Introducción a las redes neuronales

Francisco J. Ribadas-Pena

28 de abril de 2021

Grupo CoLe – 2021

Contenidos

Fundamentos redes neuronales

¿Qué son las Redes Neuronales?

Redes Neuronales: Campo de la IA que pretende construir sistemas inteligentes que emulen el funcionamiento a bajo nivel de los cerebros biológicos

- Conjunto de neuronas artificiales interconectadas
- Neuronal artificiales ”replican” de forma simplificada el funcionamiento de las neuronas biológicas
- Muchas neuronas simples (normalmente) altamente conectadas
- Procesamiento altamente distribuido y paralelo
- Capacidad de aprendizaje $\rightarrow$ ajuste de los pesos sinápticos entre neuronas
- Gran variedad de modelos de redes, neuronas, interconexión y métodos de aprendizaje

En realidad, sólo son ”algoritmos” que multiplican matrices y vectores” :-)

Evolución de las RN

Orígenes

- *McCulloch, Pitts*(1943): Primer modelo de neurona artificial
- *Hebb*(1949): Aprendizaje neuronal (regla de Hebb)
  "Una sinapsis aumenta su eficacia (peso) si las dos neuronas que conecta tienden a estar activas o inactivas simultánemente. Ocurriendo, en el caso contrario, una atenuación de ese peso sináptico"
- *Rossemblatt*(1958): Desarrollo *perceptron* (red simple, 1 capa)
- *Widrow, Hoff*(1960): Desarrollo *adaline*
  - Primera aplicación industrial real (cancelación ecos linea telefónica)

Declive Finales 60's-80's

- *Minsky, Papert*: Estudio sobre limitaciones del perceptron.
- Reducción investigación, Falta de modelos de aprendizaje.

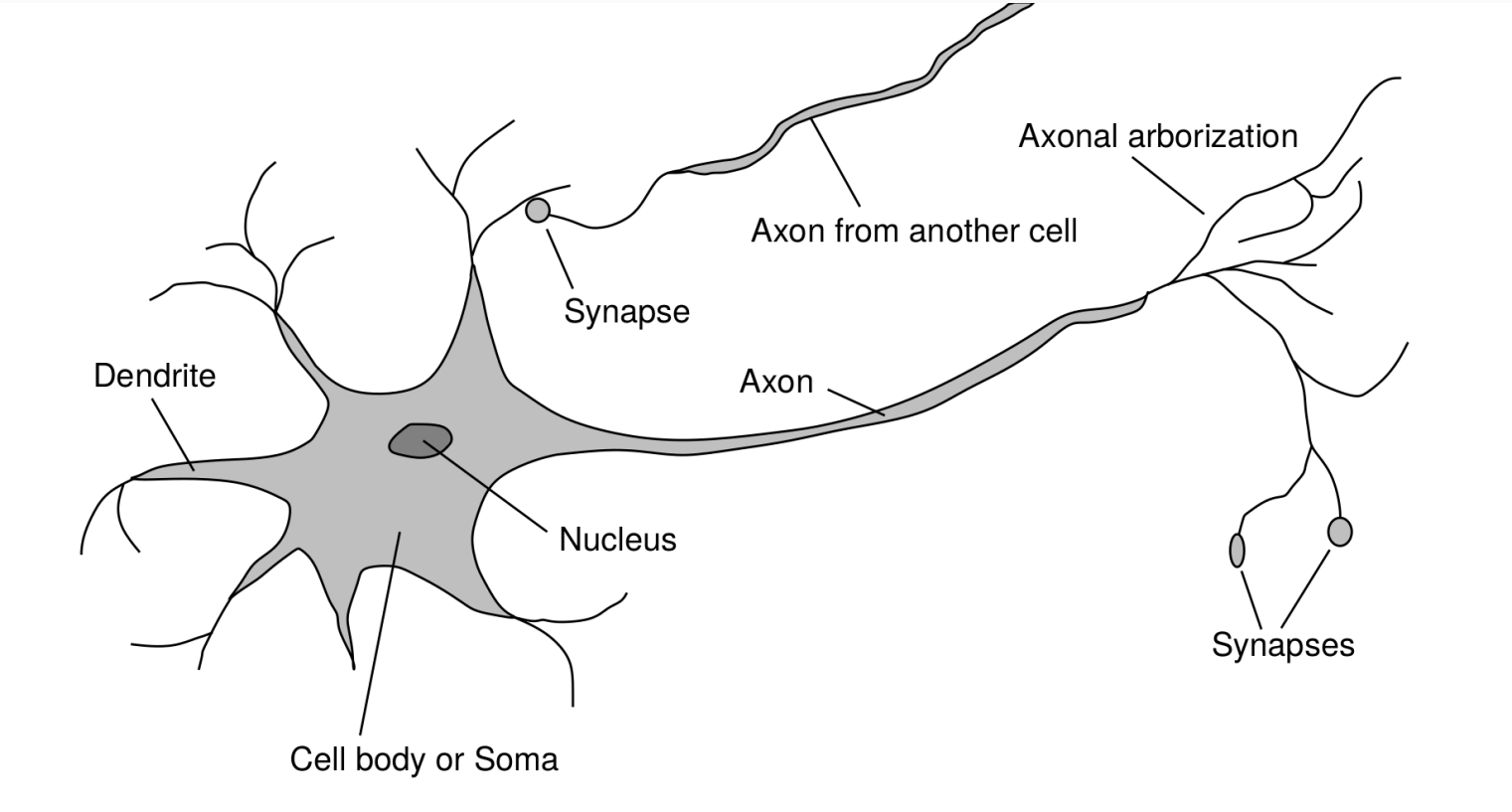
Resurgir, 80's-90's

- Nuevos modelos: Red de *Hopfield* (1982), Mapas autoorganizativos de Kohonen(1982)
- *Rumelhart, McLellan*(1986): Alg. aprendizaje retropropagación
  - Aplicable a perceptrones complejos (multicapa)
- *LeCunn*(1995): Redes convolucionales (CNN)
- *Hochreiter*(1997): Long short-term memory (LSTM)

Auge Deep Learning, $>$ 2010

Fundamentos de las RN (I)

Neurona biológica

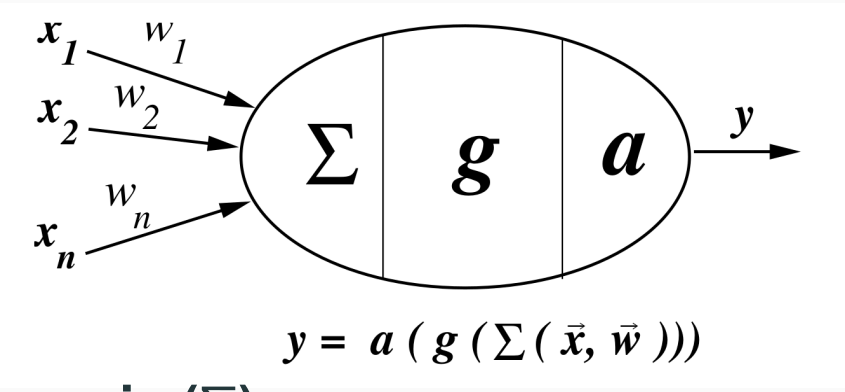


$\approx 86 \cdot 10^9$ neuronas

$\approx 1{,}5 \cdot 10^{14}$ conexiones (sinapsis)

Fundamentos de las RN (II)

Modelo de neurona artificial



Función de transferencia (Σ)

- Combina entradas y pesos sinápticos
- Suele ser la suma ponderada ($\equiv$ producto escalar de vectores $\vec{x} \cdot \vec{w}$)
- En CNN (*convolutional neural nets*) se emplea operador de convolución

Función de activación (g)

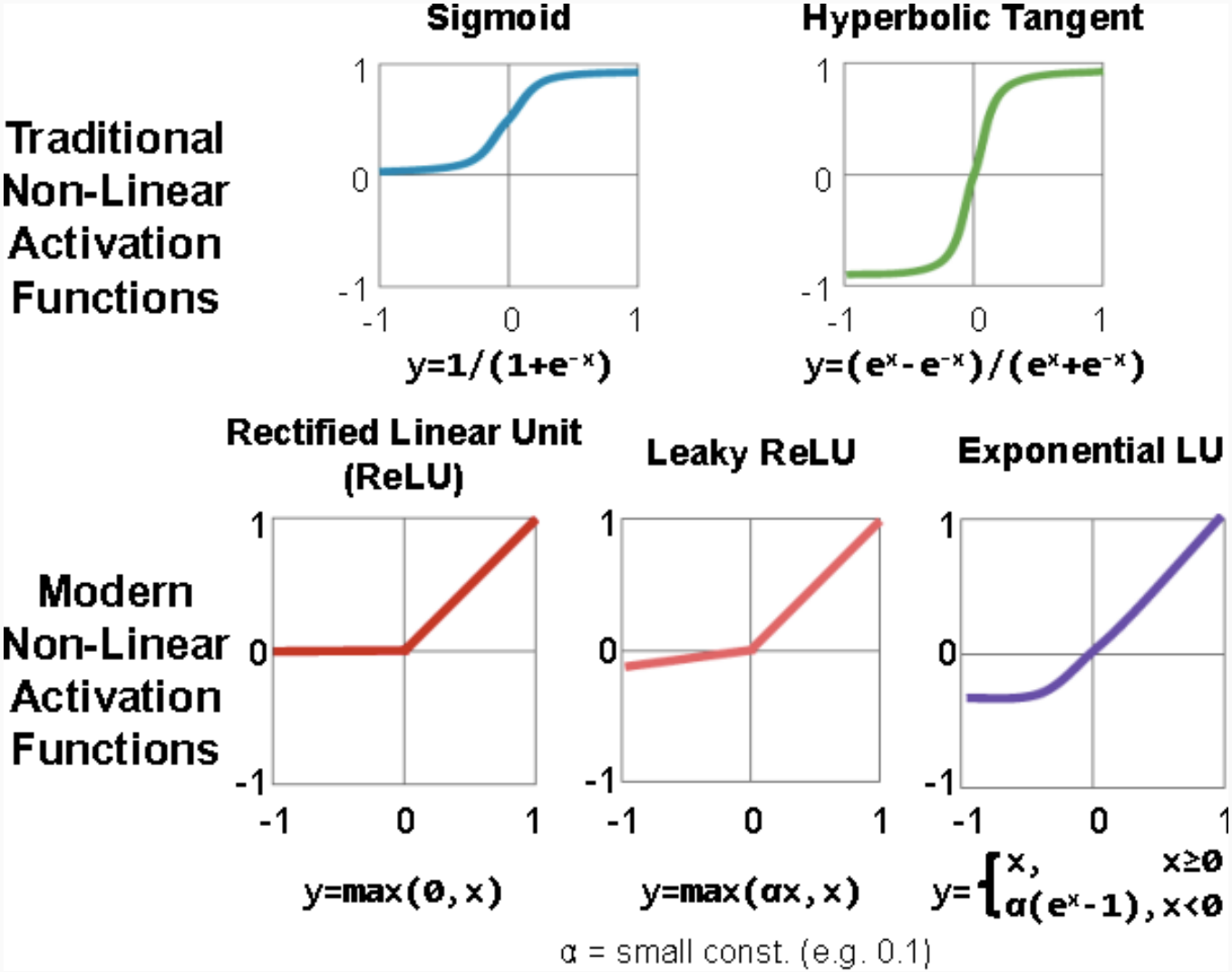
- Determina el estado de activación de la neurona
- Diversas funciones posibles con propiedades específicas (influye en la convergencia del proceso de aprendizaje)

Función de salida (a)

- Suele ser la identidad

Fundamentos de las RN (III)

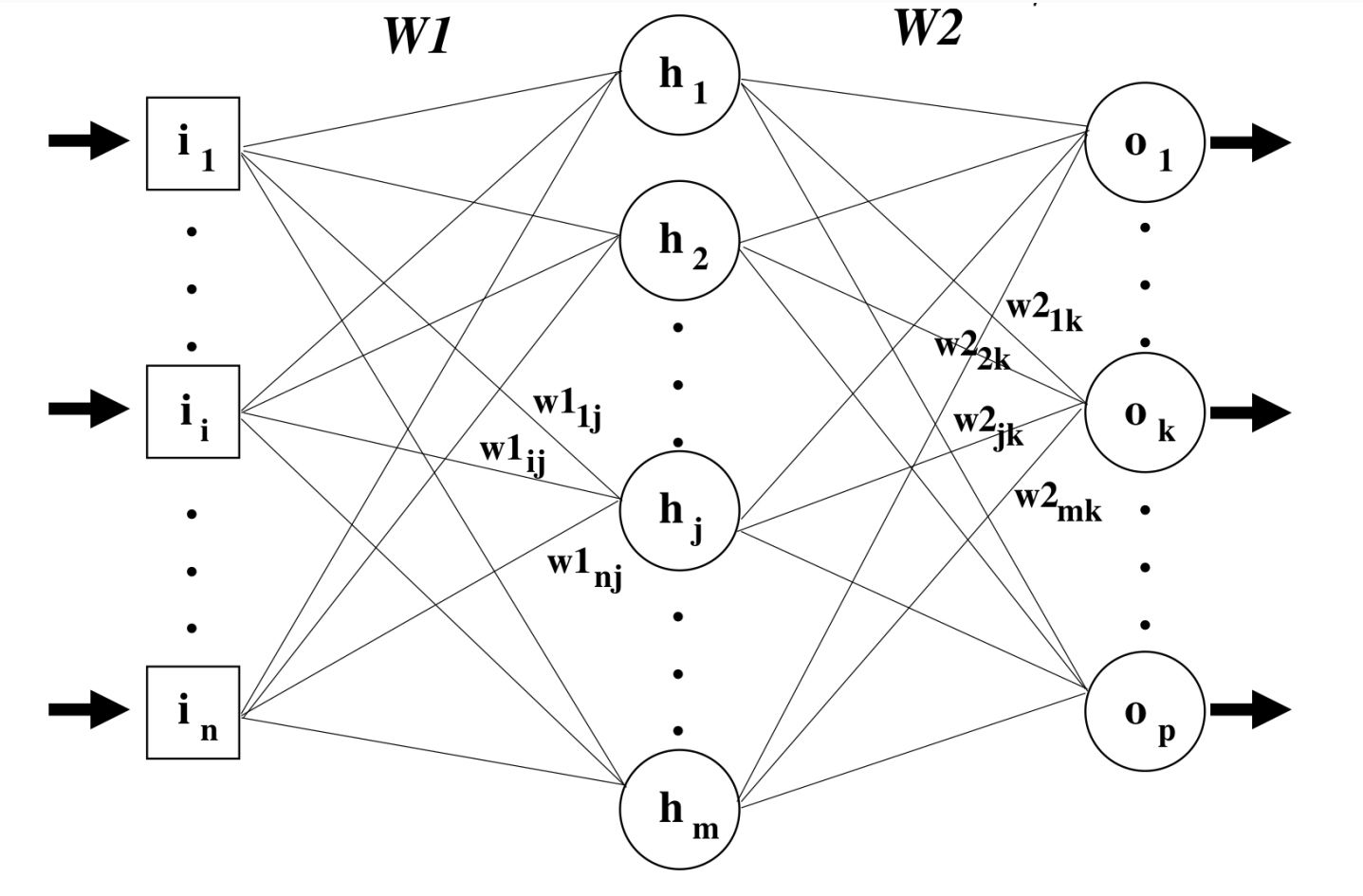
Funciones de activación (”clásicas” *vs.* ”modernas”)



Fuente: `http://eyeriss.mit.edu/tutorial.html`

Fundamentos de las RN (IV)

Ejemplo: red neuronal totalmente conectada con propagación hacia adelante



Perceptron multicapa (MLP) con 1 capa oculta

Aprendizaje en redes neuronales

Aprendizaje en RN (I)

Idea base: ajuste de pesos sinápticos a partir del comportamiento de la red ante los ejemplos de entrenamiento.

- **Aprend. no supervisado:** sólo dispone de los vectores de entrada
- **Aprend. supervisado:** se dispone de los vectores de entrada y sus correspondientes vectores de salida

Algoritmo de retropropagación (*Backpropagation*)

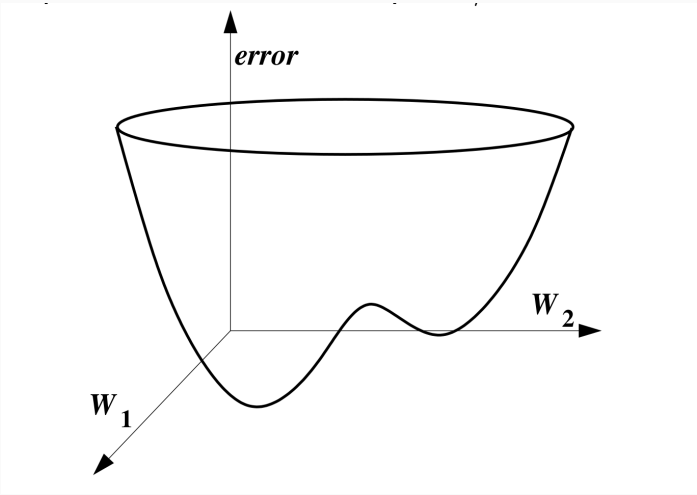
- Método supervisado (usando en MLP y redes similares)
- Propaga vector de entrada sobre la red para obtener un vector de salida
- Compara salida obtenida con salida deseada y propaga el error obtenido hacia atrás
  - ”Reparte” el error *real* de cada neurona de salida entre sus pesos d forma proporcional a la intensidad de cada conexión
  - En neuronas ocultas el error se *estima* ponderando su contribución en los errores de la capa siguiente

Aprendizaje en RN (II)

Algoritmo de retropropagación (*Backpropagation*) (cont.)

- Pertenece a la familia de algoritmos de optimización de funciones basados en descenso de gradiente (SGD: *stochastic gradient descent*, etc)

- Ajuste iterativo de los parámetros de la función objetivo (en RNAs pesos sinápticos) para minimizar el valor la *loss function* (función de pérdida) que cuantifica la ”bondad del modelo”
  - Distintas *loss funtion* posibles: error cuadrático medio, *cross entropy loss*, etc
  - Suelen interpretarse como ”medidas del error” del modelo a optimizar
- Ajuste de parámetros del modelo guiado por zonas de máxima pendiente de la *loss funtion* a minimizar.



Aprendizaje en RN (III)

Método de retropropagación del error (*backpropagation*) para MLP

Exige función de activación (g) continua (y derivable) $\Rightarrow$ tradicionalmente sigmoidal.

$$g(x) = \frac{1}{e^{-x}} \qquad g'(x) = g(x)(1 - g(x))$$

Ajuste pesos en CAPA SALIDA $\boxed{W2_{jk} = W2_{jk} + \alpha\, h_j\, \Delta_k}$

- Usa:
  - h_j : activación neurona oculta h_j
  - $(T_k - o_k)$: error (salida deseada - salida obtenida)
  - $g'(ent_k)$: derivada f. activación ($g'(ent_k) = o_k(1 - o_k)$)
  - α : tasa de aprendizaje

$$\boxed{\Delta_k = g'(ent_k)\,(T_k - o_k) = o_k\,(1 - o_k)\,(T_k - o_k)}$$

Ajuste pesos CAPA/S OCULTA/S $\boxed{W1_{ij} = W1_{ij} + \alpha\, i_i\, \Delta_j}$

- Problema: Cuantificar error en las capas ocultas (Δ_j).
- Idea: Propagar la parte proporcional del error en la capa de salida del cual es "responsable" cada neurona oculta h_j.
- **Error en neurona oculta h_j:**

$$\boxed{\Delta_j = g'(ent_j)\ \sum_{k=1}^{p}(W2_{jk}\Delta_k) = h_j\,(1 - h_j)\ \sum_{k=1}^{p}(W2_{jk}\Delta_k)}$$

Tipos de redes neuronales

Tipos de RNA

Diversos criterios: flujo de información, modo de entrenamiento, ”memoria”, etc

- Posible parametrización: tipo y n^{o} de capas, tipo de neuronas, funciones activación, etc

Algunas redes de uso habitual

Redes totalmente conectadas flujo ”hacia adelante” con conexión total entre capas (ej. MLP)

$\rightarrow$ uso habitual en tareas de clasificación y predicción numérica

Redes convolucionales (CNN) flujo ”hacia adelante” con capas de convolución con conexiones parciales y pesos compartidos

$\rightarrow$ uso habitual en procesamiento de imágenes (reconocim. de formas,etc)

Redes recurrentes (RNN) posibilidad de conexiones ”hacia atrás” y entre capas no adyacentes

$\rightarrow$ uso habitual en procesamiento de secuencias y series temporales

Long Short-Term Memory (LSTM) redes recurrentes que emplean neuronas ”con memoria”

$\rightarrow$ uso habitual en procesamiento de secuencias y procesamiento del lenguaje

Más en *The Neural Network Zoo* (`http://www.asimovinstitute.org/neural-network-zoo/`)

Deep Learning

¿Qué es Deep Learning?

Deep Learning (DL):(tb. Aprendizaje Profundo)

> *métodos de aprendizaje automático caracterizados por emplear modelos complejos con múltiples "capas", aportando cada capa un grado de abstracción creciente en las representaciones generadas*

Habitualmente: distintos tipos redes neuronales complejas (muchas capas de gran tamaño)

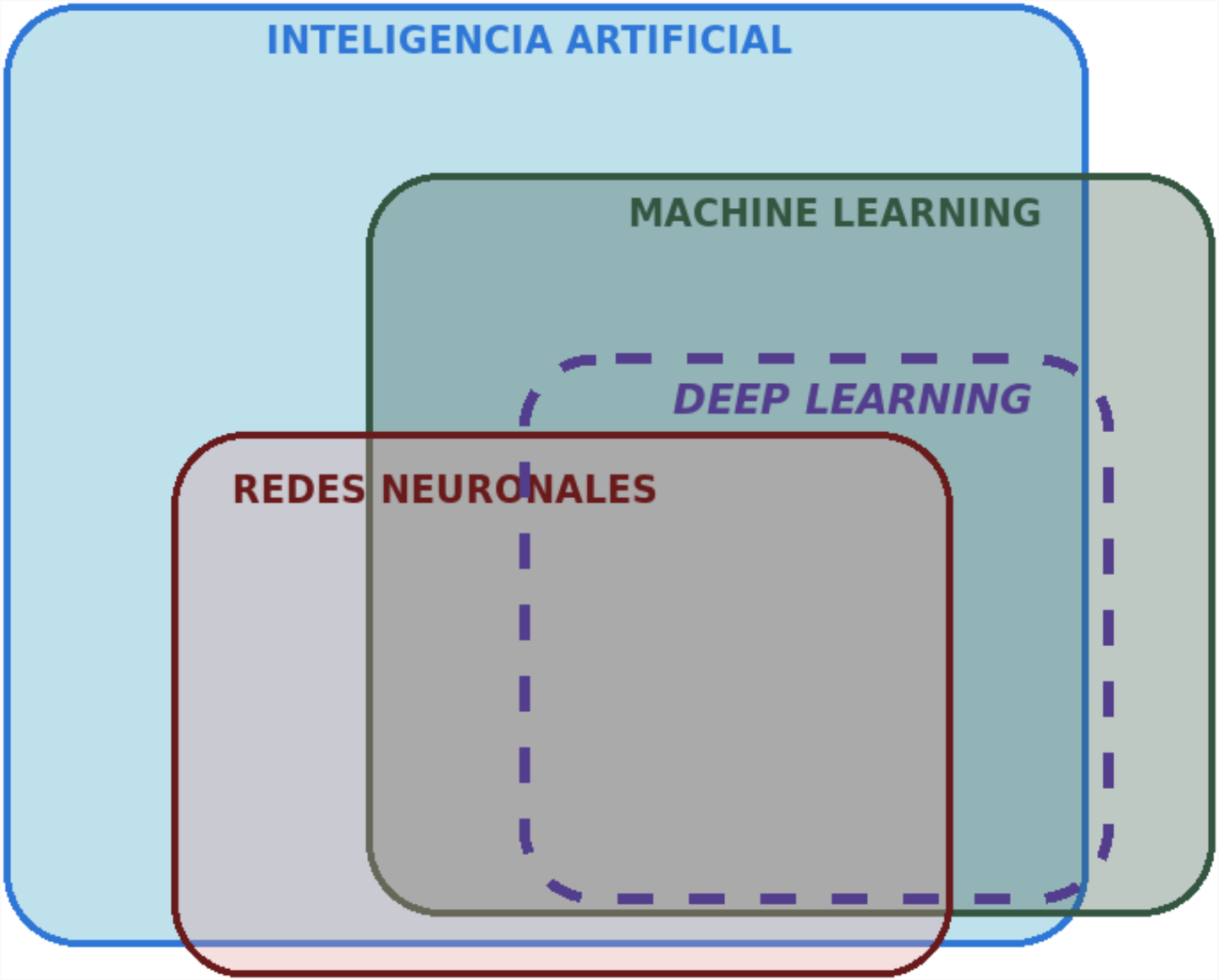
- *Deep Believe Networks* (redes bayesianas "profundas") con características y filosofías de funcionamiento análogas.
- *Word embebedings* usados en procesameinto del lenguaje suelen incluirse en las técnicas DL

Red compleja (muchas capas grandes) $\Rightarrow$ muchos parámetros a ajustar $\Rightarrow$ requiere gran cantidad de datos de entrenamiento

Pros: Gran capacidad para representación de conceptos complejos

Contra: Alto coste de entrenamiento (mitigado con GPUs y TPUs)

Resumen


INTELIGENCIA ARTIFICIAL
MACHINE LEARNING
DEEP LEARNING
REDES NEURONALES

Deep Neural Nets

Explosión de las *Deep Neural Nets* al unirse:

1. disponibilidad de grandes cantidades de datos de entrenamiento (fuentes: redes sociales, big data empresarial, IOT, ...)
2. aumento de la capacidad de procesamiento (procesamiento vectorial paralelo sobre GPUs y, recientemente, TPUs)
3. nuevas propuestas de modelos de redes neuronales (o mejoras sobre los ya existentes)
    - modelos clásicos a muy gran escala: *Deep Convolutional Neural Nets (DCNN)*, *Deep Autoencoders*, etc
    - técnicas ”nuevas”: *Generative Adversarial Networks (GANs)*, *Word Embeddings* (Word2Vec, GloVe), etc

Adicionalmente,

- interés de la industria: coche autónomo, traducción automática, seguridad y vigilancia, etc
- disponibilidad de frameworks y librerías *open source* (TensorFlow, Theano, Torch, CNTK, Cafee, ...)

Diferencias DL vs. ML "clásico"

Punto clave: *feature engineering* implícito en DL

- Algoritmos ML "clásicos" suelen requerir fase previa de *Feature Engineering* (extracción de características)
  - Uso de conocimiento experto (específico de cada dominio) para identificar, definir y extraer de los datos en bruto los atributos/características (*features*) que se usarán para describir los ejemplos de entrenamiento
  - Tarea costosa, condiciona mucho la efectividad final de los métodos de aprendizaje
- Mayor parte de modelos DL suelen trabajar con datos en bruto Propio modelo "aprende" a realizar la extracción de características en capas iniciales de la red.
  - Primeras capas realizan extracción y transformación de caracterísitcas de forma implícita, que va irán siendo refinadas en las sucesivas capas
  - También, preprocesamiento de datos en bruto con métodos no supervisados (autoencoders, word embedding)
    - aprenden a identificar regularidades y crean modelos capaces de capturar/codificar características relevantesde los datos en bruto

Frameworks Deep Learning

Frameworks/Librerías DL (I)

TensorFlow (`https://www.tensorflow.org/`)

- Desarrollado por Google Brain
- Núcleo escrito en C++ con interfaces en Python y otros lenguajes
- Incluye soporte para múltiples GPUs y TPUS (*Tensor Processing Unit*)
- Es un motor para cálculo de tensores ($\approx$ arrays multidimensionales) que emplea ”grafos de computación” (*data flow graphs*) para describir las operaciones a realizar (bien sobre CPUs, GPUs o TPUs)
- Sirve de base a herramientas/interfaces de más alto nivel (Keras, TF-Learn, TF-Slim, Sonnet) que simplifican la programación

Theano

(`http://www.deeplearning.net/software/theano/`)

- Desarrollado por Montréal University (actualmente se ha detenido su desarrollo)
- Misma aproximación que TensorFlow: definición de grafos de operaciones sobre tensores

Frameworks/Librerías DL (II)

Torch/Pytorch (`http://torch.ch/`, `http://pytorch.org/`)

- Evolución de la librería de *Machine Learning* Torch
- Núcleo escrito en C/C++ con capa externa en Lua y Python (PyTorch)

Caffe / Caffe2 (`http://caffe.berkeleyvision.org/` `https://caffe2.ai/`)

- *Convolutional Architecture for Fast Feature Embedding*, desarrollado originalmente por Berkeley Vision and Learning Center, la nueva versión (Caffe2) respaldada por Facebook
- Escrito en C++ con interface en Python. Soporta GPUs.

CNTK
(`https://www.microsoft.com/en-us/cognitive-toolkit/`, `https://github.com/Microsoft/CNTK`)

- Microsoft Cognitive Toolkit
- Escrito en C++ con APIs para otros lenguajes, incluye soporte para GPUs

DL4J (`http://deeplearning4j.org/`)

- Framework de DL disponible para la JVM (accesible desde Java, Scala, etc)
- Partes del núcleo implementadas en C/C++. Soporte para GPUs

Keras (I)

Librería Python de alto nivel y modular (`https://keras.io/`)

- No implementa los algoritmos sino que delega su ejecución a motores como TensorFlow, Theano o CNTK
- **Objetivo:** simplificar desarrollo/experimentación con modelos DL

Definición de modelos (redes) (2 modos de especificación)

- *Sequential model*: define una secuencia ordenada de capas (*layers*) [sólo redes con flujo hacia adelante]
- *Functional API*: mayor flexibilidad en la topología de la red
- Métodos comunes:
  - `compile(...)`: configura el proceso de entrenamiento a usar (función *loss*, algoritmo de aprendizaje [*optimizer*], métricas de calidad a usar)
  - `fit(...)`: lleva a cabo el entrenamiento con los datos de entrenamiento y parámetros aportados (*batch size*, *epochs*, ...)
  - `evaluate()`, `predict()`, ...
  - `save_weights()`, `load_weights()`, ...

Keras (II)

Elementos de los modelos

- Capas (*Layers*): `Dense`, `Conv2D`, `Conv3D`, `Activation`, `Flatten`, `MaxPooling`, ...
- Funciones de activación: `softmax`, `relu`, `elu`, `sigmoid`, ...
- Funciones *loss*: `mean_squared_error`, `mean_absolute_error`, `hinge`, `categorical_crossentropy`, ...
- Algoritmos de optimización: `SGD`, `RMSprop`, `Adam`, ...

Otras funcionalidades

- Funciones para carga y preprocesamiento de imágenes, texto y secuencias
- Utilidades de acceso a datasets públicos (CIFAR10/100, NIST, IMDB, ...) y carga y serialización de modelos
- Visualización de modelos y del proceso de entrenamiento

Keras (Ejemplo 1)

MLP de 2 capas para el dataset MNIST

```
...
model = Sequential()

model.add(Flatten(input_shape=(28, 28, 1)))
model.add(Dense(128))
model.add(Activation('sigmoid'))
model.add(Dense(64))
model.add(Activation('sigmoid'))
model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(optimizer='rmsprop',
        loss='categorical_crossentropy',
        metrics=['accuracy'])

model.fit(X_train, Y_train,
        epochs=10,
        validation_data=(X_test, Y_test))
...
```

Fuente: https://github.com/stared/keras-mini-examples

Clasificación de dígitos

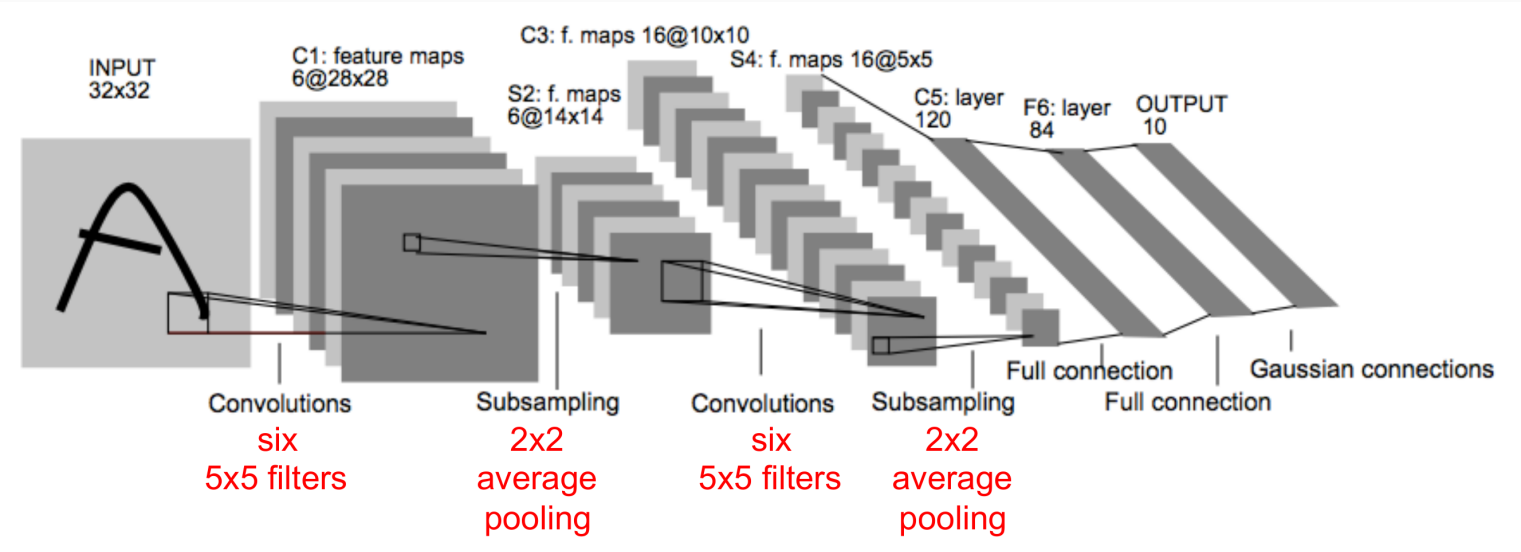
- Imágenes 28x28 B/N
- 10 clases
- 60.000 train + 10.000 test

Keras (Ejemplo 2)

CNN para el dataset MNIST: ”tipo LeNet-5, 1998”

```
model = Sequential()

# CONV => RELU => POOL
model.add(Conv2D(6, (5, 5), border_mode="same",
                         input_shape=(28, 28, 1)))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))

# CONV => RELU => POOL
model.add(Conv2D(16, (5, 5), border_mode="same"))
model.add(Activation("relu"))
model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
...
```

```
# Full Connected => RELU
model.add(Flatten())
model.add(Dense(120))
model.add(Activation("relu"))
model.add(Dense(84))
model.add(Activation("relu"))

# softmax
model.add(Dense(10))
model.add(Activation("softmax"))

model.compile(optimizer=SGD(lr=0.01),
   loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, Y_train,epochs=10,
   validation_data=(X_test, Y_test))
```



Fuente: https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/