mnist original

September 21, 2023

[46]:
```
import tensorflow as tf
import keras as keras
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation
from tensorflow.keras.optimizers import RMSprop, SGD
```

[20]:
```
learning_rate = 0.001
epochs = 30
batch_size = 120
```

[11]:
```
from tensorflow.keras.datasets import mnist#cargar los datos desde internet
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

[12]:
```
X_train.shape#dimensión de los datos
```

[12]:
```
(60000, 28, 28)
```

[22]:
```
x_trainv = X_train.reshape(60000, 784)#redimensionar la matriz de datos
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')#tipo de dato de salida para que no se vaya␣
↪a cerlo

x_trainv /= 255  # x_trainv = x_trainv/255
x_testv /= 255
```

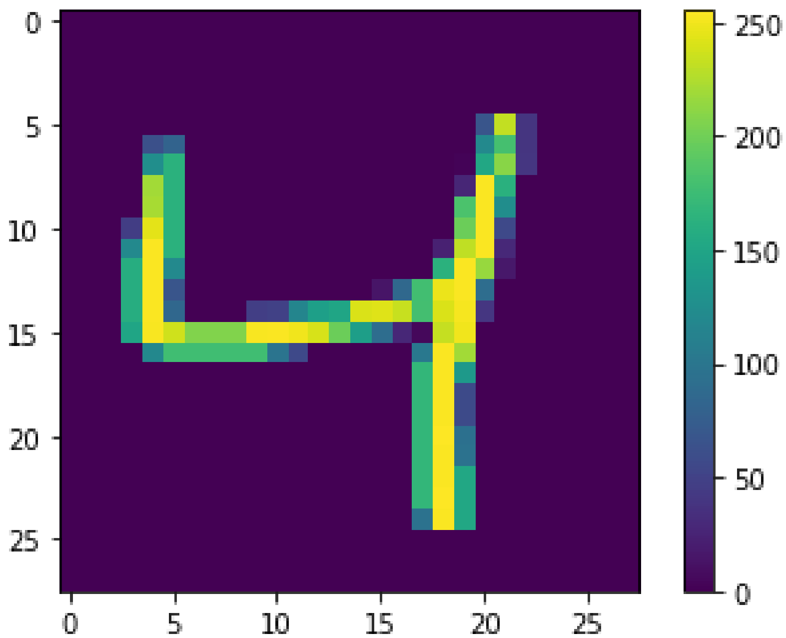
[24]:
```
print(Y_train[10000])
```

```
3
```

[25]:
```
num_classes=10
```

```
y_trainc = keras.utils.to_categorical(Y_train, num_classes)#devuelve una matriz␣
↪de valores binarios no. de filas igual a la longitud del vector de entrada y␣
↪un número de columnas igual al número de clases.
y_testc = keras.utils.to_categorical(Y_test, num_classes)
```

[14]:
```
plt.figure()
plt.imshow(X_train[2])#número de imagen en el mnist
plt.colorbar()
plt.grid(False)
plt.show()
```



[15]:
```
#otra forma de pre-procesamiento
train_images = X_train / 255.0#escalara los valores

test_images = Y_train / 255.0
```

[37]:
```
model = Sequential()##modelo
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))##capa de entrada
model.add(Dense(num_classes, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 dense_4 (Dense)             (None, 512)               401920

 dense_5 (Dense)             (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

[38]:
```
#model.compile(optimizer='adam',
           #    loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
```

[47]:
```
model.
 ↪compile(loss='categorical_crossentropy',optimizer=SGD(learning_rate=learning_rate),metrics=
##funcion de perdida,optimizador,taza de aprendizaje,métrica
```

[48]:
```
history = model.fit(x_trainv, y_trainc,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_testv, y_testc)
                    )
```

```
Epoch 1/30
500/500 [==============================] - 2s 4ms/step - loss: 2.2789 -
accuracy: 0.1781 - val_loss: 2.2231 - val_accuracy: 0.3118
Epoch 2/30
500/500 [==============================] - 2s 3ms/step - loss: 2.1839 -
accuracy: 0.3973 - val_loss: 2.1411 - val_accuracy: 0.4883
Epoch 3/30
500/500 [==============================] - 2s 4ms/step - loss: 2.1050 -
accuracy: 0.5251 - val_loss: 2.0624 - val_accuracy: 0.5741
Epoch 4/30
500/500 [==============================] - 2s 3ms/step - loss: 2.0291 -
accuracy: 0.5928 - val_loss: 1.9861 - val_accuracy: 0.6454
Epoch 5/30
500/500 [==============================] - 2s 3ms/step - loss: 1.9558 -
accuracy: 0.6388 - val_loss: 1.9127 - val_accuracy: 0.6686
Epoch 6/30
500/500 [==============================] - 2s 4ms/step - loss: 1.8850 -
accuracy: 0.6639 - val_loss: 1.8422 - val_accuracy: 0.6918
Epoch 7/30
500/500 [==============================] - 2s 4ms/step - loss: 1.8168 -
accuracy: 0.6848 - val_loss: 1.7736 - val_accuracy: 0.7032
Epoch 8/30
500/500 [==============================] - 2s 4ms/step - loss: 1.7508 -
accuracy: 0.7006 - val_loss: 1.7079 - val_accuracy: 0.7154
```

```
Epoch 9/30
500/500 [==============================] - 2s 3ms/step - loss: 1.6871 -
accuracy: 0.7125 - val_loss: 1.6447 - val_accuracy: 0.7430
Epoch 10/30
500/500 [==============================] - 2s 3ms/step - loss: 1.6261 -
accuracy: 0.7298 - val_loss: 1.5834 - val_accuracy: 0.7410
Epoch 11/30
500/500 [==============================] - 2s 3ms/step - loss: 1.5675 -
accuracy: 0.7381 - val_loss: 1.5253 - val_accuracy: 0.7530
Epoch 12/30
500/500 [==============================] - 2s 3ms/step - loss: 1.5114 -
accuracy: 0.7480 - val_loss: 1.4694 - val_accuracy: 0.7666
Epoch 13/30
500/500 [==============================] - 2s 4ms/step - loss: 1.4580 -
accuracy: 0.7584 - val_loss: 1.4166 - val_accuracy: 0.7748
Epoch 14/30
500/500 [==============================] - 2s 3ms/step - loss: 1.4069 -
accuracy: 0.7663 - val_loss: 1.3662 - val_accuracy: 0.7860
Epoch 15/30
500/500 [==============================] - 2s 4ms/step - loss: 1.3586 -
accuracy: 0.7749 - val_loss: 1.3182 - val_accuracy: 0.7901
Epoch 16/30
500/500 [==============================] - 2s 3ms/step - loss: 1.3127 -
accuracy: 0.7812 - val_loss: 1.2733 - val_accuracy: 0.7970
Epoch 17/30
500/500 [==============================] - 2s 3ms/step - loss: 1.2693 -
accuracy: 0.7873 - val_loss: 1.2305 - val_accuracy: 0.8026
Epoch 18/30
500/500 [==============================] - 2s 3ms/step - loss: 1.2284 -
accuracy: 0.7938 - val_loss: 1.1902 - val_accuracy: 0.8078
Epoch 19/30
500/500 [==============================] - 2s 3ms/step - loss: 1.1896 -
accuracy: 0.7994 - val_loss: 1.1522 - val_accuracy: 0.8091
Epoch 20/30
500/500 [==============================] - 2s 3ms/step - loss: 1.1532 -
accuracy: 0.8033 - val_loss: 1.1165 - val_accuracy: 0.8148
Epoch 21/30
500/500 [==============================] - 2s 3ms/step - loss: 1.1189 -
accuracy: 0.8070 - val_loss: 1.0830 - val_accuracy: 0.8183
Epoch 22/30
500/500 [==============================] - 2s 3ms/step - loss: 1.0867 -
accuracy: 0.8117 - val_loss: 1.0515 - val_accuracy: 0.8183
Epoch 23/30
500/500 [==============================] - 2s 3ms/step - loss: 1.0563 -
accuracy: 0.8151 - val_loss: 1.0216 - val_accuracy: 0.8226
Epoch 24/30
500/500 [==============================] - 2s 3ms/step - loss: 1.0277 -
accuracy: 0.8186 - val_loss: 0.9941 - val_accuracy: 0.8221
```

```
Epoch 25/30
500/500 [==============================] - 2s 3ms/step - loss: 1.0008 -
accuracy: 0.8205 - val_loss: 0.9673 - val_accuracy: 0.8280
Epoch 26/30
500/500 [==============================] - 2s 3ms/step - loss: 0.9755 -
accuracy: 0.8232 - val_loss: 0.9427 - val_accuracy: 0.8324
Epoch 27/30
500/500 [==============================] - 2s 3ms/step - loss: 0.9516 -
accuracy: 0.8260 - val_loss: 0.9195 - val_accuracy: 0.8343
Epoch 28/30
500/500 [==============================] - 2s 3ms/step - loss: 0.9292 -
accuracy: 0.8285 - val_loss: 0.8975 - val_accuracy: 0.8360
Epoch 29/30
500/500 [==============================] - 2s 3ms/step - loss: 0.9080 -
accuracy: 0.8303 - val_loss: 0.8768 - val_accuracy: 0.8393
Epoch 30/30
500/500 [==============================] - 2s 3ms/step - loss: 0.8880 -
accuracy: 0.8322 - val_loss: 0.8574 - val_accuracy: 0.8407
```

[49]:

```
score = model.evaluate(x_testv, y_testc, verbose=1) #evaluar la eficiencia del␣
 ↪modelo
print(score)
a=model.predict(x_testv) #predicción de la red entrenada
print(a.shape)
print(a[1])
print("resultado correcto:")
print(y_testc[1])
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.8574 -
accuracy: 0.8407
[0.8574431538581848, 0.8406999707221985]
313/313 [==============================] - 0s 1ms/step
(10000, 10)
[0.5683311  0.5392339  0.88698    0.7709753  0.10433239 0.69017476
 0.79657227 0.11404323 0.5960265  0.10767218]
resultado correcto:
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

[ ]: