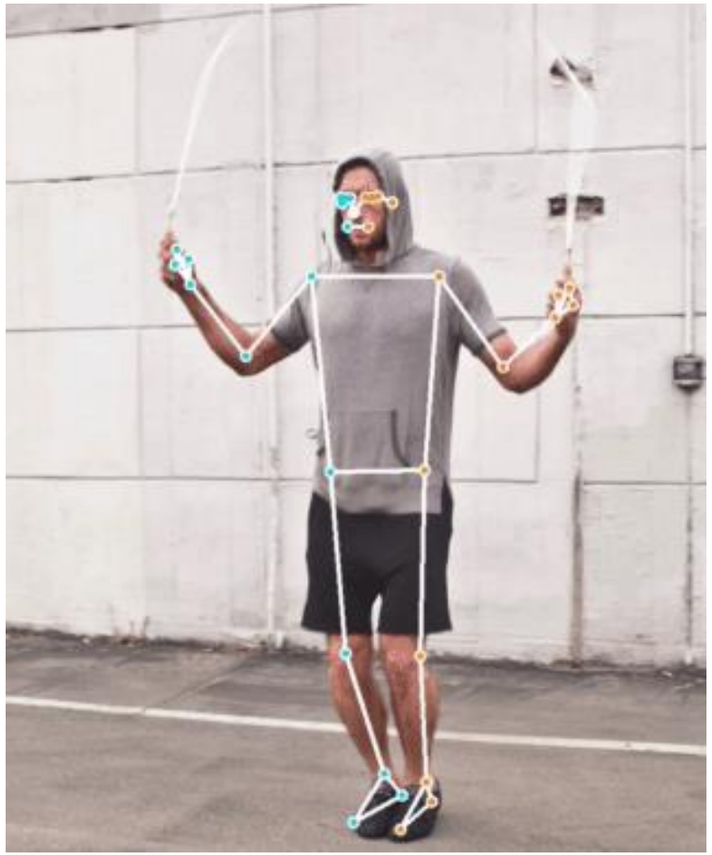
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 - Inteligencia Artificial

Introducción a Redes Neuronales

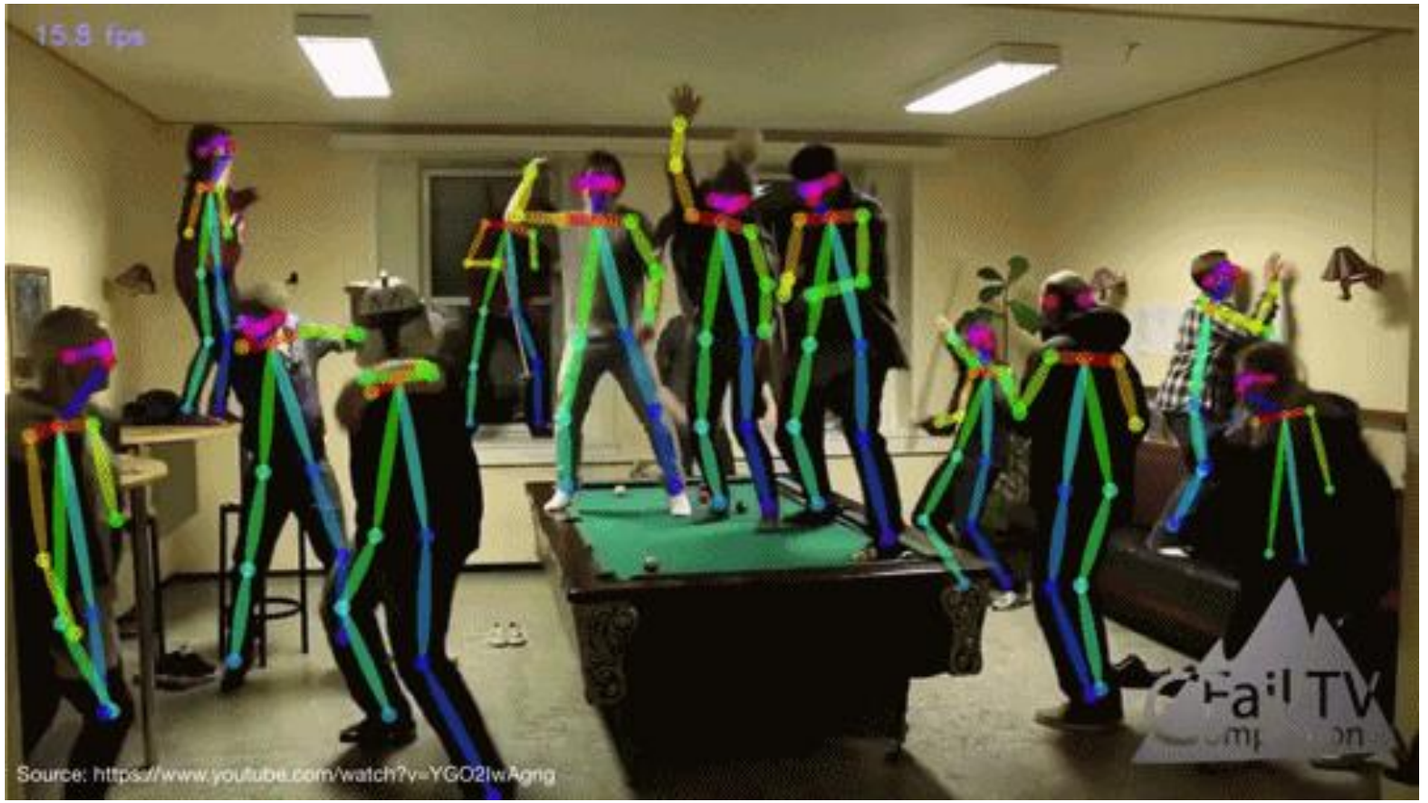
Hans Löbel
Dpto. Ingeniería de Transporte y Logística
Dpto. Ciencia de la Computación

Consideremos un problema altamente complejo y poco estructurado: estimación de pose humana



No tengo idea siquiera de cómo empezar a enfrentar este problema con lo que hemos visto de ML.

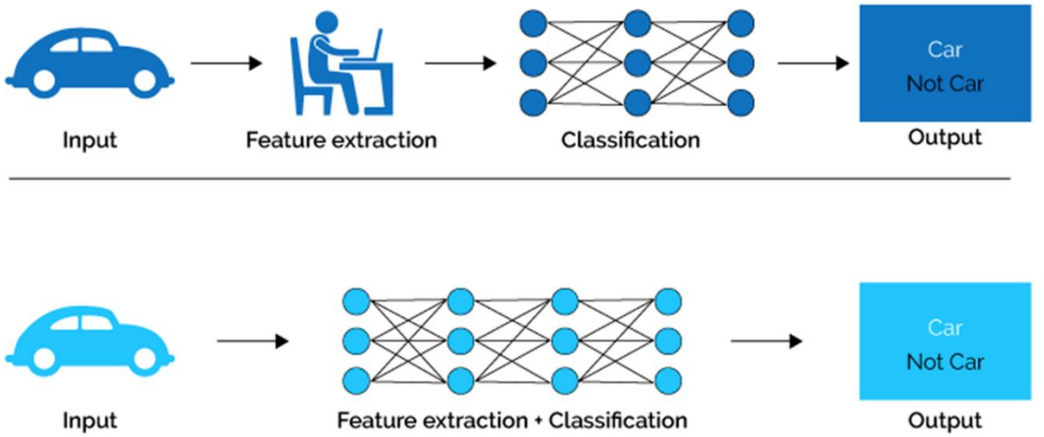
Consideremos un problema altamente complejo y poco estructurado: estimación de pose humana



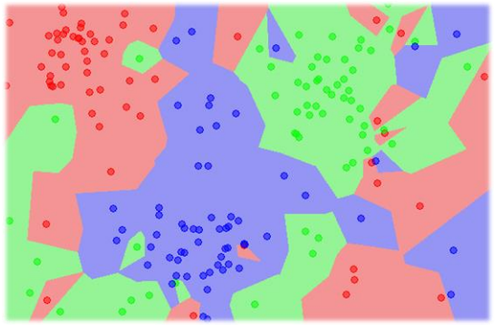
Asumiendo que tengo datos rotulados, qué features utilizo, qué modelo entreno,…

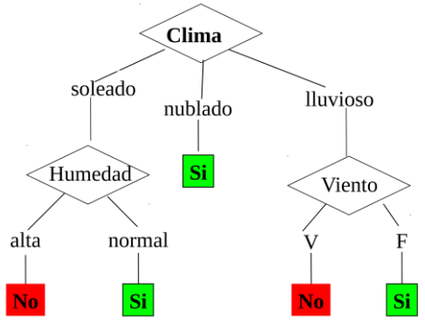
La clave para esto viene de pensar el problema de las features y el modelo como uno conjunto

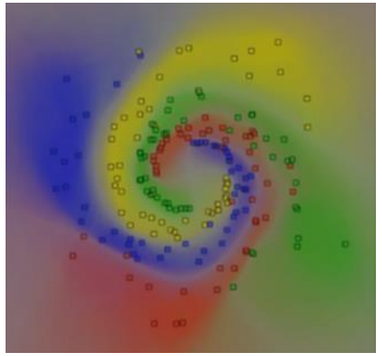
- Hasta ahora, hemos estado usando características (*features/embeddings*) de los datos para clasificar/predecir.
- Estas tienen que ver más con el conocimiento experto que tengamos (o no) del problema.
- Esto no es ideal por tres motivos:
  i. Un experto es caro y nada asegura que su conocimiento sea óptimo
  ii. Para datos no estructurados, no son evidentes las mejores características y/o especificaciones de modelo
  iii. No es siempre evidente como combinar distintos tipos de datos para extraer información, ni las relaciones que hay entre ellos

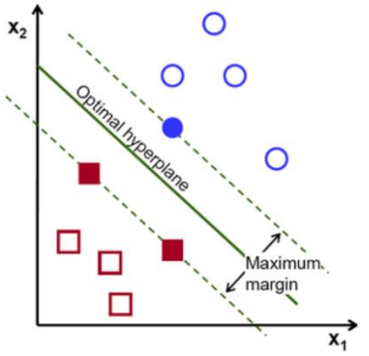


¿Qué es lo que realmente están aprendiendo estos modelos?







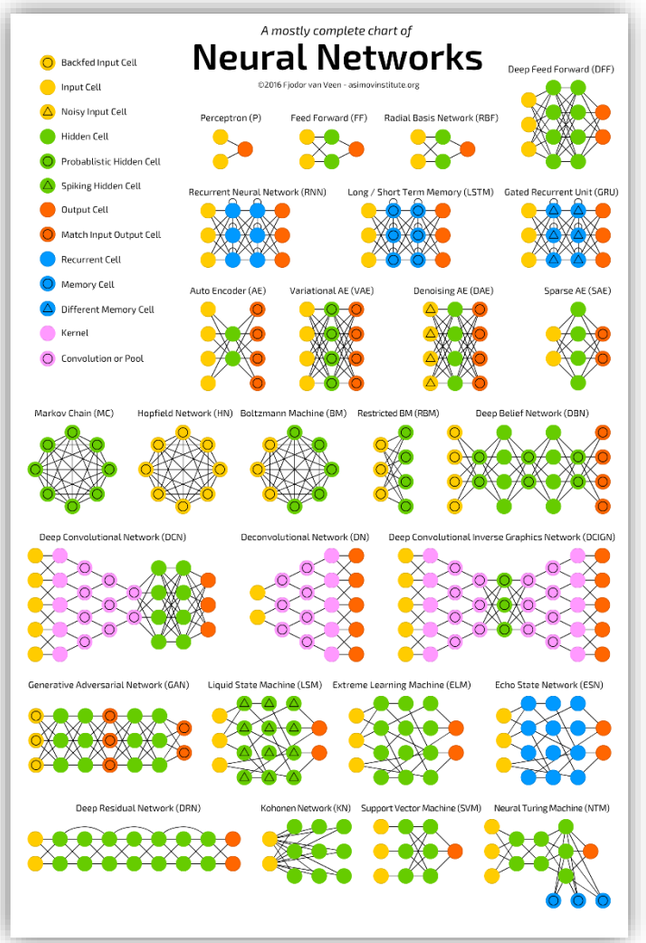


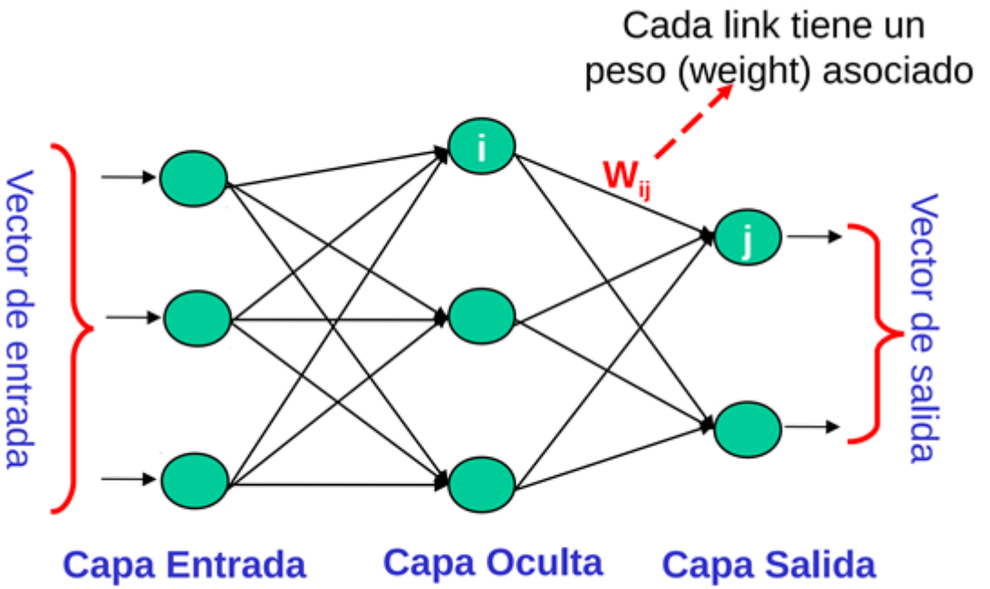
Hasta ahora no estamos “realmente” aprendiendo de los datos, solo como categorizarlos en base a características hechas a manos

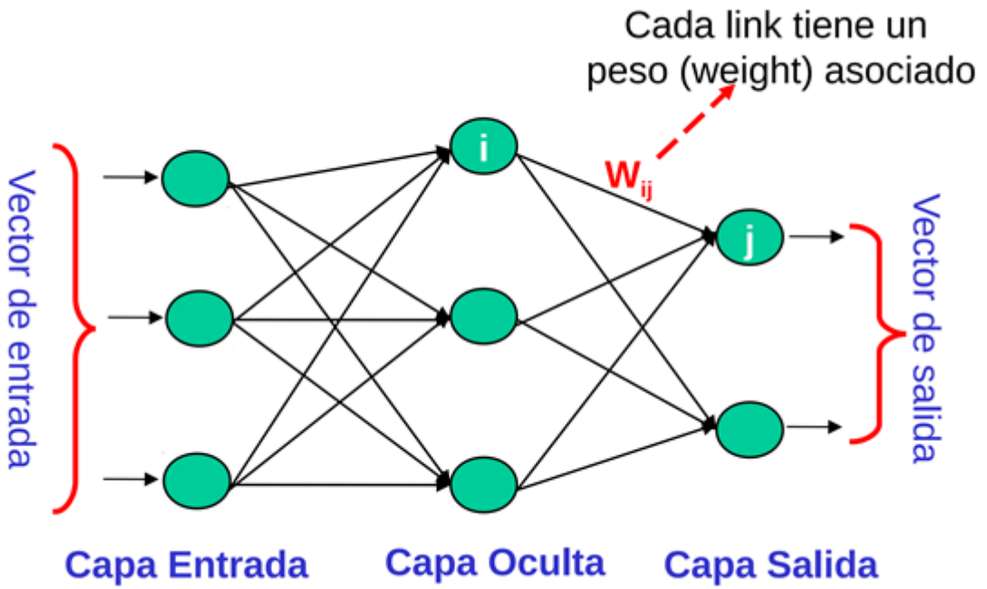
Buscamos un mecanismo de aprendizaje suficientemente poderoso y general, que permita aprender features/embeddings y modelos predictivos que capturen aspectos complejos y eventualmente desconocidos de los datos, con el fin de aprender a realizar tareas de manera óptima

Redes neuronales son modelos altamente prácticos y generales para aprender funciones continuas y discretas

- Brillan en la presencia de grandes volúmenes de datos.
- Son capaces de aprender transformaciones de los datos (*embeddings*), que luego pueden ser usadas como *features* en otros problemas.
- Existen redes para prácticamente cualquier problema.
- Difíciles de entrenar, interpretabilidad es compleja (caja negra) y sufren de serios problemas de sobreajuste (esto último ya no es tan claro).

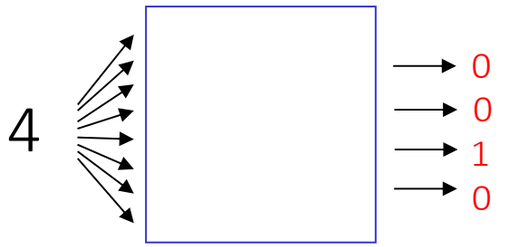


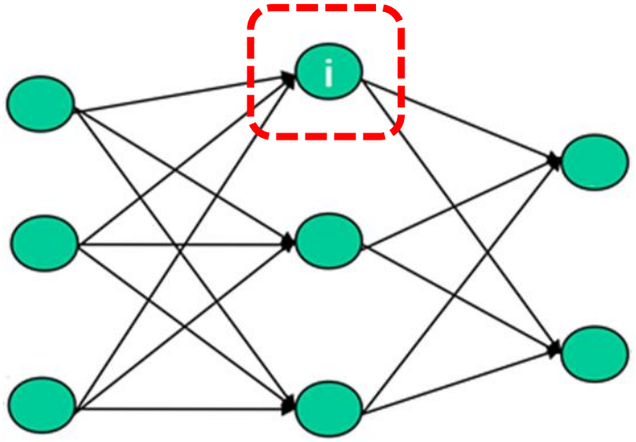

Cada link tiene un
peso (weight) asociado
W_{ij}
i
j
Vector de entrada
Vector de salida
Capa Entrada
Capa Oculta
Capa Salida



Dada la estructura de la red, ¿cómo podemos ajustar los pesos para que al ingresar un vector de entrada determinado obtengamos la salida deseada?

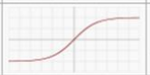
Ejemplo: cuando la red recibe una imagen de un 4, su codificación de salida debe indicar un 4.

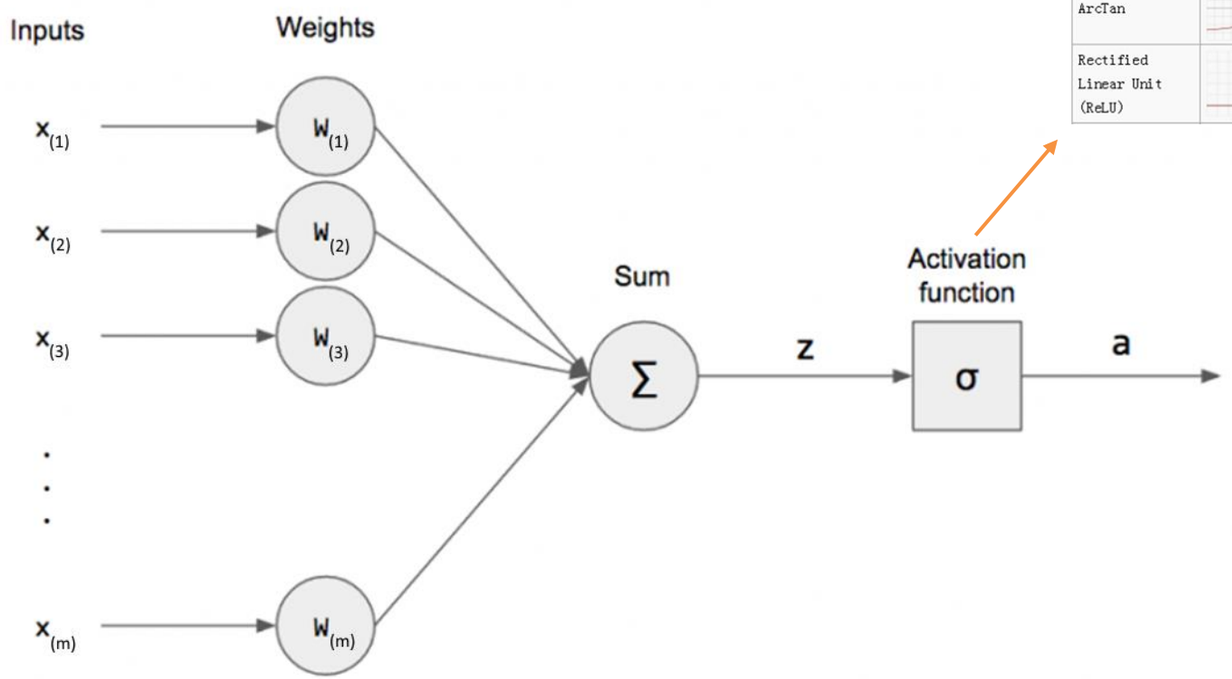


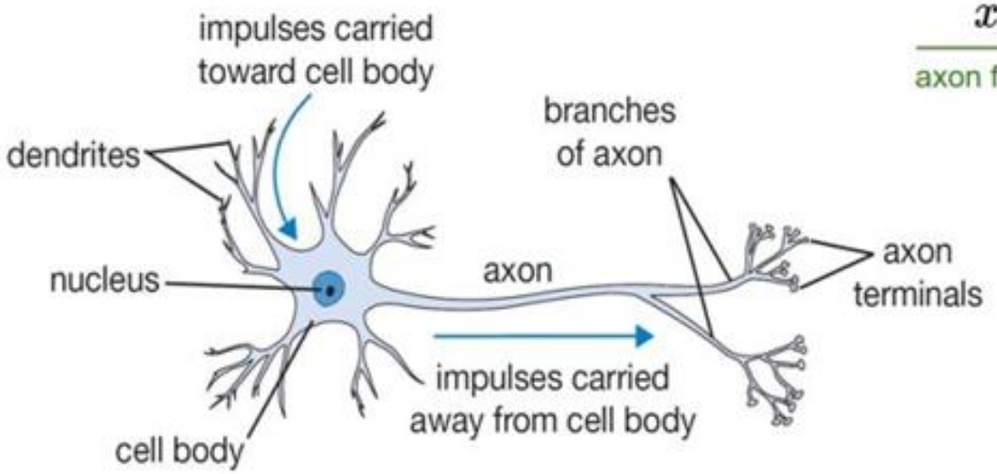

i

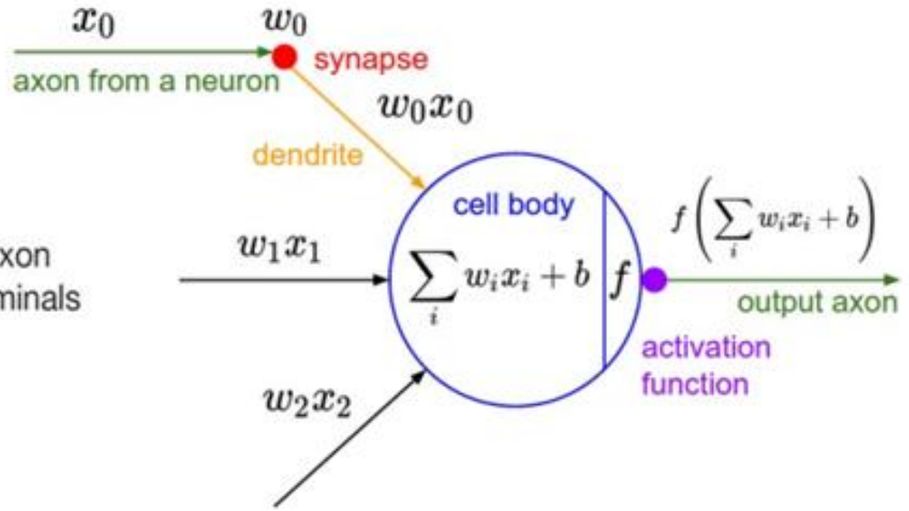
1958 Perceptron

El Perceptron puede considerarse como el precursor de las redes neuronales

Name	Plot	Equation
Identity		$f(x) = x$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$




impulses carried
toward cell body
dendrites
nucleus
cell body
branches
of axon
axon
axon
terminals
impulses carried
away from cell body


x_0
w_0
axon from a neuron
synapse
$w_0 x_0$
dendrite
cell body
$w_1 x_1$
$\sum_i w_i x_i + b$
f
$f\left(\sum_i w_i x_i + b\right)$
output axon
activation
function
$w_2 x_2$

Un Perceptron permite estimar funciones de manera supervisada

- Si tenemos suficientes datos, pares (x_i, y_i), buscamos construir una función de pérdida, que nos indique cuán buena es en promedio la estimación del Perceptron.
- Sea $f(x; w) = \sigma(\sum_{i=1}^{m} w_{(i)} x_{(i)}) = \sigma(w \cdot x)$, una función que modela la aplicación de un Perceptrón lineal* parametrizado por un vector de pesos w, a un vector de características x.
- Sin suponer cosas raras sobre los datos, una posible función de pérdida (o costo, o error) puede quedar de la siguiente manera:

$$J(w) = \frac{1}{2n} \sum_{i=1}^{n} (f(x_i; w) - y_i)^2$$

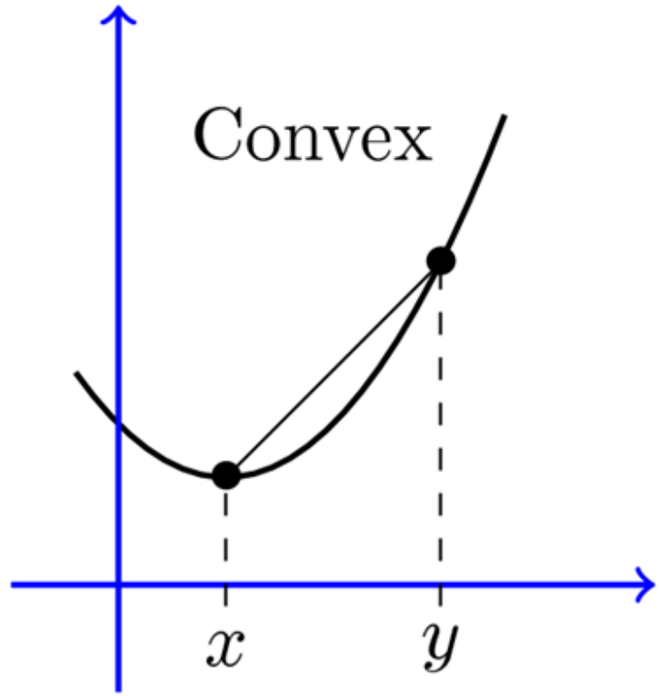
- ¿Qué representa el valor mínimo de esta función, con respecto a los pesos w?

* función identidad como activación

¿Cómo podemos entrenar un Perceptron (lineal) para estimar funciones?

- La función de pérdida es convexa (siempre que la función de activación también lo sea, lo que ocurre regularmente)

$$J(w) = \frac{1}{2n}\sum_{i=1}^{n}(f(x_i; w) - y_i)^2$$



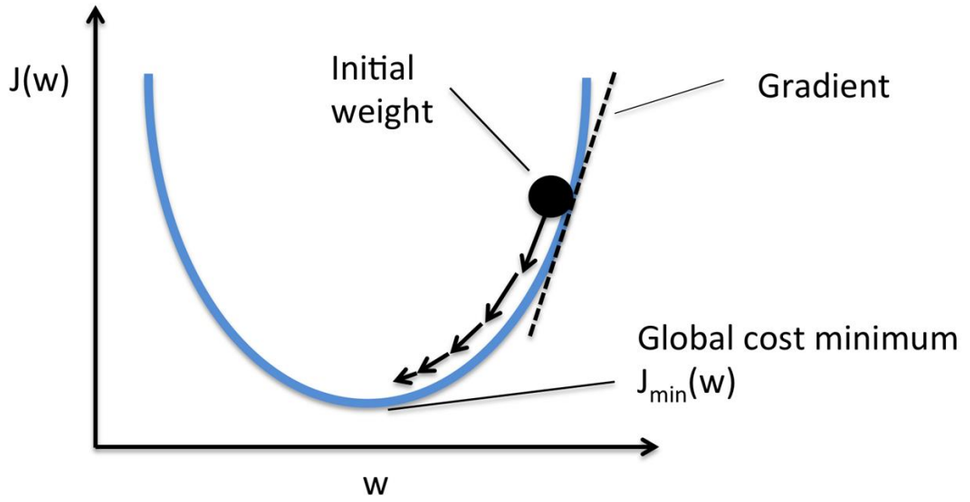
¿Cómo podemos entrenar un Perceptron (lineal) para estimar funciones?

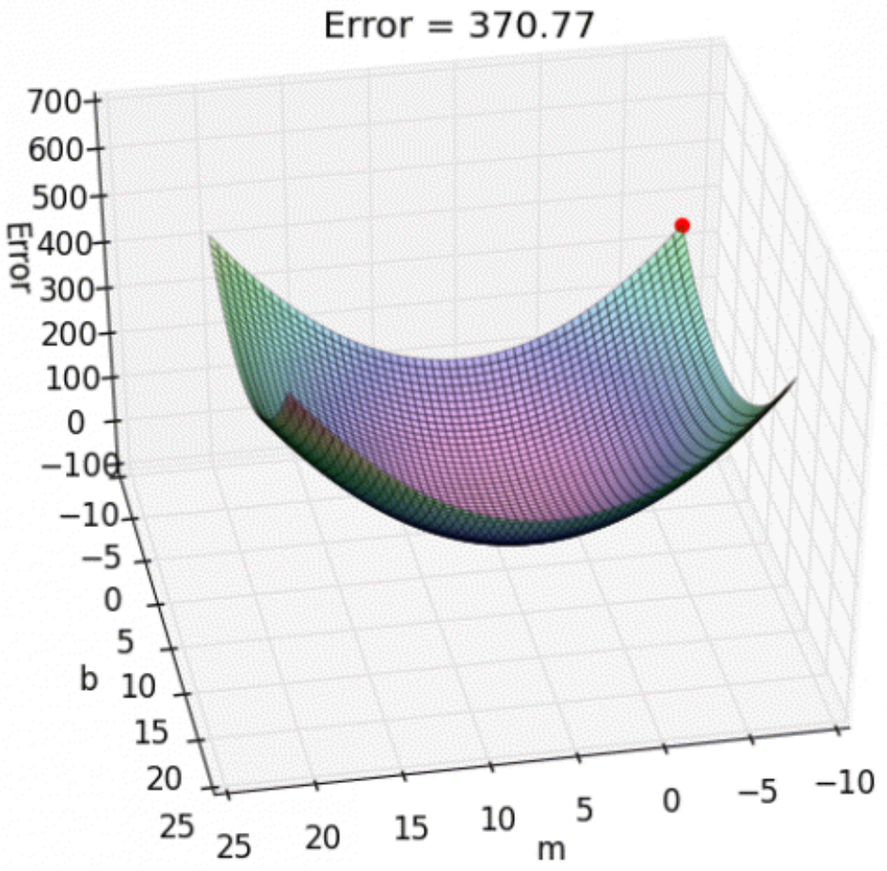
- La función de pérdida es convexa (si la función de activación también lo es, lo que ocurre regularmente)

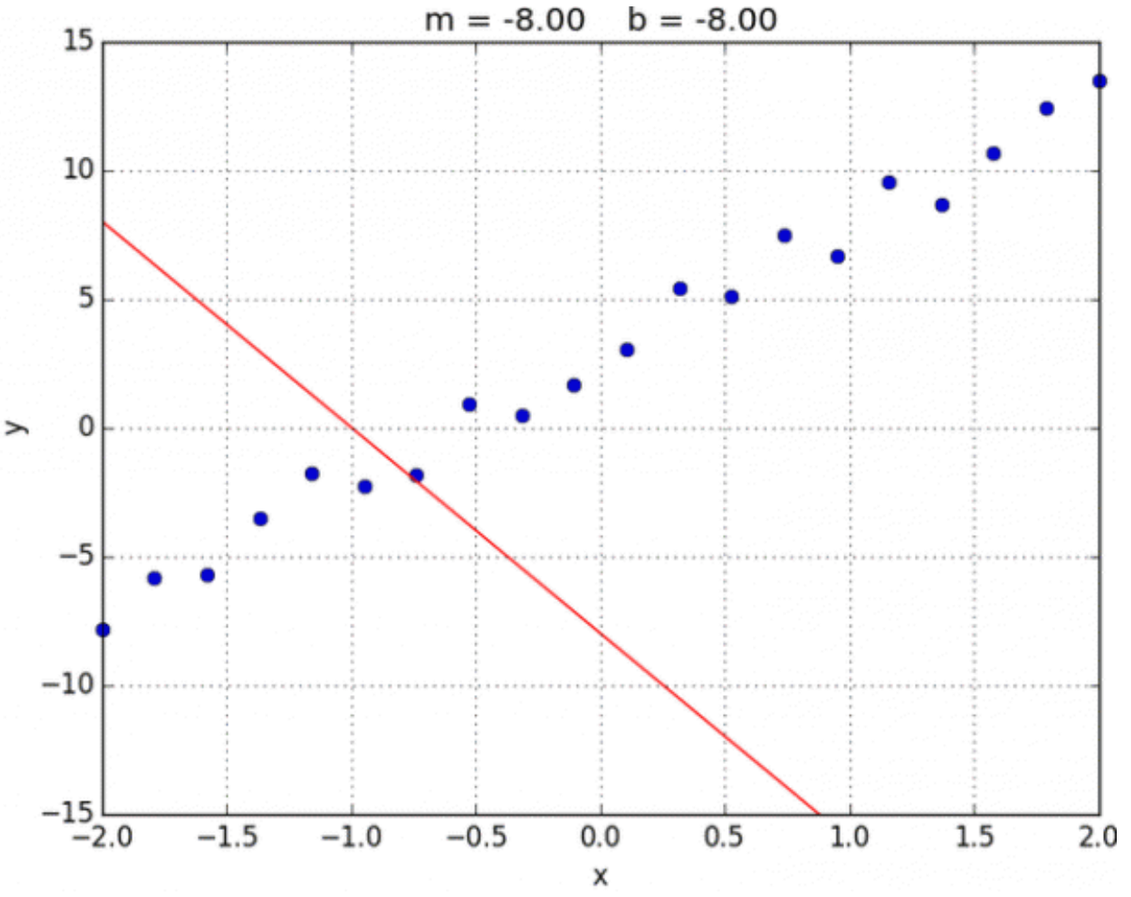
$$J(w) = \frac{1}{2n}\sum_{i=1}^{n}(f(x_i; w) - y_i)^2$$

- Dado esto, es posible utilizar el algoritmo de descenso de gradiente

btw, ¿cuál es la derivada de la función de pérdida con respecto a w?




Error = 370.77
Error
700
600
500
400
300
200
100
0
−100
b
−10
−5
0
5
10
15
20
25
m
25
20
15
10
5
0
−5
−10


m = -8.00 b = -8.00
y
15
10
5
0
−5
−10
−15
x
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Dado que el problema de aprendizaje es convexo, podemos usar “tranquilamente” descenso de gradiente

---

Require: Pesos $w_{(i)}$ inicializados con un valor aleatorio pequeño

do

  $\Delta w \leftarrow \vec{0}$

  for each $(x, y) \in \mathcal{S}$ **do**

    Aplicar Perceptron a x y almacenar salida en $\hat{y}$

    for each $\Delta w_{(i)} \in \Delta w$ **do**

      $\Delta w_{(i)} \leftarrow \Delta w_{(i)} + (y - \hat{y}) x_{(i)}$

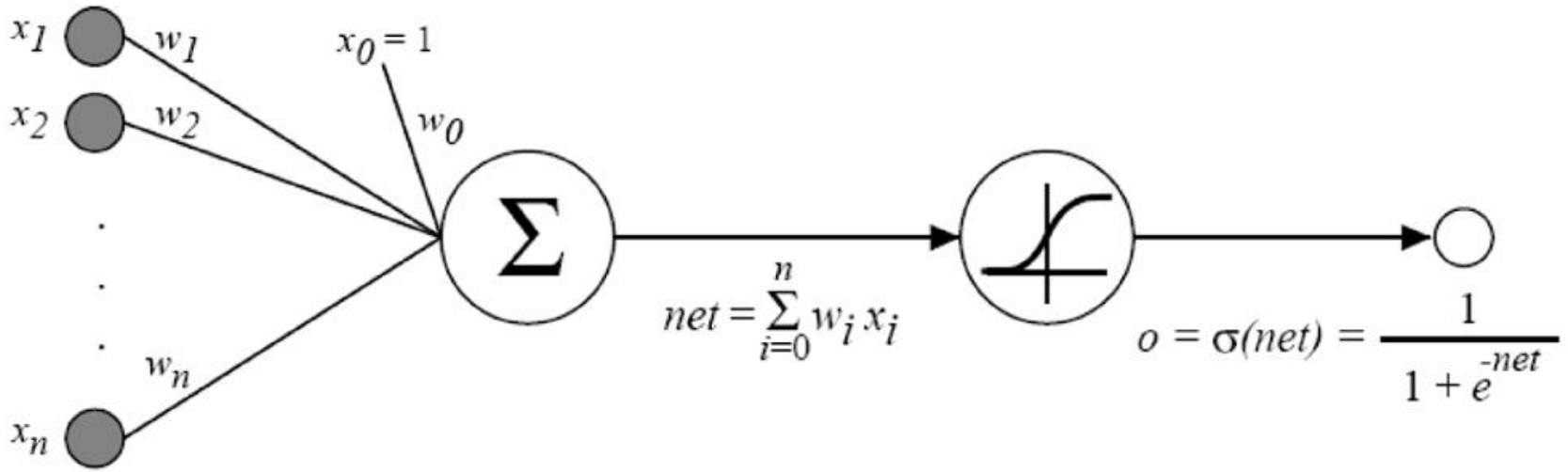
    end for

  end for

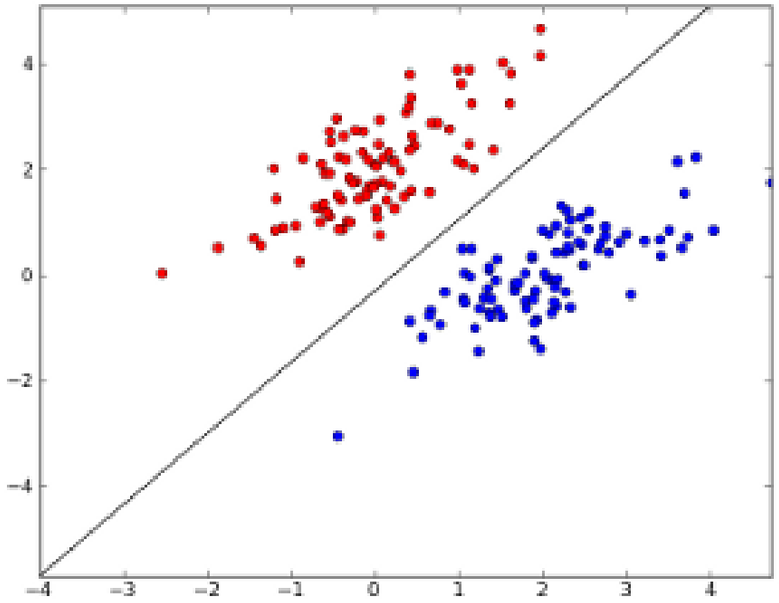
  $w \leftarrow w - \alpha \frac{\Delta w}{|\mathcal{S}|}$

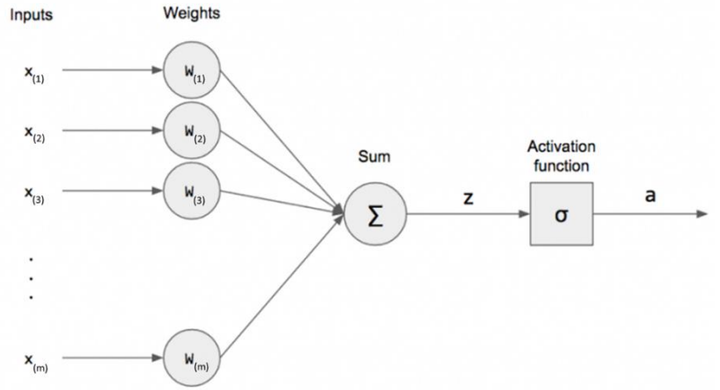
while condición de término == **False**

Si incorporamos una función de activación no lineal, la situación no cambia mucho

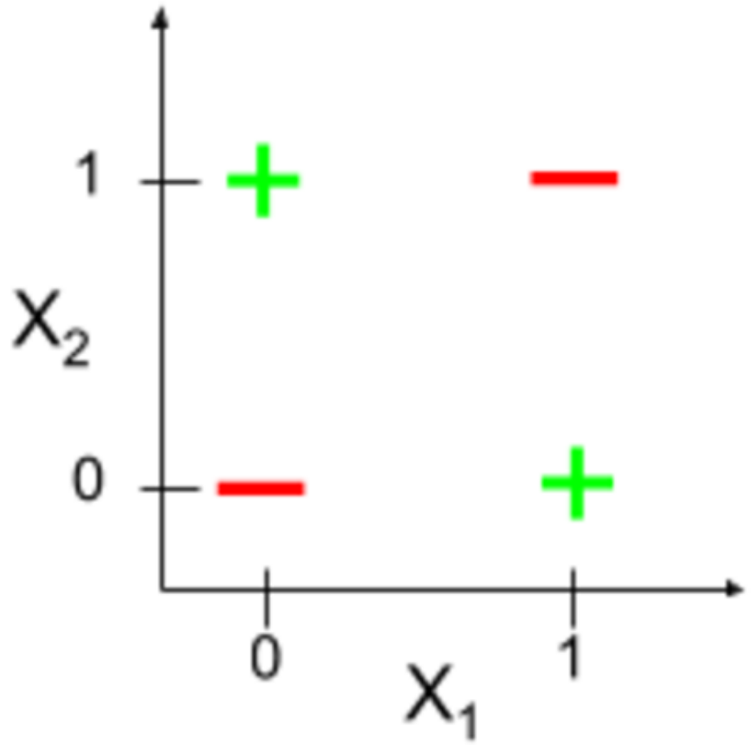


$$\frac{\partial J}{\partial w_i} = -\sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

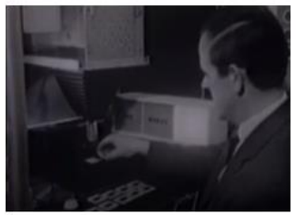




Perceptron puede resolver problemas lineales (clasificación y regresión)

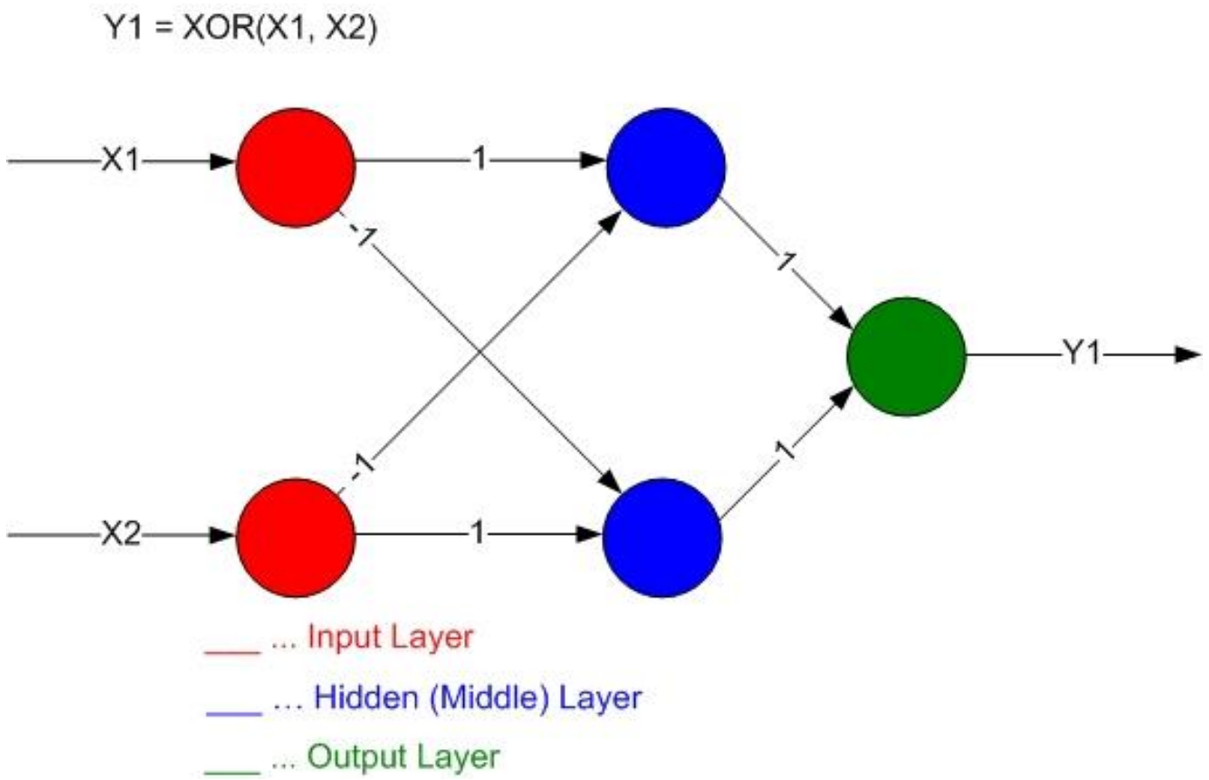


¿Cómo podríamos extender al Perceptron, para que sea capaz de resolver este tipo de problemas (no linealmente separables)?



1958 Perceptron

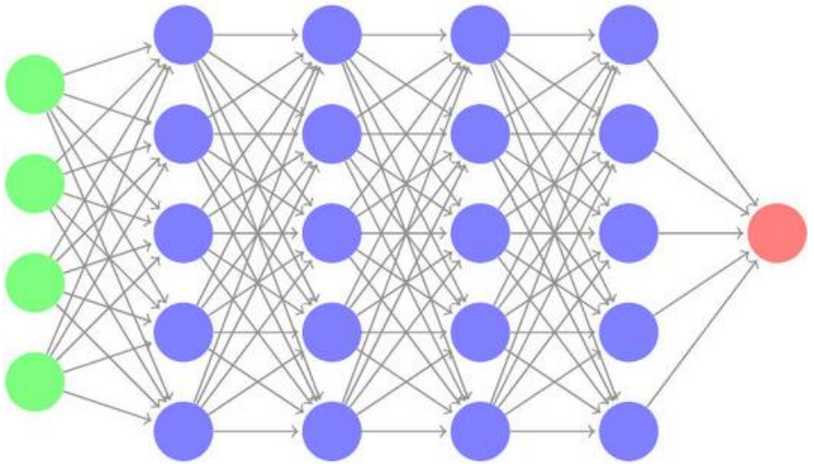
1969
Perceptron criticized



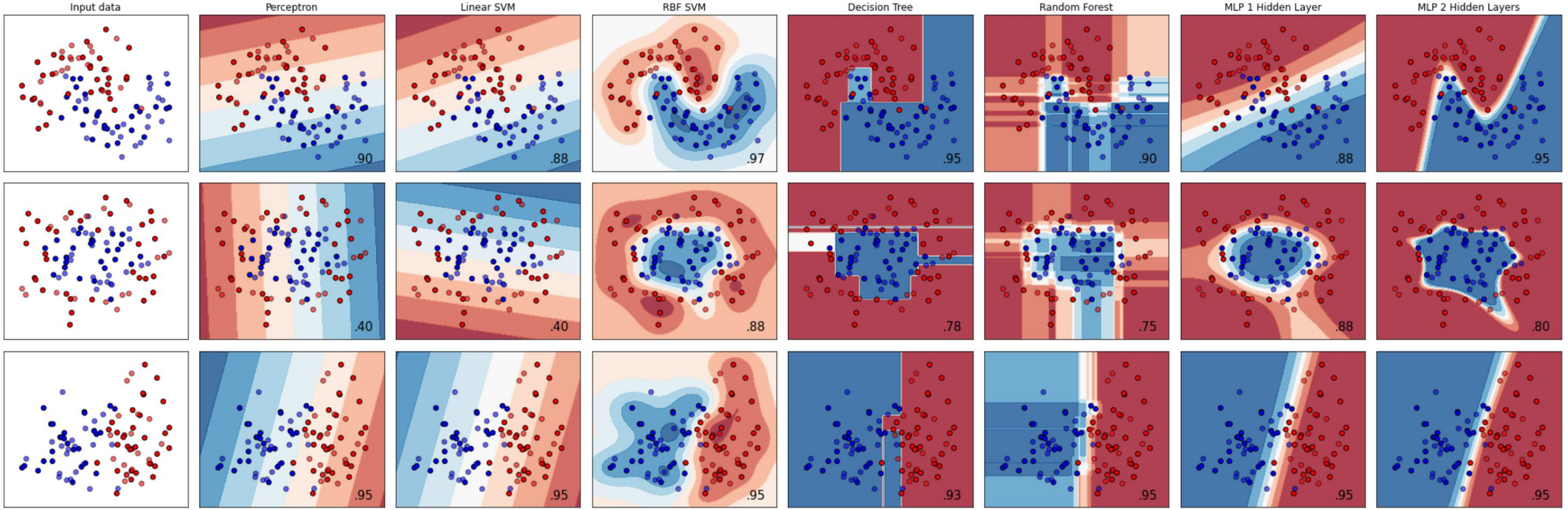
Al agregar una nueva capa (oculta), podemos representar conceptos intermedios entre el input y el output.

MLP generaliza el concepto de Perceptron

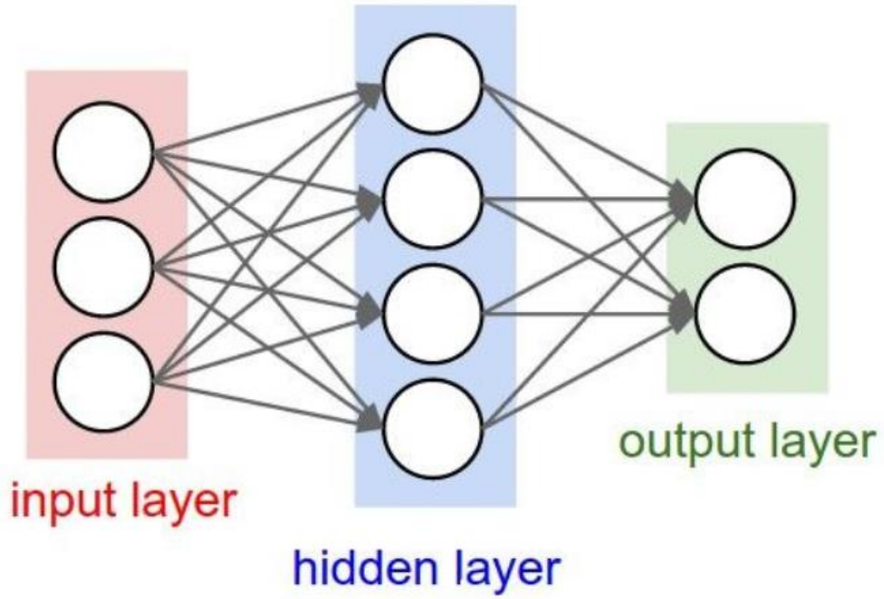
- Un Multilayer Perceptron (MLP), combina múltiples perceptrones en múltiples capas. Es conocido también como Deep Feed Forward Network (DFF).
- Permite resolver problemas más complejos que los que puede resolver un Perceptron.



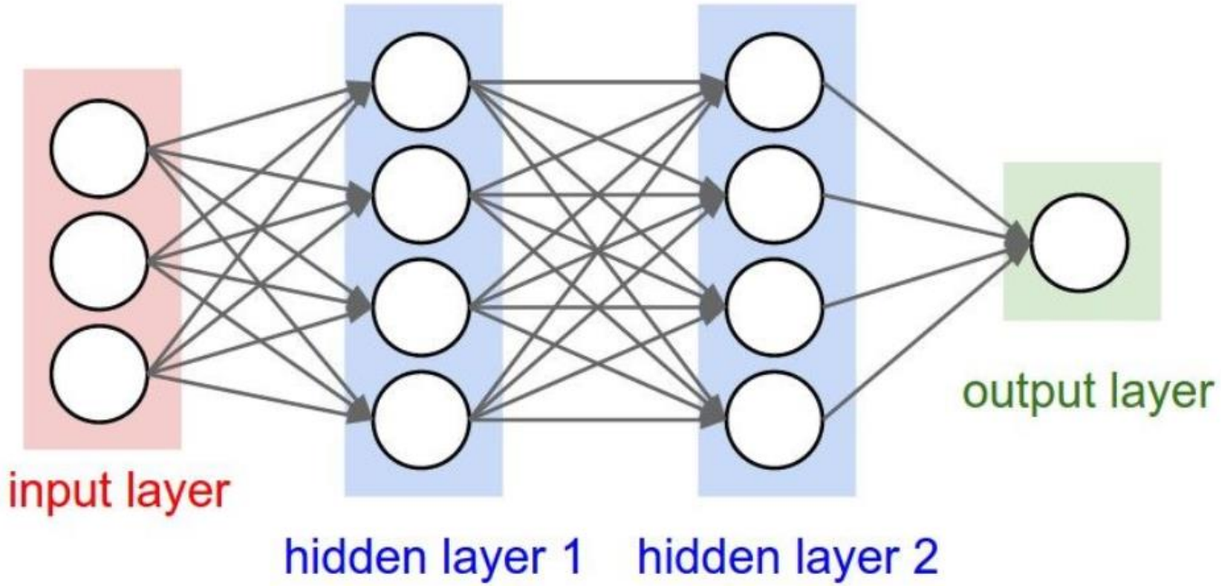
Nuevas capas aumentan poder discriminativo comparado con un Perceptron



Visualmente, representaremos los MLP utilizando capas



MLP (NN) de 2 capas, o NN de 1 capa oculta



MLP (NN) de 3 capas, o NN de 2 capas ocultas

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2613 - Inteligencia Artificial

Introducción a Redes Neuronales

Hans Löbel
Dpto. Ingeniería de Transporte y Logística
Dpto. Ciencia de la Computación