Topics : Introduction

LAB 1

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

allows to print all the variables in the cell (without it only the last will be printed)

In [2]:
```
import tensorflow as tf
import numpy as np

tf.__version__
tf.keras.__version__
```

Out[2]: '2.1.0'

Out[2]: '2.2.4-tf'

Tensors

NumPy

In [3]:
```
# Create a tensor
# ------------------------------------------------------------------------------
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

# np.array(object, dtype)
array2d = np.array(matrix, dtype=np.int32)
array2d

# other alternatives:
# np.zeros
zeros = np.zeros(shape=[4, 4], dtype=np.int32)
zeros

# np.ones
ones = np.ones(shape=[4, 4], dtype=np.int32)
ones

# np.zeros_like / np.ones_like
zeros_like = np.zeros_like(array2d)
zeros_like

# np.arange
range_array = np.arange(start=0, stop=10, step=2)
range_array

# random integers over [low, high)
rnd_int = np.random.randint(low=0, high=10, size=3)
rnd_int

# sampling from distribution: np.random
# uniform [0, 1)
rnd_uniform = np.random.rand(3, 3)  # rand(d0, d1, ..., dN)
rnd_uniform

# normal with mean 0 and variance 1
rnd_normal = np.random.randn(3, 3)  # randn(d0, d1, ..., dN)
rnd_normal

# many others (see documentation)
?np.random
```

Out[3]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Out[3]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

Out[3]:
```
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
```

Out[3]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

Out[3]:
```
array([0, 2, 4, 6, 8])
```

Out[3]:
```
array([1, 8, 7])
```

Out[3]:
```
array([[0.13402089, 0.45400892, 0.01314201],
       [0.17221622, 0.74476802, 0.72909835],
       [0.46892895, 0.26987469, 0.23907854]])
```

Out[3]:
```
array([[-1.51245328, -1.15482814,  0.70083901],
       [ 0.11563247,  0.26962634,  1.30057603],
       [ 0.26182614,  1.25481143, -2.012974  ]])
```

In [4]:
```
# Attributes
# ------------------------------------------------------------------------------
# dtype: type of the elements
print('array.dtype: %s' % array2d.dtype)

# shape: dimensions of the array
print('array.shape: ', array2d.shape)

# ndim: number of dimensions of the array (axis)
print('array.ndim: {}'.format(array2d.ndim))
```

```
array.dtype: int32
array.shape:  (3, 3)
array.ndim: 2
```

Tensorflow

In [5]:
```
# Create a tensor
# ------------------------------------------------------------------------------
# tf.constant(object, dtype)
tensor2d = tf.constant(matrix, dtype=tf.int32)
tensor2d

# tf.convert_to_tensor(object, dtype)
tensor2d = tf.constant(array2d, dtype=tf.float32)
tensor2d

# other alternatives:
# tf.zeros
zeros = tf.zeros(shape=[4, 4], dtype=tf.int32)
zeros

# tf.ones
ones = tf.ones(shape=[4, 4], dtype=tf.int32)
ones

# tf.zeros_like / tf.ones_like
zeros_like = tf.zeros_like(array2d)
zeros_like

# np.arange
range_array = tf.range(start=0, limit=10, delta=2)
range_array

# tf.random.uniform
rnd_uniform = tf.random.uniform(shape=[5, 5], minval=10, maxval=20)
rnd_uniform

# ?tf.random

# remember you can use convert_to_tensor()
# e.g.
rnd_rayleigh = tf.convert_to_tensor(
    np.random.rayleigh(size=[2, 2]), dtype=tf.float32)
rnd_rayleigh
```

we sample each element of the tensor from a rayleigh distribution

Out[5]:
```
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])>
```

Out[5]:
```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]], dtype=float32)>
```

Out[5]:
```
<tf.Tensor: shape=(4, 4), dtype=int32, numpy=
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])>
```

Out[5]:
```
<tf.Tensor: shape=(4, 4), dtype=int32, numpy=
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])>
```

Out[5]:
```
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])>
```

Out[5]:
```
<tf.Tensor: shape=(5,), dtype=int32, numpy=array([0, 2, 4, 6, 8])>
```

Out[5]:
```
<tf.Tensor: shape=(5, 5), dtype=float32, numpy=
array([[19.055876 , 19.397755 , 16.354115 , 17.45838  , 18.25626  ],
       [18.165476 , 16.553072 , 16.24017  , 14.982023 , 17.357567 ],
       [13.668299 , 19.875946 , 18.928833 , 18.223024 , 14.381664 ],
       [12.0960655, 10.1256   , 11.9855   , 10.604209 , 12.5583725],
       [10.379154 , 18.865265 , 10.4804945, 12.521774 , 12.956839 ]],
      dtype=float32)>
```

Out[5]:
```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[3.092807  , 0.45312917],
       [1.4323252 , 2.5773199 ]], dtype=float32)>
```

In [6]:
```
# Attributes
# ------------------------------------------------------------------------------
# dtype: type of the elements
print('tensor2d.dtype: %s' % tensor2d.dtype)

# shape: dimensions of the array
print('tensor2d.shape: ', tensor2d.shape)

# ndim: number of dimensions of the array (axis)
print('tensor2d.ndim: {}'.format(tensor2d.ndim))

# device: device placement for tensor
print('tensor2d.device: {}'.format(tensor2d.device))
```

```
tensor2d.dtype: <dtype: 'float32'>
tensor2d.shape:  (3, 3)
tensor2d.ndim: 2
tensor2d.device: /job:localhost/replica:0/task:0/device:CPU:0
```

Operations on tensors

In [7]: (12)
```
# Cast: tf.cost(x, dtype)
# ------------------------------------------------------------------------------
tensor2d.dtype
tensor2d = tf.cast(tensor2d, dtype=tf.int32)
tensor2d.dtype
```

```
# ?tf.DType
```

Out[12]:
```
tf.float32
```

Out[12]:
```
tf.int32
```

In [13]:
```
# Reshape:
# ------------------------------------------------------------------------------
# tf.reshape(tensor, shape)
tensor1d = tf.range(9)
tensor1d

tensor2d = tf.reshape(tensor1d, shape=[3, 3])
# tensor2d = tf.reshape(tensor1d, shape=[3, -1]) # negative dimension is accepted
tensor2d

flattened = tf.reshape(tensor2d, shape=[-1])
flattened

# tf.expand_dims(input, axis)
tensor2d = tf.reshape(tf.range(1, 5), shape=[2, 2])
tensor2d
tensor3d = tf.expand_dims(tensor2d, axis=-1)
# tensor3d = tf.reshape(tensor2d, shape=[2, 2, 1])
tensor3d

# tf.squeeze(input, axis)
tensor2d = tf.squeeze(tensor3d, axis=-1)
# tensor2d = tf.reshape(tensor3d, shape=[2, 2])
tensor2d
```

this means that the second dimension adapts to the first (tensor1D = 3×4 (vector)) ⇒ shape = [3, -1] generates automatically 3×4)

Out[13]:
```
<tf.Tensor: shape=(9,), dtype=int32, numpy=array([0, 1, 2, 3, 4, 5, 6, 7, 8])>
```

Out[13]:
```
<tf.Tensor: shape=(3, 3), dtype=int32, numpy=
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])>
```

Out[13]:
```
<tf.Tensor: shape=(9,), dtype=int32, numpy=array([0, 1, 2, 3, 4, 5, 6, 7, 8])>
```

Out[13]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4]])>
```

Out[13]:
```
<tf.Tensor: shape=(2, 2, 1), dtype=int32, numpy=
array([[[1],
        [2]],

       [[3],
        [4]]])>
```

Out[13]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[1, 2],
       [3, 4]])>
```

In [14]:
```
# Math:
# ------------------------------------------------------------------------------
# +, -, *, / operators (element-wise)
t1 = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=tf.float32)
t2 = tf.eye(num_rows=3, num_columns=3)
t1+t2
# tf.add(t1, t2)
t1-t2
# tf.subtract(t1, t2)
t1*t2
# tf.multiply(t1, t2)
t1/(t2+1)
# tf.divide(t1, tf.add(t2, 1))

# tf.tensordot(a, b, axes)
t = tf.constant([[1, 2], [3, 4]], dtype=tf.int32)
tf.tensordot(t, t, axes=[[1], [0]]) # matrix multiplication
```

= I

Out[14]:
```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[ 2.,  2.,  3.],
       [ 4.,  6.,  6.],
       [ 7.,  8., 10.]], dtype=float32)>
```

Out[14]:
```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[0., 2., 3.],
       [4., 4., 6.],
       [7., 8., 8.]], dtype=float32)>
```

Out[14]:
```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[1., 0., 0.],
       [0., 5., 0.],
       [0., 0., 9.]], dtype=float32)>
```

Out[14]:
```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[0.5, 2. , 3. ],
       [4. , 2.5, 6. ],
       [7. , 8. , 4.5]], dtype=float32)>
```

Out[14]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[ 7, 10],
       [15, 22]])>
```

In [15]:
```
# Slicing:
# ------------------------------------------------------------------------------
tensor2d = tf.convert_to_tensor(matrix, dtype=tf.float32)
tensor2d

# Pythonic '[]'
# ------------------------------------------------------------------------------
# Get a single element: tensor[idx1, idx2, ..., idxN]
# e.g. get element '3'
tensor2d[0, 2]
tensor2d[0, -1]
tensor2d[-3, -1]
```

```
# Get a slice: tensor[startidx1:endidx1:step1,..., startidxN:endidxN:stepN] (endidx excluded)
#          |5 6|
# e.g. get |8 9| sub-tensor
tensor2d[1:3:1, 1:3:1]
tensor2d[1:3, 1:3]
tensor2d[1:, 1:]

# Missing indices are considered complete slices
# e.g. get first row
tensor2d[0, :]

# Negative indices are accepted
tensor2d[0:-1, 0]

# e.g. get first row reversed
tensor2d[0, ::-1]
# --------------------------------------------------------------------------------

# API
# --------------------------------------------------------------------------------
# tf.slice(input_, begin, size)

#          |5 6|
# e.g. get |8 9| sub-tensor
tf.slice(tensor2d, [1, 1], [2, 2])

# indexing with condition
array2d = np.array(matrix, dtype=np.float32)
array2d[np.where(array2d % 2 == 0)]

# tensor2d[tf.where(array2d % 2 == 0)] does not work!
# use tf.gather_nd(params, indices) and tf.where(condition) to find indices
indices = tf.where(tensor2d % 2 == 0)
tf.gather_nd(tensor2d, indices)
```

Out[15]:
```
<tf.Tensor: shape=(3, 3), dtype=float32, numpy=
array([[1., 2., 3.],
       [4., 5., 6.],
       [7., 8., 9.]], dtype=float32)>
```

Out[15]: `<tf.Tensor: shape=(), dtype=float32, numpy=3.0>`

Out[15]: `<tf.Tensor: shape=(), dtype=float32, numpy=3.0>`

Out[15]: `<tf.Tensor: shape=(), dtype=float32, numpy=3.0>`

Out[15]:
```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[5., 6.],
       [8., 9.]], dtype=float32)>
```

Out[15]:
```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[5., 6.],
       [8., 9.]], dtype=float32)>
```

Out[15]:
```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[5., 6.],
       [8., 9.]], dtype=float32)>
```

Out[15]: `<tf.Tensor: shape=(3,), dtype=float32, numpy=array([1., 2., 3.], dtype=float32)>`

Out[15]: `<tf.Tensor: shape=(2,), dtype=float32, numpy=array([1., 4.], dtype=float32)>`

Out[15]: `<tf.Tensor: shape=(3,), dtype=float32, numpy=array([3., 2., 1.], dtype=float32)>`

Out[15]:
```
<tf.Tensor: shape=(2, 2), dtype=float32, numpy=
array([[5., 6.],
       [8., 9.]], dtype=float32)>
```

Out[15]: `array([2., 4., 6., 8.], dtype=float32)`

Out[15]: `<tf.Tensor: shape=(4,), dtype=float32, numpy=array([2., 4., 6., 8.], dtype=float32)>`

In [17]:
```
# Let's play with an image!
# --------------------------------------------------------------------------------
from PIL import Image

img = Image.open('airplane.jpg')
img
```

Out[17]:

In [18]:
```
# Convert image to tensor
rgb_tensor = tf.convert_to_tensor(np.array(img))
# rgb_tensor

gs_tensor = tf.convert_to_tensor(np.array(img.convert('L')))
# gs_tensor

# Change the color of the airplane!
# tf.where again, but tf.where(condition, x, y)
new_t = tf.where(tf.expand_dims(gs_tensor, -1) > 235, [255, 178, 102], rgb_tensor)
new_img = Image.fromarray(new_t.numpy().astype(np.uint8))
new_img
```

Out[18]:

In [19]:
```
# Splitting:
# -------------------------------------------------------------------------------
rgb_tensor.shape

# tf.split(value, num_or_size_splits, axis)
# vertical
output = tf.split(rgb_tensor, 3, axis=0)
output[0].shape
output[1].shape
output[2].shape

# or
top, bottom = tf.split(rgb_tensor, [500, -1], axis=0)
top.shape
bottom.shape

# horizontal
top_left, top_right = tf.split(top, [900, -1], axis=1)
bottom_left, bottom_right = tf.split(bottom, [900, -1], axis=1)

Image.fromarray(top_left.numpy())
Image.fromarray(top_right.numpy())
Image.fromarray(bottom_left.numpy())
Image.fromarray(bottom_right.numpy())
```

Out[19]: TensorShape([999, 1778, 3])

Out[19]: TensorShape([333, 1778, 3])

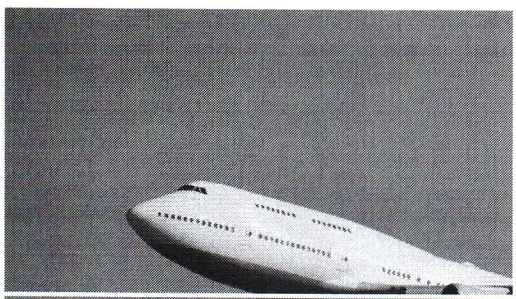
Out[19]: TensorShape([333, 1778, 3])

Out[19]: TensorShape([333, 1778, 3])

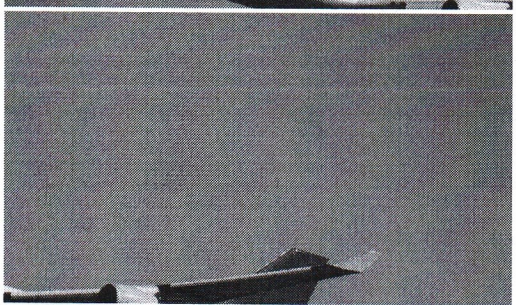
Out[19]: TensorShape([500, 1778, 3])

Out[19]: TensorShape([499, 1778, 3])

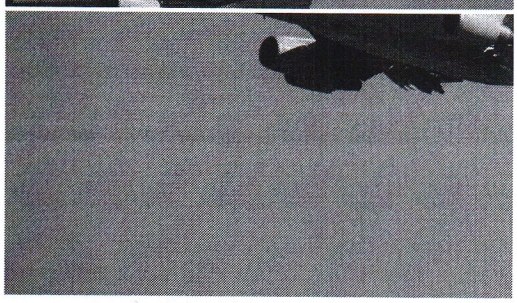
Out[19]:



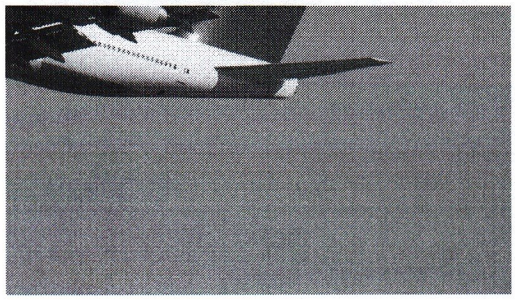
Out[19]:



Out[19]:



Out[19]:



In [20]:
```
# Stacking:
# ---------------------------------------------------------------------------
# tf.concat(values, axis)

# horizontally
top = tf.concat([top_left, top_right], axis=1)
bottom = tf.concat([bottom_left, bottom_right], axis=1)

# vertically
restored = tf.concat([top, bottom], axis=0)

Image.fromarray(restored.numpy())
```

Out[20]:

In [ ]:

Topics: Example (fashion MNIST)

In [1]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [2]:
```
import tensorflow as tf
import numpy as np

# Set the seed for random operations.
# This let our experiments to be reproducible.
tf.random.set_seed(1234)
```

Dataset

tf.data.Dataset.range

In [3]:
```
# Create a Dataset of sequential numbers
# -----------------------------------------------------------------------------
print("Dataset.range examples:")
print("-----------------------")

range_dataset = tf.data.Dataset.range(0, 20, 1)

print("\n1. Dataset")
for el in range_dataset:
    print(el)

# Divide in batches
bs = 3
range_dataset = tf.data.Dataset.range(0, 20, 1).batch(bs, drop_remainder=False)

print("\n2. Dataset + batch")
for el in range_dataset:
    print(el)

# Apply a transformation to each element
def map_fn(x):
    return x**2

range_dataset = tf.data.Dataset.range(0, 20, 1).batch(bs, drop_remainder=False).map(map_fn)

print("\n3. Dataset + batch + map")
for el in range_dataset:
    print(el)

# Filter dataset based on a condition
def filter_fn(x):
    return tf.equal(tf.math.mod(x, 2), 0)

range_dataset = tf.data.Dataset.range(0, 20, 1).filter(filter_fn)

print("\n4. Dataset + filter")
for el in range_dataset:
    print(el)

# Random shuffling
range_dataset = tf.data.Dataset.range(0, 20, 1).shuffle(
    buffer_size=20, reshuffle_each_iteration=False, seed=1234).batch(bs)

print("\n5. Dataset + shuffle + batch")
for el in range_dataset:
    print(el)
```

initial value
final value
step

Advice: keep the mini-batches as large as we can

we can apply methods sequentially

Common practice to avoid overfitting: at each epoch we present data to the model with a different order (how? we shuffle the data (whole data) and then we re-divide it in batches)
⟹ shuffle + batching

```
Dataset.range examples:
-----------------------

1. Dataset
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor(4, shape=(), dtype=int64)
tf.Tensor(5, shape=(), dtype=int64)
tf.Tensor(6, shape=(), dtype=int64)
tf.Tensor(7, shape=(), dtype=int64)
tf.Tensor(8, shape=(), dtype=int64)
tf.Tensor(9, shape=(), dtype=int64)
tf.Tensor(10, shape=(), dtype=int64)
tf.Tensor(11, shape=(), dtype=int64)
tf.Tensor(12, shape=(), dtype=int64)
tf.Tensor(13, shape=(), dtype=int64)
tf.Tensor(14, shape=(), dtype=int64)
tf.Tensor(15, shape=(), dtype=int64)
tf.Tensor(16, shape=(), dtype=int64)
tf.Tensor(17, shape=(), dtype=int64)
tf.Tensor(18, shape=(), dtype=int64)
tf.Tensor(19, shape=(), dtype=int64)

2. Dataset + batch
tf.Tensor([0 1 2], shape=(3,), dtype=int64)
tf.Tensor([3 4 5], shape=(3,), dtype=int64)
tf.Tensor([6 7 8], shape=(3,), dtype=int64)
tf.Tensor([ 9 10 11], shape=(3,), dtype=int64)
tf.Tensor([12 13 14], shape=(3,), dtype=int64)
tf.Tensor([15 16 17], shape=(3,), dtype=int64)
tf.Tensor([18 19], shape=(2,), dtype=int64)

3. Dataset + batch + map
tf.Tensor([0 1 4], shape=(3,), dtype=int64)
tf.Tensor([ 9 16 25], shape=(3,), dtype=int64)
tf.Tensor([36 49 64], shape=(3,), dtype=int64)
tf.Tensor([ 81 100 121], shape=(3,), dtype=int64)
tf.Tensor([144 169 196], shape=(3,), dtype=int64)
tf.Tensor([225 256 289], shape=(3,), dtype=int64)
tf.Tensor([324 361], shape=(2,), dtype=int64)
```

each element is a tensor of a single integer

```
4. Dataset + filter
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(4, shape=(), dtype=int64)
tf.Tensor(6, shape=(), dtype=int64)
tf.Tensor(8, shape=(), dtype=int64)
tf.Tensor(10, shape=(), dtype=int64)
tf.Tensor(12, shape=(), dtype=int64)
tf.Tensor(14, shape=(), dtype=int64)
tf.Tensor(16, shape=(), dtype=int64)
tf.Tensor(18, shape=(), dtype=int64)

5. Dataset + shuffle + batch
tf.Tensor([15 10  1], shape=(3,), dtype=int64)
tf.Tensor([13  8 19], shape=(3,), dtype=int64)
tf.Tensor([18  7  9], shape=(3,), dtype=int64)
tf.Tensor([16 11  5], shape=(3,), dtype=int64)
tf.Tensor([14  0  2], shape=(3,), dtype=int64)
tf.Tensor([ 3 12  6], shape=(3,), dtype=int64)
tf.Tensor([ 4 17], shape=(2,), dtype=int64)
```

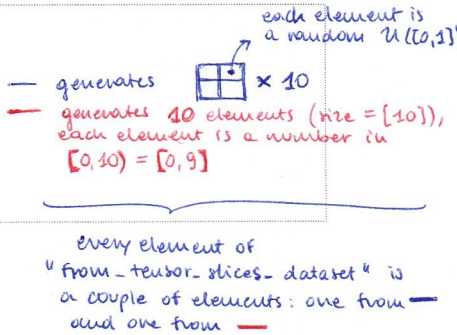
tf.data.Dataset.from_tensors

In [4]:
```
# Create Dataset as unique element
# ------------------------------------------------------------------------------------
from_tensors_dataset = tf.data.Dataset.from_tensors([1, 2, 3, 4, 5, 6, 7, 8, 9])

print("Dataset.from_tensors example:")
print("-----------------------------")
for el in from_tensors_dataset:
    print(el)
```

```
Dataset.from_tensors example:
-----------------------------
tf.Tensor([1 2 3 4 5 6 7 8 9], shape=(9,), dtype=int32)
```

tf.data.Dataset.from_tensor_slices

In [5]:
```
# Create a Dataset of slices
# ------------------------------------------------------------------------------------
# All the elements must have the same size in first dimension (axis 0)
from_tensor_slices_dataset = tf.data.Dataset.from_tensor_slices(
    (np.random.uniform(size=[10, 2, 2]), np.random.randint(10, size=[10])))

print("Dataset.from_tensor_slices example:")
print("-----------------------------")
for el in from_tensor_slices_dataset:
    print(el)
```




```
Dataset.from_tensor_slices example:
-----------------------------
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.46801031, 0.74462572],
       [0.6595736 , 0.26592613]])>, <tf.Tensor: shape=(), dtype=int32, numpy=8>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.89771532, 0.15131925],
       [0.39321184, 0.15791354]])>, <tf.Tensor: shape=(), dtype=int32, numpy=7>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.40230281, 0.01883359],
       [0.52606547, 0.67724535]])>, <tf.Tensor: shape=(), dtype=int32, numpy=0>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.59194418, 0.10261444],
       [0.54985181, 0.66751669]])>, <tf.Tensor: shape=(), dtype=int32, numpy=4>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.86016045, 0.93407796],
       [0.27190257, 0.40923308]])>, <tf.Tensor: shape=(), dtype=int32, numpy=4>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.16573255, 0.54617194],
       [0.15159143, 0.76427884]])>, <tf.Tensor: shape=(), dtype=int32, numpy=3>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.21061241, 0.98098502],
       [0.77284271, 0.02090551]])>, <tf.Tensor: shape=(), dtype=int32, numpy=0>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.78888649, 0.59296053],
       [0.12786717, 0.01551178]])>, <tf.Tensor: shape=(), dtype=int32, numpy=5>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.98894392, 0.57957078],
       [0.59907886, 0.8149864 ]])>, <tf.Tensor: shape=(), dtype=int32, numpy=8>)
(<tf.Tensor: shape=(2, 2), dtype=float64, numpy=
array([[0.47735905, 0.26010211],
       [0.9813605 , 0.61711928]])>, <tf.Tensor: shape=(), dtype=int32, numpy=1>)
```

tf.data.Dataset.zip

In [6]:
```
# Combine multiple datasets
# ------------------------------------------------------------------------------------
x = tf.data.Dataset.from_tensor_slices(np.random.uniform(size=10))
y = tf.data.Dataset.from_tensor_slices([1, 2, 3, 4, 5, 6, 7, 8, 9])

zipped = tf.data.Dataset.zip((x, y))

print("Dataset.from_tensors example:")
print("-----------------------------")
for el in zipped:
    print(el)
```

```
Dataset.from_tensors example:
-----------------------------
(<tf.Tensor: shape=(), dtype=float64, numpy=0.06927035983239826>, <tf.Tensor: shape=(), dtype=int32, numpy=1>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.32319964831333536>, <tf.Tensor: shape=(), dtype=int32, numpy=2>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.10593619871938231>, <tf.Tensor: shape=(), dtype=int32, numpy=3>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.40690838163134624>, <tf.Tensor: shape=(), dtype=int32, numpy=4>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.4248562103997421>, <tf.Tensor: shape=(), dtype=int32, numpy=5>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.2917787574122138>, <tf.Tensor: shape=(), dtype=int32, numpy=6>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.25207302164277123>, <tf.Tensor: shape=(), dtype=int32, numpy=7>)
```

```
(<tf.Tensor: shape=(), dtype=float64, numpy=0.841191551726564>, <tf.Tensor: shape=(), dtype=int32, numpy=8>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.7176242978966889>, <tf.Tensor: shape=(), dtype=int32, numpy=9>)
```

In [7]:
```
# Iterate over range dataset
# ----------------------------------------------------------------------------
# for a in b
for el in zipped:
    print(el)

print('\n')

# for a in enumerate(b)
for el_idx, el in enumerate(zipped):
    print(el)

print('\n')

# get iterator
iterator = iter(zipped)
print(next(iterator))
```

```
(<tf.Tensor: shape=(), dtype=float64, numpy=0.06927035983239826>, <tf.Tensor: shape=(), dtype=int32, numpy=1>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.32319964831333536>, <tf.Tensor: shape=(), dtype=int32, numpy=2>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.10593619871938231>, <tf.Tensor: shape=(), dtype=int32, numpy=3>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.40690838163134624>, <tf.Tensor: shape=(), dtype=int32, numpy=4>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.4248562103997421>, <tf.Tensor: shape=(), dtype=int32, numpy=5>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.2917787574122138>, <tf.Tensor: shape=(), dtype=int32, numpy=6>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.25207302164277123>, <tf.Tensor: shape=(), dtype=int32, numpy=7>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.841191551726564>, <tf.Tensor: shape=(), dtype=int32, numpy=8>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.7176242978966889>, <tf.Tensor: shape=(), dtype=int32, numpy=9>)


(<tf.Tensor: shape=(), dtype=float64, numpy=0.06927035983239826>, <tf.Tensor: shape=(), dtype=int32, numpy=1>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.32319964831333536>, <tf.Tensor: shape=(), dtype=int32, numpy=2>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.10593619871938231>, <tf.Tensor: shape=(), dtype=int32, numpy=3>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.40690838163134624>, <tf.Tensor: shape=(), dtype=int32, numpy=4>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.4248562103997421>, <tf.Tensor: shape=(), dtype=int32, numpy=5>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.2917787574122138>, <tf.Tensor: shape=(), dtype=int32, numpy=6>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.25207302164277123>, <tf.Tensor: shape=(), dtype=int32, numpy=7>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.841191551726564>, <tf.Tensor: shape=(), dtype=int32, numpy=8>)
(<tf.Tensor: shape=(), dtype=float64, numpy=0.7176242978966889>, <tf.Tensor: shape=(), dtype=int32, numpy=9>)


(<tf.Tensor: shape=(), dtype=float64, numpy=0.06927035983239826>, <tf.Tensor: shape=(), dtype=int32, numpy=1>)
```

Example: Fashion MNIST - Multi-class classification

Dataset

In [8]:
```
# Load built-in dataset
# ----------------------------------------------------------------------------
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
```

there are some datasets in tf.keras (available without download)

dataset we're choosing

```
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 [==============================] - 6s 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 [==============================] - 1s 0us/step
```

In [9]:
```
x_train.shape
y_train.shape
```

x_train = images (60K 28x28 elements)
y_train = targets = classes (60K elements)

```
Out[9]: (60000, 28, 28)

Out[9]: (60000,)
```

images and targets for testing

In [10]:
```
# Split in training and validation sets
# e.g., 50000 samples for training and 10000 samples for validation

x_valid = x_train[50000:, ...]
y_valid = y_train[50000:, ...]

x_train = x_train[:50000, ...]
y_train = y_train[:50000, ...]
```

In [11]:
```
# Create Training Dataset object
# ----------------------------------------------------------------------------
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))

# Shuffle
train_dataset = train_dataset.shuffle(buffer_size=x_train.shape[0])

# Normalize images
def normalize_img(x_, y_):
    return tf.cast(x_, tf.float32) / 255., y_
train_dataset = train_dataset.map(normalize_img)

# 1-hot encoding <- for categorical cross entropy
def to_categorical(x_, y_):
    return x_, tf.one_hot(y_, depth=10)
train_dataset = train_dataset.map(to_categorical)

# Divide in batches
bs = 32
train_dataset = train_dataset.batch(bs)

# Repeat
# Without calling the repeat function the dataset
```

we treat each datum as a couple: [image, target]

we first need to convert integers to floats

this normalizes the elements of the image (x): now every element will be $\in [0,1]$ (after the application)

because we have 10 classes

when we have more classes we cannot use labels as {0,1,2,3,..}, we need to do a transformation: 0 → [1,0,0,0..], 1 → [0,1,0,0,..],... For every label we have a 0/1 vector.

```
# will be empty after consuming all the images
train_dataset = train_dataset.repeat()
```

repeat() allows for continuous training of multiple epochs (without this the training will stop after one epoch)

In [12]:
```
# Create Validation Dataset
# ----------------------------------------------------------------------------
valid_dataset = tf.data.Dataset.from_tensor_slices((x_valid, y_valid))

# Normalize images
valid_dataset = valid_dataset.map(normalize_img)

# 1-hot encoding
valid_dataset = valid_dataset.map(to_categorical)

# Divide in batches
valid_dataset = valid_dataset.batch(1)

# Repeat
valid_dataset = valid_dataset.repeat()
```

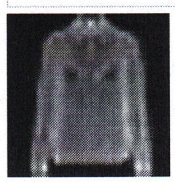
Same things as the training data

In [13]:
```
# Create Test Dataset
# ----------------------------------------------------------------------------
test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))

test_dataset = test_dataset.map(normalize_img)

test_dataset = test_dataset.map(to_categorical)

test_dataset = test_dataset.batch(1)
```

Same "

In [14]:
```
# Check that is everything is ok with the dataLoader

iterator = iter(train_dataset)
sample, target = next(iterator)

# Just for visualization purpouses
sample_ = sample[0, ...]  # select first image in the batch

from PIL import Image
img = Image.fromarray(np.uint8(np.array(sample_)*255.))
img = img.resize([128, 128])
img

sample_
target[0]  # select corresponding target
```

Out[14]:



Out[14]:
```
<tf.Tensor: shape=(28, 28), dtype=float32, numpy=
array([[0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.01176471, 0.01176471,
        0.        , 0.02352941, 0.56078434, 0.27450982, 0.29803923,
        0.27450982, 0.6117647 , 0.        , 0.        , 0.01176471,
        0.01176471, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.62352943, 0.8       , 0.52156866, 0.8980392 ,
        0.50980395, 0.8117647 , 0.54901963, 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        , 0.12156863, 0.24705882,
        0.5372549 , 0.7372549 , 0.69803923, 0.69803923, 0.77254903,
        0.69803923, 0.65882355, 0.6745098 , 0.62352943, 0.27450982,
        0.09803922, 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.4117647 , 0.6862745 , 0.6745098 ,
        0.54901963, 0.6117647 , 0.62352943, 0.62352943, 0.63529414,
        0.62352943, 0.54901963, 0.54901963, 0.57254905, 0.65882355,
        0.65882355, 0.42352942, 0.        , 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.01176471,
        0.        , 0.38431373, 0.65882355, 0.4862745 , 0.54901963,
        0.4745098 , 0.4745098 , 0.6       , 0.34901962, 0.50980395,
        0.29803923, 0.49803922, 0.4745098 , 0.42352942, 0.54901963,
        0.4862745 , 0.65882355, 0.38431373, 0.        , 0.01176471,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.57254905, 0.54901963, 0.50980395, 0.52156866,
        0.54901963, 0.43529412, 0.34901962, 0.36078432, 0.6       ,
        0.38431373, 0.36078432, 0.42352942, 0.50980395, 0.6       ,
        0.52156866, 0.5372549 , 0.58431375, 0.        , 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.        , 0.57254905, 0.56078434, 0.57254905, 0.58431375,
        0.57254905, 0.50980395, 0.42352942, 0.4117647 , 0.56078434,
        0.42352942, 0.43529412, 0.52156866, 0.54901963, 0.54901963,
        0.58431375, 0.58431375, 0.62352943, 0.13725491, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.10980392, 0.6117647 , 0.5372549 , 0.58431375, 0.58431375,
        0.56078434, 0.50980395, 0.42352942, 0.4117647 , 0.5372549 ,
        0.42352942, 0.4745098 , 0.49803922, 0.6117647 , 0.6117647 ,
        0.6862745 , 0.54901963, 0.6117647 , 0.21176471, 0.        ,
        0.        , 0.        , 0.        ],
       [0.        , 0.        , 0.        , 0.        , 0.        ,
        0.14901961, 0.6       , 0.5372549 , 0.6117647 , 0.6       ,
        0.58431375, 0.6       , 0.44705883, 0.42352942, 0.49803922,
        0.44705883, 0.4745098 , 0.6117647 , 0.5372549 , 0.6       ,
```

```
 0.7607843 , 0.4745098 , 0.6117647 , 0.4       , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.2       , 0.6       , 0.54901963, 0.64705884, 0.6117647 ,
 0.44705883, 0.52156866, 0.54901963, 0.49803922, 0.4745098 ,
 0.4862745 , 0.63529414, 0.58431375, 0.42352942, 0.6117647 ,
 0.7372549 , 0.45882353, 0.6117647 , 0.49803922, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.36078432, 0.6       , 0.56078434, 0.6745098 , 0.6745098 ,
 0.34901962, 0.28627452, 0.54901963, 0.57254905, 0.43529412,
 0.6862745 , 0.57254905, 0.29803923, 0.34901962, 0.65882355,
 0.7372549 , 0.44705883, 0.6       , 0.52156866, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.45882353, 0.6       , 0.5372549 , 0.69803923, 0.69803923,
 0.4117647 , 0.4117647 , 0.4745098 , 0.4745098 , 0.45882353,
 0.56078434, 0.44705883, 0.43529412, 0.42352942, 0.65882355,
 0.7372549 , 0.45882353, 0.62352943, 0.58431375, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.5372549 , 0.56078434, 0.50980395, 0.6862745 , 0.6862745 ,
 0.49803922, 0.49803922, 0.4745098 , 0.49803922, 0.45882353,
 0.4745098 , 0.4862745 , 0.54901963, 0.52156866, 0.64705884,
 0.7490196 , 0.52156866, 0.6       , 0.56078434, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.63529414, 0.57254905, 0.52156866, 0.69803923, 0.65882355,
 0.49803922, 0.4862745 , 0.45882353, 0.50980395, 0.44705883,
 0.4862745 , 0.45882353, 0.50980395, 0.52156866, 0.63529414,
 0.78431374, 0.54901963, 0.58431375, 0.63529414, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.6       , 0.57254905, 0.54901963, 0.70980394, 0.63529414,
 0.52156866, 0.49803922, 0.43529412, 0.4862745 , 0.49803922,
 0.50980395, 0.45882353, 0.49803922, 0.54901963, 0.63529414,
 0.8117647 , 0.58431375, 0.6       , 0.64705884, 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.63529414, 0.56078434, 0.50980395, 0.69803923, 0.63529414,
 0.5372549 , 0.52156866, 0.45882353, 0.4862745 , 0.52156866,
 0.52156866, 0.4862745 , 0.5372549 , 0.54901963, 0.62352943,
 0.84705883, 0.6       , 0.58431375, 0.7372549 , 0.01176471,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.6117647 , 0.56078434, 0.50980395, 0.6862745 , 0.6745098 ,
 0.54901963, 0.54901963, 0.49803922, 0.50980395, 0.50980395,
 0.50980395, 0.4862745 , 0.57254905, 0.56078434, 0.63529414,
 0.85882354, 0.62352943, 0.52156866, 0.7490196 , 0.07450981,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.13725491,
 0.70980394, 0.56078434, 0.56078434, 0.7372549 , 0.63529414,
 0.52156866, 0.54901963, 0.50980395, 0.52156866, 0.5372549 ,
 0.54901963, 0.50980395, 0.57254905, 0.57254905, 0.63529414,
 0.8235294 , 0.6       , 0.4745098 , 0.7372549 , 0.18431373,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.18431373,
 0.69803923, 0.52156866, 0.57254905, 0.7372549 , 0.6117647 ,
 0.49803922, 0.54901963, 0.52156866, 0.54901963, 0.54901963,
 0.56078434, 0.5372549 , 0.56078434, 0.56078434, 0.6117647 ,
 0.8       , 0.57254905, 0.43529412, 0.7607843 , 0.24705882,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.29803923,
 0.70980394, 0.4745098 , 0.64705884, 0.72156864, 0.62352943,
 0.49803922, 0.52156866, 0.54901963, 0.54901963, 0.56078434,
 0.52156866, 0.54901963, 0.58431375, 0.6       , 0.6       ,
 0.8352941 , 0.70980394, 0.45882353, 0.7490196 , 0.36078432,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.37254903,
 0.69803923, 0.4117647 , 0.72156864, 0.7607843 , 0.58431375,
 0.52156866, 0.5372549 , 0.54901963, 0.57254905, 0.57254905,
 0.5372549 , 0.56078434, 0.57254905, 0.57254905, 0.57254905,
 0.8352941 , 0.7607843 , 0.45882353, 0.8235294 , 0.43529412,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.4117647 ,
 0.62352943, 0.44705883, 0.7607843 , 0.7490196 , 0.6       ,
 0.5372549 , 0.5372549 , 0.5372549 , 0.6       , 0.6       ,
 0.56078434, 0.58431375, 0.58431375, 0.6       , 0.62352943,
 0.8       , 0.8235294 , 0.49803922, 0.7490196 , 0.44705883,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.54901963,
 0.6117647 , 0.57254905, 0.65882355, 0.7372549 , 0.6       ,
 0.56078434, 0.54901963, 0.5372549 , 0.6117647 , 0.6117647 ,
 0.5372549 , 0.62352943, 0.65882355, 0.6       , 0.65882355,
 0.77254903, 0.7490196 , 0.56078434, 0.7607843 , 0.63529414,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.62352943,
 0.6       , 0.6745098 , 0.58431375, 0.7372549 , 0.64705884,
 0.58431375, 0.58431375, 0.58431375, 0.6117647 , 0.6117647 ,
 0.6117647 , 0.63529414, 0.6745098 , 0.62352943, 0.69803923,
 0.77254903, 0.65882355, 0.69803923, 0.7372549 , 0.65882355,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.63529414,
 0.52156866, 0.69803923, 0.5372549 , 0.8352941 , 0.65882355,
 0.6745098 , 0.65882355, 0.65882355, 0.6745098 , 0.6745098 ,
 0.6862745 , 0.70980394, 0.7372549 , 0.70980394, 0.7490196 ,
 0.8862745 , 0.50980395, 0.72156864, 0.6745098 , 0.72156864,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.6862745 ,
 0.57254905, 0.78431374, 0.14901961, 0.7490196 , 0.58431375,
 0.58431375, 0.56078434, 0.56078434, 0.58431375, 0.57254905,
 0.56078434, 0.6       , 0.6745098 , 0.6       , 0.69803923,
 0.30980393, 0.28627452, 0.8235294 , 0.7372549 , 0.7490196 ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.77254903,
 0.63529414, 1.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.2       , 0.972549  , 0.77254903, 0.85882354,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.5372549 ,
```

```
       0.52156866, 0.70980394, 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.01176471, 0.72156864, 0.64705884, 0.6745098 ,
       0.        , 0.        , 0.        ]], dtype=float32)>
```

Out[14]: `<tf.Tensor: shape=(10,), dtype=float32, numpy=array([0., 0., 0., 0., 0., 0., 1., 0., 0., 0.], dtype=float32)>`

Model

In [15]:

```
# Fashion MNIST classification
# ------------------------------------------------------------------------------
# x: 28x28
# y: 10 classes

# Create Model
# ------------------------------------------------------------------------------
# e.g. in: 28x28 -> h: 10 units -> out: 10 units (number of classes)

# Define Input keras tensor
x = tf.keras.Input(shape=[28, 28])

# Define intermediate hidden layers and chain
flatten = tf.keras.layers.Flatten()(x)
h = tf.keras.layers.Dense(units=10, activation=tf.keras.activations.sigmoid)(flatten)

# Define output layer and chain

# Define the last fully-connected layer, which is composed by 10 neurons (the number of classes).
# Finally, the softmax activation function is applied for multiclass classification
out = tf.keras.layers.Dense(units=10, activation=tf.keras.activations.softmax)(h)

# Create Model instance defining inputs and outputs
model = tf.keras.Model(inputs=x, outputs=out) # Note: you can have a model with multiple inputs and multiple outputs
```

MODEL STRUCTURE

shape of the input

we cannot pass a bidimensional tensor to a layer ⇒ we flatten it

here we're adding a layer (a hidden layer fully connected ("dense") with 10 units)

the input is the flattened input

* 10 because 10 are the classes we want to predict

In [16]:

```
# Visualize created model as a table

# I can visualise the model I create and the weights initialisation
model.summary()

# Visualize initialized weights
model.weights
# As you can see it will show you the size of the output of your layers and the number of parameters (weights + biases)
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28)]          0
_________________________________________________________________
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 10)                7850
_________________________________________________________________
dense_1 (Dense)              (None, 10)                110
=================================================================
Total params: 7,960
Trainable params: 7,960
Non-trainable params: 0
_________________________________________________________________
```

PARAMETERS: #weights + #biases for each layer

28x28

hidden layer

output

None represent the dimension of the batch (which can be dynamic)

Out[16]:

```
[<tf.Variable 'dense/kernel:0' shape=(784, 10) dtype=float32, numpy=
 array([[-0.03036179, -0.06281489,  0.0257396 , ...,  0.03800482,
         -0.01909882,  0.05336354],
        [-0.00394604,  0.00170591, -0.0514029 , ...,  0.05030797,
          0.0382886 , -0.03114992],
        [ 0.0818864 ,  0.05883654,  0.04829731, ..., -0.05160644,
         -0.05101755,  0.05875803],
        ...,
        [ 0.05402381, -0.00915979, -0.02856489, ...,  0.04777338,
          0.05148549, -0.08004379],
        [ 0.03537666,  0.06979673, -0.05969963, ...,  0.01393002,
          0.06334465, -0.04543552],
        [ 0.03403018, -0.05196487, -0.00468538, ..., -0.01439087,
          0.05410584,  0.00825624]], dtype=float32)>,
 <tf.Variable 'dense/bias:0' shape=(10,) dtype=float32, numpy=array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=floa
t32)>,
 <tf.Variable 'dense_1/kernel:0' shape=(10, 10) dtype=float32, numpy=
 array([[ 0.10681671, -0.21565178,  0.54518306,  0.33767676, -0.09082237,
         -0.52187127,  0.12106484, -0.5191472 ,  0.5226437 , -0.21219748],
        [ 0.25416505,  0.11191785, -0.09809113, -0.3264152 ,  0.25440925,
         -0.49294007, -0.0512577 , -0.33606952,  0.24689281, -0.20821053],
        [-0.29728   , -0.32662296,  0.45996022, -0.13579571, -0.03329766,
          0.24442732, -0.16079873, -0.1618894 ,  0.42001778, -0.51496035],
        [ 0.5448096 , -0.5391234 , -0.12660852,  0.07686871, -0.23395827,
         -0.17641094,  0.03499007,  0.15497702,  0.00352871, -0.50480026],
        [ 0.28958637, -0.49887916,  0.5176666 ,  0.09415799, -0.54766876,
         -0.4372638 ,  0.3983506 ,  0.3647986 , -0.01561809,  0.33814573],
        [-0.1780693 ,  0.54150176,  0.10118383,  0.4683069 ,  0.22114873,
          0.2281444 ,  0.21671736, -0.48920813,  0.36766064, -0.00699753],
        [ 0.47458422,  0.49016976, -0.08482856,  0.20801061, -0.33792675,
         -0.30647153,  0.10554636, -0.2665343 ,  0.03095192, -0.11156419],
        [-0.1206241 ,  0.16199297,  0.31660575, -0.08503553,  0.45165688,
         -0.40501696,  0.5115924 ,  0.29534042,  0.17607939,  0.3442729 ],
        [-0.21178561, -0.28189176,  0.47168422,  0.06032556, -0.1062856 ,
          0.28818172, -0.33715773, -0.09632024, -0.41079062, -0.31406647],
        [ 0.3211699 ,  0.07407689,  0.47473812,  0.4545287 ,  0.279549  ,
         -0.35434338, -0.4861652 , -0.4469988 ,  0.02550262,  0.0224725 ]],
       dtype=float32)>,
 <tf.Variable 'dense_1/bias:0' shape=(10,) dtype=float32, numpy=array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=fl
oat32)>]
```

In [17]:

```
# Equivalent formulation
# ------------------------------------------------------------------------------
# Create model with sequential
# (uncomment to run)
# seq_model = tf.keras.Sequential()
```

this allows us to treat layers as lists

```
# seq_model.add(tf.keras.layers.Flatten(input_shape=(28, 28))) # or as a list
# seq_model.add(tf.keras.layers.Dense(units=10, activation=tf.keras.activations.sigmoid))
# seq_model.add(tf.keras.layers.Dense(units=10, activation=tf.keras.activations.softmax))
```

In [18]:
```
# seq_model.summary()
# seq_model.weights
```

Prepare the model for training

In [ ]:
```
# Optimization params
# -------------------------------------------------------------------------------------
# Before training the network we have to 'compile' the model by defining the following hyperparameters

# Loss
loss = tf.keras.losses.CategoricalCrossentropy()

# Learning rate
lr = 1e-3
optimizer = tf.keras.optimizers.Adam(learning_rate=lr)
# -------------------------------------------------------------------------------------

# Validation metrics
# -------------------------------------------------------------------------------------
# Define the metrics we want to compute during validation
# (keras will automatically evaluate them also on the training set).
# In this example we compute the accuracy, i.e., the frequency of correctly predicted classes

metrics = ['accuracy']
# -------------------------------------------------------------------------------------
```

HOW TO TRAIN THE MODEL

In [19]:
```
# Finally, we call model.compile

# Compile Model
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
```

Training

In [20]:
```
# Now we are ready to start training our network.
# This is done by calling the 'fit' function. We need to set some parameters. The main ones are:

# 1) Training set.
# 'x' and 'y' params represent the input and targets for training, respectively.
# If we have a dataloader which already provides <input, target> pairs, we have to set only the 'x' param.
# If you have a very small dataset, it could be convenient instead to set 'x' and 'y' directly with your
# numpy arrays containing all your training set (e.g., x = x_train, y = y_train in the example).

# 2) Number of epochs (how many times we want to process all the dataset).

# 3) Steps per epoch, i.e., (# training images) / (batch size)

# 4) Validation set
# 'validation_data' will be our dataloader for the validation set (also in this case we can give the numpy array directl
# 'validation_steps' is similar to the 'steps_per_epoch'. In the example it is 10000, since we have
# chosen arbitrarily a batch size of 1 sample for the validation dataset.

model.fit(x=train_dataset,  # you can give directly numpy arrays x_train
          y=None,
          epochs=10,
          steps_per_epoch=int(np.ceil(x_train.shape[0] / bs)),  # how many batches per epoch
          validation_data=valid_dataset,
          validation_steps=10000)  # number of batches in validation set

# Here we are! While the network is training keras provides useful information.
# In particular, we can see the loss and the metrics computed on training batches, and, at the end of each epoch,
# the same quantities are computed on the validation set (very useful to understand if we are overfitting..).
# But we can also inspect these and other details in a smarter way..see you next lesson!:)
```

#images / batch-size

that's the reason why later on we'll put "None" in y="..".

```
Train for 1563 steps, validate for 10000 steps
Epoch 1/10
1563/1563 [==============================] - 21s 13ms/step - loss: 1.1347 - accuracy: 0.6909 - val_loss: 0.7176 - val_ac
curacy: 0.7790
Epoch 2/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.6038 - accuracy: 0.8103 - val_loss: 0.5375 - val_ac
curacy: 0.8244
Epoch 3/10
1563/1563 [==============================] - 21s 13ms/step - loss: 0.5007 - accuracy: 0.8338 - val_loss: 0.4870 - val_ac
curacy: 0.8368
Epoch 4/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.4578 - accuracy: 0.8438 - val_loss: 0.4550 - val_ac
curacy: 0.8442
Epoch 5/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.4332 - accuracy: 0.8503 - val_loss: 0.4447 - val_ac
curacy: 0.8464
Epoch 6/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.4177 - accuracy: 0.8547 - val_loss: 0.4301 - val_ac
curacy: 0.8512
Epoch 7/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.4063 - accuracy: 0.8571 - val_loss: 0.4259 - val_ac
curacy: 0.8514
Epoch 8/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.3979 - accuracy: 0.8607 - val_loss: 0.4223 - val_ac
curacy: 0.8500
Epoch 9/10
1563/1563 [==============================] - 18s 12ms/step - loss: 0.3889 - accuracy: 0.8628 - val_loss: 0.4160 - val_ac
curacy: 0.8516
Epoch 10/10
1563/1563 [==============================] - 19s 12ms/step - loss: 0.3847 - accuracy: 0.8644 - val_loss: 0.4056 - val_ac
curacy: 0.8572
```

Out[20]: <tensorflow.python.keras.callbacks.History at 0x195b4fac488>

if one of the two is not moving anymore then we have to stop the training (since we're not learning anymore)

In [ ]: