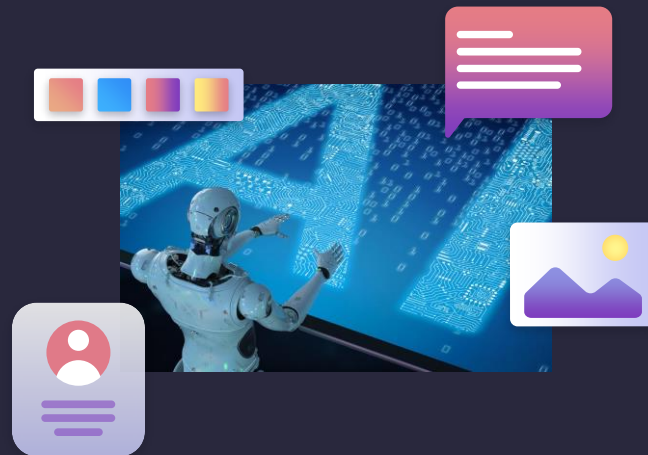


/TEMA 5

Programación de redes neuronales artificiales



<https://www.youtube.com/watch?v=8Dotiqbtvoo>



/CONTENIDOS



- /01** /Introducción
- /02** /El Perceptrón Multicapa
- /03** /2 Prácticas de introducción a Tensorflow y Keras
- /04** /Profundizando en tensorflow y Keras
- /05** /Evaluación de modelos

Parte

I

II

III

/01 /INTRODUCCIÓN

Un poco de cultura - Librerías para trabajo con RNAs

PyTorch: Framework para DeepLearning. Desarrollado por Facebook. En 2017 se hizo open source. Se usa fundamentalmente para aplicaciones de procesamiento de lenguaje natural.

TensorFlow: Framework para DeepLearning. Desarrollado por Google como open-source. Es una librería de matemáticas simbólicas para redes neuronales.

Keras: Librería de alto nivel de Python para redes neuronales construida sobre TensorFlow. Son envoltorios para las librerías de TensorFlow.

Antes de comenzar

Un simple ejercicio: ¿Cuál es la relación entre estos valores?

$X = -1, 0, 1, 2, 3, 4$

$Y = -3, -1, 1, 3, 5, 7$

Si has visto la relación... ¿Cómo has llegado a ella?



pip install tensorflow

Práctica 5.1. Keras Hello World - Modela la relación con keras de tensorflow

```
import tensorflow as tf
import numpy as np
from tensorflow import keras

#crear la neurona más sencilla
model = tf.keras.Sequential([keras.layers.Dense(units=1, input_shape=[1])])

#compilar el modelo
model.compile(optimizer='sgd', loss='mean_squared_error')

#alimentar los datos
xs = np.array([-1, 0, 1, 2, 3, 4], dtype=float)
ys = np.array([-3, -1, 1, 3, 5, 7], dtype=float)

#entrenar
model.fit(xs, ys, epochs=500)

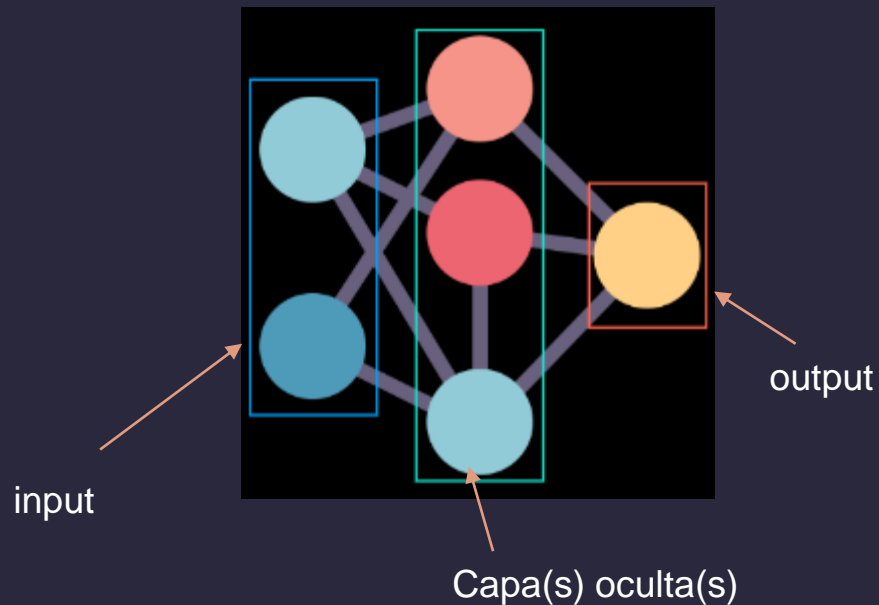
#predecir
print(model.predict([10.0]))
```



/02 /El perceptrón multicapa



Anatomía de una red neuronal

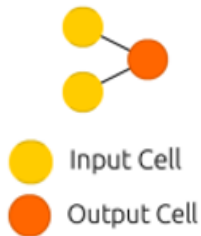


Aclaración sobre “nomenclatura”: ¿Cuántas neuronas tiene esta red?

Neurona artificial

Dos entradas
(x,y)

Perceptron (P)



Una salida

Las limitaciones se pueden superar utilizando RNAs (perceptrones multicapa)

Red de neuronas artificiales

Feed Forward (FF)



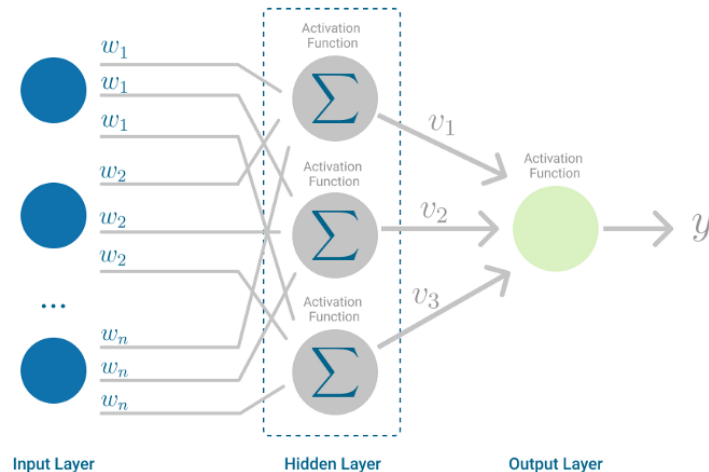
$$W_1 = \begin{bmatrix} w_{0,1}^{(1)} & w_{0,2}^{(1)} & \dots & w_{0,U_1}^{(1)} \\ w_{1,1}^{(1)} & w_{1,2}^{(1)} & \dots & w_{1,U_1}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{N,1}^{(1)} & w_{N,2}^{(1)} & \dots & w_{N,U_1}^{(1)} \end{bmatrix}$$

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} y_i(n)$$

N, número de neuronas en la capa oculta

U1, número de entradas

En la entrada a cada capa se aplica una función de activación



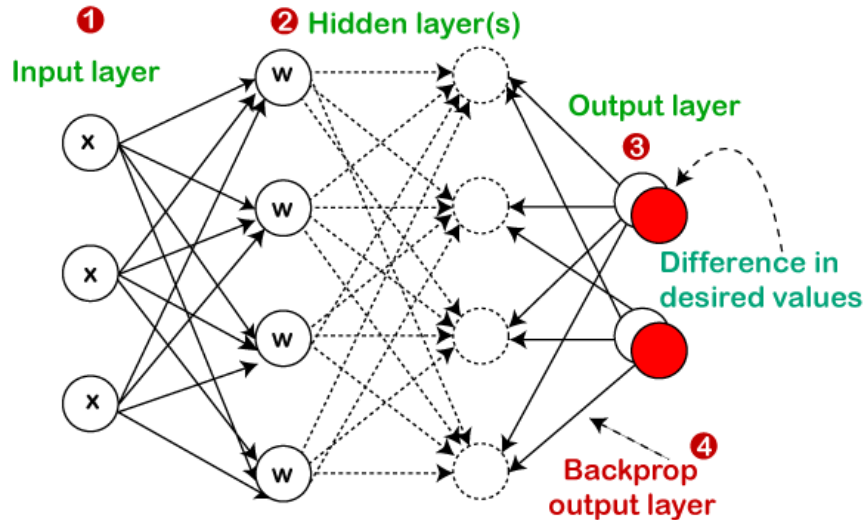
1. Feedforward | Mean Squared Error (MSE) computed

2. Backpropagation | Gradient is computed

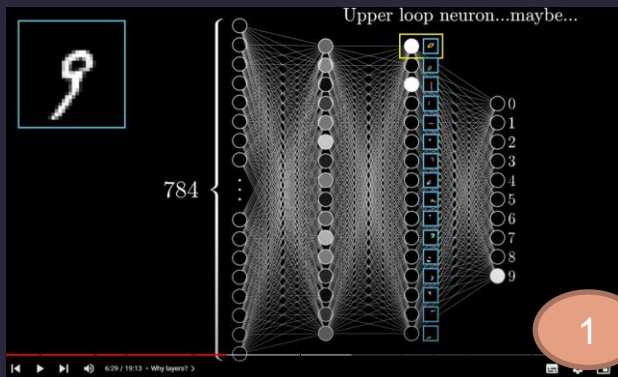
$$\Delta_w(t) = \underbrace{-\epsilon}_{\text{Gradient Current Iteration}} \underbrace{\frac{dE}{dw(t)}}_{\text{Weight vector}} + \underbrace{\alpha}_{\text{Learning Rate}} \underbrace{\Delta_{w(t-1)}}_{\text{Gradient Previous Iteration}}$$

Propagación hacia atrás (Backpropagation):

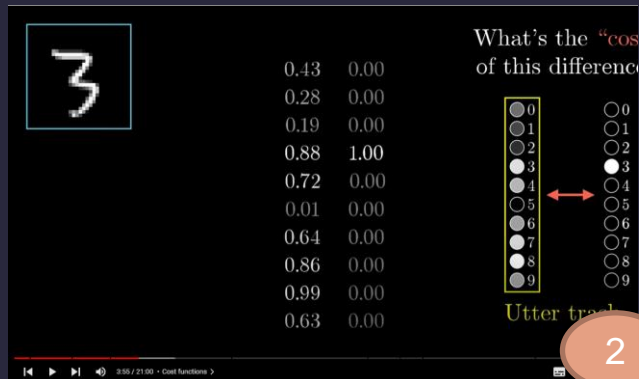
- Es la esencia del entrenamiento de una red neuronal
- Realiza el cálculo del descenso del gradiente para toda la red: **Regla de la cadena**
- Itera por las capas ocultas de la red propagando hacia atrás la actualización de pesos



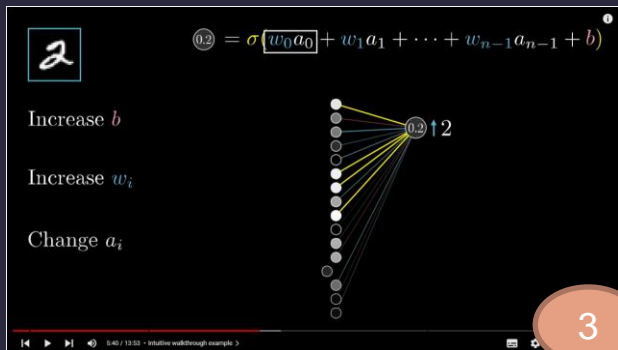
Series de vídeos de animated maths que merece la pena ver:



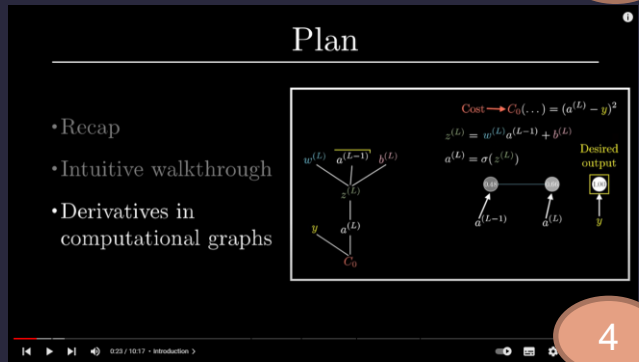
1



2



3



4

Neuronas en redes: RNAs : Red de neuronas artificiales

-  Input Cell
-  Backfed Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Capsule Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Gated Memory Cell
-  Kernel
-  Convolution or Pool

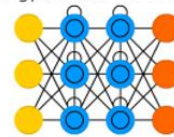
Deep Feed Forward (DFF)



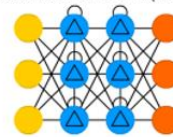
Recurrent Neural Network (RNN)



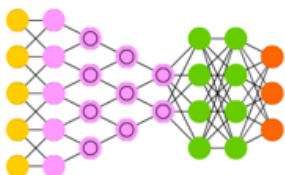
Long / Short Term Memory (LSTM)



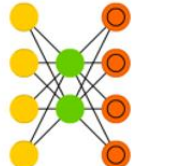
Gated Recurrent Unit (GRU)



Deep Convolutional Network (DCN)



Auto Encoder (AE)



Variational AE (VAE)



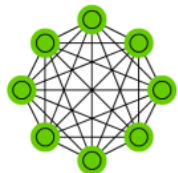
Denoising AE (DAE)



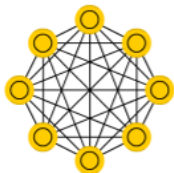
Sparse AE (SAE)



Markov Chain (MC)



Hopfield Network (HN)



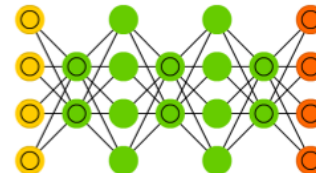
Boltzmann Machine (BM)



Restricted BM (RBM)

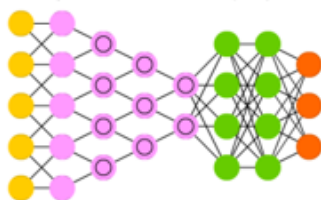


Deep Belief Network (DBN)

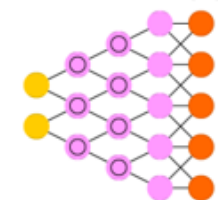


-  Input Cell
-  Backfed Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Capsule Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Gated Memory Cell
-  Kernel
-  Convolution or Pool

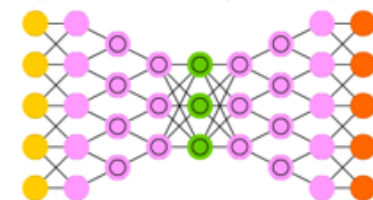
Deep Convolutional Network (DCN)



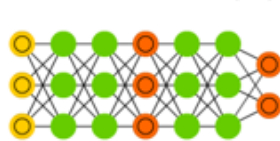
Deconvolutional Network (DN)



Deep Convolutional Inverse Graphics Network (DCIGN)



Generative Adversarial Network (GAN)



Liquid State Machine (LSM)



Extreme Learning Machine (ELM)



Echo State Network (ESN)



Deep Residual Network (DRN)



Differentiable Neural Computer (DNC)



Neural Turing Machine (NTM)



Capsule Network (CN)



Kohonen Network (KN)



Attention Network (AN)

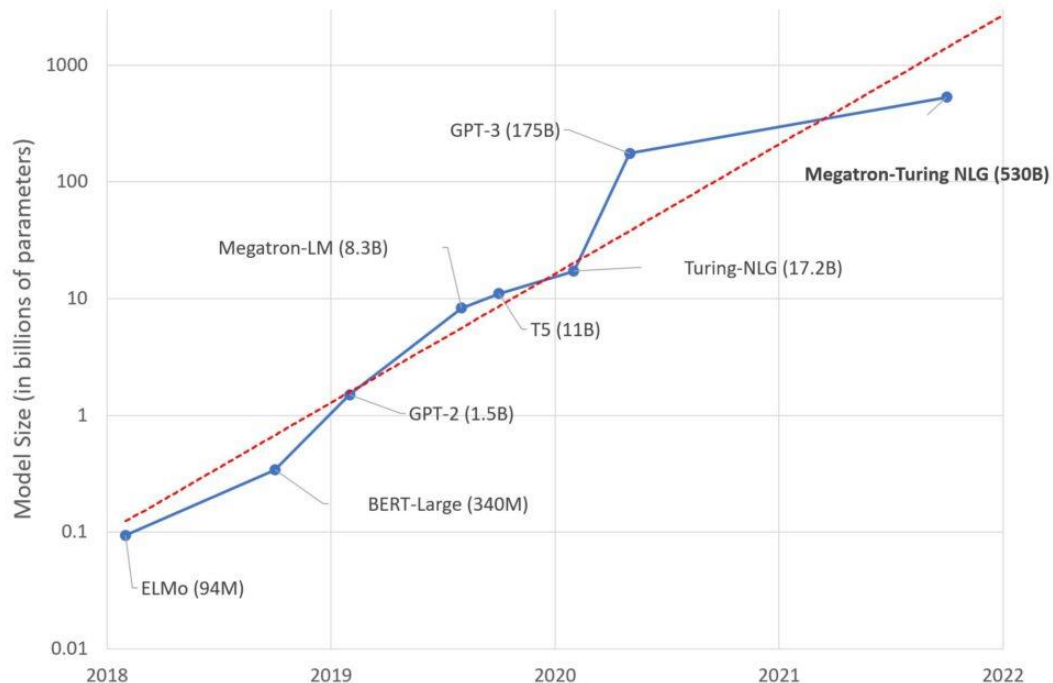


¿Cómo de grandes pueden ser las RNAs?

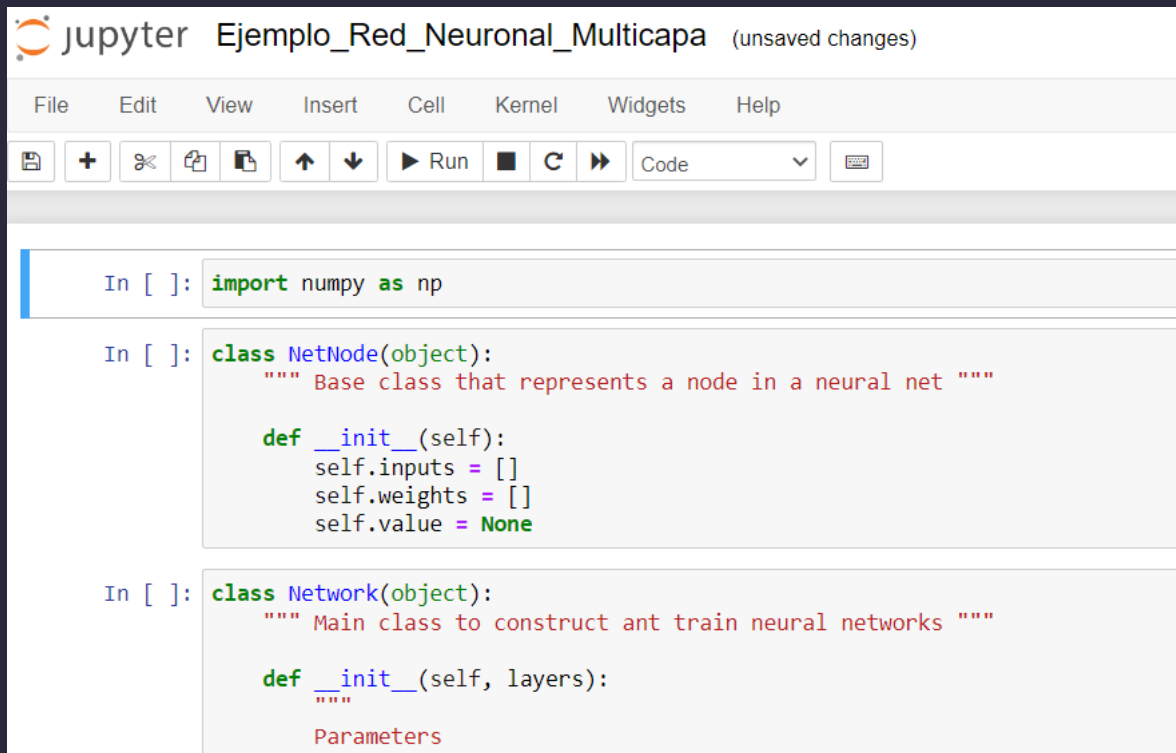
**El cerebro humano tiene
alrededor de 80 millones de
neuronas**

**GPT-3 tiene 175 billones de
neuronas (Billón americano)**

**Megatron-Turing NGL tiene 530
billones de neuronas**



Ejemplo: Iris con una red de tipo feed forward con backpropagation



```
Jupyter Ejemplo_Red_Neuronal_Multicapa (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
[Icons] Run [Icons] Code v [Icon]

In [ ]: import numpy as np

In [ ]: class NetNode(object):
        """ Base class that represents a node in a neural net """

        def __init__(self):
            self.inputs = []
            self.weights = []
            self.value = None

In [ ]: class Network(object):
        """ Main class to construct and train neural networks """

        def __init__(self, layers):
            """
            Parameters
```



<https://playground.tensorflow.org/>

Epoch
000,375

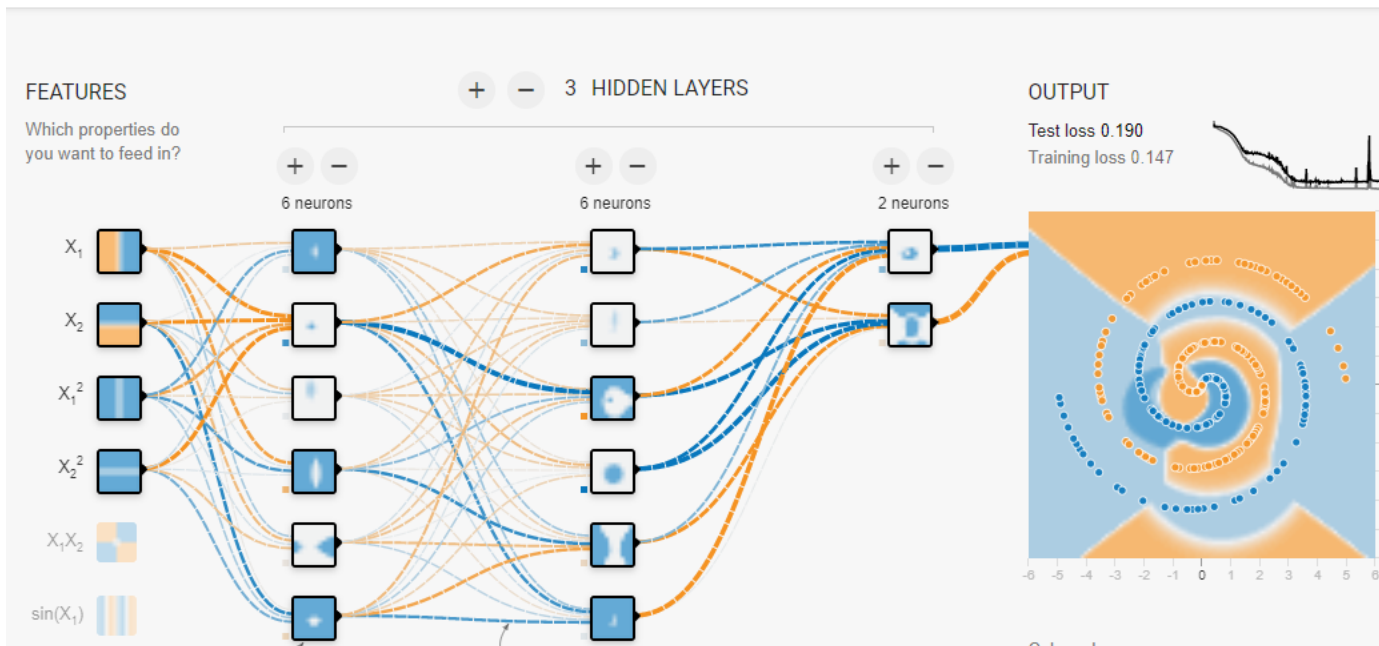
Learning rate
0.03

Activation
ReLU

Regularization
None

Regularization rate
0

Problem type
Classification





/03

/Prácticas con Tensorflow y Keras



PRACTICA 5.2

Entrenamiento de una RNA con keras para el conjunto IRIS



PRACTICA 5.3

Entrenamiento de una RNA

+ Input layer

+ Hidden layer

+ Output Layer



Un problema de visión artificial con Deep Neural Network
* El clasificador de ropa (MNIST Zalando) con DNNs

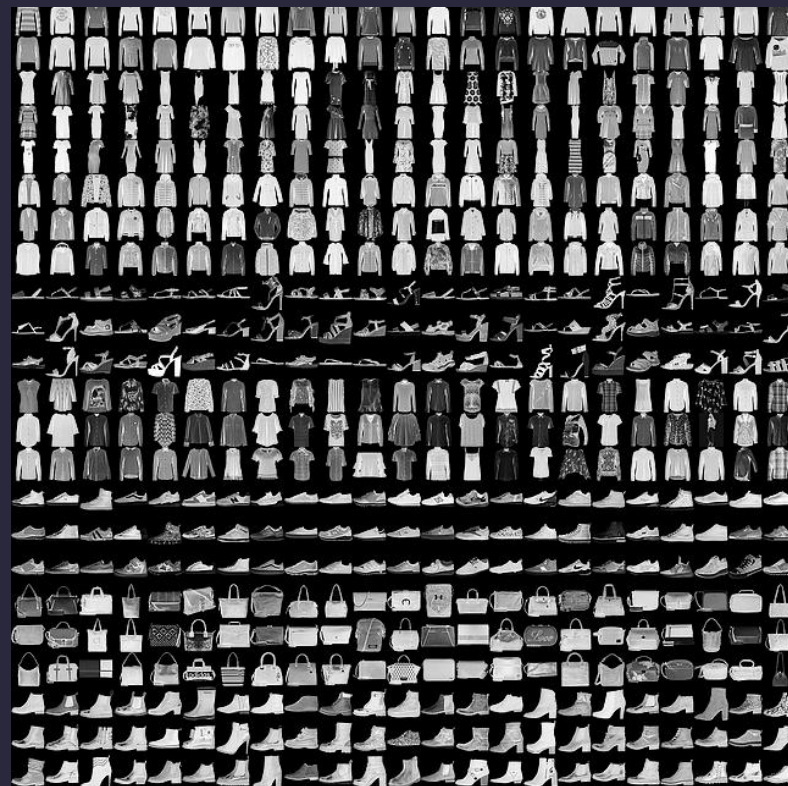
Esta práctica nos sirve de enlace con el tema siguiente: **VISIÓN ARTIFICIAL**

Entrenar el modelo Fashion MNIST con TensorFlow y Keras:

MNIST:
Modified National Institute of Standards
and Technology database

Crear con TensorFlow una RNA para
reconocimiento de imágenes de moda

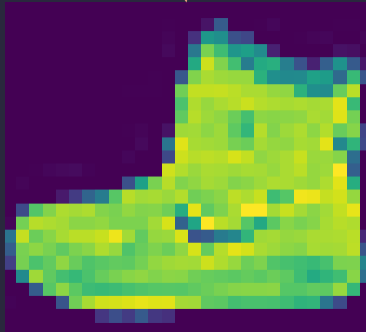
<https://github.com/zalandoresearch/fashion-mnist>



a) Cargar el modelo (MNIST se incorpora en los datasets de keras)

```
import tensorflow as tf  
from tensorflow import keras
```

```
mnist = tf.keras.datasets.fashion_mnist  
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
```



9

```
class_names = ['Camiseta', 'Pantalón', 'Jersey',  
               'Vestido', 'Abrigo', 'Sandalia', 'Camisa',  
               'Zapatilla', 'Bolso', 'Bota']  
x, idx = tf.unique(training_labels)  
for i in x:  
    print(class_names[i])
```

b) Crear un modelo con Keras

¿Qué significa ese 28,28 y ese 10 en la RNA?

```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28,28)),  
    tf.keras.layers.Dense(128, activation=tf.nn.relu),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])
```



¿Qué significa ese 128?

```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28,28)),  
    tf.keras.layers.Dense(128, activation=tf.nn.relu),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])
```





f_0

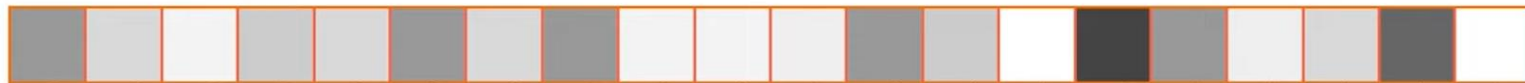
f_1

.....

f_{127}

$f_0 . f_1 . f_2 \dots f_{127} = 9$





f0

f1

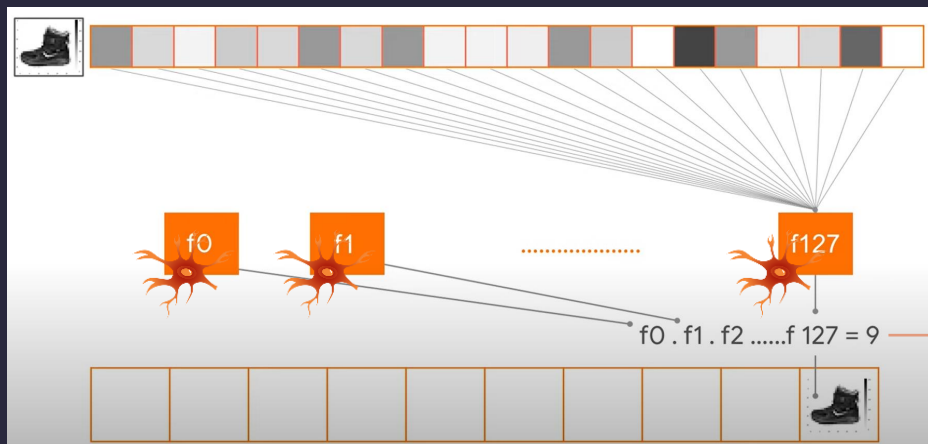
.....

f127

$f_0 . f_1 . f_2 \dots f_{127} = 9$




```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28,28)),  
    tf.keras.layers.Dense(128, activation=tf.nn.relu),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])
```



Cada f tiene sus propias variables

Sus pesos inicializados aleatoriamente

Cada función debe intentar mapear las entradas a las salidas a través de un entrenamiento

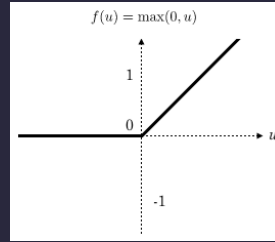
La combinación de todas estas funciones debe producir el valor deseado

```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28,28)),  
    tf.keras.layers.Dense(128, activation=tf.nn.relu),  
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)  
])
```



¿Qué hacen las funciones de activación?

```
tf.nn.relu:  
  if(x>0) return X  
  else return 0
```



tf.nn.softmax: Para que todos los elementos de la salida sumen 1

0.02	0.01	0.02	0.01	0.05	0.01	0.08	0.02	9.78
0	0	0	0	0	0	0	0	1.0

c) Compilar el modelo con un optimizador y una función de coste

```
model.compile(optimizer = tf.keras.optimizers.Adam(),  
              loss = 'sparse_categorical_crossentropy')
```

optimizer → Función que intenta encontrar los valores apropiados para generar los resultados correctos

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

loss → Función que mide cómo de bien o de mal hace las cosas en cada época

https://www.tensorflow.org/api_docs/python/tf/keras/losses

d) Entrena el modelo con el conjunto de entrenamiento

```
model.fit(training_images, training_labels, epochs=5)
```

```
model.summary()
```

```
Epoch 1/5
1875/1875 [=====] - 3s 1ms/step - loss: 0.5016 - accuracy: 0.8235
Epoch 2/5
1875/1875 [=====] - 3s 1ms/step - loss: 0.3793 - accuracy: 0.8641
Epoch 3/5
1875/1875 [=====] - 3s 1ms/step - loss: 0.3405 - accuracy: 0.8753
Epoch 4/5
1875/1875 [=====] - 3s 1ms/step - loss: 0.3159 - accuracy: 0.8836
Epoch 5/5
1875/1875 [=====] - 3s 1ms/step - loss: 0.2972 - accuracy: 0.8904
Model: "sequential"
```

Layer (type)	Output Shape	Param #
=====		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290

```
=====
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
```

e) evalúa cómo de bien o de mal funciona con el conjunto de test

```
model.evaluate(test_images, test_labels)
```

```
313/313 [=====] - 0s 999us/step - loss: 0.3550 - accuracy: 0.8753  
[0.35497158765792847, 0.8752999901771545]
```

Ahora continúa tu: Realiza los ejercicios del cuaderno
practica4.3_mnist zalando_enunciado.ipynb

Explora los resultados

- Realiza los ejercicios propuestos y contesta a las preguntas razonando las respuestas

Ejercicio 1:

Para este primer ejercicio, ejecuta el siguiente código: crea un conjunto de clasificaciones para cada una de las imágenes de prueba y luego imprime la primera entrada en las clasificaciones. La salida, después de ejecutarla, es una lista de números. ¿Por qué crees que es así y qué representan esos números?



RESUMEN



Resumen: Hiperparámetros de una RNA

Hyperparameter	Binary Classification	Multiclass classification
Input layer shape	Same as number of features (e.g. 5 for age, sex, height, weight, smoking status in heart disease prediction)	Same as binary classification
Hidden layer(s)	Problem specific, minimum = 1, maximum = unlimited	Same as binary classification
Neurons per hidden layer	Problem specific, generally 10 to 100	Same as binary classification
Output layer shape	1 (one class or the other)	1 per class (e.g. 3 for food, person or dog photo)
Hidden activation	Usually ReLU (rectified linear unit)	Same as binary classification
Output activation	Sigmoid	Softmax
Loss function	Cross entropy (tf.keras.losses.BinaryCrossentropy in TensorFlow)	Cross entropy (tf.keras.losses.CategoricalCrossentropy in TensorFlow)
Optimizer	SGD (stochastic gradient descent), Adam	Same as binary classification

```

# 1. Create a model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.Input(shape=(224, 224, 3)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(3, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train, y_train, epochs=5)

# 4. Evaluate the model
model.evaluate(X_test, y_test)

```


CONCEPTOS:

CLASES, MODELO, PREDICCIÓN, ÉPOCAS, RANGO DE APRENDIZAJE, PRECISIÓN, PÉRDIDA, ACTIVACIÓN, DENSA, CONVOLUCIÓN

```
# 1. Create the model (specified to your problem)
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])

# 2. Compile the model
model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(lr=0.0001),
              metrics=["accuracy"])

# 3. Fit the model
model.fit(X_train_full, y_train_full, epochs=100)
```

Conclusiones

Las redes neuronales artificiales son una gran herramienta para aprendizaje Machine Learning. Permiten realizar predicciones en problemas de **clasificación, regresión y clustering** y son ideales para visión artificial y percepción computacional.

Pero no son las únicas herramientas:

En sistemas de aprendizaje automáticos y modelos de inteligencia artificial estáis aprendiendo otras herramientas como SVM, árboles de decisión, RandomForest, Kmeans, etc. que superan muchas veces en rendimiento a las RNAs

Rule #1: Don't be afraid to launch a product without machine learning

<https://developers.google.com/machine-learning/guides/rules-of-ml>

Rule #1: Don't be afraid to launch a product without machine learning
if you can build a simple rule-based system that doesn't require Machine learning, do that

Rule #2: First, design and implement metrics

Rule #3: Choose machine learning over a complex heuristic.

Rule #4: Keep the first model simple and get the infrastructure right.

Rule #5: Test the infrastructure independently from the machine learning

Rule #6: Be careful about dropped data when copying pipelines



Yashaswi Kulshreshtha

I think you can use ML for literally anything as long as you can convert it into numbers and program it to find patterns. Literally it could be anything any input or output from the universe



/TEMA 5 – RNAs II

TensorFlow y Keras

Vamos a perfeccionar nuestros conocimientos sobre
Tensorflow y Keras con dos cuadernos de Jupyter:

Tensorflow: Fundamentos de las RNAs
KERAS: Regresión y clasificación

