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

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
```

1. Classification with neural network

1. Bulding a neural network

The following creates a 'model'. It is an object containing the ML model itself - a simple 3-layer fully connected neural network, optimization parameters, as well as tha interface for model training.

In [0]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

Model summary provides information about the model's layers and trainable parameters

In [3]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

2. Model training

The `fit` function is the interface for model training. Here one can specify training and validation datasets, minibatch size, and the number of training epochs.

We will also save the state of the trainable variables after each epoch:

In [4]:
```
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']
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
```

In [5]:
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(10, activation='softmax')
])

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

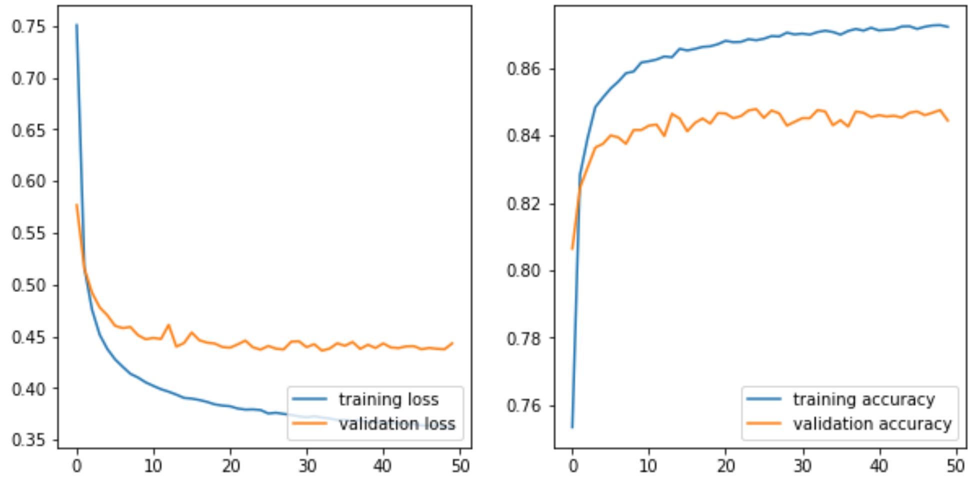
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                7850
=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

Here during training we also save the trained models checkpoints after each epoch of training.

In [6]:
```
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=50, batch_size=128,
                 validation_data=(x_test, y_test),
                 callbacks=[save_callback])
```

```
Epoch 1/50
469/469 [==============================] - 2s 4ms/step - loss: 0.7509 - a
ccuracy: 0.7534 - val_loss: 0.5770 - val_accuracy: 0.8064
Epoch 2/50
469/469 [==============================] - 2s 4ms/step - loss: 0.5161 - a
ccuracy: 0.8283 - val_loss: 0.5155 - val_accuracy: 0.8245
Epoch 3/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4758 - a
ccuracy: 0.8393 - val_loss: 0.4922 - val_accuracy: 0.8305
Epoch 4/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4515 - a
ccuracy: 0.8484 - val_loss: 0.4780 - val_accuracy: 0.8364
Epoch 5/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4378 - a
ccuracy: 0.8512 - val_loss: 0.4704 - val_accuracy: 0.8375
Epoch 6/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4279 - a
ccuracy: 0.8539 - val_loss: 0.4603 - val_accuracy: 0.8400
Epoch 7/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4206 - a
ccuracy: 0.8559 - val_loss: 0.4580 - val_accuracy: 0.8394
Epoch 8/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4138 - a
ccuracy: 0.8584 - val_loss: 0.4591 - val_accuracy: 0.8375
Epoch 9/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4102 - a
ccuracy: 0.8589 - val_loss: 0.4512 - val_accuracy: 0.8416
Epoch 10/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4056 - a
ccuracy: 0.8616 - val_loss: 0.4472 - val_accuracy: 0.8416
Epoch 11/50
469/469 [==============================] - 2s 4ms/step - loss: 0.4022 - a
ccuracy: 0.8620 - val_loss: 0.4486 - val_accuracy: 0.8429
Epoch 12/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3990 - a
ccuracy: 0.8625 - val_loss: 0.4475 - val_accuracy: 0.8432
Epoch 13/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3965 - a
ccuracy: 0.8634 - val_loss: 0.4612 - val_accuracy: 0.8398
Epoch 14/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3937 - a
ccuracy: 0.8632 - val_loss: 0.4401 - val_accuracy: 0.8464
Epoch 15/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3906 - a
ccuracy: 0.8657 - val_loss: 0.4436 - val_accuracy: 0.8450
Epoch 16/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3899 - a
ccuracy: 0.8652 - val_loss: 0.4538 - val_accuracy: 0.8412
Epoch 17/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3886 - a
ccuracy: 0.8656 - val_loss: 0.4463 - val_accuracy: 0.8437
Epoch 18/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3867 - a
ccuracy: 0.8662 - val_loss: 0.4440 - val_accuracy: 0.8450
Epoch 19/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3843 - a
ccuracy: 0.8664 - val_loss: 0.4431 - val_accuracy: 0.8435
Epoch 20/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3831 - a
ccuracy: 0.8670 - val_loss: 0.4398 - val_accuracy: 0.8466
Epoch 21/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3825 - a
ccuracy: 0.8680 - val_loss: 0.4391 - val_accuracy: 0.8465
Epoch 22/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3803 - a
ccuracy: 0.8676 - val_loss: 0.4423 - val_accuracy: 0.8451
Epoch 23/50
469/469 [==============================] - 2s 4ms/step - loss: 0.3793 - a
```

In [7]:
```
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()
```



Current model performance can be evaluated on a dataset:

In [8]:
```
model.evaluate(x_test,  y_test, verbose=2)
```

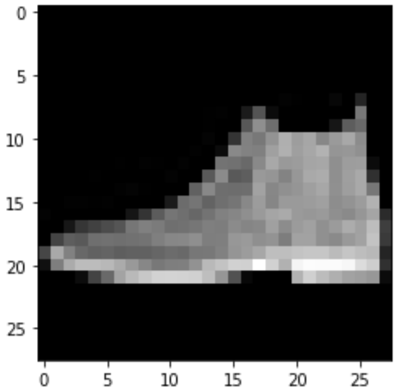
```
313/313 - 1s - loss: 0.4433 - accuracy: 0.8444
```

Out[8]:
```
[0.4432746469974518, 0.8443999886512756]
```

We can test trained model on a image:

In [9]:
```
im_id = 0
y_pred = model(x_test)

y_pred_most_probable = np.argmax(y_pred[im_id])
print('true lablel: ', y_test[im_id],
      '; predicted: ',  y_pred_most_probable,
      f'({class_names[y_pred_most_probable]})')
plt.imshow(x_test[im_id], cmap='gray');
```

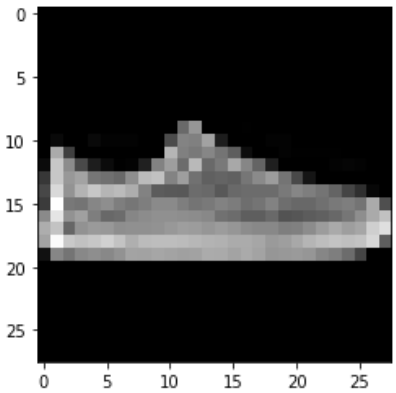
```
true lablel:  9 ; predicted:  9 (Ankle boot)
```



As well as inspect on which samples does the model fail:

In [10]:
```
y_pred_most_probable_all = np.argmax(y_pred, axis=1)
wrong_pred_map = y_pred_most_probable_all!=y_test
wrong_pred_idx = np.arange(len(wrong_pred_map))[wrong_pred_map]

im_id = wrong_pred_idx[0]

y_pred_most_probable = y_pred_most_probable_all[im_id]
print('true lablel: ', y_test[im_id],
      f'({class_names[y_test[im_id]]})',
      '; predicted: ',  y_pred_most_probable,
      f'({class_names[y_pred_most_probable]})')
plt.imshow(x_test[im_id], cmap='gray');
```

```
true lablel:  7 (Sneaker) ; predicted:  5 (Sandal)
```



3. Loading trained model

In [11]:
```
model.load_weights('save/mnist_1.ckpt')
model.evaluate(x_test,  y_test, verbose=2)

model.load_weights('save/mnist_12.ckpt')
model.evaluate(x_test,  y_test, verbose=2)

model.load_weights('save/mnist_18.ckpt')
model.evaluate(x_test,  y_test, verbose=2)
```

```
313/313 - 1s - loss: 0.5770 - accuracy: 0.8064
313/313 - 1s - loss: 0.4475 - accuracy: 0.8432
313/313 - 1s - loss: 0.4440 - accuracy: 0.8450
```

Out[11]:
```
[0.4440324306488037, 0.8450000286102295]
```

4. Inspecting trained variables

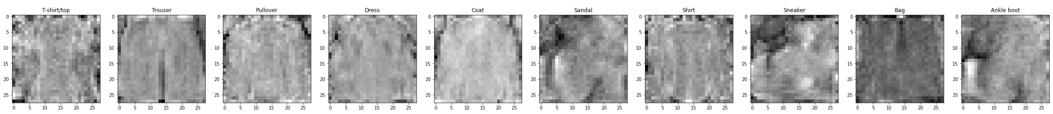
We can obtain the trained variables from model layers:

In [12]:
```
l = model.get_layer(index=1)
w, b = l.weights

w = w.numpy()
b = b.numpy()
print(w.shape, b.shape)
w = w.reshape((28,28,-1)).transpose((2, 0, 1))
```

```
(784, 10) (10,)
```

Let's visualize first 5:

In [13]:
```
n = 10
fig, axs = plt.subplots(1, n, figsize=(4.1*n,4))
for i, wi in enumerate(w[:n]):
  axs[i].imshow(wi, cmap='gray')
  axs[i].set_title(class_names[i])
```



6. Inspecting gradients

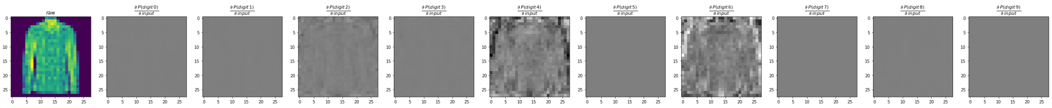
We can also evaluate the gradients of each output with respect to an input:

In [17]:
```
idx = 112
inp_v = x_train[idx:idx+1]  # use some image to compute gradients with r
espect to

inp = tf.constant(inp_v)  # create tf constant tensor
with tf.GradientTape() as tape:  # gradient tape for gradint evaluation
  tape.watch(inp)  # take inp as variable
  preds = model(inp) # evaluate model output

grads = tape.jacobian(preds, inp)  # evaluate d preds[i] / d inp[j]
print(grads.shape, '<- (Batch_preds, preds[i], Batch_inp, inp[y], inp
[x])')
grads = grads.numpy()[0,:,0]
```

```
(1, 10, 1, 28, 28) <- (Batch_preds, preds[i], Batch_inp, inp[y], inp[x])
```

In [18]:
```
print('prediction:', np.argmax(preds[0]))
fig, axs = plt.subplots(1, 11, figsize=(4.1*11,4))
axs[0].imshow(inp_v[0])
axs[0].set_title('raw')
vmin,vmax = grads.min(), grads.max()
for i, g in enumerate(grads):
  axs[i+1].imshow(g, cmap='gray', vmin=vmin, vmax=vmax)
  axs[i+1].set_title(r'$\frac{\partial\;P(digit\,%d)}{\partial\;input}$'
% i, fontdict={'size':16})
```

```
prediction: 6
```



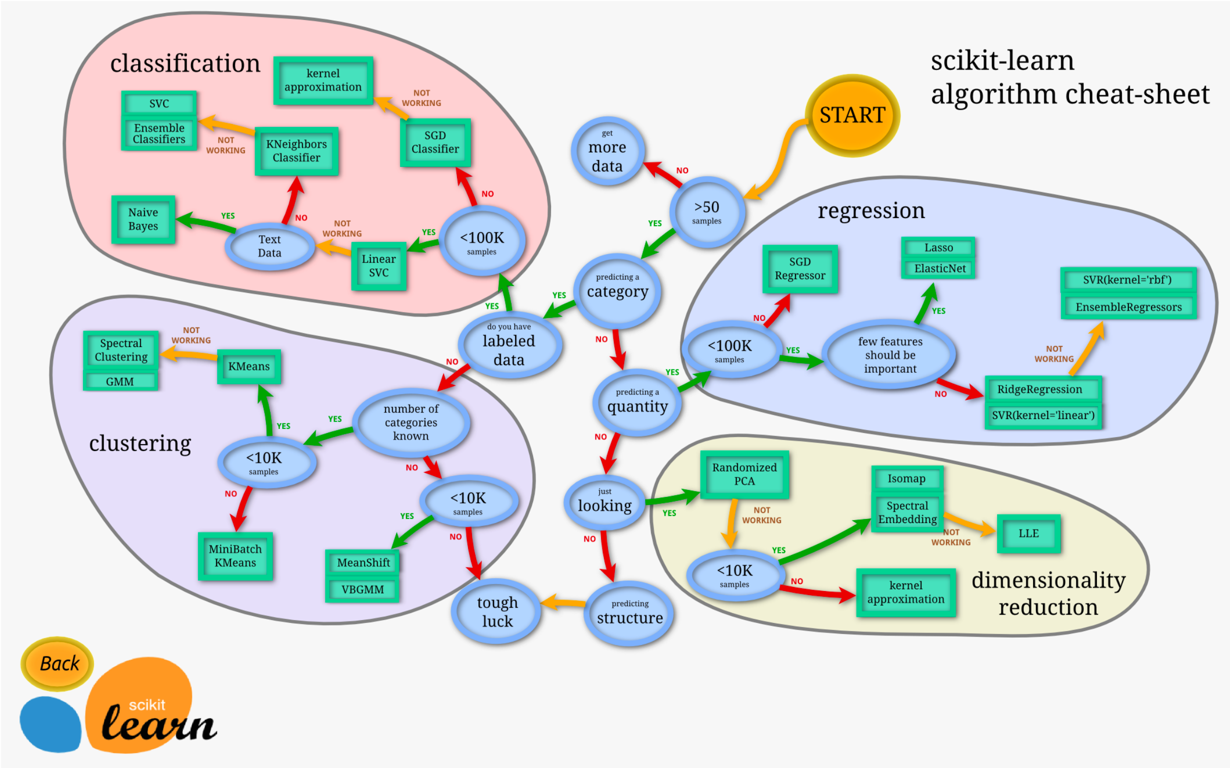
EXERCISE 1: Train deeper network

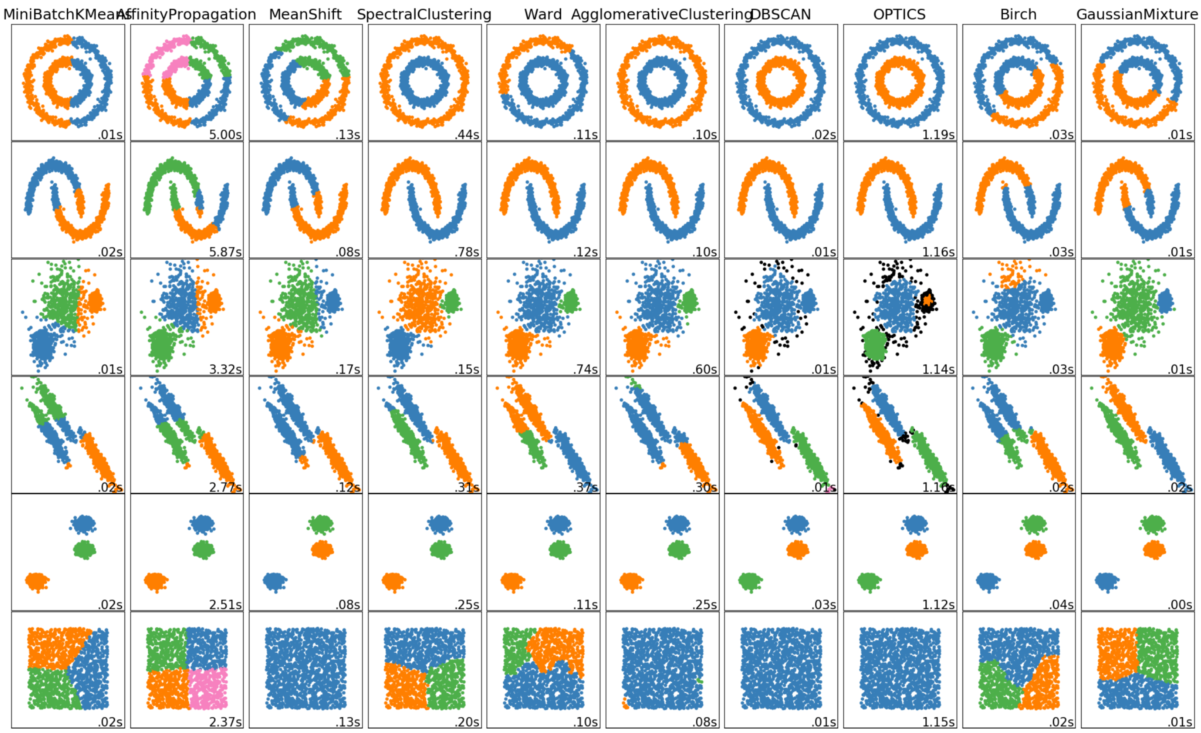
Make a deeper model, with wider layers. Remember to `'softmax'` activation in the last layer, as required for the classification task to encode pseudoprobabilities. In the other layers you could use `'relu'`.

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

In [0]:
```
# 1. create model
# 2. train the model
# 3. plot the loss and accuracy evolution during training
# 4. evaluate model in best point (before overfitting)
```

2. Extras and Q&A


scikit-learn
algorithm cheat-sheet
START
classification
regression
clustering
dimensionality
reduction
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
<100K samples
SGD Classifier
kernel approximation
KNeighbors Classifier
SVC
Ensemble Classifiers
Naive Bayes
Text Data
Linear SVC
number of categories known
<10K samples
KMeans
Spectral Clustering
GMM
MiniBatch KMeans
MeanShift
VBGMM
SGD Regressor
Lasso
ElasticNet
few features should be important
RidgeRegression
SVR(kernel='linear')
SVR(kernel='rbf')
EnsembleRegressors
Randomized PCA
Isomap
Spectral Embedding
LLE
YES
NO
NOT WORKING
Back
scikit
learn


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
AgglomerativeClustering
DBSCAN
OPTICS
Birch
GaussianMixture