Introducción a Redes Neuronales Artificiales

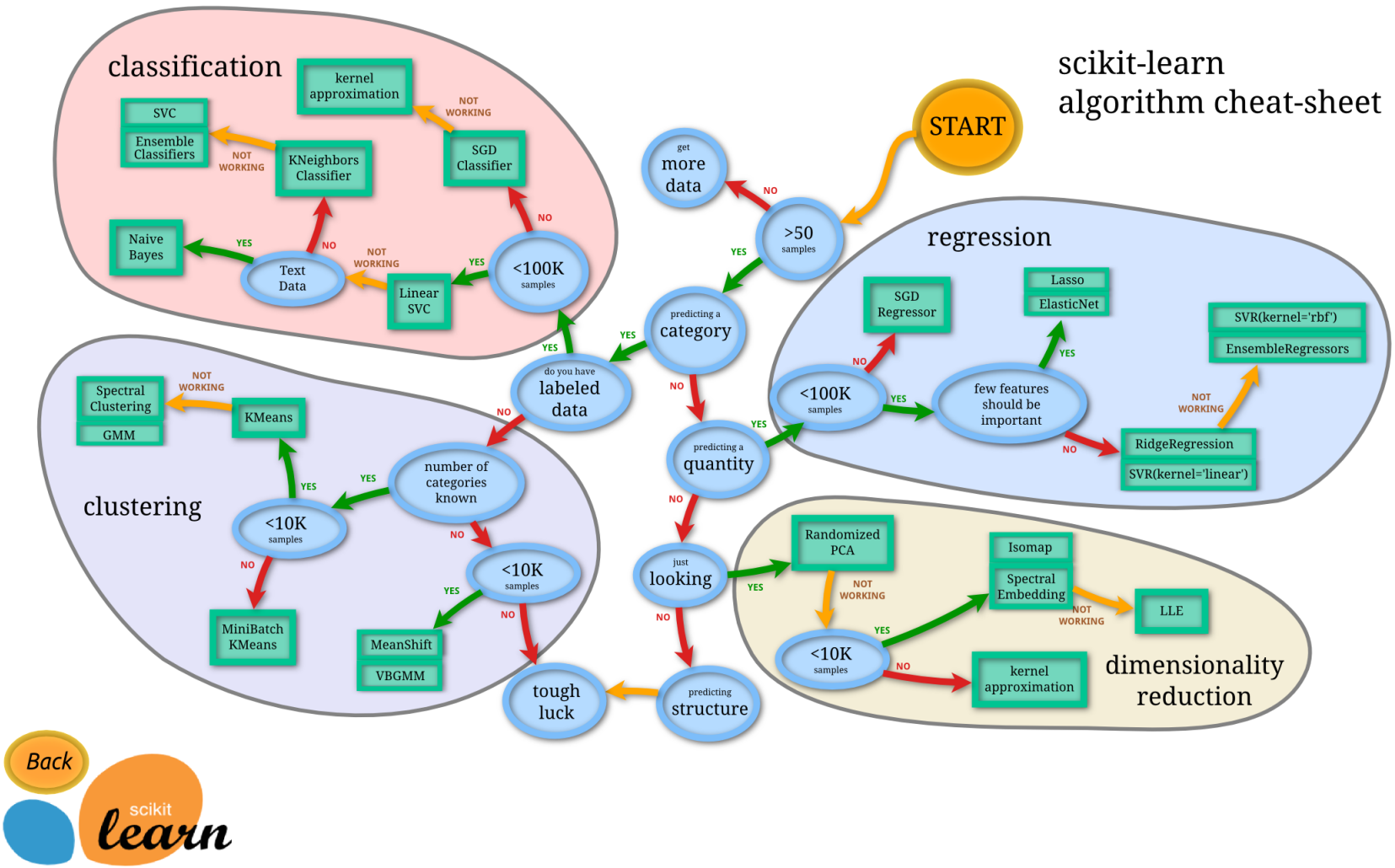
A. M. Alvarez-Meza, Ph.D.
D. F. Collazos-Huertas, Ph.D(c)
amalvarezme@unal.edu.co, dfcollazos@unal.edu.co

Analítica de datos
Departamento de ing. eléctrica, electrónica y computación
Universidad Nacional de Colombia-sede Manizales

Contenido

Contenido

Métodos clásicos de aprendizaje de máquina - Scikit-learn



Artificial neural networks (ANNs): Motivación

Google Images

YouTube

Hey Siri

Las ANNs están en el centro del aprendizaje automática actual, son versátiles, potentes y escalables, lo que las hace ideales para abordar tareas sobre grandes cantidades de datos.

De las neuronas biológicas a las artificiales

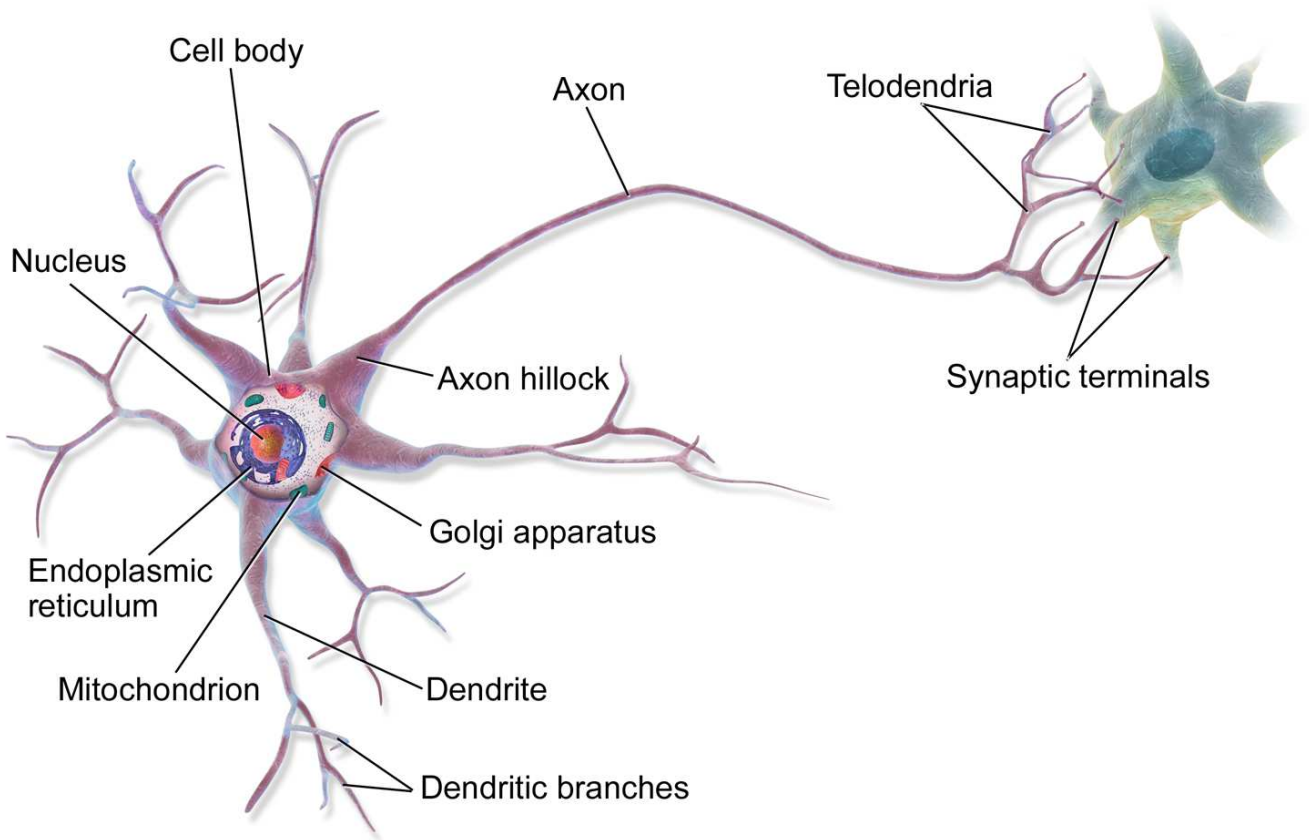
- Introducidos en **1943** por el *Warren McCulloch* y *Walter Pitts*.Modelo computacional de cómo las neuronas biológicas podrían trabajar juntas en los cerebros de los animales.
- En la década **1960** surgieron los primeros éxitos de las ANNs. Sin embargo, los fondos volaron a otra parte y las ANNs entraron en un largo invierno.
- A principios de los **80's** hubo un resurgimiento del interés por el **conexionismo**, se inventaron nuevas arquitecturas y se desarrollaron mejores técnicas de entrenamiento.
- En la década de 1990 se inventaron otras poderosas técnicas de *Machine Learning*, como SVMs, por lo que una vez más el estudio de las redes neuronales entró en un largo invierno.

Por qué esta ola es diferente?

- **Gran cantidad de datos** disponibles.
- El aumento de la potencia informática (Cantidad vs. Tiempo).
- Producción de **tarjetas GPU** potentes.
- Los **algoritmos de entrenamiento** han sido mejorados.
- Algunas limitaciones teóricas de los ANN han resultado ser benignas en la práctica (problema de óptimos locales).
- Las ANNs parecen haber entrado en un círculo virtuoso de financiación y progreso.

Contenido

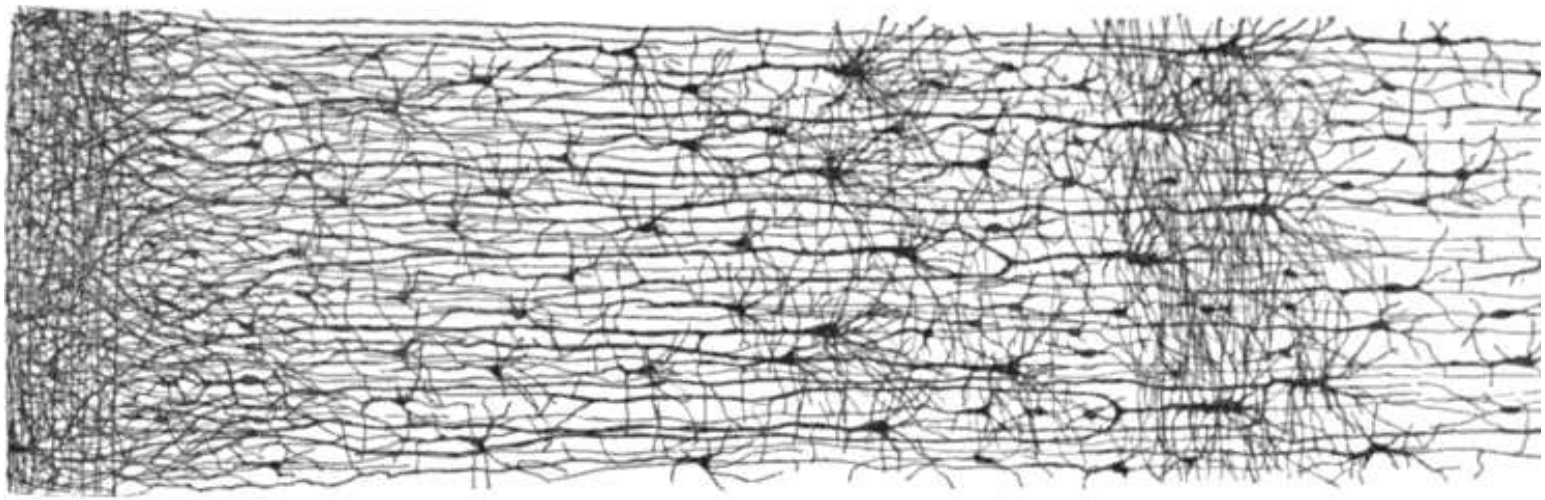
Neurona I



Es una célula que se encuentra principalmente en la corteza cerebral animal, compuesta por: el **núcleo** y muchas extensiones ramificadas llamadas **dendritas**, más una extensión muy larga llamada **axón**.

Neurona II

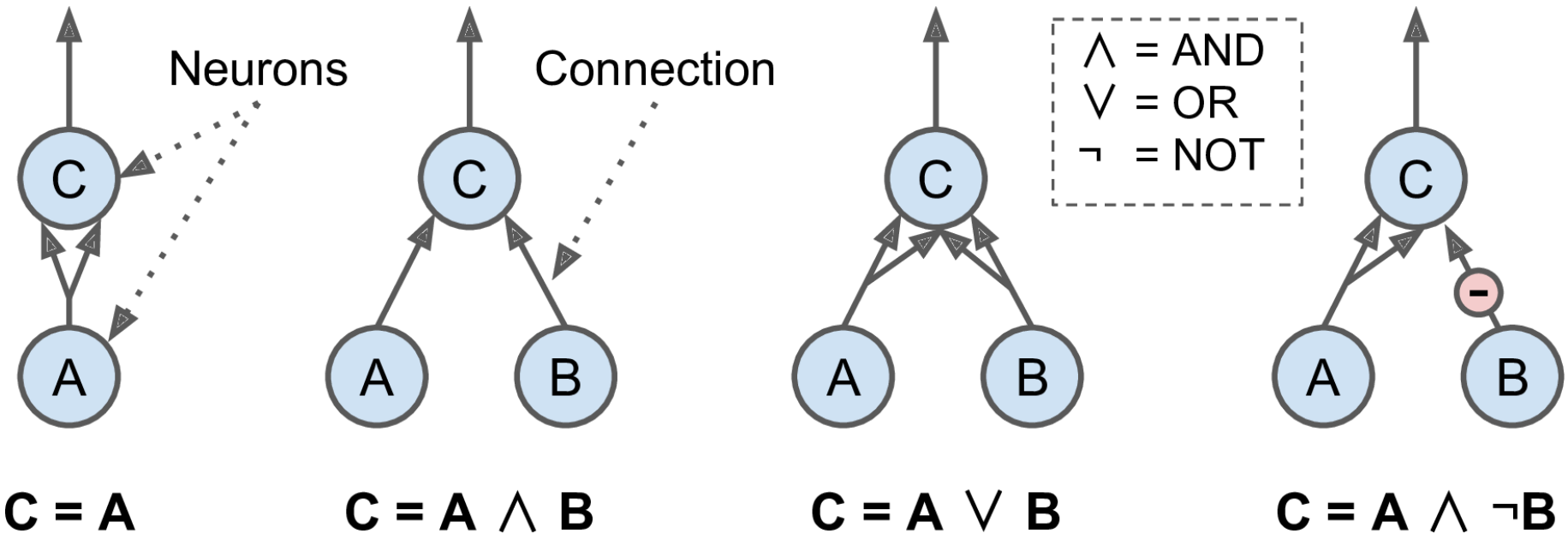
- Las neuronas biológicas individuales parecen comportarse de una manera bastante simple, pero están organizadas en una vasta red de miles de millones de neuronas, cada neurona típicamente **conectada a miles de otras neuronas**.



Contenido

Cálculos lógicos con neuronas I

Modelo de neurona artificial: Tiene una o más entradas binarias (encendido/apagado) y una salida binaria. La neurona artificial simplemente activa su salida cuando más de un cierto número de sus entradas están activas.



Se supone que una neurona se activa cuando al menos dos de sus entradas están activas.

Cálculos lógicos con neuronas II

- Función de identidad: si la neurona **A** está activada, entonces la neurona **C** también se activa.

- AND lógico: la neurona **C** se activa solo cuando ambas neuronas **A** y **B** están activadas.

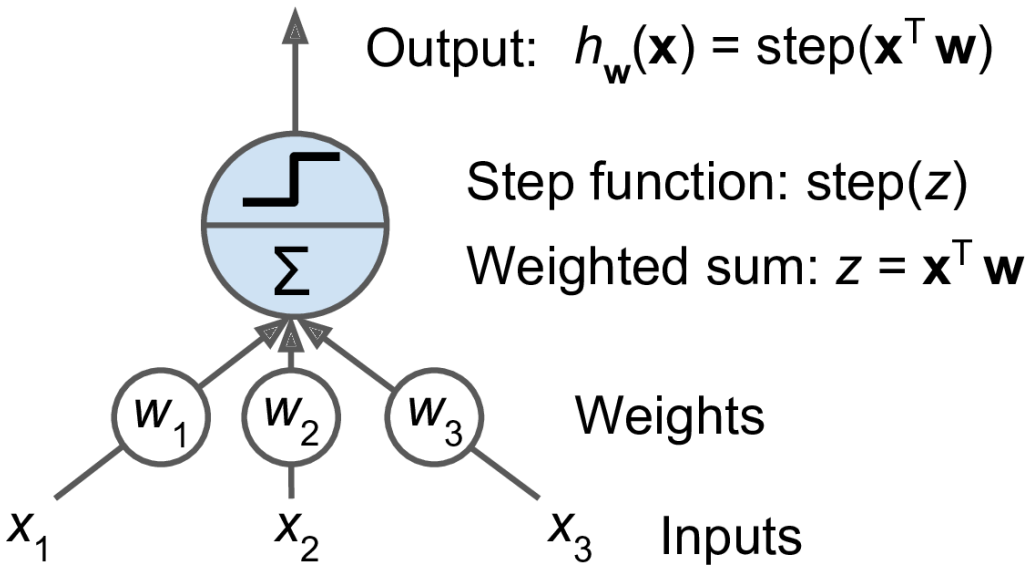
- OR lógico: la neurona **C** se activa si se activa la neurona **A** o la neurona **B** (o ambas).

- NOT lógico: la neurona **C** está activa cuando la neurona **B** está apagada, y viceversa.

Podemos fácilmente imaginar cómo se pueden combinar estas redes para calcular expresiones lógicas complejas.

Contenido

Perceptrón I



- El **Perceptrón** es una de las arquitecturas ANN más simples, inventada en 1957 por Frank Rosenblatt.
- Se basa en una neurona artificial llamada unidad lógica de umbral (TLU) o, a veces, una unidad de umbral lineal (LTU).

Las entradas y salidas pueden ser **números reales** (en lugar de valores binarios de activación/desactivación) y cada conexión de entrada está asociada a un **peso**.

Perceptrón II

- La TLU calcula una suma ponderada de sus entradas $z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \boldsymbol{X}^T \boldsymbol{W}$, luego aplica una *step function* a esa suma: $h_w(\boldsymbol{X}) = step(z)$, donde $z = \boldsymbol{X}^T \boldsymbol{W}$.
- La *step function* más común utilizada en Perceptrons es la **Heaviside**.

$$heaviside(z) = \begin{cases} 0 & if \quad z < 0 \\ 1 & otherwise \end{cases}$$

- A veces se usa la función de signo en su lugar.

$$sgn(z) = \begin{cases} -1 & if \quad z < 0 \\ 0 & if \quad z = 0 \\ +1 & if \quad z > 0 \end{cases}$$

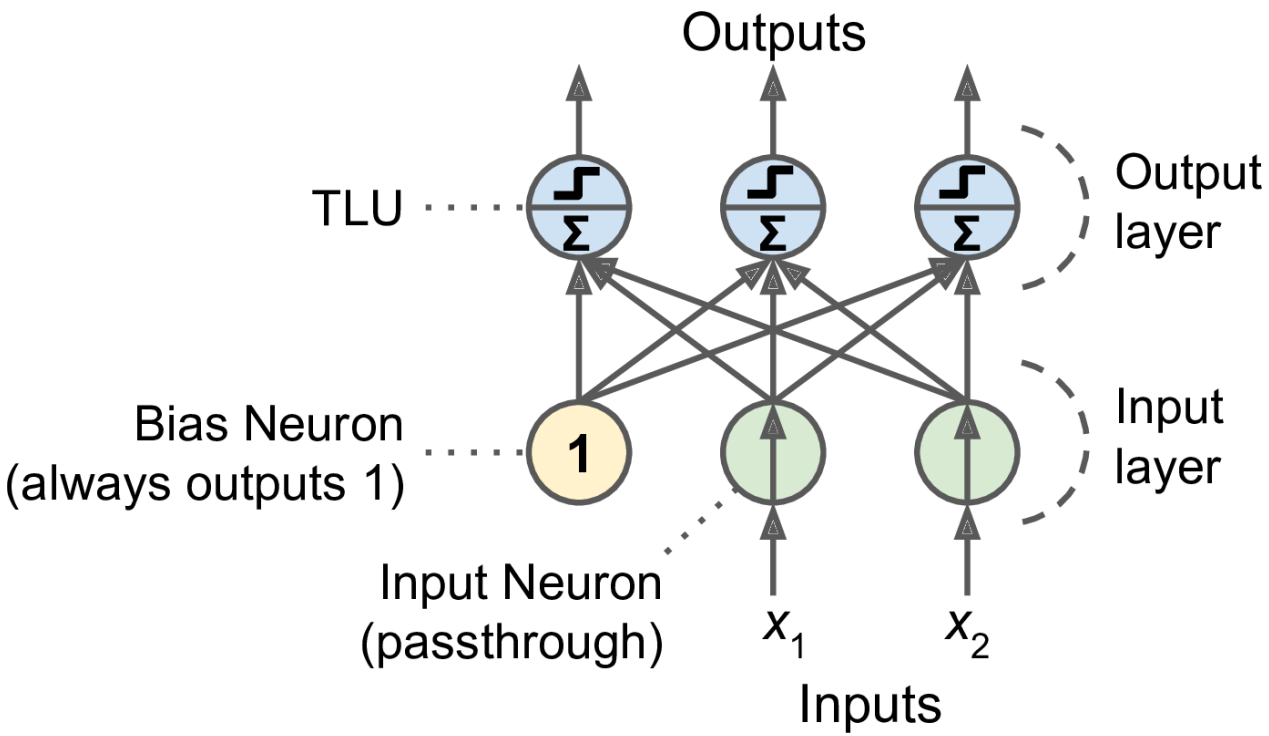
Se calcula una combinación lineal de las entradas y, si el resultado excede un umbral, genera la clase positiva o la clase negativa

Perceptrón III

- Un **perceptrón** está compuesto por una sola capa de TLU conectada a todas las entradas.
- Cuando todas las **neuronas** en una capa **están conectadas** a cada neurona en la **capa anterior**, se define como una capa completamente conectada o densa.
- Todas las neuronas de entrada forman la **capa de entrada**.
- Generalmente se agrega una característica de sesgo adicional ($x_0 = 1$), representada por la **neurona de sesgo**.

Perceptrón IV

Ejemplo: Un Perceptrón con dos entradas y tres salidas.



Este Perceptrón puede clasificar instancias simultáneamente en tres clases binarias diferentes **(clasificador de salida múltiple)**.

Perceptrón V

Gracias a la magia del álgebra lineal, es posible calcular eficientemente los resultados de una capa de neuronas artificiales para varias instancias:

$$h_{\boldsymbol{W},\boldsymbol{b}}(\boldsymbol{X}) = \phi\left(\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}\right) \tag{1}$$

* $\boldsymbol{X}$ representa la matriz de características de entrada.
* La matriz de peso $\boldsymbol{W}$ contiene todos los pesos de conexión.
* El vector bias $\boldsymbol{b}$ contiene todos los pesos de conexión entre la neurona de bias y las neuronas artificiales.
* La función $\phi\left(\cdot\right)$ se conoce como función de activación.

Perceptrón VI: Cómo se entrena?

El algoritmo de entrenamiento de Perceptrón se inspiró en gran medida en la **regla de Hebb**.

- "El peso de conexión entre dos neuronas aumenta cuando tienen la misma salida."
- Se **tiene en cuenta el error cometido por la red**; refuerza las conexiones que ayudan a reducir el error.
- Específicamente, el perceptrón se alimenta de una instancia de entrenamiento a la vez, y **para cada instancia hace sus predicciones**.

Para cada neurona de salida que produjo una predicción incorrecta, **se refuerzan los pesos de conexión de las entradas que habrían contribuido a la predicción correcta**.

Perceptrón VII: Cómo se entrena?

Regla de aprendizaje de perceptrón **(actualización de peso)**:

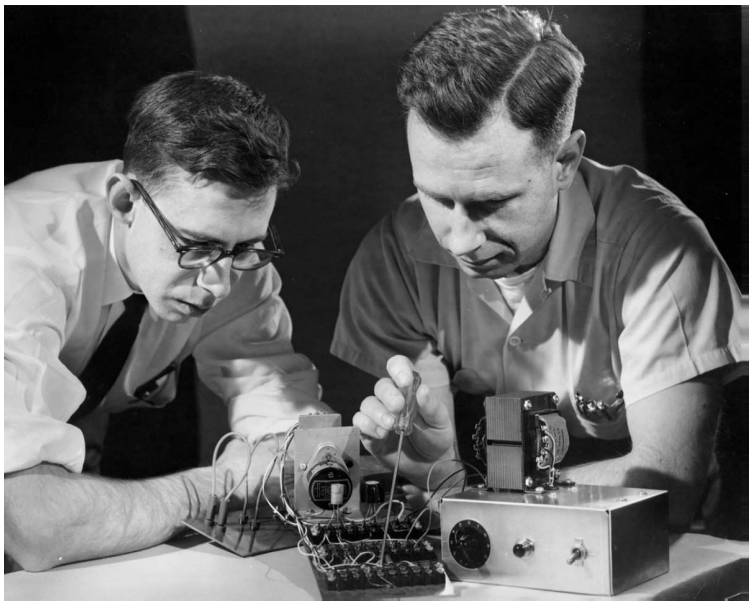
$$w_{i,j}^{next\_step} = w_{i,j} + \eta \left(y_j - \hat{y}_j\right) x_i \tag{2}$$

- wi, j es el peso de conexión entre la i-ésima neurona de entrada y la j-ésima neurona de salida.
- x_i es el i-ésimo valor de **entrada** de la instancia de entrenamiento actual.
- $\hat{y}_j$ es la **salida** de la neurona de salida j para la instancia de entrenamiento actual.
- y_j es la salida objetivo de la neurona de salida j para la instancia de entrenamiento actual.
- η es la tasa de aprendizaje.

Teorema de convergencia del perceptrón

El **límite de decisión** de cada neurona de salida es lineal, por lo que los perceptrones son incapaces de aprender patrones complejos.

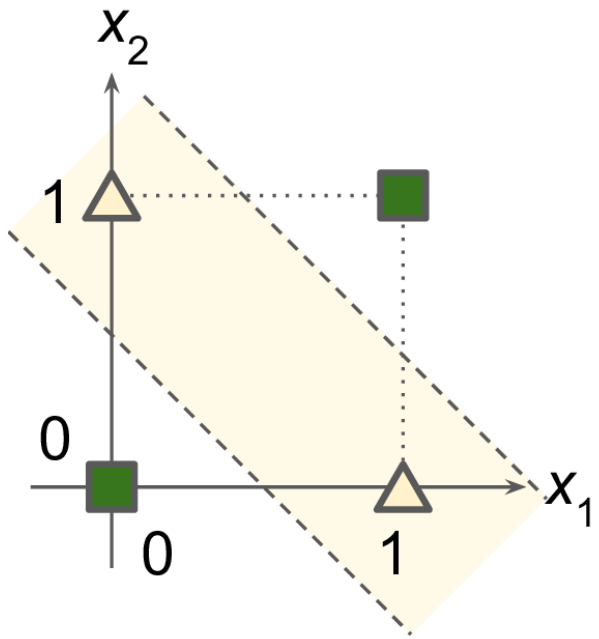
Sin embargo, si las instancias de entrenamiento son **linealmente separables**, *Rosenblatt* demostró que este algoritmo convergería en una solución. Esto se denomina teorema de convergencia de perceptrón.



Perceptrón

Perceptrón: Debilidades

- *Marvin Minsky* y *Seymour Papert* 1969: "**Perceptrons**".
- Los perceptrones son incapaces de resolver algunos problemas triviales (por ejemplo, el problema de clasificación Exclusivo OR (XOR).



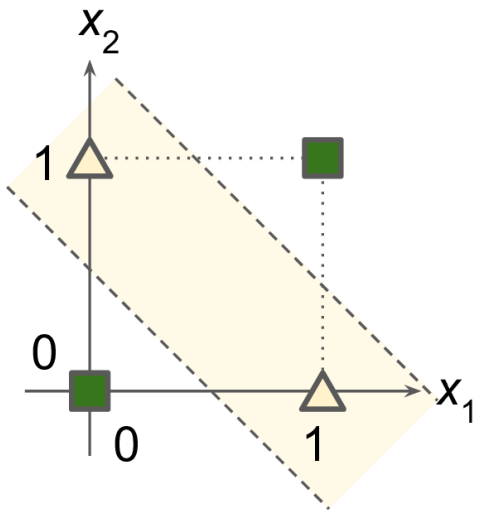
- Por supuesto, esto también es cierto para cualquier otro modelo de clasificación lineal **(como los clasificadores de regresión logística)**.

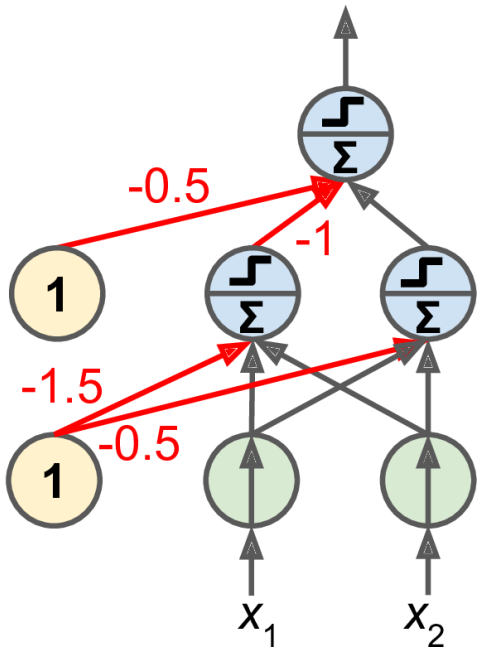
Muchos investigadores abandonaron las redes neuronales por completo.

Contenido

MLP

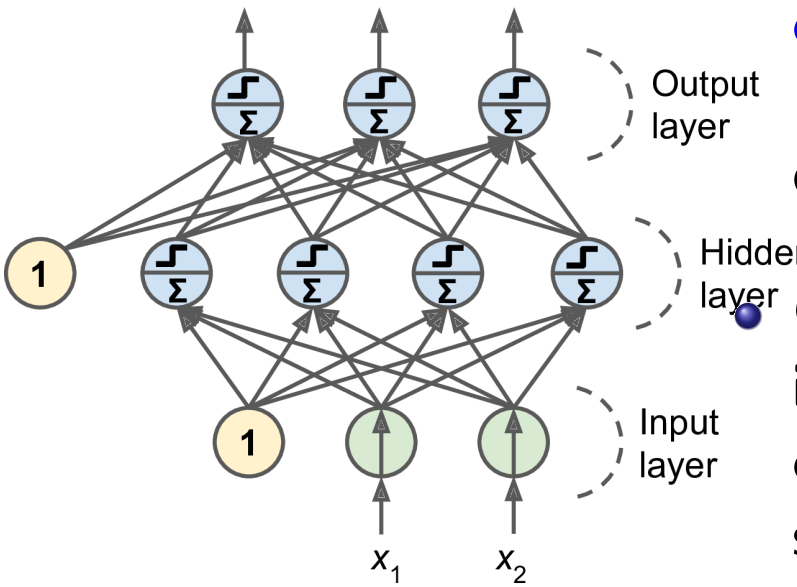
- Algunas de las limitaciones de los perceptrones se pueden eliminar apilando múltiples perceptrones.
- La ANN resultante se llama un **perceptrón multicapa (MLP)**.
- Un MLP **puede resolver el problema XOR**: con las entradas (0, 0) o (1, 1) las salidas de red 0, y con las entradas (0, 1) o (1, 0) emite 1.





Todas las conexiones tienen un peso igual a 1, excepto las cuatro conexiones donde se muestra el peso.

MLP and backpropagation I



- Un MLP se compone de una capa de entrada, una o más capas de TLU llamadas capas ocultas, y una capa final de TLU llamada capa de salida.
- Cada capa, excepto la de salida, **incluye una neurona bias** y está completamente conectada a la siguiente.

La señal fluye solo en una dirección, desde las entradas a las salidas: Feedforward Neural Network (FNN).

MLP and backpropagation II

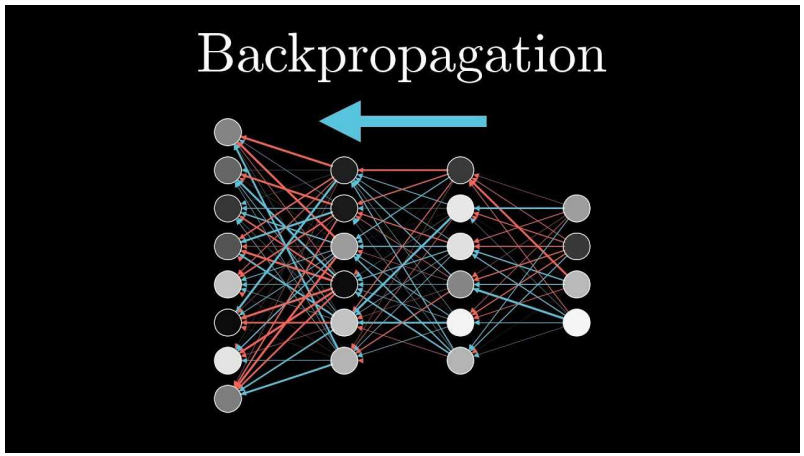
* Durante muchos años, los investigadores lucharon por **encontrar una manera de entrenar MLP**, sin éxito.

* En **1986**, *David Rumelhart*, *Geoffrey Hinton* y *Ronald Williams* publicaron un documento innovador que presenta el algoritmo de entrenamiento *backpropagation.*

* En resumen, es simplemente **Gradiente descendente** usando una técnica eficiente para calcular los gradientes automáticamente.

MLP and backpropagation III

En solo dos pasos a través de la red **(uno hacia adelante, uno hacia atrás)**, el algoritmo de *backpropagation* puede calcular el gradiente del error de la red con respecto a cada parámetro del modelo.



Es decir, **puede descubrir cómo se debe ajustar cada peso de conexión y cada término de sesgo para reducir el error**.

Una vez que tiene estos gradientes, solo realiza un paso GD regular, y todo el proceso se repite hasta que la red **converja en la solución**.

MLP and backpropagation IV

1. Toma un mini-lote a la vez, y pasa por todo el conjunto de entrenamiento varias veces. Cada pasada se llama **época**.
2. Cada mini-lote es enviado desde la **capa de entrada** a la primera **capa oculta**. Luego calcula la salida de todas las neuronas en esta capa. El resultado se pasa a la siguiente capa y así sucesivamente hasta la **capa de salida**. Este es el *forward pass*.
3. A continuación, el algoritmo mide el **error de salida** de la red.
4. Luego **calcula cuánto contribuyó cada conexión de salida al error** (regla de la cadena).
5. Luego, mide la cantidad de estas contribuciones de error, de una capa superior a una inferior, y así sucesivamente hasta la capa de entrada (*backward pass*).
6. Se realiza **un paso de GD para ajustar todos los pesos de conexión en la red**, utilizando los gradientes de error calculados.

En resumen...

- Para cada instancia de entrenamiento, el **algoritmo de retropropagación** primero realiza una predicción (*forward pass*), mide el error, luego pasa a través de cada capa en reversa para medir la contribución del error de cada conexión (*backward pass*), y finalmente modifica ligeramente los pesos de conexión para reducir el error (*GD pass*).

`https://thumbs.gfycat.com/FickleHorribleBlackfootedferret-small.gif`

MLP: Funciones de activación I

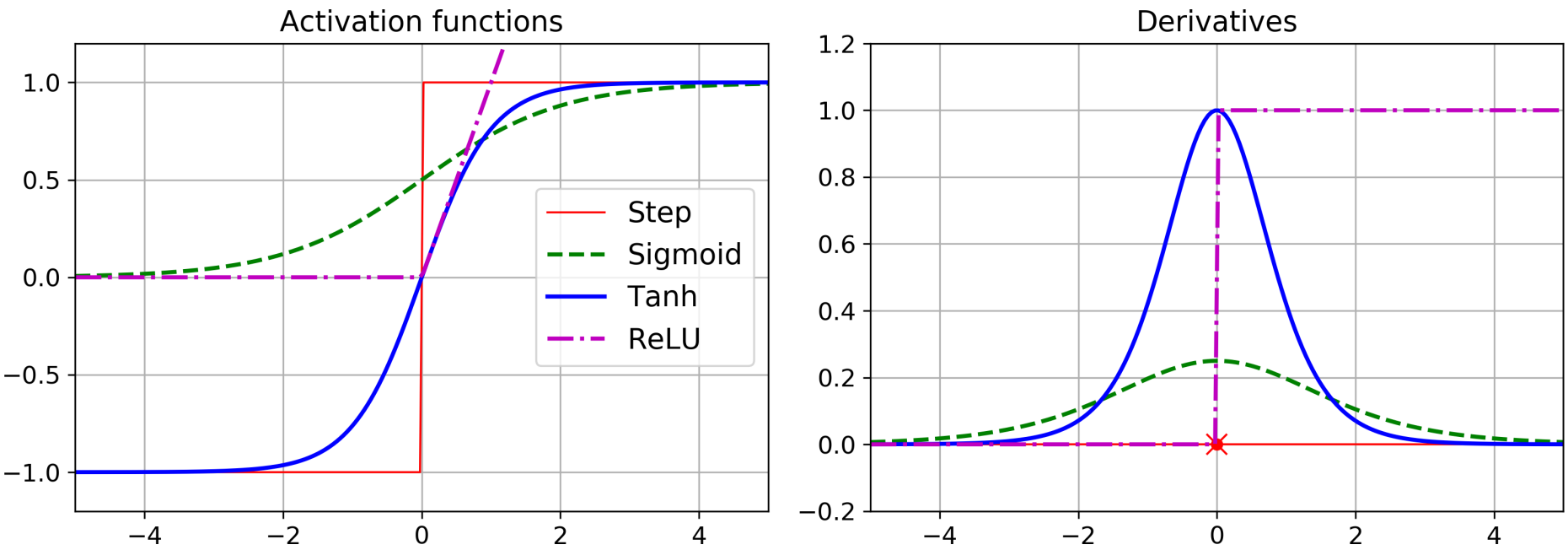
- Los autores realizaron un **cambio clave en la arquitectura de MLP**: reemplazaron la función de paso con la función logística, $\sigma(z) = 1/(1 + \exp(-z))$.

- La función de paso contiene solo segmentos planos, por lo que no hay gradiente con el que trabajar.

- La **función logística tiene una derivada no nula**, lo que permite que el GD progresar a cada paso.

- Otras dos funciones de activación populares son:
  - The **hyperbolic tangent function** $\tanh(z) = 2\sigma(2z)-1$
  - The **Rectified Linear Unit function**: $ReLU(z) = max(0, z)$

MLP: Funciones de activación II



- Si **encadenamos varias transformaciones lineales**, todo lo que obtiene es una **transformación lineal**.
- Entonces, si no tiene cierta **no-linealidad** entre las capas, incluso una pila profunda de capas es equivalente a una sola capa.
- No puede resolver **problemas muy complejos** con eso:

Funciones de activación

Contenido

MLP para regresión I

- Si desea predecir un valor único.
  - **Ejemplo:** el precio de una casa dada muchas de sus características, entonces solo necesita una neurona de salida única: su salida es el valor predicho.
- Regresión multivariada (es decir, para predecir múltiples valores a la vez), necesita una neurona de salida por dimensión de salida.
  - **Ejemplo:** para ubicar el centro de un objeto en una imagen, debe predecir las coordenadas 2D, por lo que necesita dos neuronas de salida. Si también desea colocar un cuadro delimitador alrededor del objeto, entonces necesita dos números más: el ancho y la altura del objeto. Entonces terminas con 4 neuronas de salida.

MLP para regresión II

- Al construir un MLP para la regresión, **no se utiliza ninguna función de activación para las neuronas de salida.**
- Sin embargo, si desea garantizar que la salida siempre será positiva, se puede usar la función de activación *ReLU* o la función de activación *softplus* en la capa de salida.
- Si desea garantizar que las predicciones se encuentren dentro de un rango de valores dado, puede usar la función $\sigma()$ o la tanh() y escalar las etiquetas al rango apropiado.

La *loss function* que se usa durante el entrenamiento suele ser el **error cuadrático medio**, pero si tiene muchos valores atípicos, puede preferir usar el **error absoluto medio**.

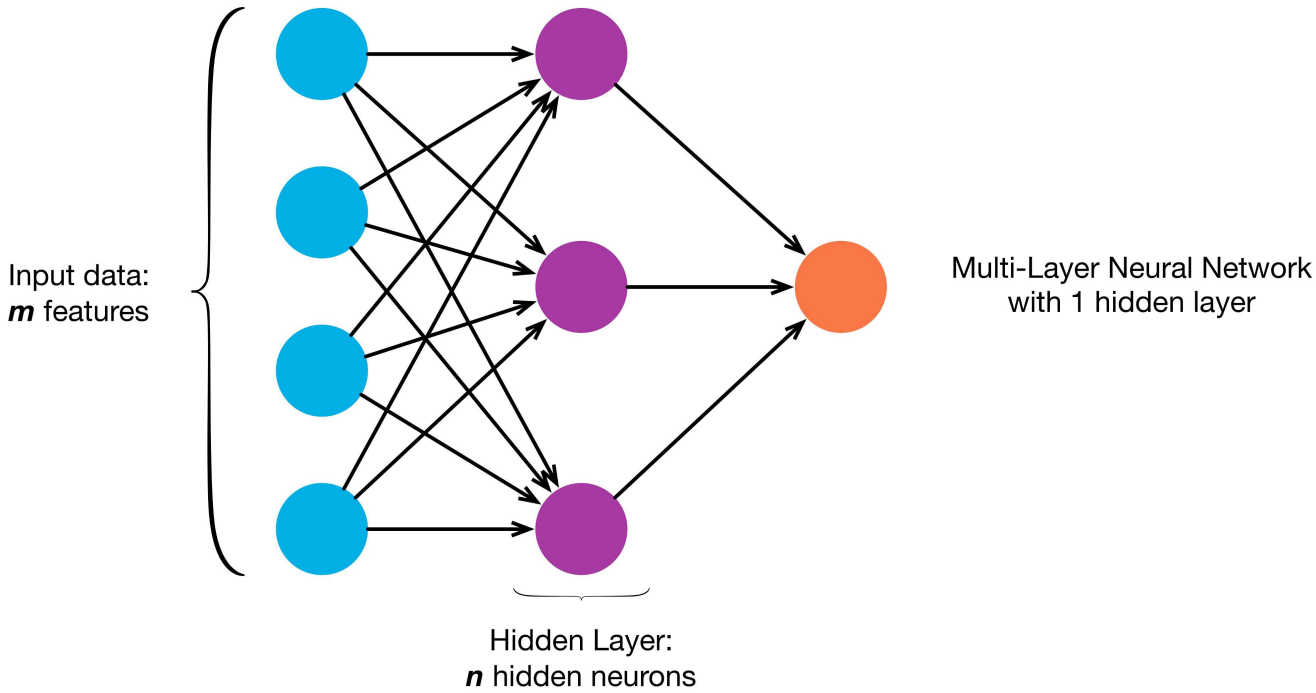
Arquitectura de MLP de regresión típica

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Tomado de Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, Second Edition

MLP para clasificación I

Para un problema de **clasificación binaria**, solo se necesita una neurona de salida única que utilice la función de activación logística: la salida será un número entre 0 y 1.

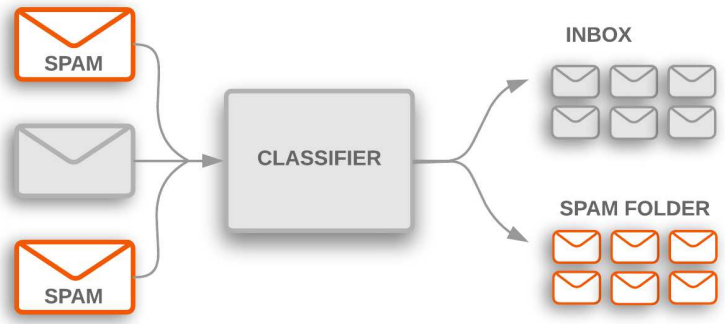


Esto se puede interpretar como la probabilidad estimada de la clase positiva.

MLP para clasificación II

- Los MLP también pueden manejar fácilmente tareas de **clasificación binaria** de múltiples etiquetas.

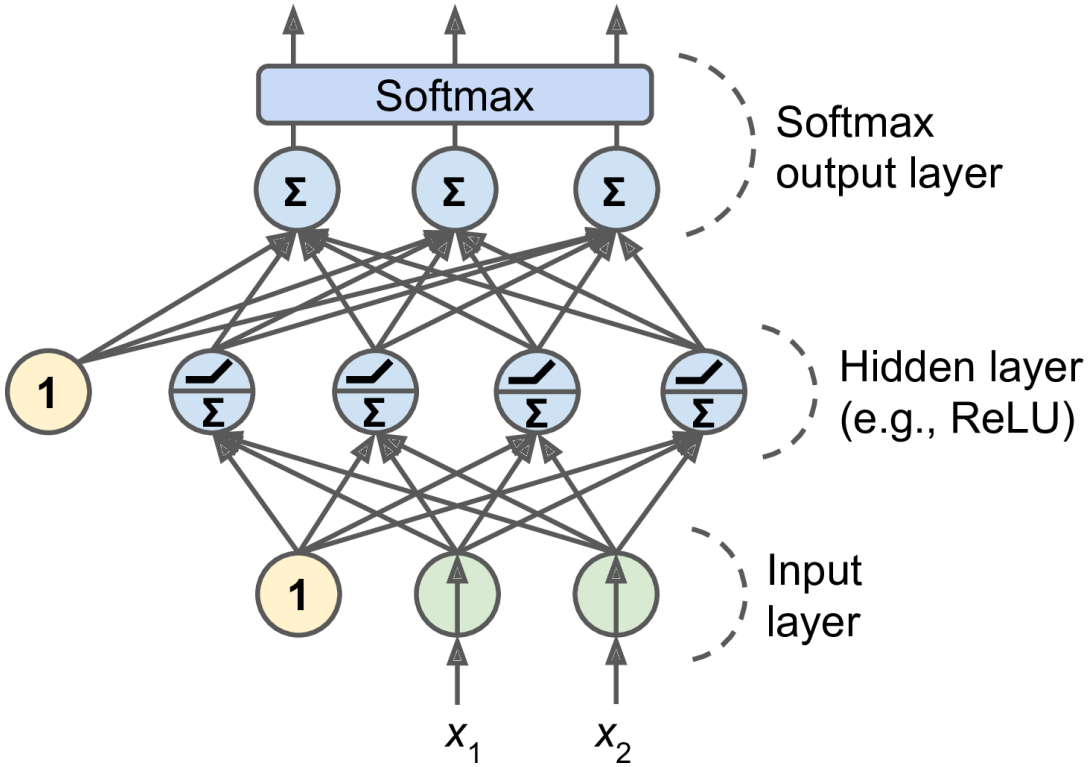
– **Ejemplo:** podría tener un sistema de clasificación de correo electrónico que prediga si cada correo electrónico entrante es no deseado o no deseado, y simultáneamente predice si es un correo electrónico urgente o no urgente.



En este caso, necesitaría dos neuronas de salida, ambas utilizando la función de activación logística.

MLP para clasificación III

Si cada instancia puede pertenecer solo a una sola clase, de 3 o más clases posibles, entonces debe tener una neurona de salida por clase, y debe usar la función de activación *softmax* para toda la capa de salida.



La función *softmax* asegurará que todas las probabilidades estimadas estén entre 0 y 1 y que sumen uno. Esto se llama **clasificación multi-clase**.

Arquitectura típica de MLP para clasificación

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

Tomado de Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, Second Edition

Contenido

Keras + TensorFlow

* Keras es una **API de aprendizaje profundo de alto nivel** que le permite construir, entrenar, evaluar y ejecutar fácilmente todo tipo de redes neuronales.

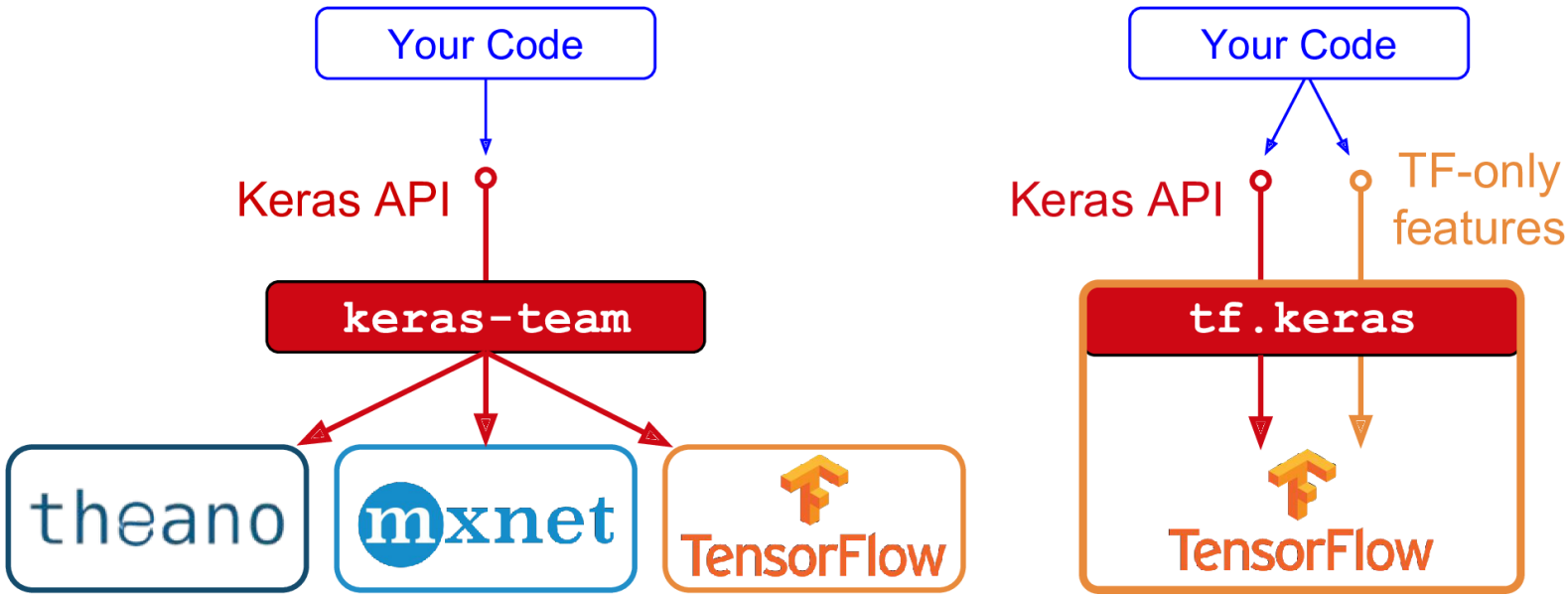
* Fue desarrollado por *François Chollet* como parte de un proyecto de investigación y lanzado como un proyecto de código abierto en **marzo de 2015**.

* Rápidamente ganó popularidad debido a su facilidad de uso, flexibilidad y hermoso diseño.

* En la actualidad, puede elegir entre tres bibliotecas de aprendizaje profundo de código abierto populares: TensorFlow, Microsoft Cognitive Toolkit (CNTK) o Theano.

Keras + TensorFlow

- Desde finales de 2016, se han lanzado **otras implementaciones**.
- TensorFlow ahora viene incluido con su propia implementación de Keras llamada `tf.keras`. Solo es compatible con TensorFlow como *back-end*, pero tiene la ventaja de ofrecer algunas características adicionales muy útiles:
  - Es compatible con la API de datos de TensorFlow que hace que sea bastante fácil **cargar y pre-procesar datos de manera eficiente**.



Instalando TensorFlow==2.0

1. Abrir Anaconda Prompt (o terminal de Ubuntu si utiliza Linux)
2. `pip install tensorflow==2.0`
3. `python`
4. `>> import tensorflow as tf`
5. `>> tf.__version__`
6. `>> from tensorflow import keras`
7. `>> keras.__version__`

Creación de un clasificador de imágenes con la API secuencial

- Abordaremos **Fashion MNIST**, que es un reemplazo directo de MNIST.
- Tiene exactamente el mismo formato que MNIST (70,000 imágenes en escala de grises de 28×28 píxeles cada una, con 10 clases).
- Las imágenes representan artículos de moda en lugar de dígitos escritos a mano, por lo que cada clase es más diversa y el problema resulta ser significativamente más desafiante que MNIST.
- **Ejemplo:** un modelo lineal simple alcanza un 92% **de precisión** en MNIST, pero solo un 83% en Fashion MNIST.

MLP con API secuencial

Creación de un MLP de regresión utilizando la API secuencial

- Pasemos al problema de *California housing* y abordemos el problema utilizando una **red neuronal de regresión**.
- Para simplificar, usaremos la función `fetch_california_housing()` de Scikit-Learn para cargar los datos.
- Este conjunto de datos es más simple que el que usamos anteriormente, ya que contiene solo características numéricas, y no hay *missing data*.

MLP con API secuencial