usa grilla
- NG es máx flexible ya que no está restringido por la forma de la grilla

Algoritmo NG

1. Encontrar BMU
2. Actualizar prototipos según BMU y **ranking**
3. Disminuir la vecindad y la tasa de aprendizaje

$$\alpha(t) = \alpha_0 \left(\frac{\alpha_f}{\alpha_0} \right)^{\frac{t}{t_f}} \qquad \lambda(t) = \lambda_0 \left(\frac{\lambda_f}{\lambda_0} \right)^{\frac{t}{t_f}}$$

Neural Gas (NG)

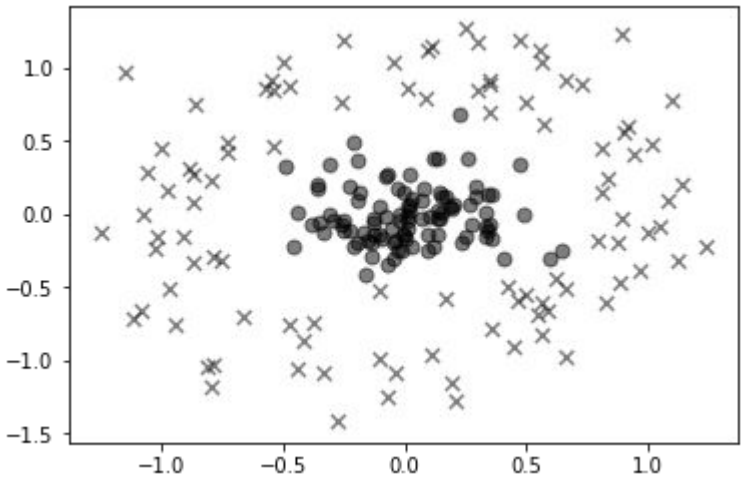
- Es una variante de SOM que no usa grilla
- NG es máx flexible ya que no está restringido por la forma de la grilla

Algoritmo NG

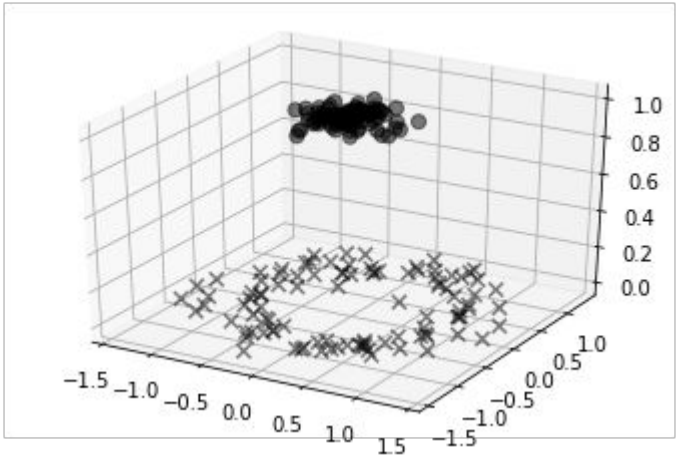
1. Encontrar BMU
2. Actualizar prototipos según BMU y **ranking**
3. Disminuir la vecindad y la tasa de aprendizaje
4. Si no hay cambios, terminar

Transformación no lineal

- Es posible transformar nuestros datos a un nuevo espacio de mayor dimensionalidad
- Los algoritmo lineales se pueden usar en el espacio transformado
- Generalmente es más fácil encontrar hiperplanos separadores o proyecciones en el espacio aumentado los cuales equivalen a fronteras o proyecciones no-lineales en el espacio de entrada



$$x_i \mapsto \Phi(x_i)$$



Kernel

- Un kernel es:
  - Un producto punto generalizado
  - Una generalización de una matriz/función definida positiva
  - Un operador de mapeo que tiene la propiedad de reproducción
- Truco del kernel

$$\kappa(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$$

- Para un cierto conjunto de datos definimos la matriz Gram como

$$K = \begin{pmatrix} \kappa(x_1, x_1) & \kappa(x_1, x_2) & \cdots & \kappa(x_1, x_N) \\ \kappa(x_2, x_1) & \kappa(x_2, x_2) & \cdots & \kappa(x_2, x_N) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(x_N, x_1) & \kappa(x_N, x_2) & \cdots & \kappa(x_N, x_N) \end{pmatrix} \quad z^T K z \geq 0 \; \forall \; z \neq 0$$

Sección 5.8.1, Hastie, Tibshirani, Friedman

Kernel

- Sea un kernel

$$\kappa(x, z) = \left(\langle x, z \rangle\right)^2$$

- ¿Qué transformación no-lineal induce en los datos?

$$\left(\langle x, z \rangle\right)^2 = \left(x_1 z_1 + x_2 z_2\right)^2$$

$$= \left(x_1 z_1\right)^2 + 2\left(x_1 z_1 x_2 z_2\right) + \left(x_2 z_2\right)^2$$

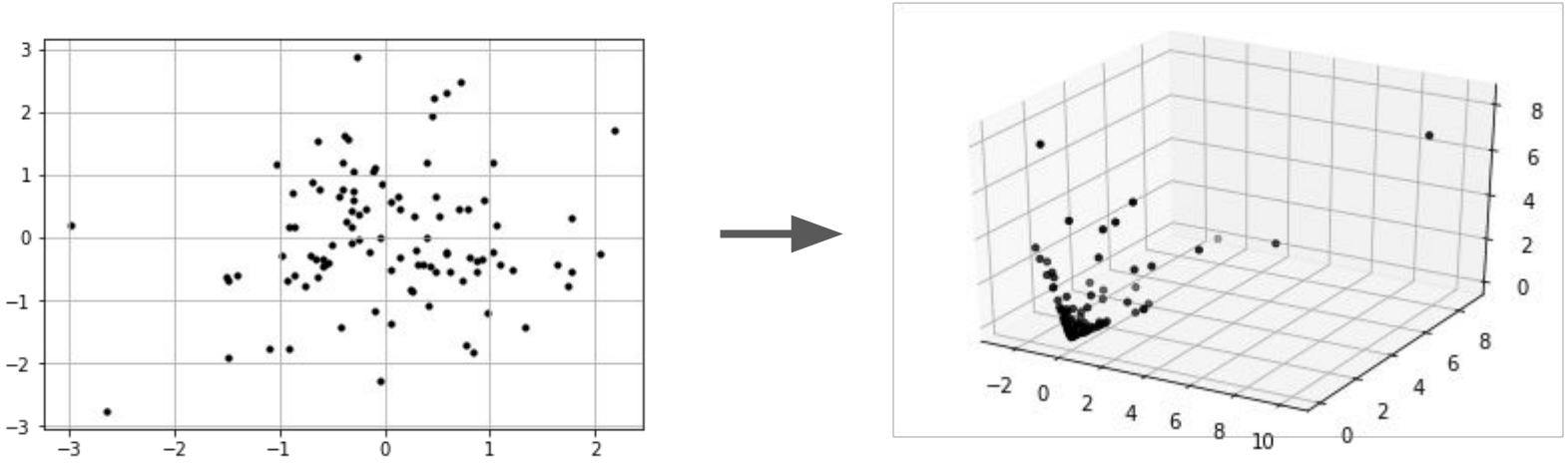
$$= \left\langle \begin{pmatrix} x_1^2 \\ \sqrt{2} x_1 x_2 \\ x_2^2 \end{pmatrix}, \begin{pmatrix} z_1^2 \\ \sqrt{2} z_1 z_2 \\ z_2^2 \end{pmatrix} \right\rangle$$

Kernel

- Sea un kernel

$$\kappa(x, z) = (\langle x, z \rangle)^2$$

$$= \left\langle \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}, \begin{pmatrix} z_1^2 \\ \sqrt{2}z_1z_2 \\ z_2^2 \end{pmatrix} \right\rangle$$



Ejemplos de kernels

Kernel polinomial inhomogeneo

$$\kappa(x, z) = (\langle x, z \rangle + 1)^d$$

Kernel Gaussiano o Radial Basis Function (RBF)

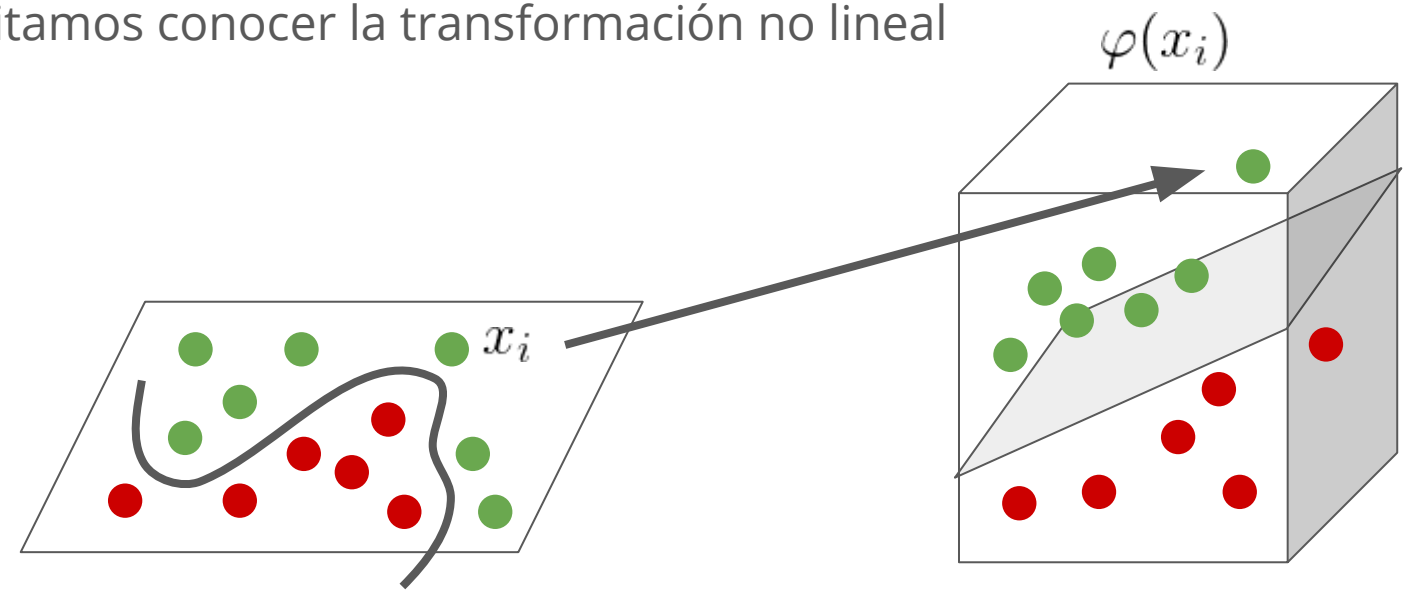
$$\kappa(x, z) = \exp\left(-\frac{\|x - z\|_2^2}{2\sigma^2}\right) = \exp\left(-\gamma\|x - z\|_2^2\right)$$

Kernel tangente hiperbólica

$$\kappa(x, z) = \tanh(c_1\langle x, z \rangle + c_2)$$

Kernel PCA

- Es una versión "*kernelizada*" de PCA
- *Kernel* es un producto interno en un espacio de alta dimensión que induce una transformación no lineal
- Usando *kernels* podemos generalizar PCA al caso no-lineal
- No necesitamos conocer la transformación no lineal



1. B. Sholkopf, A. Smola, KR Muller, "Kernel Principal Component Analysis", Springer, 1997
2. Sección 12.3, Bishop, Pattern Recognition and Machine Learning

Kernel PCA

Recordemos, PCA es una proyección de máxima varianza que resulta de resolver el problema de valores propios para la matriz de covarianza

$$C = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$$

Sea ahora una transformación no lineal que aumenta la dimensionalidad de los datos, definimos una covarianza generalizada (matriz gram)

$$K = \frac{1}{N}\sum_{i=1}^{N}\varphi(x_i)\varphi(x_i)^T$$

Donde asumimos que los datos transformados están centrados

Kernel PCA

Luego el objetivo es resolver el problema de autovalores para K

$$Kv_j = \lambda_j v_j \quad \forall j = 1, \ldots, M$$

Los valores propios tienen la forma

$$v_j = \sum_{i=1}^{N} \alpha_{ij} \varphi(x_i)$$

Son una combinación lineal que usa la transformación no lineal como base

Kernel PCA

Reemplazando arriba se tiene y multiplicando por φ^T

$$N\lambda K\alpha = K^2\alpha$$

Se encuentra autovalores equivalentes resolviendo

$$N\lambda\alpha = K\alpha$$

Usamos el **truco del kernel** para formar la matriz Gram

$$K_{ij} = \langle \varphi(x_i), \varphi(x_j) \rangle$$

Kernel PCA

Se restringe que los componentes de V estén normalizados

$$1 = v_j^T v_j = \alpha_j^T K \alpha_j$$

Para proyectar una nueva muestra al espacio de los componentes principales *kernelizados*

$$y_j(x) = \varphi(x)^T v_j = \sum_{i=1}^{N} \alpha_{ij} \kappa(x_i, x)$$

Kernel PCA

Algoritmo

1. **Computar matriz gram**
2. Normalizar matriz gram
3. Resolver el problema de v.p.
4. Normalizar los coeficientes
5. Proyectar los datos

$$K_{ij} = \kappa(x_i, x_j)$$

Kernel PCA

Algoritmo

1. Computar matriz gram
2. **Normalizar matriz gram**
3. Resolver el problema de v.p.
4. Normalizar los coeficientes
5. Proyectar los datos

$$\tilde{K}_{ij} = K_{ij} - \frac{1}{N}\sum_{p=1}^{N} K_{ip} - \frac{1}{N}\sum_{q=1}^{N} K_{qj} + \frac{1}{N^2}\sum_{p=1,q=1}^{N,N} K_{qp}$$

Kernel PCA

Algoritmo

1. Computar matriz gram
2. Normalizar matriz gram
3. **Resolver el problema de v.p.**
4. Normalizar los coeficientes
5. Proyectar los datos

$$N\lambda\alpha = K\alpha$$

Kernel PCA

Algoritmo

1. Computar matriz gram
2. Normalizar matriz gram
3. Resolver el problema de v.p.
4. **Normalizar los coeficientes**
5. Proyectar los datos

$$1 = \alpha^* \cdot K\alpha^* = \Lambda\alpha^* \cdot \alpha^*$$

Kernel PCA

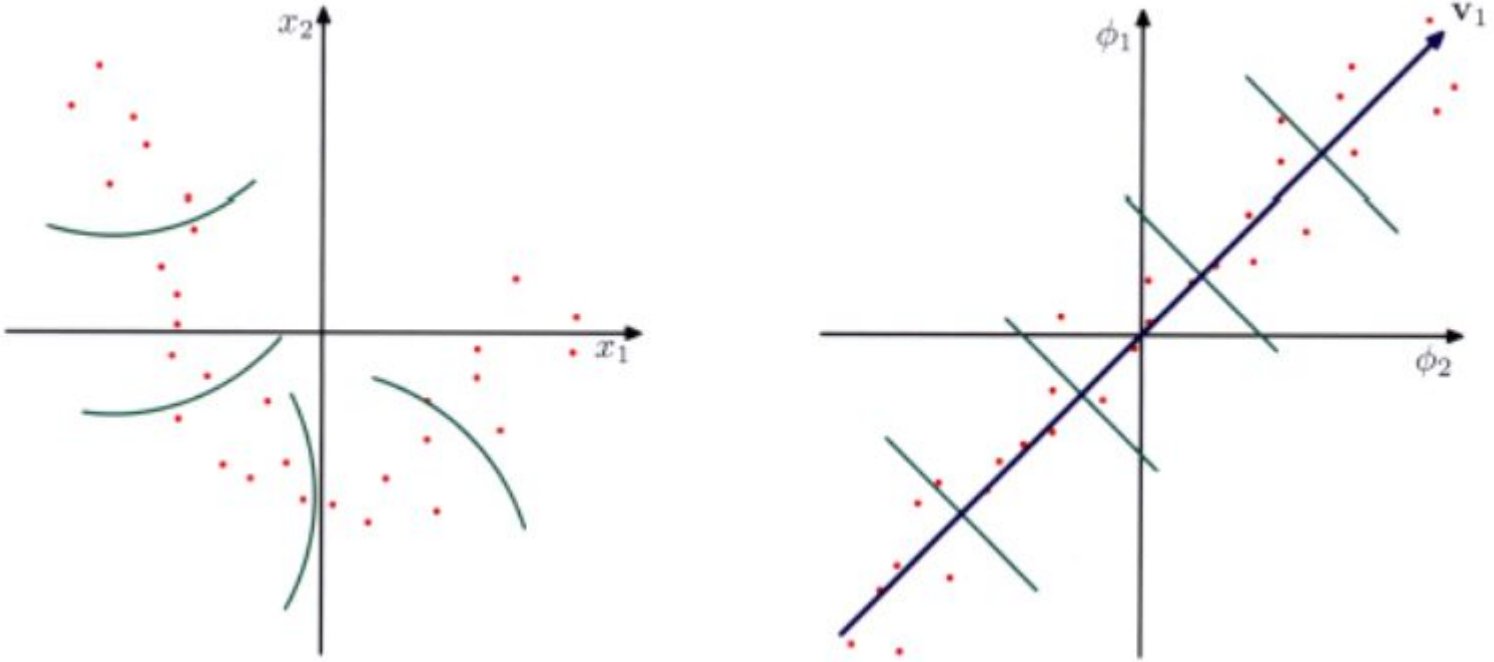
Algoritmo

1. Computar matriz gram
2. Normalizar matriz gram
3. Resolver el problema de v.p.
4. Normalizar los coeficientes
5. **Proyectar los datos**

$$y_j(x) = \sum_{i=1}^{N} \alpha_{ij} \kappa(x_i, x)$$

Kernel PCA

Proyección lineal en el espacio de alta dimensionalidad equivale a una proyección no-lineal en el espacio de entrada

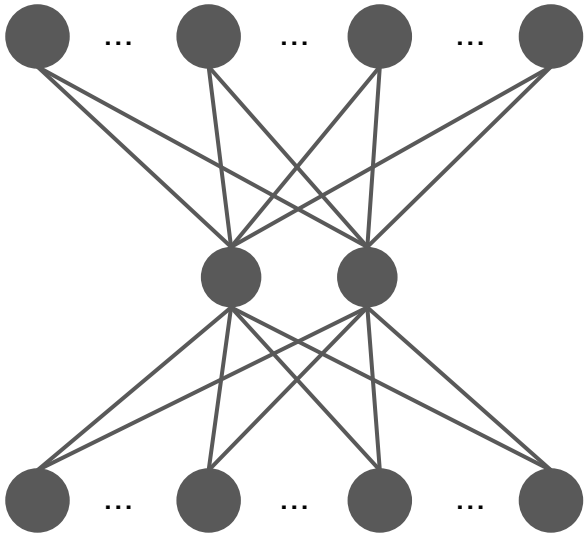


Kernel PCA

- No necesitamos conocer la transformación no lineal (truco del kernel)
- Lo complejidad computacional no aumenta con la dimensión de la transformación no lineal
- Si se limita el número de componentes principales en la proyección se reduce la dimensionalidad de los datos
- Proyectar de vuelta al espacio original no es trivial
- Dos estrategias:
  - Características no lineales + clasificador lineal
  - Características lineales + clasificador no lineal

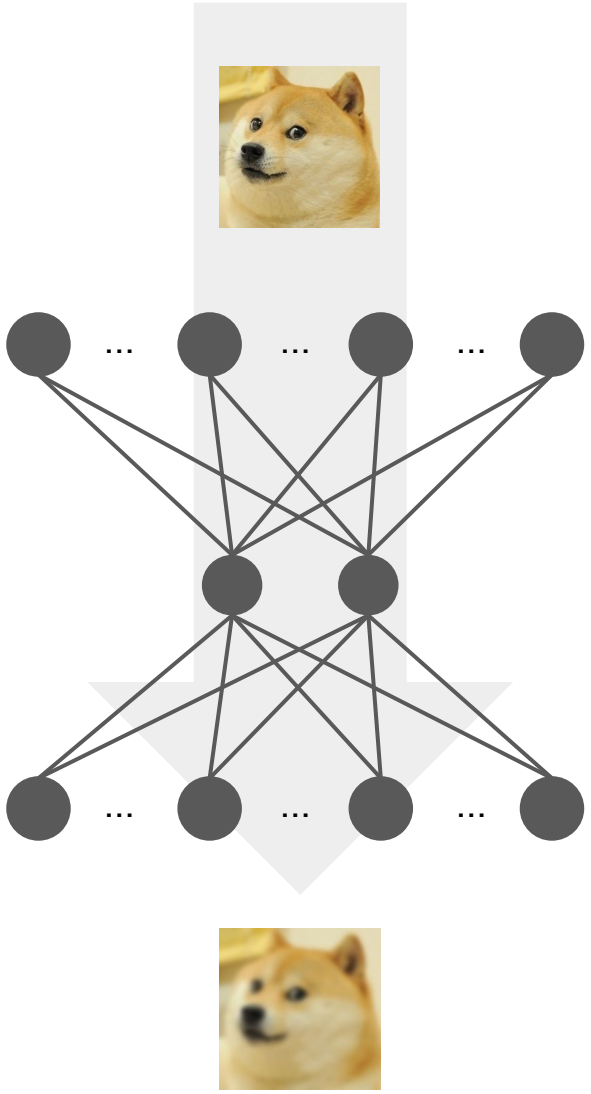
Autoencoder

- Redes neuronales no-supervisadas
- Arquitectura diabolo



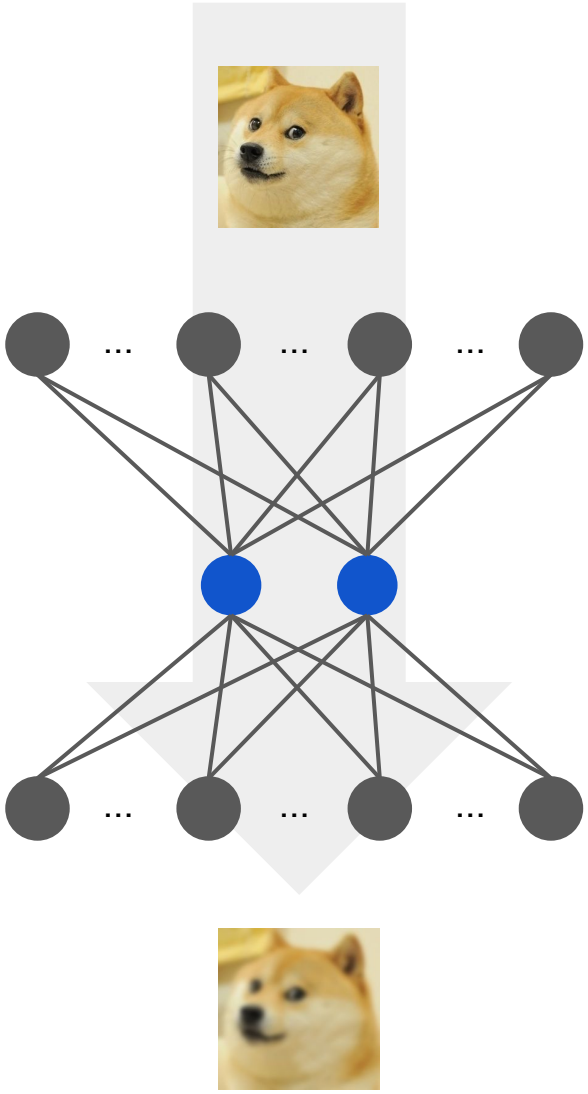
Autoencoder

- Redes neuronales no-supervisadas
- Arquitectura diabolo
- Paradigma *self-taught learning (auto-aprendizaje)*
- El objetivo es reproducir la entrada



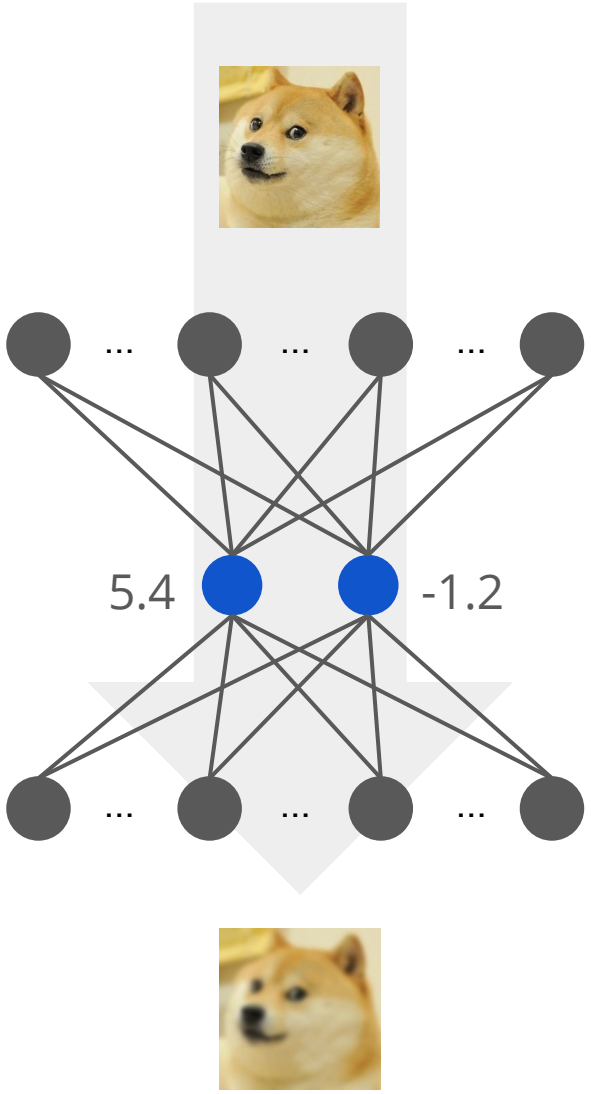
Autoencoder

- Redes neuronales no-supervisadas
- Arquitectura diabolo
- Paradigma *self-taught learning (auto-aprendizaje)*
- El objetivo es reproducir la entrada
- Cuello de botella: **variable latente**



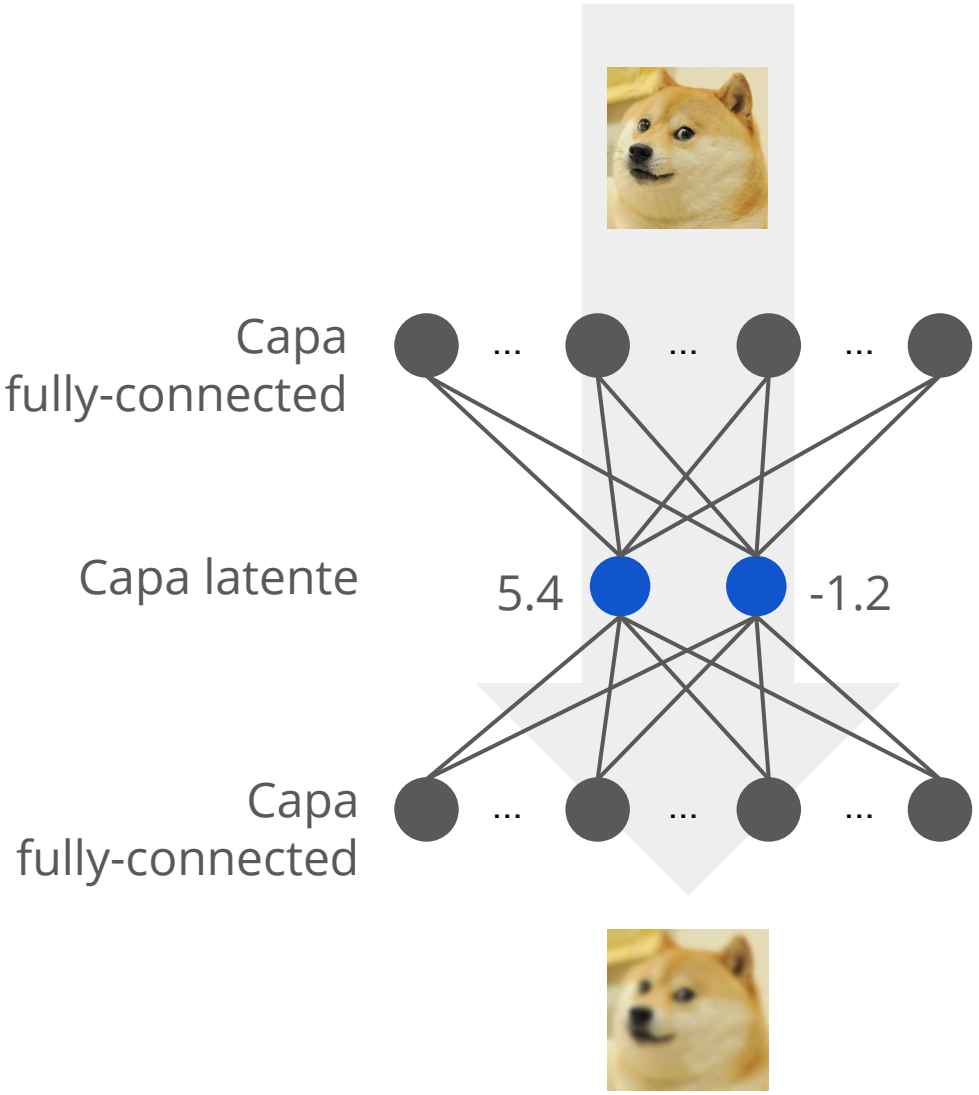
Autoencoder

- Redes neuronales no-supervisadas
- Arquitectura diabolo
- Paradigma *self-taught learning (auto-aprendizaje)*
- El objetivo es reproducir la entrada
- Cuello de botella: **variable latente**
- Se usa para aprender representaciones latentes comprimidas específicas al conjunto de datos



Autoencoder

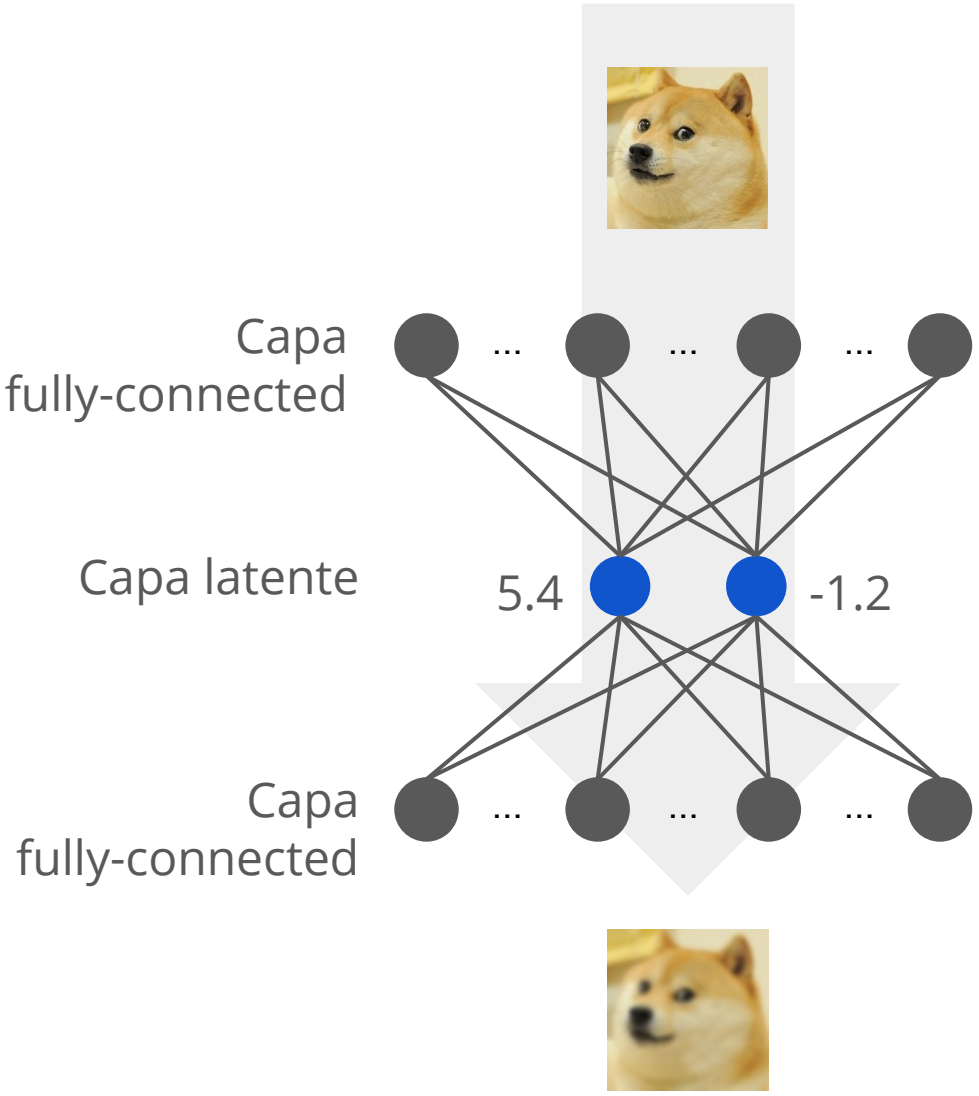
Ejemplo de arquitectura de autoencoder



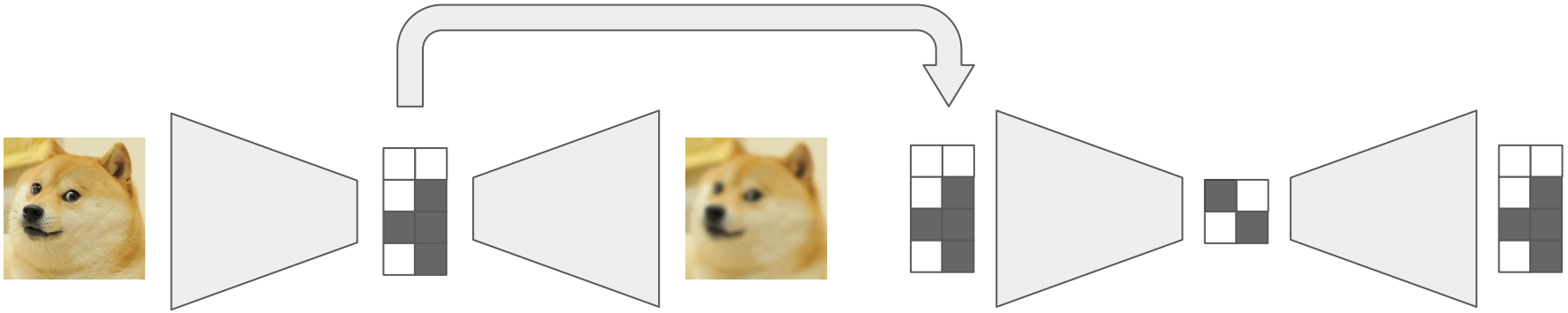
Autoencoder

Ejemplo de arquitectura de autoencoder

- Pueden concatenarse varias capas densas
- Si se usa activación lineal se recupera una solución similar a PCA
- Se entrena usando MSE para variables continuas o cross-entropy para variables categóricas (imágenes)
- Los autoencoders profundos son difíciles de entrenar

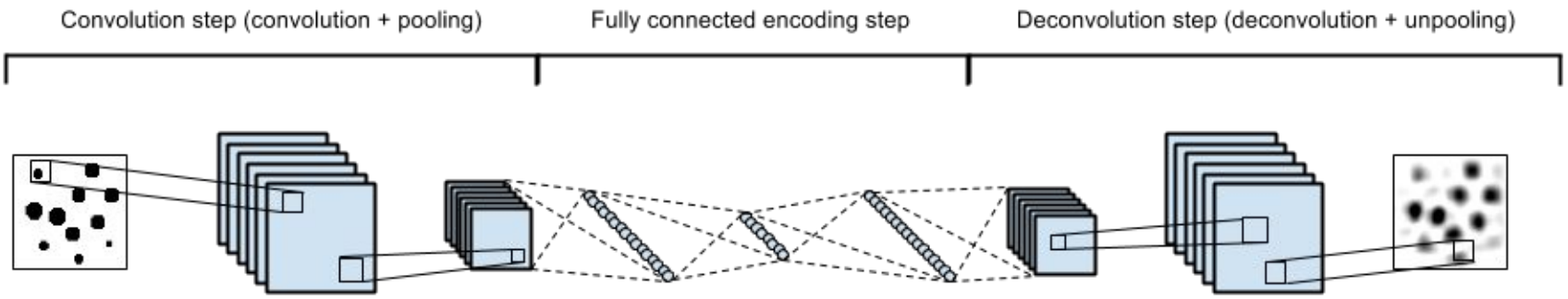


Autoencoder



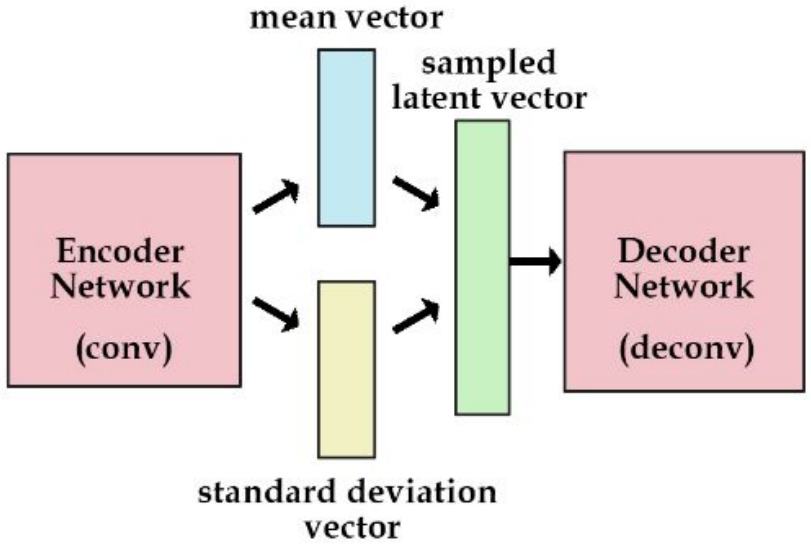
- **Stacked autoencoder:** Autoencoders conectados en serie

Autoencoder



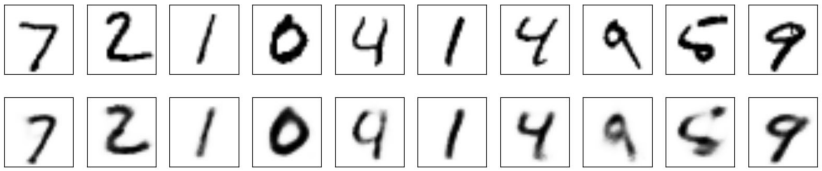
- **Stacked autoencoder:** Autoencoders conectados en serie
- **Convolutional autoencoder:** Se usan capas convoluciones y de-convolucionales para aprovechar las invarianzas y características locales que se aprenden con estas arquitecturas

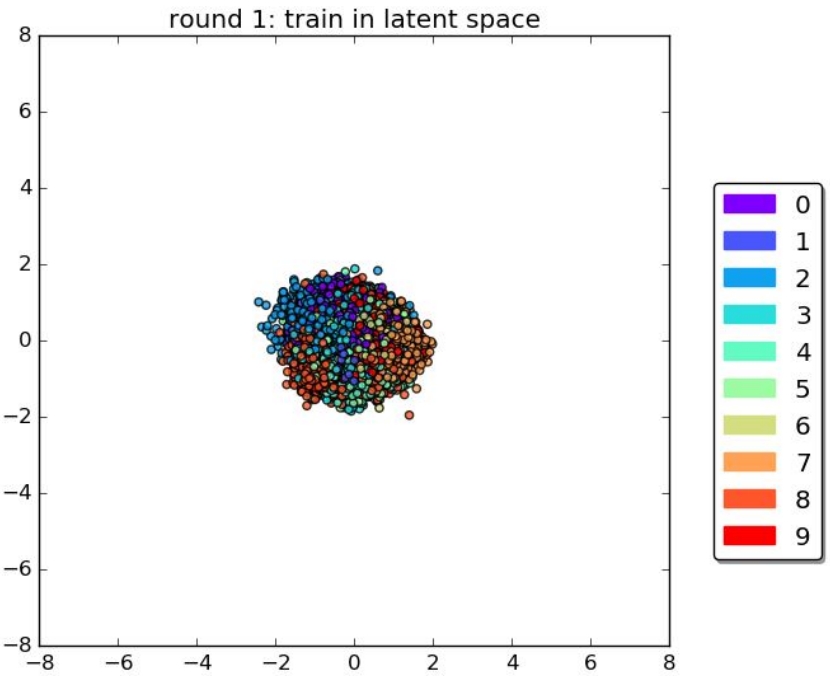
Autoencoder

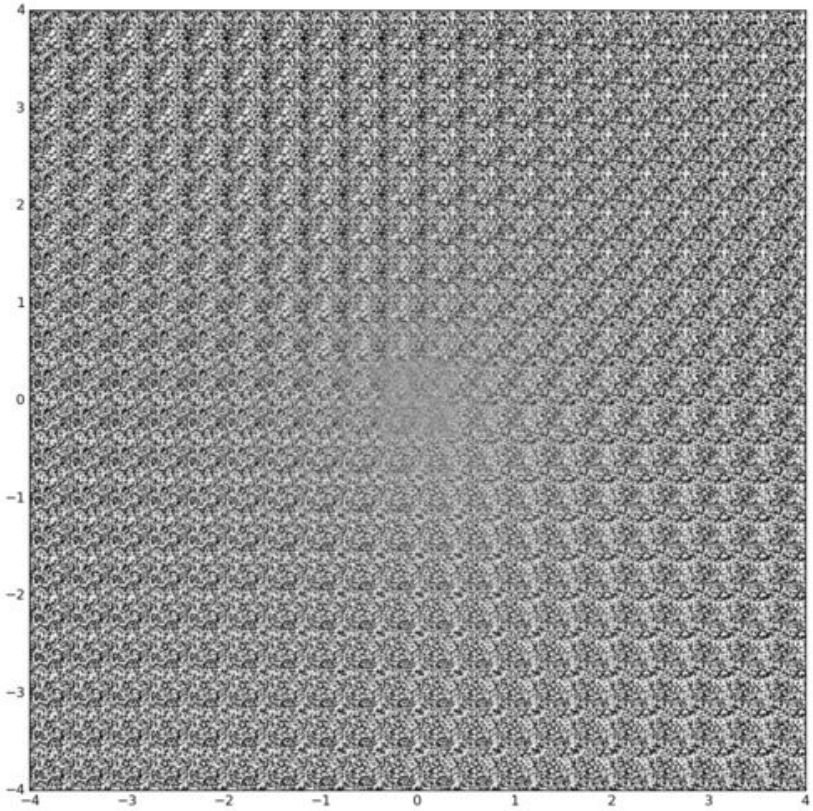


- **Stacked autoencoder:** Autoencoders conectados en serie
- **Convolutional autoencoder:** Se usan capas convoluciones y de-convolucionales para aprovechar las invarianzas y características locales que se aprenden con estas arquitecturas
- **Sparse autoencoder:** Se penaliza con norma L1 la capa latente
- **Denoising autoencoder:** Se agrega ruido a la entrada antes de entrenar
- **Variational autoencoder:** Utiliza un modelo generativo en la capa latente

Autoencoder Variacional







GIFs: http://blog.fastforwardlabs.com

t-distributed Stochastic Neighbor Embedding

- Es una técnica de reducción de dimensionalidad no lineal
- Enfocado a 2 (3) dimensiones (visualización)
- Usa métricas de teoría de la información
- Generalización de Stochastic Neighbor Embedding (SNE)

L.J.P. van der Maaten, G.E. Hinton, "Visualizing High-Dimensional Data Using t-SNE", *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008

t-distributed Stochastic Neighbor Embedding

Generalización de **Stochastic Neighbor Embedding** (SNE)

Sea

$$X = (x_1, x_2, \ldots, x_N) \; x_i \in \mathbb{R}^M$$
$$Y = (y_1, y_2, \ldots, y_N) \; y_i \in \mathbb{R}^2$$

Se generan probabilidades condicionales

$$p(x_j|x_i) = p_{i|j} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

$$q_{i|j} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k\neq i}\exp(-\|y_i - y_k\|^2)}$$

Se interpreta como la pbb de que xi sea vecino de xj dentro de una vecindad de tamaño si

t-distributed Stochastic Neighbor Embedding

Generalización de **Stochastic Neighbor Embedding** (SNE)

El objetivo es preservar las similitudes, que se traduce a igualar las pbb condicionales, para esto se minimiza

$$L = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

La divergencia KL no es simétrica. Para seleccionar el tamaño de vecindad:

$$Perp = 2^{H(P_i)} \qquad H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

La “perplejidad” se interpreta como el Nº efectivo de vecinos

t-distributed Stochastic Neighbor Embedding

En t-SNE se generaliza SNE

- Versión simétrica de la función de costo
- Distribución t-Student en vez de Gaussiana

Esto facilita la optimización del funcional y alivia el “problema de *crowding*”

t-distributed Stochastic Neighbor Embedding

Se reemplazan las probabilidades condicionales por conjuntas

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq h}\exp(-\|x_h - x_k\|^2/2\sigma^2)} \qquad p_{ij} \sim \frac{(p_{j|i} + p_{i|j})}{2n}$$

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k\neq h}\exp(-\|y_h - y_k\|^2)}$$

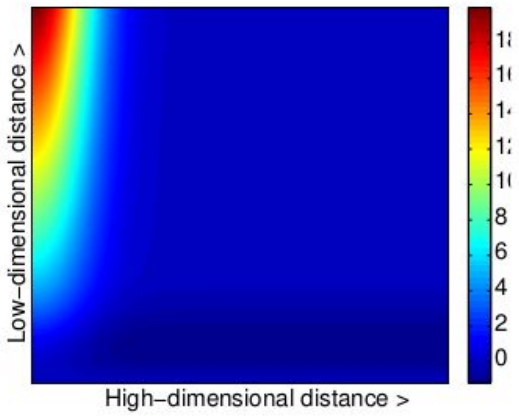
$$L = KL(P||Q) = \sum_i \sum_j p_{ji} \log \frac{p_{ji}}{q_{ji}}$$

$$\frac{dL}{dy_i} = 4\sum_j (p_{ji} - q_{ji})(y_i - y_j)$$

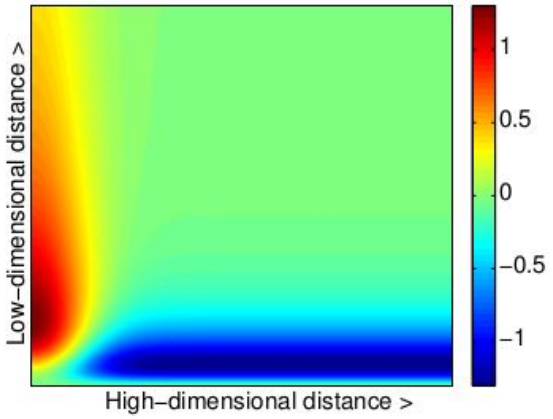
t-distributed Stochastic Neighbor Embedding

Para el espacio de baja dimensionalidad se reemplaza la distribución Gaussiana por una distribución t-Student de colas anchas

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq h}(1 + \|y_k - y_h\|^2)^{-1}}$$



(a) Gradient of SNE.



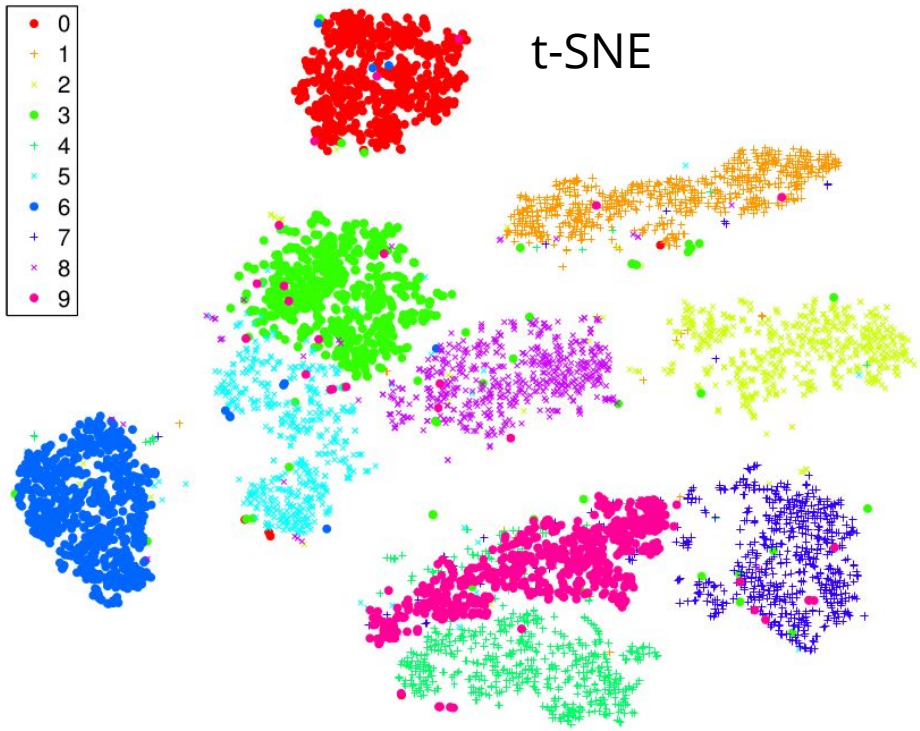
(c) Gradient of t-SNE.

t-distributed Stochastic Neighbor Embedding

Algorithm 1: Simple version of t-Distributed Stochastic Neighbor Embedding.

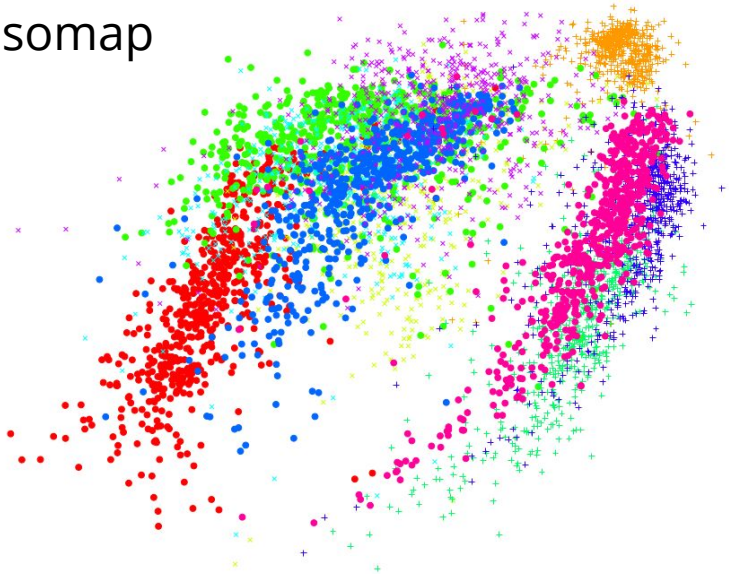
Data: data set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations T, learning rate η, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, ..., y_n\}$.
begin
  compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
  set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
  sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, ..., y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
  for t=*1* **to** T **do**
    compute low-dimensional affinities q_{ij} (using Equation 4)
    compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
    set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)} \right)$
  end
end

t-distributed Stochastic Neighbor Embedding

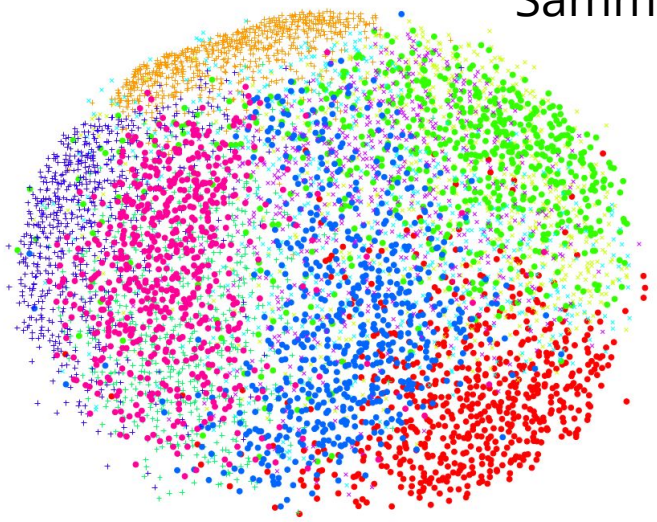


t-SNE

Isomap



Sammon



http://distill.pub/2016/misread-tsne/

t-distributed Stochastic Neighbor Embedding

- Revela clusters y retiene estructuras locales de los datos
- Mejor desempeño que otros algoritmos de embedding
- No funciona bien si la dimensionalidad intrínseca del manifold es grande
- Función de costo no convexa y complejidad cuadrática (barnes-hut)
- No es inductivo