In [2]:
```
#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
```

In [3]:
```
learning_rate = 0.001
epochs = 20
batch_size = 120
```

In [4]:
```
from tensorflow.keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

In [5]:
```
X_train.shape
```

Out[5]:
```
(60000, 28, 28)
```

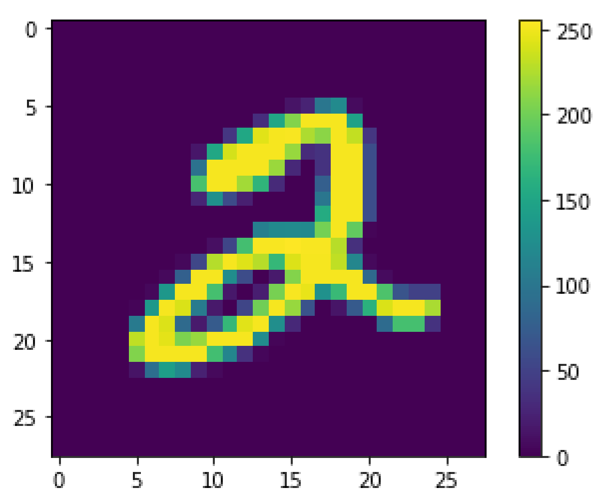
In [6]:
```
x_trainv = X_train.reshape(60000, 784)
x_testv = X_test.reshape(10000, 784)
x_trainv = x_trainv.astype('float32')
x_testv = x_testv.astype('float32')

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

In [7]:
```
print(Y_train[10000])
```

```
3
```

In [8]:
```
num_classes=10
y_trainc = keras.utils.to_categorical(Y_train, num_classes)
y_testc = keras.utils.to_categorical(Y_test, num_classes)
```

In [9]:
```
plt.figure()
plt.imshow(X_train[5])#número de imagen en el mnist
plt.colorbar()
plt.grid(False)
plt.show()
```



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

```python
test_images = Y_train / 255.0
```

In [11]:

```python
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,),kernel_regularizer=regularize
model.add(Dense(num_classes, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               401920

 dense_1 (Dense)             (None, 10)                5130

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

In [12]:

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

In [13]:

```python
model.compile(loss='categorical_crossentropy',optimizer=Nadam(learning_rate=learning_rat
```

In [14]:

```python
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: 9.3836 - accuracy: 0.7510
- val_loss: 1.5697 - val_accuracy: 0.8326
Epoch 2/20
500/500 [==============================] - 2s 4ms/step - loss: 1.4702 - accuracy: 0.8391
- val_loss: 1.3611 - val_accuracy: 0.8585
Epoch 3/20
500/500 [==============================] - 2s 5ms/step - loss: 1.3335 - accuracy: 0.8525
- val_loss: 1.2674 - val_accuracy: 0.8670
Epoch 4/20
500/500 [==============================] - 2s 4ms/step - loss: 1.2643 - accuracy: 0.8598
- val_loss: 1.2173 - val_accuracy: 0.8733
Epoch 5/20
500/500 [==============================] - 2s 4ms/step - loss: 1.2189 - accuracy: 0.8645
- val_loss: 1.1831 - val_accuracy: 0.8704
Epoch 6/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1868 - accuracy: 0.8681
- val_loss: 1.1499 - val_accuracy: 0.8777
Epoch 7/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1629 - accuracy: 0.8704
- val_loss: 1.1299 - val_accuracy: 0.8789
Epoch 8/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1404 - accuracy: 0.8730
- val_loss: 1.1020 - val_accuracy: 0.8847
Epoch 9/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1233 - accuracy: 0.8745
- val_loss: 1.0878 - val_accuracy: 0.8833
Epoch 10/20
500/500 [==============================] - 2s 4ms/step - loss: 1.1097 - accuracy: 0.8771
- val_loss: 1.0773 - val_accuracy: 0.8850
```

```
Epoch 11/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0955 - accuracy: 0.8787
- val_loss: 1.0716 - val_accuracy: 0.8855
Epoch 12/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0846 - accuracy: 0.8796
- val_loss: 1.0595 - val_accuracy: 0.8849
Epoch 13/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0742 - accuracy: 0.8805
- val_loss: 1.0436 - val_accuracy: 0.8906
Epoch 14/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0650 - accuracy: 0.8826
- val_loss: 1.0312 - val_accuracy: 0.8868
Epoch 15/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0571 - accuracy: 0.8830
- val_loss: 1.0192 - val_accuracy: 0.8897
Epoch 16/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0480 - accuracy: 0.8845
- val_loss: 1.0215 - val_accuracy: 0.8947
Epoch 17/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0412 - accuracy: 0.8844
- val_loss: 1.0145 - val_accuracy: 0.8924
Epoch 18/20
500/500 [==============================] - 2s 5ms/step - loss: 1.0356 - accuracy: 0.8859
- val_loss: 1.0068 - val_accuracy: 0.8923
Epoch 19/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0294 - accuracy: 0.8867
- val_loss: 1.0037 - val_accuracy: 0.8912
Epoch 20/20
500/500 [==============================] - 2s 4ms/step - loss: 1.0236 - accuracy: 0.8867
- val_loss: 0.9925 - val_accuracy: 0.8941
```

In [15]:

```python
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: 0.9925 - accuracy: 0.8941
[0.9925473928451538, 0.89410001039505]
313/313 [==============================] - 0s 1ms/step
(10000, 10)
[7.6535463e-01 1.1963156e-02 9.2632908e-01 4.3228441e-01 3.9403640e-05
 5.3911662e-01 6.8487692e-01 1.9600082e-05 1.5796481e-01 1.3078690e-04]
resultado correcto:
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

In [ ]: