TAREA 4

Thursday, February 20, 2020 3:07 PM

Javier Valencia Goujon 123227
Alfie Sergio González Salcedo 181574

```
La celda siguiente contiene dos modelos de redes neuronales. Comenta uno y usa otro.
# Evalua uno a la vez y reporta tus resultados.
#
# Nota que importamos layers tipo BatchNorm y Dropout, los cuales son usados en
# distintas secciones del modelo.
# También importamos regularizadores l1 y l2, los cuales son usados como parametros
# de las capas.
#
# Nota también el uso del parametro "mertics" en la compilación del modelo.
# Este parámetro es sólo informativo, pero no se usa para optimizar el modelo.
# En el caso particular de este modelo, se indica la exactitud de la clasificación,
# sin embargo la minimización del error se hace con la función de pérdida "categorical_crossentropy"
Puedes revisar más detalles en la documentación de Keras o tf.
```

Q1: Según los resultados que hayas obtenido, ¿cuál de los dos modelos es preferible y por qué?

Es preferible el modelo con regularizadores porque agiliza el proceso de optimización. De igual forma nos resultó más facil llegar al threshold de los valores de exactitud y error de entrenamiento y validación

Q2: ¿Por qué usamos softmax en la salida de la red?
Porque se está haciendo clasificación multi-clase, y queremos maximizar la probabilidad de salida de la clase mas probable.

Q3: Ajusta el primer modelo (sin regularizadores) para obtener una pérdida de "entrenamiento"
```
#     menor o igual a 0.08 y exactitud mayor o igual a 98%.
#     Reporta el número de capas y sus tamaños.
```

Modelo sin regularizacion
```
# -- Try 01 --
#DNN.add(InputLayer(input_shape=x_train.shape[1:]))
#DNN.add(Dense(40, activation='relu'))
#DNN.add(Dense(50, activation='relu'))
#DNN.add(Dense(15, activation='relu'))
# -- ------ --
```

Modelo regularizado
```
# -- Try 02 --
DNN.add(InputLayer(input_shape=x_train.shape[1:]))
DNN.add(Dropout(rate=0.20))
DNN.add(Dense(40, activation='relu'))
DNN.add(BatchNormalization())
DNN.add(Activation('relu'))
DNN.add(Dense(50, activation='relu'))
DNN.add(Dense(15, activation='relu', activity_regularizer=l1(l=70e-15)))
#DNN.add(Dense(15, activation='relu', activity_regularizer=l2(l=3e-15)))
# -- ------ --
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout_1 (Dropout)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 40)                31400
_________________________________________________________________
batch_normalization_1 (Batch (None, 40)                160
_________________________________________________________________
activation_1 (Activation)    (None, 40)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                2050
_________________________________________________________________
dense_3 (Dense)              (None, 15)                765
_________________________________________________________________
dense_4 (Dense)              (None, 10)                160
=================================================================
Total params: 34,535
Trainable params: 34,455
Non-trainable params: 80
_________________________________________________________________
time: 177 ms
```

Q4: Ahora usa esos mismos valores de hiperparámetros (número de capas y sus tamaños)
```
#     en el segundo modelo, y ajusta la taza de dropout, y las alfas en los regularizadores l1 y l2
#     para disminuir el error de generalización (validación).
#     Reporta el modelo regularizado que te haya dado mejores resultados.
```

(dropout = 0.10),25,45,25 (l1 = 3e-15)

Q5: Partiendo del mejor modelo que hayas obtenido anteriormente, modifica
```
#     el número de sus capas y tamaños para disminuir aún más los errores, tanto
#     el de entrenamiento como el de validación.
#     Reporta tu mejor modelo.
```
Train on 54000 samples, validate on 6000 samples
```
Epoch 23/50
```

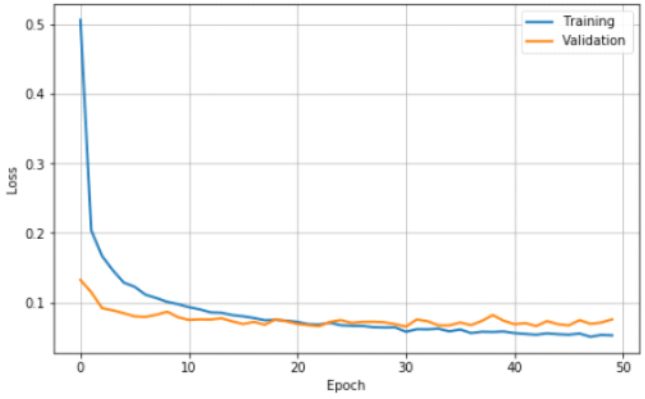
```
Epoch 23/50
54000/54000 [==============================] - 6s 112us/step - loss: 0.0681 - acc: 0.9777 - val_loss: 0.0658 - val_acc: 0.9822
Epoch 24/50
54000/54000 [==============================] - 5s 102us/step - loss: 0.0706 - acc: 0.9770 - val_loss: 0.0720 - val_acc: 0.9807
Epoch 25/50
54000/54000 [==============================] - 6s 107us/step - loss: 0.0671 - acc: 0.9784 - val_loss: 0.0745 - val_acc: 0.9780
Epoch 26/50
54000/54000 [==============================] - 6s 108us/step - loss: 0.0663 - acc: 0.9783 - val_loss: 0.0705 - val_acc: 0.9798
Epoch 27/50
54000/54000 [==============================] - 6s 109us/step - loss: 0.0661 - acc: 0.9785 - val_loss: 0.0719 - val_acc: 0.9817
Epoch 28/50
```

Test Error

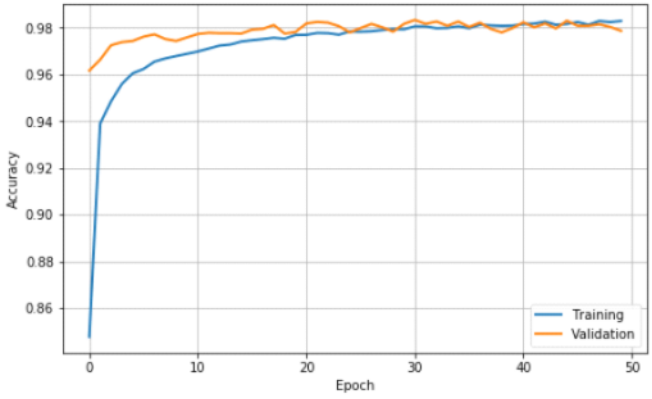
```
# Compute test loss
test_loss, test_acc = DNN.evaluate(x=x_test, y=y_test, verbose=False)
print("Test loss:", test_loss)
print("Test acc:", test_acc)
```

```
Test loss: 0.07996194914911176
Test acc: 0.9752
time: 636 ms
```

Grafica comparativa de Error de entrenamiento vs Error de validacion en el tiempo

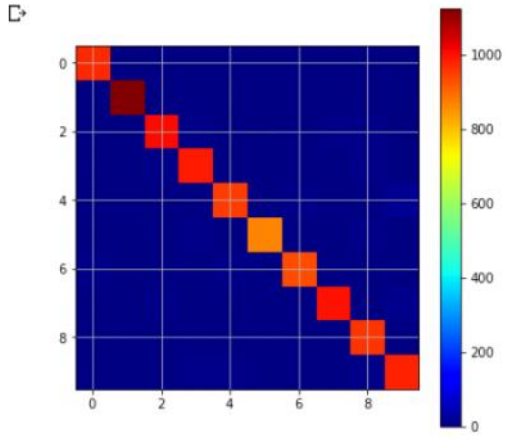


Grafica comparativa de Accuracy en el tiempo




```
time: 360 ms
```

```
[27] # Confussion Matrix
     from sklearn.metrics import confusion_matrix
     Conf_Mat = confusion_matrix(y_test_cat, y_test_hat_cat)
     plt.figure(figsize=(6, 6))
     plt.imshow(Conf_Mat, cmap='jet')
     plt.grid()
     plt.colorbar()
     plt.show()
```



```
time: 519 ms
```