Data Science Fundamentals 5

Basic introduction on how to perform typical machine learning tasks with Python.

Prepared by Mykhailo Vladymyrov & Aris Marcolongo, Science IT Support, University Of Bern, 2020

Solutions to Part 4.

In [0]:
```
from matplotlib import  pyplot as plt
import numpy as np
from imageio import imread
import pandas as pd
from time import time as timer

import tensorflow as tf

%matplotlib inline
from matplotlib import animation
from IPython.display import HTML
```

EXERCISE 1: Train deeper network

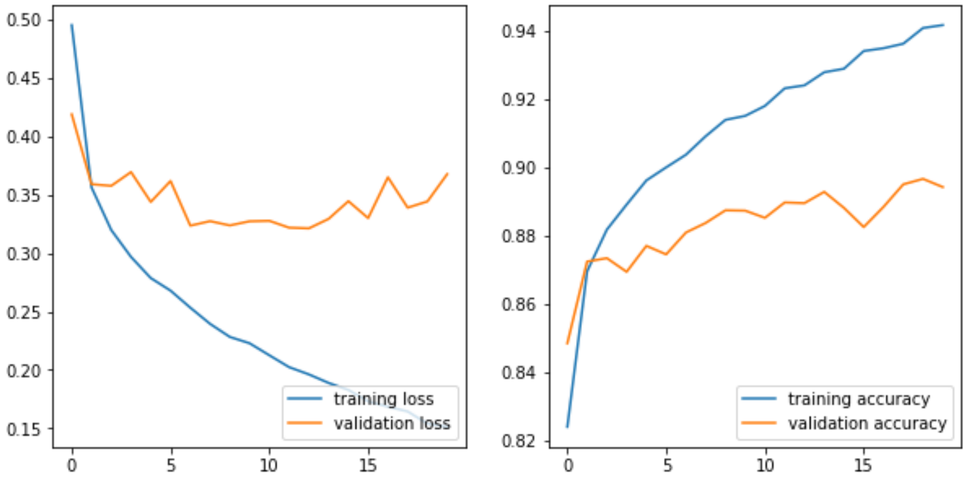
Make a deeper model, with wider layers. Remember to `'softmax'` activation in the last layer, as required for the classification task to encode pseudoprobabilities. In the other layers you could use `'relu'`.

Try to achieve 90% accuracy. Does your model overfit?

In [0]:
```
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
x_train = x_train/255
x_test = x_test/255

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [3]:
```
# 1. create model
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(1024, activation='relu'),
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])


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

model.summary()

# 2. train the model
save_path = 'save/mnist_{epoch}.ckpt'
save_callback = tf.keras.callbacks.ModelCheckpoint(filepath=save_path, s
ave_weights_only=True)

hist = model.fit(x=x_train, y=y_train,
                 epochs=20, batch_size=128,
                 validation_data=(x_test, y_test),
                 callbacks=[save_callback])

# 3. plot the loss and accuracy evolution during training
fig, axs = plt.subplots(1, 2, figsize=(10,5))
axs[0].plot(hist.epoch, hist.history['loss'])
axs[0].plot(hist.epoch, hist.history['val_loss'])
axs[0].legend(('training loss', 'validation loss'), loc='lower right')
axs[1].plot(hist.epoch, hist.history['accuracy'])
axs[1].plot(hist.epoch, hist.history['val_accuracy'])

axs[1].legend(('training accuracy', 'validation accuracy'), loc='lower r
ight')
plt.show()

# 4. evaluate model in best point (before overfitting)
model.load_weights('save/mnist_10.ckpt')
model.evaluate(x_test,  y_test, verbose=2)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten (Flatten)            (None, 784)               0         
_________________________________________________________________
dense (Dense)                (None, 1024)              803840    
_________________________________________________________________
dense_1 (Dense)              (None, 256)               262400    
_________________________________________________________________
dense_2 (Dense)              (None, 64)                16448     
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650       
=================================================================
Total params: 1,083,338
Trainable params: 1,083,338
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 2s 4ms/step - loss: 0.4948 - a
ccuracy: 0.8242 - val_loss: 0.4185 - val_accuracy: 0.8486
Epoch 2/20
469/469 [==============================] - 2s 3ms/step - loss: 0.3561 - a
ccuracy: 0.8697 - val_loss: 0.3588 - val_accuracy: 0.8725
Epoch 3/20
469/469 [==============================] - 2s 3ms/step - loss: 0.3196 - a
ccuracy: 0.8819 - val_loss: 0.3574 - val_accuracy: 0.8735
Epoch 4/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2966 - a
ccuracy: 0.8892 - val_loss: 0.3692 - val_accuracy: 0.8695
Epoch 5/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2786 - a
ccuracy: 0.8963 - val_loss: 0.3436 - val_accuracy: 0.8771
Epoch 6/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2678 - a
ccuracy: 0.9001 - val_loss: 0.3615 - val_accuracy: 0.8746
Epoch 7/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2533 - a
ccuracy: 0.9038 - val_loss: 0.3234 - val_accuracy: 0.8810
Epoch 8/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2394 - a
ccuracy: 0.9092 - val_loss: 0.3272 - val_accuracy: 0.8838
Epoch 9/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2283 - a
ccuracy: 0.9140 - val_loss: 0.3236 - val_accuracy: 0.8875
Epoch 10/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2228 - a
ccuracy: 0.9151 - val_loss: 0.3271 - val_accuracy: 0.8874
Epoch 11/20
469/469 [==============================] - 2s 3ms/step - loss: 0.2126 - a
ccuracy: 0.9180 - val_loss: 0.3275 - val_accuracy: 0.8853
Epoch 12/20
469/469 [==============================] - 1s 3ms/step - loss: 0.2024 - a
ccuracy: 0.9232 - val_loss: 0.3216 - val_accuracy: 0.8898
Epoch 13/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1962 - a
ccuracy: 0.9241 - val_loss: 0.3210 - val_accuracy: 0.8896
Epoch 14/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1889 - a
ccuracy: 0.9279 - val_loss: 0.3292 - val_accuracy: 0.8929
Epoch 15/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1829 - a
ccuracy: 0.9289 - val_loss: 0.3443 - val_accuracy: 0.8882
Epoch 16/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1732 - a
ccuracy: 0.9341 - val_loss: 0.3298 - val_accuracy: 0.8826
Epoch 17/20
469/469 [==============================] - 2s 3ms/step - loss: 0.1686 - a
```




```
313/313 - 1s - loss: 0.3272 - accuracy: 0.8838
```

```
Out[3]: [0.3271946609020233, 0.8838000297546387]
```