Aprendizaje No Supervisado

INF-396

Prof: Juan G. Pavez S.

Aprendizaje no supervisado

- ***Aprendizaje Supervisado:***
  En el enfoque supervisado aprendemos una distribución **p(y|x)** con ejemplos de la forma (x_i, y_i). Es llamado supervisado porque tenemos la supervisión de y.

- ***Aprendizaje no Supervisado*:**
  En el aprendizaje no supervisado no tenemos supervisión de y, sin embargo las tareas pueden ser muy diferentes:
  - **Clustering**: Es como la clasificación pero sin supervisión.
  - **Dimensionality reduction**: Estamos tratando de aprender una representación de menos dimensiones de **x.**
  - **Feature learning**: Estamos tratando de aprender características que representen datos muy complejos (por ejemplo imágenes).
  - ….

Aprendizaje no supervisado

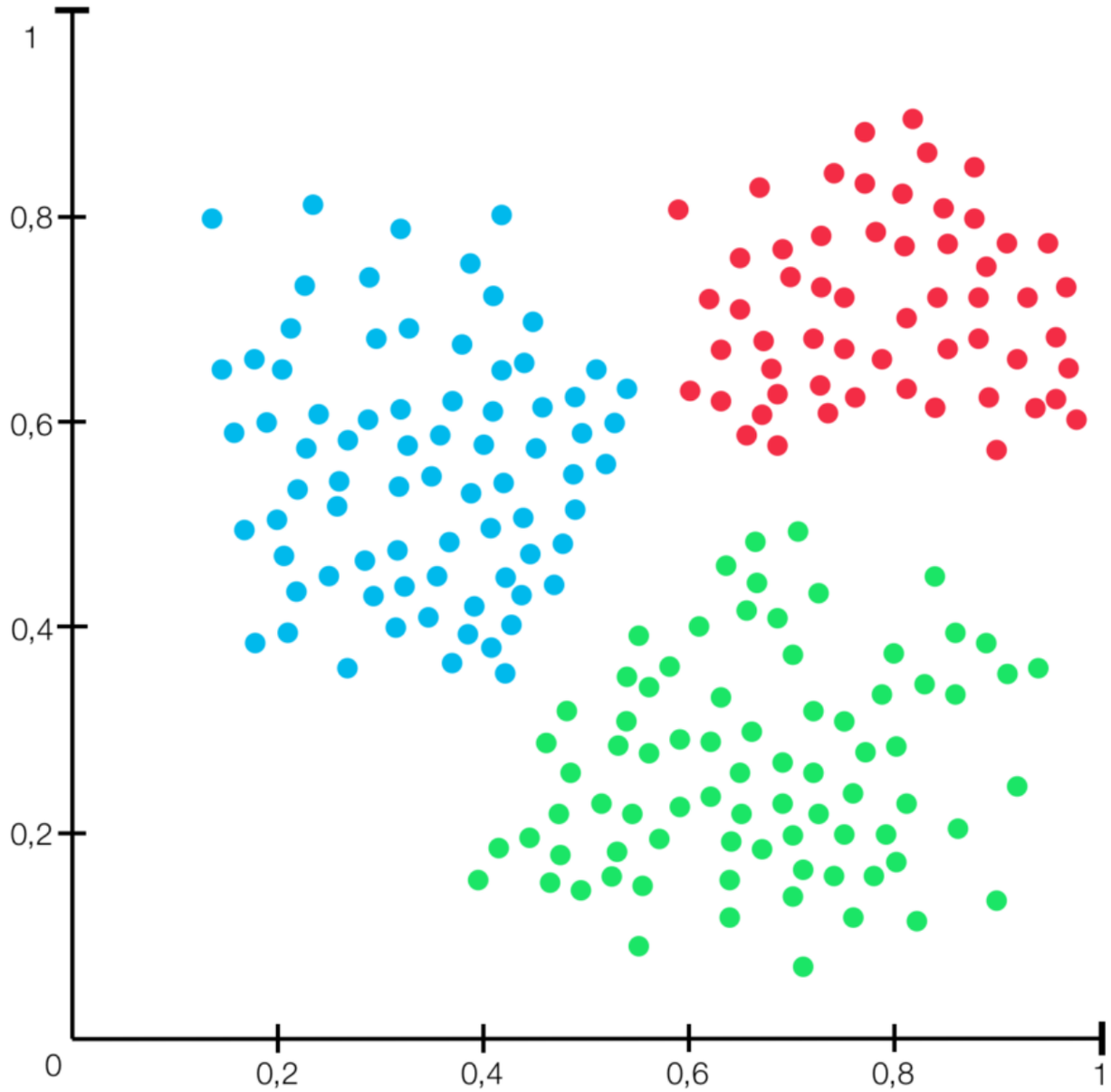
- ***Aprendizaje Supervisado:***
  En el enfoque supervisado aprendemos una distribución **p(y|x)** con ejemplos de la forma (x_i, y_i). Es llamado supervisado porque tenemos la supervisión de y.

- ***Aprendizaje no Supervisado*:**
  En el aprendizaje no supervisado no tenemos supervisión de y, sin embargo las tareas pueden ser muy diferentes:
  - **Clustering**: Es como la clasificación pero sin supervisión.
  - **Dimensionality reduction**: Estamos tratando de aprender una representación de menos dimensiones de **x**.
  - **Feature learning**: Estamos tratando de aprender características que representen datos muy complejos (por ejemplo imágenes).
  - ….

- Veremos que todas estas tareas se pueden ver desde una perspectiva unificada utilizando la teoría de probabilidades.

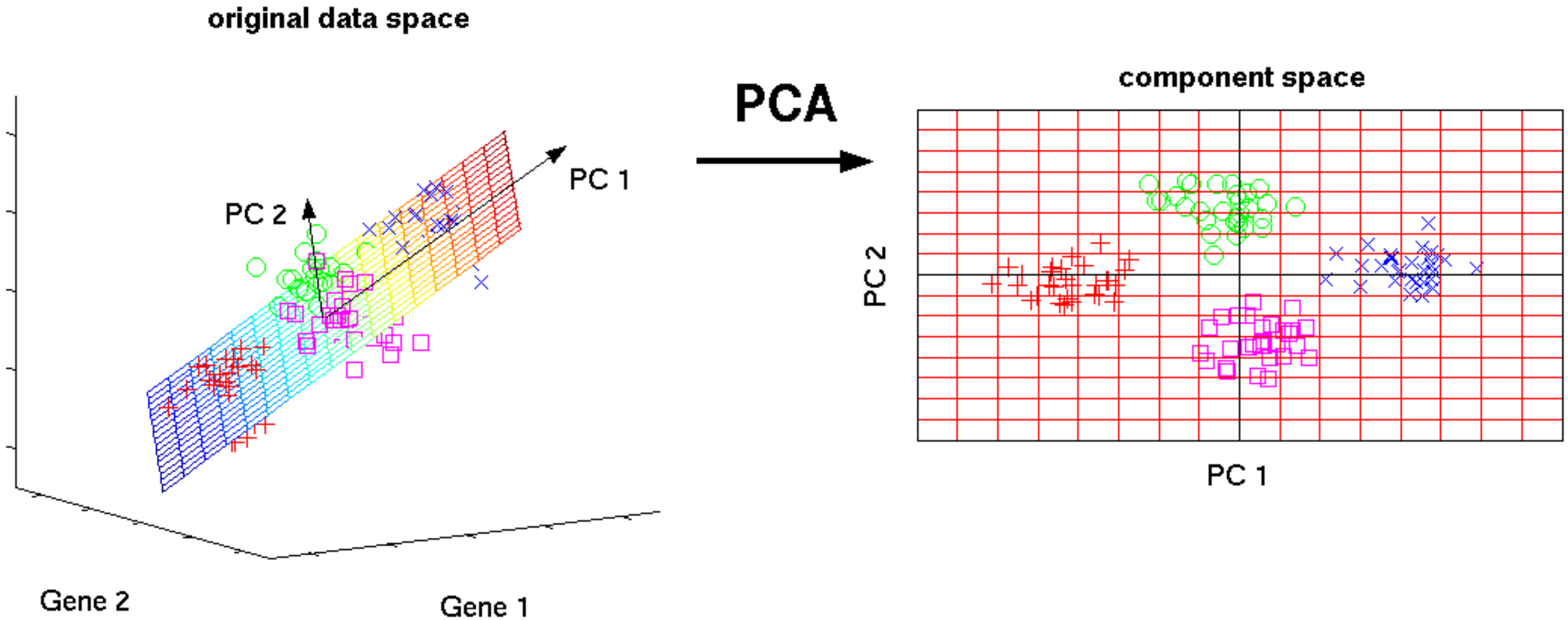
Aprendizaje no supervisado

- Clustering



Aprendizaje no supervisado

- Dimensionality reduction



Aprendizaje no supervisado

- Data generation (learning latent space)

https://www.youtube.com/watch?v=djsEKYuiRFE

Latent Variable Models

- Asumamos que hay L variables aleatorias latentes (no observadas) $(z_{i1}, \ldots z_{iL})$ y D variables aleatorias observadas $(x_{i1}, \ldots x_{iD})$

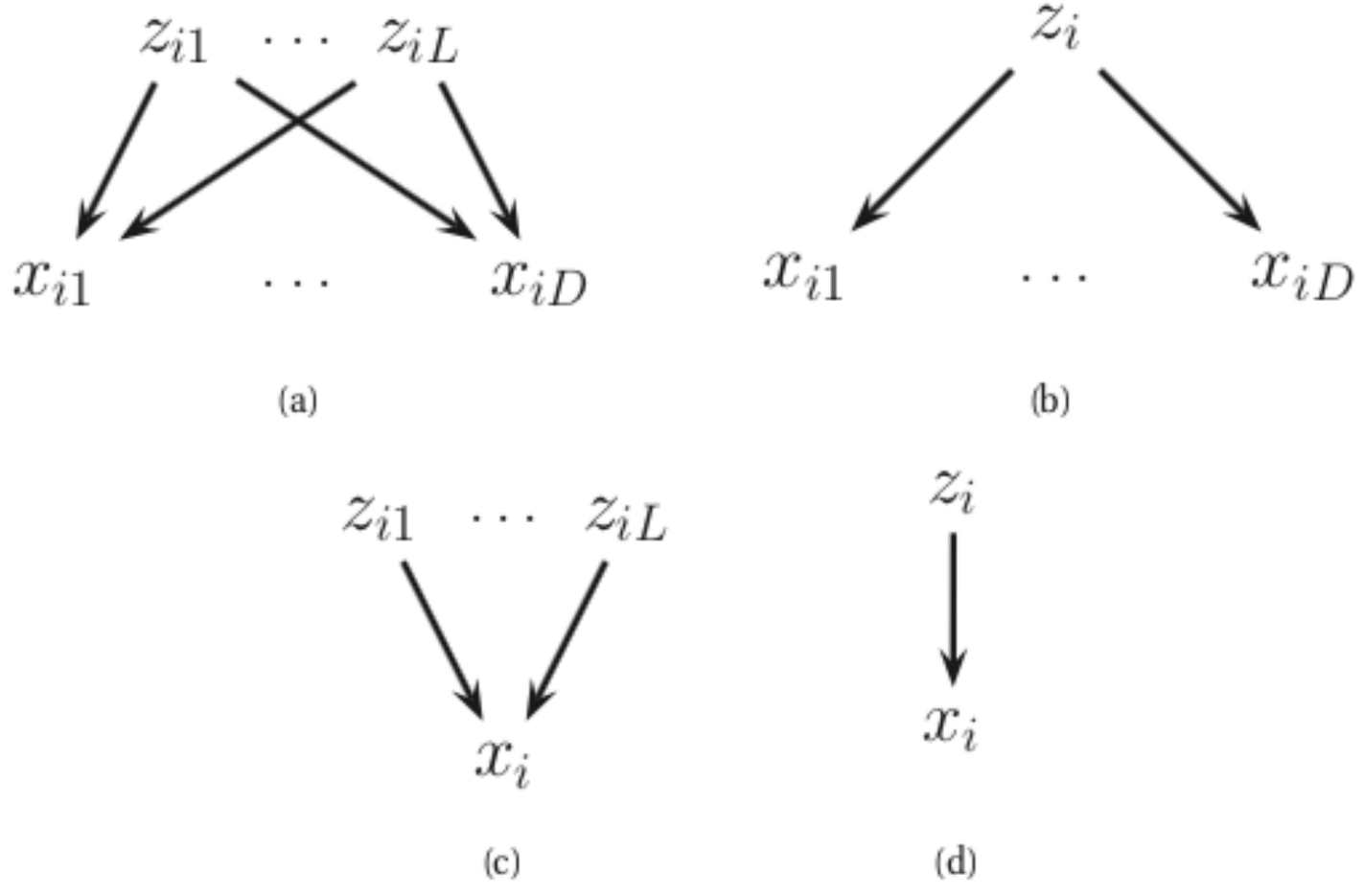


Image from Machine Learning a Probabilistic Perspectiv K. Murphy

PCA

- **Teorema:**
  Queremos encontrar un conjunto de L vectores $w_j \in R^D$ que formen una base (linealmente independientes) y un conjunto de pesos $z_i \in R^L$ tal que

$$J(W,Z) = \frac{1}{N}\sum_{i=1}^{N} ||x_i - \hat{x}_i||^2$$

Donde $\hat{x}_i = Wz_i$ y W es ortonormal (vectores columnas son ortogonales a todos los otros y su norma es 1), equivalentemente

$$J(W,Z) = ||X - WZ^T||_F$$

PCA

- **Teorema:**
  Queremos encontrar un conjunto de L vectores $w_j \in R^D$ que formen una base (linealmente independientes) y un conjunto de pesos $z_i \in R^L$ tal que

PCA

- **Teorema:**
  Queremos encontrar un conjunto de L vectores $w_j \in R^D$ que formen una base (linealmente independientes) y un conjunto de pesos $z_i \in R^L$ tal que

$$J(W, Z) = \frac{1}{N} \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2$$

- **Solución**:
  La solución óptima es $\hat{W} = V_L$ donde V_L los auto vectores más grandes (correspondientes a los autovalores más grandes) de la matriz de covarianza empírica, que asumiendo que tenemos vectores centrados es:

PCA

- **Teorema:**
  La solución óptima es $\hat{W} = V_L$ donde V_L los auto vectores más grandes (correspondientes a los autovalores más grandes) de la matriz de covarianza empírica, que asumiendo que tenemos vectores centrados es:

$$\Sigma = \begin{bmatrix} V(x_1) \dots & COV(x_1, x_D) \\ \vdots & \ddots \\ COV(x_D, x_1) & V(x_D) \end{bmatrix}$$

$$= \begin{bmatrix} \sum_i x_{i1}x_{i1} \dots & \sum_i x_{i1}x_{iD} \\ \vdots & \ddots \\ \sum_i x_{iD}x_{i1} & \sum_i x_{iD}x_{iD} \end{bmatrix}$$

$$= \sum_{i=1}^{N} x_i x_i^T$$

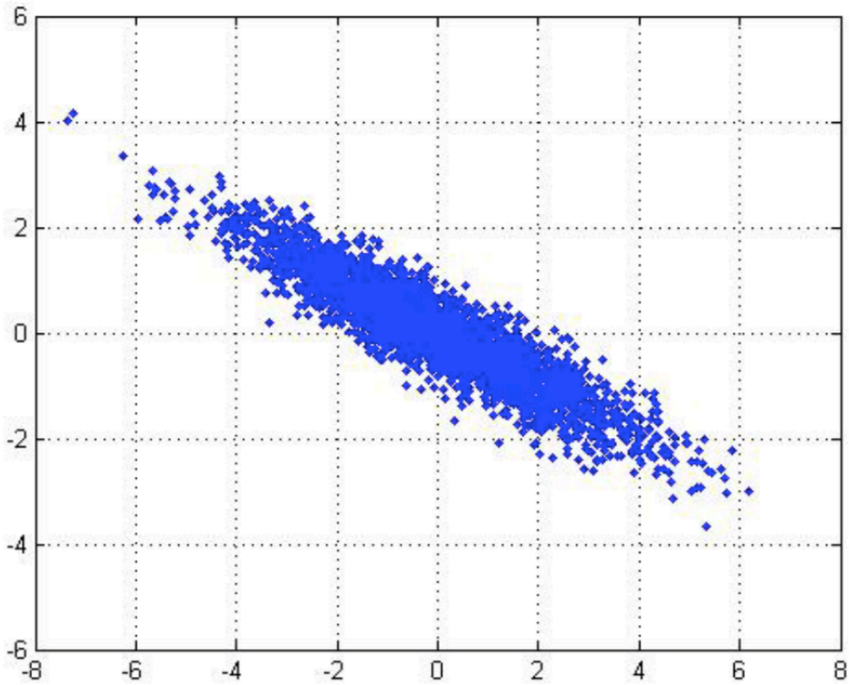
PCA

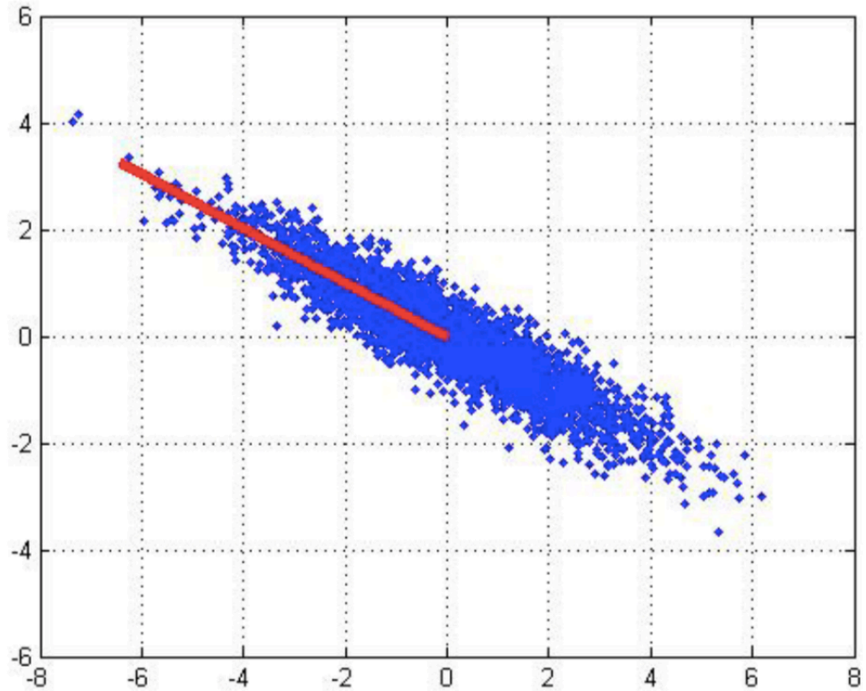
- **Theorem:**
  Queremos encontrar un conjunto de L vectores $w_j \in R^D$ que formen una base (linealmente independientes) y un conjunto de pesos $z_i \in R^L$ tal que

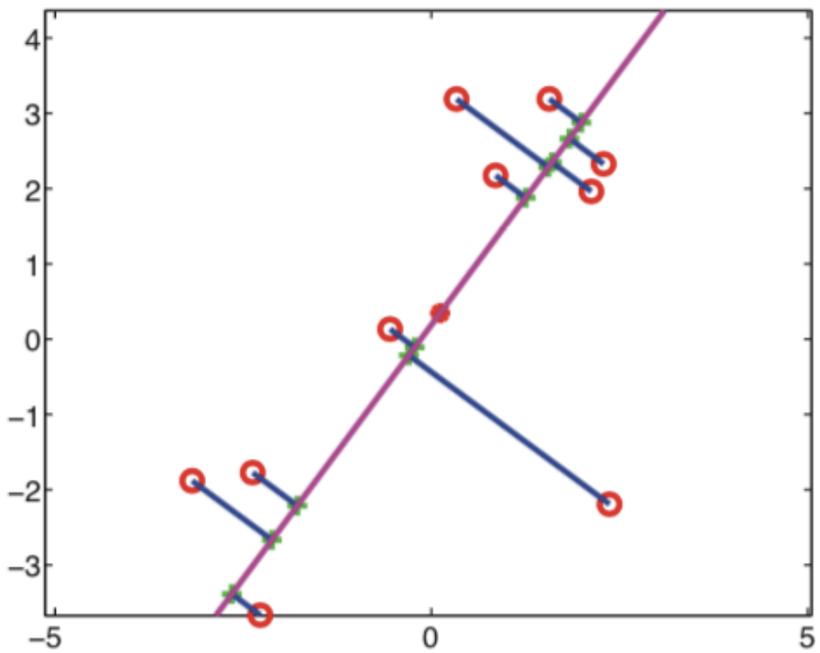
$$J(W, Z) = \frac{1}{N} \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2$$

- **Solution**:
  La codificación de baja dimensión óptima es $\hat{z}_i = W^T x_i$

PCA







PCA

- **Proof:**
  Comencemos por encontrar la solución para la primera dimensión $w_1 \in R^D, x_i \in R^D$

$$J(w_1, z_1) = \frac{1}{N} \sum_{i=1}^{N} ||x_i - z_{i1} w_1||^2$$

PCA

- **Proof:**
  Comencemos por encontrar la solución para la primera dimensión $w_1 \in R^D, x_i \in R^D$

PCA

- **Proof:**

- Ahora $Var[\hat{z}_1] = w_i^T \hat{\Sigma} w_1 = \lambda_1$ Así que la podemos maximizar, cumpliendo la restricción anterior, tomando el valor propio de mayor valor y vector propio correspondiente.

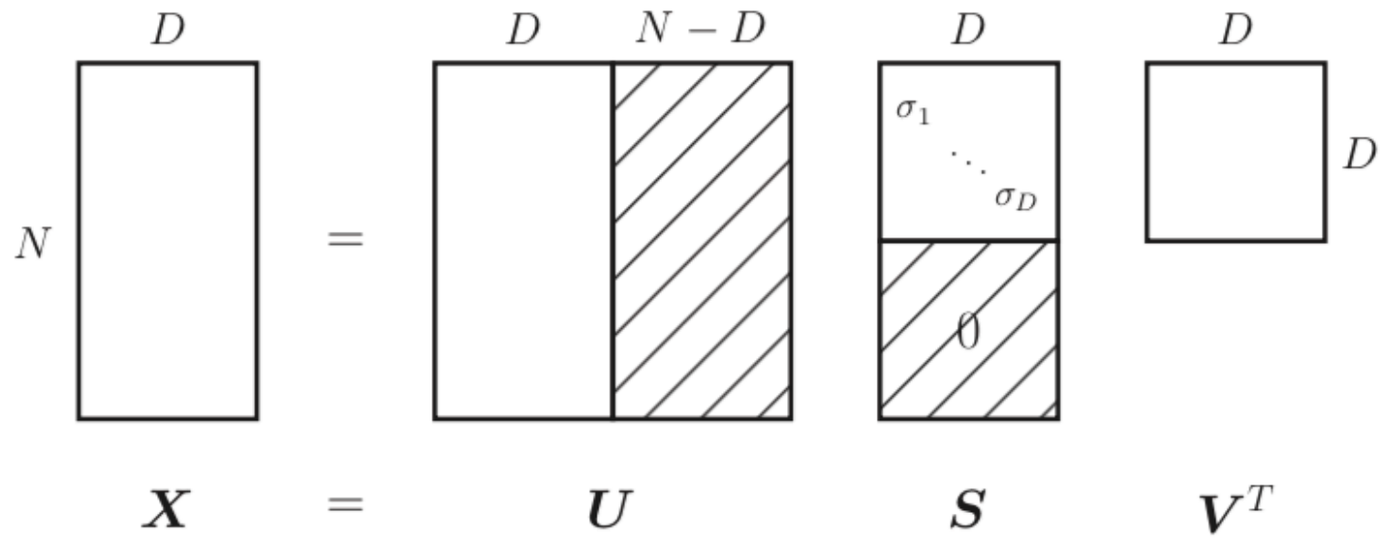
- Siguiendo el mismo análisis podemos concluir que la segunda dirección de proyección es el segundo vector propio y así con los otros también.

SVD

- Para cualquier matriz NxD real X:

$$X = U\ S\ V^T$$

NxD NxN NxD DxD

- 

(a)

SVD

- Para cualquier matriz real NxD X:

$$X = U\ S\ V^T$$

NxD NxN NxD DxD

- Donde

$$U^T U = I_N$$

$$V^T V = V V^T = I_D$$

- y S es una matris diagonal con r = min(N,D) valores singulares $\sigma \geq 0$

SVD

- Para cualquier matriz real X

  $X^T X = VS^T U^T USV^T = V(S^T S)V^T = VDV^T$

  Con $D = S^2$

  $(X^T X)V = VD$

  Entonces los vectores propios de $X^T X = \hat{\Sigma}$ son iguales a V (vectores singulares derechos) y los valores propios iguales a D.

  Y también

  $(XX^T)U = UD$

  Entonces los vectores propios de XX^T son iguales a U (vectores singulares izquierdos) y los valores propios iguales a D.

SVD

- Considerar la descomposición SVD $X = USV^T$

  Para conectar al PCA sabemos que
  $\hat{W} = V$ and $\hat{Z} = X\hat{W}$
  Entonces

$$\hat{Z} = USV^TV = US$$

Y dado que $\hat{X} = Z\hat{W}^T$ la reconstrucción óptima es

$$\hat{Z} = X\hat{W}$$

El SVD truncado es

$$X = U_{:,1:L}S_{1:L,1:L}V^T_{1,1:L}$$

PCA

```python
def pca(X):
  # Data matrix X, assumes 0-centered
  n, m = X.shape
  assert np.allclose(X.mean(axis=0), np.zeros(m))
  # Compute covariance matrix
  C = np.dot(X.T, X) / (n-1)
  # Eigen decomposition
  eigen_vals, eigen_vecs = np.linalg.eig(C)
  # Project X onto PC space
  X_pca = np.dot(X, eigen_vecs)
  return X_pca
```

pca_eigen_docomposition_np.py hosted with ❤️ by GitHub view raw

SVD

- Dataset labeled faces in the wild: 1348 imágenes de tamaño 62x47 (>3000 features).

- Obtén 150 componentes con PCA (150 matrices 62x47) y luego graficandolos

SVD

- Dateset labeled faces in the wild: 1348 imágenes de tamaño 62x47 (>3000 features).

- Obtén los datos proyectados 1348x150 luego usa los componentes para reconstruir las imágenes.

https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

UNSUPERVISED DEEP LEARNING

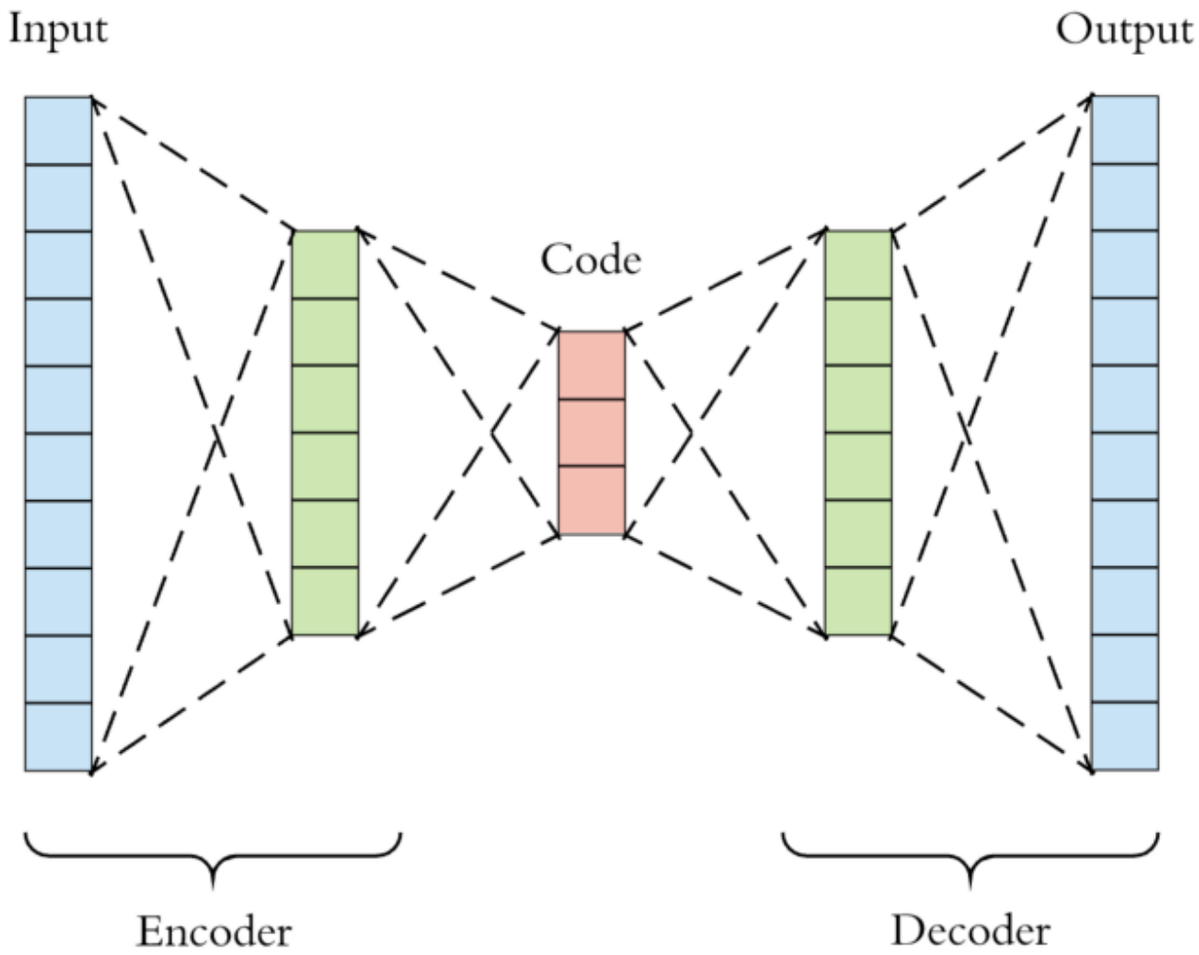
- Muchas de las ideas que vimos para aprendizaje no supervisado podemos traspasarla a redes neuronales.

- Reemplazando redes neuronales por las operaciones ‘simples’ de los métodos vistos obtenemos mayor expresividad.

- Por ejemplo **PCA con redes neuronales: Autoencoders**

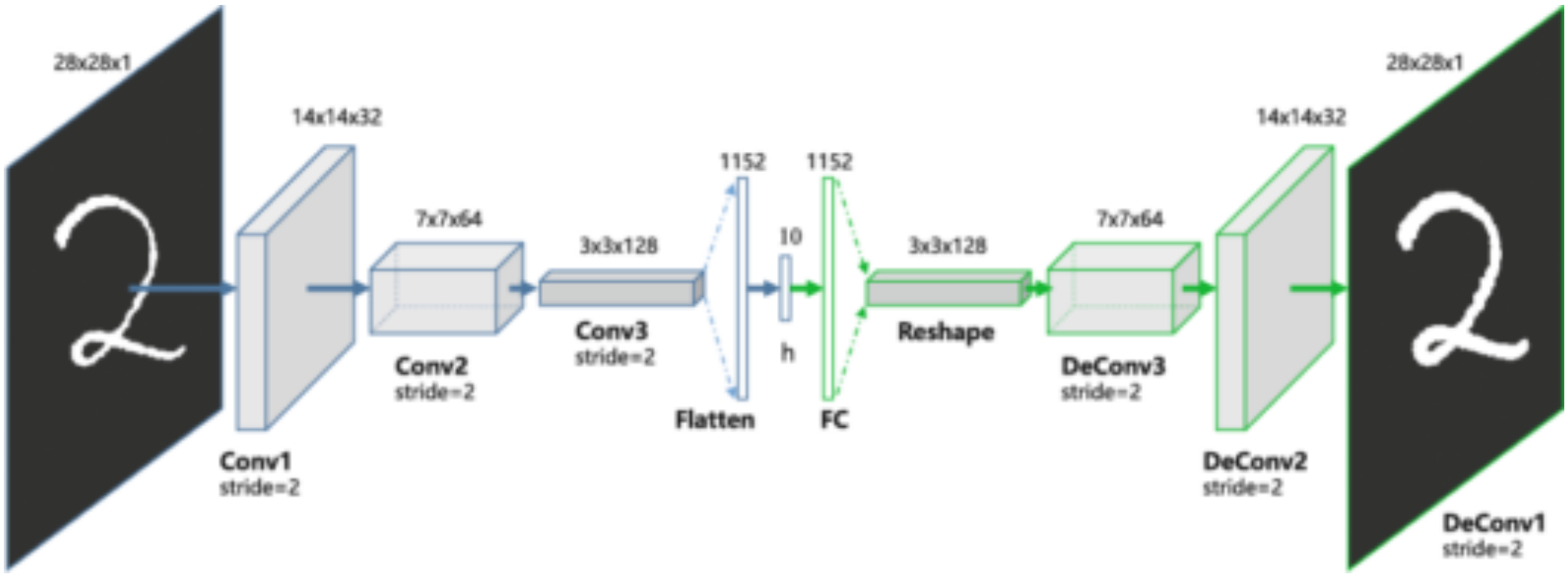
- Definiendo una sola capa lineal en el encoder y una lineal en el decoder obtenemos el PCA.

UNSUPERVISED DEEP LEARNING



- $J(W,Z) = \frac{1}{N}\sum_{i=1}^{N} ||x_i - f_W(x_i)||^2 = \frac{1}{N}\sum_{i=1}^{N} ||x_i - (f_Z \circ f_V)(x_i)||^2$

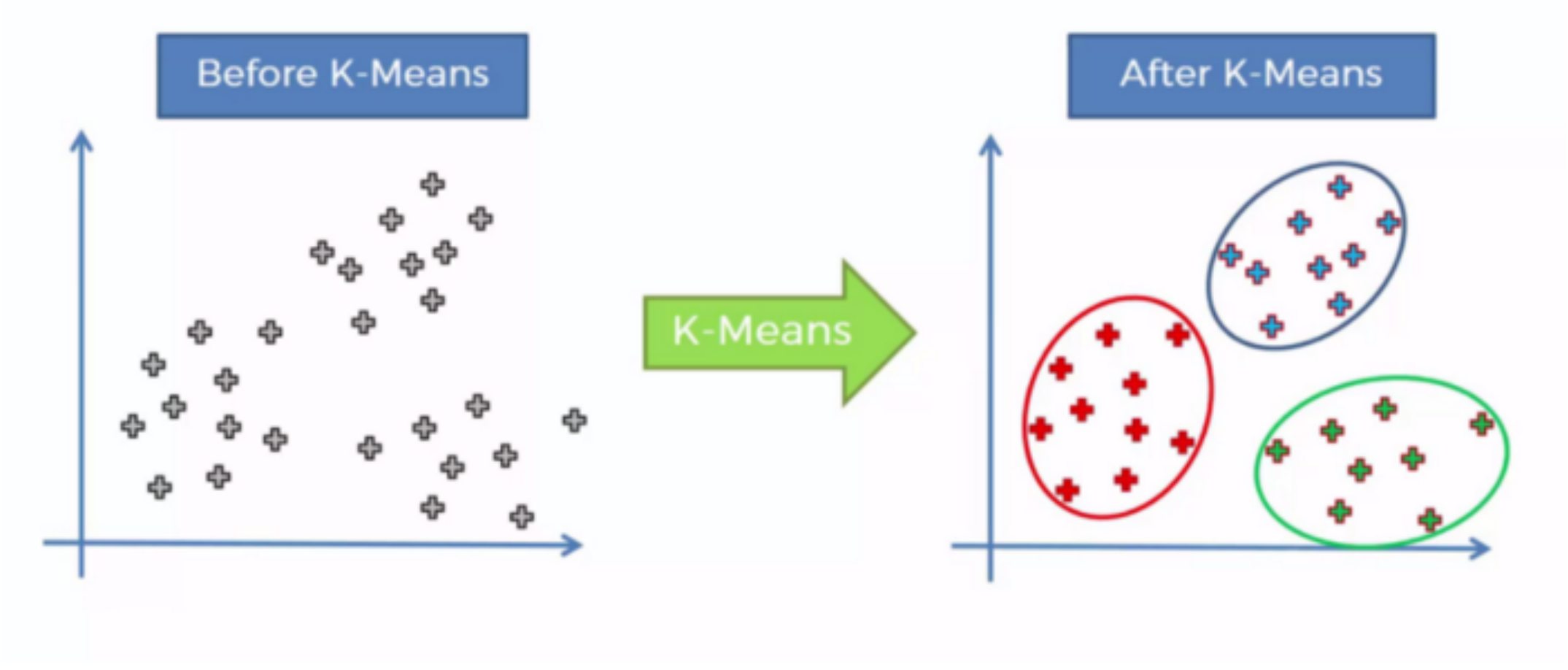
UNSUPERVISED DEEP LEARNING



- $$J(W,Z) = \frac{1}{N}\sum_{i=1}^{N} ||x_i - f_W(x_i)||^2 = \frac{1}{N}\sum_{i=1}^{N} ||x_i - (f_Z \circ f_V)(x_i)||^2$$

The EM algorithm

- ***K-means***



The EM algorithm

- ***K-means***

Algorithm 11.1: K-means algorithm

initialize $\mathbf{m}_k$;
repeat
  Assign each data point to its closest cluster center: $z_i = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$;
  Update each cluster center by computing the mean of all points assigned to it: $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$;
until *converged*;

Image from Machine Learning a Probabilistic Perspectiv K. Murphy

The EM algorithm

- ***K-means***

Algorithm 11.1: K-means algorithm

1 *initialize* $\mathbf{m}_k$;
2 **repeat**
3 | Assign each data point to its closest cluster center: $z_i = \arg\min_k ||\mathbf{x}_i - \boldsymbol{\mu}_k||_2^2$;
4 | Update each cluster center by computing the mean of all points assigned to it: $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i:z_i=k} \mathbf{x}_i$;
5 **until** *converged*;

https://stanford.edu/class/engr108/visualizations/kmeans/kmeans.html

Image from Machine Learning a Probabilistic Perspectiv K. Murphy

The EM algorithm

- **Choosing K**

  Aun tenemos que elegir K.
  Dado que K-means no es probabilistico, no se puede calcular la verosimilitud.
  Uno comúnmente gráfica el error de reconstrucción de los puntos definidos como

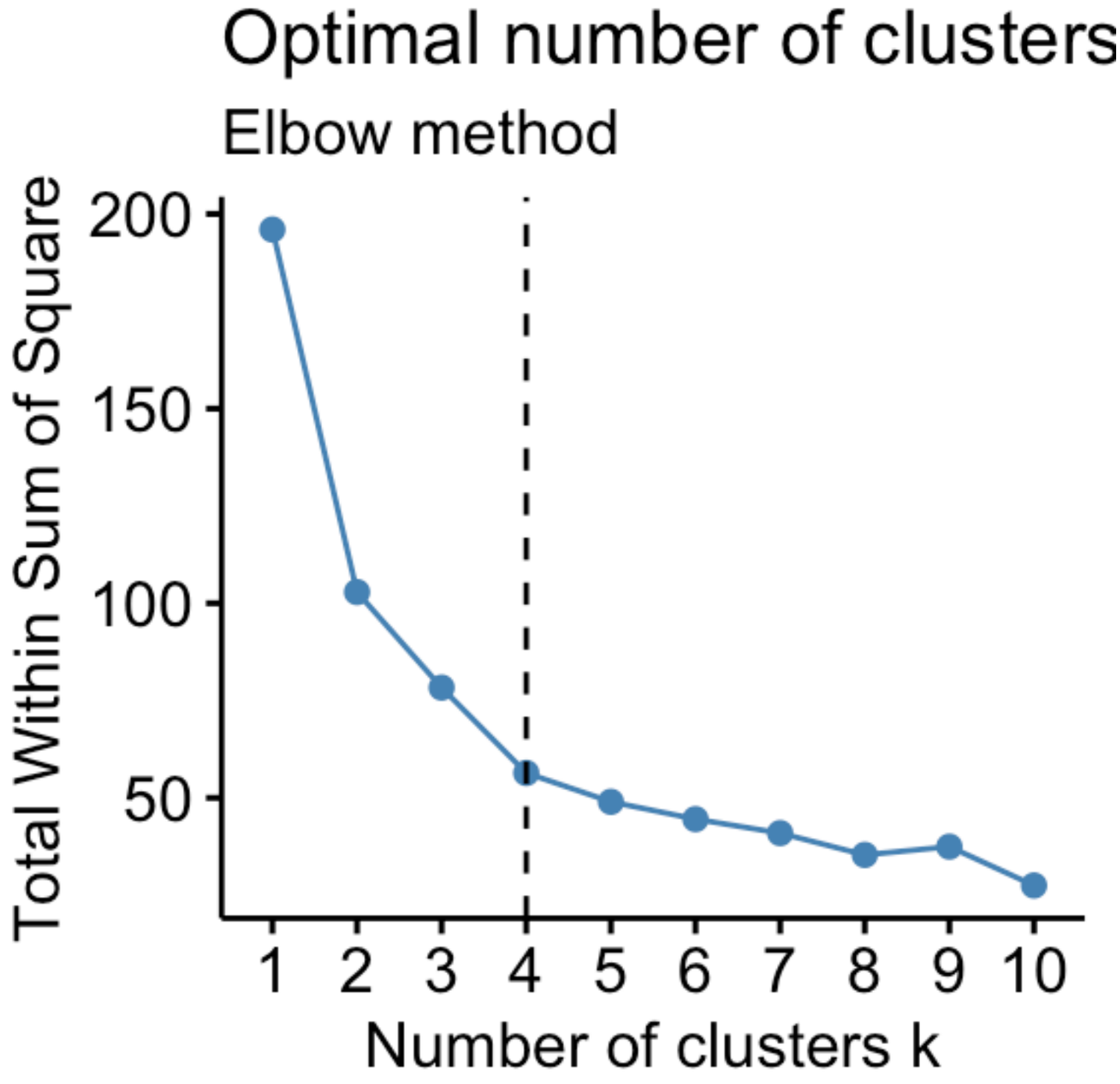
$$E(D,K) = \frac{1}{D}\sum_{i \in D} ||x_i - \hat{x}_i||^2$$

Con $\hat{x}_i = \mu_{z_i}$ $z_i = argmin_k ||x_i - \mu_k||_2^2$

The EM algorithm

- **Choosing K**

  Le idea es que
  - E(K,D) >> E(K*,D) para K < K*
  - E(K,D) < E(K*,D) por poco para K > K*



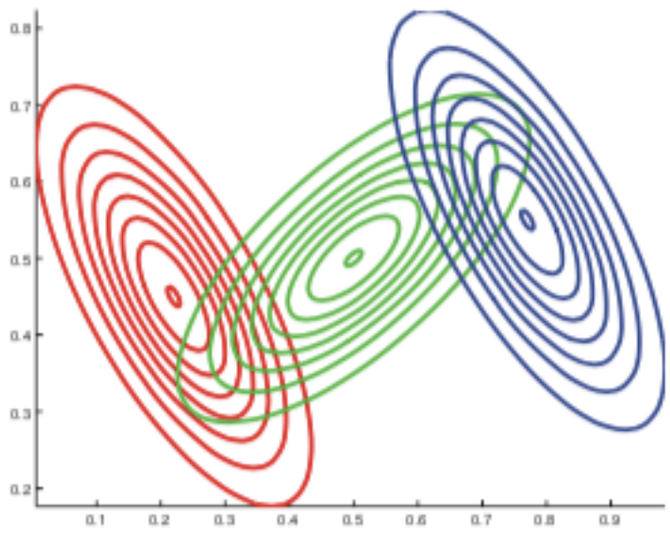
Latent Variable Models

- ***Modelos de Mezcla***

  - Asumamos $z_i \in \{1,...K\}$ y $p(z_i) = Cat(\pi)$

  - También $p(x_i \mid z_i = k) = p_k(x_i)$ así que tenemos K distribuciones base

- $p(x_i \mid \theta) = \sum_{k=1}^{K} \pi_k p_k(x_i \mid \theta)$



(a)

(b)

Image from Machine Learning a Probabilistic Perspectiv K. Murphy

Latent Variable Models

- ***Mixture of Gaussians***

  - $p(x_i \mid \theta) = \sum_{k=1}^{K} \pi_k N(x_i \mid \mu_k, \Sigma_k)$

Podemos también definir otro modelo, usando por ejemplo una función de densidad condicional de clases Bernoulli y podemos representar vectores binarios

$$p(x_i \mid z_i = k, \theta) = \prod_{j=1}^{D} Ber(x_{ij} \mid \mu_{jk})$$

Image from Machine Learning a Probabilistic Perspectiv K. Murphy

Latent Variable Models

- ***Mixture of Gaussians***

  - $p(x_i \mid \theta) = \sum_{k=1}^{K} \pi_k N(x_i \mid \mu_k, \Sigma_k)$

Para clusterizar los datos, podemos ajustar (maximizando la verosimilitud) nuestro modelo de mezclas y luego calcular la 'responsabilidad'

$$r_{ik} = p(z_i = k \mid x_i, \theta) = \frac{p(z_i = k \mid \theta) p(x_i \mid z_i = k, \theta)}{\sum_{k'=1}^{K} p(z_i = k' \mid \theta) p(x_i \mid z_i = k', \theta)}$$

Y podemos definir un 'hard clustering' cómo

$$z_i^* = argmax_k r_{ik}$$

Latent Variable Models

- ***Calcular el MAP no es convexo***
  - La verosimilitud de un modelo de variable latente es, con $D = \{x_1, \ldots, x_N\}$

$$\log p(D \mid \theta) = \sum_i \log p(x_i \mid \theta)$$

- $$= \sum_i \log \left[\sum_{z_i} p(x_i, z_i \mid \theta) \right]$$

  Ahora, los z son desconocidos y no podemos mover el logaritmo dentro de la suma. Si conociésemos z podríamos calcular

$$l_c(\theta) = \sum_{i=1}^{N} \log p(x_i, z_i \mid \theta)$$

Latent Variable Models

- ***EM***

  - Supongamos (sólo por ejercicio) que conocemos los z_i entonces podemos calcular la verosimilitud, cómo

$$l_c(\theta) = \sum_{i=1}^{N} \log p(x_i, z_i; \theta)$$

$$= \sum_{i=1}^{N} \log p(x_i \mid z_i; \mu, \theta) p(z_i \mid \theta)$$

$$= \sum_{i=1}^{N} \sum_{k} \log \left[\frac{1}{(2\pi)^{n/2} \mid \Sigma \mid^2} \exp\Big(-\frac{1}{2}(x_i - u_k)^T \Sigma^{-1} (x_i - u_k) \Big) \right]$$

$$+ \sum_{k} \log \left[\theta^{I(z_i = k)} \right]$$

Latent Variable Models

- ***EM***
  - Obtener los estimadores de máxima verosimilitud para este caso es fácil (ya lo hicimos cuando vimos el modelo de LDA)

$$\mu_j = \frac{\sum_{i=1}^{N} I\{z_i = j\} x_i}{\sum_{i=1}^{N} I\{z_i = j\}} \qquad \theta_j = \frac{1}{N} \sum_{i=1}^{N} I\{z_i = j\}$$

The EM algorithm

- ***EM***

  - Idea: Estimemos $p(z_i \mid x_i; \theta)$ primero, luego usemos esto para estimar los estimadores de máxima verosimilitud, luego recalculemos

$$r_{ij} = p(z_i = j \mid x_i; \theta) = \frac{p(x_i \mid z_i = j; \theta) p(z_i = j; \theta)}{\sum_{l=1}^{l} p(x_i \mid z_i = l; \theta) p(z_i = l; \theta)}$$

- Y luego volvemos a calcular el estimador de máxima verosimilitud para los parámetros de $p(x_i, z_i; \theta)$

The EM algorithm

- ***EM***
  - Se puede demostrar que los parámetros para $p(x_i \mid z_i; \theta)$ se estiman por

- $$\mu_j = \frac{\sum_{i=1}^{N} r_{ij} x_i}{\sum_{i=1}^{N} r_{ij}} \qquad \theta_j = \frac{1}{N} \sum_{i=1}^{N} r_{ij}$$

$$\Sigma_j = \frac{\sum_i r_{ij} (x_i - \mu_j)(x_i - \mu_j)^T}{\sum_i r_{ij}}$$

The EM algorithm

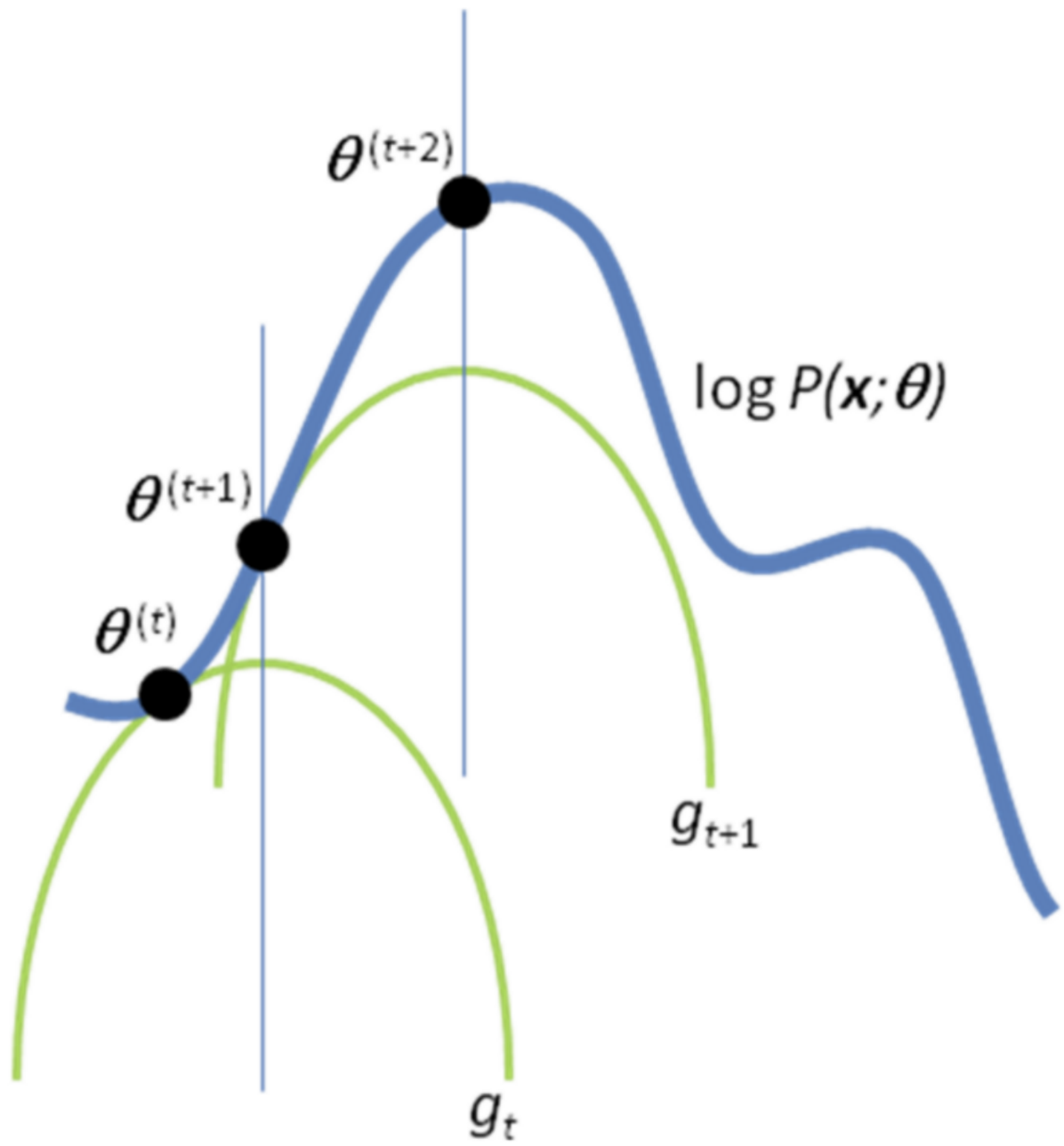
- ***EM***
  Este es de hecho un algoritmo general para aproximar la verosimilitud en modelos de variable latente, llamado expectación-maximizatión.

- Formalmente, lo que haremos es calcular una lower bound para la verosimilitud completa $l_c(\theta)$

- De he hecho lo haremos en dos pasos:
  E-step: Crear una lower bound para $l_c(\theta)$
  M-step: Optimiza la lower bound respecto a los parámetros θ
  Repite.

The EM algorithm

- ***EM***



The EM algorithm

- ***E-Step***
  Comenzaremos por construir una lower bound

$$\log p(x;\theta) = \log \sum_z p(x,z;\theta)$$

$$= \log \sum_z Q(z) \frac{p(x,z;\theta)}{Q(z)}$$

- En esta paso introducimos $Q(z)$ que es una distribución sobre z, es decir suma 1.

The EM algorithm

- ***E-Step***
  Comenzaremos por construir una lower bound

$$\log p(x;\theta) = \log \sum_z p(x,z;\theta)$$

$$= \log \sum Q(z) \frac{p(x,z;\theta)}{Q(z)}$$

- Ahora utilizaremos la desigualdad de **Jensen**, para una función $f(x)$ convexa, es decir $f''(x) < 0$, entonces $E[f(x)] \leq f[E(x)]$

- Para el logaritmo se cumple $f''(x) = -\frac{1}{x^2} < 0$

The EM algorithm

- ***E-Step***
  Comenzaremos por construir una lower bound

$$\log p(x;\theta) = \log \sum_z p(x,z;\theta)$$

$$= \log \sum_z Q(z) \frac{p(x,z;\theta)}{Q(z)}$$

$$= \log E_{z\sim Q}\left[\frac{p(x,z;\theta)}{Q(z)}\right]$$

$$\geq E_{z\sim Q}\left[\log \frac{p(x,z;\theta)}{Q(z)}\right]$$

$$= \sum_z Q(z) \log \frac{p(x,z;\theta)}{Q(z)}$$

The EM algorithm

- ***E-Step***
  Ahora imitando lo que vimos 'gráficamente' queremos que esta lower bound 'toque' $l_c(\theta)$ en el θ actual, para lograr esto lo que hacemos es optimizar respecto a Q, es decir queremos que se cumpla

$$\log p(x;\theta) = E_{z\sim Q}\left[\log \frac{p(x,z;\theta)}{Q(z)}\right]$$

- Uno puede demostrar, (derivado de Jensen) que esto se cumple cuando

$$\frac{p(x,z;\theta)}{Q(z)} = c$$

The EM algorithm

- ***E-Step***

- *Debemos buscar $Q(z)$ para que se cumpla*

$$\frac{p(x, z; \theta)}{Q(z)} = c$$

- Es decir $Q(z) \propto p(x, z; \theta)$

- *Además debe sumar 1, normalizando*

- $$Q(z) = \frac{p(x, z; \theta)}{\sum_z p(x, z; \theta)}$$
  $$= p(z \mid x; \theta)$$

(Verificar reemplazando el la cota)

The EM algorithm

- ***E-Step***

- La cota inferior derivada se conoce como **Evidence Lower Bound**

$$ELBO(x_i; Q_i, \theta) = E_{z_i \sim Q_i}\left[\log \frac{p(x_i, z_i; \theta)}{Q_i(z_i)}\right]$$

$$= \sum_z Q_i(z_i) \log \frac{p(x_i, z_i; \theta)}{Q_i(z_i)}$$

- Que cumple $\log p(D; \theta) \geq \sum_{i=1}^{N} ELBO(x_i; Q_i, \theta)$

- Y $Q_i(z_i) = p(z_i \,|\, x_i; \theta)$

The EM algorithm

- **M-*Step***

- Ahora, calculamos los parámetros θ que optimizan el ELBO (**Evidence Lower Bound)**

$$\theta = argmax_\theta \sum_{i=1}^{N} ELBO(x_i; Q_i, \theta)$$

The EM algorithm

- **EM**

- **E-step:** Calcular

$$Q_i(z_i) = p(z_i \mid x_i; \theta^t) = \frac{p(x_i \mid z_i = j; \theta^t) p(z_i = j; \theta^t)}{\sum_{l=1}^{k} p(x_i \mid z_i = l; \theta^t) p(z_i = l; \theta^t)}$$

- **M-step**

$$\theta^{t+1} = argmax_\theta \sum_{i=1}^{N} ELBO(x_i; Q_i, \theta)$$

Ahora define $\theta^t := \theta^{t+1}$ y repite

The EM algorithm

- ***EM for a GMM***

  E-step: Calcular Q con

$$r_{ik} = \frac{\pi_k p(x_i \mid \theta_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(x_i \mid \theta_{k'}^{(t-1)})}$$

The EM algorithm

- ***EM for a GMM***

  E-step: Calcular Q con

$$r_{ik} = \frac{\pi_k p(x_i \mid \theta_k^{(t-1)})}{\sum_{k'} \pi_{k'} p(x_i \mid \theta_{k'}^{(t-1)})}$$

M-step: Optimiza Q wrt π_k y θ_k

$$Q(\theta, \theta^{t-1}) = \sum_i \sum_{k=1}^{K} r_{ik} \log \pi_k + r_{ik} \log(p(x_i \mid \theta_k))$$

The EM algorithm

- ***EM for a GMM***

  M-step: Optimiza Q wrt π_k y θ_k

$$Q(\theta, \theta^{t-1}) = \sum_i \sum_{k=1}^{K} r_{ik} \log \pi_k + r_{ik} \log(p(x_i \mid \theta_k))$$

  Optimizar para π_k es similar a optimizar para la categórica

$$\pi_k = \frac{1}{N} \sum_k r_{ik}$$

  Optimizar para μ_k y Σ_k es equivalente a optimizar una MVN con pesos.

The EM algorithm

- ***EM for a GMM***

  M-step: Optimiza Q wrt π_k y θ_k

$$Q(\theta, \theta^{t-1}) = \sum_i \sum_{k=1}^{K} r_{ik} \log \pi_k + r_{ik} \log(p(x_i \mid \theta_k))$$

  Optimizar μ_k y Σ_k es equivalente a optimizar una MVN con pesos.

- $$\mu_k = \frac{\sum_i r_{ik} x_i}{\sum_i r_{ik}} \qquad \Sigma_k = \frac{\sum_i r_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i r_{ik}}$$

The EM algorithm

- ***GMM***

(a) (b) (c) (d)

Image from Machine Learning a Probabilistic Perspectiv K. Murphy

The EM algorithm

- ***K-means***

  Si asumimos la simplifación $\Sigma_k = \sigma^2 I_D$ y $\pi_k = 1/K$

  Entonces sólo μ_k la media del cluster debe ser estimada.
  También usaremos una aproximación ‘hard’

$$p(z_i = k | x_i, \theta) = I(k = z_i^*)$$

Dado eso sólo la distancia a la media está siendo cambiado, podemos clasificar cada punto cómo

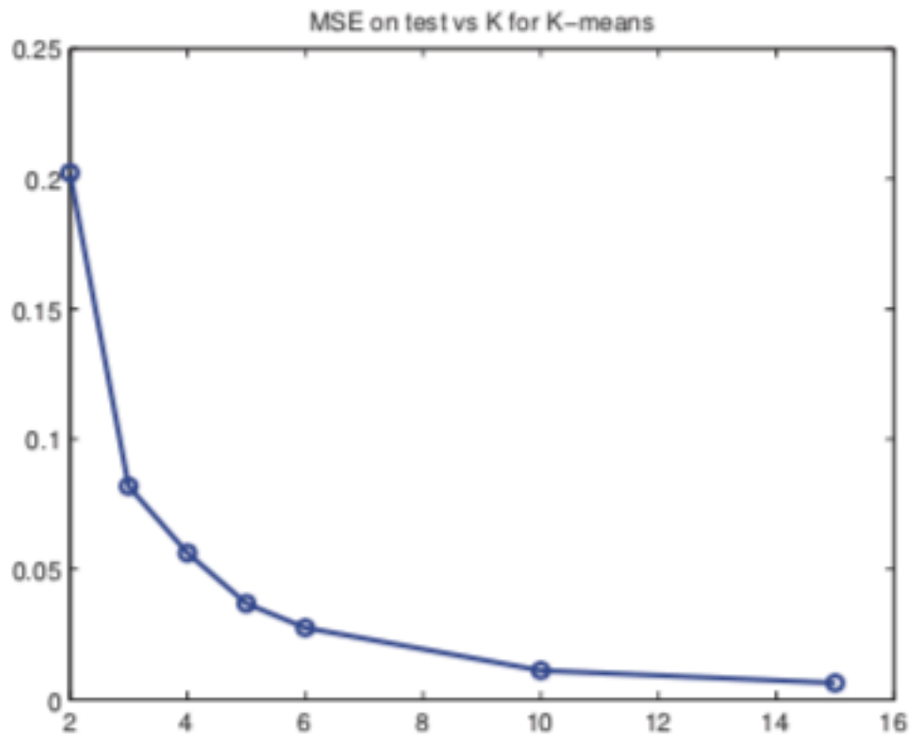
$$z_i^* = argmin_k ||x_i - \mu_k||_2^2$$

En el paso M-step calculamos

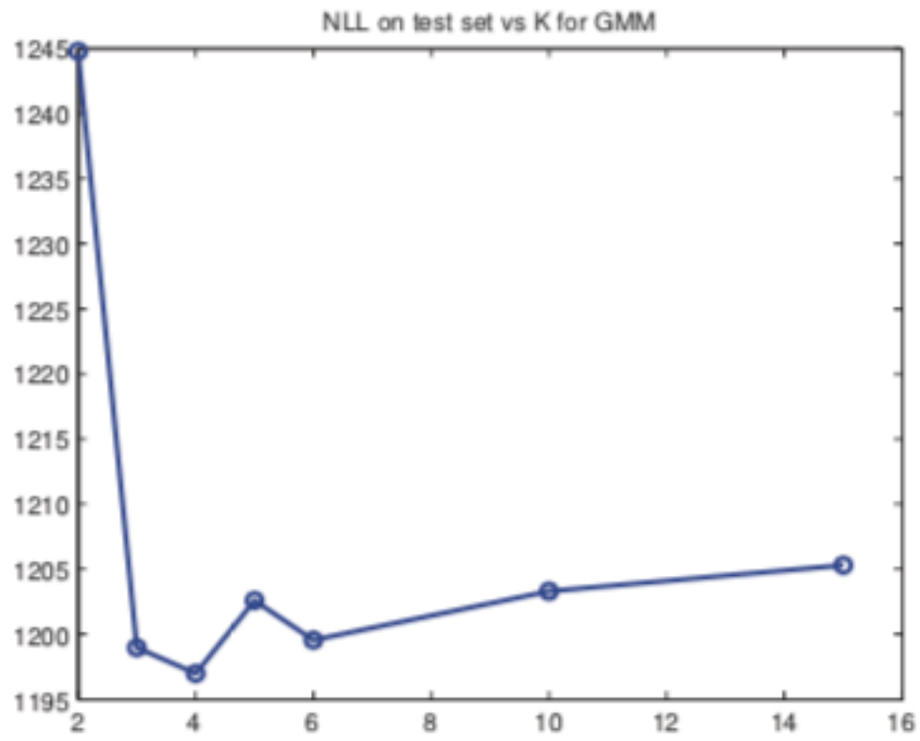
$$\mu_k = \frac{1}{N_k} \sum_{i:z_i=k} x_i$$

The EM algorithm

- **Choosing K**
  A diferencia del K-means usar un modelo probabístico nos permite calcular la verosimilitud negativa del modelo en los datos. A diferencia del error de reconstrucción esta no siempre disminuye.



(a)



(b)

UNSUPERVISED DEEP LEARNING

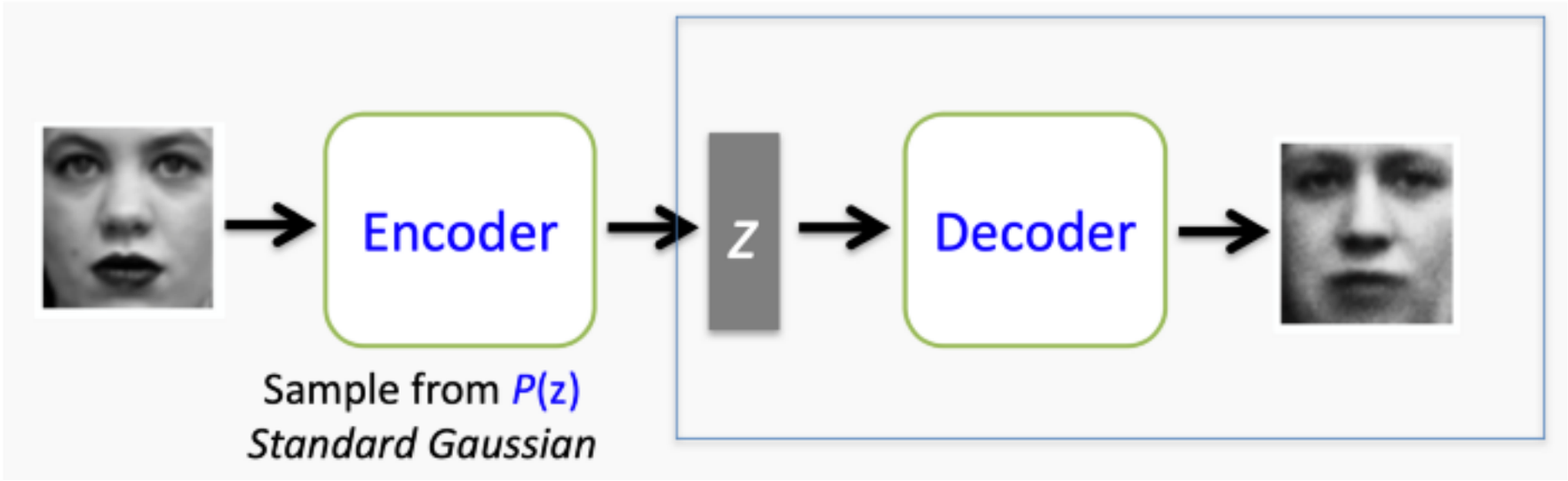
- **Denoising autoencoders:** Añade un poco de ruido a los datos de entrada y reconstruye el original. Fuerza a la red a no aprender el ruido.

- El problema con los autoencoder es que sólo aprenden a comprimir y reconstruir X X->Z->X_rec

- Sería interesante aprender una distribución de z $p(z \mid x)$ de esta manera podríamos generar nuevos datos y generalizar mejor en espacios del espacio latente no observados.

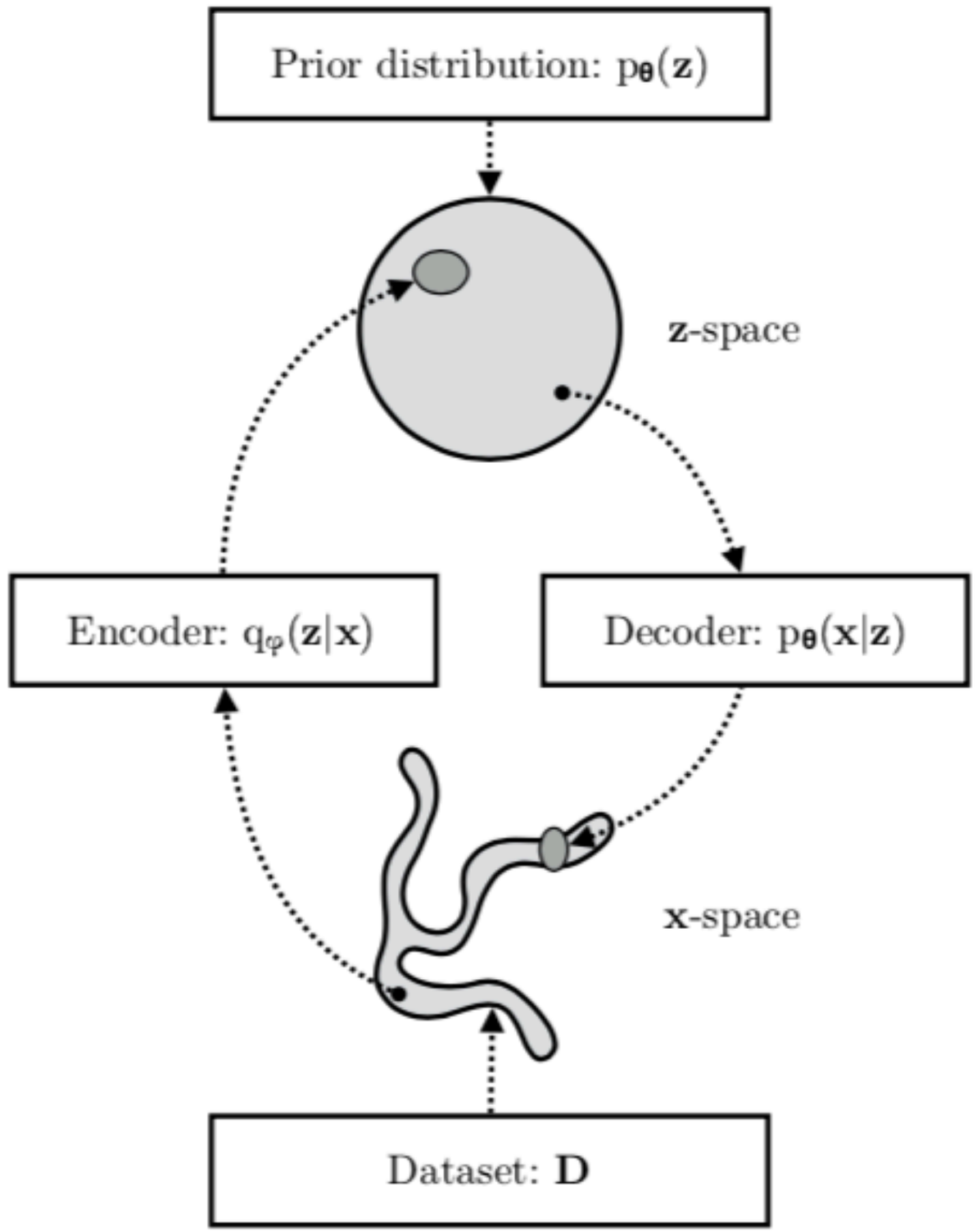
UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).



UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).



An Introduction to Variational Autoencoders. Kingma D., Welling M.

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

  $p_\theta(z)$: no depende de los datos, podemos definirlo nosotros.

  $p_\theta(x, z) = p_\theta(z)p_\theta(x \mid z)$: p(x|z) es representada por red neuronal, la podemos aprender igual cómo hemos aprendido redes anteriormente.

  $p(x) = \int_Z p(x, z)dz$ esta integral no la conocemos, o no es tratable, por lo tanto

  $p(z \mid x) = \frac{p(x, z)}{p(x)}$ tampoco es tratable.

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

- En el variational autoencoder definiremos el modelo $q_\phi(z)$ que aproxima $q_\phi(z\,|\,x) \sim p(z\,|\,x)$
  El truco será optimizar este modelo con una función de pérdida que al optimizar q, empujara p hacia arriba, asi que convertimos el problema en un problema de optimización (muy similar a EM).

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

- En resumen tenemos las siguientes redes neuronales

Decoder: $g(z;\theta)$: $R^k \rightarrow R^d$ que utilizaremos para modelar

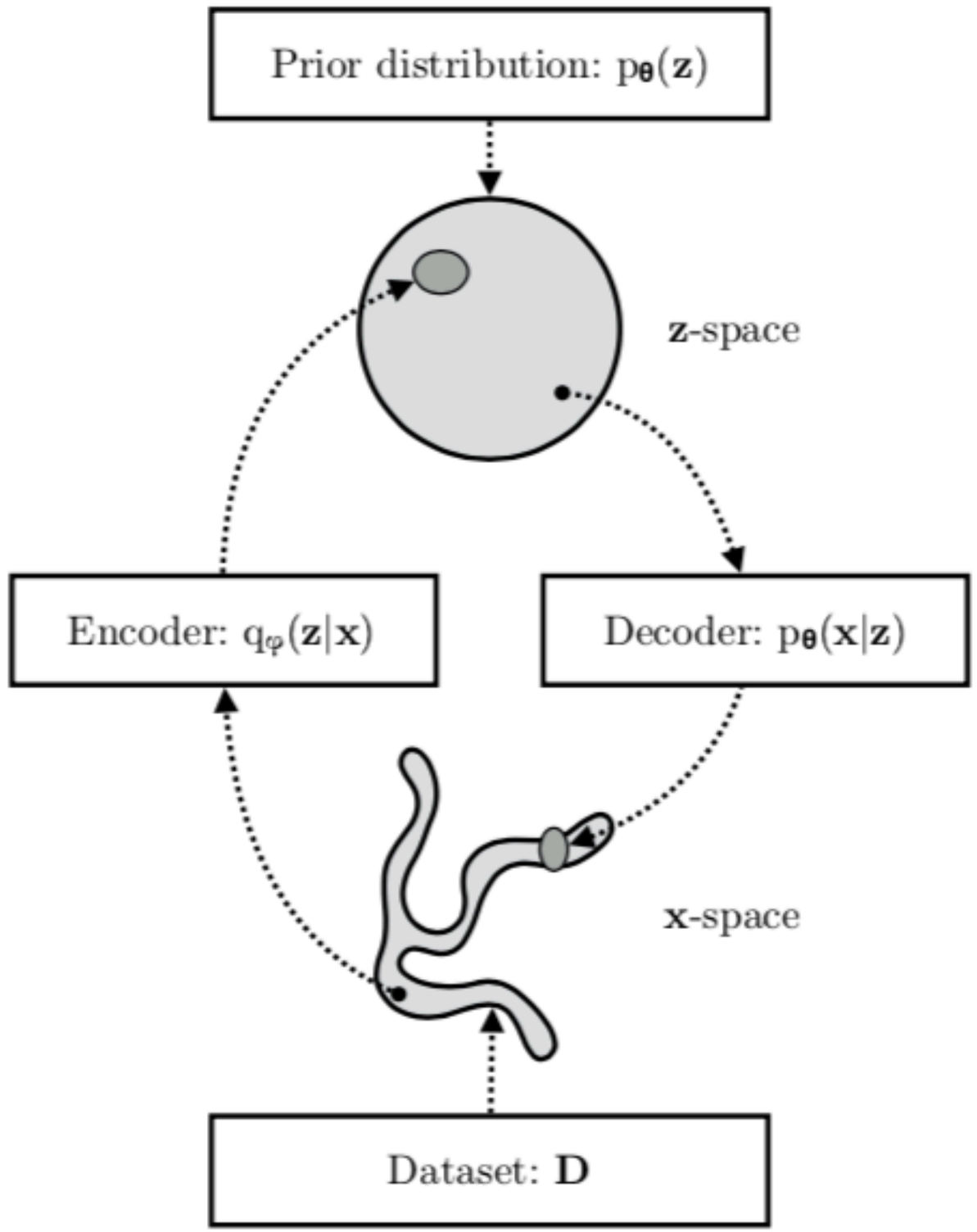
$$p(x \mid z;\theta) = N(g(z;\theta);\sigma^2 I_{d\times d})$$

Encoder: $q(x;\phi)$: $R^d \rightarrow R^k$ y $v(x;\gamma)$ $R^d \rightarrow R^k$ que utilizaremos para modelar

$$Q_i = N(q(x_i;\phi);diag(v(x_i;\gamma)))$$

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).



An Introduction to Variational Autoencoders. Kingma D., Welling M.

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

- Para esto queremos optimizar el ELBO, y esto lo haremos por gradiente descendiente (ascendiente).

$$\theta = argmax_{\theta,\phi,\gamma} \sum_{i=1}^{N} ELBO(x_i; Q_i(\phi, \gamma), \theta)$$

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

- El ELBO a su vez puede ser escrito como

$$ELBO(x; Q, \theta) = \sum_{i=1}^{N} E_{z_i \sim Q_i}\left[\log \frac{p(x_i, z_i; \theta)}{Q_i(z_i)}\right]$$

$$= \sum_{i=1}^{N} \sum_{z} Q_i(z_i) \log \frac{p(x_i, z_i; \theta)}{Q_i(z_i)}$$

- Donde $Q_i = N(q(x_i; \phi); diag(v(x_i; \gamma)))$

- Es decir, para evaluar este ELBO, lo que hacemos es recibir un x, pasarlo por el encoder, obtener Q, samplear de Q varias veces y luego calculamos la suma.

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

- Lo único que nos faltan son los gradientes del ELBO, así lo podemos optimizar, respecto a θ es facilito

$$\nabla_\theta ELBO(x; Q, \theta) = \nabla_\theta \sum_{i=1}^{N} E_{z_i \sim Q_i}\left[\log \frac{p(x_i, z_i; \theta)}{Q_i(z_i)}\right]$$

$$= \nabla_\theta \sum_{i=1}^{N} E_{z_i \sim Q_i}\left[\log p(x_i, z_i; \theta)\right]$$

$$= \sum_{i=1}^{N} E_{z_i \sim Q_i}\left[\nabla_\theta \log p(x_i, z_i; \theta)\right]$$

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

- Para ϕ, γ es otro cuento, ahora estamos derivando Q, que es parte de la definición de la esperanza

$$\nabla_\phi ELBO(x; Q(\phi, \gamma), \theta) = \nabla_\phi \sum_{i=1}^{N} E_{z_i \sim Q_i(\phi,\gamma)} \left[\log \frac{p(x_i, z_i; \theta)}{Q_i(z_i; \phi, \gamma)} \right]$$

Para resolver esto se utiliza el reparametrization trick

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

  Para resolver esto se utiliza el reparametrization trick:
  Reemplazaremos Q por

$$Q_i = N(q(x_i;\phi); diag(v(x_i;\gamma)))$$

- $$= q(x_i;\phi) + diag(v(x_i;\gamma)) \odot N(0,1)$$

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

$$Q_i = N(q(x_i;\phi); diag(v(x_i;\gamma)))$$

- $$= q(x_i;\phi) + diag(v(x_i;\gamma)) \odot N(0, I_{k\times k})$$

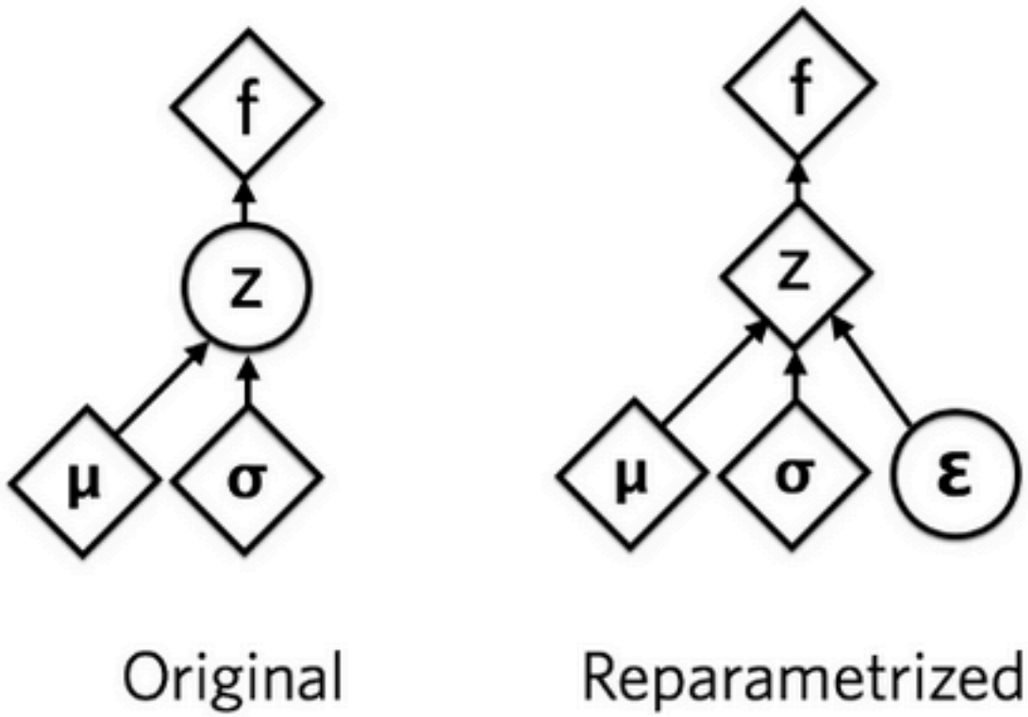
y con esto

$$\nabla_\phi ELBO(x; Q(\phi,\gamma),\theta) = \nabla_\phi \sum_{i=1}^{N} E_{\epsilon_i \sim N(0,I_{k\times k})}\left[\log \frac{p(x_i, z_i;\theta)}{Q_i(z_i;\phi,\gamma)}\right]$$

$$= \sum_{i=1}^{N} E_{\epsilon_i \sim N(0,I_{k\times k})}\left[\nabla_\phi \log \frac{p(x_i, q(x_i;\phi) + diag(v(x_i;\gamma))\epsilon_i;\theta)}{Q_i(q(x_i;\phi) + diag(v(x_i;\gamma))\epsilon_i;\phi,\gamma)}\right]$$

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).

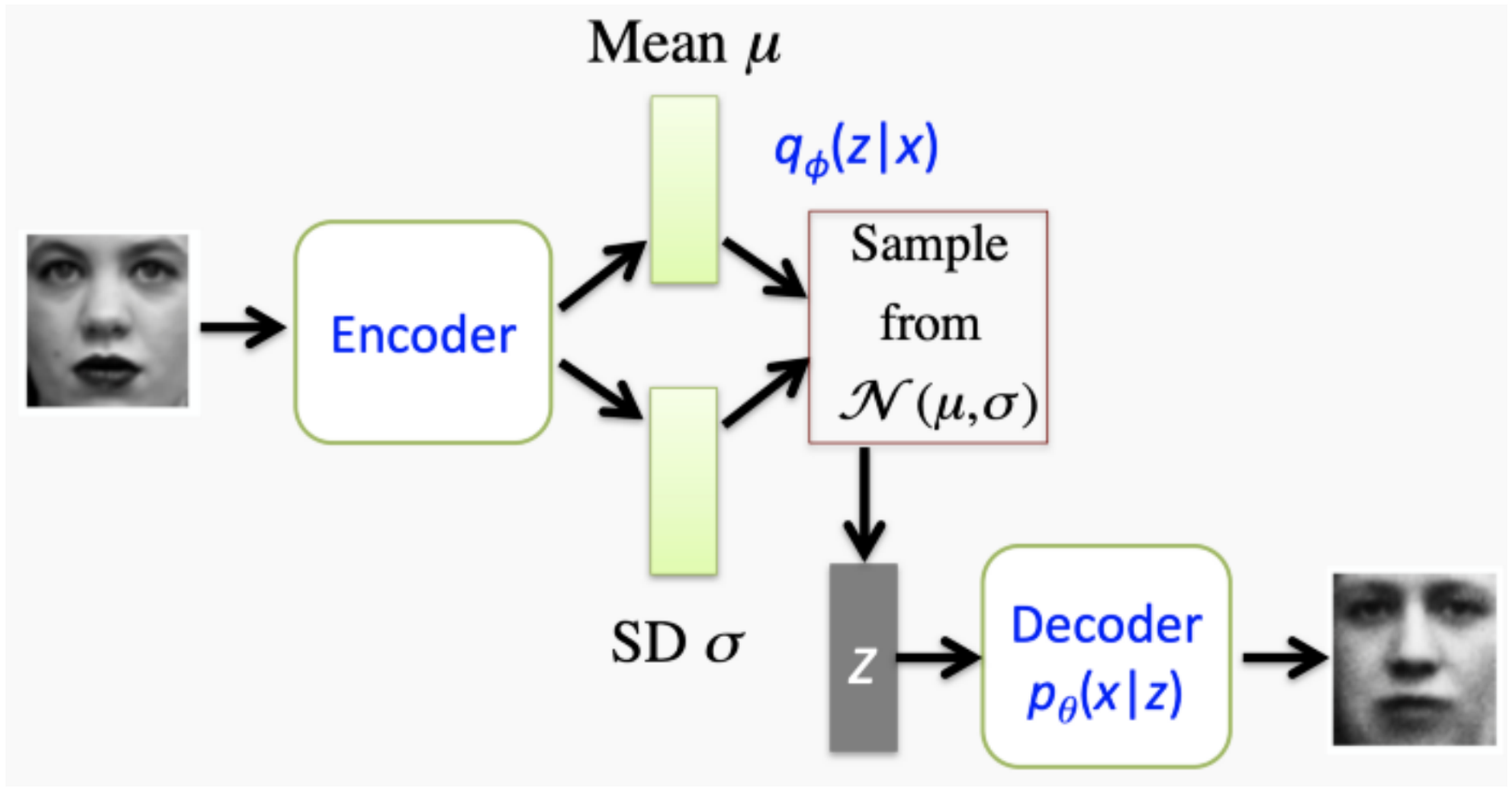
- 

Original Reparametrized

$$\text{Reparametrization Trick}: \; z = \mu + \sigma * \epsilon; \quad \epsilon \sim \mathcal{N}(0,1)$$

UNSUPERVISED DEEP LEARNING

- **Autoencoder Variacionales:** En este caso queremos aprender p(z|x) que no conocemos y p(x|z).



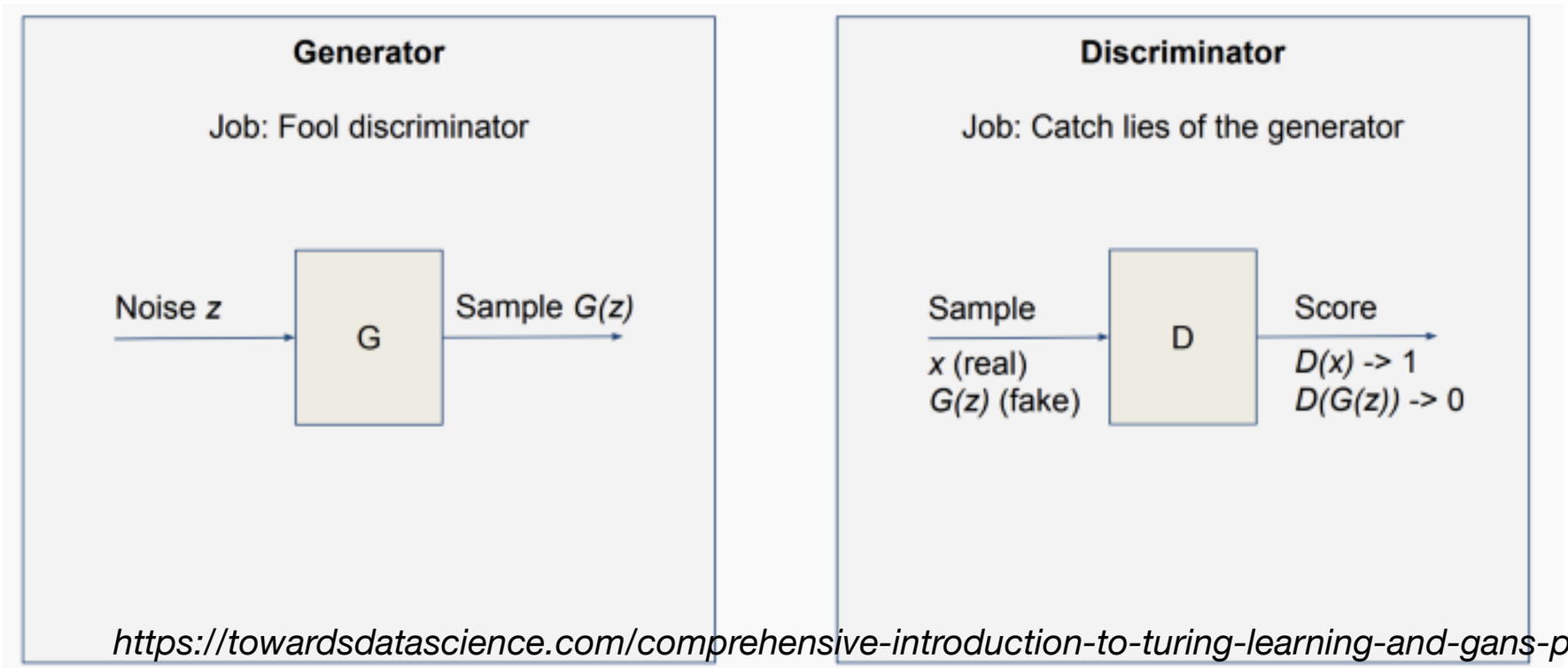
https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

UNSUPERVISED DEEP LEARNING

- **Generative Adversarial Networks:** La idea de este modelo es tener dos modelos G y D, parametrizados como redes neuronales que compiten entre ellos:
  G: Genera datos ‘reales’ desde datos aleatorios
  D: Trata de diferenciar datos reales de los datos ‘reales’

- El truco es que a medida que avanza el aprendizaje D se vuelve mejor en identificar datos falsos, por lo que G se tiene que volver mejor en generar datos falsos.

UNSUPERVISED DEEP LEARNING

- **Generative Adversarial Networks:** La idea de este modelo es tener dos modelos G y D, parametrizados como redes neuronales que compiten entre ellos:
  G: Genera datos ‘reales’ desde datos aleatorios
  D: Trata de diferenciar datos reales de los datos ‘reales’
- 

https://towardsdatascience.com/comprehensive-introduction-to-turing-learning-and-gans-part-1-81f6d02e644d

UNSUPERVISED DEEP LEARNING

- **Generative Adversarial Networks:** La idea de este modelo es tener dos modelos G y D, parametrizados como redes neuronales que compiten entre ellos:
  G: Genera datos ‘reales’ desde datos aleatorios
  D: Trata de diferenciar datos reales de los datos ‘reales’

- **Discriminador quiere predecir 1 en ejemplos reales x** — **Discriminador quiere predecir 0 en ejemplos falsos**

$$V_{\theta^D,\theta^G} = E_{x\sim P_{data}} \log D(x) - E_{z\sim p(z)} \log(1 - D(G(z))$$

Generador quiere que Discriminador prediga 1 en ejemplos ‘reales’

UNSUPERVISED DEEP LEARNING

- **Generative Adversarial Networks:** La idea de este modelo es tener dos modelos G y D, parametrizados como redes neuronales que compiten entre ellos:
  G: Genera datos ‘reales’ desde datos aleatorios
  D: Trata de diferenciar datos reales de los datos ‘reales’
- **Discriminador quiere predecir 1 en ejemplos reales x** **Discriminador quiere predecir 0 en ejemplos falsos**

$$J(\theta^D, \theta^G) = E_{x \sim P_{data}} \log D(x) - E_{z \sim p(z)} \log(1 - D(G(z))$$

Generador quiere que Discriminador prediga 1 en ejemplos ‘reales’

$$\min_{\theta^G} \max_{\theta^D} J(\theta^D, \theta^G)$$

UNSUPERVISED DEEP LEARNING

Generative Adversarial Nets. Goodfellow I., et al.

- **GAN**

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

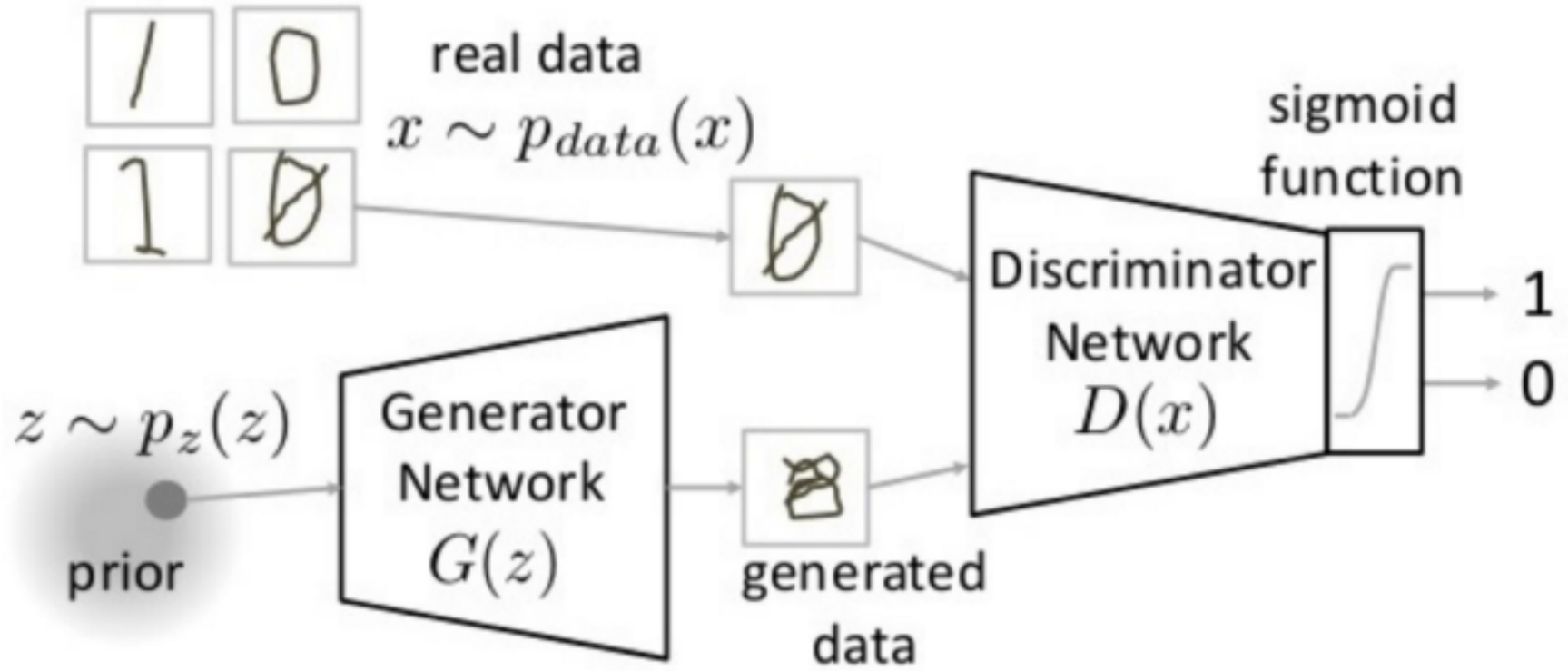
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Proposition 2. *If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion*

$$\mathbb{E}_{\boldsymbol{x} \sim p_{data}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x} \sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))]$$

UNSUPERVISED DEEP LEARNING

- **GAN**

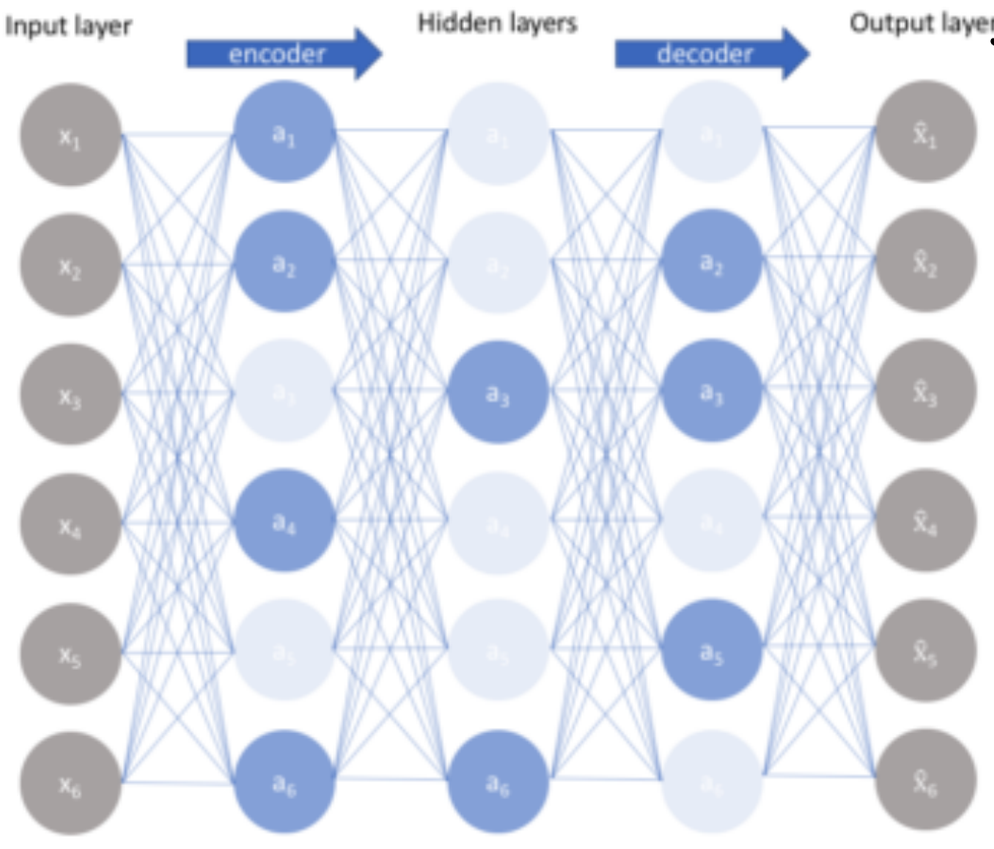


UNSUPERVISED DEEP LEARNING

- **GAN**

UNSUPERVISED DEEP LEARNING

- Sea $\hat{\rho}_j$ la activación de la neurona j, queremos que en promedio las activaciones de una capa sea un número pequeño



$$J*(W,Z) = J(W,Z) + \sum_{j=1}^{s} \rho \log \frac{\rho}{\hat{\rho}} + (1-\rho)\log \frac{(1-\rho)}{(1-\hat{\rho})}$$

$$= \sum_{j=1}^{s} KL(\rho \,||\, \hat{\rho})$$

$$J(W,Z) = \frac{1}{N}\sum_{i=1}^{N} ||\, x_i - f_W(x_i) \,||^2 = \frac{1}{N}\sum_{i=1}^{N} ||\, x_i - (f_Z \circ f_V)(x_i) \,||^2$$