Deep Learning - lab 2

An introduction to Numpy, Matplotlib, Scipy and TensorFlow

Prof. Stefano Carrazza

University of Milan and INFN Milan

Installing libraries

Third-party libraries can be imported easily in your code:

Numerical Python
https://numpy.org/

Visualization with Python
https://matplotlib.org/

Example:

```
pip install numpy matplotlib pandas scipy tensorflow
# Or
conda install numpy matplotlib pandas scipy tensorflow
```

Numpy basics

Arrays

Let's suppose we have the matrix (32bit floats):

$$a = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ -1 & 0 & 3 \end{pmatrix}$$

We allocate this matrix with:

```
import numpy as np
a = np.array([[1, 0, 0], [0, 1, 2], [-1, 0, 3]], dtype=np.float32)

type(a) # <class 'numpy.ndarray'>
a.shape # returns (3,3)
a.ndim  # 2
a.dtype # dtype('float32')
a.flatten() # [1, 0, 0, 0, 1, 2, -1, 0, 3]
```

Numpy provides several dtypes: int32, int64, complex32, complex64, float32, float64...

Arrays

```python
import numpy as np

# building array with zeros:
z = np.zeros(5, dtype=np.int32)
# array([0, 0, 0, 0, 0], dtype=int32)

# building array with ones
o = np.ones((5,1), dtype=np.float64)

# reshaping array
w = o.reshape(1, 5)
# array([[1., 1., 1., 1., 1.]])
```

Math operations

```
import numpy as np
a = np.array([1, 2, 3, 5])
b = np.ones(4)

c = a + b
# array([2., 3., 4., 6.])

d = c ** 2
e = 2 * np.sin(d)
# array([-1.51360499,  0.82423697, -0.57580663, -1.98355771])

e > 0
# array([False,  True, False, False])
```

Linear algebra

```python
import numpy as np

A = np.eye(2) # [[1, 0], [0, 1]]
B = np.array([[2, 1], [3, 4]])

# elementwise product
A * B # [[2, 0], [0, 4]]

# matrix product
A @ B # or A.dot(B)
# [[2, 1], [3, 4]]

# transpose matrix
B.T
# [[2, 3],[1, 4]]
```

Slicing

```
import numpy as np

C = np.array([[2, 1], [3, 4]])

m = 0
C[m, m]  # 2
C[:, m]  # [2, 3]
C[m, :]  # [2, 1]
C[m, -1] # 1
C[-1, m] # 3

C.flatten()[0]   # 2
C.flatten()[-1]  # 4
C.flatten()[0:2] # [2, 1]
```

Copy

```python
import numpy as np

a = np.array([[2, 1], [3, 4]])

# no copy
b = a # a and b are the same ndarray

# shallow copy
c = a.view()
c[0, 0] = -1 # a's data change
c = c.reshape(4) # a's shape doesn't change

# deep copy
d = a.copy()
```

Input/Output

```python
import numpy as np

# store array
a = np.array([[2, 1], [3, 4]])
np.save("myarray.npy", a)     # binary format
np.savetxt("myarray.txt", a) # or text format

# read from file
b = np.load("myarray.npy")
b = np.loadtxt("myarray.txt")
```

Matplotlib basics

Basics

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2, 100) # 100 points linearly spaced [0,2]

plt.plot(x, x**2, label="quadratic")

plt.title("Quadratic curve")
plt.xlabel("x label")
plt.ylabel("y label")
plt.legend()

plt.savefig("myplot.png")
plt.show() # keeps the plot open if script
```

Basics

`https://matplotlib.org/cheatsheets/_images/cheatsheets-1.png`

Pandas basics

Series

```python
import pandas as pd
import numpy as np

s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s)
# 0    1.0
# 1    3.0
# 2    5.0
# 3    NaN
# 4    6.0
# 5    8.0
# dtype: float64
```

Dataframes

```python
import pandas as pd
import numpy as np
df = pd.DataFrame({
        "A": 1.0,
        "B": pd.Series(1, index=list(range(4)), dtype="float32"),
        "C": np.array([3] * 4, dtype="int32"),
        "D": pd.Categorical(["test", "train", "test", "train"]),
        "E": "foo",})

# A    B   C      D    E
# 0  1.0  1.0  3   test  foo
# 1  1.0  1.0  3  train  foo
# 2  1.0  1.0  3   test  foo
# 3  1.0  1.0  3  train  foo
```

Viewing table

```
df.head()  # head of the table
df.tail(3) # tail (last 3 lines) of the table

df.index
# Int64Index([0, 1, 2, 3], dtype='int64')

df.columns
# Index(['A', 'B', 'C', 'D'], dtype='object')

df.to_numpy() # export dataframe to numpy.ndarray
df.describe() # export mean, std, min, metrics for each column
df.mean()     # computes mean
df.dropna()   # drop NaN (not a number)
df.fillna(value=1) # fill entries with specific value
```

Accessing data

```
df["A"] # returns Series (column A)

df[0:2] # returns the first 2 rows

# boolean indexing
mask = df["A"] > 1
filtered_df = df[mask]
```

Scipy basics

Non-linear least squares

```python
from scipy.optimize import curve_fit

def myfunc(x, a, b, c):
  return a * np.exp(-b * x) + c

xdata = np.load("x.dat")
ydata = np.load("y.dat")

popt, pcov = curve_fit(myfunc, xdata, ydata)
# popt: optimal values for the parameters
# pcov: estimated covariance of popt
```

Hessian approach

```python
from scipy.optimize import minimize

xdata = np.load("x.dat")
ydata = np.load("y.dat")

def myfunc(x, a, b, c):
  return a * np.exp(-b * x) + c

def myloss(x):
  return np.sum(np.square(myfunc(xdata, x[0], x[1], x[2]) - ydata))

res = minimize(myloss, x0, method="BFGS", options={"disp": True})
# res.x: the solution of the optimization
```

TensorFlow basics

TensorFlow

TensorFlow is a library for high performance numerical computation.

TensorFlow

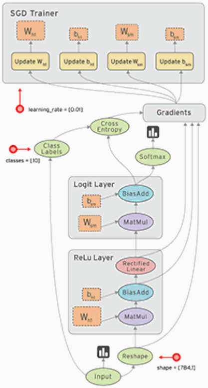
TensorFlow is a library for high performance numerical computation.

Pros:

- solves optimization problems with automatic differentiation.
- can be extended in python and c/c++.
- runs seamlessly on CPU and GPU, and can uses JIT technology.

Cons:

- do not provides builtin models from the core framework
- less automation for cross-validation and hyperparameter tune



Tensors

```python
import tensorflow as tf

x = tf.constant([1, 2, 3]) # immutable object

x.shape
x.dtype
x ** 2
tf.reduce_sum(x)

# similarly to numpy
a = tf.zeros(5, dtype=tf.float32)
b = tf.ones(5, dtype=np.float32)

# convert to numpy
c = b.numpy()
```

Variables

```python
import tensorflow as tf

var = tf.Variable([0, 0, 0]) # mutable object
var.assign([1, 1, 1])
```

Automatic differentiation

```
import tensorflow as tf

var = tf.Variable(1.0) # mutable object

def myfunc(x):
  return x ** 2 + 2 * x - 5

f(x) # <tf.Tensor: shape=(), dtype=float32, numpy=-2.0>

with tf.GradientTape() as tape:   # gradient calculation
  y = f(x)

g_x = tape.gradient(y, x) # g(x) = dy/dx = (2 * x + 2)
g_x # <tf.Tensor: shape=(), dtype=float32, numpy=4.0>
```

Graphs - performance optimization

```python
import tensorflow as tf

@tf.function
def myfunc(x):
  print("Tracing")
  return tf.reduce_sum(x)

myfunc(tf.constant([1, 2, 3]))
# Tracing
# <tf.Tensor: shape=(), dtype=int32, numpy=6>

myfunc(tf.constant([-1, 2, -1]))
# <tf.Tensor: shape=(), dtype=int32, numpy=0>

myfunc(tf.constant([10.0, 9.1, 8.2], dtype=tf.float32))
# Tracing -> <tf.Tensor: shape=(), dtype=float32, numpy=27.3>
```