



ARISA Learning Material

Educational Profile and EQF level: DATA SCIENTIST – EQF 6

PLO: 1, 2, 3, 4, 5

Learning Unit (LU): MACHINE LEARNING: SUPERVISED

Topic: LINEAR MODELS AND LOGISTIC REGRESSION



www.aiskills.eu

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Copyright © 2024 by the Artificial Intelligence Skills Alliance

All learning materials (including Intellectual Property Rights) generated in the framework of the ARISA project are made freely available to the public under an open license [Creative Commons Attribution–NonCommercial](#) (CC BY-NC 4.0).

ARISA Learning Material 2024

This material is a draft version and is subject to change after review coordinated by the European Education and Culture Executive Agency (EACEA).

Authors: Universidad Internacional de La Rioja (UNIR)

Disclaimer: This learning material has been developed under the Erasmus+ project ARISA (Artificial Intelligence Skills Alliance) which aims to skill, upskill, and reskill individuals into high-demand software roles across the EU.



**Co-funded by
the European Union**

This project has been funded with support from the European Commission. The material reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- About ARISA

- The Artificial Intelligence Skills Alliance (ARISA) is a four-year transnational project funded under the EU's Erasmus+ programme. It delivers a strategic approach to sectoral cooperation on the development of Artificial Intelligence (AI) skills in Europe.
- ARISA fast-tracks the upskilling and reskilling of employees, job seekers, business leaders, and policymakers into AI-related professions to open Europe to new business opportunities.
- ARISA regroups leading ICT representative bodies, education and training providers, qualification regulatory bodies, and a broad selection of stakeholders and social partners across the industry.

[ARISA Partners & Associated Partners](#) | [LinkedIn](#) | [Twitter](#)

- CODE
- EN AMAZON CLOUD ARISA 1 – ANN
- EN LOCAL TUTORIAL_DEEP_LEARNING_basics

[ARISA Partners & Associated Partners](#) | [LinkedIn](#) | [Twitter](#)



ARISA Learning Material

Educational Profile and EQF level: DATA SCIENTIST – EQF 6

PLO: 1, 2, 3, 4, 5

Learning Unit (LU): MACHINE LEARNING: SUPERVISED

Topic: LINEAR MODELS AND LOGISTIC REGRESSION



www.aiskills.eu

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

Parámetros (continuación NN)

- **hidden_layer_sizes:** sets the number of hidden layers (number of elements in list), and number of hidden units per layer (each list element). *Default: (100).*
- **alpha:** controls weight on the regularization penalty that shrinks weights to zero. *Default: $\alpha = 0.0001$.*
- **activation:** controls the nonlinear function used for the activation function, including: 'relu' (default), 'logistic', 'tanh'.

Parámetros (continuación NN)

- **Alpha:**El parámetro alpha representa la magnitud del término de regularización L2, que penaliza el tamaño de los pesos para evitar el sobreajuste.
- `alpha=0` implica sin regularización (no recomendable, riesgo alto de overfitting)
- `alpha` bajo (por ejemplo, `0.0001`): Regularización ligera, es el valor por defecto en `scikit-learn`.
- `alpha` alto (por ejemplo, `0.1`, `1` o más): Fuerte regularización, se penalizan más los pesos grandes, reduciendo complejidad del modelo.

Backpropagation

- Es un método para entrenar redes neuronales artificiales.
- Permite ajustar los pesos para que la red mejore su rendimiento.
- Calcula el gradiente del error de salida respecto a cada peso.
- Usa el método de descenso del gradiente para minimizar el error.

¿Cómo Funciona?

1- Propagación hacia adelante:

- Los datos se introducen y se calcula una salida.

2- Cálculo del error:

- Se compara la salida con la respuesta correcta.

3- Propagación hacia atrás:

- Se calcula cómo el error se distribuye hacia los pesos.

4- Actualización de pesos:

- Se ajustan los pesos para reducir el error en la próxima iteración.

1. Inicializa pesos aleatorios.
2. Para cada muestra:
 - Hacer una pasada hacia adelante.
 - Calcular error (Ej: error cuadrático medio).
 - Retropropagar el error desde la salida hacia la entrada.
 - Actualizar pesos con:

$$w = w - \eta \cdot \frac{\partial E}{\partial w}$$

Donde:

- w : peso
- η : tasa de aprendizaje
- E : error

Algoritmo backpropagation I

⌘ Descripción:

Adelante

- Tras inicializar los pesos de forma aleatoria y con valores pequeños, seleccionamos el primer par de entrenamiento.
- Calculamos la salida de la red
- Calculamos la diferencia entre la salida real de la red y la salida deseada, con lo que obtenemos el vector de error

Atrás

- Ajustamos los pesos de la red de forma que se minimice el error
- Repetimos los tres pasos anteriores para cada par de entrenamiento hasta que el error para todos los conjuntos de entrenamiento sea aceptable.

⌘ Descenso por la superficie del error



⌘ Cálculo de derivadas del error respecto de los pesos y de las bias.

Algoritmo backpropagation II



⌘ Detalles:

- SSE: $E = \sum E_p = \sum (y_{pk} - o_{pk})^2$
- $\Delta w_{ij} = -\eta \partial E / \partial w_{ij}$

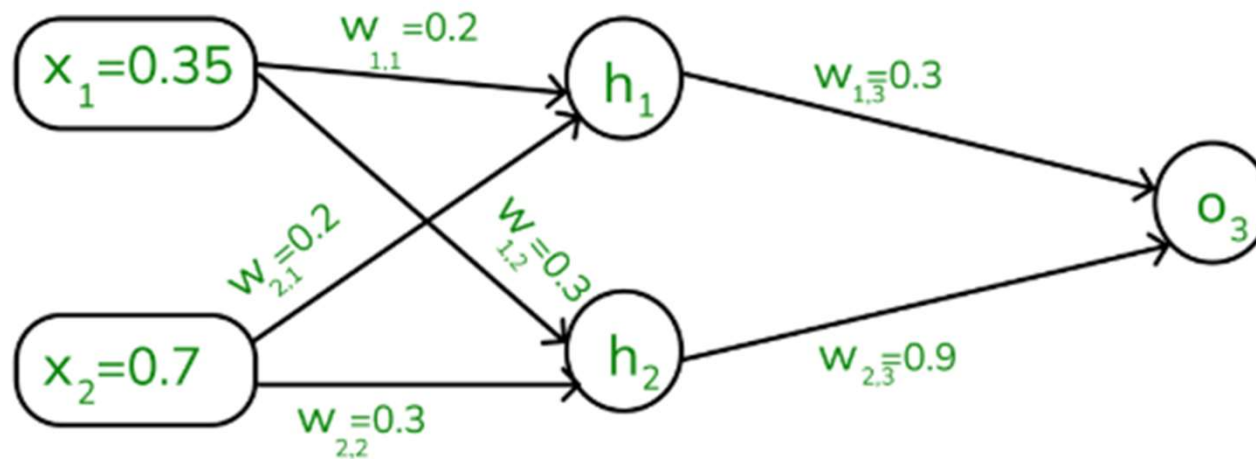
⌘ Pasos:

- Inicialización:
 - ☒ Construcción de la red.
 - ☒ Inicialización aleatoria de pesos y umbrales (-0.5, 0.5)
 - ☒ Criterio de terminación (número máximo de iteraciones,...).
 - ☒ Contador de iteraciones $n=0$.
- Fase hacia delante:
 - ☒ Calcular la salida de la red para cada patrón de entrada.
 - ☒ Calcular el error total cometido (SSE)
 - ☒ Si la condición de terminación se satisface, parar
- Fase hacia atrás:

Algoritmo backpropagation III

- Fase hacia atrás:
 - ☒ Incrementar el contador $n=n+1$.
 - ☒ Para cada neurona de salida calcular: $\delta_k = (o_k - y_k)f'(net_k)$
donde $net_j = \sum_i w_{ij}x_i + b_j$
 - ☒ Para cada unidad oculta, calcular $\delta_j = f'(net_j) \sum_k \delta_k w_{jk}$
 - ☒ Actualizar pesos: $\Delta w_{ij}(n+1) = \eta \delta_j o_i + \alpha \Delta w_{ij}(n)$
 - ☒ Volver a la fase hacia delante.
- ⌘ Inconvenientes del algoritmo backpropagation:
 - Tiempo de entrenamiento no acotado.
 - Dependiente de las condiciones iniciales:
 - ☒ Parálisis de la red.
 - ☒ Mínimos locales.

- Punto de Entrenamiento (0.35,0.7) con $y=0.5$. Learning Rate=1
- RNA con función sigmoide



Paso 1: Forward Propagation (hacia adelante)

Cálculo de las entradas y salidas para las neuronas ocultas:

Para neurona h_1 :

$$h_1^{in} = x_1w_{1,1} + x_2w_{2,1} = (0.35 \times 0.2) + (0.7 \times 0.2) = 0.07 + 0.14 = 0.21$$

Salida de la neurona oculta h_1 :

$$h_1 = \frac{1}{1 + e^{-0.21}} \approx 0.5523$$

Para neurona h_2 :

$$h_2^{in} = x_1w_{1,2} + x_2w_{2,2} = (0.35 \times 0.3) + (0.7 \times 0.3) = 0.105 + 0.21 = 0.315$$

Salida de la neurona oculta h_2 :

$$h_2 = \frac{1}{1 + e^{-0.315}} \approx 0.5781$$

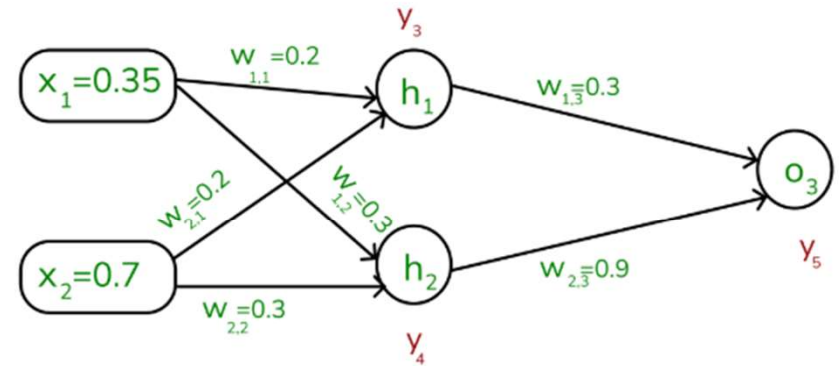
Cálculo de entrada y salida de la neurona de salida o_3 :

Entrada a o_3 :

$$o_3^{in} = h_1w_{1,3} + h_2w_{2,3} = (0.5523 \times 0.3) + (0.5781 \times 0.9) = 0.1657 + 0.5203 = 0.686$$

Salida de o_3 :

$$o_3 = \frac{1}{1 + e^{-0.686}} \approx 0.6651$$



Fórmula general para actualizar pesos:

$$w^{\text{nuevo}} = w^{\text{viejo}} + (\eta \times \delta \times \text{entrada})$$

$$\Delta w_{ij} = \eta \times \delta_j \times O_j$$

¿Cómo se calcula exactamente el error (delta)?

El error en la neurona de salida **no** es simplemente la diferencia entre salida esperada y salida obtenida, aunque sí depende de ella.

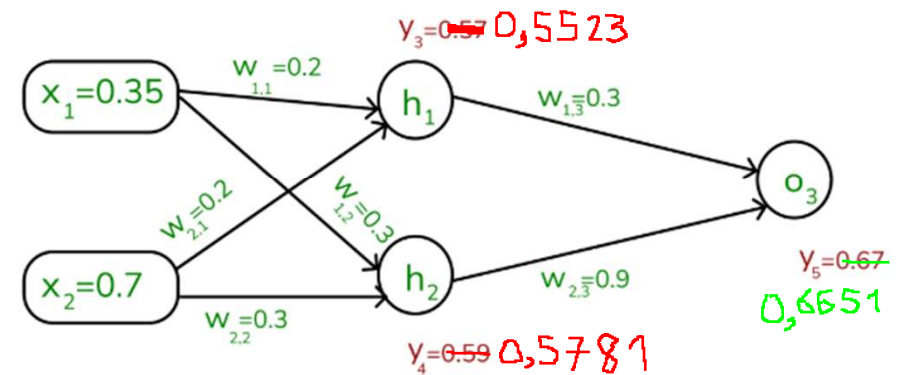
La fórmula general es:

$$\delta = (\text{salida esperada} - \text{salida obtenida}) \times f'(\text{entrada})$$

donde:

- $f'(\text{entrada})$ es la derivada de la función sigmoide evaluada en la entrada, que simplificada es:

$$f'(x) = \text{salida} \times (1 - \text{salida})$$



- Error en la neurona salida o_3 :

$$\delta_{o3} = (y - o_3) \times o_3 \times (1 - o_3)$$

Calculando los errores:

- Para la neurona de salida o_3 , salida esperada $y = 0.5$:

$$\delta_{o3} = (0.5 - 0.6651) \times 0.6651 \times (1 - 0.6651)$$

$$\delta_{o3} = (-0.1651) \times 0.6651 \times 0.3349 \approx -0.0368$$

- Errores en neuronas ocultas:

El error para las neuronas ocultas es la suma ponderada del error de la siguiente capa (hacia delante) multiplicada por la derivada de la salida de esa neurona oculta:

- Para h_1 :

$$\delta_{h1} = h_1(1 - h_1) \times w_{1,3}\delta_{o3}$$

$$\delta_{h1} = 0.5523 \times (1 - 0.5523) \times (0.3 \times -0.0368)$$

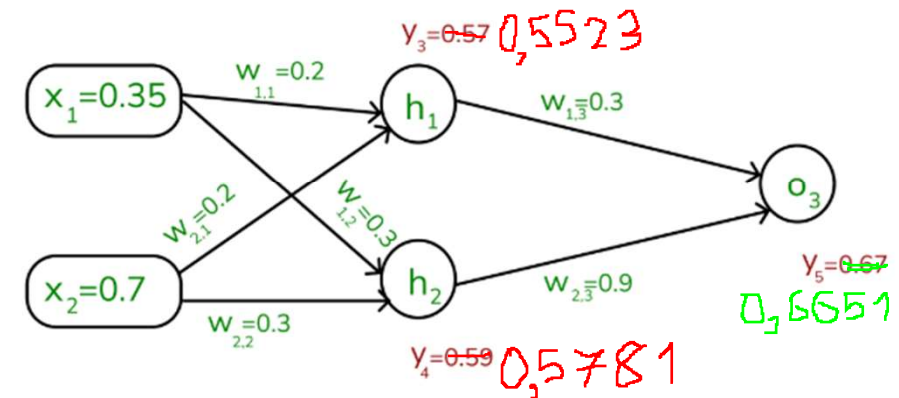
$$\delta_{h1} = 0.2473 \times (-0.0110) \approx -0.0027$$

- Para h_2 :

$$\delta_{h2} = h_2(1 - h_2) \times w_{2,3}\delta_{o3}$$

$$\delta_{h2} = 0.5781 \times (1 - 0.5781) \times (0.9 \times -0.0368)$$

$$\delta_{h2} = 0.2439 \times (-0.0331) \approx -0.0081$$



Fórmula general para actualizar pesos:

$$w^{\text{nuevo}} = w^{\text{viejo}} + (\eta \times \delta \times \text{entrada})$$

Actualizando capa salida:

- $w_{1,3}$:

$$0.3 + 1 \times (-0.0368) \times 0.5523 = 0.3 - 0.0203 = 0.2797$$

- $w_{2,3}$:

$$0.9 + 1 \times (-0.0368) \times 0.5781 = 0.9 - 0.0213 = 0.8787$$

Actualizando capa oculta:

- $w_{1,1}$:

$$0.2 + 1 \times (-0.0027) \times 0.35 = 0.2 - 0.00095 = 0.19905$$

- $w_{2,1}$:

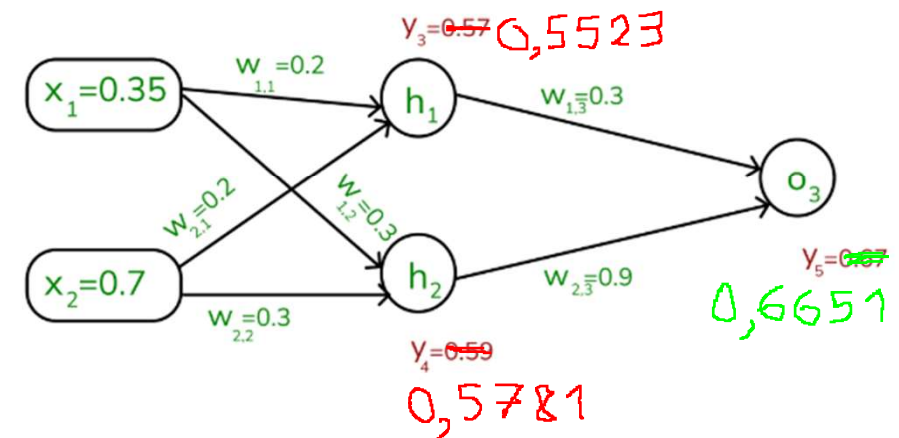
$$0.2 + 1 \times (-0.0027) \times 0.7 = 0.2 - 0.00189 = 0.19811$$

- $w_{1,2}$:

$$0.3 + 1 \times (-0.0081) \times 0.35 = 0.3 - 0.00284 = 0.29716$$

- $w_{2,2}$:

$$0.3 + 1 \times (-0.0081) \times 0.7 = 0.3 - 0.00567 = 0.29433$$



● Pesos antes y después (resumen):

Peso	Antes	Después
$w_{1,1}$	0.2	0.19905
$w_{2,1}$	0.2	0.19811
$w_{1,2}$	0.3	0.29716
$w_{2,2}$	0.3	0.29433
$w_{1,3}$	0.3	0.2797
$w_{2,3}$	0.9	0.8787

Observa que la salida de la red ahora es 0.6595, ligeramente más cerca del objetivo (0.5) en comparación con la anterior (0.6651), pero todavía bastante lejos. Esto indica que se necesitarán más iteraciones para acercarse significativamente a la salida deseada. [↗]

Parámetros

- ¿Qué es solver en MLPClassifier? El parámetro solver en MLPClassifier define el algoritmo de optimización utilizado para actualizar los pesos de la red neuronal durante el entrenamiento

Solver	Tipo	Características
'lbfgs'	Algoritmo de optimización cuasi-Newton	Mejor para conjuntos de datos pequeños y bien condicionados. Generalmente converge más rápido pero consume más memoria.
'sgd'	Descenso de gradiente estocástico	Funciona bien con grandes volúmenes de datos pero requiere una buena selección de tasa de aprendizaje. Puede ser más lento en converger.
'adam'	Adaptative Moment Estimation	Recomendada por defecto. Funciona bien en la mayoría de los casos sin necesidad de ajuste fino. Combina momentums y tasas de aprendizaje adaptativas.

Redes neuronales:

Los perceptrones multicapa (MLP) son una de las arquitecturas más fundamentales y ampliamente utilizadas en el aprendizaje profundo.

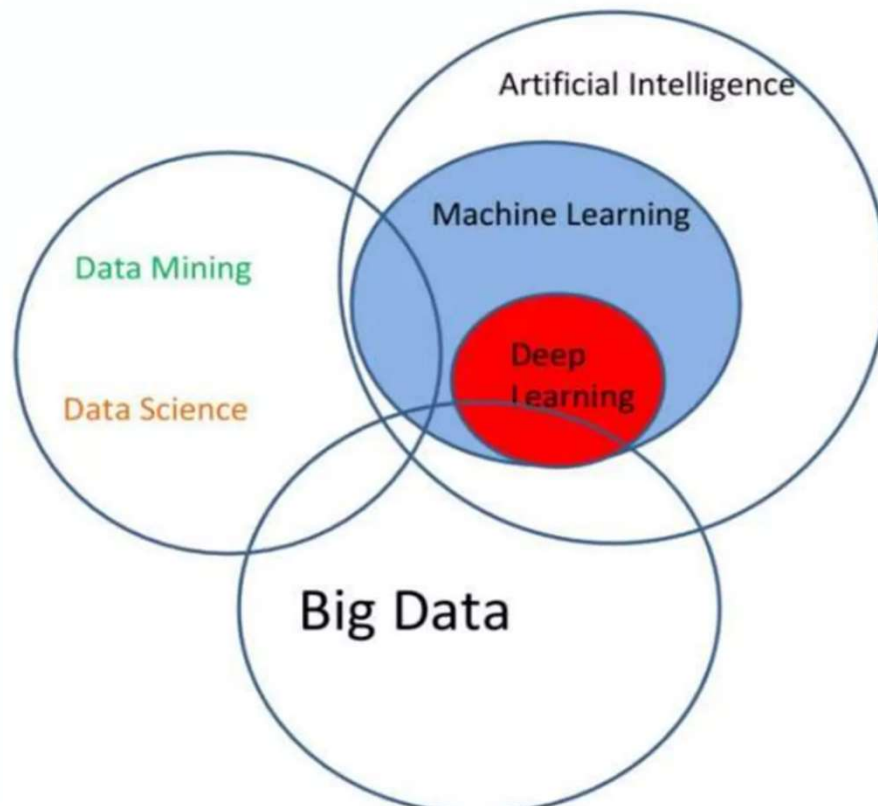
Su capacidad para modelar relaciones complejas y no lineales los hace aplicables a una amplia gama de problemas.

Su eficiencia y eficacia dependen del dominio del problema, la calidad de las características de entrada y la cantidad de datos de entrenamiento disponibles.

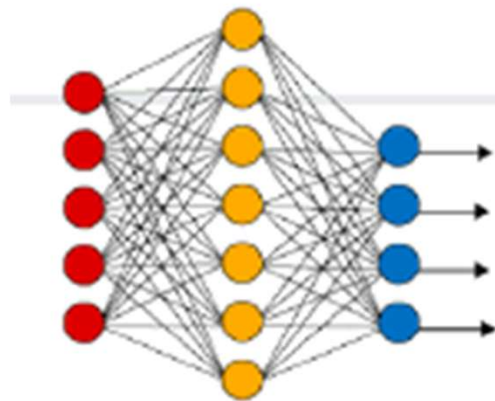
Los perceptrones multicapa también se denominan redes neuronales totalmente conectadas.

Las redes neuronales convolucionales (CNN) superan a las MLP en tareas relacionadas con imágenes debido a su capacidad para capturar jerarquías espaciales. Las redes neuronales recurrentes (RNN) y los transformadores sobresalen en tareas secuenciales como el procesamiento del lenguaje natural y la predicción de series temporales.

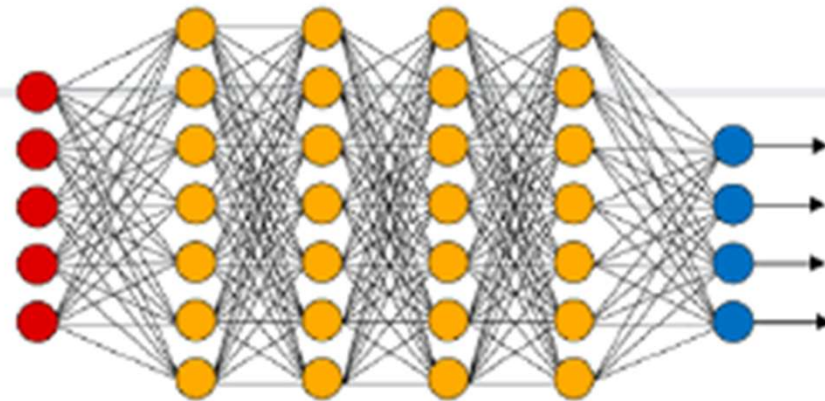
Aprendizaje Profundo



Simple Neural Network



Deep Learning Neural Network



● Input Layer

● Hidden Layer

● Output Layer

Deep Learning

- Las arquitecturas de aprendizaje profundo combinan una fase de extracción automática de características con una fase de aprendizaje supervisado.
- La fase de extracción de entidades utiliza una jerarquía de varias capas de extracción de entidades.
- A partir de entidades primitivas de bajo nivel en la capa inicial, la salida de cada capa de entidades proporciona las entidades de entrada a la siguiente capa de entidades superior.
- Todas las características se utilizan en el modelo final de aprendizaje supervisado.

Un ejemplo de arquitectura

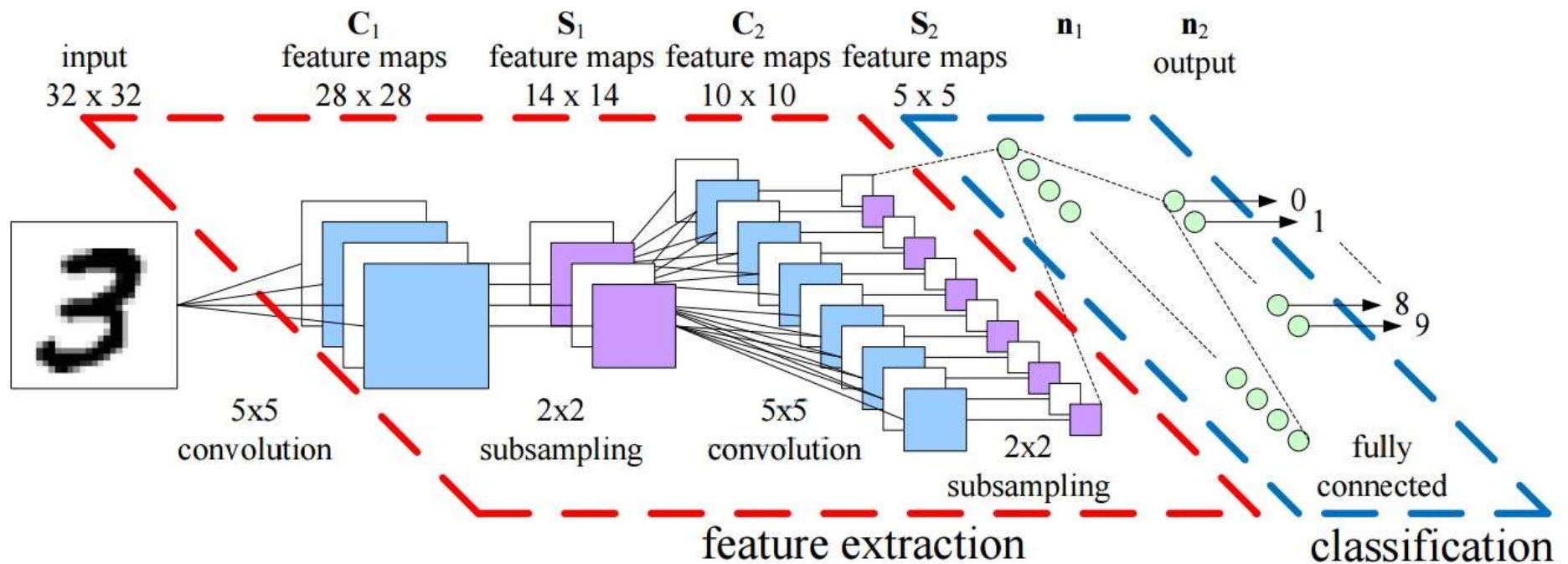
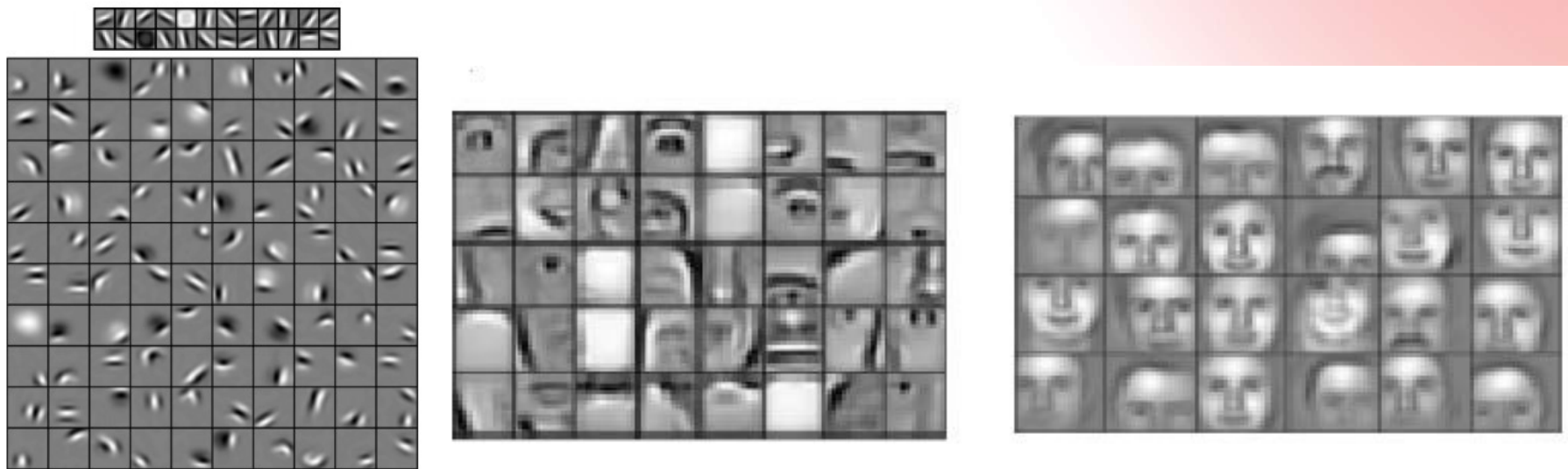


Image: M. Peemen, B. Mesman, and H. Corporaal. Efficiency Optimization of Trainable Feature Extractors for a Consumer Platform. Proceedings of the 13th International Conference on Advanced Concepts for Intelligent Vision Systems, 2011.

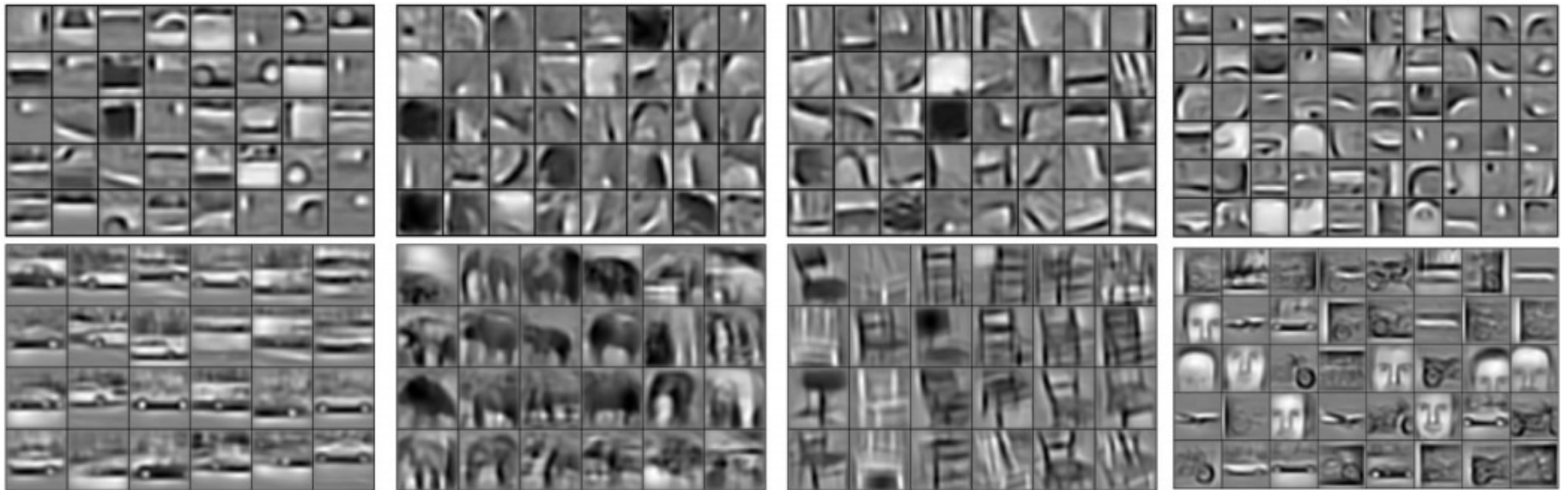


First, second, and third feature layer bases learned for faces

Image: Honglak Lee and colleagues (2011) from “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks”. Communications of the ACM, Vol. 54 No. 10, Pages 95-103

Second
layer

Third
layer



Cars

Elephants

Chairs

Faces, cars, airplanes,
motorbikes

Image: Honglak Lee and colleagues (2011) from “Unsupervised Learning of Hierarchical Representations with Convolutional Deep Belief Networks”. Communications of the ACM, Vol. 54 No. 10, Pages 95-103

CNN and RNN

Las redes neuronales convolucionales (CNN) son una clase de arquitecturas de aprendizaje profundo diseñadas para imágenes y videos.

La fortaleza clave de las CNN radica en su capacidad para aprender automáticamente representaciones de características jerárquicas a través de campos receptivos locales, compartir el peso e invariancia espacial.

A diferencia de las redes neuronales tradicionales, que tratan las entradas como vectores planos, las CNN conservan la estructura topológica de los datos, lo que les permite reconocer patrones independientemente de la posición o la orientación

Las CNN constan de varias capas clave que transforman los datos de entrada en representaciones significativas de alto nivel. Los tres tipos principales de capas en las CNN son las capas convolucionales, las capas de agrupación y las capas totalmente conectadas. Se puede acceder a una gran visualización de las redes neuronales convolucionales

https://adamharley.com/nn_vis/cnn/3d.html

CNN

LeNet-5 (1998) – Reference: Yann LeCun et al., Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 1998. DOI: 10.1109/5.726791

AlexNet (2012) – Deep Learning Breakthrough: Reference: Alex Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NeurIPS, 2012. DOI: 10.1145/3065386

VGGNet (2014) – Simplicity & Uniformity: Reference: Karen Simonyan, Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR, 2015. arXiv:1409.1556

GoogLeNet (Inception) (2014) – Reference: Christian Szegedy et al., Going Deeper with Convolutions, CVPR, 2015. DOI: 10.1109/CVPR.2015.7298594

ResNet (2015) – Reference: Kaiming He et al., Deep Residual Learning for Image Recognition, CVPR, 2016. DOI: 10.1109/CVPR.2016.90

•

RNN

Las redes neuronales recurrentes (RNN) son un tipo de red neuronal diseñada para procesar datos secuenciales. A diferencia de las redes feedforward, que tratan cada entrada de forma independiente, las RNN tienen una estructura recurrente que permite que la información persista a lo largo de los periodos de tiempo. Esta capacidad de retener memoria hace que las RNN sean especialmente eficaces para tareas como el procesamiento del lenguaje natural (NLP), el reconocimiento de voz y la previsión de series temporales.

La característica fundamental de las RNN es su arquitectura en bucle, que les permite mantener un estado oculto que se actualiza con cada nueva entrada. Esto los hace adecuados para aplicaciones que requieren conocimiento contextual, como predecir la siguiente palabra en una oración, reconocer el lenguaje hablado o pronosticar los precios de las acciones.

Las RNN constan de capas similares a las redes neuronales tradicionales, pero incluyen conexiones recurrentes que permiten que la información fluya a través de periodos de tiempo

Activation Functions for Output Layers

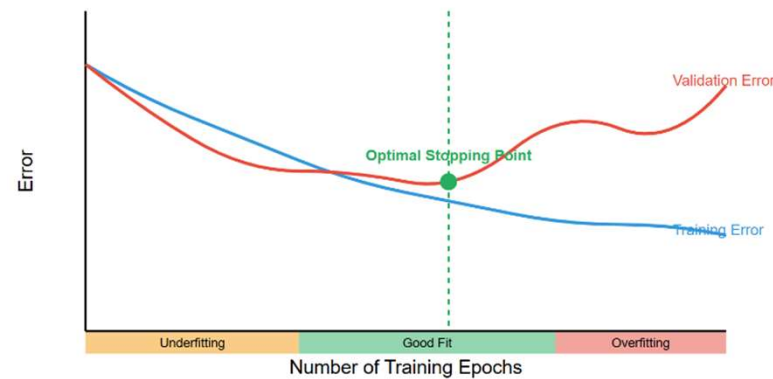
- Activación lineal: Para las tareas de regresión, a menudo se utiliza una función de activación lineal en la capa de salida. En este caso, no se aplica ninguna transformación a la suma ponderada. Este enfoque es apropiado cuando la salida puede tomar cualquier valor real.
- Activación sigmoide: Para problemas de clasificación binaria o tareas de clasificación de etiquetas múltiples, la función de activación sigmoide se usa comúnmente en la capa de salida. Comprime la salida a un rango entre 0 y 1, lo que puede interpretarse como una probabilidad. Esto facilita la aplicación de un umbral (normalmente 0,5) para decidir la etiqueta de clase.
- Activación softmax: Para problemas de clasificación multiclase, donde la red debe elegir entre varias clases, se utiliza la función softmax en la capa de salida. Softmax convierte un vector de puntuaciones arbitrarias de valor real (logits) en una distribución de probabilidad sobre las clases.

Regularizacion

- Regularización de L2 (disminución de peso): La regularización de L2 agrega un término de penalización a la función de pérdida original, desalentando los valores de peso grandes. fomenta modelos más simples, con menos pesos distintos de cero.
- Regularización L1: La regularización L1 agrega una penalización proporcional a los valores absolutos de los pesos: Esta técnica a menudo conduce a modelos dispersos al llevar algunos pesos a cero, realizando efectivamente la selección de características.
- Abandono (Dropout): El abandono es una técnica en la que, durante el entrenamiento, un subconjunto aleatorio de neuronas se desactiva temporalmente. Esto obliga a la red a aprender representaciones redundantes, lo que reduce la dependencia de una sola neurona y mitiga el sobreajuste. Es como entrenar un conjunto de subredes. Durante la inferencia, todas las neuronas se reactivan.

Regularizacion

- Detención temprana: la detención temprana supervisa el rendimiento en un conjunto de validación y detiene el entrenamiento cuando la pérdida de validación comienza a aumentar. Esto evita que el modelo sobreajuste los datos de entrenamiento.



Hyperparametros

Learning Rate: La tasa de aprendizaje determina la magnitud de las actualizaciones de los parámetros del modelo durante el entrenamiento. Una tasa de aprendizaje alta puede hacer que el modelo sobrepase los mínimos y conduzca a un entrenamiento inestable. Una tasa de aprendizaje baja puede dar lugar a una convergencia muy lenta.

Batch Size: El tamaño del lote hace referencia al número de muestras de entrenamiento procesadas antes de actualizar los parámetros del modelo. Lotes pequeños introducen ruido en las estimaciones de gradiente, lo que puede ayudar al optimizador a escapar de los mínimos locales. Los lotes más grandes proporcionan estimaciones de gradiente más estables y precisas, pero requieren más memoria y pueden provocar un sobreajuste.

Regularization Parameters: Las técnicas de regularización ayudan a prevenir el sobreajuste. Los parámetros de regularización, como L1 y L2, influyen en los resultados finales.

Número of Epochs: El número de épocas determina cuántas veces se pasa todo el conjunto de datos de entrenamiento a través de la red. Muy pocas épocas pueden llevar a un ajuste insuficiente. Demasiadas épocas pueden dar lugar a un sobreajuste. La detención temprana basada en el rendimiento de la validación es una técnica común para determinar dinámicamente el número óptimo de épocas.

Hyperparametros

Arquitecturas de modelos y recuento de parámetros: la arquitectura (su profundidad (número de capas), su anchura (número de neuronas por capa) y su conectividad general, es en sí misma un hiperparámetro. Al experimentar con arquitecturas de modelos, comience con un modelo de referencia pequeño y bien establecido y modifique gradualmente los parámetros.

Profundidad y anchura: Las redes más profundas pueden aprender características más abstractas y jerárquicas, mientras que las redes más amplias pueden capturar una mayor variedad de patrones en los datos. Sin embargo, el aumento de la profundidad y el ancho también aumenta el riesgo de sobreajuste y el costo computacional.

• **Innovaciones arquitectónicas:** Las arquitecturas actuales incluyen componentes especializados como capas convolucionales, capas recurrentes o mecanismos de atención. Cada uno de estos componentes introduce hiperparámetros adicionales.

Pros y contras del aprendizaje profundo

- **Pros:**

- *Potente: el aprendizaje profundo ha logrado avances significativos sobre otros enfoques de aprendizaje automático en muchas tareas de aprendizaje difíciles, lo que ha llevado a un rendimiento de vanguardia en muchos dominios diferentes..*
- *Realiza una extracción automática efectiva de características, lo que reduce la necesidad de conjeturas y heurísticas sobre este problema clave.*
- *El software actual proporciona arquitecturas flexibles que se pueden adaptar a nuevos dominios con bastante facilidad.*

- **Cons:**

- *Puede requerir grandes cantidades de datos de entrenamiento.*
- *Puede requerir grandes cantidades de potencia de cómputo.*
- *Las arquitecturas pueden ser complejas y, a menudo, deben adaptarse en gran medida a una aplicación específica.*
- *Los modelos resultantes pueden no ser fácilmente interpretables.*

Software de aprendizaje profundo para Python

- **Keras** <https://keras.io/>
 - (alto nivel) – parte de tensorflow. Interfaz de alto nivel
- PyTorch – principalmente NLP
- **TensorFlow** <https://www.tensorflow.org/> (bajo nivel)

Ejemplo con Keras

Source: F. Chollet. Deep Learning with Python.

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

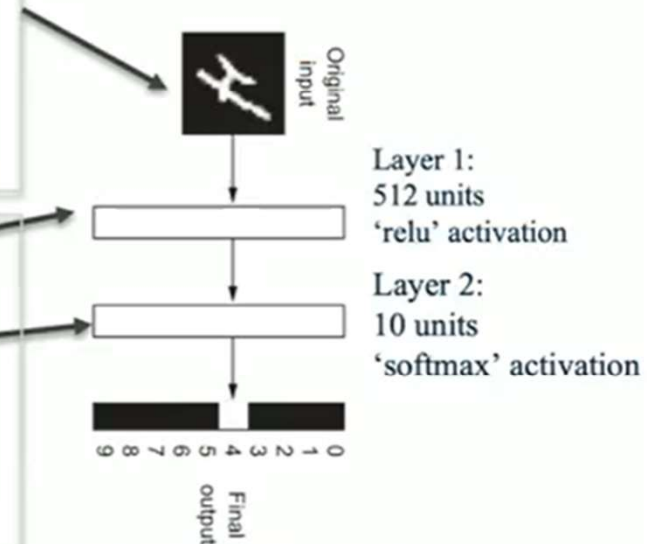
1. Data preparation

```
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

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

```
network.fit(train_images, train_labels, epochs=5, batch_size=128)
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

2. Model definition



3. Model training

4. Model evaluation



Hardware stack



- Las redes neuronales realizan operaciones matriciales en pasos hacia adelante y hacia atrás
- Muchas de estas operaciones se pueden realizar en forma de elemento o en fragmentos
- Estos elementos / fragmentos se pueden asignar a varios núcleos
- Las GPU modernas tienen 1000 núcleos, que son mucho más rápidos en el cálculo paralelo que las CPU con unos pocos 100 núcleos.

GPU de consumo

- Bueno para proyectos pequeños y para un número limitado de GPU
- Funciones limitadas y RAM de GPU
- Optimizado para el rendimiento visual
- Refrigeración activa
- La serie A0B0 más popular, donde A = 1,2,3,4 y B = 6,7,8,9
- Por ejemplo, la mejor elección en 2023: NVIDIA RTX 4090 24GB
- Más información: <https://www.nvidia.com/en-eu/geforce/graphics-cards/>

- GPUs semiprofesionales
- Bueno para proyectos pequeños y para un número limitado de GPU
- Optimizado para el rendimiento visual, 64 bits
- Refrigeración activa
- Ada Lovelace y la arquitectura de Ampere
- Por ejemplo, RTX 6000, 5000, 4500, 4000, A6000, A5000, etc.
- Más información: <https://www.nvidia.com/en-us/design-visualization/desktop-graphics/>



- GPU de nivel de servidor
- Seamless” integration
 - PCIe 6.0 16 lanes: 128 Gb/s
 - In-GPU: 3.35/2/7.8 TB/s (SMX/PCIe/NVL)
 - Multi GPU: NVSwitch 900 Gb/s
 - Multi node: Infiniband/Ethernet 400Gb/s
 - Passive cooling
- Servidor listo para usar DGX H100
- More info:
 - GPU: <https://www.nvidia.com/en-us/data-center/h100/>
 - Server: <https://www.nvidia.com/en-us/data-center/dgx-h100/>

- NVIDIA DGX H100 SuperPod
- Off-the-shelf solution for AI infrastructure
- Built from 31...127 DGX H100 systems
- Important requirements:
 - Dry-Bulb temperature: 18-27 °C
 - Humidity range: 5.5 °C to 60% RH and 15 °C DP
- More details: <https://docs.nvidia.com/nvidia-dgx-superpod-data-center-design-dgx-h100.pdf>

GPUs en la nube

- Google Cloud, AWS, MS Azure, Oracle, NVIDIA DGX Cloud.
- Flexible, bueno para escalar
- Los entrenamientos a gran escala no son triviales
- Excelente para opciones de inferencia
- Comparaciones: <https://fullstackdeeplearning.com/cloud-gpus/>
 - <https://cloud-gpus.com/>



#AIskills4all |  @AIskillsEU |  [linkedin.com/AIskillsEU](https://www.linkedin.com/AIskillsEU)



www.aiskills.eu

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.