Introduction to TensorFlow

Puteaux, Fall/Winter 2020-2021

```
##############################
##                          ##
##  Deep Learning in Python  ##
##                          ##
##############################

§2 Introduction to TensorFlow in Python

§2.1 Introduction to TensorFlow
```

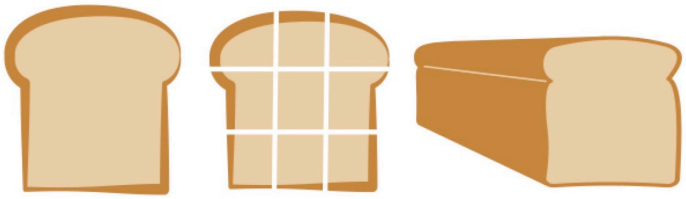
1 Constants and variables

1. What is TensorFlow?

- An open-source library for graph-based numerical computation:
  - developed by the Google Brain team
- Has both low and high-level APIs:
  - can be performed for addition, multiplication, differentiation
  - can be used to design and train machine learning models
- Important changes in TensorFlow 2.0:
  - eager execution is now available by default, which allows users to write simple and more intuitive code
  - model building is now centered around high-level APIs Keras and Estimators

2. What is a tensor?

- It is a generalization of vectors and matrices to potentially higher dimensions.
- It is a collection of numbers, which is arranged into a specific shape.



Source: Public Domain Vectors

3. Code of defining tensors in TensorFlow:

[1]:
```
import tensorflow as tf

# 0D Tensor
d0 = tf.ones((1, ))

d0
```

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

[2]:
```
# 1D Tensor
d1 = tf.ones((2, ))

d1
```

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

[3]:
```
# 2D Tensor
d2 = tf.ones((2, 2))

d2
```

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

[4]:
```
# 3D Tensor
d3 = tf.ones((2, 2, 2))

d3
```

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

       [[1., 1.],
        [1., 1.]]], dtype=float32)>
```

[5]:
```
# Print the 3D tensor
print(d3.numpy())
```

```
[[[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]]
```

4. How to define constants in TensorFlow?

- A constant is the simplest category of tensor:
  - cannot be changed and not trainable
  - can have any dimension

5. Code of defining constants in TensorFlow:

[6]:
```
from tensorflow import constant

# Define a 2x3 constant.
a = constant(3, shape=[2, 3])

a
```

[6]:
```
<tf.Tensor: shape=(2, 3), dtype=int32, numpy=
array([[3, 3, 3],
       [3, 3, 3]], dtype=int32)>
```

[7]:
```
# Define a 2x2 constant.
b = constant([1, 2, 3, 4], shape=[2, 2])

b
```

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

6. How to use convenience functions to define constants?

Operation	Example
`tf.constant()`	`constant([1, 2, 3])`
`tf.zeros()`	`zeros([2, 2])`
`tf.zeros_like()`	`zeros_like(input_tensor)`
`tf.ones()`	`ones([2, 2])`
`tf.ones_like()`	`ones_like(input_tensor)`
`tf.fill()`	`fill([3, 3], 7)`

7. Code of defining and initializing variables:

[8]:
```
import tensorflow as tf

# Define a variable
a0 = tf.Variable([1, 2, 3, 4, 5, 6], dtype=tf.float32)
a1 = tf.Variable([1, 2, 3, 4, 5, 6], dtype=tf.int16)

a0, a1
```

[8]:
```
(<tf.Variable 'Variable:0' shape=(6,) dtype=float32, numpy=array([1., 2., 3.,
4., 5., 6.], dtype=float32)>,
 <tf.Variable 'Variable:0' shape=(6,) dtype=int16, numpy=array([1, 2, 3, 4, 5,
6], dtype=int16)>)
```

[9]:
```
# Define a constant
b = tf.constant(2, tf.float32)

b
```

[9]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=2.0>
```

[10]:
```
# Compute their product
c0 = tf.multiply(a0, b)
c1 = a0 * b

c0, c1
```

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

8. Practice exercises for constants and variables:

► Package pre-loading:

[11]:
```
import pandas as pd
import numpy as np
```

▶ **Data pre-loading:**

[12]:
```
df = pd.read_csv('ref3. UCI credit card.csv', dtype=np.float64)
credit_numpy = df[['EDUCATION', 'MARRIAGE', 'AGE', 'BILL_AMT1']].to_numpy()
```

▶ **Defining data as constants practice:**

[13]:
```
# Import constant from TensorFlow
from tensorflow import constant

# Convert the credit_numpy array into a tensorflow constant
credit_constant = constant(credit_numpy)

# Print constant datatype
print('The datatype is:', credit_constant.dtype)

# Print constant shape
print('The shape is:', credit_constant.shape)
```

```
The datatype is: <dtype: 'float64'>
The shape is: (30000, 4)
```

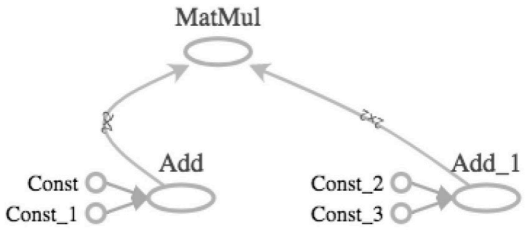
▶ **Defining variables practice:**

[14]:
```
import tensorflow as tf

# Define the 1-dimensional variable A1
A1 = tf.Variable([1, 2, 3, 4])

# Print the variable A1
print(A1)

# Convert A1 to a numpy array and assign it to B1
B1 = A1.numpy()

# Print B1
print(B1)
```

```
<tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4],
dtype=int32)>
[1 2 3 4]
```

2 Basic operations

1. What is a TensorFlow operation?



2. Code of applying the addition operator:

[15]:
```
#Import constant and add from tensorflow
from tensorflow import constant, add

# Define 0-dimensional tensors
A0 = constant([1])
B0 = constant([2])

A0, B0
```

[15]:
```
(<tf.Tensor: shape=(1,), dtype=int32, numpy=array([1], dtype=int32)>,
 <tf.Tensor: shape=(1,), dtype=int32, numpy=array([2], dtype=int32)>)
```

[16]:
```
# Define 1-dimensional tensors
A1 = constant([1, 2])
B1 = constant([3, 4])

A1, B1
```

[16]:
```
(<tf.Tensor: shape=(2,), dtype=int32, numpy=array([1, 2], dtype=int32)>,
 <tf.Tensor: shape=(2,), dtype=int32, numpy=array([3, 4], dtype=int32)>)
```

[17]:
```
# Define 2-dimensional tensors
A2 = constant([[1, 2], [3, 4]])
B2 = constant([[5, 6], [7, 8]])

A2, B2
```

[17]:
```
(<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
 array([[1, 2],
        [3, 4]], dtype=int32)>,
 <tf.Tensor: shape=(2, 2), dtype=int32, numpy=
 array([[5, 6],
        [7, 8]], dtype=int32)>)
```

[18]:
```
# Perform tensor addition with add()
C0 = add(A0, B0)
```

```
C1 = add(A1, B1)
C2 = add(A2, B2)

C0, C1, C2
```

[18]:
```
(<tf.Tensor: shape=(1,), dtype=int32, numpy=array([3], dtype=int32)>,
 <tf.Tensor: shape=(2,), dtype=int32, numpy=array([4, 6], dtype=int32)>,
 <tf.Tensor: shape=(2, 2), dtype=int32, numpy=
 array([[ 6,  8],
        [10, 12]], dtype=int32)>)
```

3. How to perform tensor addition?

- The `add()` operation performs **element-wise addition** with two tensors.
- Element-wise addition requires both tensors to have the same shape:
  - scalar addition:

    $1 + 2 = 3$

  - vector addition:

    $[1, 2] + [3, 4] = [4, 6]$

  - matrix addition:

    $$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$
- The `add()` operator is overloaded.

4. How to perform multiplication in TensorFlow?

- **Element-wise multiplication** performed using `multiply()` operation.
- The tensors multiplied must have the same shape:
  - e.g., $[1, 2, 3]$ and $[3, 4, 5]$ or $[1, 2]$ and $[3, 4]$
- **Matrix multiplication** is performed with `matmul()` operator.
  - The `matmul(A, B)` operation multiplies `A` by `B`.
  - The number of columns of `A` must equal the number of rows of `B`.

5. Code of applying the multiplication operators:

[19]:
```
# Import operators from tensorflow
from tensorflow import ones, matmul, multiply

# Define tensors
A0 = ones(1)
A31 = ones([3, 1])
A34 = ones([3, 4])
A43 = ones([4, 3])
```

```
A0, A31, A34, A43
```

[19]:
```
(<tf.Tensor: shape=(1,), dtype=float32, numpy=array([1.], dtype=float32)>,
 <tf.Tensor: shape=(3, 1), dtype=float32, numpy=
 array([[1.],
        [1.],
        [1.]], dtype=float32)>,
 <tf.Tensor: shape=(3, 4), dtype=float32, numpy=
 array([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]], dtype=float32)>,
 <tf.Tensor: shape=(4, 3), dtype=float32, numpy=
 array([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]], dtype=float32)>)
```

[20]:
```
A0_A0 = multiply(A0, A0)
A31_A31 = multiply(A31, A31)
A34_A34 = multiply(A34, A34)

A0_A0, A31_A31, A34_A34
```

[20]:
```
(<tf.Tensor: shape=(1,), dtype=float32, numpy=array([1.], dtype=float32)>,
 <tf.Tensor: shape=(3, 1), dtype=float32, numpy=
 array([[1.],
        [1.],
        [1.]], dtype=float32)>,
 <tf.Tensor: shape=(3, 4), dtype=float32, numpy=
 array([[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]], dtype=float32)>)
```

[21]:
```
A43_A34 = matmul(A43, A34)

A43_A34
```

[21]:
```
<tf.Tensor: shape=(4, 4), dtype=float32, numpy=
array([[3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.],
       [3., 3., 3., 3.]], dtype=float32)>
```

6. How to sum over tensor dimensions?

- The `reduce_sum()` operator sums over the dimensions of a:

- `tensorreduce_sum(A)` sums over all the dimensions of `A`
- `reduce_sum(A, i)` sums over the dimension `i`

7. Code of summing over tensor dimensions:

[22]:
```
# Import operations from tensorflow
from tensorflow import ones, reduce_sum

# Define a 2x3x4 tensor of ones
A = ones([2, 3, 4])

A
```

[22]:
```
<tf.Tensor: shape=(2, 3, 4), dtype=float32, numpy=
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]], dtype=float32)>
```

[23]:
```
# Sum over all dimensions
B = reduce_sum(A)

B
```

[23]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=24.0>
```

[24]:
```
# Sum over dimensions 0, 1, and 2
B0 = reduce_sum(A, 0)
B1 = reduce_sum(A, 1)
B2 = reduce_sum(A, 2)

B0, B1, B2
```

[24]:
```
(<tf.Tensor: shape=(3, 4), dtype=float32, numpy=
 array([[2., 2., 2., 2.],
        [2., 2., 2., 2.],
        [2., 2., 2., 2.]], dtype=float32)>,
 <tf.Tensor: shape=(2, 4), dtype=float32, numpy=
 array([[3., 3., 3., 3.],
        [3., 3., 3., 3.]], dtype=float32)>,
 <tf.Tensor: shape=(2, 3), dtype=float32, numpy=
 array([[4., 4., 4.],
        [4., 4., 4.]], dtype=float32)>)
```

8. Practice exercises for basic operations:

▶ **Package pre-loading:**

[25]:
```
from tensorflow import constant, ones_like, multiply
```

▶ **Element-wise multiplication performing practice:**

[26]:
```
# Define tensors A1 and A23 as constants
A1 = constant([1, 2, 3, 4])
A23 = constant([[1, 2, 3], [1, 6, 4]])

# Define B1 and B23 to have the correct shape
B1 = ones_like([1, 2, 3, 4])
B23 = ones_like([[1, 2, 3], [1, 6, 4]])

# Perform element-wise multiplication
C1 = multiply(A1, B1)
C23 = multiply(A23, B23)

# Print the tensors C1 and C23
print('C1: {}'.format(C1.numpy()))
print('C23: {}'.format(C23.numpy()))
```

```
C1: [1 2 3 4]
C23: [[1 2 3]
 [1 6 4]]
```

▶ **Package re-pre-loading:**

[27]:
```
from tensorflow import matmul
```

▶ **Matrix multiplication predictions practice:**

[28]:
```
# Define features, params, and bill as constants
features = constant([[2, 24], [2, 26], [2, 57], [1, 37]])
params = constant([[1000], [150]])
bill = constant([[3913], [2682], [8617], [64400]])

# Compute billpred using features and params
billpred = matmul(features, params)

# Compute and print the error
error = bill - billpred
print(error.numpy())
```

```
[[-1687]
 [-3218]
 [-1933]
 [57850]]
```

9. Practice question for summing over tensor dimensions:

- There is a matrix, `wealth`. This contains the value of the bond and stock wealth for five individuals in thousands of dollars.
- $\texttt{wealth} = \begin{bmatrix} 11 & 50 \\ 7 & 2 \\ 4 & 60 \\ 3 & 0 \\ 25 & 10 \end{bmatrix}$
- The first column corresponds to bonds, and the second corresponds to stocks. Each row gives the bond and stock wealth for a single individual. Use `wealth`, `reduce_sum()`, and `.numpy()` to determine which statements are correct about `wealth`.

  ☐ The individual in the first row has the highest total wealth (i.e., stocks + bonds).

  ☐ Combined, the 5 individuals hold $\$50,000$ in stocks.

  ☒ Combined, the 5 individuals hold $\$50,000$ in bonds.

  ☐ The individual in the second row has the lowest total wealth (i.e., stocks + bonds).

▶ Package pre-loading:

[29]:
```
from tensorflow import constant, reduce_sum
```

▶ Data pre-loading:

[30]:
```
wealth = constant([[11, 50], [7, 2], [4, 60], [3, 0], [25, 10]])
```

▶ Question-solving method:

[31]:
```
wealth
```

[31]:
```
<tf.Tensor: shape=(5, 2), dtype=int32, numpy=
array([[11, 50],
       [ 7,  2],
       [ 4, 60],
       [ 3,  0],
       [25, 10]], dtype=int32)>
```

[32]:
```
wealth.numpy()
```

[32]:
```
array([[11, 50],
       [ 7,  2],
       [ 4, 60],
       [ 3,  0],
       [25, 10]], dtype=int32)
```

[33]:
```
reduce_sum(wealth)
```

[33]:
```
<tf.Tensor: shape=(), dtype=int32, numpy=172>
```

[34]:
```
reduce_sum(wealth, 0)
```

[34]:
```
<tf.Tensor: shape=(2,), dtype=int32, numpy=array([ 50, 122], dtype=int32)>
```

[35]:
```
reduce_sum(wealth, 1)
```

[35]:
```
<tf.Tensor: shape=(5,), dtype=int32, numpy=array([61,  9, 64,  3, 35],
dtype=int32)>
```

3 Advanced operations

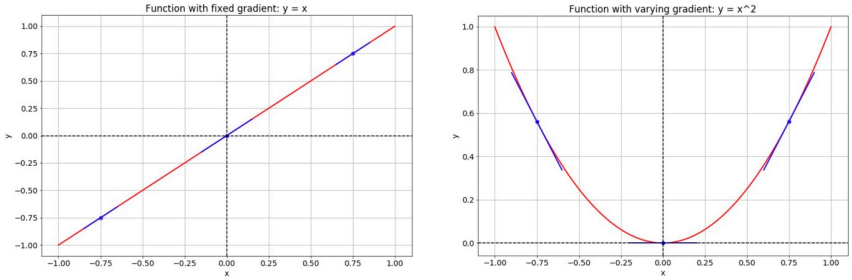
1. What are the advanced operations?

Operation	Use
`gradient()`	Computes the slope of a function at a point
`reshape()`	Reshapes a tensor (e.g. 10x10 to 100x1)
`random()`	Populates tensor with entries drawn from a probability distribution

2. How to find the optimum?

- In many problems, it is in need to find the optimum of a function:
  - **Minimum**: the lowest value of a loss function
  - **Maximum**: the highest value of the objective function
- It is possible to do this by using the `gradient()` operation:
  - **Optimum**: find a point where $gradient = 0$
  - **Minimum**: change in $gradient > 0$
  - **Maximum**: change in $gradient < 0$

3. How to calculate the gradient?



4. Code of gradients in TensorFlow:

[36]:
```
# Import tensorflow under the alias tf
import tensorflow as tf
```

```
# Define x
x = tf.Variable(-1.0)

x
```

[36]:
```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=-1.0>
```

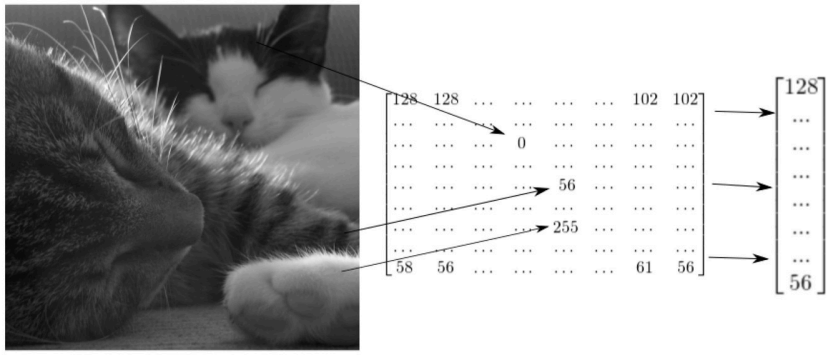
[37]:
```
# Define y within instance of GradientTape
with tf.GradientTape() as tape:
    tape.watch(x)
    y = tf.multiply(x, x)

y
```

[37]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=1.0>
```

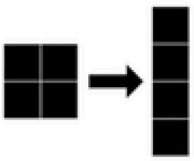
[38]:
```
# Evaluate the gradient of y at x = -1
g = tape.gradient(y, x)
print(g.numpy())
```

```
-2.0
```

5. How to deal with images as tensors?



6. How to reshape a grayscale image?



7. Code of reshaping a grayscale image:

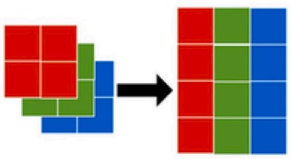
[39]:
```
# Import tensorflow as alias tf
import tensorflow as tf

# Generate grayscale image
gray = tf.random.uniform([2, 2], maxval=255, dtype='int32')

gray
```

[39]:
```
<tf.Tensor: shape=(2, 2), dtype=int32, numpy=
array([[ 59, 143],
       [ 21, 139]], dtype=int32)>
```

[40]:
```
# Reshape grayscale image
gray = tf.reshape(gray, [2 * 2, 1])

gray
```

[40]:
```
<tf.Tensor: shape=(4, 1), dtype=int32, numpy=
array([[ 59],
       [143],
       [ 21],
       [139]], dtype=int32)>
```

8. How to reshape a color image?



9. Code of reshaping a color image:

[41]:
```
# Import tensorflow as alias tf
import tensorflow as tf

# Generate color image
color = tf.random.uniform([2, 2, 3], maxval=255, dtype='int32')

color
```

[41]:
```
<tf.Tensor: shape=(2, 2, 3), dtype=int32, numpy=
array([[[174,  43,   5],
        [ 88, 221, 231]],

       [[194,  89,  70],
        [ 61, 233,  64]]], dtype=int32)>
```

[42]:
```
# Reshape color image
color = tf.reshape(color, [2 * 2, 3])

color
```

[42]:
```
<tf.Tensor: shape=(4, 3), dtype=int32, numpy=
array([[174,  43,   5],
       [ 88, 221, 231],
       [194,  89,  70],
       [ 61, 233,  64]], dtype=int32)>
```

10. Practice exercises for advanced operations:

▶ Diagram of images for reshaping:

▶ Package pre-loading:

[43]:
```
import numpy as np
from tensorflow import reshape
```

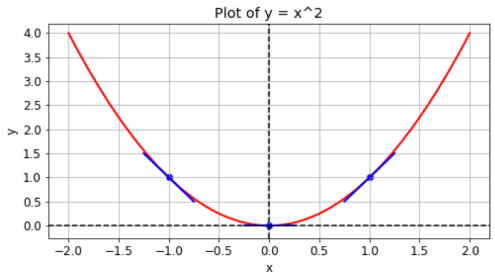
▶ Data pre-loading:

[44]:
```
gray_tensor = np.loadtxt("ref11. Gray tensor.csv", delimiter=',')

color_tensor = np.loadtxt("ref12. Color tensor.csv", delimiter=',')
color_tensor = color_tensor.reshape(color_tensor.shape[0],
                                    color_tensor.shape[1] // 3, 3)
```

▶ Tensor reshaping practice:

[45]:
```
# Reshape the grayscale image tensor into a vector
gray_vector = reshape(gray_tensor, (-1, 1))

# Reshape the color image tensor into a vector
color_vector = reshape(color_tensor, (-1, 1))
```

▶ Diagram of gradient descent:



▶ **Package re-pre-loading:**

[46]:
```
from tensorflow import Variable, GradientTape, multiply
```

▶ **Gradients optimization practice:**

[47]:
```
def compute_gradient(x0):
    # Define x as a variable with an initial value of x0
    x = Variable(x0)
    with GradientTape() as tape:
        tape.watch(x)
        # Define y using the multiply operation
        y = multiply(x, x)
    # Return the gradient of y with respect to x
    return tape.gradient(y, x).numpy()


# Compute and print gradients at x = -1, 1, and 0
print(compute_gradient(-1.0))
print(compute_gradient(1.0))
print(compute_gradient(0.0))
```

```
-2.0
2.0
0.0
```

▶ **Package re-pre-loading:**

[48]:
```
from tensorflow import constant, matmul, reduce_sum
```

▶ **Data re-pre-loading:**

[49]:
```
letter = constant([[1., 0., 1.], [1., 1., 0.], [1., 0., 1.]])

model = constant([[1., 0., -1.]])
```

▶ **Image data working practice:**

[50]:
```
# Reshape model from a 1x3 to a 3x1 tensor
model = reshape(model, (3, 1))
```

```
# Multiply letter by model
output = matmul(letter, model)

# Sum over output and print prediction using the numpy method
prediction = reduce_sum(output)
print(prediction.numpy())
```

```
1.0
```