exp1b-DROPOUT

September 22, 2023

[16]:
```
#para este experimento se utilizó la función de activación relu,no. de␣
 ↪clases,NADAM
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,Adam,Nadam
from tensorflow.keras import regularizers
```

[17]:
```
learning_rate = 0.001
epochs = 20
batch_size = 120
```

[18]:
```
from tensorflow.keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

[19]:
```
X_train.shape
```

[19]:
```
(60000, 28, 28)
```

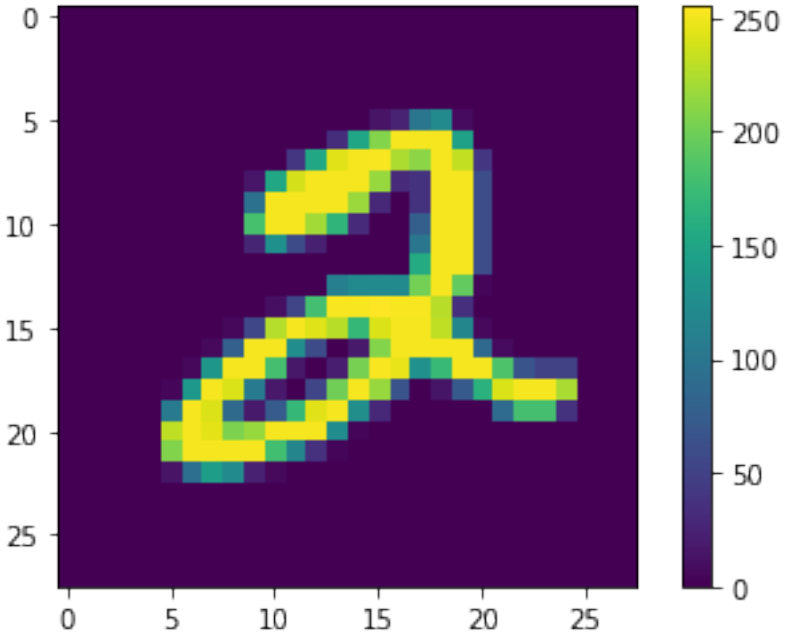
[20]:
```
x_trainv = X_train.reshape(60000, 784)
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')

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

[21]:
```
print(Y_train[10000])
```

```
3
```

[22]:
```
num_classes=10
y_trainc = keras.utils.to_categorical(Y_train, num_classes)
y_testc = keras.utils.to_categorical(Y_test, num_classes)
```

[23]:
```
plt.figure()
plt.imshow(X_train[5])#número de imagen en el mnist
plt.colorbar()
plt.grid(False)
plt.show()
```



[10]:
```
#pre-procesamiento
train_images = X_train / 255.0#escalara los valores

test_images = Y_train / 255.0
```

[26]:
```
model = Sequential()
model.add(Dense(512, activation='relu',␣
 ↪input_shape=(784,),kernel_regularizer=regularizers.L1L2(0.01)))
tf.keras.layers.Dropout(0.2)
model.add(Dense(num_classes, activation='sigmoid'))

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 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
_________________________________________________________________
```

[27]:
```
#model.compile(optimizer='adam',
           #   loss='sparse_categorical_crossentropy',  metrics=['accuracy'])
```

[28]:
```
model.
 ↪compile(loss='categorical_crossentropy',optimizer=Nadam(learning_rate=learning_rate),metric
```

[29]:
```
history = model.fit(x_trainv, y_trainc,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_testv, y_testc)
                    )
```

```
Epoch 1/20
500/500 [==============================] - 3s 5ms/step - loss: 9.3617 -
accuracy: 0.7538 - val_loss: 1.5804 - val_accuracy: 0.8342
Epoch 2/20
500/500 [==============================] - 2s 5ms/step - loss: 1.4745 -
accuracy: 0.8423 - val_loss: 1.3622 - val_accuracy: 0.8549
Epoch 3/20
500/500 [==============================] - 2s 4ms/step - loss: 1.3349 -
accuracy: 0.8532 - val_loss: 1.2748 - val_accuracy: 0.8651
Epoch 4/20
500/500 [==============================] - 2s 4ms/step - loss: 1.2630 -
accuracy: 0.8611 - val_loss: 1.2145 - val_accuracy: 0.8701
Epoch 5/20
500/500 [==============================] - 2s 4ms/step - loss: 1.2157 -
accuracy: 0.8668 - val_loss: 1.1666 - val_accuracy: 0.8753
Epoch 6/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1799 -
accuracy: 0.8704 - val_loss: 1.1449 - val_accuracy: 0.8789
Epoch 7/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1527 -
accuracy: 0.8733 - val_loss: 1.1159 - val_accuracy: 0.8838
Epoch 8/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1288 -
accuracy: 0.8765 - val_loss: 1.0981 - val_accuracy: 0.8794
Epoch 9/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1101 -
accuracy: 0.8788 - val_loss: 1.1032 - val_accuracy: 0.8779
Epoch 10/20
```

```
500/500 [==============================] - 2s 4ms/step - loss: 1.0960 -
accuracy: 0.8807 - val_loss: 1.0759 - val_accuracy: 0.8847
Epoch 11/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0818 -
accuracy: 0.8820 - val_loss: 1.0498 - val_accuracy: 0.8901
Epoch 12/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0686 -
accuracy: 0.8839 - val_loss: 1.0397 - val_accuracy: 0.8906
Epoch 13/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0583 -
accuracy: 0.8855 - val_loss: 1.0290 - val_accuracy: 0.8933
Epoch 14/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0498 -
accuracy: 0.8863 - val_loss: 1.0248 - val_accuracy: 0.8949
Epoch 15/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0413 -
accuracy: 0.8879 - val_loss: 1.0154 - val_accuracy: 0.8932
Epoch 16/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0329 -
accuracy: 0.8884 - val_loss: 1.0083 - val_accuracy: 0.8970
Epoch 17/20
500/500 [==============================] - 2s 5ms/step - loss: 1.0254 -
accuracy: 0.8909 - val_loss: 1.0032 - val_accuracy: 0.8941
Epoch 18/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0168 -
accuracy: 0.8906 - val_loss: 0.9855 - val_accuracy: 0.8966
Epoch 19/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0112 -
accuracy: 0.8910 - val_loss: 0.9894 - val_accuracy: 0.8970
Epoch 20/20
500/500 [==============================] - 2s 5ms/step - loss: 1.0055 -
accuracy: 0.8926 - val_loss: 0.9801 - val_accuracy: 0.9002
```

[30]:
```
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: 0.9801 -
accuracy: 0.9002
[0.9800727367401123, 0.9002000093460083]
313/313 [==============================] - 0s 1ms/step
(10000, 10)
[5.9321892e-01 1.2692228e-02 9.4624716e-01 3.4844494e-01 7.1239488e-06
```

```
 2.5475672e-01 5.8134240e-01 1.3372081e-05 8.2872801e-02 2.7725277e-05]
resultado correcto:
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

[ ]: