Lecture 0 Python Tutorial

Fei Tan

Department of Economics
Chaifetz School of Business
Saint Louis University

E6930 Introduction to Neural Networks

July 9, 2024

Software Setup

- Working on cloud (recommended)
  - GitHub Codespace, Google Colaboratory
  - free access to hardware with preinstalled packages
- Working on local machine

```
$ python3 -m venv e6930 # virtual environment
    (isolated from system-wide packages)
$ source e6930/bin/activate
$ pip3 install numpy
$ cd e6930
$ python3 homework.py    # run script or just
    $ python3 to enter interactive mode
$ deactivate
```

The Road Ahead...

1. Built-in Functionality
2. External Packages

Basic Data Type: Numbers

```
x = 3
print(type(x)) # print "<class 'int'>"
print(x)       # print "3"
print(x + 1)   # addition; print "4"
print(x - 1)   # subtraction; print "2"
print(x * 2)   # multiplication; print "6"
print(x ** 2)  # exponentiation; print "9"
x += 1
print(x)       # print "4"
x *= 2
print(x)       # print "8"
y = 2.5
print(type(y)) # print "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # print "2.5 3.5
    5.0 6.25"
```

Basic Data Type: Booleans

```
t = True
f = False
print(type(t)) # print "<class 'bool'>"
print(t and f) # logical AND; print "False"
print(t or f)  # logical OR; print "True"
print(not t)   # logical NOT; print "False"
print(t != f)  # logical XOR; print "True"
```

Basic Data Type: Strings

```
a = 'hello'     # single quotes string literals
b = "world"     # double quotes string literals
print(a)        # print "hello"
print(len(a))   # string length; print "5"
ab = a + ' ' + b  # concatenation
print(ab)       # print "hello world"
ab12 = '%s %s %d' % (a, b, 12)  # format
print(ab12)     # print "hello world 12"
print(a.capitalize()) # capitalize; print "Hello"
print(a.upper())       # uppercase; print "HELLO"
print(a.rjust(7))      # right-justify with space
    padding; print "  hello"
print(a.center(7))     # center with space padding;
     print " hello "
print(a.replace('l', '(ell)'))  # replace all
    instances of one substring with another; print
     "he(ell)(ell)o"
print('  world '.strip())       # strip leading
    and trailing whitespace; print "world"
```

Immutables

```
x = 5            # primitives/immutables (cannot be
    modified in-place)
y = x            # deep copy/pass by value
print(y is x)    # test if x and y refer to same
    object; print "True"
y = 10           # y points to new object
print(x)         # x is not affected; print "5"
print(y is x)    # x and y refer to different
    objects; print "False"
print(id(x))     # check address of int object (on
    heap) that x points to, but no direct way to
    check address of x as reference (on stack)
print(id(y))     # x and y store different addresses
x += 1           # augmented assignment (+=, -=, *=,
     /=, etc.); same as x = x + 1
print(x)         # print "6"
print(id(x))     # x points to new object
```

Container: Lists

```
xs = [3, 1, 2]     # create list (ordered)
print(xs[2])       # print "2"
print(xs[-1])      # negative index counts from
    list end; print "2"
xs[2] = 'foo'      # can contain different types
print(xs)          # print "[3, 1, 'foo']"
xs.append('bar')  # add new element to list end
print(xs)          # print "[3, 1, 'foo', 'bar']"
x = xs.pop()       # remove and return last element
print(x)           # print "bar"
```

Container: List Slicing

```
nums = list(range(5))    # create list of integers
print(nums)              # print "[0, 1, 2, 3, 4]"
print(nums[2:4])         # slice from index 2 to 4
    (exclusive); print "[2, 3]"
print(nums[2:])          # slice from index 2 to
    end; print "[2, 3, 4]"
print(nums[:2])          # slice from start to
    index 2 (exclusive); print "[0, 1]"
print(nums[:])           # slice of whole list;
    print "[0, 1, 2, 3, 4]"
print(nums[:-1])         # slice indices can be
    negative; print "[0, 1, 2, 3]"
nums[2:4] = [8, 9]       # assign new sublist to a
    slice
print(nums)              # print "[0, 1, 8, 9, 4]"
```

Container: List Comprehension

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)  # print "cat", "dog", "monkey",
         each on its own line

nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 ==
    0]  # list comprehension
print(even_squares)  # print "[0, 4, 16]"
```

Container: Dictionaries

```
d = {'cat': 'cute', 'dog': 'furry'}  # create new
    dictionary (key-value pairs)
print(d['cat'])         # get entry; print "cute"
print('cat' in d)       # check if dictionary has a
    given key; print "True"
d['fish'] = 'wet'       # set entry
print(d['monkey'])      # KeyError: 'monkey' not a
    key of d
print(d.get('monkey', 'N/A'))  # get element with
    default; print "N/A"
print(d.get('fish', 'N/A'))    # print "wet"
del d['fish']           # remove element
print(d.get('fish', 'N/A'))    # "fish" is no
    longer a key; print "N/A"
```

Container: Dictionary Comprehension

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
# print "A person has 2 legs", "A cat has 4 legs",
     "A spider has 8 legs"

nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x
     % 2 == 0}
print(even_num_to_square)  # print "{0: 0, 2: 4,
    4: 16}"
```

Container: Sets

```
animals = {'cat', 'dog'}  # create set (unordered
    collection of distinct elements)
print('cat' in animals)   # check if element is in
     a set; print "True"
print('fish' in animals)  # print "False"
animals.add('fish')       # add element
print('fish' in animals)  # print "True"
print(len(animals))       # number of elements;
    print "3"
animals.add('cat')        # adding existing
    element does nothing
print(len(animals))       # print "3"
animals.remove('cat')     # remove element
print(len(animals))       # print "2"
```

Container: Set Comprehension

```
animals = {'cat', 'dog', 'fish'}
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
# print "#1: fish", "#2: dog", "#3: cat"

from math import sqrt
nums = {int(sqrt(x)) for x in range(30)}
print(nums)  # print "{0, 1, 2, 3, 4, 5}"
```

Container: Tuples

```
t = (5, 6)          # create tuple (immutable
    ordered list)
print(type(t))      # print "<class 'tuple'>"
d = {(x, x + 1): x for x in range(10)}  # create
    dictionary with tuple keys
print(d[t])         # print "5"
print(d[(1, 2)])  # print "1"
```

Mutables

```
num1 = [1, 2, 3, 4, 5]  # non-primitives/mutables
    (can be modified in-place)
num2 = num1       # shallow copy/pass by reference
    (indeed still pass by value of 'reference')
print(num2 is num1)      # num1 and num2 refer to
    same object; print "True"
num2[0] = 10             # also affect num1
print(num1[0])           # print "10"
print(num2 is num1)      # print "True"
print(id(num1))
print(id(num2))  # num1 and num2 are different
    references (holding same address on stack,
    stack overflow?) pointing to same object on
    heap (buffer overflow?)
num1 += [6]       # in-place augmented assignment
print(num1)       # print "[10, 2, 3, 4, 5, 6]"
print(id(num1))  # num1 points to same object
num1 = num1 + [7]        # create new object
print(id(num1))  # num1 points to new object
```

Function

```python
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob')                # print "Hello, Bob"
hello('Fred', loud=True)  # print "HELLO, FRED!"
```

Class: Encapsulation

```python
class Vehicle:  # superclass; encapsulation
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year

    def display_info(self):
        print(f"Vehicle Info: {self.year} {self.
            make} {self.model}")

    def start_engine(self):
        print("Engine started")
```

Class: Inheritance & Polymorphism

```
class Car(Vehicle):  # subclass; inheritance
    def __init__(self, make, model, year, doors):
        super().__init__(make, model, year)
        self.doors = doors

    def display_info(self):  # method overriding;
        polymorphism
        super().display_info()
        print(f"Number of doors: {self.doors}")

    def honk(self):
        print("Car honks: Beep beep!")

car = Car("Toyota", "Camry", 2020, 4)
car.display_info() # print "Vehicle Info: 2020
    Toyota Camry", "Number of doors: 4"
car.start_engine() # print "Engine started"
car.honk()         # print "Car honks: Beep beep!"
```

The Road Ahead...

NumPy: Arrays

```
import numpy as np

a = np.array([1, 2, 3], dtype=np.int64)  # create
    1-D array (neither row nor column vector)
print(type(a))                   # print "<class 'numpy.
    ndarray'>"
print(a.shape)                   # print "(3,)"
print(a[0], a[1], a[2])          # print "1 2 3"
a[0] = 5                         # in-place modification
print(a)                         # print "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])   # create 2-D
    array (including row or column vector)
print(b.shape)                           # print "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])         # print "1 2 4"
c = np.array([[1, 2, 3]])                # row vector
print(c.shape)                           # print "(1, 3)"
d = np.array([[1], [2], [3]])            # column vector
print(d.shape)                           # print "(3, 1)"
```

NumPy: Arrays (Cont'd)

```
a = np.zeros((2,2))   # array of all zeros
print(a)              # print "[[ 0.  0.]
                      #         [ 0.  0.]]"
b = np.ones((1,2))    # array of all ones
print(b)              # print "[[ 1.  1.]]"

c = np.full((2,2), 7) # constant array
print(c)              # print "[[ 7.  7.]
                      #         [ 7.  7.]]"
d = np.eye(2)         # identity matrix
print(d)              # print "[[ 1.  0.]
                      #         [ 0.  1.]]"
e = np.random.random((2,2))  # random array
print(e)         # "[[ 0.91940167  0.08143941]
                 #   [ 0.68744134  0.87236687]]"
print(e.dtype)        # print "float64"
```

NumPy: Array Indexing

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
b = a[:2, 1:3]    # slice (:) to extract subarray
print(b)          # print "[[2 3]
                  #          [6 7]]"
print(a[0, 1])    # print "2"
b[0, 0] = 77      # b[0, 0], a[0, 1] point to same
    data, so changing one will change the other
print(a[0, 1])    # print "77"

r1 = a[1, :]      # integer indexing, 1-D array
r2 = a[1:2, :]    # slice indexing, 2-D array
print(r1, r1.shape)  # print "[5 6 7 8] (4,)"
print(r2, r2.shape)  # print "[[5 6 7 8]] (1, 4)"
c1 = a[:, 1]
c2 = a[:, 1:2]
print(c1, c1.shape)  # print "[77  6 10] (3,)"
print(c2, c2.shape)  # print "[[77]
                     #          [ 6]
                     #          [10]] (3, 1)"
```

NumPy: Array Indexing (Cont'd)

```
a = np.array([[1,2], [3, 4], [5, 6]])
print(a[[0, 0], [1, 1]])  # returned array has
    shape (3,); print "[2 2]"
print(np.array([a[0, 1], a[0, 1]]))  # equivalent
b = np.array([0, 1, 0])   # index array
print(a[np.arange(3), b]) # select one element
    from each row; print "[ 1  4  5]"
a[np.arange(3), b] += 10  # mutate one element
    from each row
print(a)                 # print "array([[11,  2],
                         #                [ 3,  14],
                         #                [15,  6]])"

bool_idx = (a > 10) # find elements bigger than 10
print(bool_idx)      # print "[[ True False]
                     #          [False  True]
                     #          [ True False]]"
print(a[bool_idx])  # print "[11 14 15]"
print(a[a > 10])     # equivalent
```

NumPy: Array Math

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

print(x + y)           # elementwise sum
print(np.add(x, y))    # print "[[ 6.0  8.0]
                       #         [10.0 12.0]]"
print(x - y)           # elementwise difference
print(np.subtract(x, y))  # print "[[-4. -4.]
                          #         [-4. -4.]]"
print(x * y)           # elementwise product
print(np.multiply(x, y))  # print "[[ 5. 12.]
                          #         [21. 32.]]"
print(x / y)           # elementwise division
print(np.divide(x, y))# print "[[0.2 0.33333333]
                       #         [0.42857143 0.5]]"
print(np.sqrt(x))      # elementwise square root
                       # print "[[1. 1.41421356]
                       #         [1.73205081 2.]]"
```

NumPy: Array Math (Cont'd)

```
x = np.array([[1,2],[3,4]]) # 2-D arrays
y = np.array([[5,6],[7,8]])
v = np.array([9,10])        # 1-D arrays
w = np.array([11, 12])

print(v.dot(w))         # inner product
print(np.dot(v, w))     # print "219"
print(v @ w)            # equivalent; np.matmul(v, w)

print(x.dot(v))         # matrix-vector product
print(np.dot(x, v))     # print "[29 67]"
print(x @ v)            # equivalent; np.matmul(x, v)
print(v.dot(x))         # vector-matrix product
print(np.dot(v, x))     # print "[39 58]"
print(v @ x)            # equivalent; np.matmul(v, x)

print(x.dot(y))         # matrix-matrix product
print(np.dot(x, y))     # print "[[19 22]
                        #          [43 50]]"
print(x @ y)            # equivalent; np.matmul(x, y)
```

NumPy: Array Math (Cont'd)

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

print(np.sum(x))            # sum of all elements;
    print "10"
print(np.sum(x, axis=0))  # sum of each column;
    print "[4 6]"
print(np.sum(x, axis=1))  # sum of each row; print
     "[3 7]"

print(x.T)                  # print "[[1 3]
                            #         [2 4]]"
v = np.array([1,2,3])     # transpose of 1-D array
     does nothing
print(v.T)                  # print "[1 2 3]"
```

NumPy: Array Broadcasting

```
# Outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)
print(np.reshape(v, (3, 1)) * w)  # "[[ 4  5]
                                  #   [ 8 10]
                                  #   [12 15]]"
# Add vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])
print(x + v)           # "[[2 4 6]
                       #   [5 7 9]]"
# Add vector to each column of a matrix
print((x.T + w).T)
print(x + np.reshape(w, (2, 1)))  # "[[ 5  6  7]
                                  #   [ 9 10 11]]"
# Multiply matrix by a constant
print(x * 2)           # "[[ 2  4  6]
                       #   [ 8 10 12]]"
```

PyTorch: Tensors

```
import torch

V = torch.tensor([1, 2, 3])          # create 1-D
    tensor
print(V)                 # print "tensor([1, 2, 3])"
print(V[0])              # print "tensor(1)"
print(V[0].item())       # print "1"

M = torch.tensor([[1, 2], [3, 4]])  # create 2-D
    tensor
print(M)                 # print "tensor([[1, 2],
                         #                [3, 4]])"
print(M[0, :])           # print "tensor([1, 2])"
print(M.view(1, 4))      # reshape; print "tensor
    ([[1, 2, 3, 4]])"

W = torch.tensor([5, 6, 7])
print(V + W)             # print "tensor([6, 8, 10])"
```

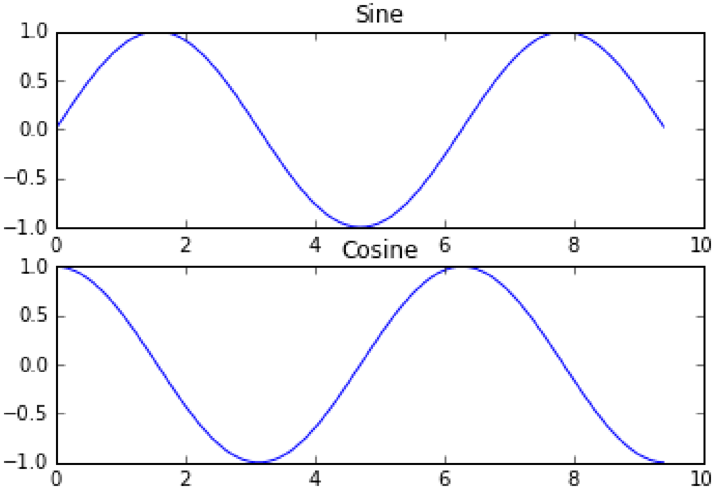
SciPy

```python
from scipy import linalg, optimize, stats

# Solve linear system of equations
A = np.array([[3, 2, 0], [1, -1, 0], [0, 5, 1]])
b = np.array([2, 4, -1])
x = linalg.solve(A, b)
print(x)             # print "[ 2. -2.  9.]"

# Optimization
def f(x):
    return x**2 + 5 * np.sin(x)

result = optimize.minimize(f, x0=0)
print(result.x)    # print "[-1.11051081]"

# Statistics
data = np.array([1, 2, 2, 3, 4, 5, 6, 7, 8, 9])
mode = stats.mode(data)
print(mode.mode)   # print "[2]"
```

Matplotlib

```python
import matplotlib.pyplot as plt

x = np.arange(0, 3 * np.pi, 0.1)  # grid points
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.subplot(2, 1, 1)  # first subplot
plt.plot(x, y_sin)
plt.title('Sine')

plt.subplot(2, 1, 2)  # second subplot
plt.plot(x, y_cos)
plt.title('Cosine')

plt.show()            # display plots
```

Matplotlib (Cont'd)



References

- cs231n.stanford.edu – CS231n: Deep Learning for Computer Vision, by Stanford University
- numpy.org – NumPy package for scientific computing
- pytorch.org – PyTorch package for deep learning
- scipy.org – SciPy package for scientific computing
- matplotlib.org – Matplotlib package for visualization