

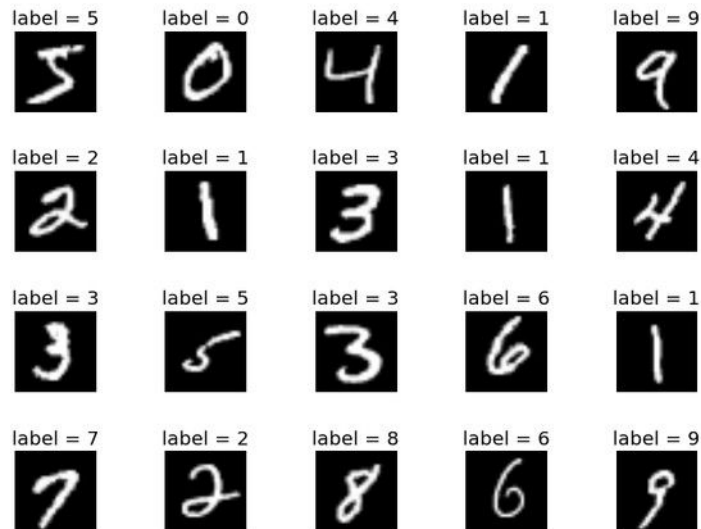


MNIST y PAPER

Sebastián Mora Carmona
Federico Idarraga Cardenas

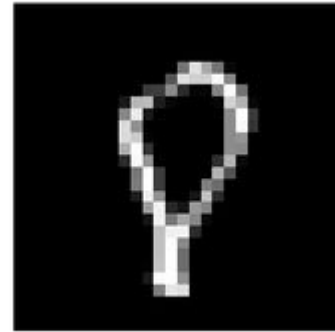
DATA MNIST

Es un conjunto de datos lleno de dígitos dibujados a mano que van del 0 al 9 con una etiqueta correspondiente que describe qué dígito se supone que representa el dibujo.



OBJETIVO MNIST

La idea detrás de trabajar con este conjunto de datos es que queremos poder entrenar un modelo que aprenda qué tipos de formas corresponden a los dígitos 0-9 y que posteriormente sea capaz de etiquetar correctamente las imágenes en las que no se ha entrenado.





CÓDIGO





IMPORTS

```
[ ] #Imports
    import tensorflow as tf
    print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.6.0
```



CARGAR DATA

```
#Get and save data
```


```
mnist = tf.keras.datasets.mnist
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

DATA

Total de 70k imágenes en el conjunto de datos, 60k de entrenamiento y 10k de pruebas.


Los dos 28 indican que cada imagen tiene 28 por 28 píxeles y las imágenes se representan como matrices de 28x28 llenas de valores de píxeles

✓ 0 s  #See Data
x_train.shape

📄 (60000, 28, 28)

✓ 0 s [13] y_train.shape
(60000,)

✓ 0 s [14] x_test.shape
(10000, 28, 28)

✓ 0 s  y_test.shape
(10000,)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51	159	253	159	50	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	238	252	252	252	237	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	54	227	253	252	239	233	252	57	6	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	10	60	224	252	253	252	202	84	252	253	122	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	163	252	252	252	253	252	252	96	189	253	167	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	51	238	253	253	190	114	253	228	47	79	255	168	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	48	238	252	252	179	12	75	121	21	0	0	253	243	50	0	0	0	0	0	0
11	0	0	0	0	0	0	0	38	165	253	233	208	84	0	0	0	0	0	0	253	252	165	0	0	0	0	0	0
12	0	0	0	0	0	0	0	7	178	252	240	71	19	28	0	0	0	0	0	0	253	252	195	0	0	0	0	0
13	0	0	0	0	0	0	0	57	252	252	63	0	0	0	0	0	0	0	0	0	253	252	195	0	0	0	0	0
14	0	0	0	0	0	0	0	198	253	190	0	0	0	0	0	0	0	0	0	0	255	253	196	0	0	0	0	0
15	0	0	0	0	0	0	76	246	252	112	0	0	0	0	0	0	0	0	0	0	253	252	148	0	0	0	0	0
16	0	0	0	0	0	0	85	252	230	25	0	0	0	0	0	0	0	0	7	135	253	186	12	0	0	0	0	0
17	0	0	0	0	0	0	85	252	223	0	0	0	0	0	0	0	0	7	131	252	225	71	0	0	0	0	0	0
18	0	0	0	0	0	0	85	252	145	0	0	0	0	0	0	48	165	252	173	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	86	253	225	0	0	0	0	0	0	114	238	253	162	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	85	252	249	146	48	29	85	178	225	253	223	167	56	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	85	252	252	252	229	215	252	252	252	196	130	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	28	199	252	252	253	252	252	233	145	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	25	128	252	253	252	141	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



CONSTRUIR MODELO



0s

```
[5] #Build Model
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



SEQUENTIAL

Lo que hace esta función es comenzar la creación de una disposición lineal (o "secuencial") de capas.

```
] #BUILD MODEL  
model = tf.keras.models.Sequential([
```

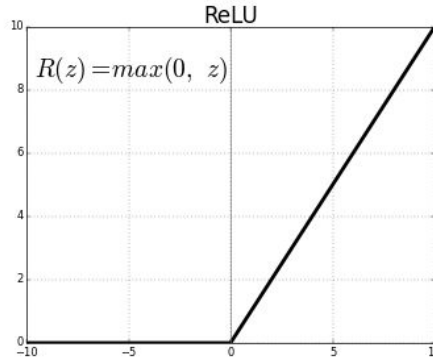


FLATTEN

Crea la primera capa en nuestra red . Cada imagen tiene $28 * 28 = 784$, y por lo tanto, crea una capa con 784 nodos

```
tf.keras.layers.Flatten(input_shape=(28, 28)),
```

DENSE



Crea lo que se llama una capa completamente conectada. El primer parámetro (128) especifica cuántos nodos deben estar en la capa. Activación 'relu'

```
tf.keras.layers.Dense(128, activation='relu'),
```



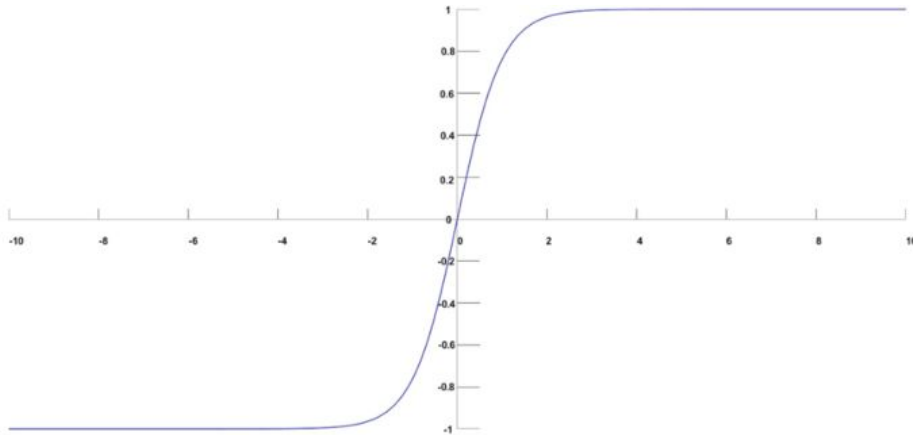
DROPOT

Hace que algunos de los nodos de una capa determinada no pasen su información a la siguiente. Hay un 0.2 de probabilidad de cada nodo ser eliminado del cálculo

```
tf.keras.layers.Dropout(0.2),
```



DENSE



Se crea la capa final (Debe ser de 10 nodos ya que el modelo intenta predecir entre 10 números). Función de activación SoftMax,

```
tf.keras.layers.Dense(10, activation='softmax')
```



COMPILADOR

Optimizador: Forma de hacer que el proceso de retropropagación sea más rápido y más efectivo

Pérdida: Cuantifica qué tan lejos está una predicción de la respuesta correcta (Descarta)

Métrica: Especifica las métricas que debe usar para evaluar el modelo

```
#Compile Model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```



ENTRENAMIENTO Y RESULTADO

```
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [=====] - 4s 2ms/step - loss: 0.2956 - accuracy: 0.9142
Epoch 2/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.1412 - accuracy: 0.9578
Epoch 3/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.1047 - accuracy: 0.9681
Epoch 4/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.0850 - accuracy: 0.9734
Epoch 5/5
1875/1875 [=====] - 3s 2ms/step - loss: 0.0724 - accuracy: 0.9778
<keras.callbacks.History at 0x7f6a323dc890>
```

```
#Evaluate Model
```

```
score = model.evaluate(x_test, y_test, verbose=2)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])
```

```
313/313 - 0s - loss: 0.0719 - accuracy: 0.9778
```

```
Test score: 0.07188113778829575
```

```
Test accuracy: 0.9778000116348267
```




Perceptrón y Backpropagation

Paper - Sebastián Mora y Federico Idarraga Cardenas

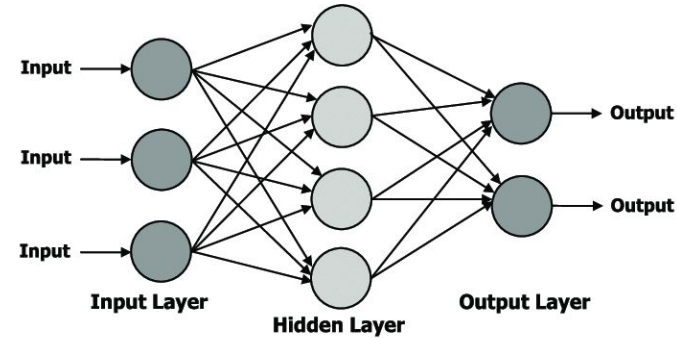


Paper

Realizamos un resumen de los principales contenidos de el perceptrón y backpropagation aquí pasaremos por su explicación, veremos las partes que los componen explicando cada una de ellas y viendo como trabaja la función de activación, la suma ponderada y el entrenamiento en general de tanto el perceptrón y el backpropagation.

Perceptrón

Un perceptrón es un clasificador lineal (binario). Además, se utiliza en aprendizaje supervisado. Ayuda a clasificar los datos de entrada dados.



Backpropagation

Conforme el uso de las redes neuronales se ha ido extendiendo a diferentes campos de la ciencia y la ingeniería la confianza en los modelos se hace fundamental. El factor más importante para lograr un modelo confiable es el entrenamiento.

