

**clusterAI**

**ciencia de datos en ingeniería industrial**

**UTN BA**

**curso I5521**

## **clase\_11: Autoencoders**

docente: martin palazzo

# agenda

## redes neuronales

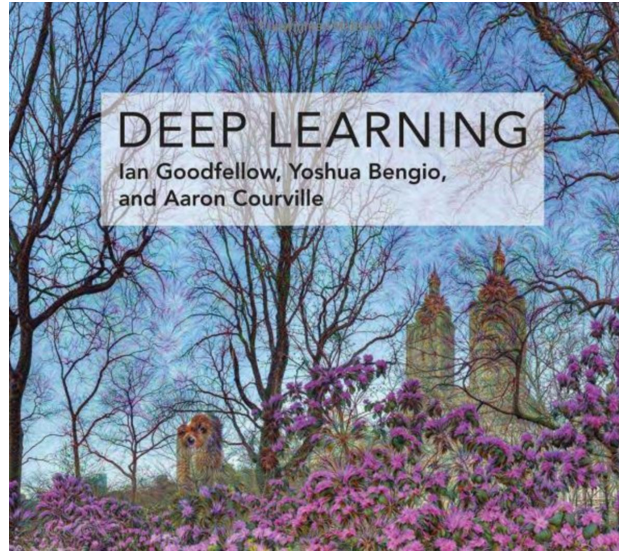
- perceptrón
- funciones de activación
- multilayer perceptrón
- arquitecturas de redes neuronales
- Entrenamiento de una red neuronal

# ¿que es una red neuronal artificial?

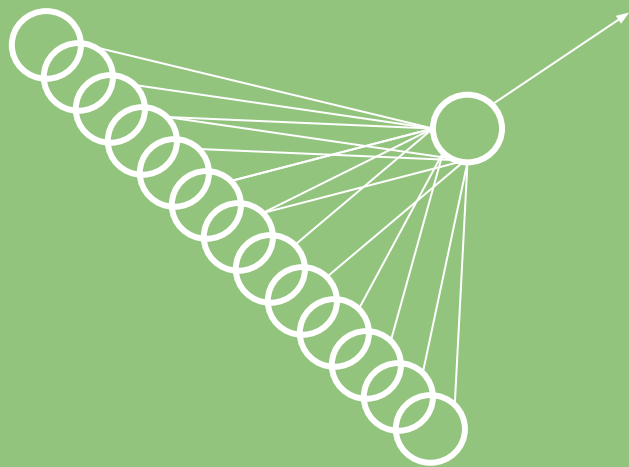
$$y = f(W, X)$$

Una red neuronal, es una función  $\mathbf{f}(\mathbf{w}, \mathbf{x})$  lineal o no lineal caracterizada por un conjunto de parámetros “W”. Esta función toma como entrada un vector  $\mathbf{x}$  d-dimensional para generar una respuesta/salida unidimensional ‘y’ o multidimensional ‘z’. Para obtener los parámetros  $\mathbf{w}$  se medirá mediante una función de costo  $\mathbf{L}$  distintos aspectos como la calidad de las variables de salida de la función  $\mathbf{f}$  (y o z).

# Deep learning book

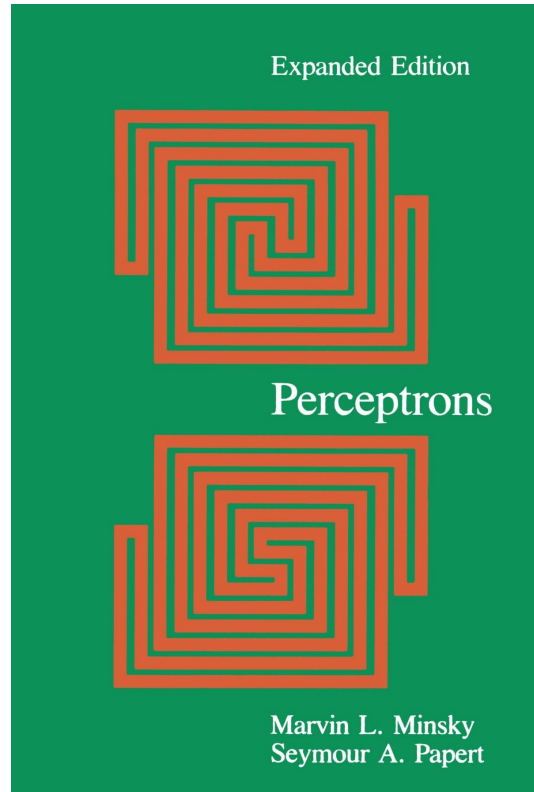


Uno de los libros más importantes en redes neuronales contemporaneas es <https://www.deeplearningbook.org/>



Modelo Perceptron

# perceptron



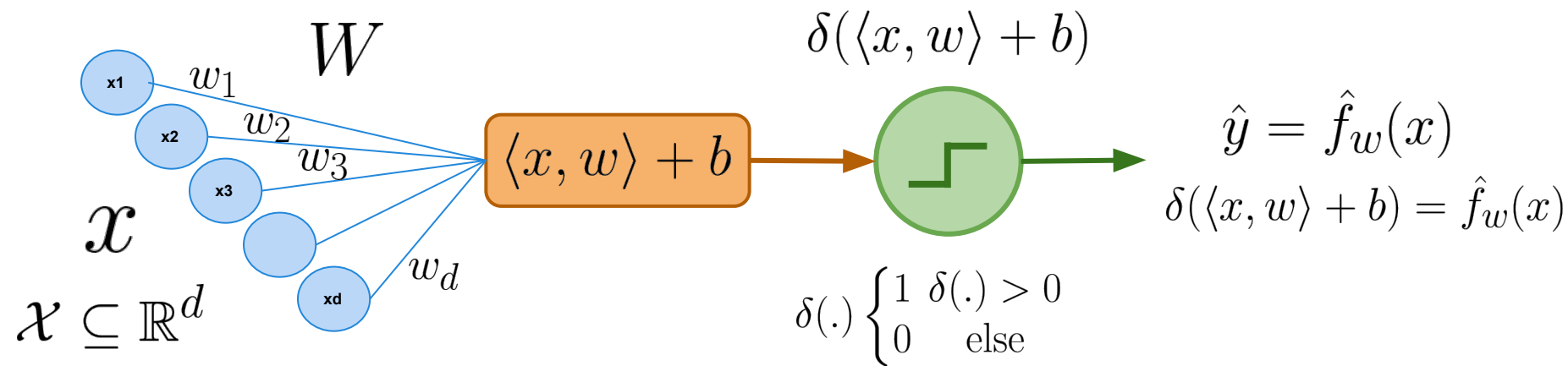
*Psychological Review*  
Vol. 65, No. 6, 1958

## THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN<sup>1</sup>

F. ROSENBLATT

*Cornell Aeronautical Laboratory*

# perceptrón



Para entender las redes neuronales primero debemos estudiar el modelo **perceptrón**. Es un algoritmo de Clasificación Lineal originado en los años 50s. El modelo tiene **d** features/variable de entradas, donde cada feature de entrada estará combinada linealmente a un “peso **w**”. Es decir que tenemos una combinación lineal **(x . w)**. Una vez realizada la combinación lineal se determina el valor de salida **(w\*x+b) = y**. Al resultado de la combinación lineal se lo afecta por una función de activación “step” que en el caso de ser binario resultará como 0 o 1.

El objetivo es aprender los pesos **w, b** que minimizan el error de clasificación **y\_hat**.

# Perceptrón

- Es un modelo que trabaja “online”. Procesa una muestra a la vez. De esta manera los pesos también se actualizan gradualmente.
- Únicamente en los casos donde el modelo realiza una predicción errónea los pesos se actualizan.
- El perceptrón computa una función de costo  **$L(y, \hat{y})$**  basada en la cantidad de veces que hubo una mal clasificación.
- La función de activación (‘neurona’) que utiliza es “signo o step” -> es 0 si  $x < 0$  y +1 si  $x > 0$ .



# Ejercicio perceptron

$$\mathbf{x}_{test} = \begin{bmatrix} x_1 & x_2 \\ 1 & 1 \\ 3 & 0.5 \\ 2 & 3 \\ 2.5 & 1.5 \\ 1.5 & 2 \\ 2.5 & 4 \\ 3.5 & 1.5 \\ 4 & 3 \end{bmatrix} \quad \mathbf{y}_{test} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

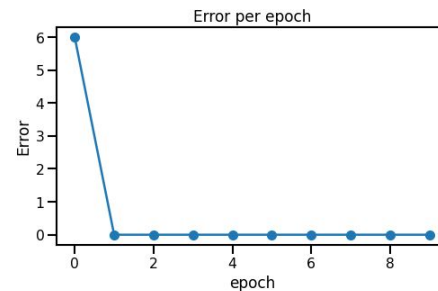
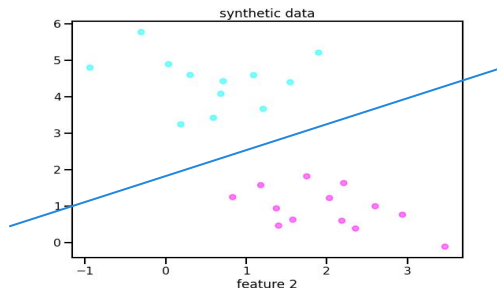
$$\delta(\langle x, w \rangle + b) = \hat{f}_w(x)$$

	<b>w1</b>	<b>w2</b>	<b>b</b>	<b>Accuracy</b>
$\hat{f}_1$	2	2	-9	?
$\hat{f}_2$	3.4	5	-14	?

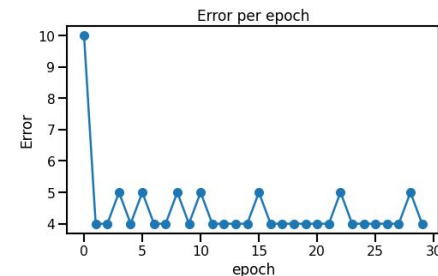
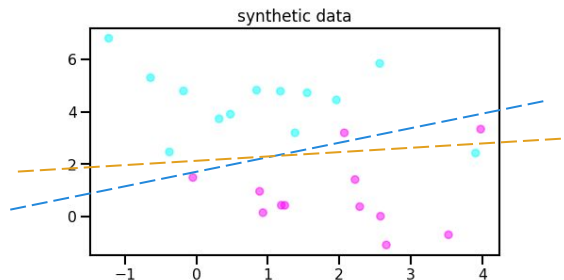
Suponer dos modelos perceptrón  $f_1$  y  $f_2$  ya entrenados previamente con un dataset que no disponemos. Por lo tanto los parámetros de los modelos están disponibles. Se solicita evaluar los dos modelos utilizando las muestras de test provistas y calcular el accuracy para determinar cual de los dos generaliza mejor.

# Limitaciones del perceptrón

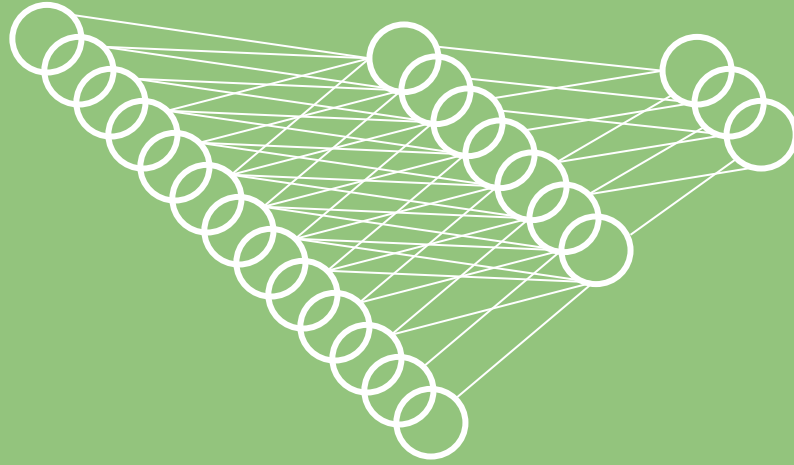
Clases separables



Clases linealmente no separables



Durante el entrenamiento el dataset es procesado por el perceptrón múltiples veces iterativamente donde cada iteración se la denomina “epoch”. El perceptrón únicamente actualiza los parámetros cuando el output  $\mathbf{\hat{y}}$  es distinto a la verdadera etiqueta  $\mathbf{y}$ . En clases linealmente separables el perceptrón convergerá a una solución. En clases no separables el perceptrón iterará la cantidad de veces que el usuario le indique sin poder llegar a una solución definitiva.



Redes neuronales artificiales

# Funciones de activación

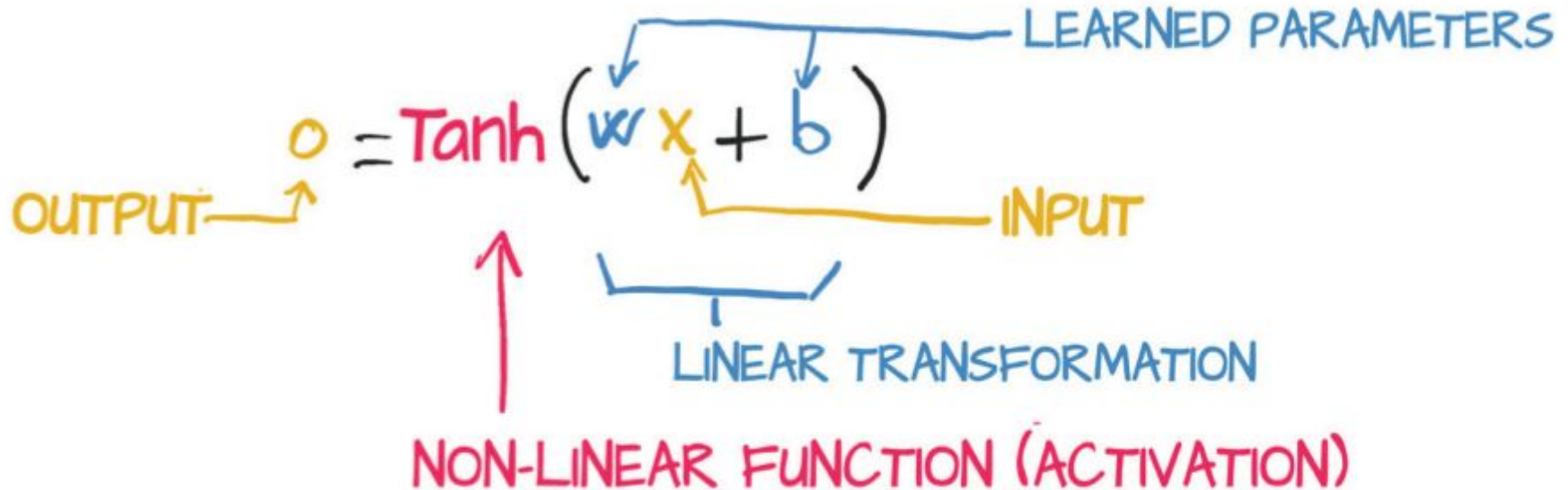
$$\mathbf{f}_i = \sigma_i(\mathbf{W}_i \mathbf{X}_{i-1} + \mathbf{b}_i)$$

La función de activación es un **nodo o neurona** que tiene múltiples entradas y define una determinada salida. Las funciones de activación pueden estar “encendidas” o “desactivadas”. Algunas populares son:

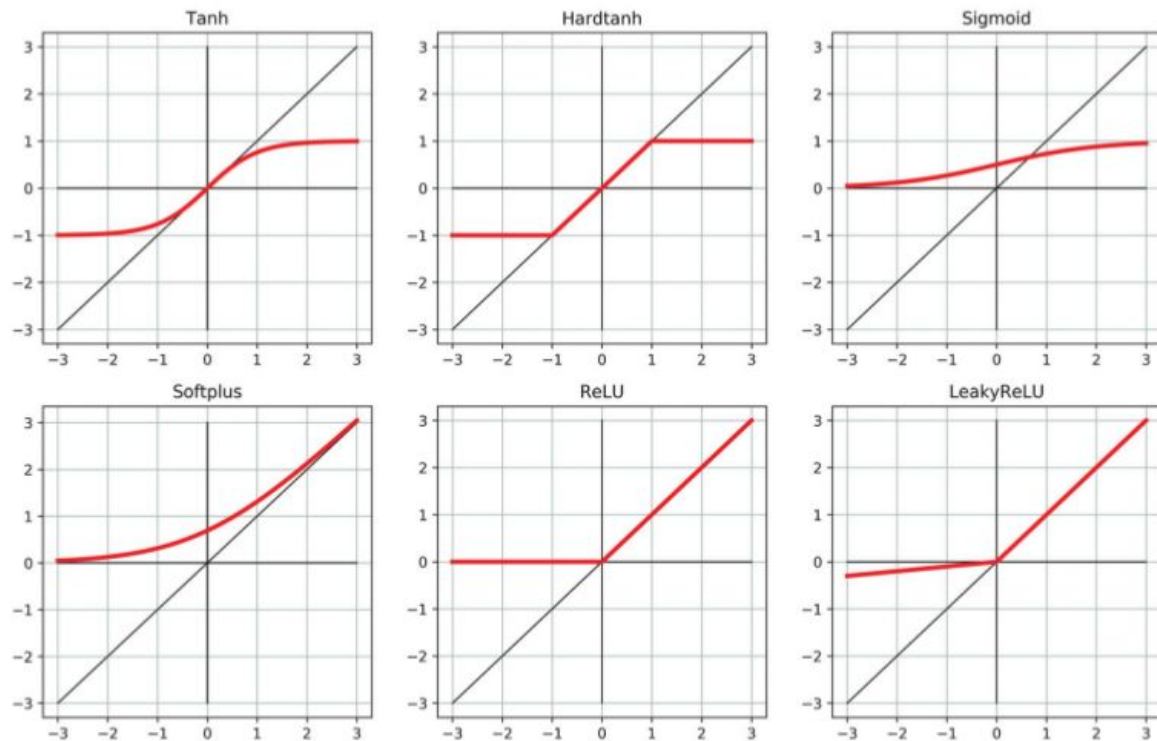
- ReLu
- Softmax
- Sigmoid
- TanH

# Funciones de activación

## THE "NEURON"

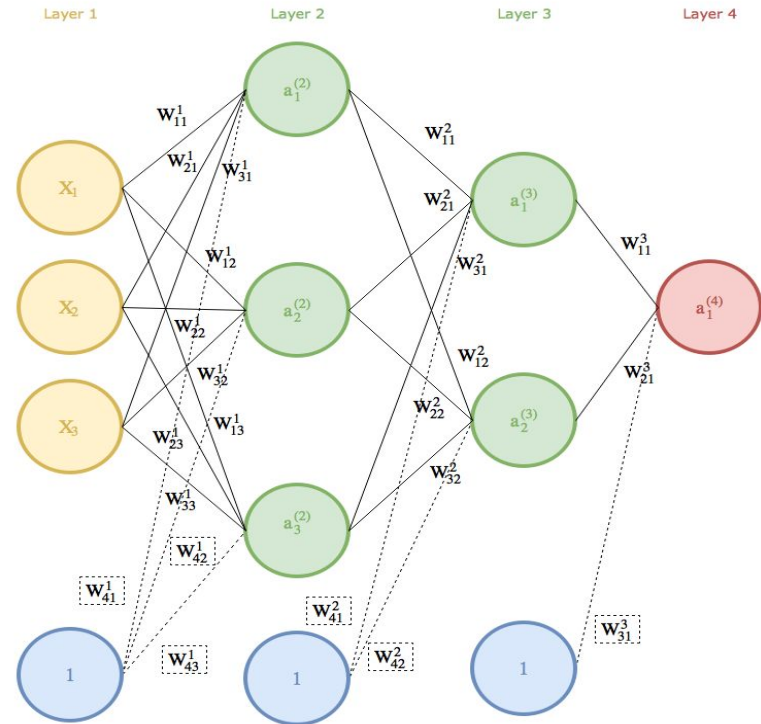


# Funciones de activación (neuronas)

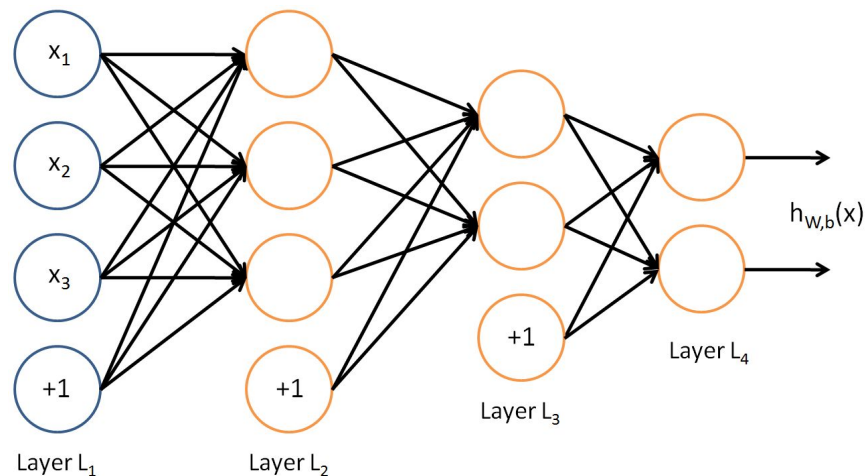


# Multilayer Perceptrón / Feedforward Network

Si al modelo perceptrón le incorporamos capas intermedias entre la entrada y la salida obtenemos lo que comúnmente se llama “red neuronal con capas ocultas” o en inglés “neural network with hidden layers”. El arco que conecta dos neuronas se caracteriza con un peso que se aprenderá con el set de entrenamiento para minimizar el error.



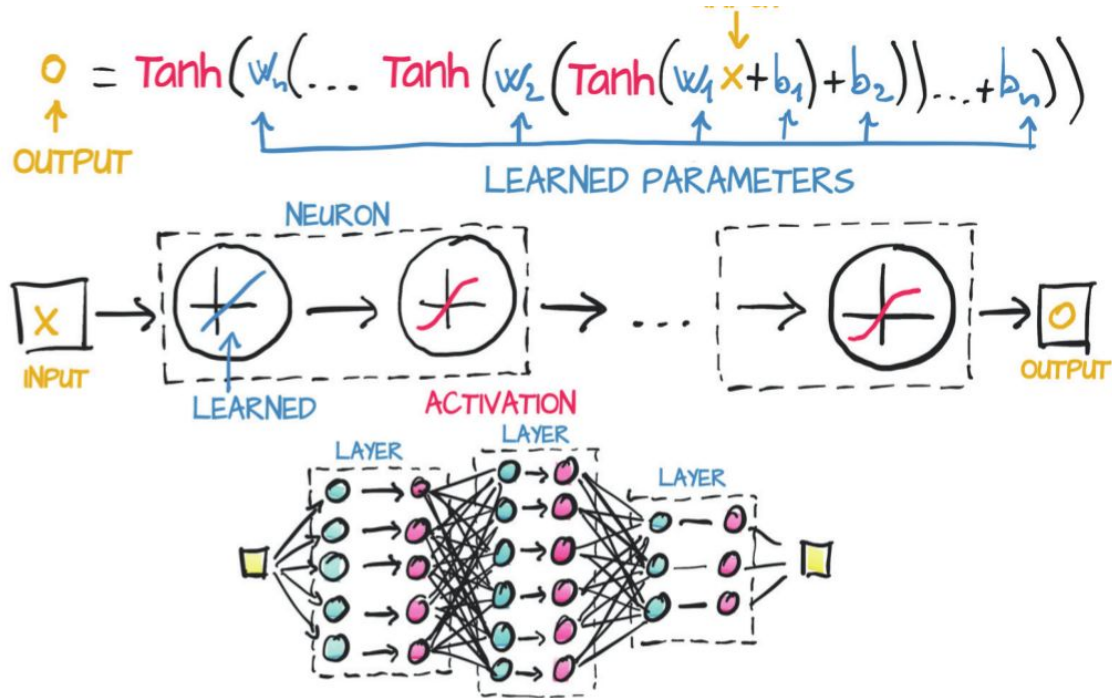
# Neural Networks: arquitectura



La arquitectura multi-capa o perceptrón multi-capa permite capturar patrones mas complejos en los datos. Cada capa es una representación de los datos donde cada neurona de una capa es una feature/variable aprendida por la combinación de variables de la capa anterior. A medida que mas capas se agregan mas complejas pueden ser las variables/features aprendidas. De todos modos agregar mas capas implica agregar mas complejidad y parámetros al modelo.

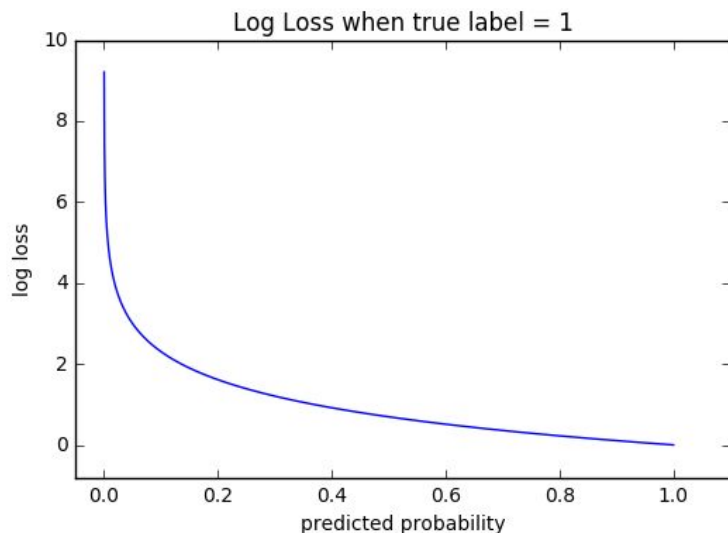


# Neural Networks: arquitectura



# Funciones de costo: Cross Entropy

Mide la performance de un modelo de clasificación cuya salida es una probabilidad de 0 a 1. El Cross Entropy Loss aumenta cuando la probabilidad predicha diverge de la etiqueta actual. Un modelo perfecto debería dar una Cross Entropy loss = 0



$$- [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

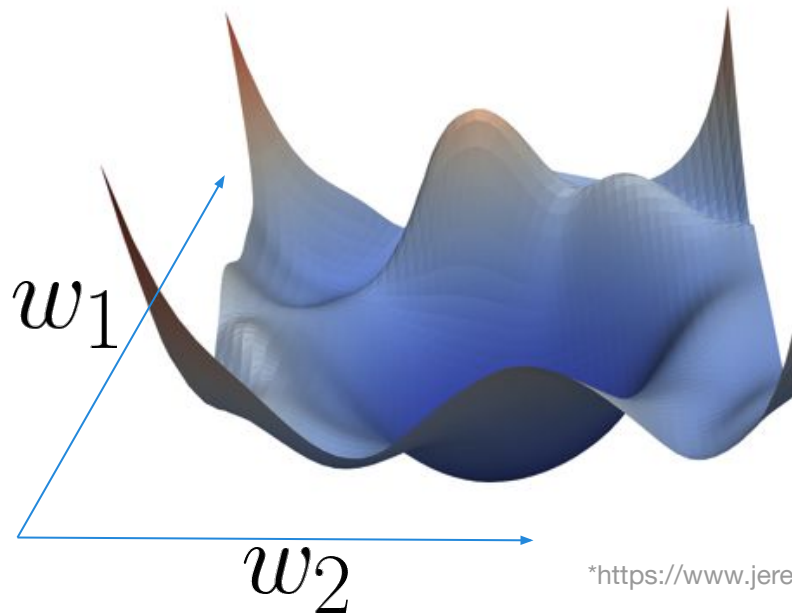
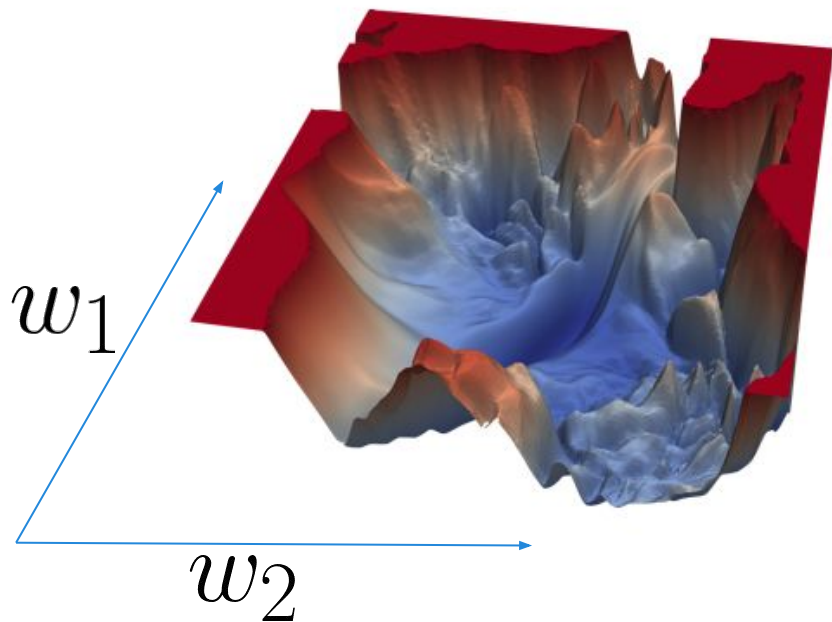
# Funciones de costo: Mean Squared Error

Mide la performance de un modelo de regresión cuya salida es la diferencia al cuadrado de los residuos. El MSE además de usarse en problemas de regresión puede usarse en autoencoders donde se busca reconstruir un vector d-dimensional.

$$\text{MSE} = \left\| y_i - y_j \right\|^2$$

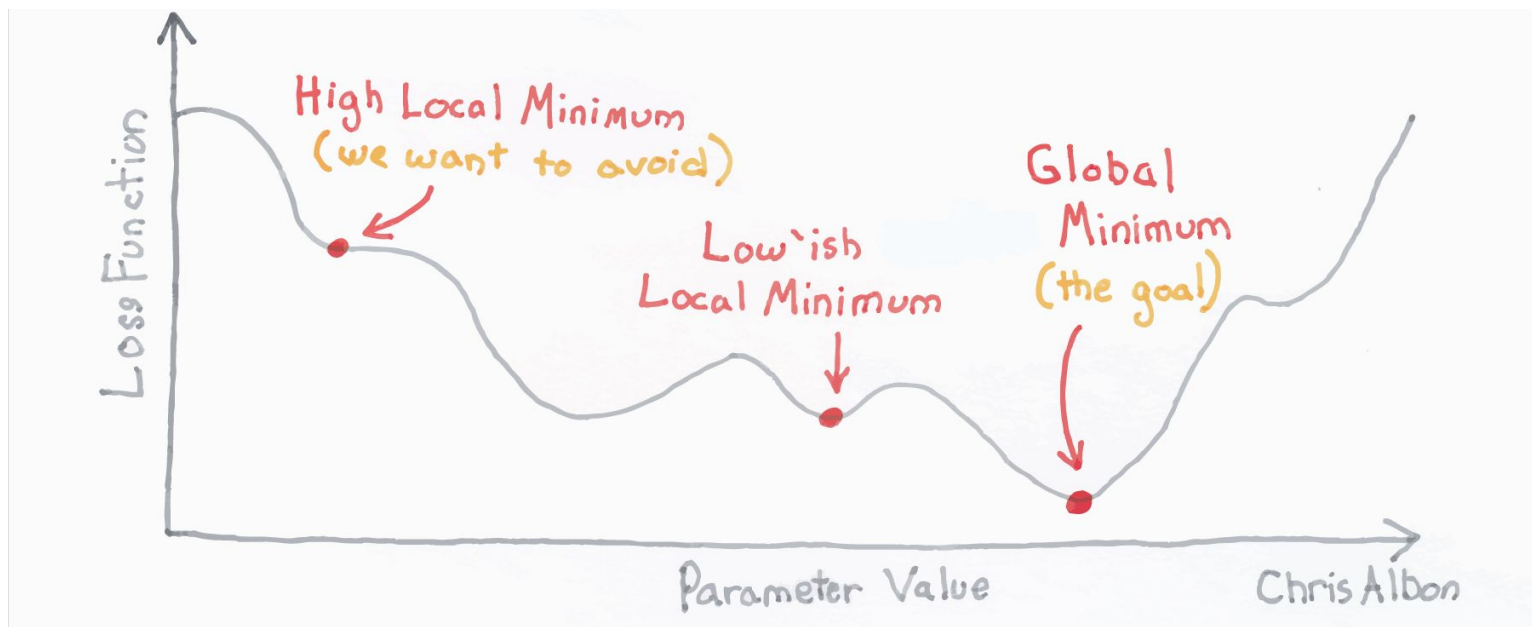
# Funciones de costo

Las funciones de costo dependen de los parámetros  $\mathbf{w}$  (pesos) del modelo y del dataset en cuestión. El objetivo es encontrar un vector de parámetros que se encuentre en un mínimo local aceptable de la función de costo.



# Funciones de costo

Nuestro deseo es encontrar la configuración de pesos  $\mathbf{w}$  que minimiza globalmente la función de costo. A veces no conseguimos ese objetivo aunque podemos llegar a mínimos locales que permiten una performance de clasificación aceptable.



# Backpropagation & Gradient Descent

**Gradient Descent** o Gradiente Descendiente es una técnica de optimización para explorar la función de costo y buscar el mínimo que optimizará nuestra red. El punto de mínimo costo en la función de error implica la combinación de parámetros (pesos) que minimiza el error o la mal clasificación con las muestras de train. Una vez que el gradiente descendiente encuentra un mínimo, detendrá la búsqueda y se quedará en dicha región de la función de costo. La manera en que se busca

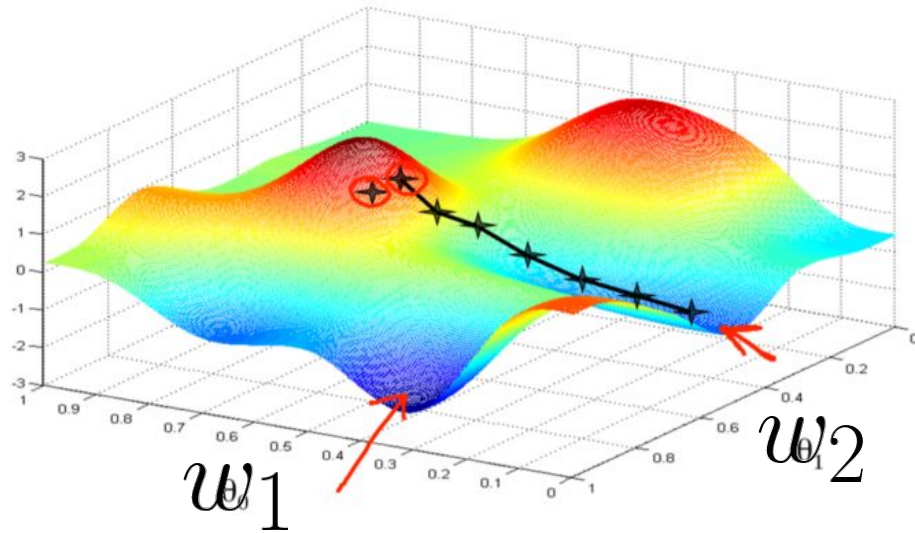
$$w_{h+1} = w_h - \alpha \frac{\partial L}{\partial w}$$

El **Learning Rate** (alpha) es un hiperparámetro que indica cuán rápido se actualizan los pesos en búsqueda de un mínimo.

# Backpropagation

**Backpropagation** es el algoritmo de actualización de pesos de una red neuronal para moverse a favor del gradiente descendiente. Sucede luego que la información fluye desde la entrada input a la salida output (forward propagation) y se conoce si el modelo clasificó bien o mal a cada muestra. Mediante la **regla de la cadena** se calculan los gradientes de la función de costo en función de los parámetros  $\mathbf{w}$ , es decir que se actualizan los pesos que aproximan en dirección al mínimo (local).

# Learning Rate y Loss Function



En este caso tenemos una función de costo determinada por dos parámetros (pesos). El Learning Rate se visualiza como los “pasos” que se dan hacia la búsqueda del mínimo. Un Learning rate bajo significa tardar mucho en llegar a un mínimo local. Un LR alto puede significar pasar “por alto” un mínimo y no converger.



# Neural Networks: components

- Activation functions
- Network architecture
- Loss Function
- Weight optimization technique
- Learning rate

# Neural Networks: clasificador

Softmax: Multi o binary  
classification

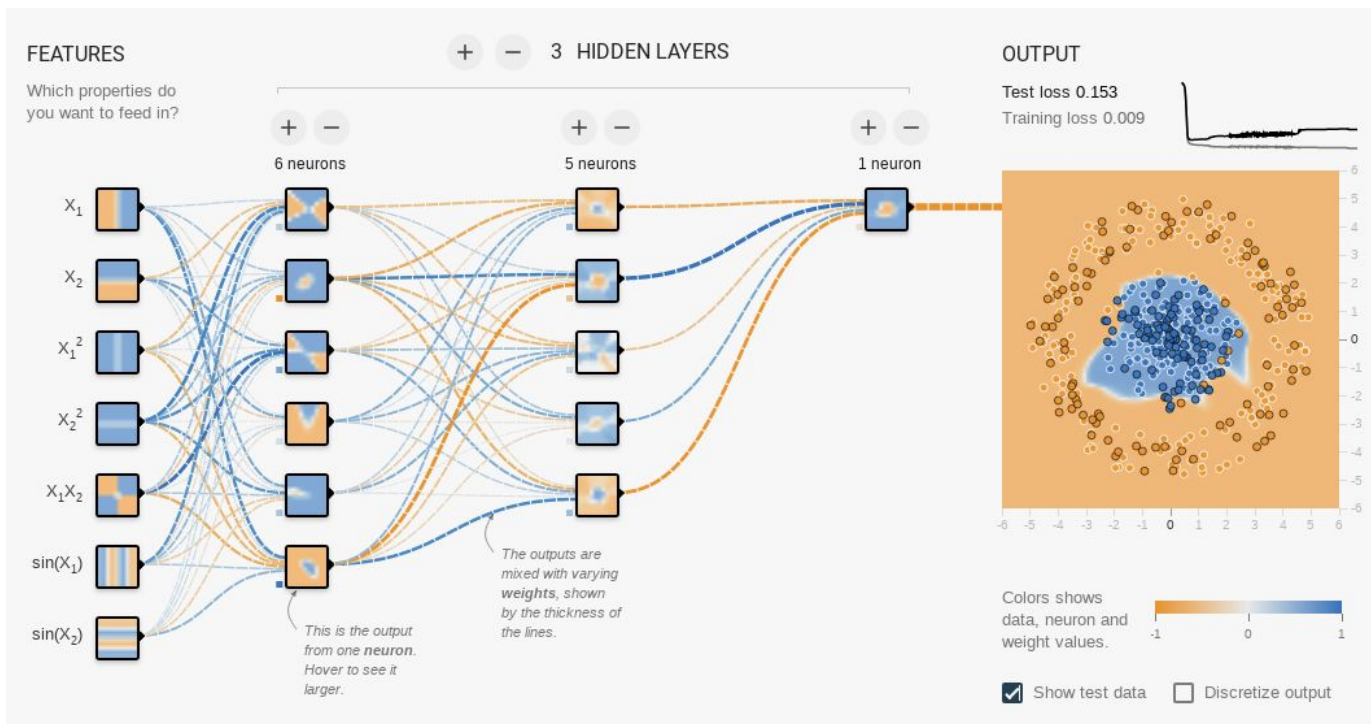
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Sigmoid: Binary  
classification

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Al final de todas las “layers” de la red neuronal se ubica una función de activación llamada “Softmax” o ‘Sigmoid’ que arroja valores entre 0 y 1 a modo de probabilidad siendo las etiquetas 0 y 1. Los pesos de la red se aprenderán de manera tal que la salida de la softmax o sigmoid coincida lo mejor posible con las etiquetas reales de cada muestra.

# Neural Networks: clasificador



# Regularización en redes neuronales

Regularización es penalizar al modelo a la hora de aprender los parámetros internos (los pesos  $W$ ). No queremos que el modelo aprenda perfectamente los datos de entrenamiento puesto que estos no son 100% representativos de la realidad. La regularización nos permite que el modelo no haga “overfitting”.

- Norma L1 sobre los pesos
- Norma L2 sobre los pesos
- Dropout en las neuronas
- Batch-Normalization

# Librerías para ANNs



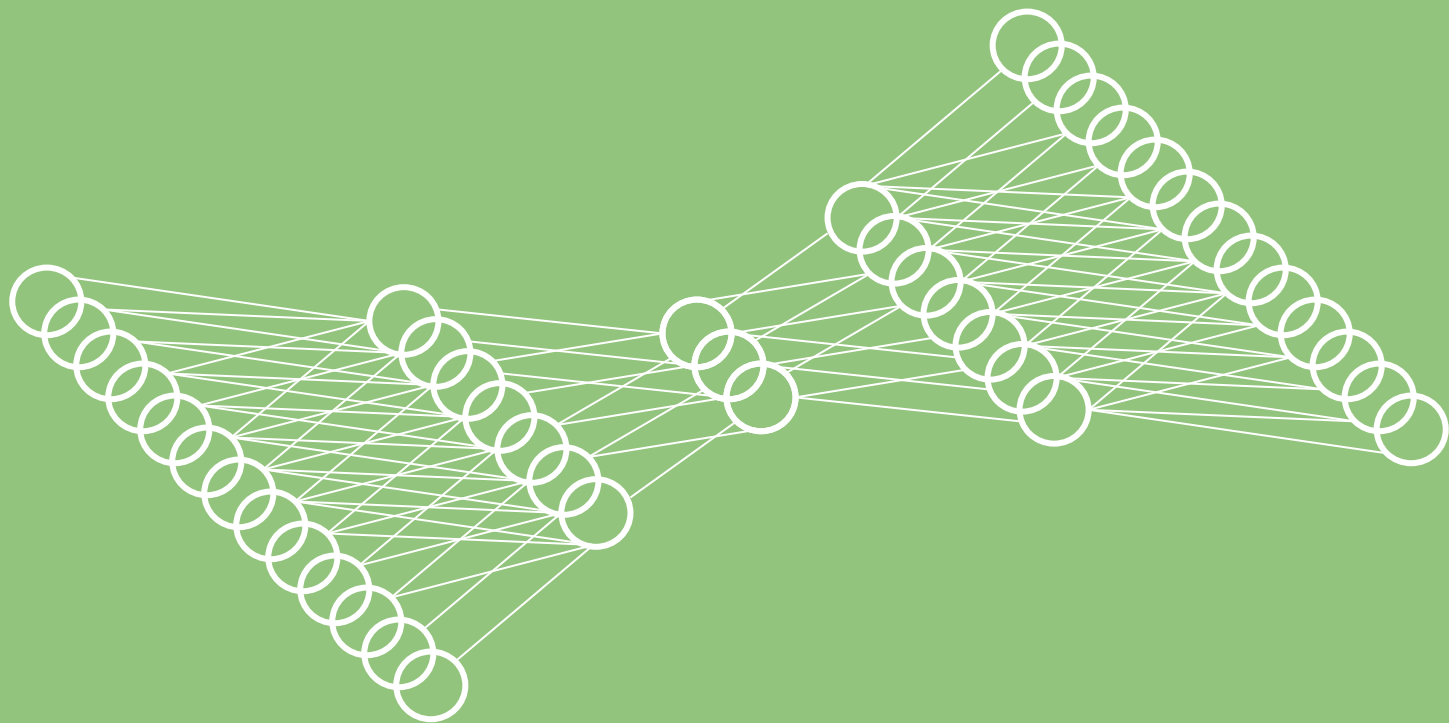
Keras



TensorFlow

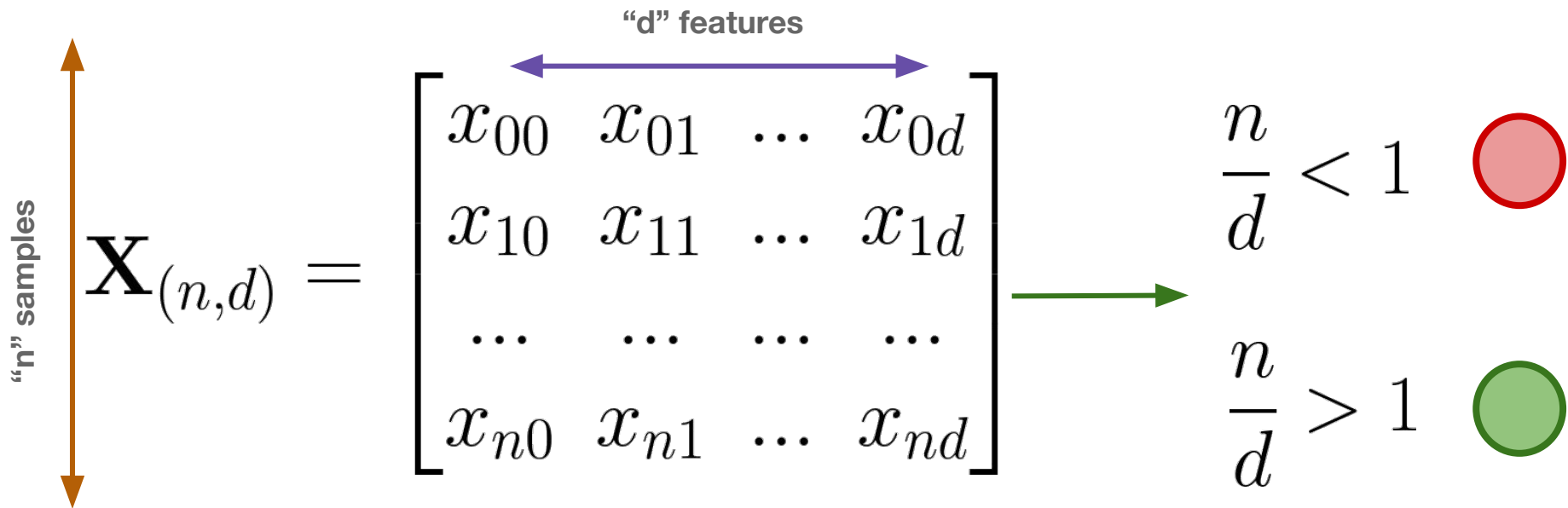


PyTorch



Autoencoders

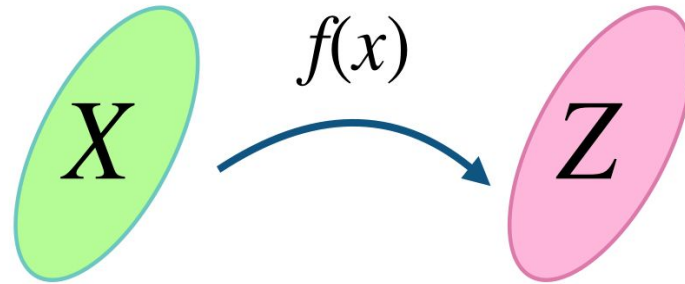
# Sample-to-feature ratio



When the number  $n$  of samples  $<$  number of features  $d$ , the relationship between instances and dimensions is less than one and that implies greater difficulty in being able to explain and describe the space where our instances live.

# Dim. Reduction: Feature extraction

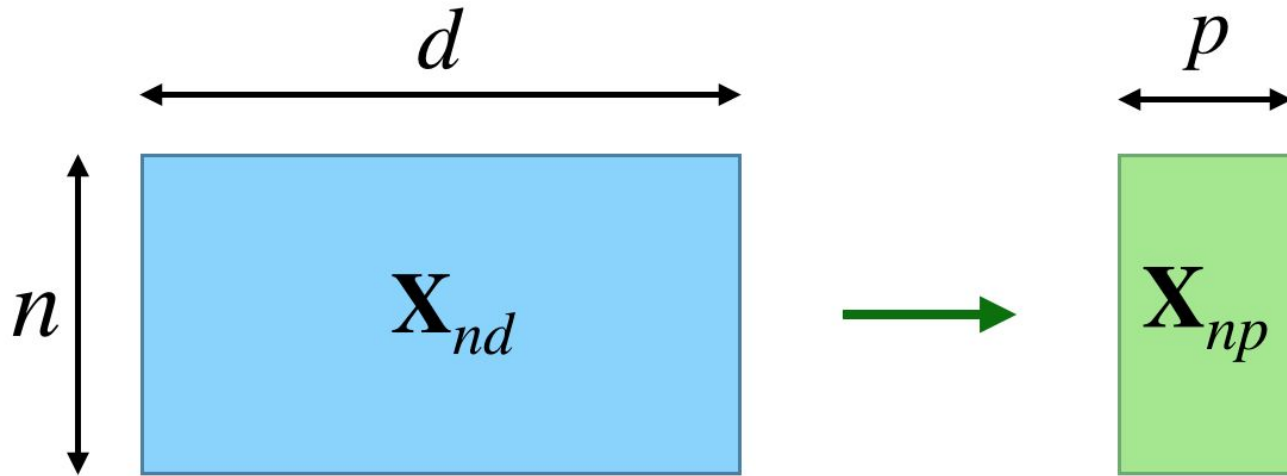
$$\begin{aligned}\mathcal{X} &\in \mathbb{R}^d \\ \mathcal{Z} &\in \mathbb{R}^p \\ p &< d\end{aligned}$$



A way to reduce dimensionality is by feature extraction methods (PCA, kPCA, Autoencoders). This method learn a transformation  $f(x)=z$  where  $\mathbf{Z}$  is a low dimensional subspace where data  $\mathbf{X}$  is projected. Then on the reduced subspace clustering can be performed.

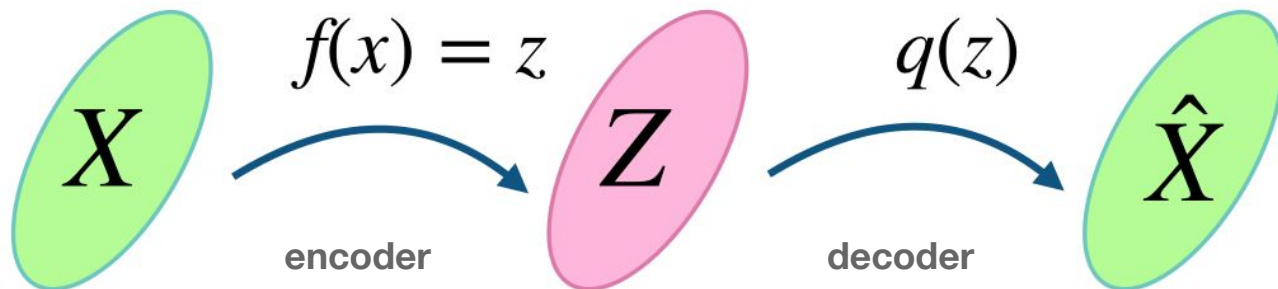


# High dimensional data



A way to reduce dimensionality is by feature extraction methods (PCA, kPCA, Autoencoders). This method learn a transformation  $f(\mathbf{x})=\mathbf{z}$  where  $\mathbf{Z}$  is a low dimensional subspace where data  $\mathbf{X}$  is projected. Then on the reduced subspace clustering can be performed.

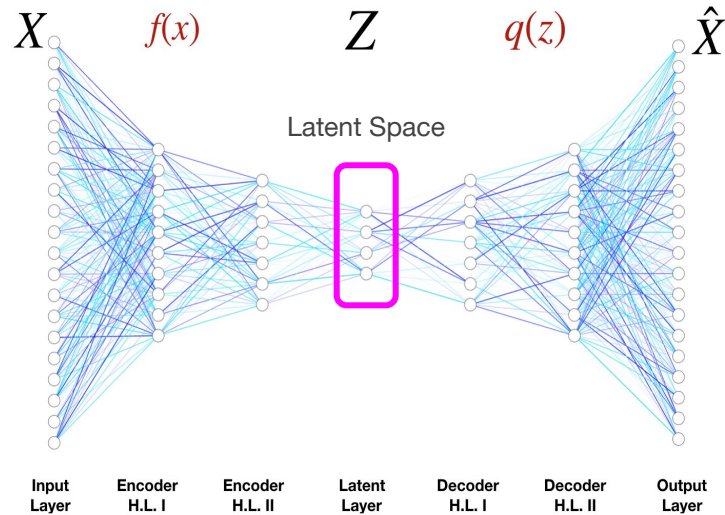
# Autoencoder



The objective is to learn two functions:

- encoder  $f(x) = z$  maps the high dimensional input vectors  $\mathbf{X}$  in a low dimensional latent space  $\mathbf{z}$
- decoder  $q(z) = x$  maps the latent vectors  $\mathbf{z}$  to the original high dimensional input space.

# Autoencoders



$$\mathbf{z} = f(\mathbf{x}, \mathbf{W}_f) = \sigma(\mathbf{W}_f \mathbf{x} + \mathbf{b}_f)$$
$$\tilde{\mathbf{x}} = q(\mathbf{z}, \mathbf{W}_q) = \sigma(\mathbf{W}_q \mathbf{z} + \mathbf{b}_q)$$

Using autoencoders it is possible learn low-dimensional representations of the data (latent space). These representations help to find clusters although they are not interpretable in biological terms.

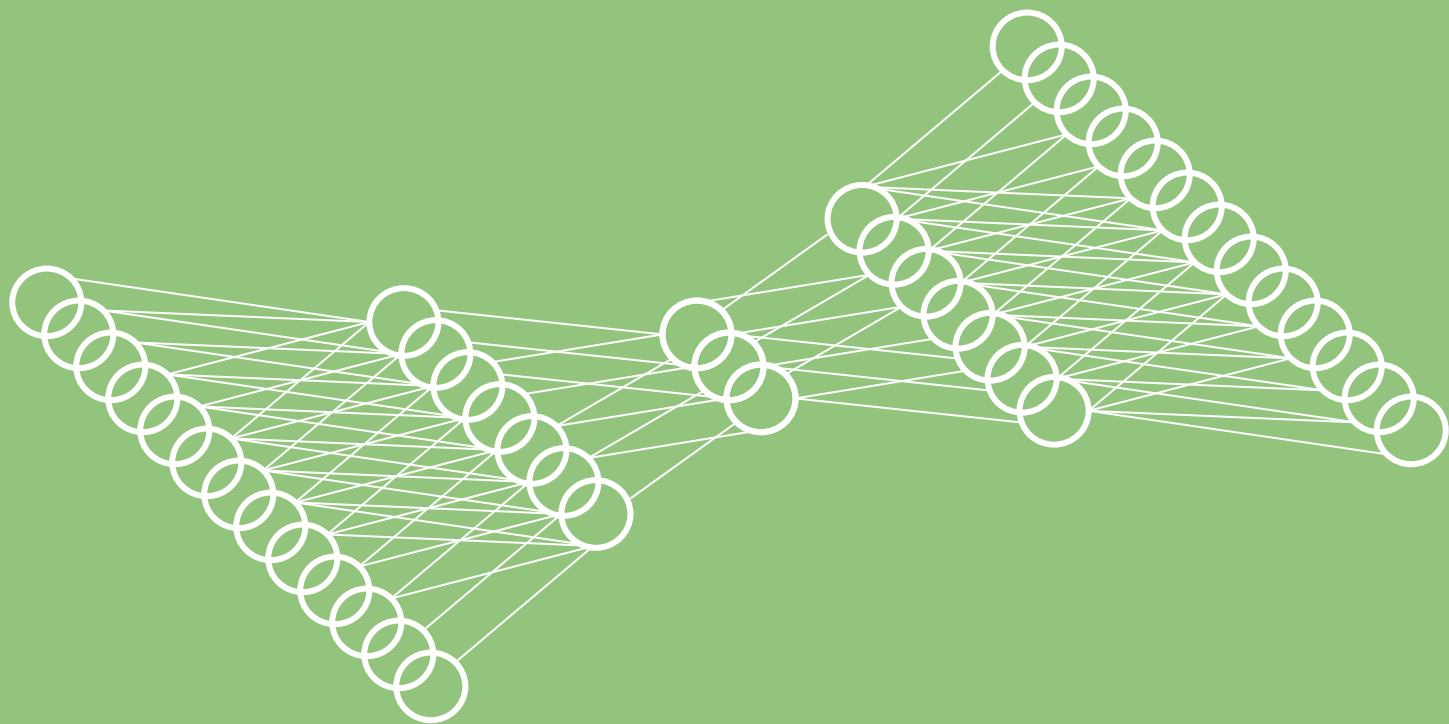
[1] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786), 504-507.

[2] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning Book. Cambridge: MIT press. (Chapter 14)

Figure: Dimension Reduction of tumor profiles using autoencoders (Palazzo M.)

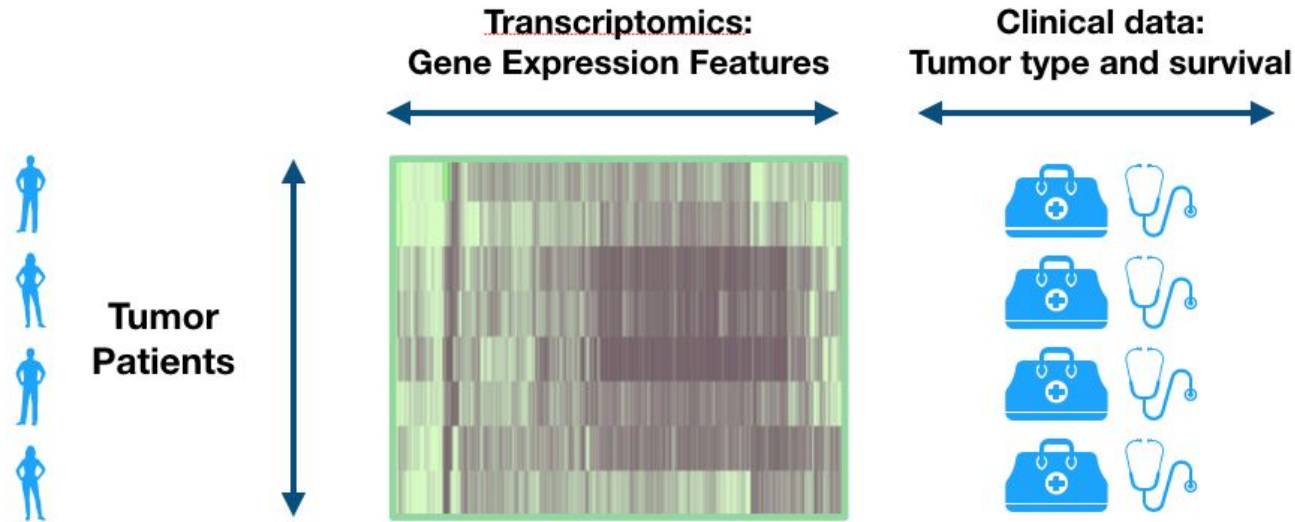
## Autoencoder Loss Function: MSE

$$L(x, \hat{x}) = L(x, q(f(x))) = \|x - \hat{x}\|^2$$



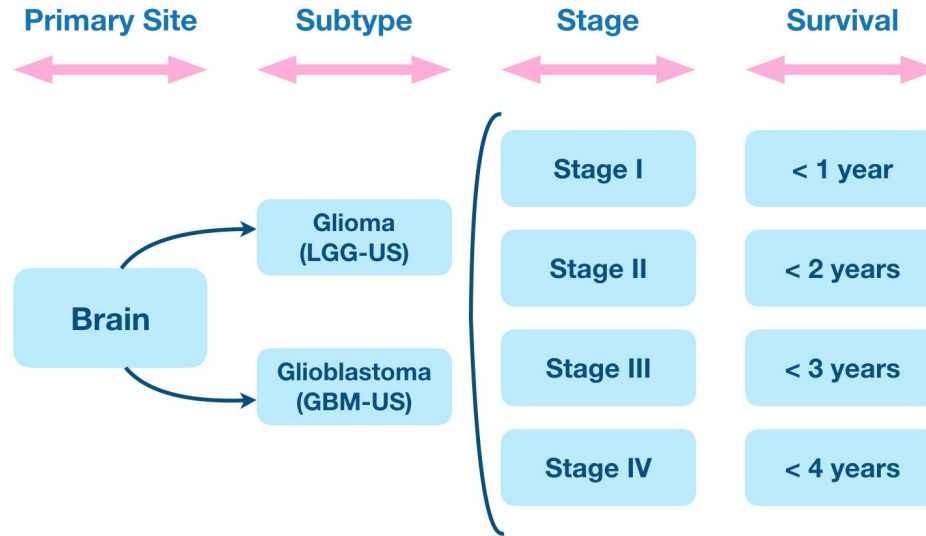
Autoencoders en biomedicina

# Biomedical data from cancer patients



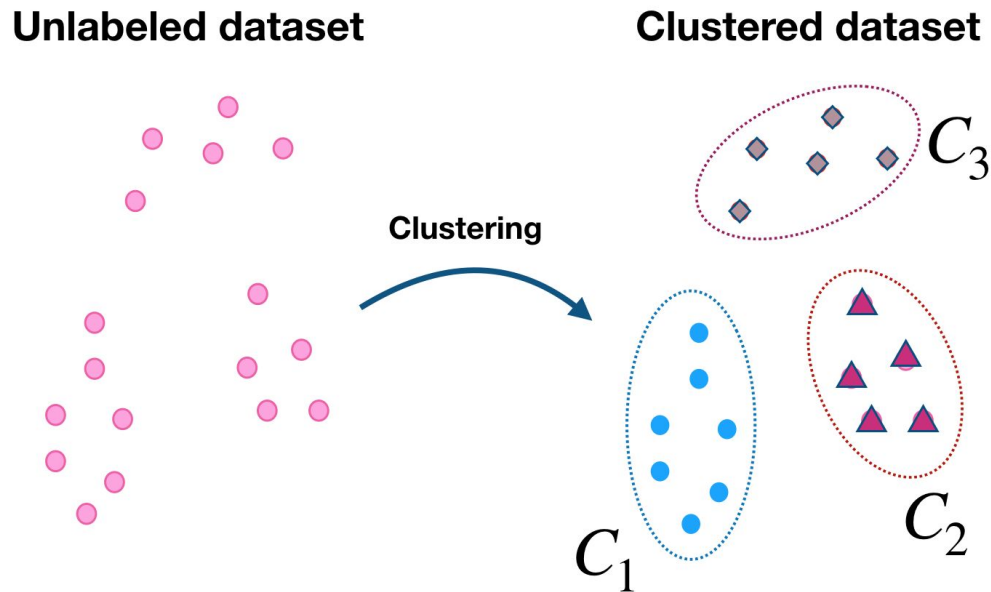
Each patient's tumor is characterized by thousands of bio-molecular features such as Genetic expression. In addition, each patient is labeled with clinical information such as tumor type or subtype, survival, and tumor stage.

# Tumor types, subtypes and stages



Tumor types presents inner heterogeneity that can be subdivided in tumor subtypes, stage or survival.

# Tumor subtype discovery

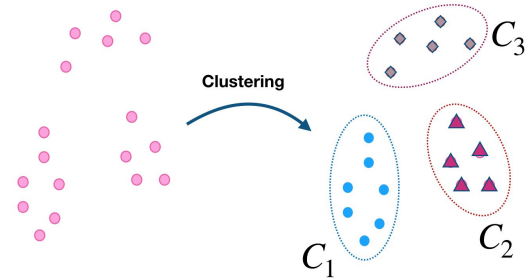
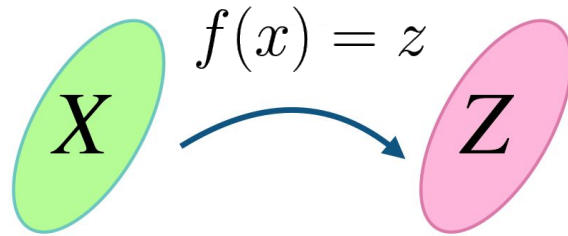


Tumors can be clustered to discover new subgroups (subtypes) with relevant clinical differences.



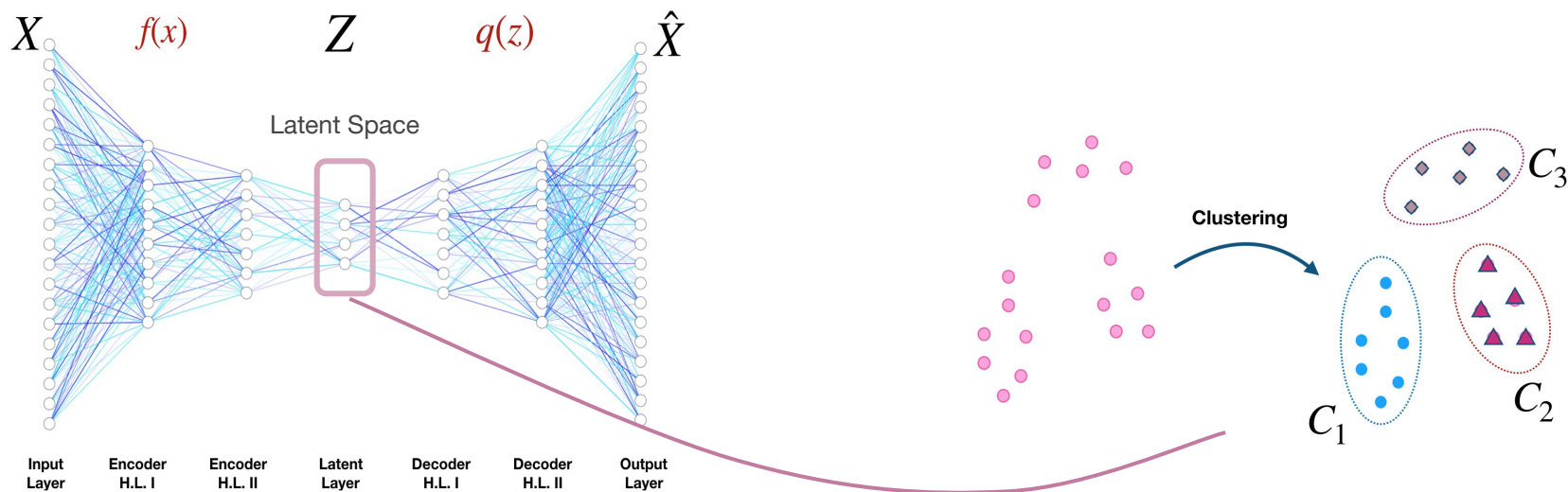
# Dim. Reduction: Feature extraction

$$\begin{aligned}\mathcal{X} &\in \mathbb{R}^d \\ \mathcal{Z} &\in \mathbb{R}^p \\ p &< d\end{aligned}$$



A way to reduce dimensionality is by feature extraction methods (PCA, kPCA, Autoencoders). This method learn a transformation  $f(x)=z$  where  $\mathbf{Z}$  is a low dimensional subspace where data  $\mathbf{X}$  is projected. Then on the reduced subspace clustering can be performed.

# Autoencoders for tumor subtype discovery



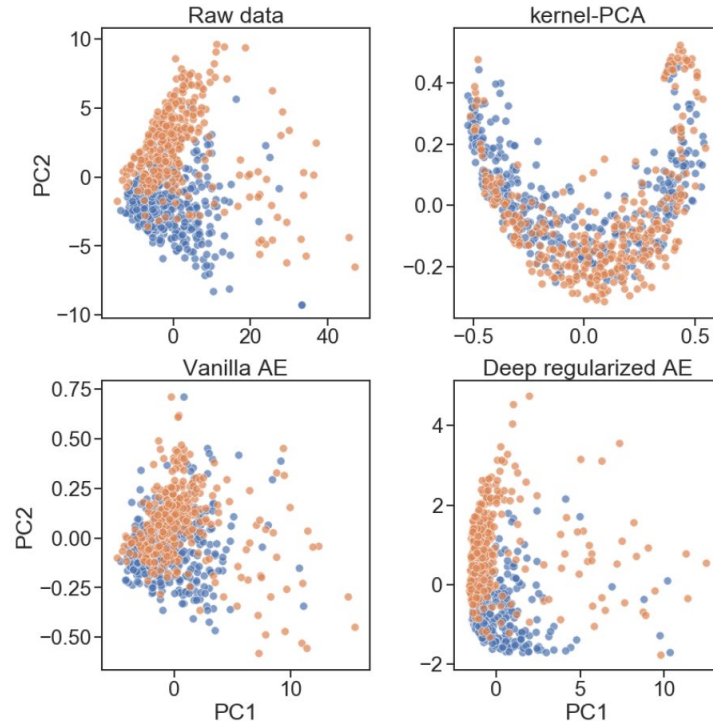
Using autoencoders it is possible learn low-dimensional representations of the data (latent space). These representations help to find clusters although they are not interpretable in biological terms.

[1] Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786), 504-507.

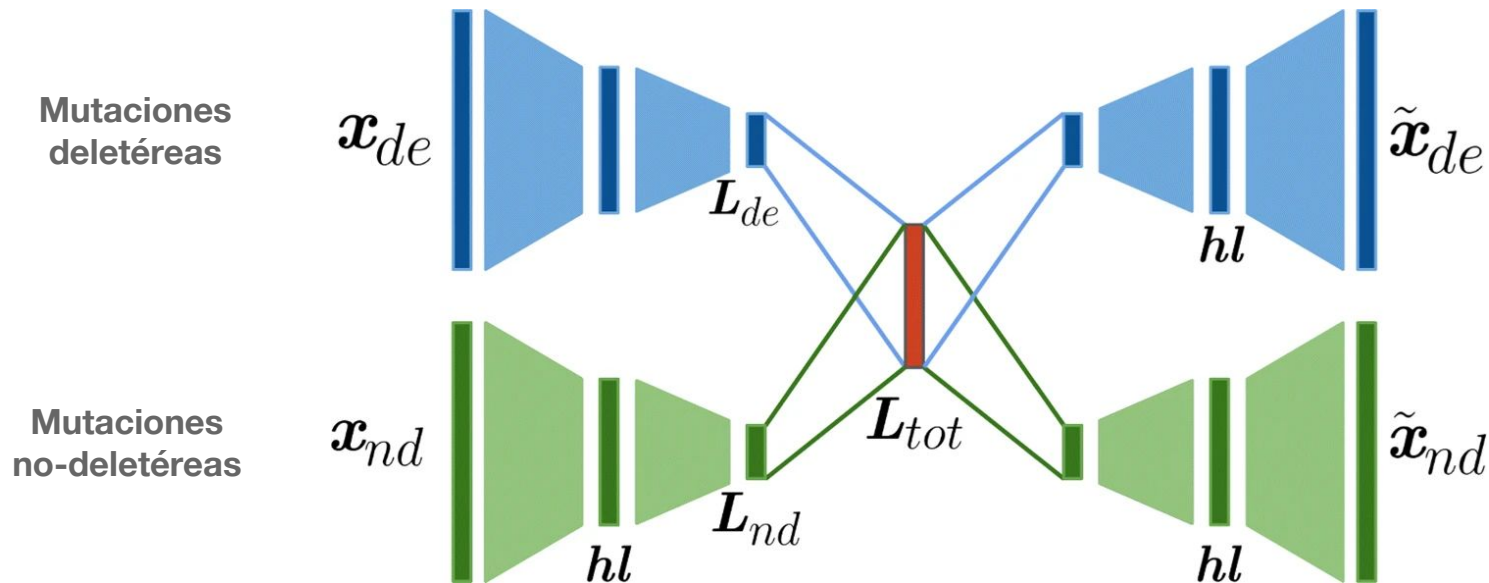
[2] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning Book. Cambridge: MIT press. (Chapter 14)

Figure: Dimension Reduction of tumor profiles using autoencoders (Palazzo M.)

# Feature extraction methods

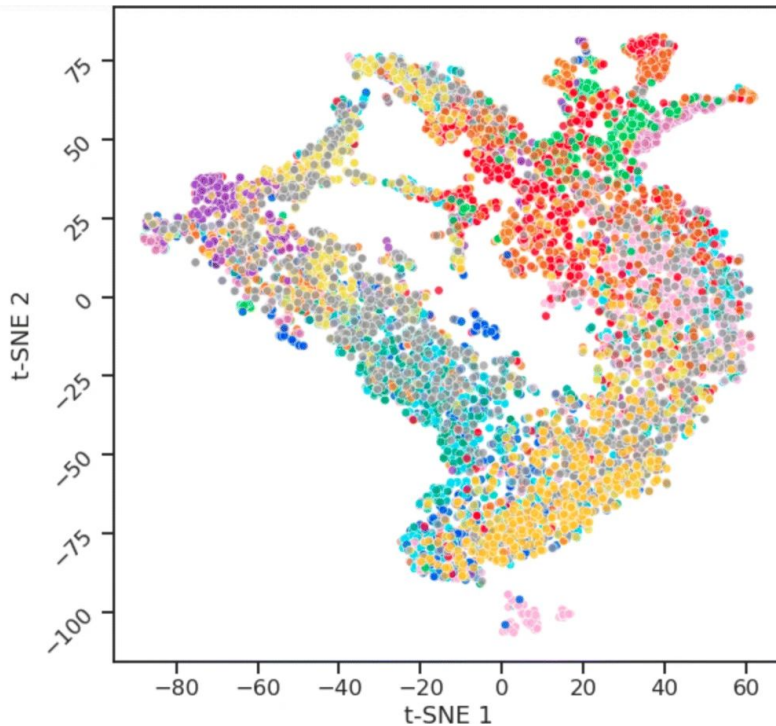
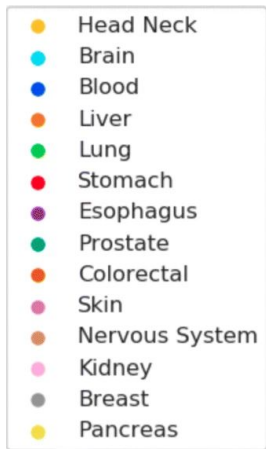


# Autoencoder multi-modal



Los autoencoders son redes neuronales que permiten reducir la dimensionalidad de los datos para mejorar tareas de aprendizaje aguas abajo. Este trabajo propone aprender desde distintas capas de información biológica de manera multi-modal.

# Aprendizaje de representaciones de perfiles tumorales usando mutaciones somáticas



Una mejor representación permite entender cómo se agrupan los distintos tipos de tumor.

La calidad del autoencoder multi-modal es validada por tareas de aprendizaje no supervisado en muestras de evaluación nunca vistas por el modelo.

El objetivo es poder estudiar la distribución de distintos tipos y sub-tipos de tumor basados en sus perfiles mutacionales.

## Metodos

Blablabaldblfbdl.s.da.....

.....

Por eso usamos el modelo de Support Vector Machines (SVM) [1] para hacer clasificacion. Luego hicimos Principal Component Analysis [2] para reducir la dimensionalidad de los datos de entrada blablaba

.....

## resultados

Blablabaldblfbdl.s.da.....

.....

## conclusion

Blablabaldblfbdl.s.da.....

.....

## Referencias

- [1] Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (Vol. 1, No. 10). New York: Springer series in statistics.
- [2] Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3), 37-52.
- [3] ....