Redes neuronales

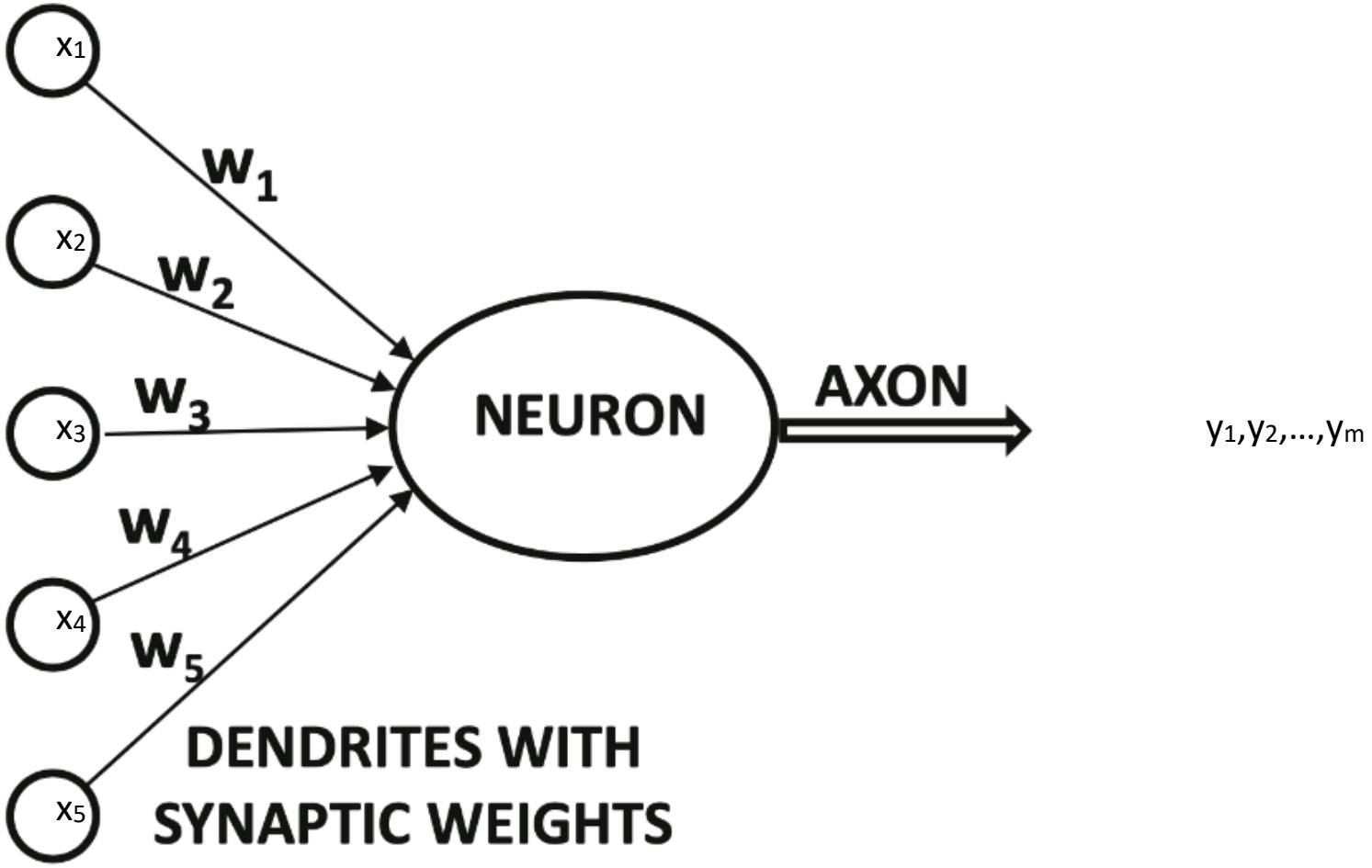
Dr. Raimundo Sánchez

raimundo.sanchez@uai.cl

@raimun2

Introducción

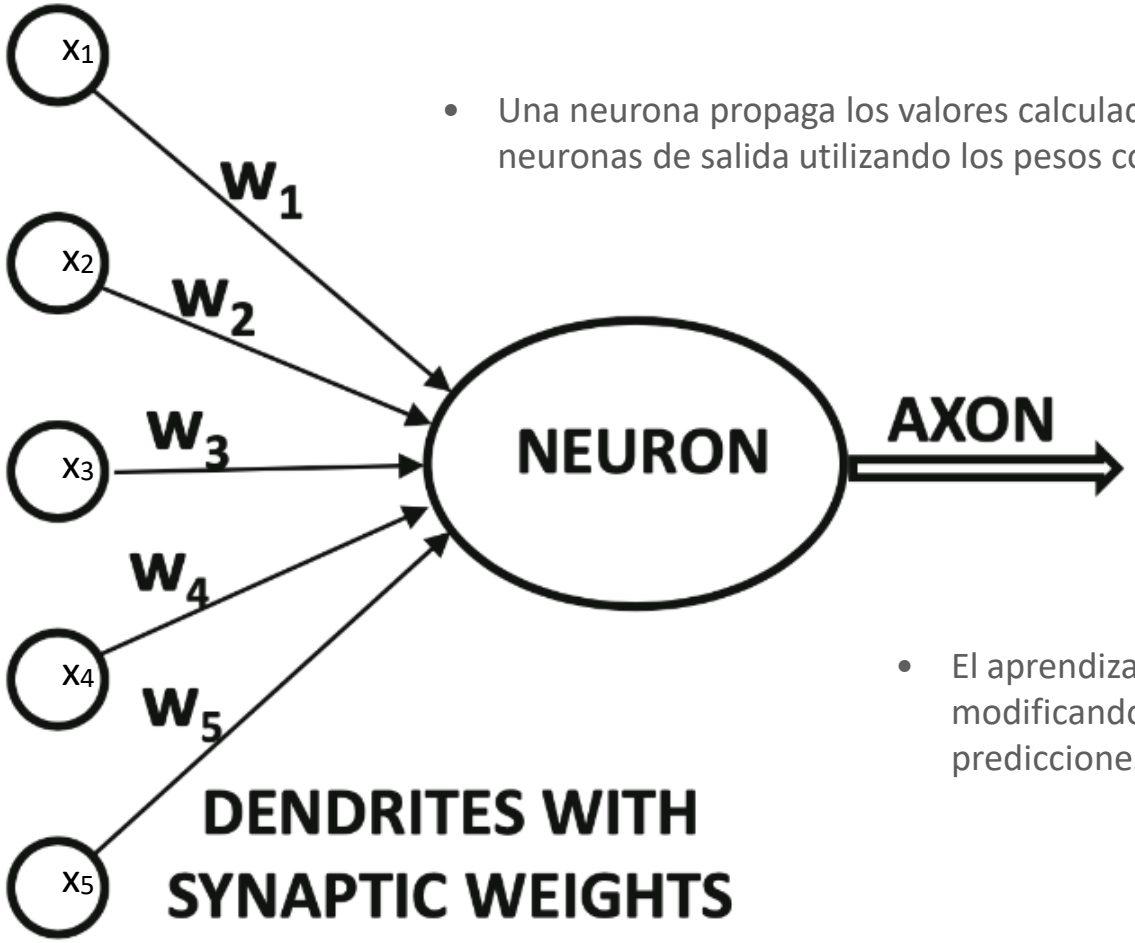
Las redes neuronales artificiales son modelos de aprendizaje automático que simulan el mecanismo cerebral de organismos biológicos, donde su unidad más básica es una neurona.



$y_1, y_2, \ldots, y_m$

Introducción

Las redes neuronales artificiales son modelos de aprendizaje automático que simulan el mecanismo cerebral de organismos biológicos, donde su unidad más básica es una neurona.



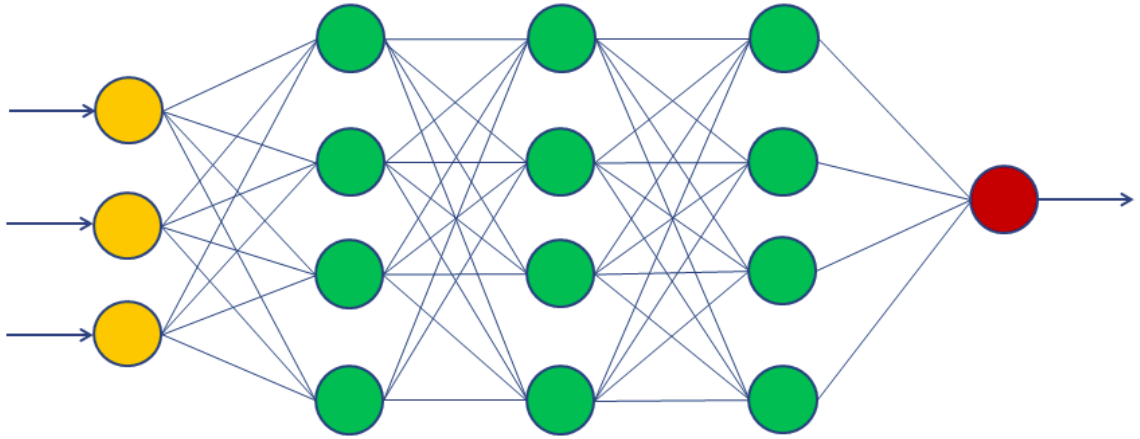
- Una neurona propaga los valores calculados de las neuronas de entrada a las neuronas de salida utilizando los pesos como parámetros intermedios.

$y_1, y_2, \ldots, y_m$

- El aprendizaje se produce cambiando los pesos, modificando la función calculada y haciendo que las predicciones sean más correctas en futuras iteraciones.

Introducción

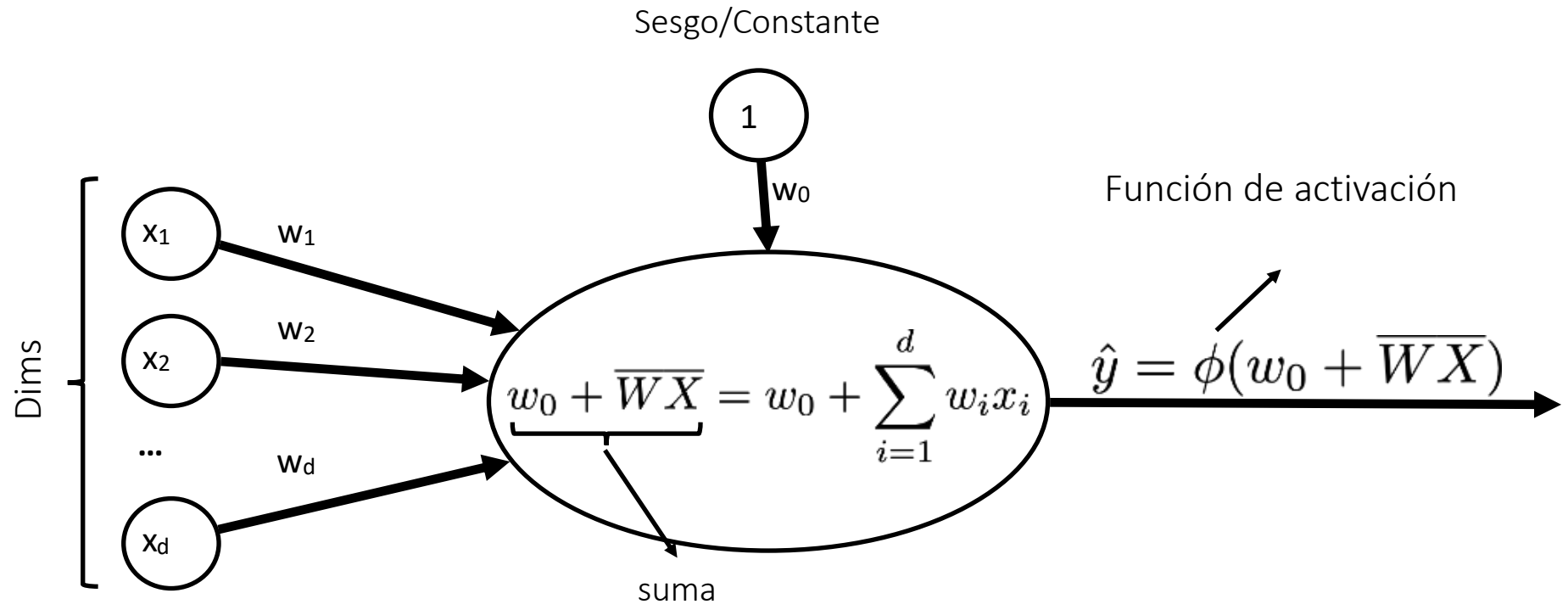
La red neuronal artificial (ANN) corresponde a múltiples neuronas conectadas entre ellas.



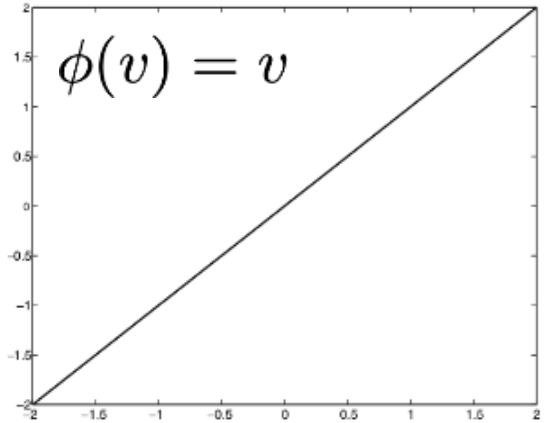
Perceptron

La red neuronal más simple es el perceptrón, donde un conjunto de entradas (b, x1, x2, ..., xd), donde **d** es la dimensionalidad de los datos, se asignan a una salida (y) utilizando un conjunto de ponderaciones (w0, w1, w2, ..., xd).

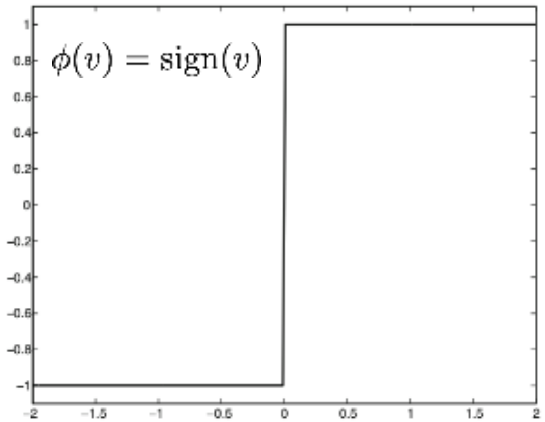
Una función lineal se calcula como un paso intermedio, dada su salida final a través de una función de activación ϕ(.).



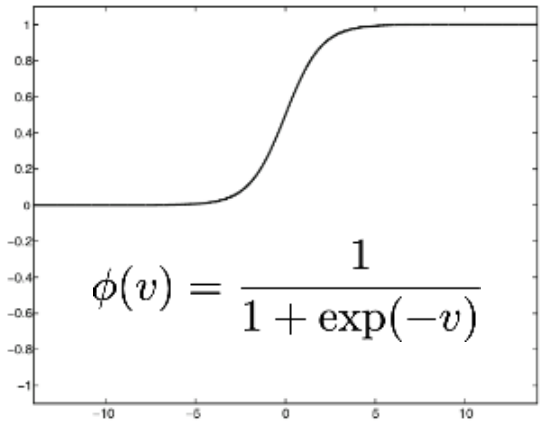
Funciones de activación populares



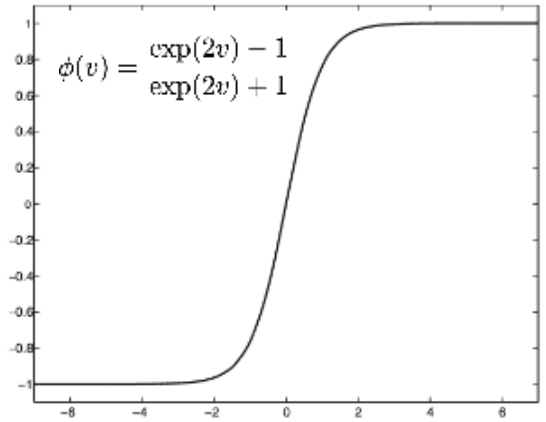
(a) Identity



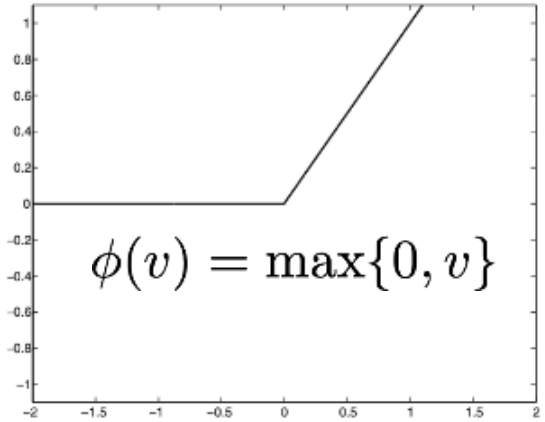
(b) Sign



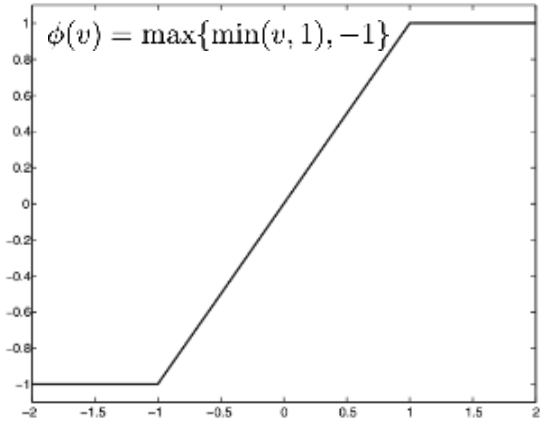
(c) Sigmoid



(d) Tanh

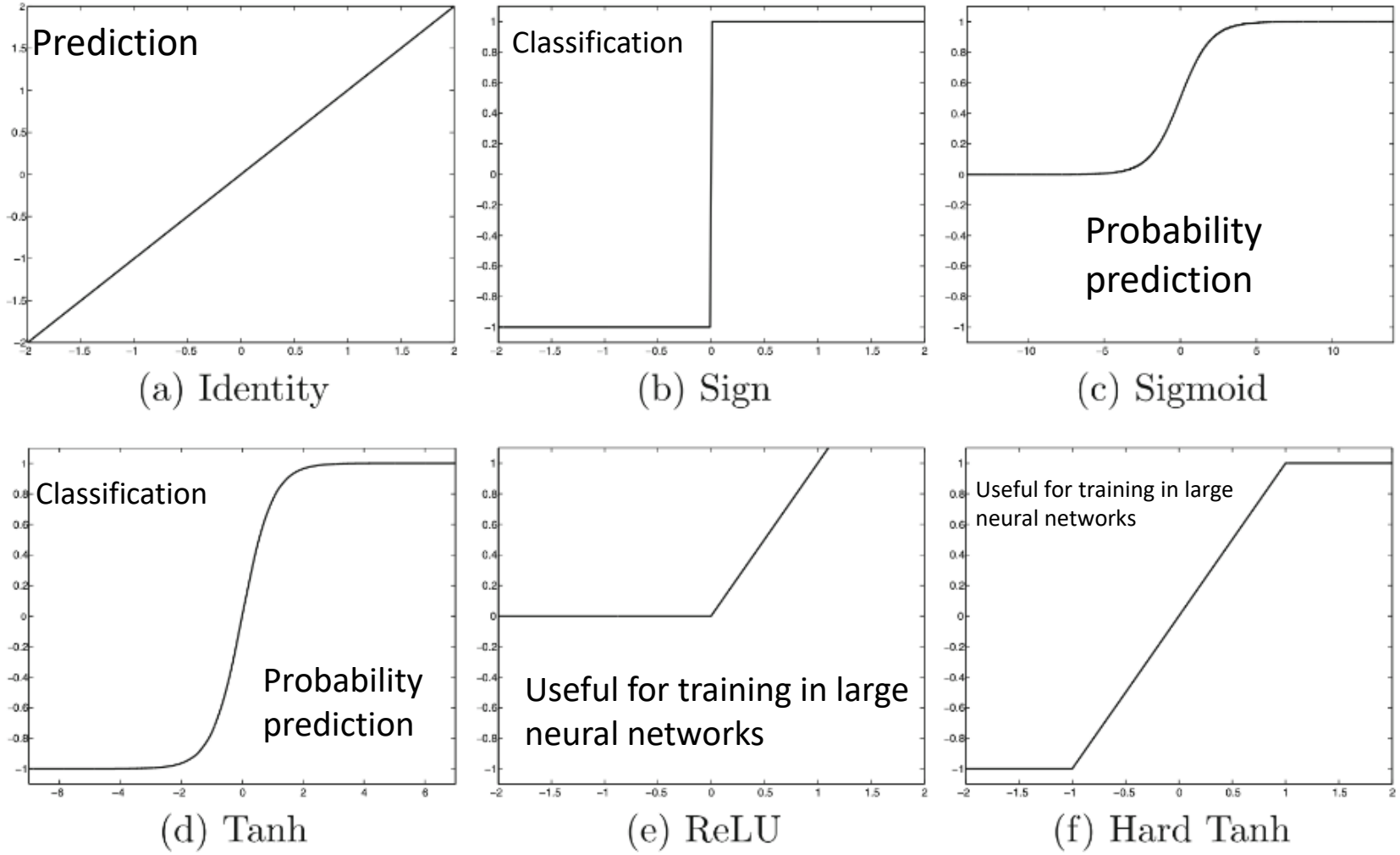


(e) ReLU



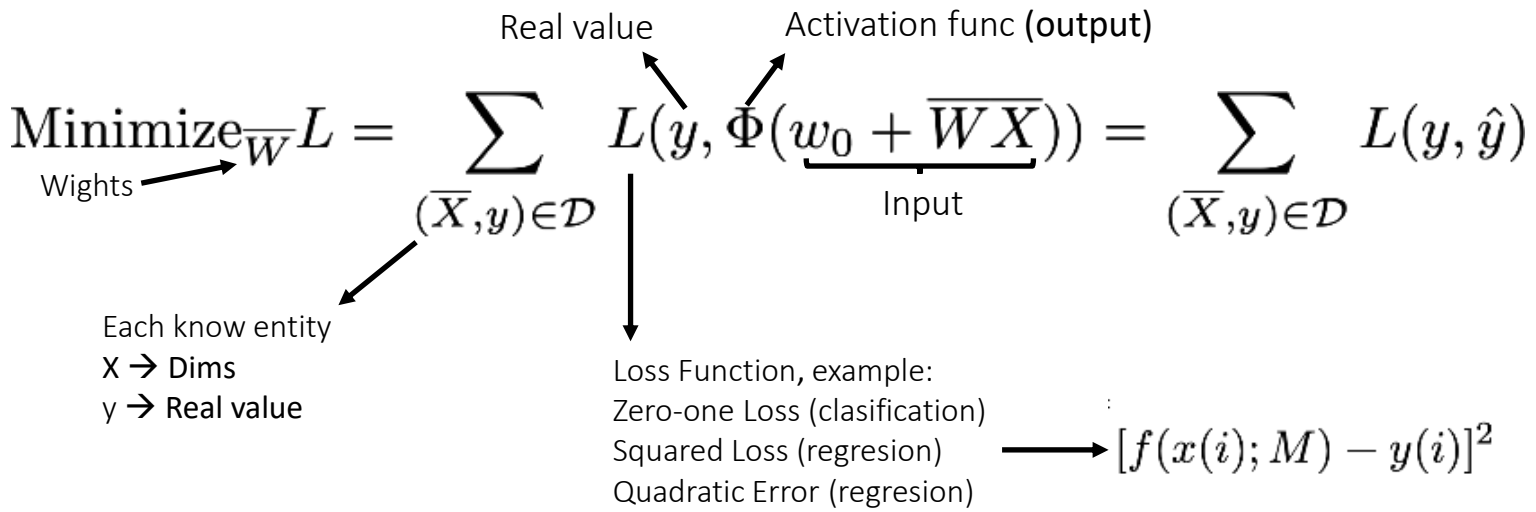
(f) Hard Tanh

Funciones de activación populares



Training

El entrenamiento es un proceso iterativo que minimiza una función de pérdida específica



- El algoritmo repasa repetidamente todos los ejemplos de entrenamiento en orden aleatorio y ajusta los pesos hasta que se alcanza la convergencia.

Training

$$\text{Minimize}_{\overline{W}} L = \sum_{(\overline{X},y)\in\mathcal{D}} L(y, \Phi(w_0 + \overline{WX})) = \sum_{(\overline{X},y)\in\mathcal{D}} L(y, \hat{y})$$

- En la mayoría de los casos, los pesos se actualizan a través de la actualización de descenso de gradiente secuencial:

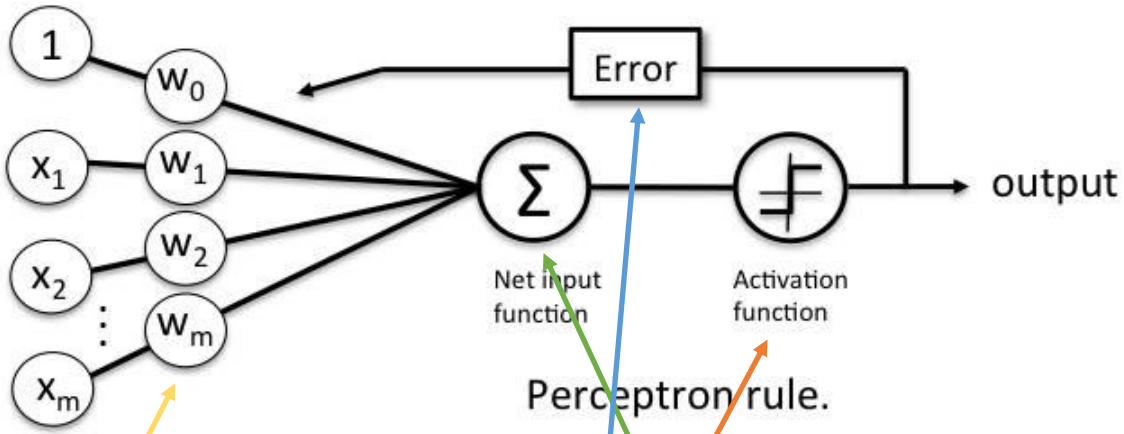
$$\overline{W}_{t+1} \Leftarrow \overline{W}_t - \alpha \nabla L$$

iteration

- α es el parámetro de aprendizaje.
- ∇L es el descenso del gradiente de la función de error, es decir, queremos movernos contra el gradiente, hacia el mínimo local.

Training

$$\overline{W}_{t+1} \Leftarrow \overline{W}_t - \alpha \nabla L$$



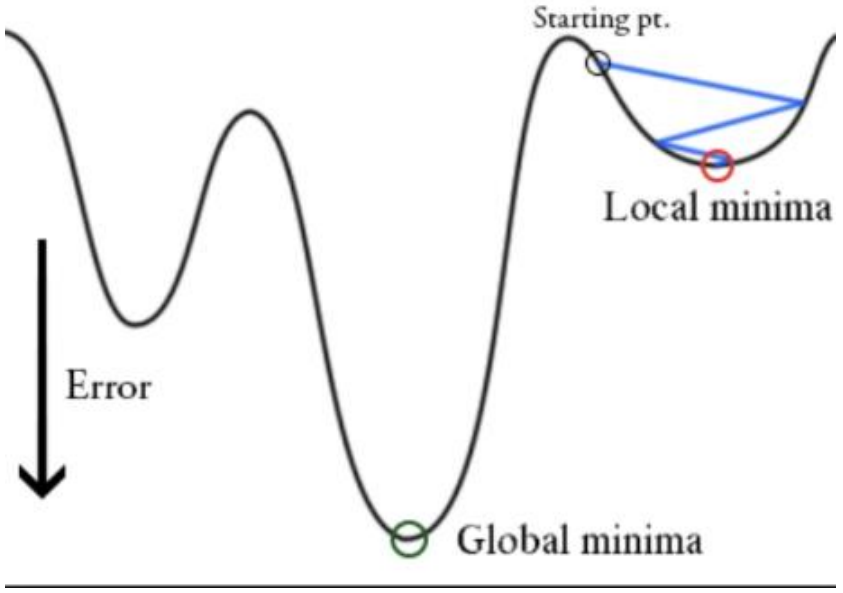
$$\text{Minimize}_{\overline{W}} L = \sum_{(\overline{X},y)\in\mathcal{D}} L(y, \Phi(w_0 + \overline{W}\overline{X})) = \sum_{(\overline{X},y)\in\mathcal{D}} L(y, \hat{y})$$

Algoritmo de optimización de descenso de gradiente

El proceso de entrenamiento se basa en el algoritmo de optimización del descenso del gradiente, es decir, los pesos de las neuronas se actualizan utilizando el gradiente de la función de pérdida.

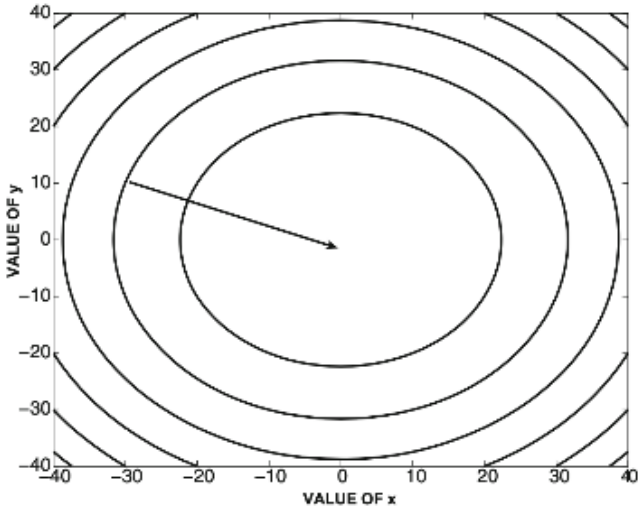
$$\overline{W}_{t+1} \Leftarrow \overline{W}_t - \alpha \nabla L$$

- El gradiente de la función de pérdida describe la pendiente, es decir, la dirección que tenemos que utilizar para minimizar el error.
- α representa el parámetro de aprendizaje (el impacto del gradiente en generar la nueva W).

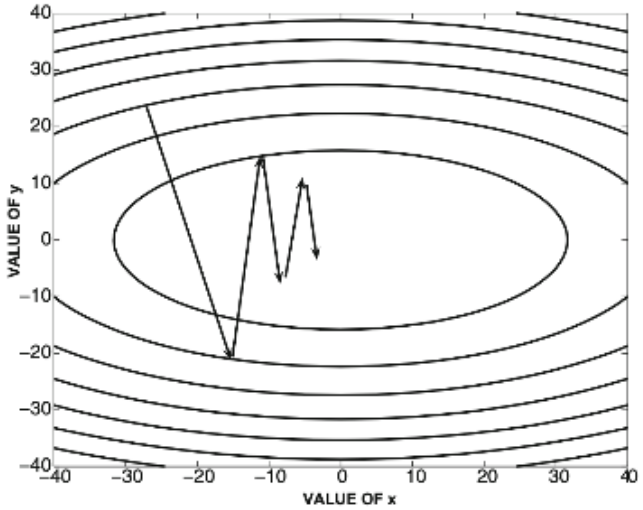


Algoritmo de optimización de descenso de gradiente

Se necesitan varios pasos, porque, solo calculamos la dirección de la minimización, y esto no necesariamente apunta al punto mínimo.
El espacio son las combinaciones de pesos posibles



(a) Loss function is circular bowl
$L = x^2 + y^2$



(b) Loss function is elliptical bowl
$L = x^2 + 4y^2$

Escenario perfecto, el punto de dirección puesto al mínimo de la función

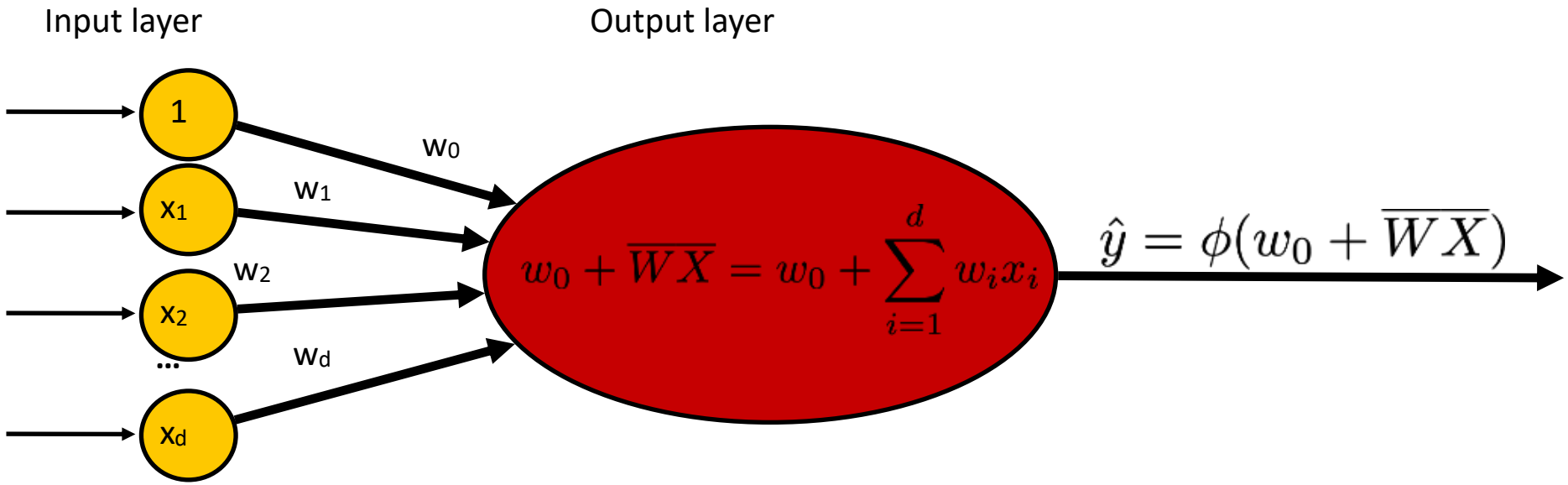
Escenario normal, la dirección apunta a una reducción, pero a la mínima

Capas

El Perceptrón se considera una red de dos capas, donde tenemos una capa de entrada y una capa de salida.

La capa de entrada no hace nada, recibe los datos y los da a la capa de salida. Su tamaño corresponde, normalmente, a la dimensión de los datos.

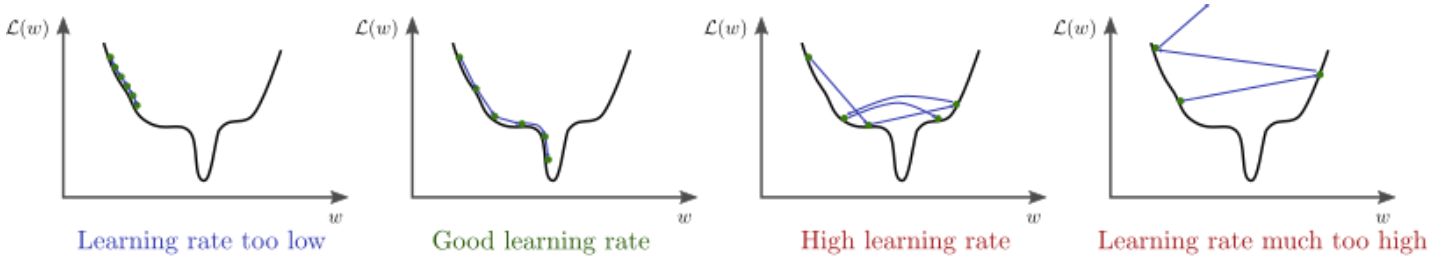
La capa de salida corresponde al perceptrón él mismo, donde la suma y la función de activación se aplican para la salida final.

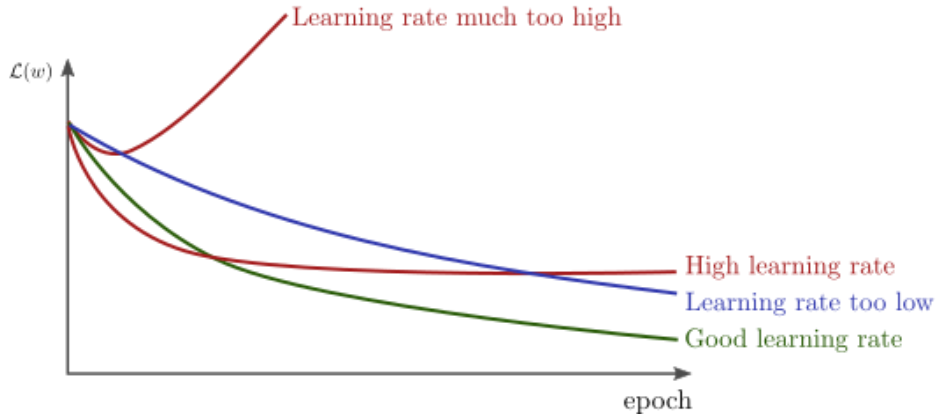


Aleatoriedad

$$\overline{W}_{t+1} \Leftarrow \overline{W}_t + \alpha(y - \hat{y})\overline{X}$$

El inicio del camino y los saltos van a estar dados por la aleatoriedad en la selección de entidades para mejorar el W en cada iteración y el valor de Alpha

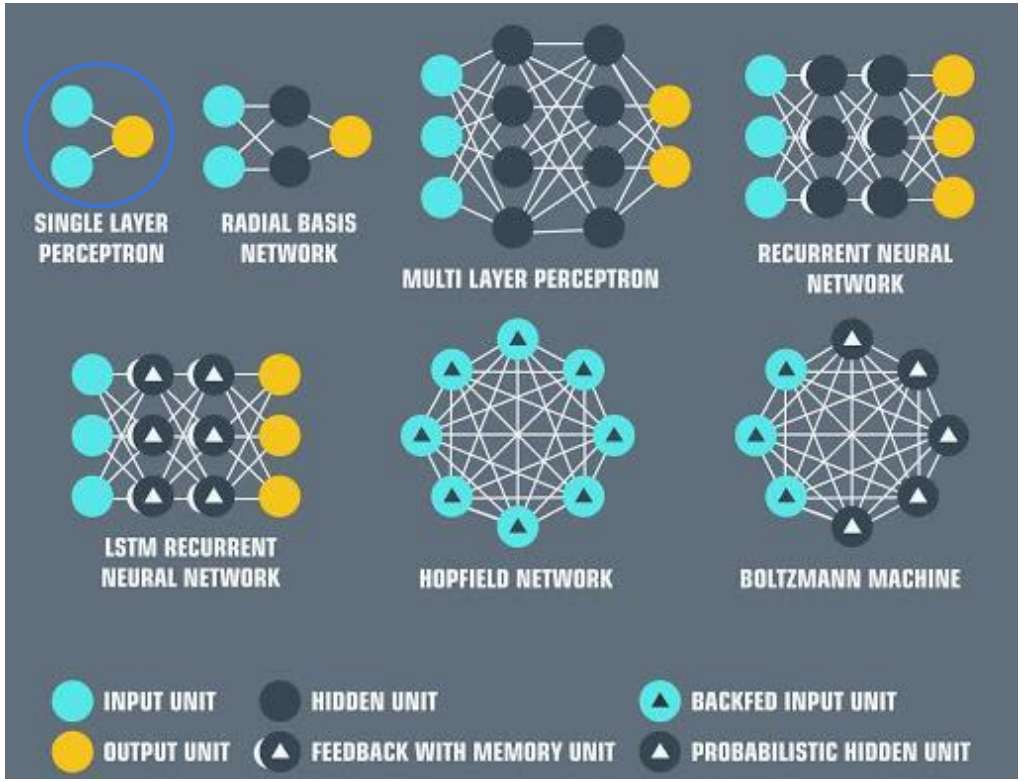




Redes neuronales multicapa

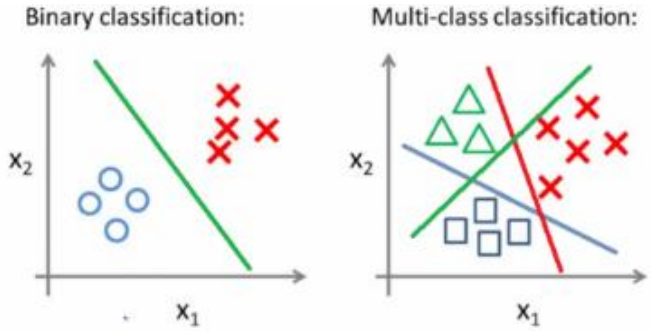
Las redes neuronales se volvieron poderosas, una vez que fueron capaces de conectar múltiples neuronas en diferentes arquitecturas, cada una de ellas con sus propios beneficios.

- La arquitectura de una red corresponde al número de neuronas, capas y conexión entre neuronas.
- Cada arquitectura podría tener un algoritmo de entrenamiento diferente.

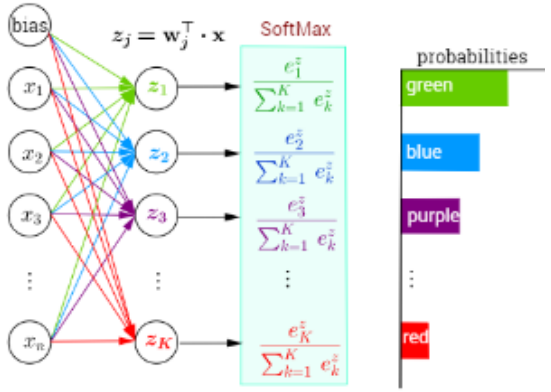


Redes neuronales multicapa

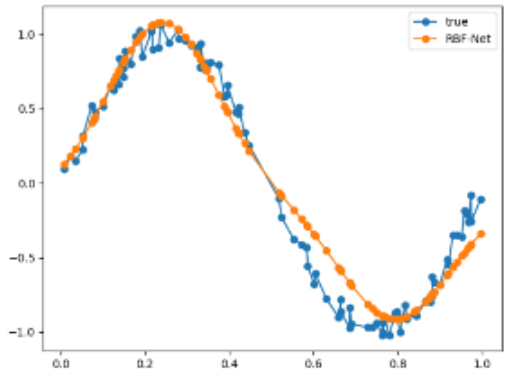
Las redes no solo pueden hacer clasificaciones binarias (0 o 1) (azul rojo); sino también multiclases (1, 2, 3 o 4) (perro, gato, panda, caballo)

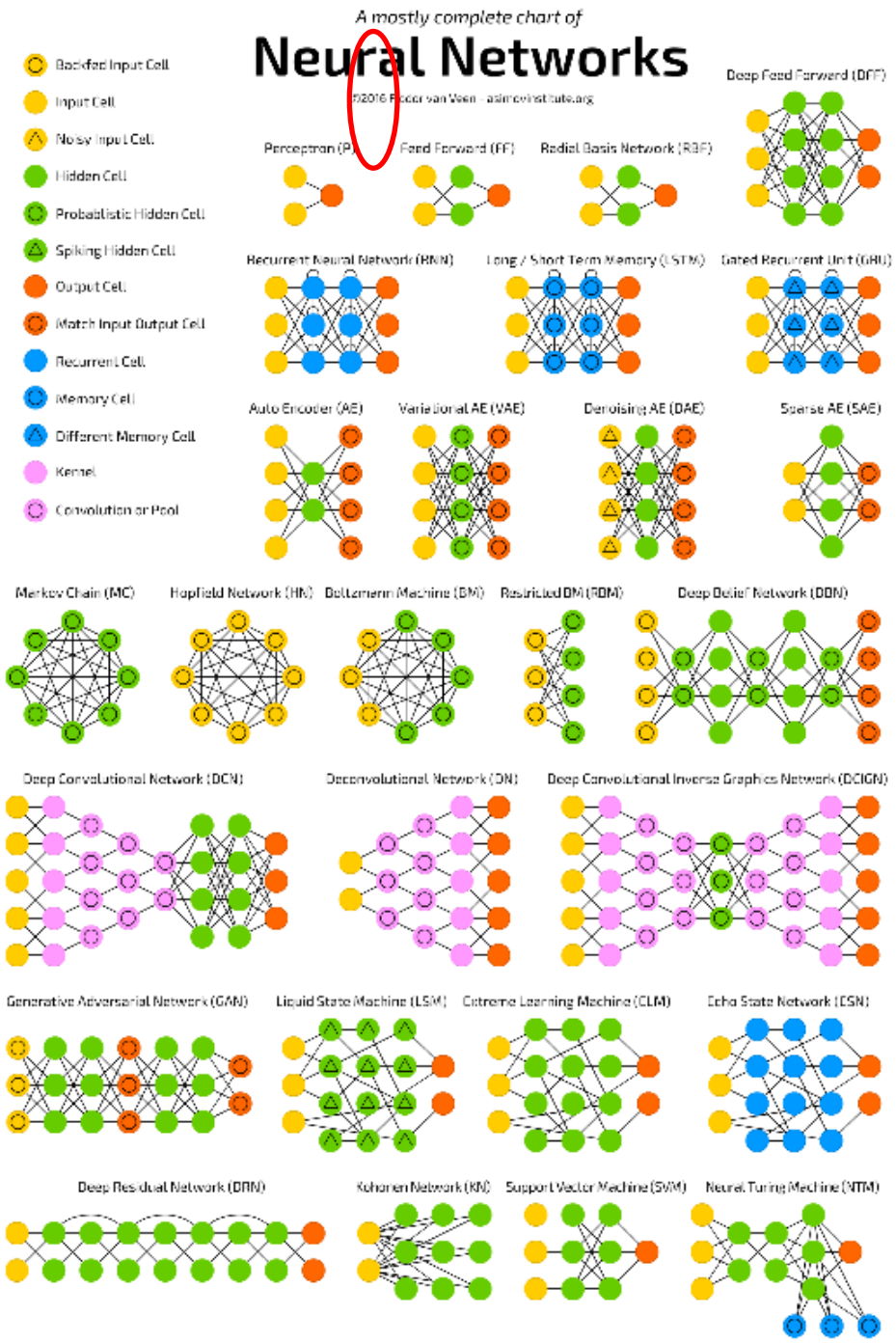


Podemos configurar las neuronas para que el output no sea una clasificación discreta (0 o 1), sino una probabilidad de pertinencia (20%, 80%). Esto para clases binarias o múltiples.



Las redes también pueden estar diseñadas para hacer regresiones (predecir números continuos, ej: temperatura dado un día).

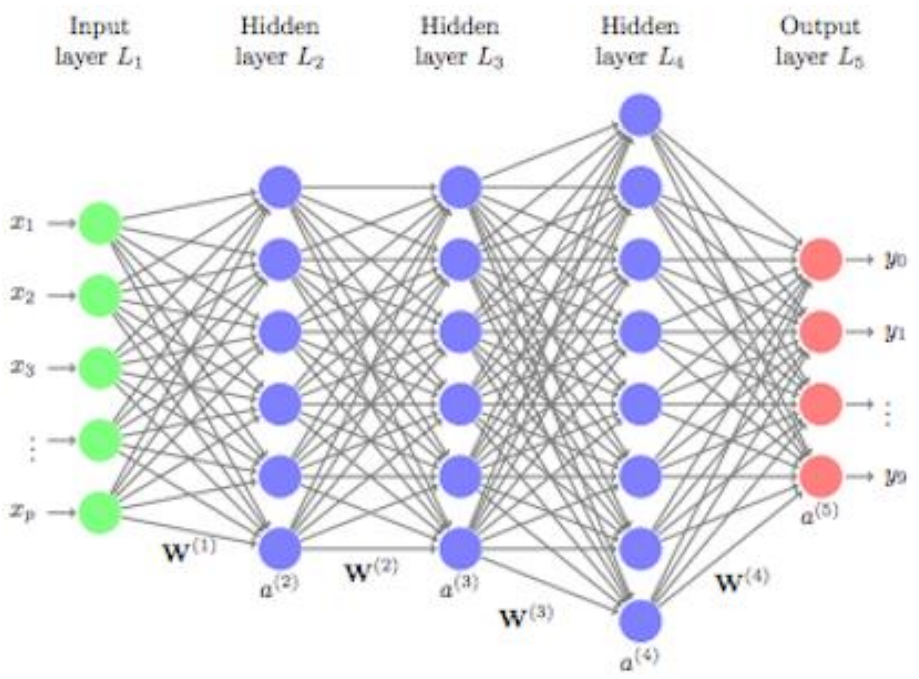


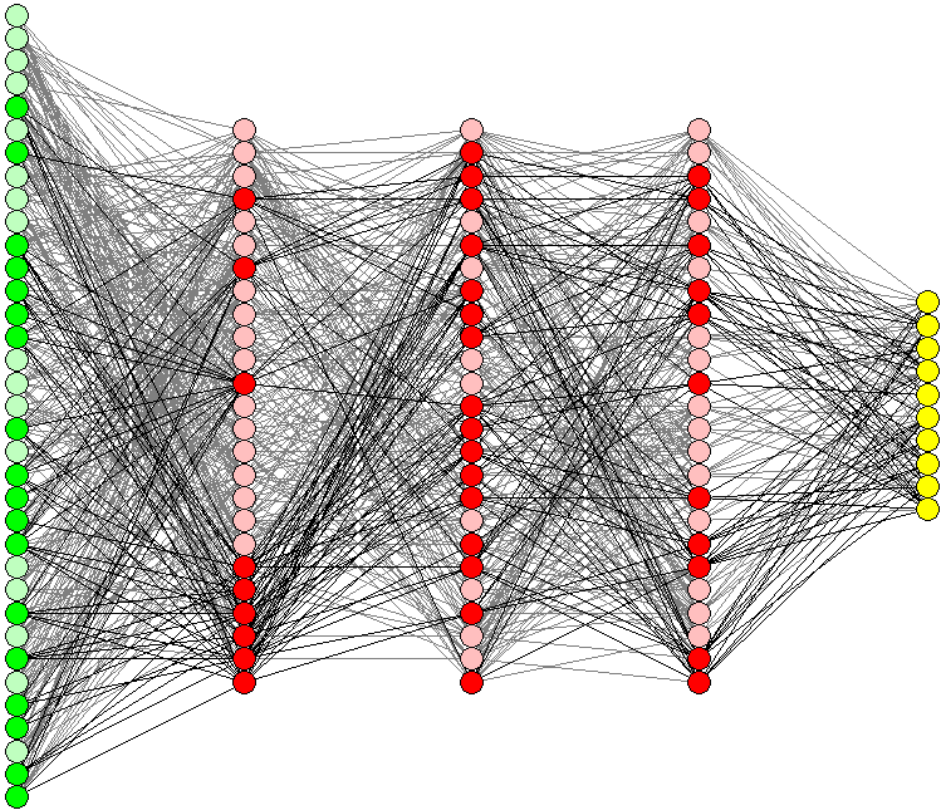


Cada arquitectura tiene su fortalezas, debilidades, formas de entrenamiento y algoritmos asociados.

Feed forward networks

- Una de las arquitecturas más populares es el feed-forward, donde las neuronas de cada capa están conectadas a todas las neuronas de la siguiente capa.
- Esta arquitectura resolvió el problema del perceptrón, ya que la adición de más neuronas y capas ocultas crea una red muy poderosa.
- Desafortunadamente, las redes poderosas eran imposibles de entrenar (establecer todo el peso en un buen valor) en términos de datos (se necesitan más datos para evitar el sobreajuste) y complejidad de tiempo.
- Estos problemas fueron "resueltos", por la enorme cantidad de datos, avances informáticos y algoritmos de entrenamiento paralelizables, lo que llevó al aprendizaje profundo.



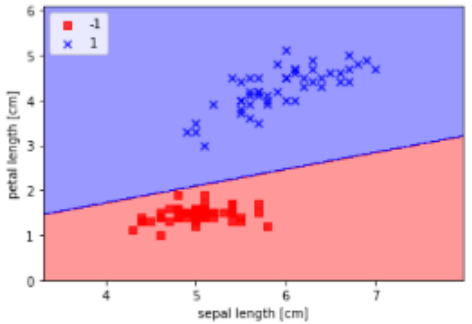


Para cada neurona existen pesos asociados a su input, funciones de activaciones especificas.

Para cada red existe una dimensionalidad de input, una cantidad de datos conocidos y un objetivo.

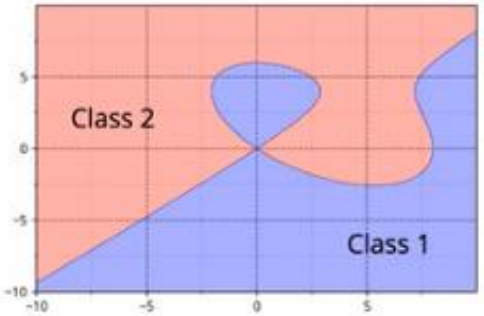
Para cada red existe una arquitectura, algoritmos para entrenar y loss func **objetivo.**

Perceptron (P)



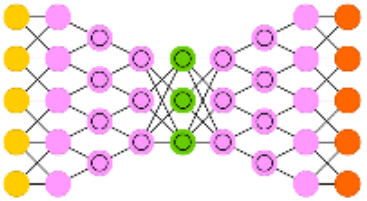
Un perceptrón no es mas que un generador de un hiperplano lineal.

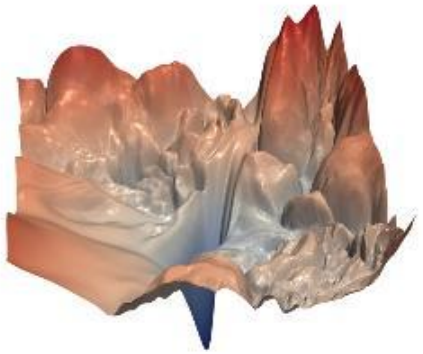
Deep Feed Forward (DFF)



Agrupar neuronas es dar la posibilidad de generar funciones que dividan el espacio, que tengan formas mas caprichosas.

Deep Convolutional Inverse Graphics Network (DCIGN)





Redes mas complejas pueden encontrar funciones que modelen problemas sumamente inauditos.

Mientras mas compleja una red → Mas datos y poder de computo necesitamos para evitar poco o sobre entrenamiento

Mientras mas compleja una red →Menos comprendemos el “porque” de la función resultante generada

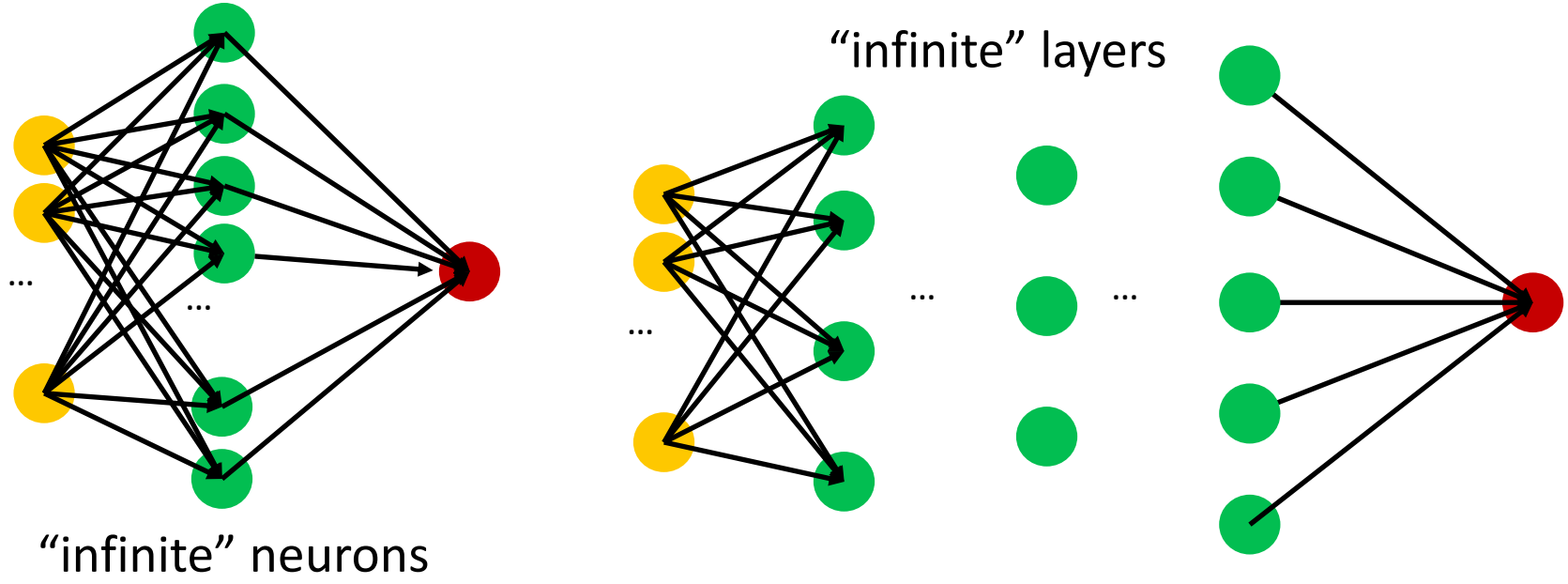
Deep

Arquitectura

La red multicapa se denomina aproximador de función universal, lo que significa que es capaz de reproducir casi cualquier función razonable.

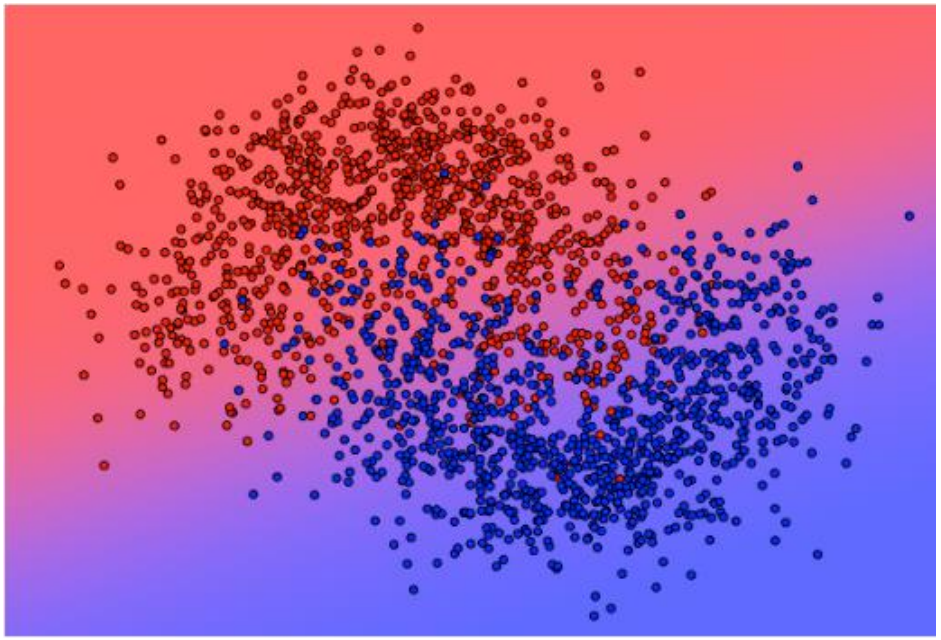
Para este propósito, solo necesitamos una red con una sola capa oculta de unidades no lineales y una sola capa de salida.

Las redes más profundas pueden realizar el mismo trabajo aumentando el número de capas ocultas, pero reduciendo el número de parámetros, mejorando el entrenamiento.

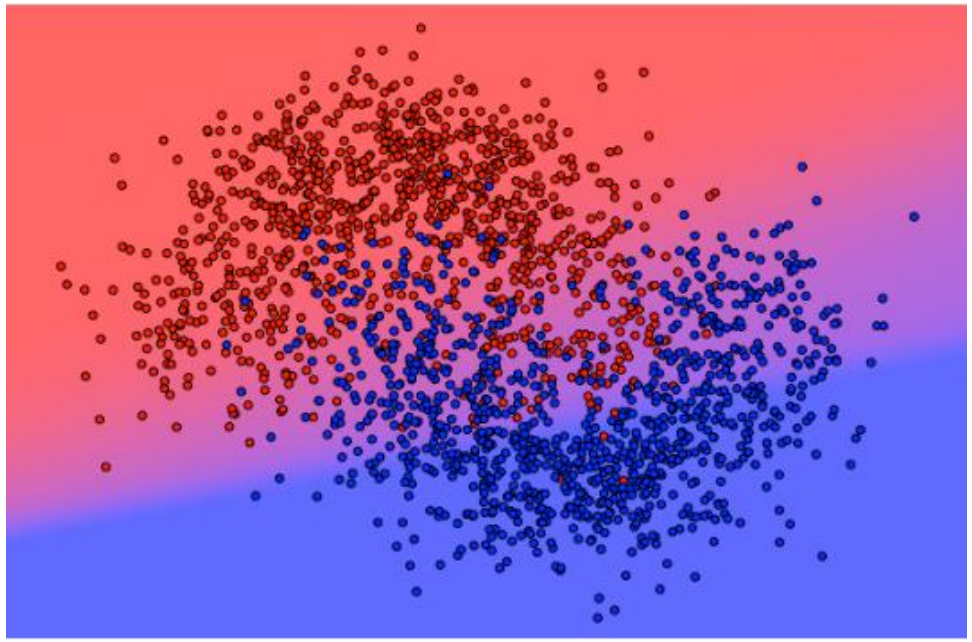


Arquitectura

Otra ventaja importante del feed forward y el uso de múltiples capas, es su posibilidad de generar cualquier tipo de límite.



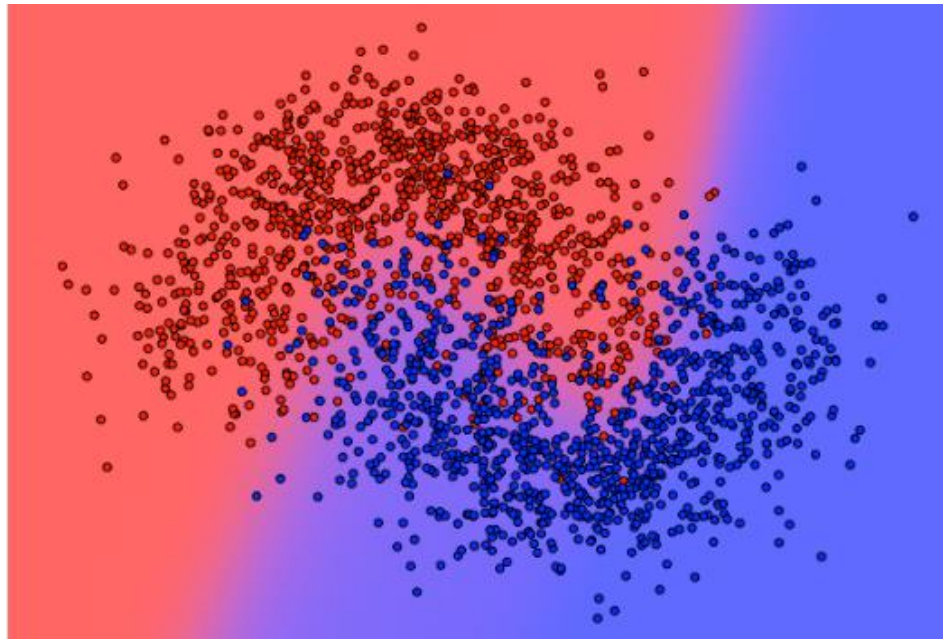
Perceptron



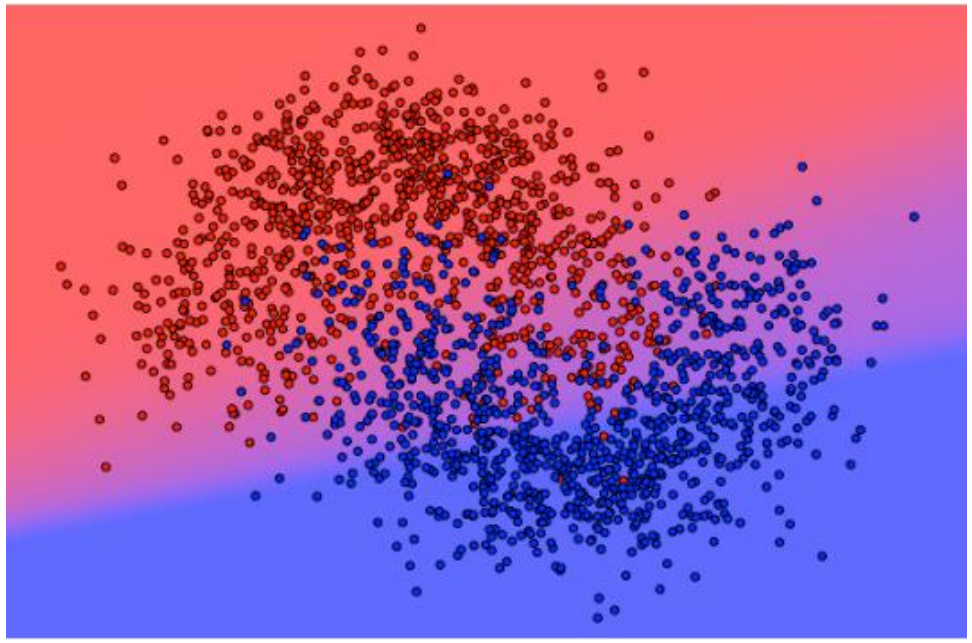
Feed forward
2 neurons in hidden layer

Arquitectura

Otra ventaja importante del feed forward y el uso de múltiples capas, es su posibilidad de generar cualquier tipo de límite.



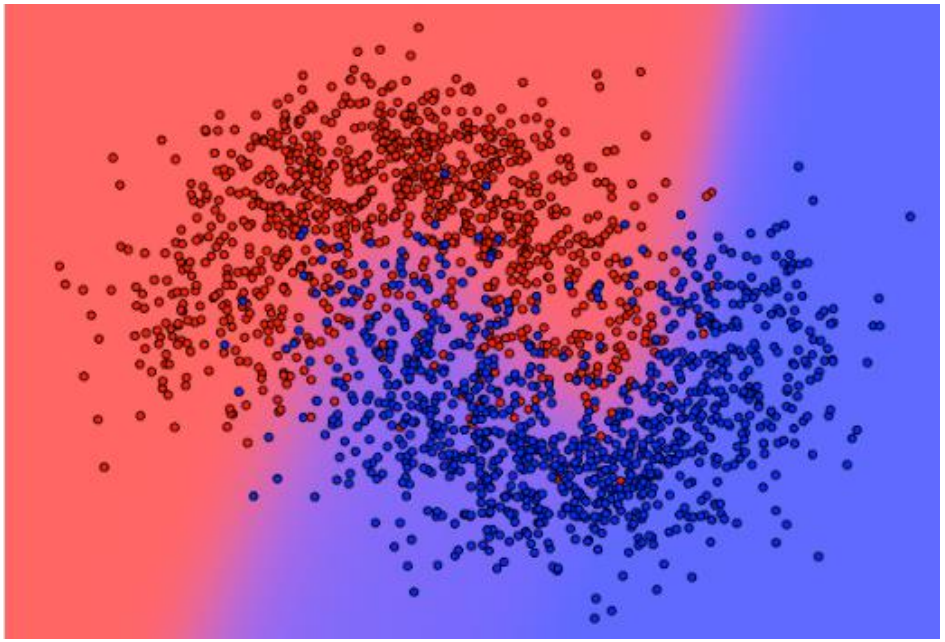
Feed forward
3 neurons in hidden layer



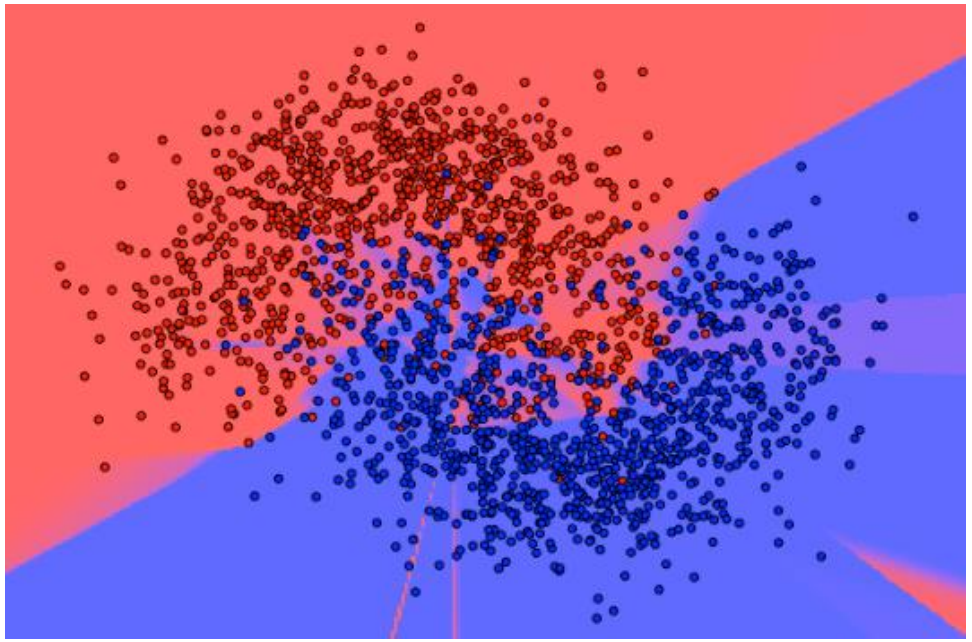
Feed forward
2 neurons in hidden layer

Arquitectura

Otra ventaja importante del feed forward y el uso de múltiples capas, es su posibilidad de generar cualquier tipo de límite.



Feed forward
3 neurons in hidden layer



Feed forward
20 neurons in hidden layer
OVERFITTING

Sobreajuste

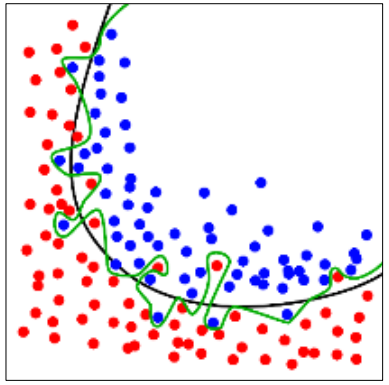
Regla general, los puntos de datos deben ser al menos de 2 a 3 veces mayores que el número de parámetros de la red neuronal (NN). Sin embargo, incluso con una gran cantidad de datos, un NN requiere un diseño cuidadoso para minimizar el sobreajuste.

Otras técnicas para evitar el sobreajuste son:

Detención temprana: detenga el proceso de aprendizaje después de algunas iteraciones.

Regularización: penaliza la función de puntuación proporcionalmente al tamaño de los parámetros.

Arquitectura: Un gran número de capas tiende a reducir el número de neuronas en cada capa, reduciendo el número de parámetros.



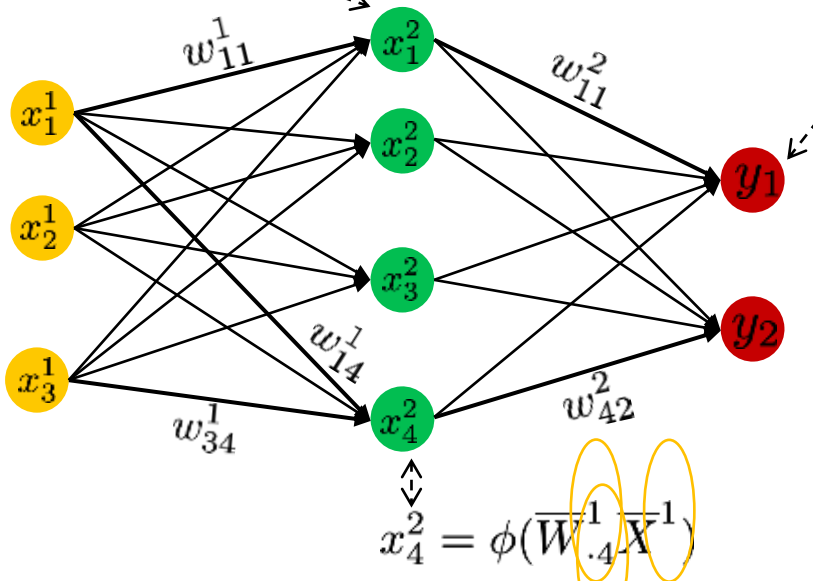
Algoritmo Back-propagation

El FFN se entrena con el algoritmo de propagación inversa, que utiliza el algoritmo de optimización de descenso de gradiente (al igual que el perceptrón) para entrenar los pesos.

Problema: ¿Cómo ajustar todos los pesos? ¿Qué orden?

Solución: Inicie la generación de una salida (error), luego desde las últimas capas propague la corrección a las capas iniciales (propagación inversa).

$$x_1^2 = \phi(\overline{W}_{.1}^1 \overline{X}^1) = \phi\left(\sum_{j=1}^{3} w_{j1} x_j^1\right)$$



$$x_4^2 = \phi(\overline{W}_{.4}^1 \overline{X}^1)$$

$$y_1 = \phi(\overline{W}_{.1}^2 \overline{X}^2) = \phi\left(\sum_{i=1}^{4} w_{i1}^2 x_i^2\right)$$

$$= \phi\left(\sum_{i=1}^{4} w_{i1}^2 \phi\left(\overline{W}_{.i}^1 \overline{X}^1\right)\right)$$

$$= \phi\left(\sum_{i=1}^{4} w_{i1}^2 \phi\left(\sum_{j=1}^{3} w_{ji}^1 x_j^1\right)\right)$$

20 pesos distintos + 6 bias (depende de arquitectura)

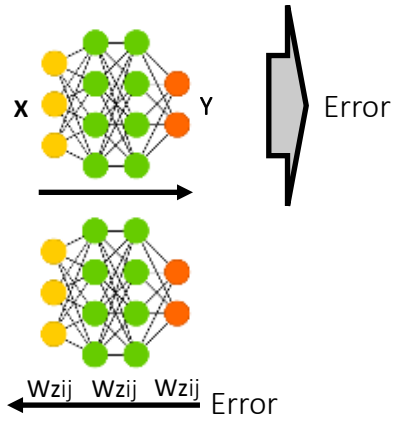
¿En que orden?

Algoritmo Back-propagation

La retropropagación se puede separar en dos fases:

Fase de reenvío: Dado un punto de datos de entrenamiento como entrada a la red, se realizan todos los cálculos, obteniendo su salida final (generando un error).

Fase hacia atrás: Dado el error obtenido en la fase de avance actualizar los pesos para minimizar el error mediante el algoritmo de optimización de descenso de gradiente.



- Recall: $\text{Minimize}_{\overline{W}} L = \sum_{(\overline{X},y)\in\mathcal{D}} \sum_{j=1}^{outputs} L(y_j, \hat{y}_j)$ $\qquad \overline{W}_{t+1} \Leftarrow \overline{W}_t - \alpha \nabla L$

- **Problema: ∇L(wi) no es simple como antes**

- **Solución: actualizar la capa externa y luego dentro de las capas**

Deep learning

Para modelar más patrones, el número de neuronas que necesitan crece exponencialmente. Hacer que los algoritmos de entrenamiento no se den éxito debido a los datos y el tiempo.

El aprendizaje profundo se refiere a algoritmos de entrenamiento para redes neuronales con múltiples capas y neuronas que también se entrenan de manera distribuida.

La arquitectura y el proceso de formación difieren según el problema Para la extracción de características, el aprendizaje no supervisado y el reconocimiento de patrones, debe usar entre RBM y Autoencoders.

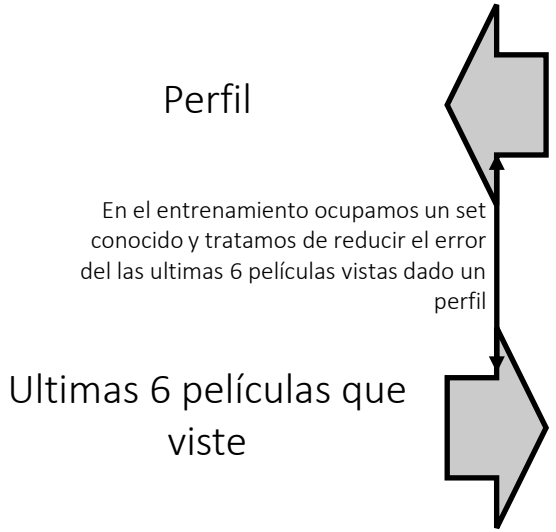
Para la clasificación, depende del problema:
Clasificación: **FNN (también conocido como MLP), DBN**
Procesamiento de texto: **RNTN, RNN**
Reconocimiento de imágenes: **DBN, CNN**
Reconocimiento de objetos: **RNTN, CNN**
Reconocimiento de voz: **RNN**
Serie temporal: **RNN**

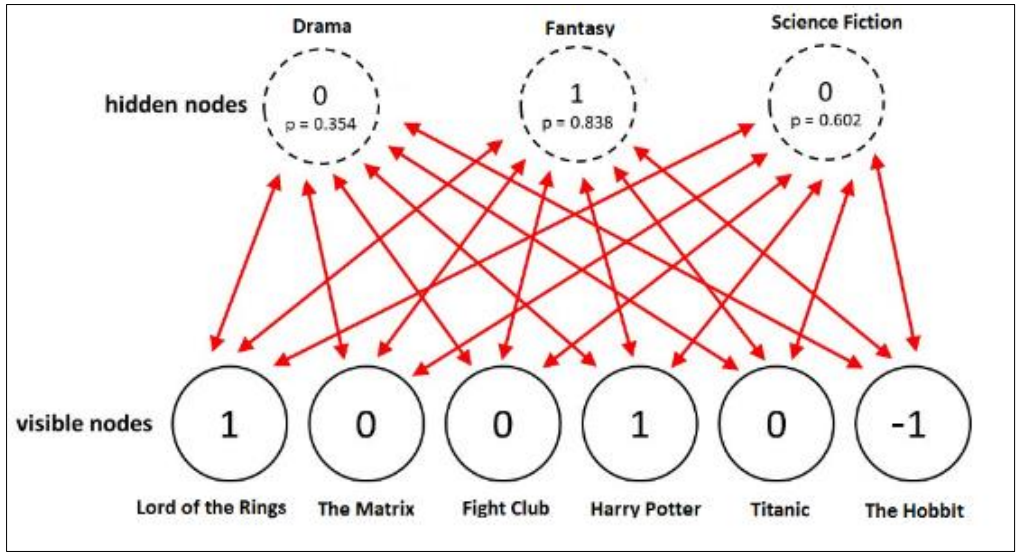
Restricted Boltzmann Machine (RBM)

El RBM puede reconstruir automáticamente el patrón a partir de los datos de entrada (uso para la extracción de características, el aprendizaje no supervisado y el reconocimiento de patrones; NO para el modelado predictivo supervisado).

El RBM es solo una red de 2 capas donde la capa de entrada está conectada a una capa oculta.
Dada una entrada, se calcula la salida, luego dada esa salida la entrada intenta reconstruirse, generando un error con respecto a la entrada original.

El proceso de entrenamiento repite este proceso varias veces, ajustando los pesos para replicar la entrada original, donde los pesos aprendieron los patrones de los datos.





Autoencoders

Los autoencoders se utilizan para el aprendizaje no supervisado, detectando patrones en los datos.

Por ejemplo, el RBM es un tipo de autoencoder (uso para la extracción de características, el aprendizaje no supervisado y el reconocimiento de patrones; NO para el modelado predictivo supervisado).

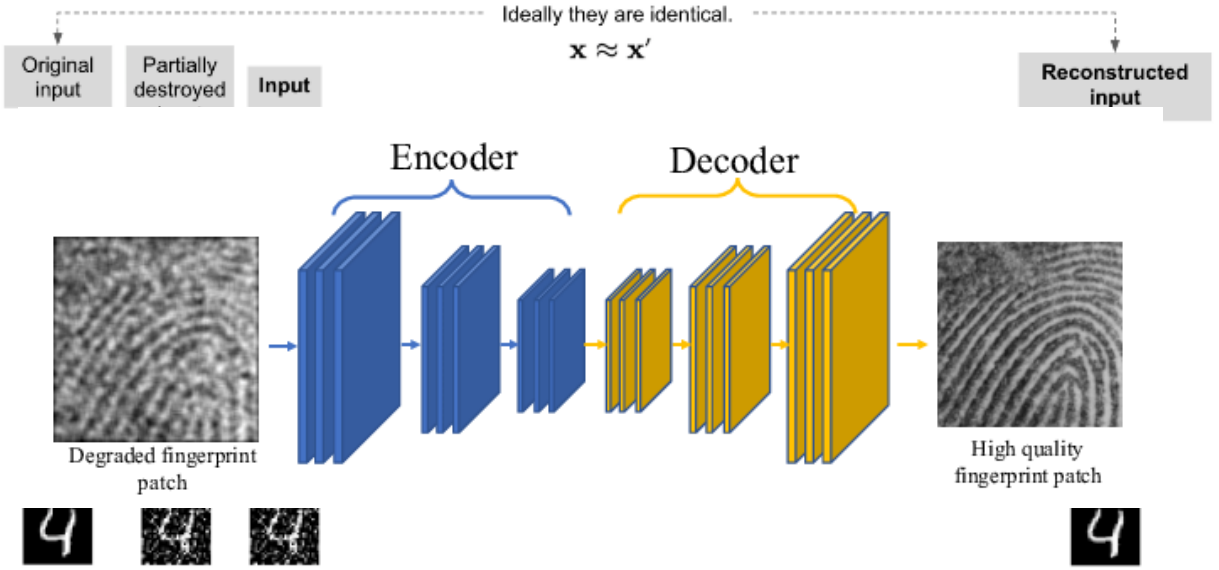
La arquitectura suele ser una red de capa de árbol (entrada, oculta, salida), pero el proceso codifica la entrada y, a continuación, descodifica la salida.

El autoenconder profundo utiliza varias capas ocultas.
Autoenconder supera significativamente a la PCA.

La "zona" encoder, extrae y comprime patrones. La "zona" decoder, convierte esos patrones en la información original.

Usamos información sin ruido y le agregamos ruido de forma intencional. Luego entrenamos la red para que genere la información original sin el radio.

Con el modelo entrenados, nos puede llegar un dato desconocido con ruido y podemos reconstruir el dato sin el ruido.



Deep Belief Net (DBN)

El DBN tiene la misma arquitectura de una red de avance y se utiliza para la clasificación, pero con un algoritmo de entrenamiento diferente (aprendizaje supervisado). Clasificación, Reconocimiento de imágenes

El pre-entrenamiento de la DBN es como una pila de RBM (Restricted Boltzmann Machine), donde las dos primeras capas se entrenan como RBM, luego la 2ª y 3ª capa, luego la 3ª y 4ª, y continúan hasta el final.

El post-entrenamiento introduce la etiqueta requerida, asociando las etiquetas a los pesos aprendidos anteriormente

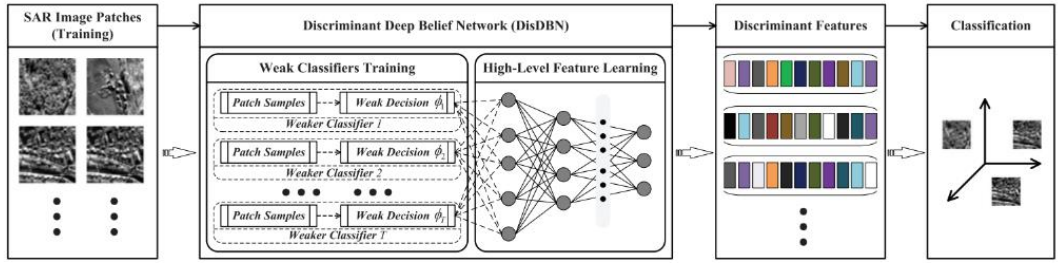
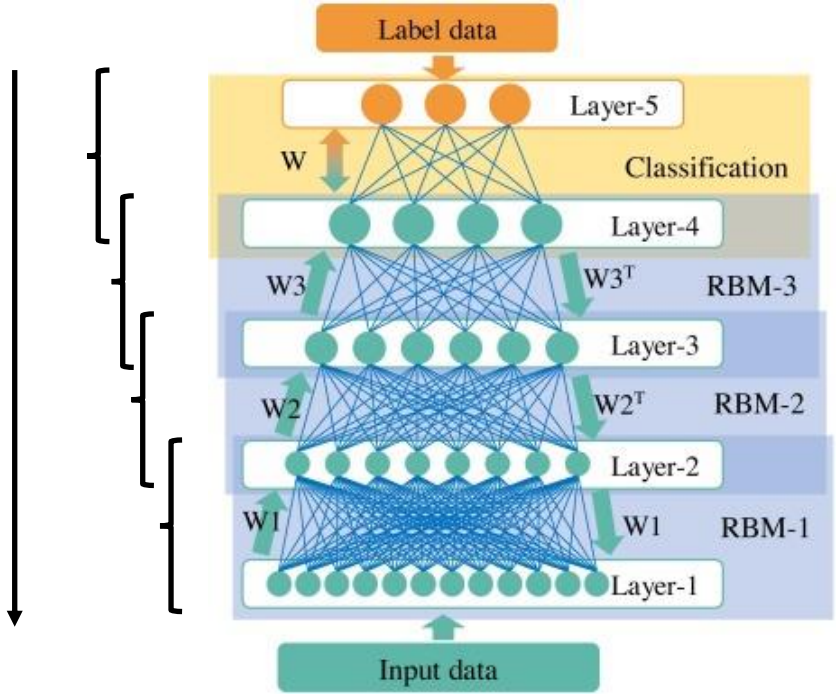


Fig. 1. Pipeline of the proposed discriminative feature learning approach (DisDBN). It consists of two parts, which are weak classifier training and followed by a deep belief network (DBN) for high-level feature learning.

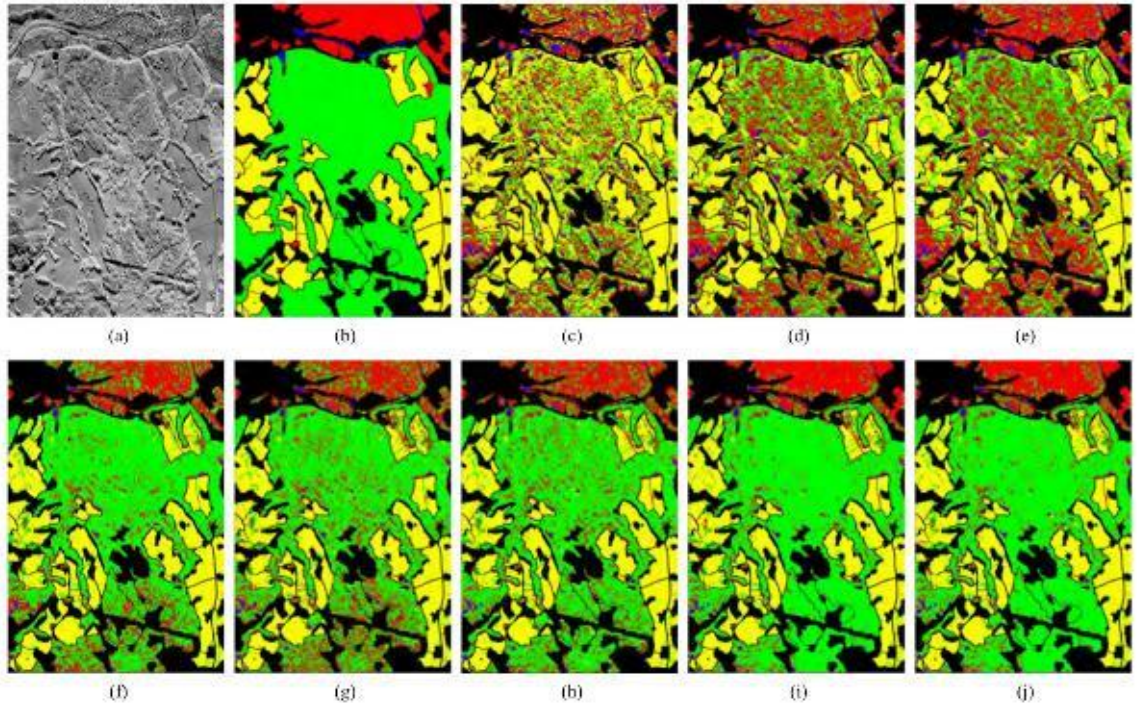


Es una combinación de FFN (la red que vimos durante la primera parte) + los beneficios de RBM.

Entrenamos de manera no supervisada cada capa de RBM de manera escalonada (preentrenamiento). Pueden ser N niveles de RBM.

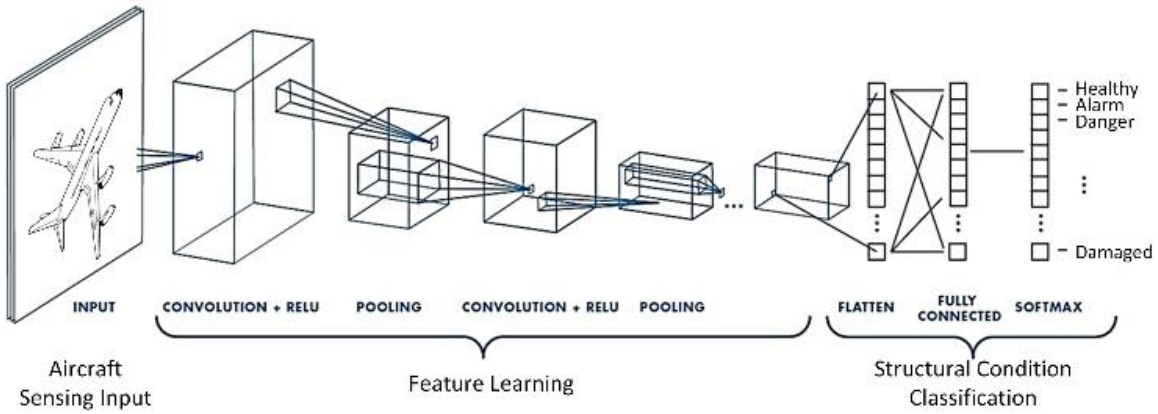
En la ultima capa incluimos las clases a clasificar (supervisado) y realizamos el backpropagation.

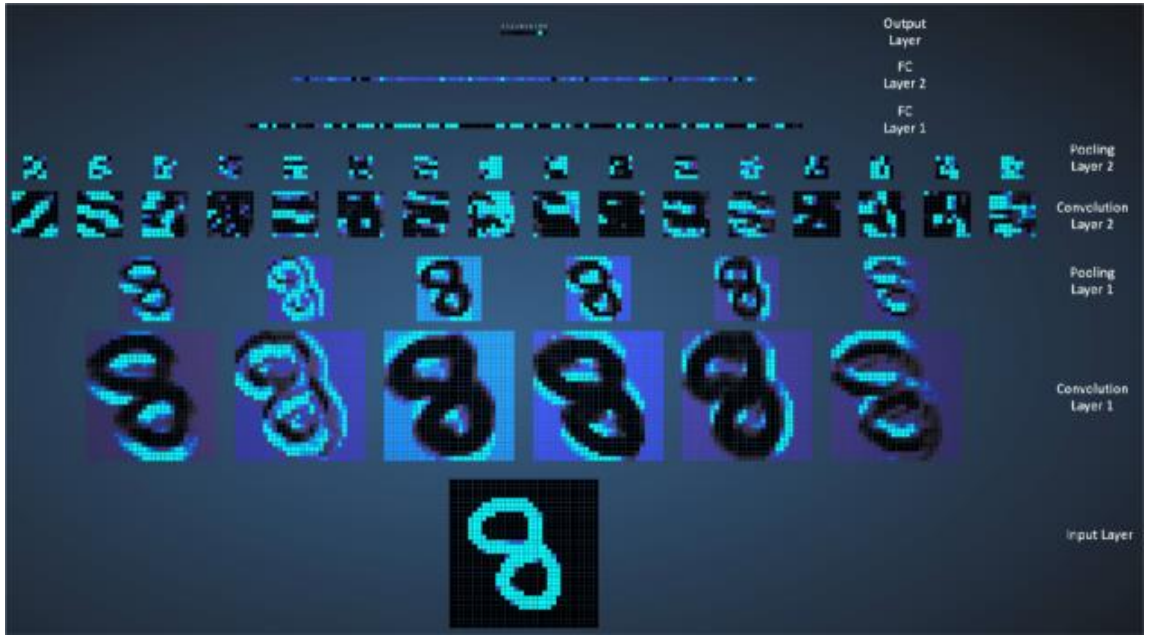
Las RBM son capaces de extraer patrones y representarlos en otra dimensionalidad; al final de la red, ocupamos esa abstracción de dimensionalidad para clasificar.



Convolutional Neural Network (CNN)

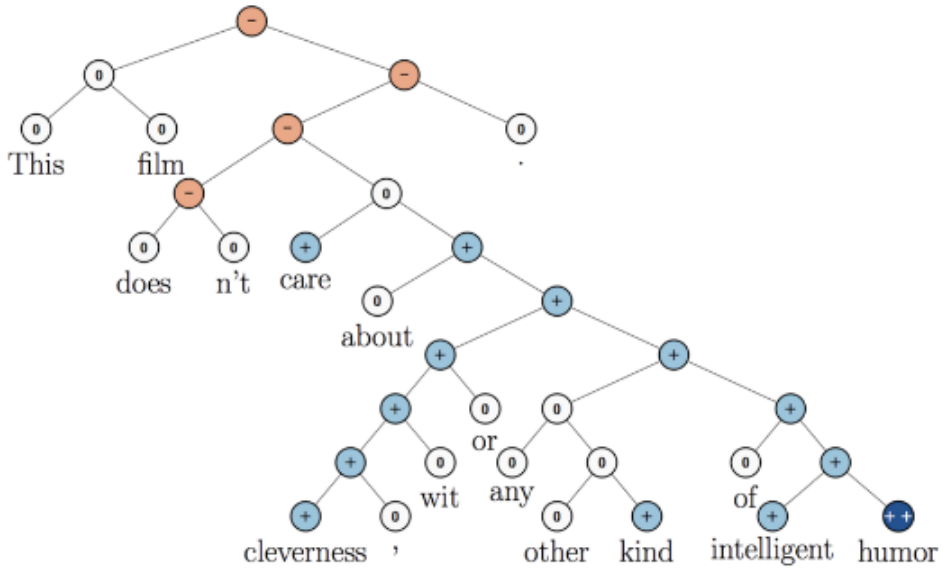
La CNN es una de las mejores herramientas disponibles para la visión artificial que combina varias veces la capa convolucional, RELU y pooling. Reconocimiento de imágenes, Reconocimiento de objetos
La parte convolucional aplica filtros a las características de extracción de datos. En concreto, cada filtro corresponde a una sola capa.
Las siguientes capas son RELU (proceso de aprendizaje) y Pooling (reducción de dimensionalidad). Ambas capas aprenden los patrones de los datos, pero sin una etiqueta.
Finalmente, una capa totalmente conectada está conectada al final agregando las etiquetas.

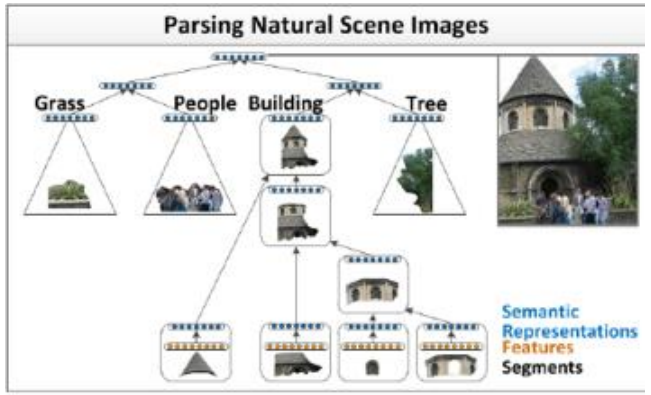


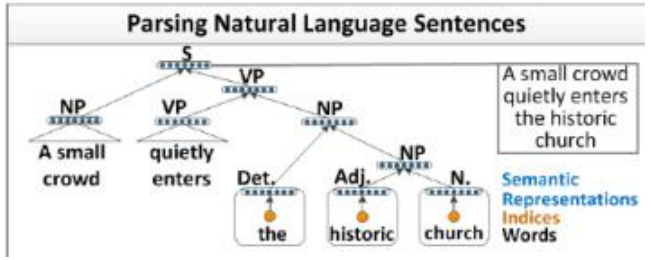


Recursive Neural Tensor Net (RNTN)

El RNTN detecta el patrón jerárquico en los datos. Se utilizan principalmente para analizar texto y analizar imágenes.
La arquitectura es como un árbol, donde los nodos del árbol corresponden a grupos de neuronas, las hojas reciben las entradas, mientras que la raíz genera la salida.
Procesamiento de texto, reconocimiento de objetos



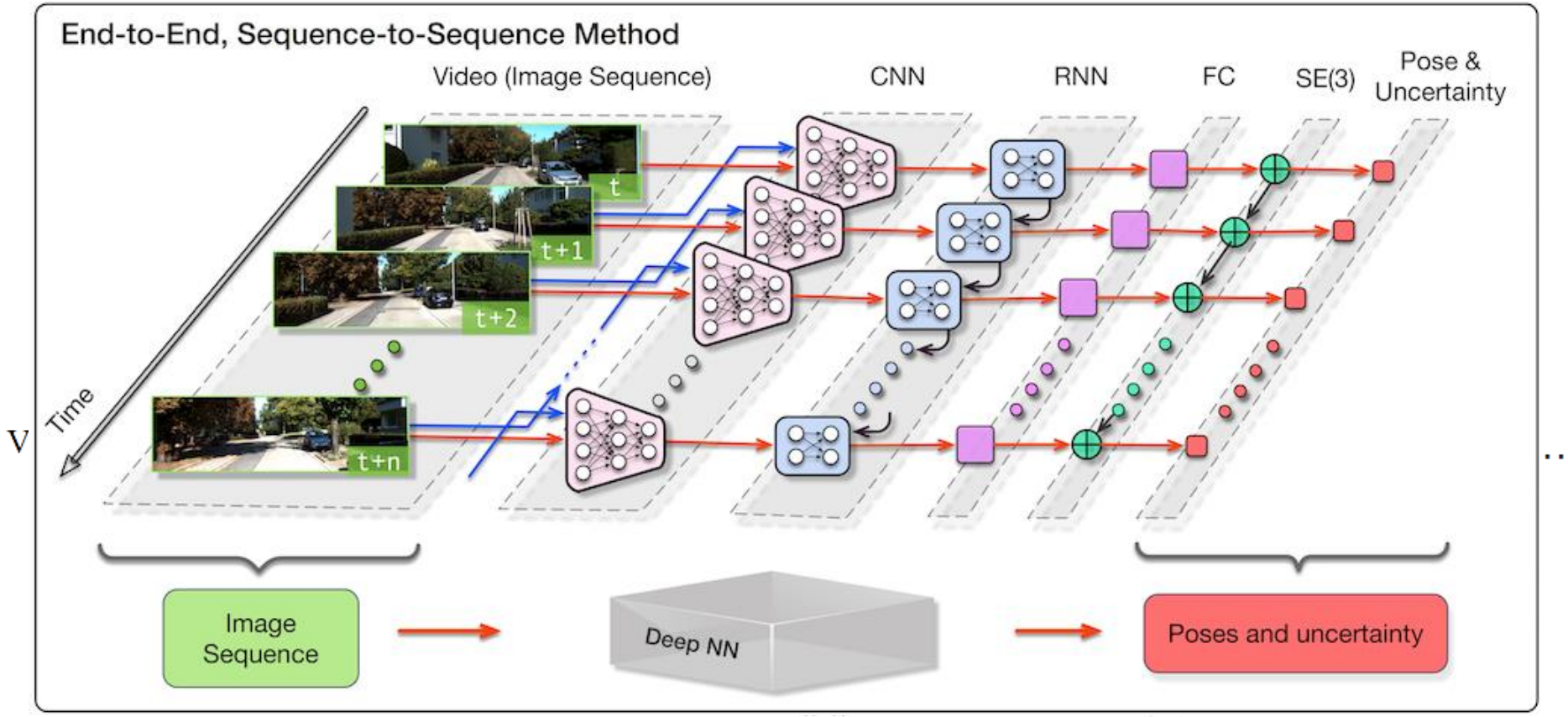




Recurrent Neural Networks (RNN)

Los RNNs son capaces de modelar datos que cambian con el tiempo, gracias a la conexión de las capas de salida/ocultas a las capas ocultas.
Cada paso en el tiempo en RNN es "similar" a una capa oculta en el FFN/MLP.

También incorporan un proceso de olvido (LSTM y GRU), descartando patrones que no se utilizan con el tiempo.
Reconocimiento de voz, Series temporales, Procesamiento de texto



Redes neuronales

Dr. Raimundo Sánchez

raimundo.sanchez@uai.cl

@raimun2