CrashCoursesDL_FNN_CNN

January 10, 2024

#

Deep Learning Crash Course with tensorflow

Abdulrahman ALLOUCHE / 9 January 2024

Tensorflow is a free software Deep Learning library, developped by Google for the Python programming language.

https://www.tensorflow.org/

0.1 Set up TensorFlow

[57]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import warnings
warnings.filterwarnings("ignore")
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.9.0
```

0.2 1.Suppervised DL using a Full Neural Networks

0.2.1 Load a dataset

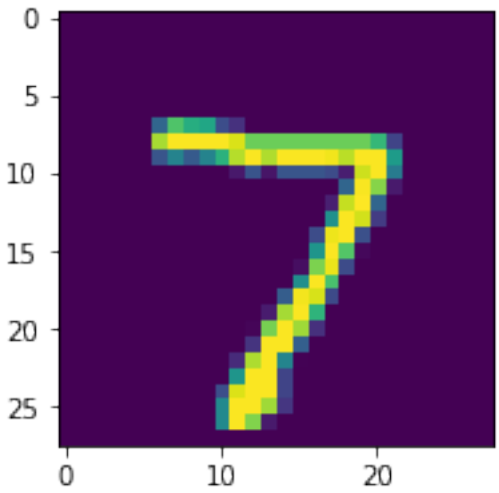
[58]:
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

[59]:
```
print("xtrain shape =", x_train.shape)
```

```
xtrain shape = (60000, 28, 28)
```

[60]:
```
plt.figure(figsize=(3,3))
plt.imshow(x_test[0])
```

[60]:
```
<matplotlib.image.AxesImage at 0x7f2618738220>
```



0.2.2 Build a model

[61]:
```
warnings.filterwarnings("ignore")
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 784)               0

 dense_8 (Dense)             (None, 128)               100480

 dropout_3 (Dropout)         (None, 128)               0

 dense_9 (Dense)             (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

[62]:
```
#tf.keras.utils.plot_model(model, show_shapes=True) #need graphviz & pydot
```

[63]:
```
predictions = model(x_train[:1]).numpy()
predictions
```

[63]:
```
array([[-0.26856932, -0.28690484, -0.2719099 , -0.337744  ,  0.37312242,
        -0.45397654, -0.27955002,  0.21283133, -0.33580464,  0.00139271]],
      dtype=float32)
```

The **tf.nn.softmax** function converts these logits to probabilities for each class:

[64]:
```
tf.nn.softmax(predictions).numpy()
```

[64]:
```
array([[0.08699868, 0.08541805, 0.08670855, 0.08118401, 0.1652707 ,
        0.07227554, 0.08604861, 0.14079341, 0.08134161, 0.11396085]],
      dtype=float32)
```

losses.SparseCategoricalCrossentropy takes a vector of logits and a True index and returns a scalar loss for each example. This loss is equal to the negative log probability of the true class: **The loss is zero if the model is sure of the correct class.**

[65]:
```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

[66]:
```
#tf.keras.losses.  # tab key to obtain the list of available fonctions
```

[67]:
```
loss_fn(y_train[:1], predictions).numpy()
```

[67]:
```
2.6272695
```

0.2.3 Before you start training, configure and compile the model

[68]:
```
model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])
```

0.2.4 Train and evaluate your model

[69]:
```
model.fit(x_train, y_train, epochs=5)
#print(x_train.shape[0]/32)
#model.fit(x_train, y_train, epochs=5, validation_split=0.2)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2957 -
accuracy: 0.9137
Epoch 2/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1430 -
accuracy: 0.9575
Epoch 3/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1065 -
accuracy: 0.9677
Epoch 4/5
```

```
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0871 -
accuracy: 0.9728
Epoch 5/5
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0758 -
accuracy: 0.9766
```

[69]:
```
<keras.callbacks.History at 0x7f26186ef400>
```

[70]:
```
#callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=2)
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(x_train, y_train,  epochs=20, callbacks=[callback], ␣
 ↪validation_split=0.2)
len(history.history['loss'])
```

```
Epoch 1/20
1500/1500 [==============================] - 2s 2ms/step - loss: 0.0658 -
accuracy: 0.9788 - val_loss: 0.0414 - val_accuracy: 0.9868
Epoch 2/20
1500/1500 [==============================] - 2s 2ms/step - loss: 0.0573 -
accuracy: 0.9811 - val_loss: 0.0424 - val_accuracy: 0.9858
Epoch 3/20
1500/1500 [==============================] - 2s 1ms/step - loss: 0.0505 -
accuracy: 0.9834 - val_loss: 0.0435 - val_accuracy: 0.9858
Epoch 4/20
1500/1500 [==============================] - 2s 2ms/step - loss: 0.0459 -
accuracy: 0.9845 - val_loss: 0.0458 - val_accuracy: 0.9846
```

[70]:
```
4
```

The Model.evaluate method checks the models performance, usually on a "Validation-set" or "Test-set".

[71]:
```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 0s - loss: 0.0694 - accuracy: 0.9801 - 323ms/epoch - 1ms/step
```

[71]:
```
[0.0694371908903122, 0.9800999760627747]
```

If you want your model to return a probability, you can wrap the trained model, and attach the softmax to it:

[72]:
```
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
```

[73]:
```
probability_model(x_test[:1])
```

[73]:
```
<tf.Tensor: shape=(1, 10), dtype=float32, numpy=
array([[1.4111662e-09, 5.9732941e-10, 5.4968194e-08, 4.8179479e-04,
```

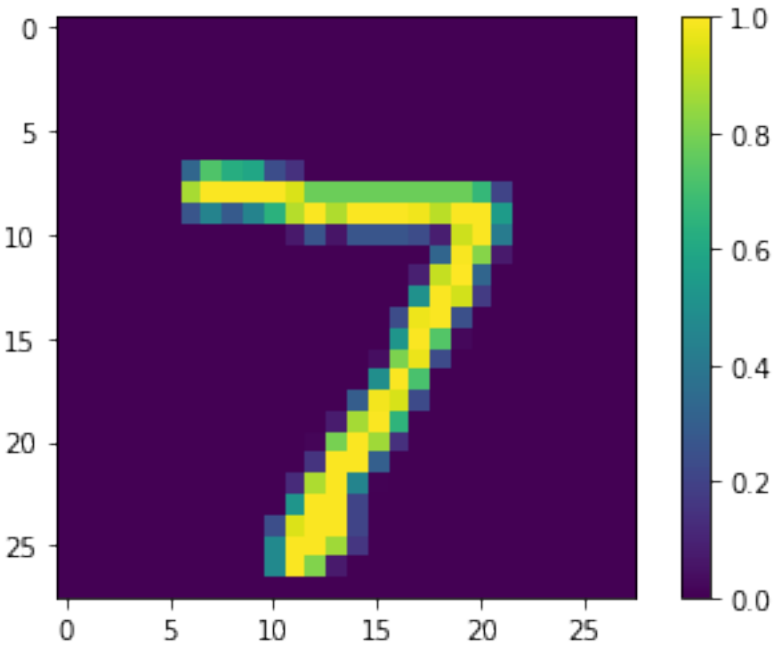
```
       1.9729827e-14, 8.8700549e-09, 3.1283237e-14, 9.9951530e-01,
       1.2299834e-07, 2.7611291e-06]], dtype=float32)>
```

[74]:
```python
#Print predicted values vs real values
def printErrValues(x_test,y_test, model):
    probability_model = tf.keras.Sequential([
      model,
      tf.keras.layers.Softmax()
    ])
    class_names = [0, 1, 2, 3, 4,5, 6, 7, 8, 9]
    #n=1000
    n=y_test.shape[0]
    class_namesp = tf.constant([class_names]*n)
    class_namesp = tf.reshape(class_namesp, [n*10])
    pm=probability_model(x_test[0:n])
    t=(pm>=tf.reduce_max(pm))
    #print(t)
    t = tf.reshape(t, [n*10])
    #print(t.shape)
    #print(class_namesp.shape)
    #print(class_namesp[t].shape)


    #print(class_namesp[t].numpy())
    #print(y_test[0:n+1])
    y_pred= class_namesp[t].numpy()
    nerr=0
    for i in range(y_pred.shape[0]):
        if y_test[i] != y_pred[i]:
            nerr += 1
            #print("#index = ",i,' y_test=',y_test[i], ' y_predic=',y_pred[i])
    print("nerr/n=",nerr, " / ",n)
```

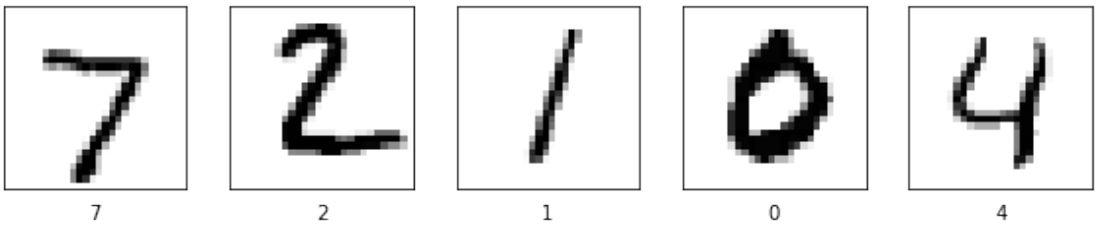
[75]:
```python
#Print predicted values vs real values
printErrValues(x_test,y_test, model)
```

```
nerr/n= 583  /  10000
```

[76]:
```python
plt.figure()
plt.imshow(x_test[0])
plt.colorbar()
plt.grid(False)
plt.show()
```



[77]:
```
class_names = ['0', '1', '2', '3', '4','5', '6', '7', '8', '9']

plt.figure(figsize=(10,10))
for i in range(5):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[y_test[i]])
plt.show()
```



0.2.5 Build a model with Functional API

[78]:
```
inputs = tf.keras.Input(shape=(28,28))
flatten= tf.keras.layers.Flatten(input_shape=(28, 28))
x=flatten(inputs)
dense = tf.keras.layers.Dense(128, activation="relu")
x = dense(x)
dropout=tf.keras.layers.Dropout(0.2)
x = dropout(x)
outputs = tf.keras.layers.Dense(10)(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs, name="mnist_model")
```

[79]:
```
model.summary()
```

```
Model: "mnist_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 28, 28)]          0

 flatten_5 (Flatten)         (None, 784)               0

 dense_10 (Dense)            (None, 128)               100480

 dropout_4 (Dropout)         (None, 128)               0

 dense_11 (Dense)            (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

[80]:
```
#tf.keras.utils.plot_model(model)
#tf.keras.utils.plot_model(model, "my_first_model_with_shape_info.png",␣
 ↪show_shapes=True)
tf.keras.utils.plot_model(model, show_shapes=True)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see
instructions at https://graphviz.gitlab.io/download/) for
plot_model/model_to_dot to work.
```

[81]:
```
model.compile(
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.Adam(),
    metrics=["accuracy"],
)
```

[82]:
```
#reload data if necessary
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

[83]:
```
history = model.fit(x_train, y_train, batch_size=64, epochs=5,␣
 ↪validation_split=0.2)

test_scores = model.evaluate(x_test, y_test, verbose=2)
print("Test loss:", test_scores[0])
print("Test accuracy:", test_scores[1])
```

```
Epoch 1/5
750/750 [==============================] - 2s 2ms/step - loss: 0.3765 -
accuracy: 0.8931 - val_loss: 0.1729 - val_accuracy: 0.9534
Epoch 2/5
750/750 [==============================] - 1s 2ms/step - loss: 0.1788 -
accuracy: 0.9481 - val_loss: 0.1334 - val_accuracy: 0.9621
Epoch 3/5
750/750 [==============================] - 1s 2ms/step - loss: 0.1318 -
accuracy: 0.9614 - val_loss: 0.1087 - val_accuracy: 0.9677
Epoch 4/5
750/750 [==============================] - 1s 2ms/step - loss: 0.1091 -
accuracy: 0.9672 - val_loss: 0.0956 - val_accuracy: 0.9722
Epoch 5/5
750/750 [==============================] - 1s 2ms/step - loss: 0.0905 -
accuracy: 0.9732 - val_loss: 0.0888 - val_accuracy: 0.9737
313/313 - 0s - loss: 0.0827 - accuracy: 0.9742 - 238ms/epoch - 762us/step
Test loss: 0.08268717676401138
Test accuracy: 0.9742000102996826
```

0.3 2. Convolutional Neural Network (CNN)

0.3.1 Load a dataset

[84]:
```
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
```

0.3.2 Build a model

[85]:
```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Conv2D(28, (3, 3), activation='relu',␣
 ↪input_shape=(28, 28,1)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(56, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(56, (3, 3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(56, activation='relu'))
model.add(tf.keras.layers.Dense(10))
```

[86]:
```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 26, 26, 28)        280

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 28)       0
 2D)

 conv2d_4 (Conv2D)           (None, 11, 11, 56)        14168

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 56)         0
 2D)

 conv2d_5 (Conv2D)           (None, 3, 3, 56)          28280

 flatten_6 (Flatten)         (None, 504)               0

 dense_12 (Dense)            (None, 56)                28280

 dense_13 (Dense)            (None, 10)                570

=================================================================
Total params: 71,578
Trainable params: 71,578
Non-trainable params: 0
_________________________________________________________________
```

[87]:
```
#tf.keras.utils.plot_model(model, show_shapes=True)
```

0.3.3 Compile the model

[88]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

0.3.4 Train and evaluate your model

[89]:
```
model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.1539 -
accuracy: 0.9517
Epoch 2/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.0463 -
accuracy: 0.9858
Epoch 3/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0339 -
accuracy: 0.9893
Epoch 4/5
1875/1875 [==============================] - 15s 8ms/step - loss: 0.0261 -
accuracy: 0.9917
Epoch 5/5
1875/1875 [==============================] - 14s 8ms/step - loss: 0.0207 -
accuracy: 0.9936
```

[89]:
```
<keras.callbacks.History at 0x7f26182b7ac0>
```

[90]:
```
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 0.0351 - accuracy: 0.9895 - 753ms/epoch - 2ms/step
```

[90]:
```
[0.0351063534617424, 0.9894999861717224]
```

[91]:
```
printErrValues(x_test,y_test, model)
```

```
nerr/n= 885  /  10000
```

0.4 Metrics

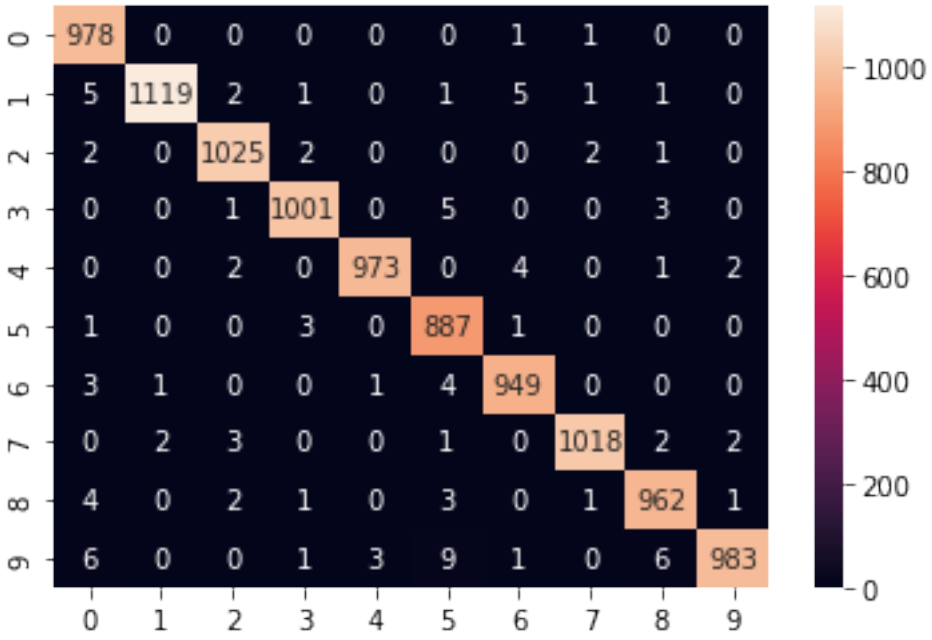
[92]:
```
from sklearn.metrics import  confusion_matrix, accuracy_score, precision_score,␣
 ↪recall_score, f1_score
import seaborn as sns
```

[93]:
```
probability_model = tf.keras.Sequential([
  model,
  tf.keras.layers.Softmax()
])
```

```
prob=probability_model(x_test)
print(prob.shape)
y_pred=tf.math.argmax(prob,axis=1)
print(y_pred.shape)
print(y_test.shape)
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True,fmt='d')
```

```
(10000, 10)
(10000,)
(10000,)
```

[93]: <AxesSubplot:>



[94]:
```
accuracy = accuracy_score(y_test, y_pred)
'''
precision = precision_score(y_test, y_pred,average=None)
recall = recall_score(y_test, y_pred,average=None)
f1score = f1_score(y_test, y_pred,average=None)
'''
precision = precision_score(y_test, y_pred,average="weighted")
recall = recall_score(y_test, y_pred,average="weighted")
f1score = f1_score(y_test, y_pred,average="weighted")

print(f"Accuracy = {accuracy}")
print(f"Precision = {precision}")
```

```
print(f"Recall = {recall}")
print(f"F1 Score = {f1score}")
```

```
Accuracy = 0.9895
Precision = 0.9895755607403444
Recall = 0.9895
F1 Score = 0.9895035541756534
```

[95]:
```
from sklearn.metrics import  classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       980
           1       1.00      0.99      0.99      1135
           2       0.99      0.99      0.99      1032
           3       0.99      0.99      0.99      1010
           4       1.00      0.99      0.99       982
           5       0.97      0.99      0.98       892
           6       0.99      0.99      0.99       958
           7       1.00      0.99      0.99      1028
           8       0.99      0.99      0.99       974
           9       0.99      0.97      0.98      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

0.5 3. Choose devices

[96]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpus = tf.config.list_physical_devices('GPU')
gpus
```

```
Num GPUs Available:  0
```
GPU compatible avec tensorflow

[96]:
```
[]
```

[97]:
```
print("Num CPUs Available: ", len(tf.config.list_physical_devices('CPU')))
cpus = tf.config.list_physical_devices('CPU')
cpus
```

```
Num CPUs Available:  1
```

[97]:
```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

[98]:
```
with tf.device('/CPU:0'):
    # Create some tensors
```
choice of using a given number of GPUs and CPUs

```
    a = tf.constant([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]])
    b = tf.constant([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])
    c = tf.matmul(a, b)
    print(c)
```

```
tf.Tensor(
[[22. 28.]
 [49. 64.]], shape=(2, 2), dtype=float32)
```

[99]:
```
tf.config.set_visible_devices(cpus[0], 'CPU')
print(1800*3)
```

```
5400
```

[100]:
```
tf.config.set_visible_devices([], 'GPU') # desactivate GPU
```

[ ]:
```

```

[ ]:
```

```