Análisis de Datos y Aprendizaje Máquina con Tensorflow 2.0: Perceptrón Multicapa

2019/09/30

Actividad Perceptron Multicapa

- Objetivo: Crear un modelo para obtener un 97% de Test accuracy con un máximo de 2 capas ocultas sin usar regularización l2, en no más de 25 épocas.

  Tiempo máximo: 1 hora

- Nota: Utilizar las técnicas de reguarización y optimización

In [1]:
```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, BatchNormalization
from tensorflow.keras import backend as K
K.clear_session()


mnist = keras.datasets.mnist

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

In [2]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

- Se modifica la forma de los datos de 2-d (n, 28, 28) a 1-d (n, 784)

In [3]:
```
x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)


print(x_train.shape) # (60000, 784)
print(y_train.shape) # (60000,)
print(x_test.shape)  # (10000, 784)
print(y_test.shape)  # (10000,)
```

```
(60000, 784)
(60000,)
(10000, 784)
(10000,)
```

Leer Dataset

In [4]:
```
epoch = 21
verbose = 0
batch = 50
```

In [5]:
```
K.clear_session()
```

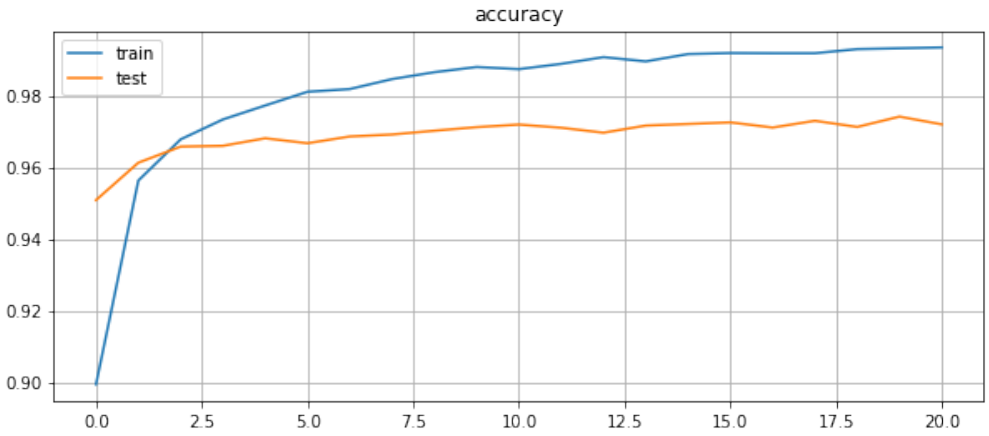
In [6]:
```
def make_model():
    model = Sequential()

    model.add(Dense(60, input_shape = (784, )))
    model.add(BatchNormalization(momentum=0.99))
    model.add(Activation('relu'))
    model.add(Dense(60))
    model.add(BatchNormalization(momentum=0.99))
    model.add(Activation('relu'))
    model.add(Dense(10))
    model.add(Activation('softmax'))

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

    return model

# Fit
model = make_model()

model.summary()
# lista de datos
history = model.fit(x_train, y_train, batch_size = batch, validation_split = 0.3,
                    epochs = epoch, verbose = 1)

test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

print('\nTest acccuracy:', test_acc)
```

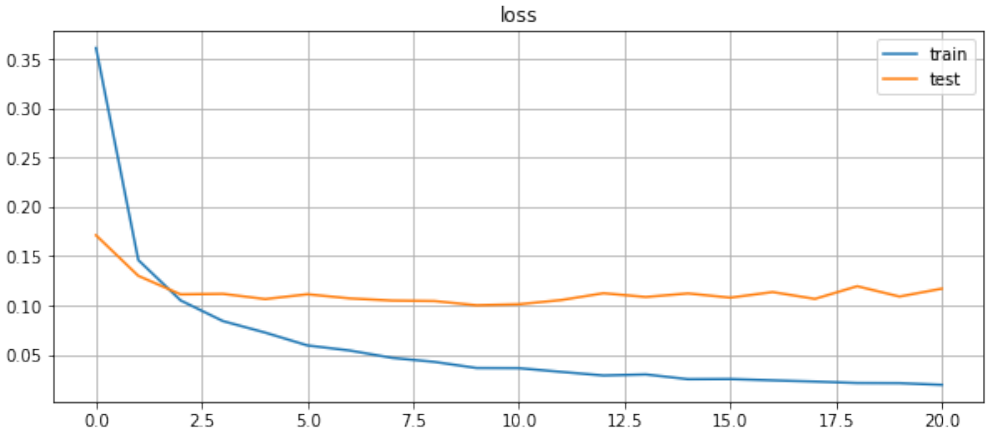
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 60)                47100
_________________________________________________________________
batch_normalization (BatchNo (None, 60)                240
_________________________________________________________________
activation (Activation)      (None, 60)                0
_________________________________________________________________
dense_1 (Dense)              (None, 60)                3660
_________________________________________________________________
batch_normalization_1 (Batch (None, 60)                240
_________________________________________________________________
activation_1 (Activation)    (None, 60)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                610
_________________________________________________________________
activation_2 (Activation)    (None, 10)                0
=================================================================
Total params: 51,850
Trainable params: 51,610
Non-trainable params: 240
_________________________________________________________________
Train on 42000 samples, validate on 18000 samples
Epoch 1/21
42000/42000 [==============================] - 4s 87us/sample - loss: 0.3610 - accuracy: 0.8994 -
Epoch 2/21
42000/42000 [==============================] - 3s 68us/sample - loss: 0.1459 - accuracy: 0.9563 -
Epoch 3/21
42000/42000 [==============================] - 3s 67us/sample - loss: 0.1050 - accuracy: 0.9679 -
Epoch 4/21
42000/42000 [==============================] - 3s 68us/sample - loss: 0.0841 - accuracy: 0.9734 -
Epoch 5/21
42000/42000 [==============================] - 3s 65us/sample - loss: 0.0724 - accuracy: 0.9773 -
Epoch 6/21
42000/42000 [==============================] - 3s 63us/sample - loss: 0.0594 - accuracy: 0.9812 -
Epoch 7/21
42000/42000 [==============================] - 3s 67us/sample - loss: 0.0542 - accuracy: 0.9819 -
Epoch 8/21
42000/42000 [==============================] - 3s 65us/sample - loss: 0.0468 - accuracy: 0.9847 -
Epoch 9/21
42000/42000 [==============================] - 3s 62us/sample - loss: 0.0427 - accuracy: 0.9866 -
Epoch 10/21
42000/42000 [==============================] - 3s 64us/sample - loss: 0.0365 - accuracy: 0.9881 -
Epoch 11/21
42000/42000 [==============================] - 3s 63us/sample - loss: 0.0363 - accuracy: 0.9875 -
Epoch 12/21
42000/42000 [==============================] - 3s 64us/sample - loss: 0.0325 - accuracy: 0.9890 -
```

```
Epoch 13/21
42000/42000 [==============================] - 3s 65us/sample - loss: 0.0289 - accuracy: 0.9909 -
Epoch 14/21
42000/42000 [==============================] - 3s 64us/sample - loss: 0.0300 - accuracy: 0.9897 -
Epoch 15/21
42000/42000 [==============================] - 3s 64us/sample - loss: 0.0251 - accuracy: 0.9917 -
Epoch 16/21
42000/42000 [==============================] - 3s 62us/sample - loss: 0.0253 - accuracy: 0.9920 -
Epoch 17/21
42000/42000 [==============================] - 3s 63us/sample - loss: 0.0240 - accuracy: 0.9920 -
Epoch 18/21
42000/42000 [==============================] - 3s 64us/sample - loss: 0.0227 - accuracy: 0.9920 -
Epoch 19/21
42000/42000 [==============================] - 3s 68us/sample - loss: 0.0213 - accuracy: 0.9931 -
Epoch 20/21
42000/42000 [==============================] - 3s 67us/sample - loss: 0.0211 - accuracy: 0.9933 -
Epoch 21/21
42000/42000 [==============================] - 3s 63us/sample - loss: 0.0194 - accuracy: 0.9936 -
10000/1 - 1s - loss: 0.0542 - accuracy: 0.9755

Test acccuracy: 0.9755
```

```
In [7]: test_loss, test_acc = model.evaluate(x_test,  y_test, verbose=2)

10000/1 - 0s - loss: 0.0542 - accuracy: 0.9755
```

```
In [8]: # plot
        plt.figure(figsize=(10,9))

        plt.subplot(211)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])

        plt.title('accuracy')
        plt.legend(['train', 'test'])
        plt.grid()
        plt.subplot(212)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('loss')
        plt.legend(['train', 'test'])
        plt.grid()



        plt.show()
```


accuracy
train
test
0.98
0.96
0.94
0.92
0.90
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


loss
train
test
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0