

Aprendizaje Profundo

INTRODUCCIÓN

¿Qué son las redes neuronales y cómo funcionan?

Presentación de la materia

Profesores

Cristian Cardellino

Métodos semi-supervisados de aprendizaje profundo para tareas de procesamiento de lenguaje natural.

Milagro Teruel

Aprendizaje de representaciones para minería de datos educacionales usando técnicas de aprendizaje profundo.



GRUPO DE PROCESAMIENTO DE LENGUAJE NATURAL

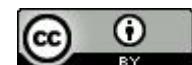
FaMAF - Universidad Nacional de Córdoba



Universidad
Nacional
de Córdoba



Facultad
de Matemática,
Astronomía, Física
y Computación



This work by Cristian Cardellino and Milagro Teruel is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/).

Sobre la materia

Objetivos

- Comprender qué son y para qué sirven las **redes neuronales**.
- Entender las redes neuronales como un **método de aprendizaje de representaciones**.
- **Implementación y entrenamiento de redes neuronales** con herramientas de alto nivel.

Materiales

Las clases se encuentran disponibles en github.com/DiploDatos/AprendizajeProfundo. Sigan las instrucciones para poder instalar y configurar el entorno de desarrollo que utilizaremos con la herramienta **Keras**.

Métodos de evaluación

- La materia cuenta con dos partes: introducción y avanzado.
- Cada parte tendrá un práctico asociado que resolver de acuerdo a los temas vistos.
- Se puede hacer la primera mitad (sólo la introducción) o la totalidad de la materia.

Recursos varios

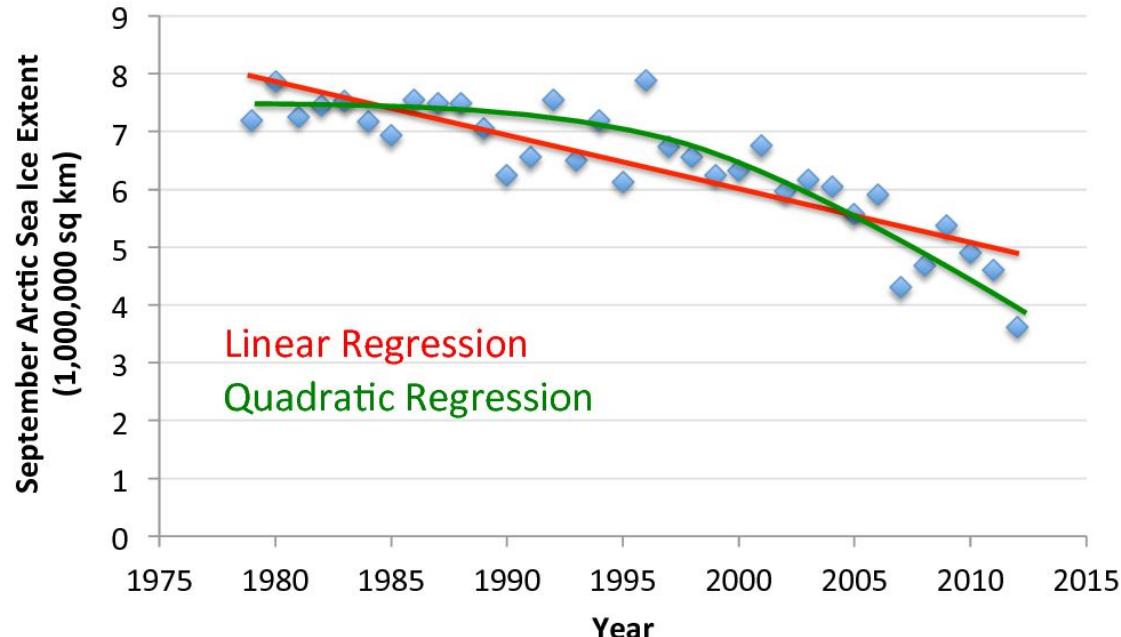
- [Neural Networks Demystified](#)
- [Curso de Aprendizaje Automático de Andrew Ng en Coursera](#)
- [Keras Documentation](#)
- [Deep Learning Book](#)
- [Deep Learning with Python](#)
- [Github de la materia](#)

Breve repaso de regresión lineal y logística

Regresión

Datos: $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$ donde $\mathbf{x}^{(i)} \in \mathbb{R}^d$

Etiquetas de los datos: $\mathbf{y} = \{y^{(1)}, \dots, y^{(n)}\}$ donde $y^{(i)} \in \mathbb{R}$



Regresión lineal

Hipótesis de regresión lineal:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_d x_d = \sum_{j=0}^d \theta_j x_j$$

Assume $x_0 = 1$



Los valores θ son los pesos de los atributos
(*features* en inglés)

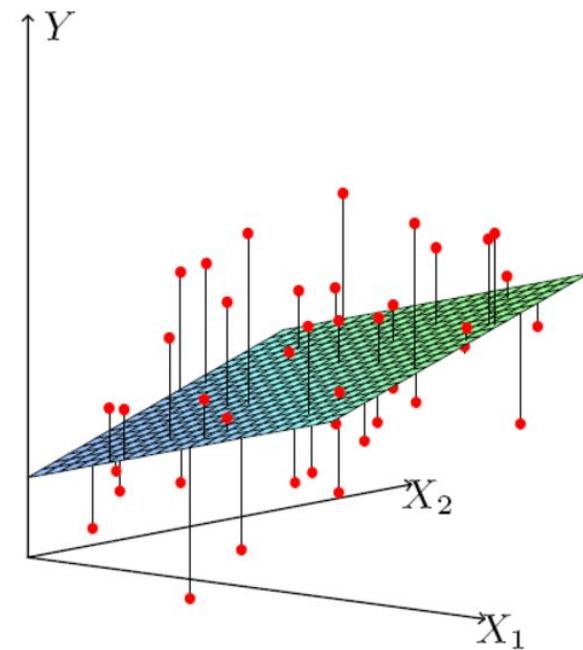
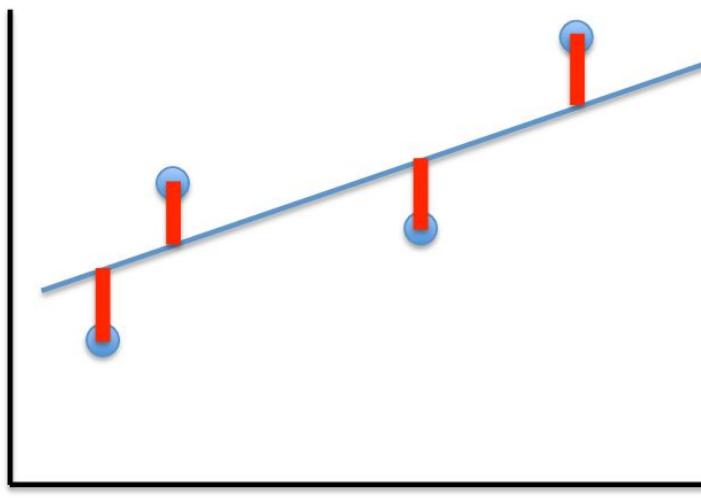
Entrenar minimizando la suma del error cuadrado

Función de coste en regresión lineal

Dada la función de coste

$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^n \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2$$

Se resuelve por $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$



Regresión logística

Uso un enfoque probabilístico:

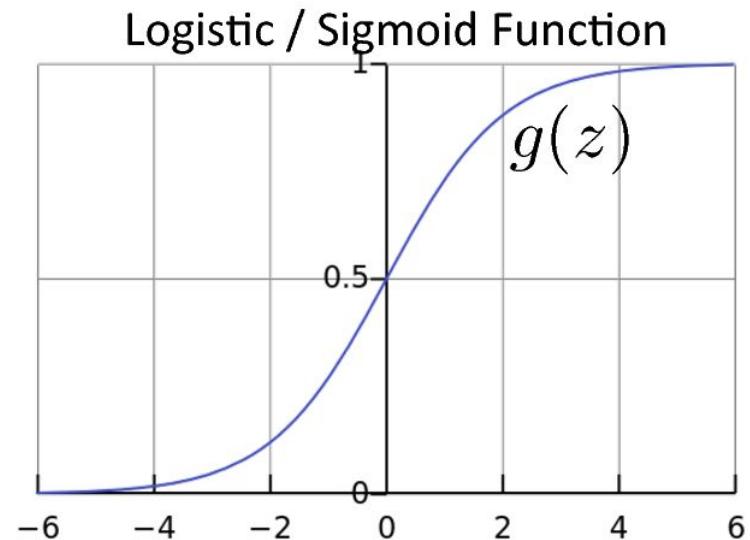
$$h_{\theta}(x) \text{ debería ser } p(y = 1 | x; \theta)$$

Modelo de regresión logística:

$$h_{\theta}(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

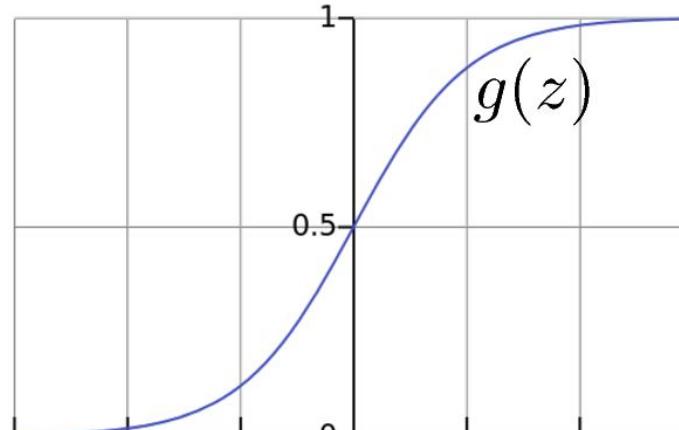
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$



Regresión logística

$$h_{\theta}(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

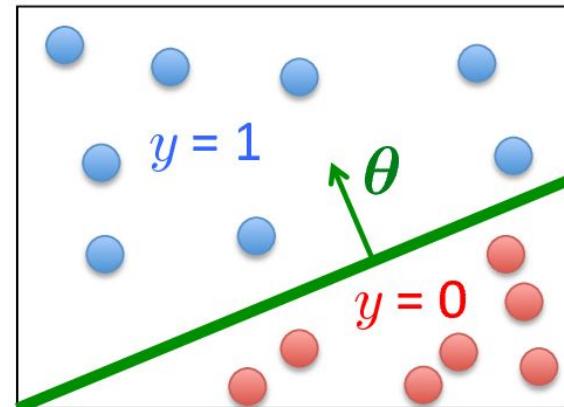


$\theta^T x$ debería tener *valores negativos* grandes para instancias negativas y *valores positivos* grandes para instancias positivas.

Definir un umbral y

Predecir $y = 1$ si $h_{\theta}(x) \geq 0.5$

Predecir $y = 0$ si $h_{\theta}(x) < 0.5$



Función de coste en regresión logística

Estimador de máxima verosimilitud

$$J(\boldsymbol{\theta}) = - \sum_{i=1}^n \left[y^{(i)} \log h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})) \right]$$

Costo de una sola instancia de los datos

$$\text{cost}(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\mathbf{x})) & \text{si } y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) & \text{si } y = 0 \end{cases}$$

Se reescribe la función de coste como

$$J(\boldsymbol{\theta}) = \sum_{i=1}^n \text{cost}\left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)}\right)$$

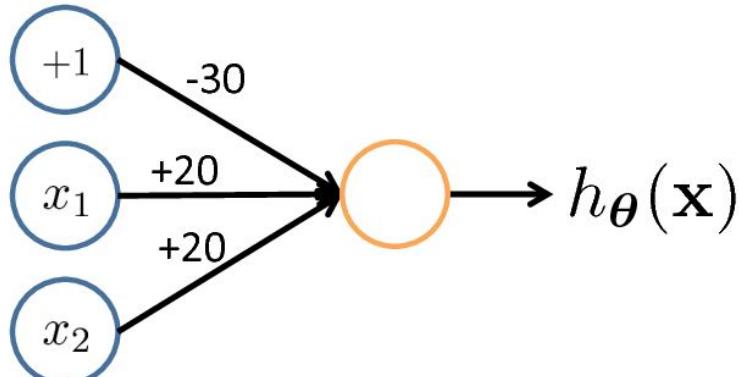
Límite de los algoritmos lineales

Funciones lógicas con regresión logística

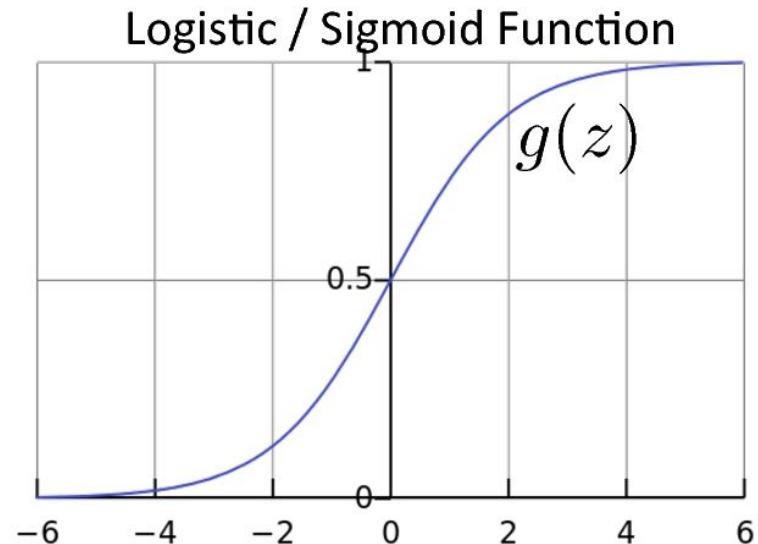
Simple example: AND

$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{ AND } x_2$$

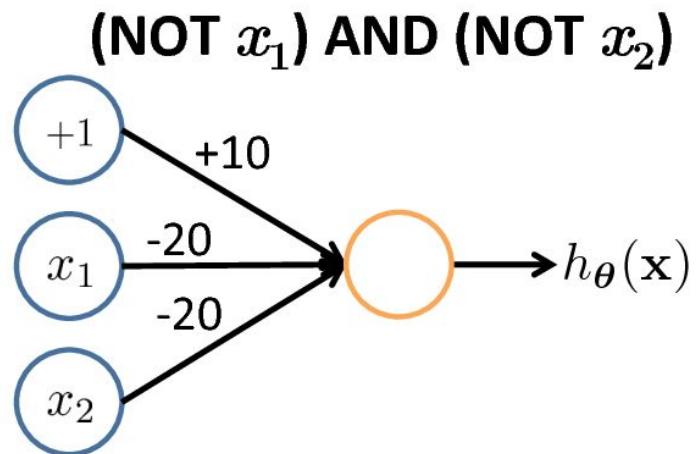
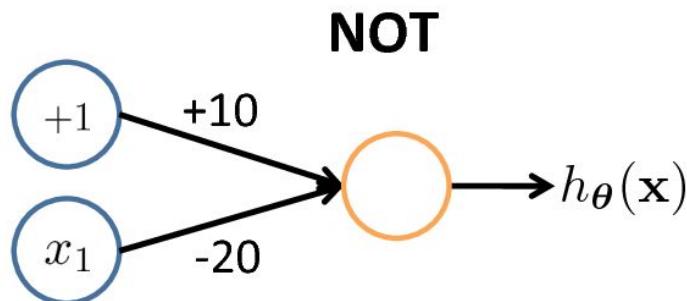
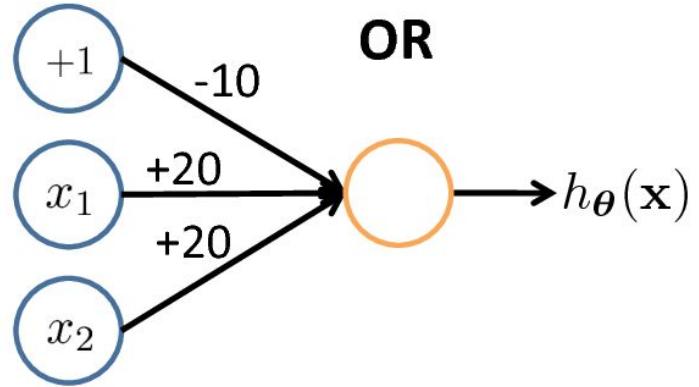
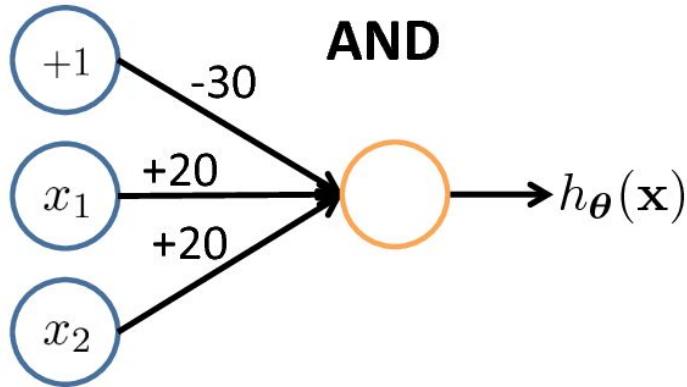


$$h_{\theta}(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

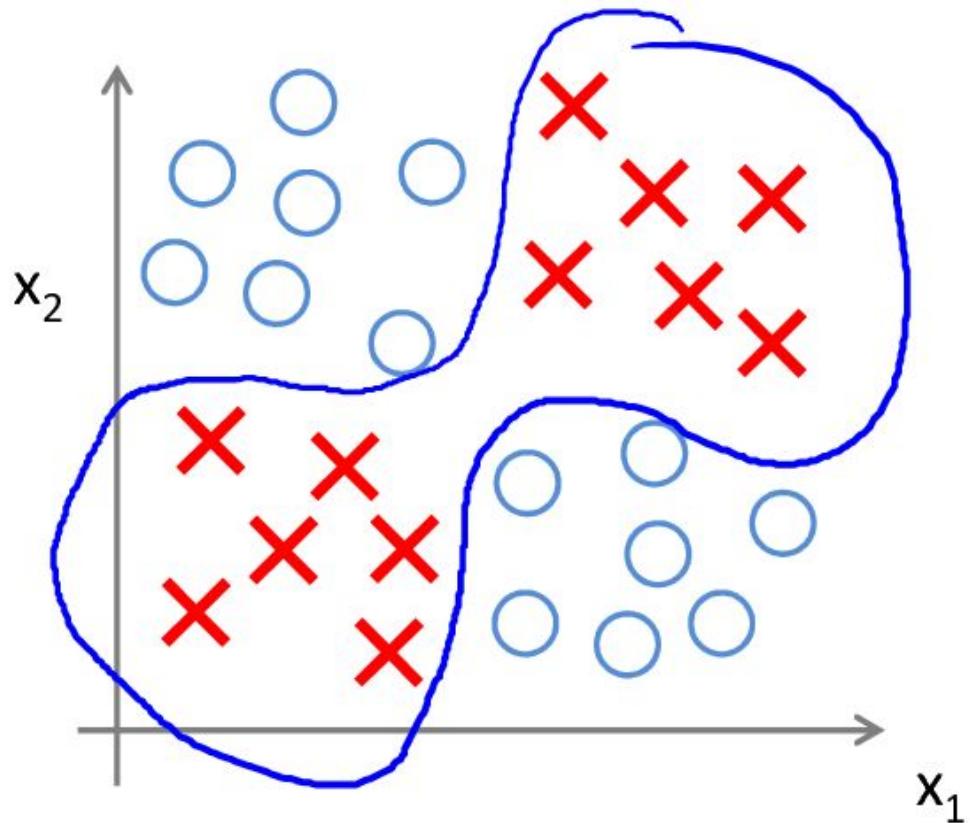
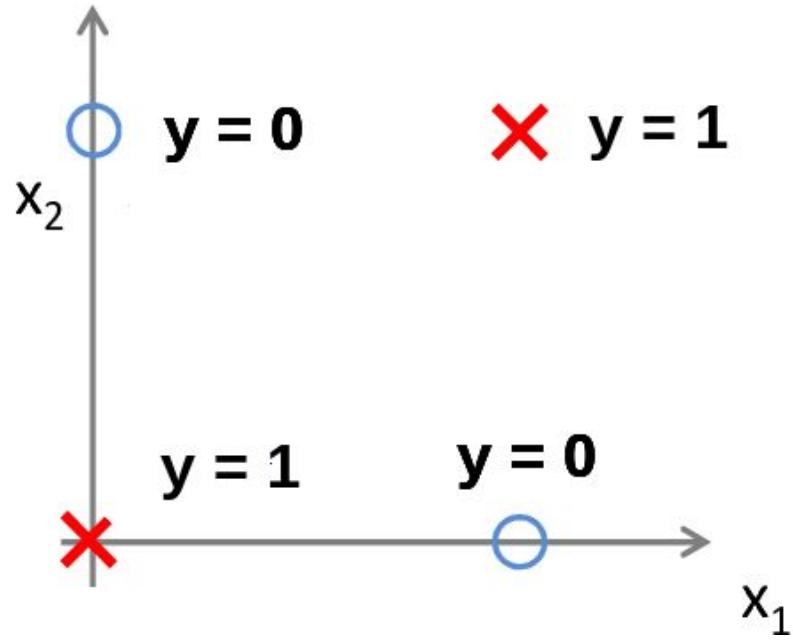


x_1	x_2	$h_{\theta}(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Funciones lógicas con regresión logística

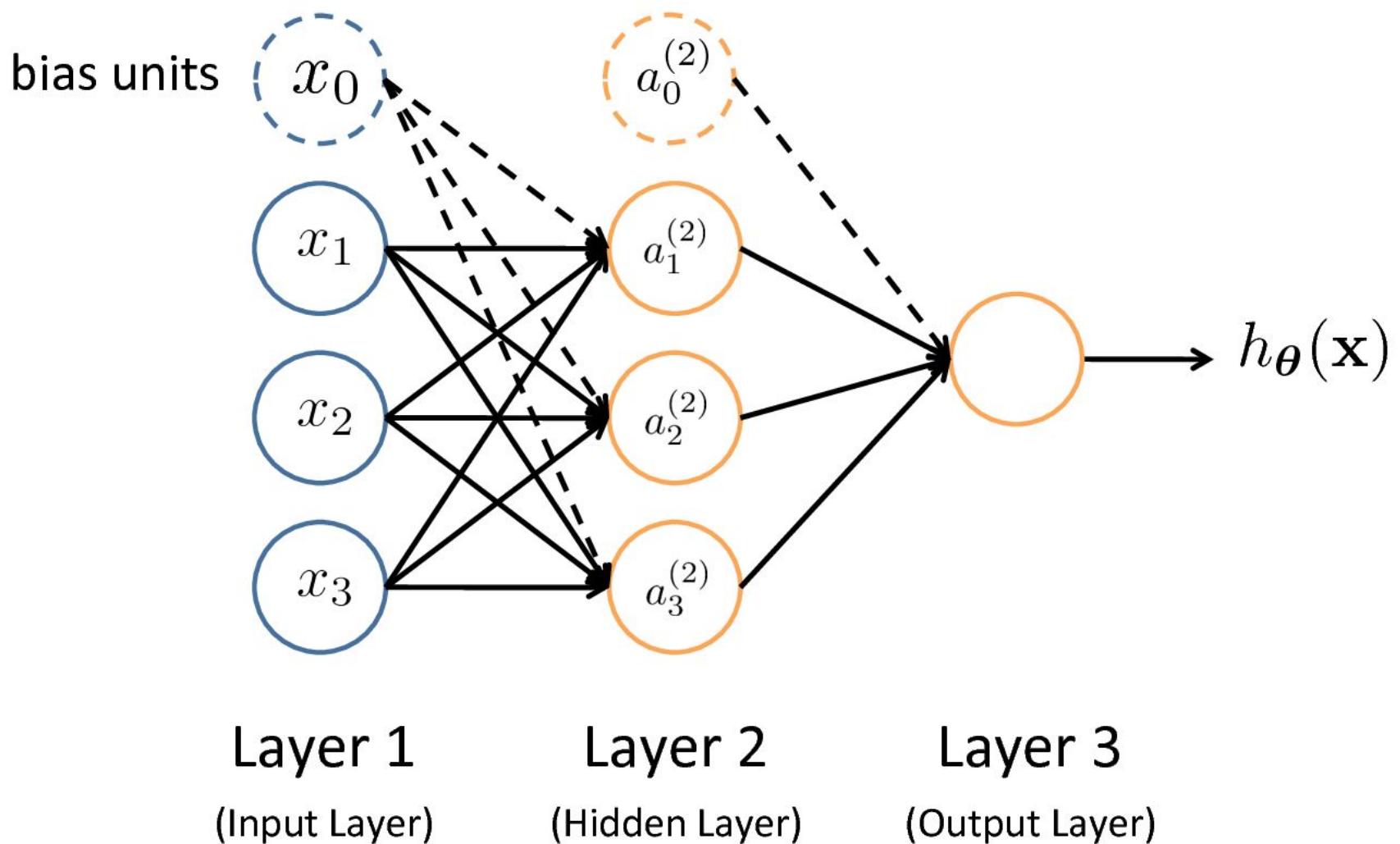


¿Cómo representar el XOR?



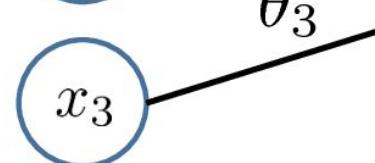
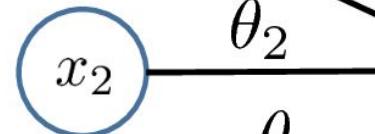
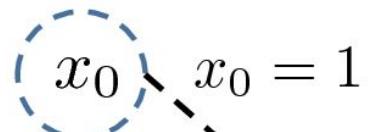
Redes neuronales artificiales

Red neuronal artificial



Unidad Neuronal

“bias unit”



$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

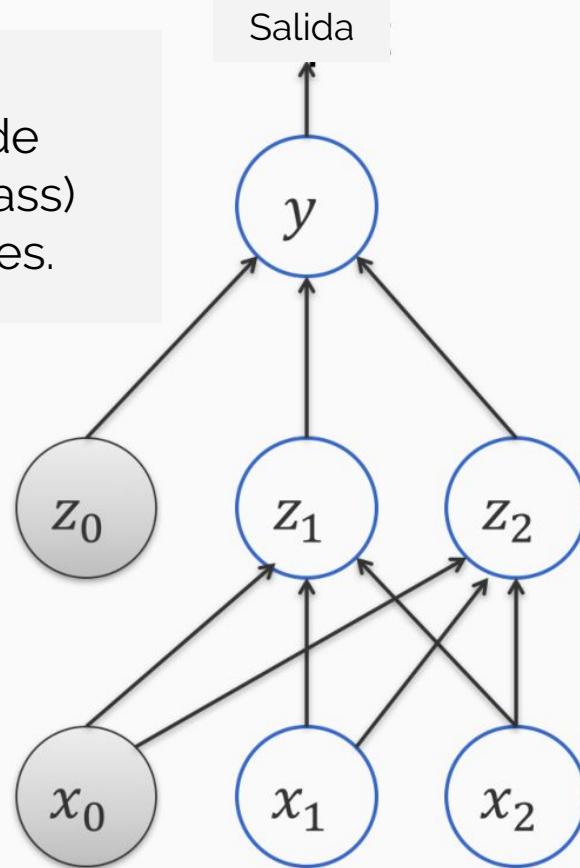
$$\sum \int \rightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x}) \\ = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}}$$

Función de activación (sigmoide): $g(z) = \frac{1}{1 + e^{-z}}$

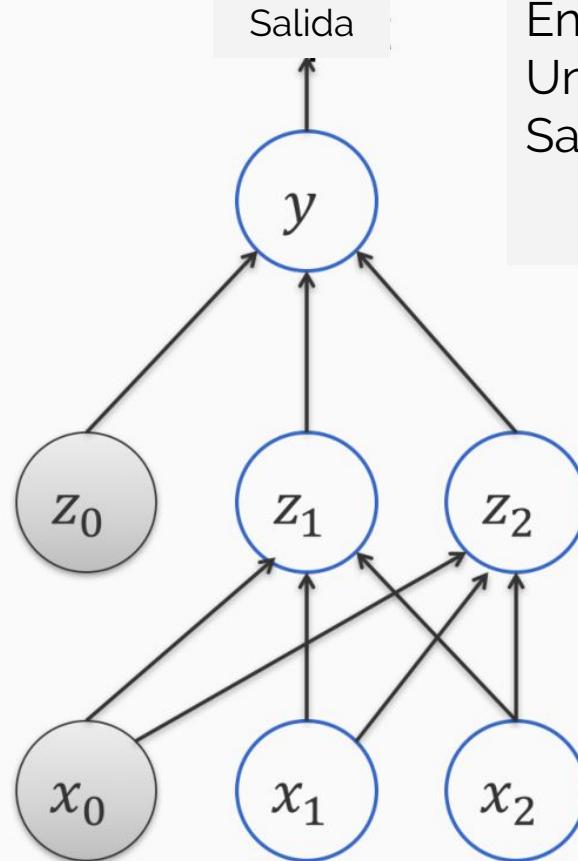
Feed-forward pass

Red de ejemplo

La red se usará para explicar el algoritmo de predicción (forward pass) en las redes neuronales.

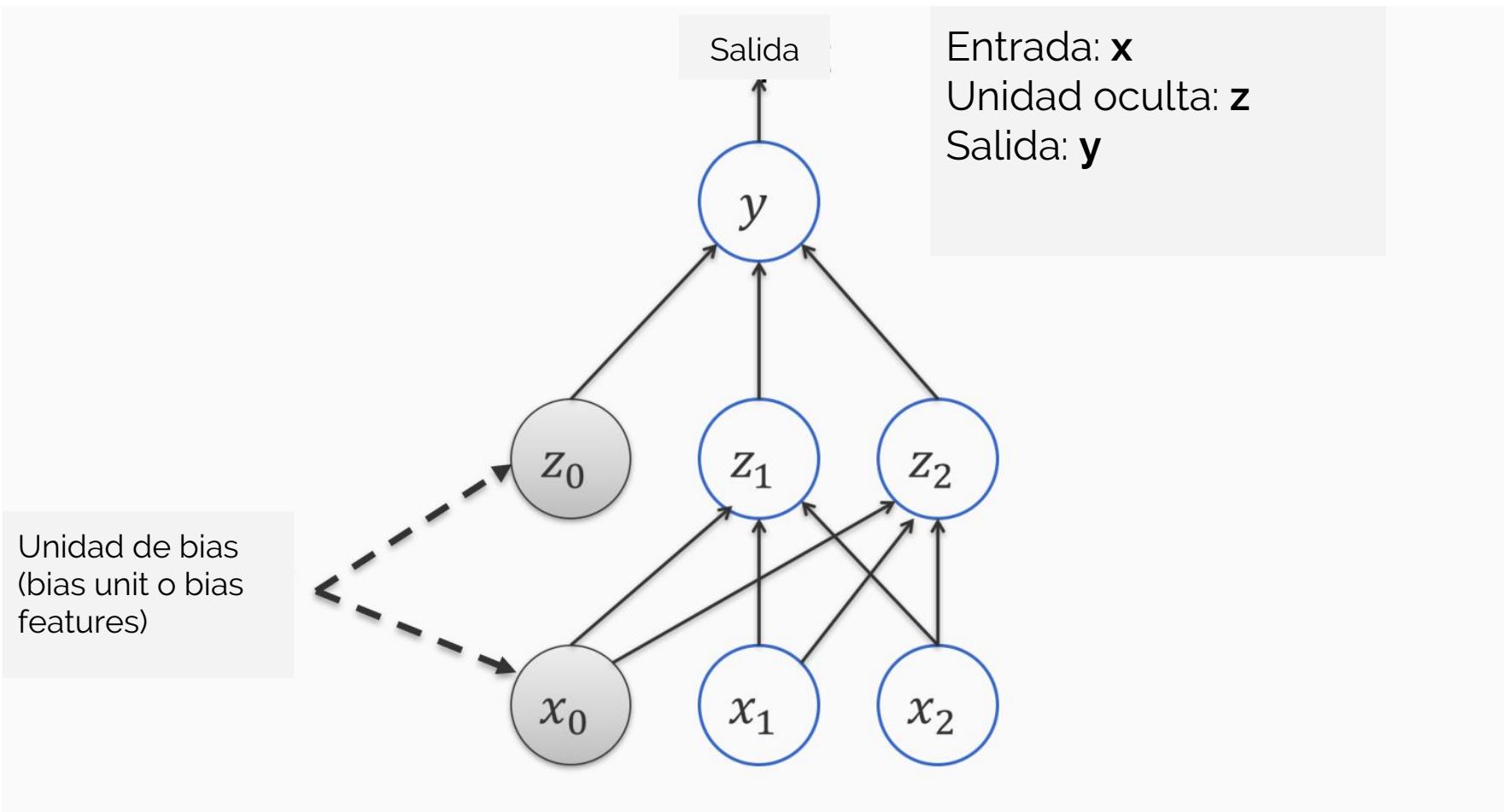


Red de ejemplo

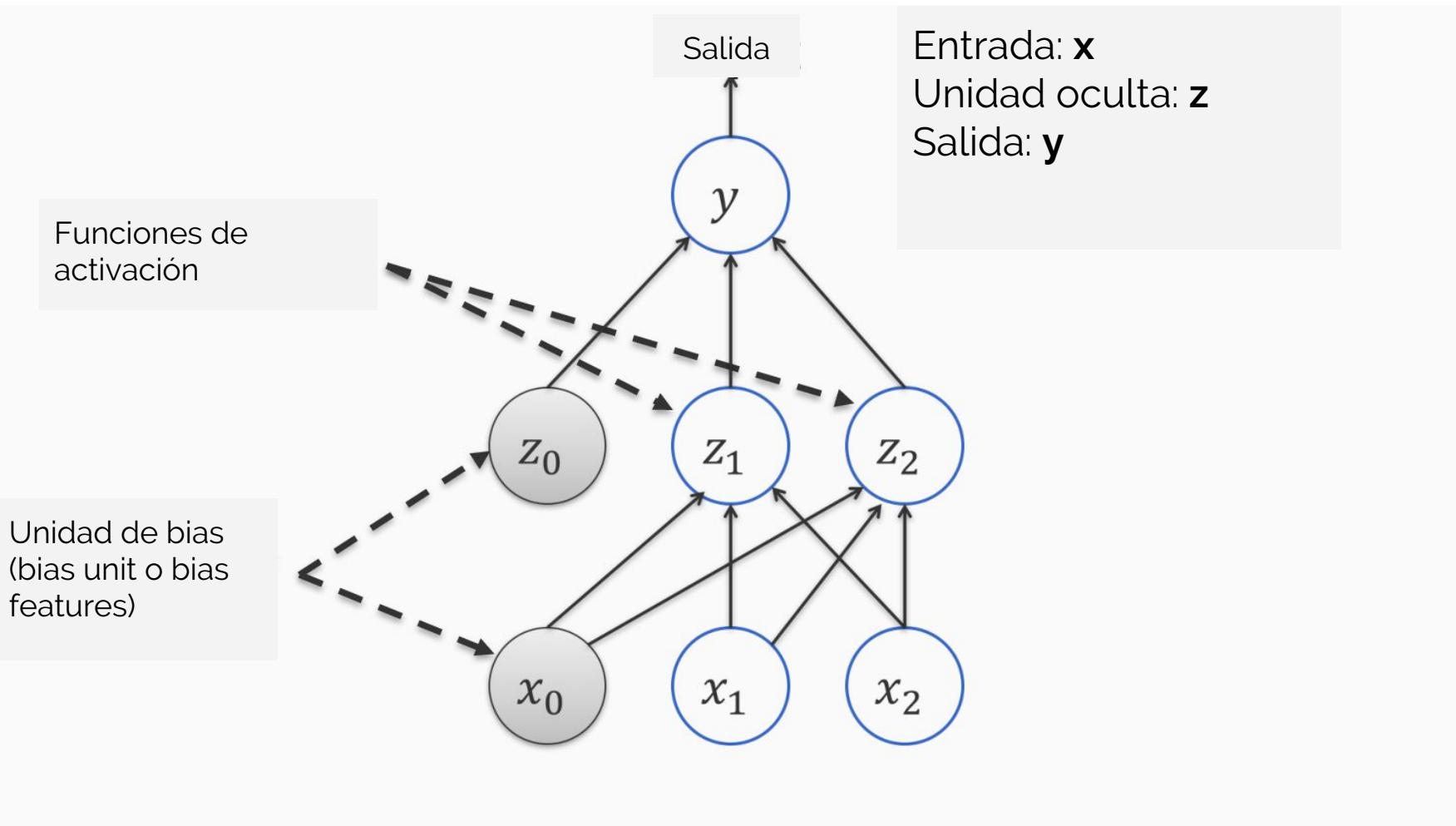


Entrada: \mathbf{x}
Unidad oculta: \mathbf{z}
Salida: \mathbf{y}

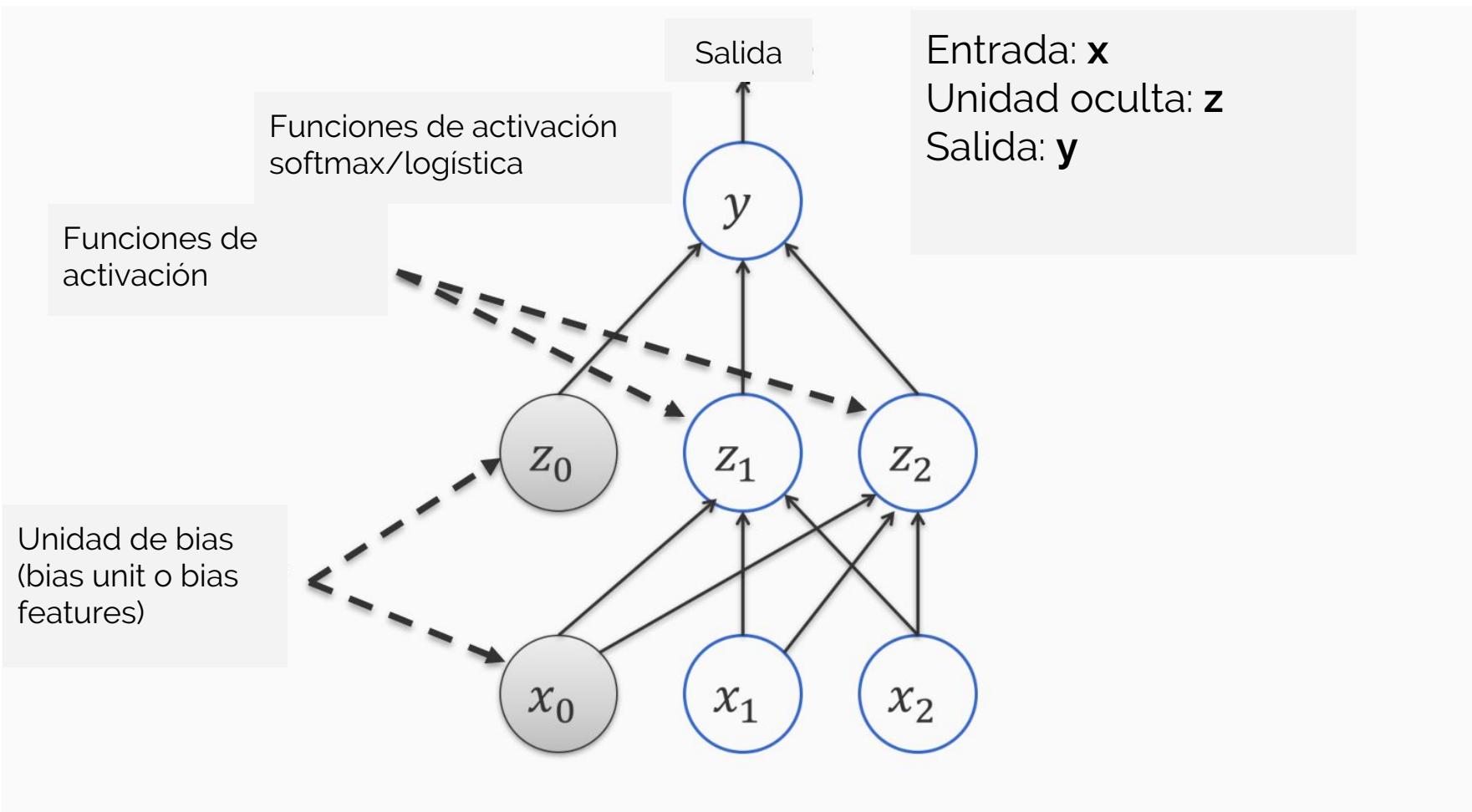
Red de ejemplo



Red de ejemplo

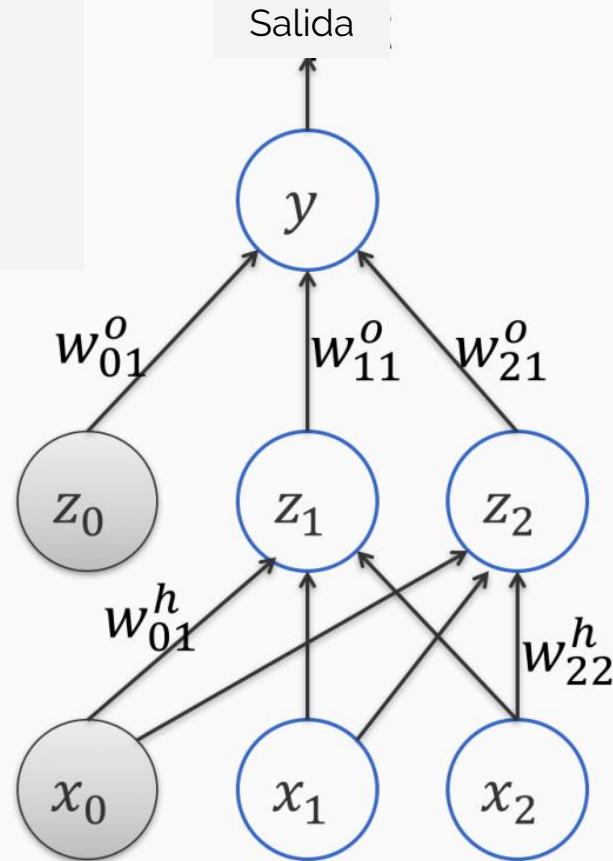


Red de ejemplo



Red de ejemplo

W Capa-Destino
desde,hacia

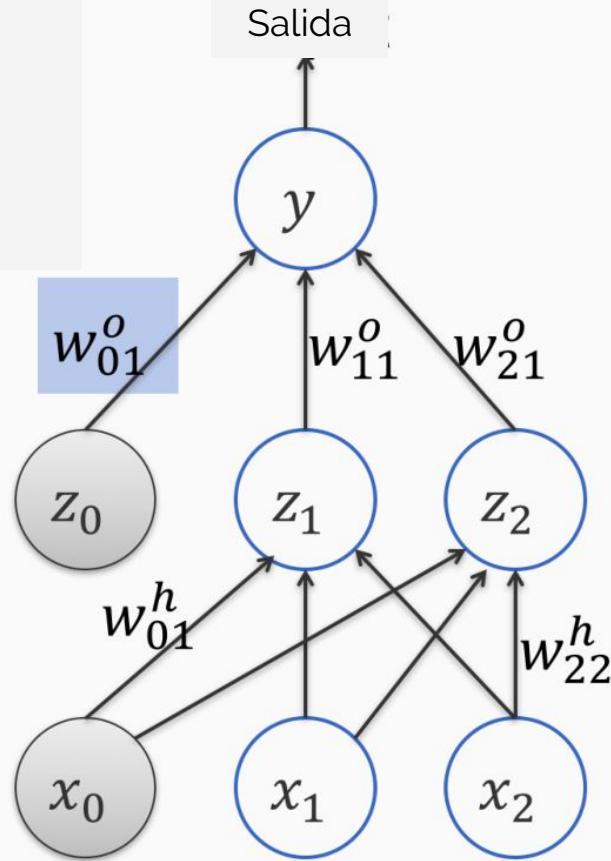


Red de ejemplo

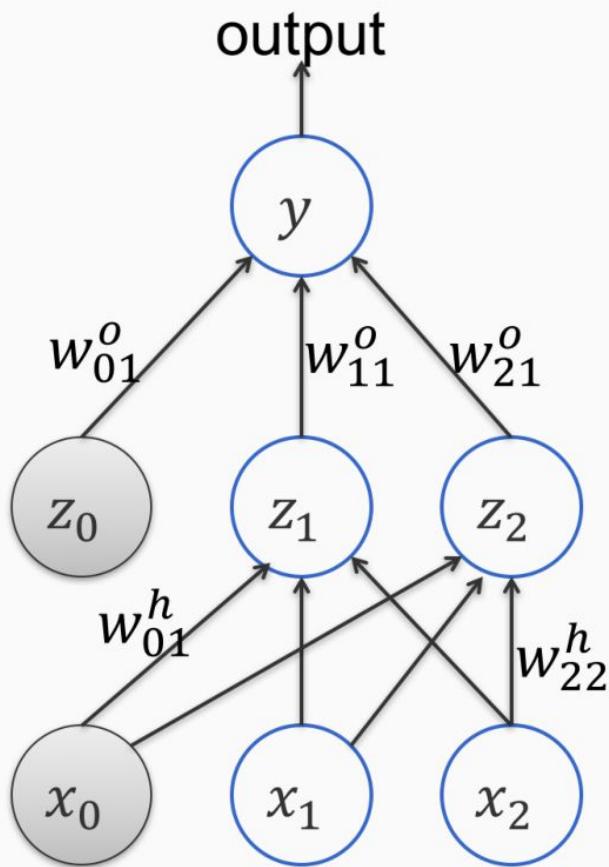
W Capa-Destino
desde,hacia

W^o
W₀₁

Desde la neurona #0 a
la neurona #1 de la
capa de salida (o)

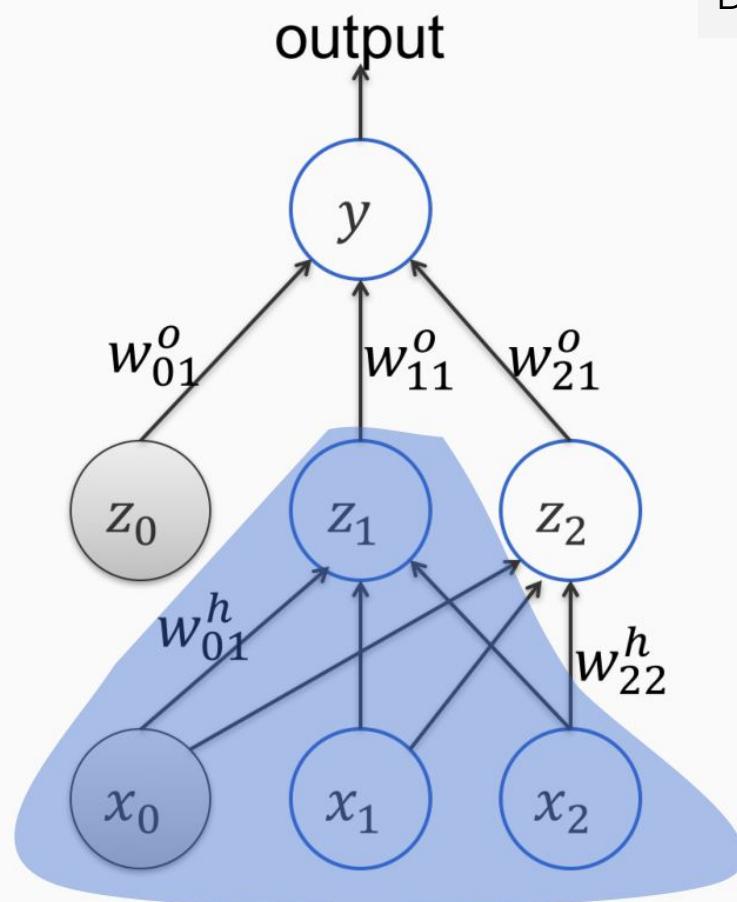


¿Cómo predecir un valor?



Dada una entrada \mathbf{x} , cómo se predice su valor?

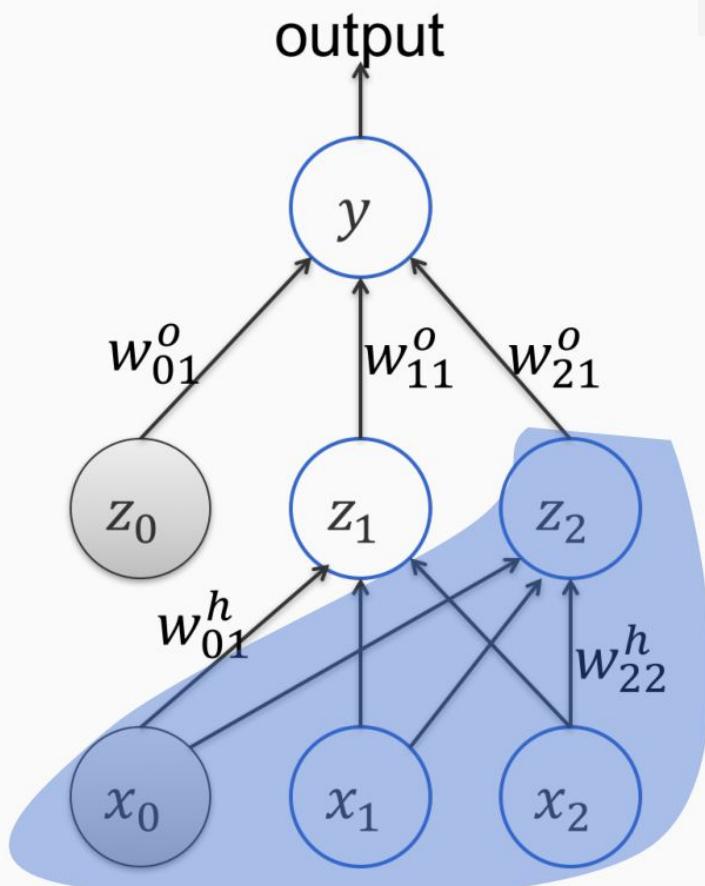
Proceso de pre-alimentación (forward pass)



Dada una entrada \mathbf{x} , cómo se predice su valor?

$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

Proceso de pre-alimentación (forward pass)

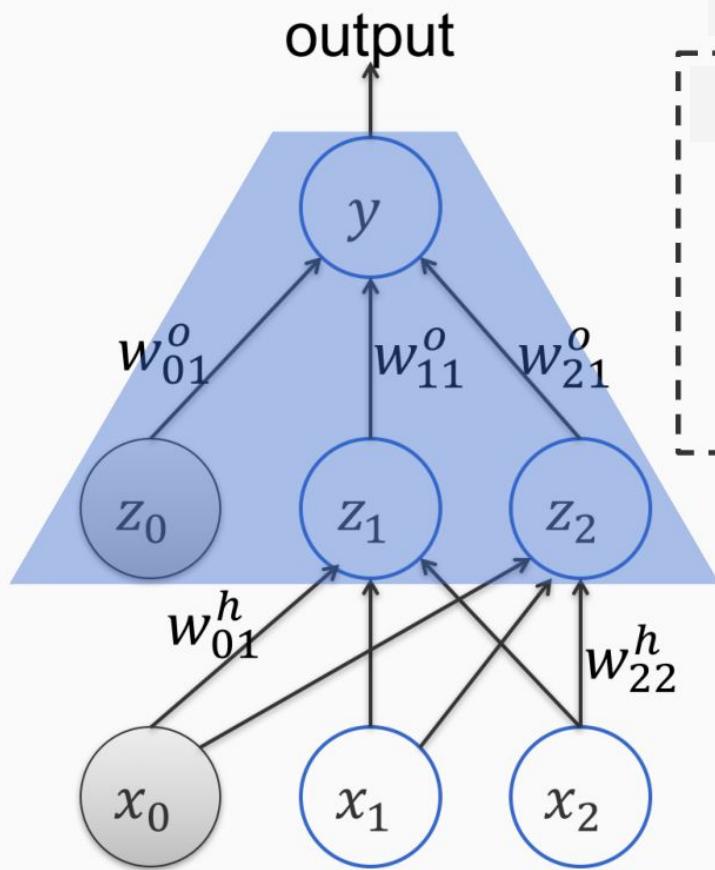


Dada una entrada \mathbf{x} , cómo se predice su valor?

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

Proceso de pre-alimentación (forward pass)



Dada una entrada \mathbf{x} , cómo se predice su valor?

Salida

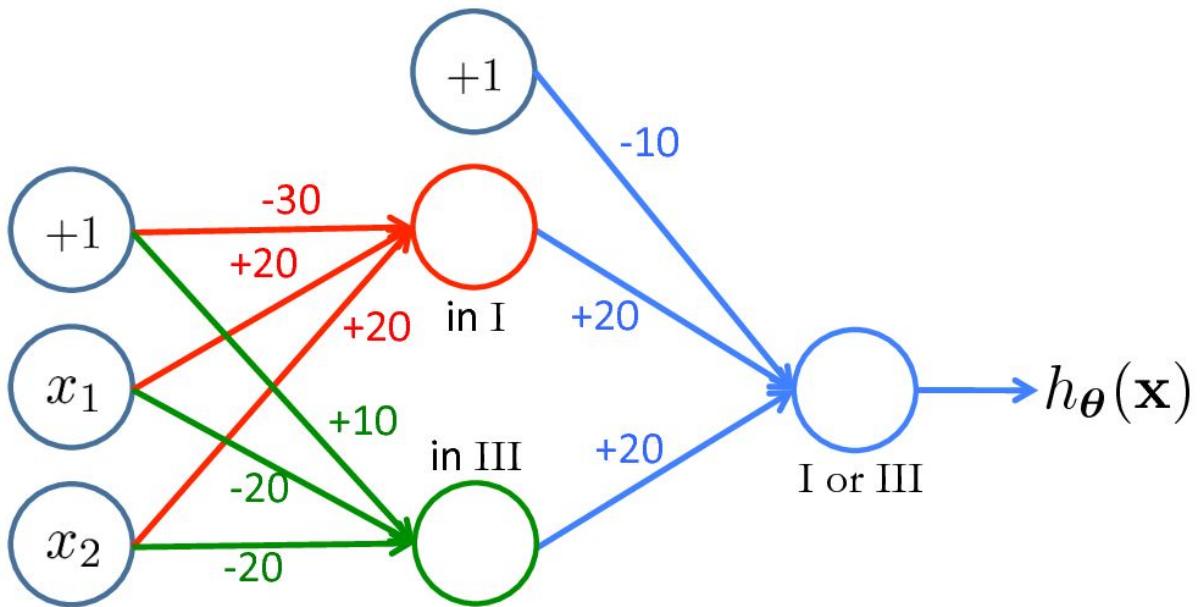
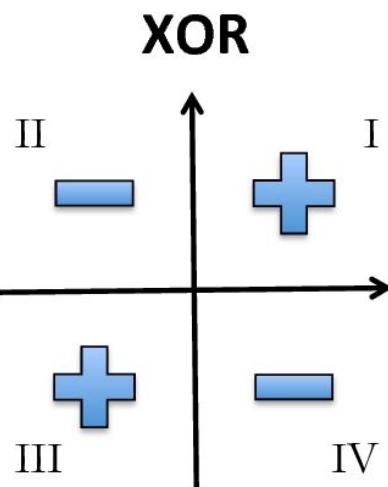
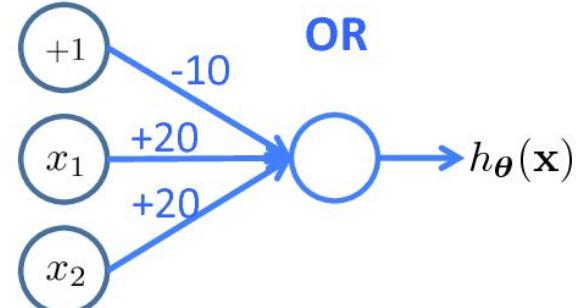
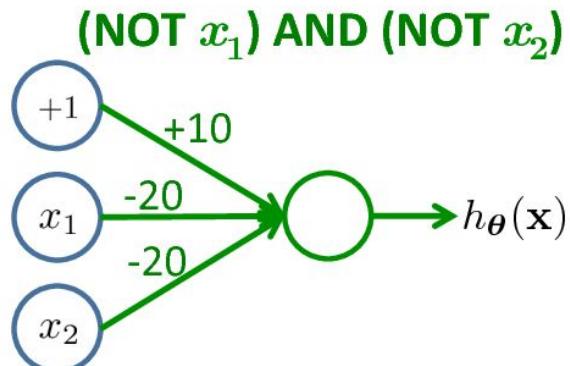
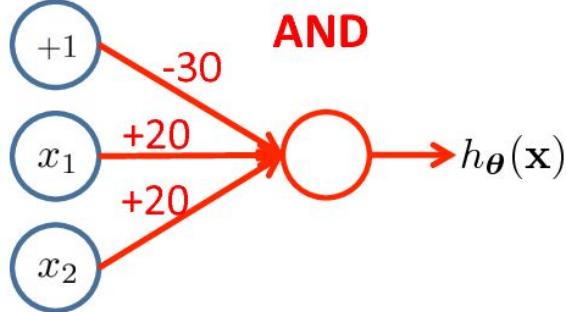
$$y = w_{01}^o + w_{11}^o z_1 + w_{21}^o z_2$$

$$z_2 = \sigma(w_{02}^h + w_{12}^h x_1 + w_{22}^h x_2)$$

$$z_1 = \sigma(w_{01}^h + w_{11}^h x_1 + w_{21}^h x_2)$$

XOR con redes neuronales

XOR con redes neuronales



Redes neuronales para aprender representaciones

Representaciones

Todo clasificador recibe como input una **representación** de los ejemplos

Machine Learning



Representaciones

Todo clasificador recibe como input una **representación** de los ejemplos

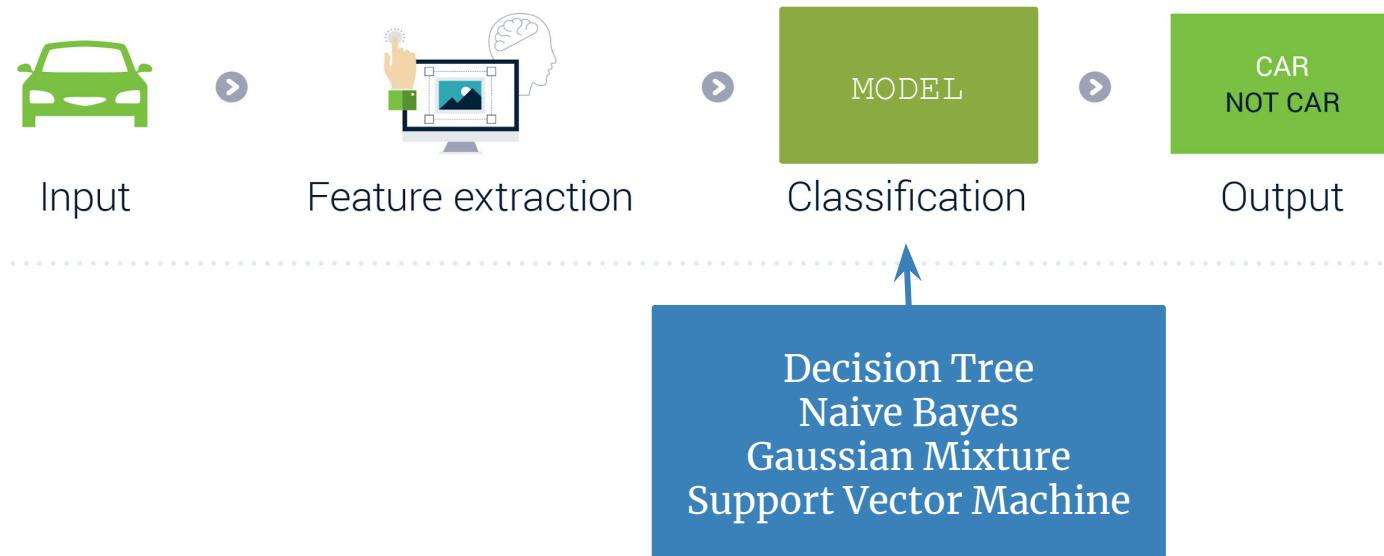
Machine Learning



Representaciones

Luego el clasificador transforma esa representación para obtener una etiqueta

Machine Learning



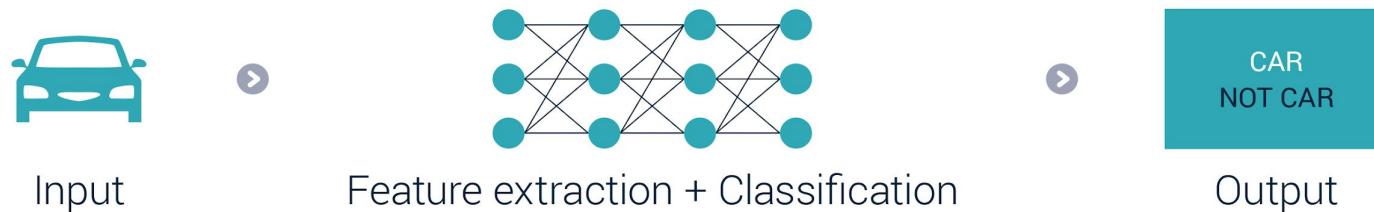
Representaciones

Las redes neuronales realizan **transformaciones** de la representación antes de obtener la etiqueta

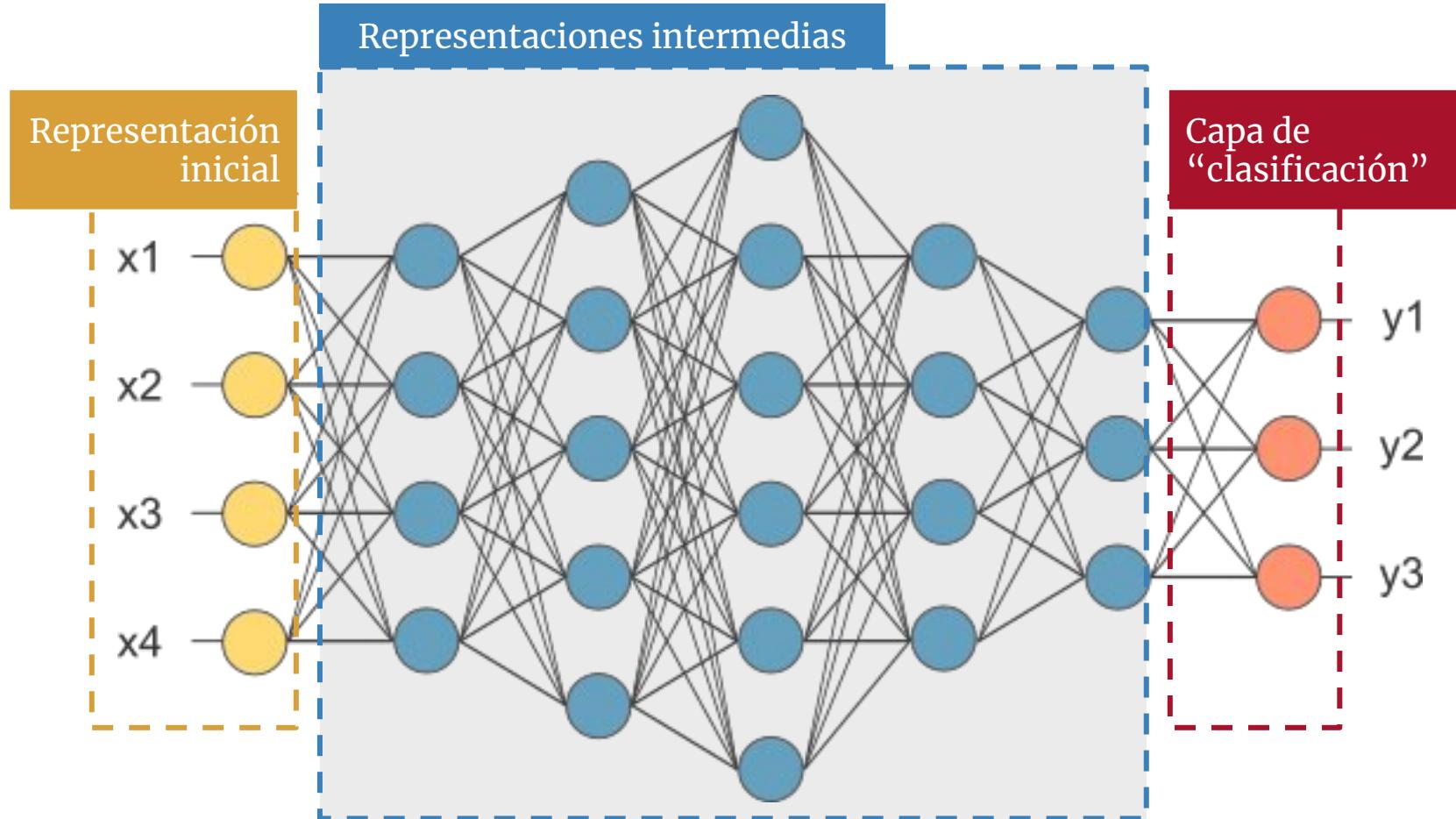
Machine Learning



Deep Learning



Otra forma de entender las redes neuronales



Todas las capas representan el mismo objeto

Handcrafted features

- Funcionan muy bien en tareas simples

Dataset del Titanic
Clasificación de Irises
PoS Tagging
Sentiment analysis (?)

Handcrafted features

- Funcionan muy bien en tareas simples
- Solamente se utilizan las características que se le ocurren al experto de dominio -> No descubrimos cosas nuevas.
- La representación suele estar ligada a la tarea de clasificación.
- Están limitados cuando el espacio a representar es infinito o no está totalmente entendido.

Handcrafted features

- Funcionan muy bien en tareas simples
- Solamente se utilizan las características que se le ocurren al experto de dominio -> No descubrimos cosas nuevas.
- La representación suele estar ligada a la tarea de clasificación.
- Están limitados cuando el espacio a representar es infinito o no está totalmente entendido.

Ej: ¿Cómo representamos el significado de las palabras?

La magia de las redes neuronales

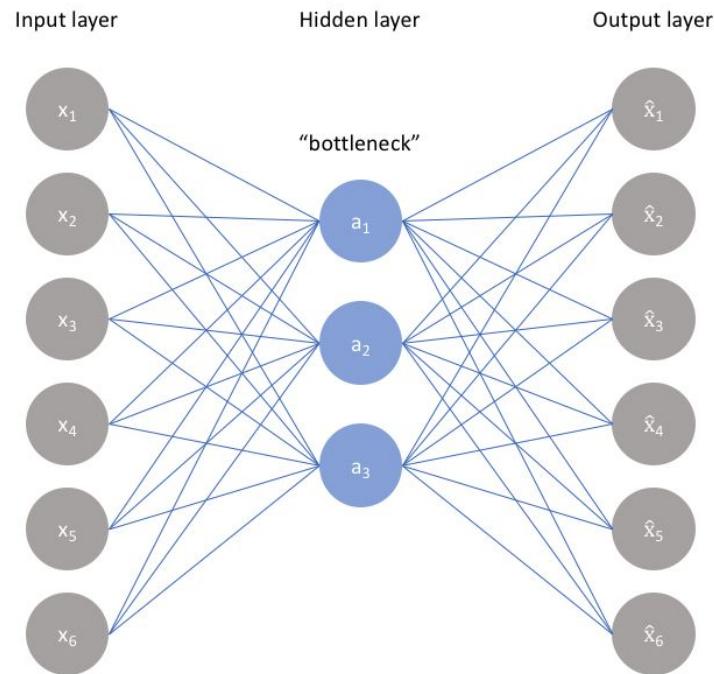
- Como las redes transforman los datos internamente, la representación inicial no es crítica ni depende de la tarea de clasificación.
- Descubren patrones nuevos automáticamente.
- Las representaciones son jerárquicas

La magia de las redes neuronales

- Como las redes transforman los datos internamente, la representación inicial no es crítica ni depende de la tarea de clasificación.
- Descubren patrones nuevos automáticamente.
- Las representaciones son jerárquicas
- Se estancan en mínimos locales y puntos de silla.
- Requieren mucho parámetros -> Requieren muchos datos para converger.

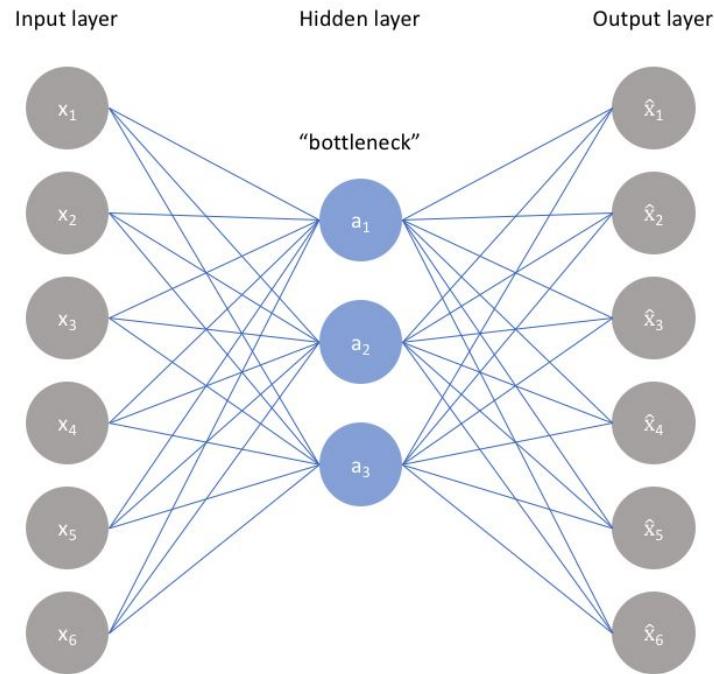
Ejemplo 1: Autoencoders

La tarea es predecir los valores exactos de la entrada



Ejemplo 1: Autoencoders

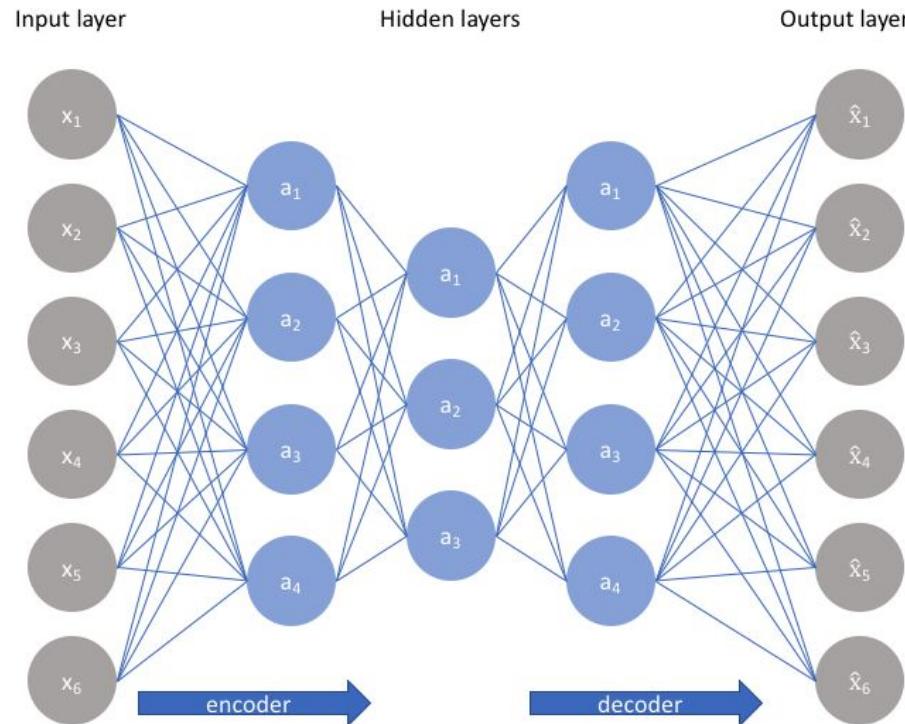
La tarea es predecir los valores exactos de la entrada



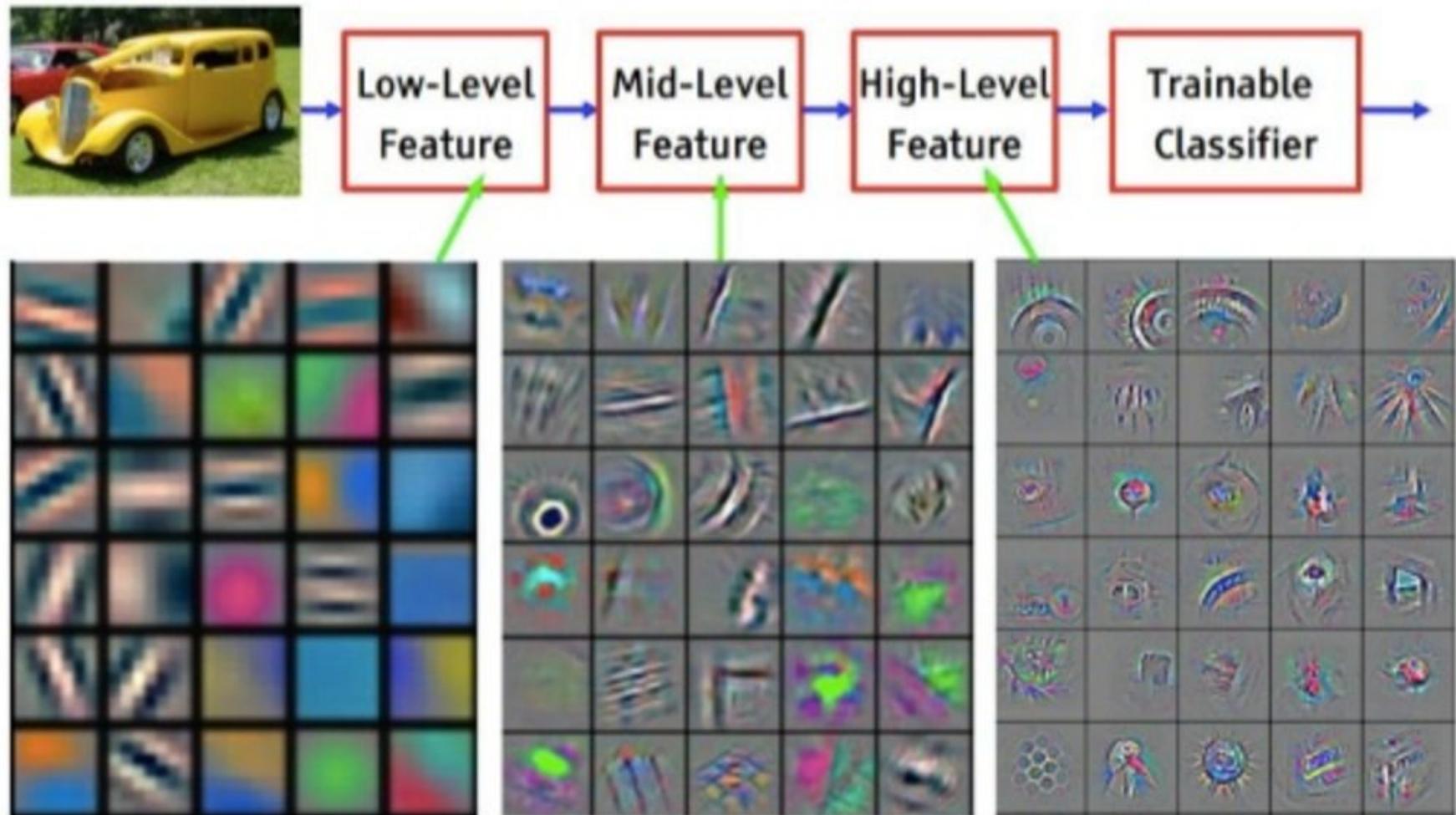
Obtenemos una representación “compacta” de la entrada que sea lo más fiel posible

Ejemplo 1: Autoencoders

Podemos usar más de una capa oculta -> Stacked autoencoder



Ejemplo 2: Convoluciones e imágenes



En conclusión...

- Las redes neuronales son clasificadores que pueden representar cualquier función.
- Se componen de muchas capas con funciones no lineales.
- Cada capa oculta transforma la entrada generando una nueva representación.

ENTRENAMIENTO

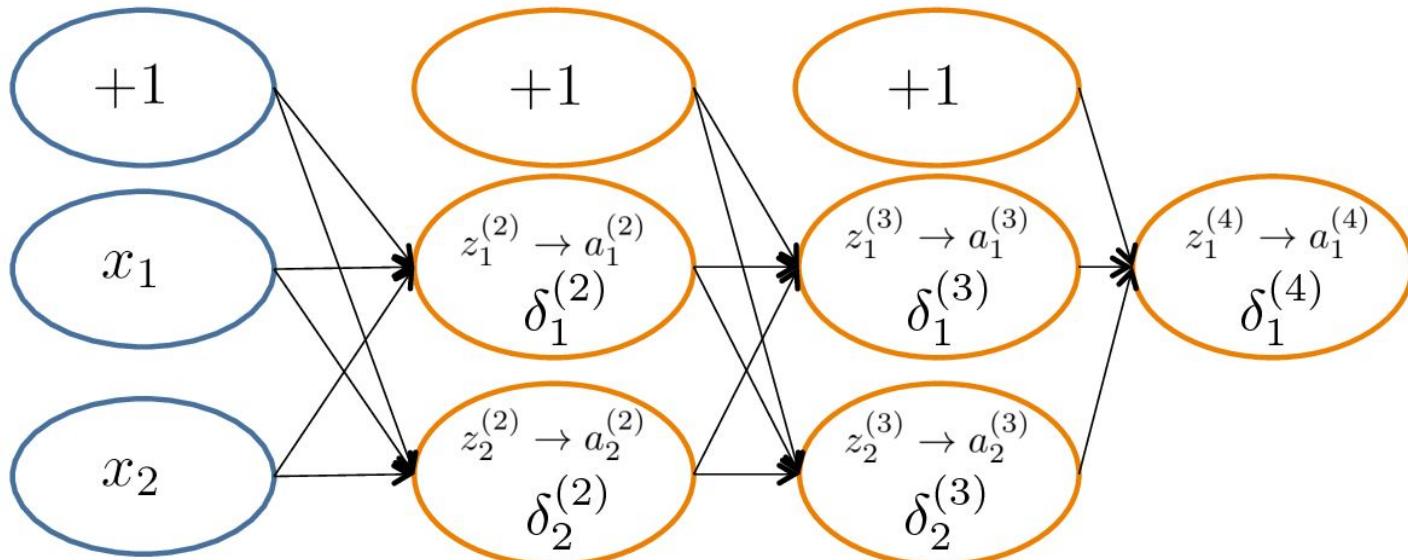
¿Cómo se entrena las redes neuronales?

Entrenamiento de la red neuronal

Intuición de la retropropagación

- Cada nodo j es "responsable" por una fracción del error $\delta_j^{(l)}$ en cada nodo de salida al que está conectado.
- $\delta_j^{(l)}$ es dividido de acuerdo a la fuerza de la conexión entre el nodo oculto y el de salida
- Luego la "culpa" es propagada hacia atrás para proveer los valores del error en la capa oculta.

Intuición de la retropropagación

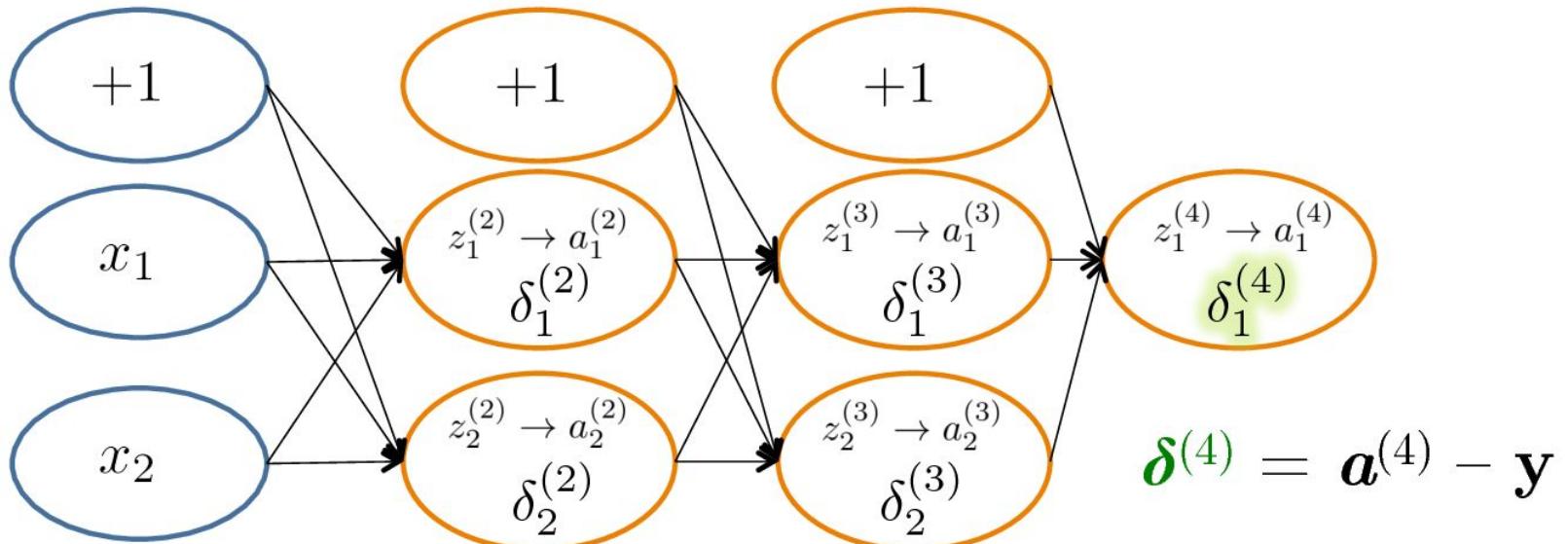


$\delta_j^{(l)}$ = error del nodo j en la capa l

Formalmente $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

donde $\text{cost}(\mathbf{x}_i) = y_i \log h_{\Theta}(\mathbf{x}_i) + (1 - y_i) \log(1 - h_{\Theta}(\mathbf{x}_i))$

Intuición de la retropropagación

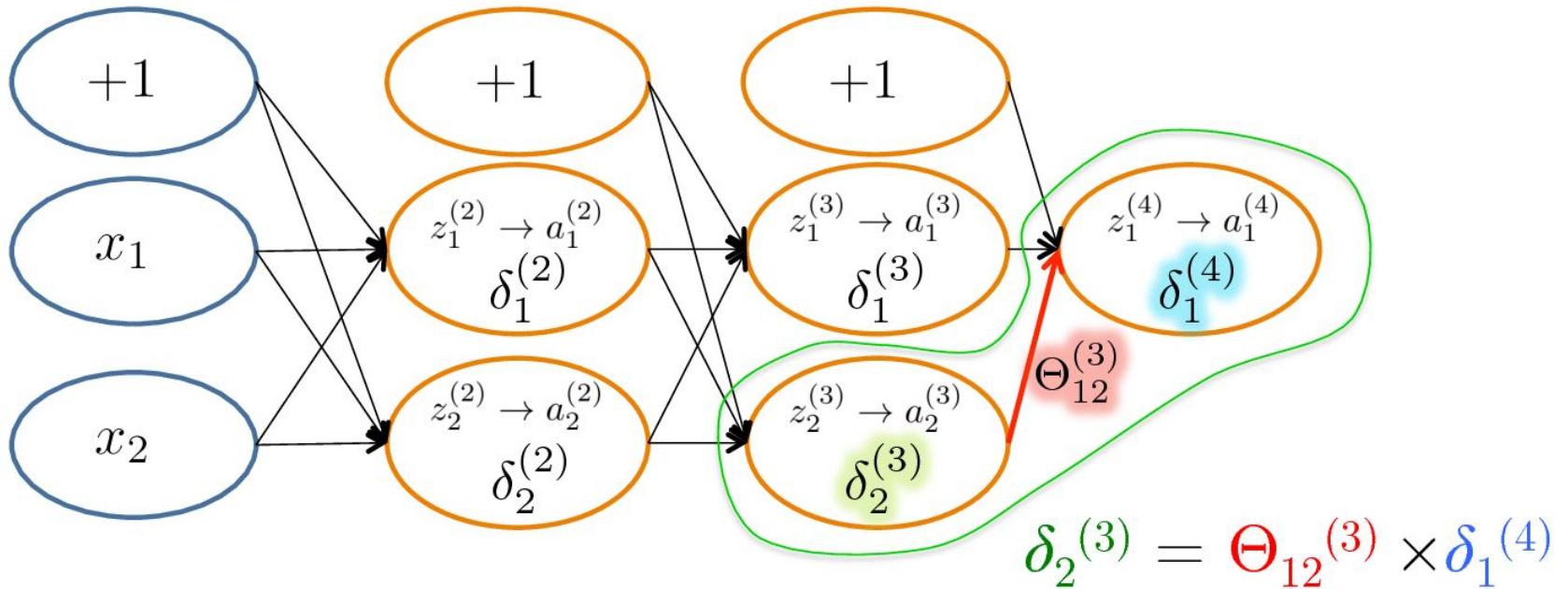


$\delta_j^{(l)}$ = error del nodo j en la capa l

Formalmente $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

donde $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Intuición de la retropropagación

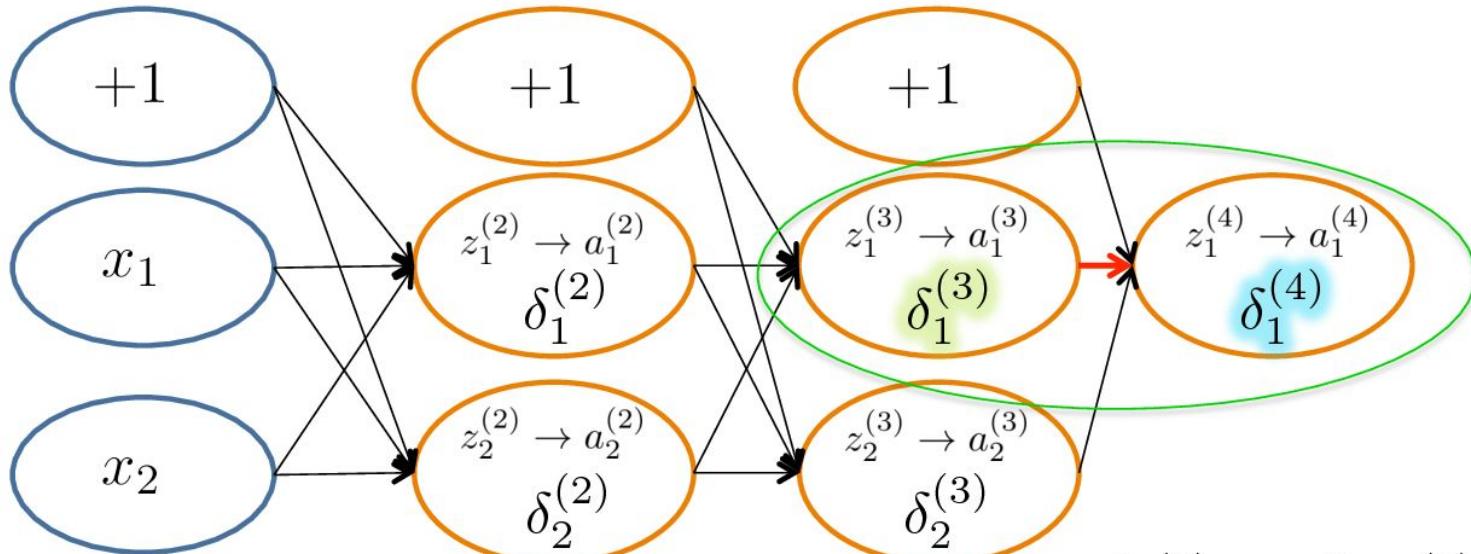


$\delta_j^{(l)}$ = error del nodo j en la capa l

Formalmente $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

donde $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Intuición de la retropropagación



$$\delta_2^{(3)} = \Theta_{12}^{(3)} \times \delta_1^{(4)}$$

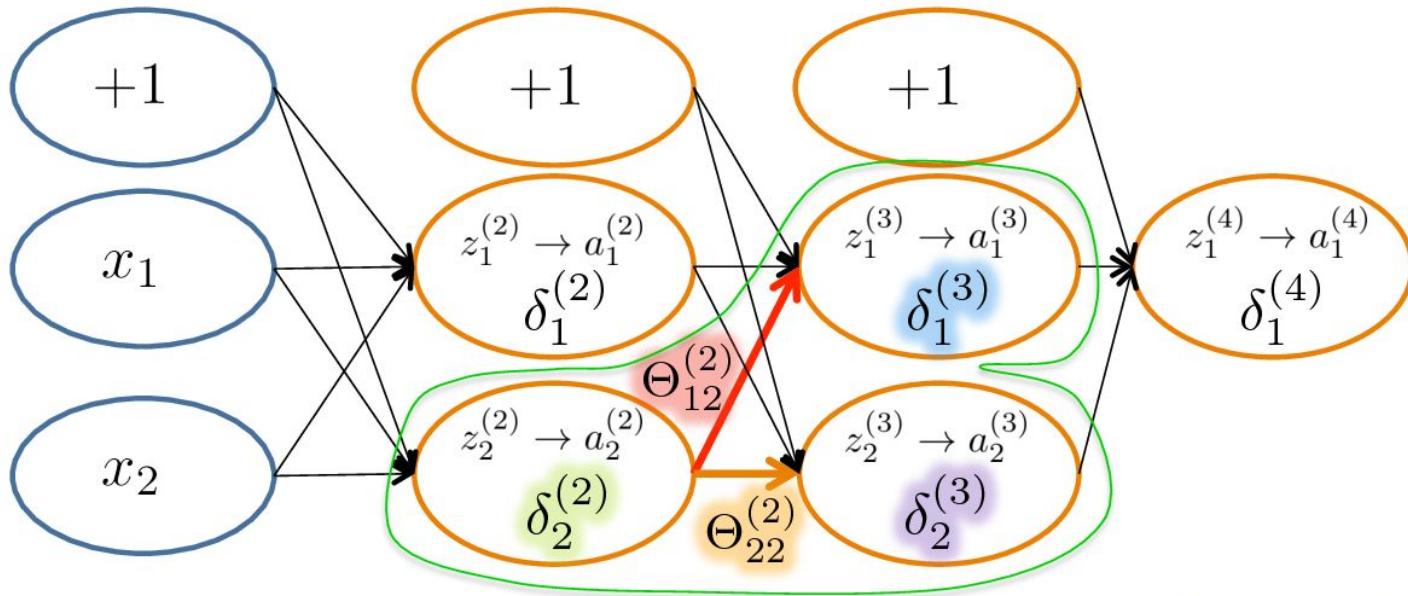
$$\delta_1^{(3)} = \Theta_{11}^{(3)} \times \delta_1^{(4)}$$

$\delta_j^{(l)}$ = error del nodo j en la capa l

Formalmente $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

donde $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$

Intuición de la retropropagación



$$\delta_2^{(2)} = \Theta_{12}^{(2)} \times \delta_1^{(3)} + \Theta_{22}^{(2)} \times \delta_2^{(3)}$$

$\delta_j^{(l)}$ = error del nodo j en la capa l

Formalmente $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} \text{cost}(\mathbf{x}_i)$

donde $\text{cost}(\mathbf{x}_i) = y_i \log h_\Theta(\mathbf{x}_i) + (1 - y_i) \log(1 - h_\Theta(\mathbf{x}_i))$



“

Backprop is the cockroach of machine learning. It's ugly, and annoying, but you just can't get rid of it

- Geoff Hinton (or someone in his behalf)

Detalles de implementación

Clasificación multiclase: OVR

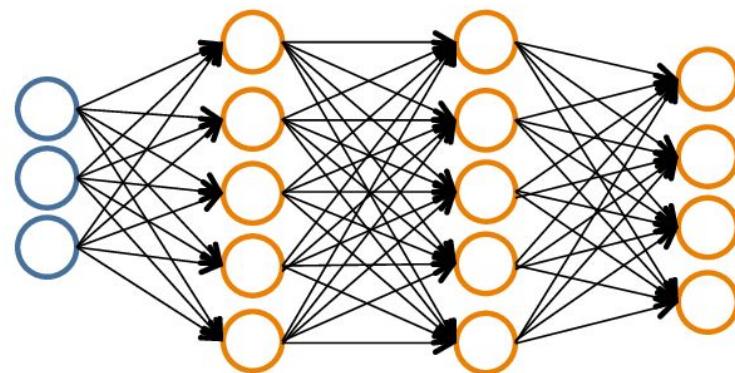


Peatón

Auto

Moto

Camión



$$h_{\Theta}(\mathbf{x}) \in \mathbb{R}^K$$

Queremos

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Para peatón

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Para auto

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Para moto

$$h_{\Theta}(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Para camión

Inicialización de pesos

- Los pesos iniciales no pueden ser uniformes
 - Con pesos uniformes la actualización de los pesos es idéntica y la red no aprende
- Los pesos se inicializan aleatoriamente
 - Se usa una distribución normal dividido por la raíz cuadrada de la dimensión de la capa
 - Esto hace que el valor aleatorio esté alrededor de 1 lo que evita problemas de desvanecimiento del gradiente

Práctico

Configuración del entorno en Keras

Etapas de un proyecto de ML

	Procesado de datos	Entrenamiento	Evaluación	Deployment
Resultado	Dataset curado	Modelo entrenado	Selección de modelo	Pipeline
Tareas	Limpieza Descripción Generación de la representación inicial	Implementación del modelo Selección de hiperparámetros Monitoreo del entrenamiento	Generación de predicciones Selección de métricas Comparación de resultados	Re-implementación del pipeline ??
Recursos	CPU / Disco	CPU / GPU / RAM	Disco	CPU / GPU / RAM
Herramientas	Scripts de Python / Bash Spark (Big Data) Notebooks	Sklearn Keras/Tensorflow/Theano Slurm (colas de trabajo) Scripts bash para entrenar	Notebooks Visualizaciones Hojas de cálculo	?? Docker

Créditos del contenido

Esta presentación está basada en el trabajo compilado por [Eric Eaton](#) y la maravillosa comunidad de académicos y profesores del área de aprendizaje automático que dispusieron de manera libre su material en internet.

Las filminas que explican el algoritmo de clasificación de las redes neuronales están basadas en las filminas del curso de la universidad de UTAH por [Vivek Srikumar](#).

Slides Credits

Special thanks to all the people who made and released these awesome resources for free:

- Minicons by Webalys
- Presentation template by SlidesCarnival
- Photographs by Unsplash