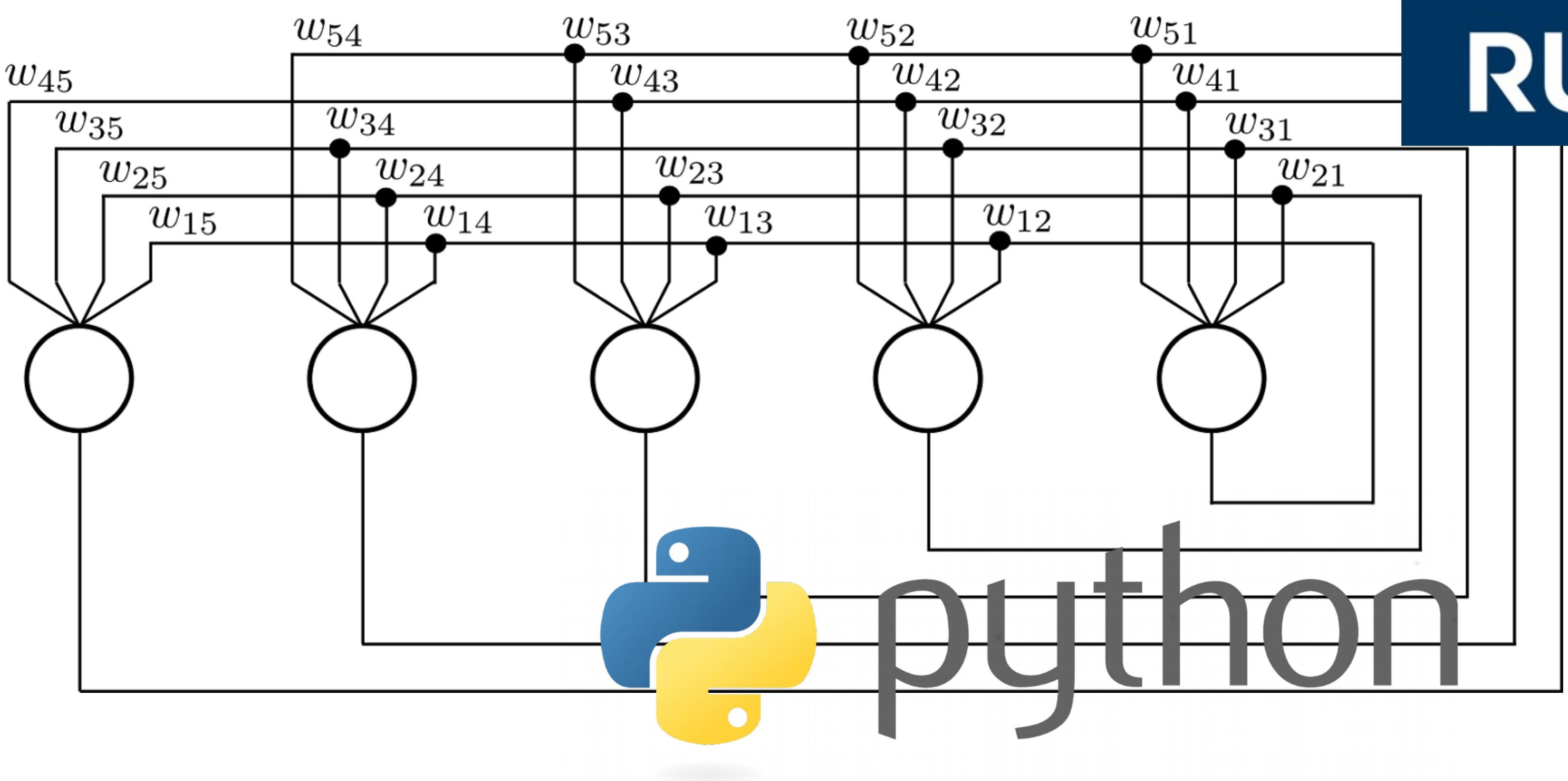
INSTITUT FÜR NEUROINFORMATIK



Artificial Neural Networks

Introduction to python

Richard Görler

WHAT'S IT ALL ABOUT?

These slides are meant as a reference. We will not go through all of them in a presentation.

Tutorial contents:

1. System setup

2. How to python
   - Operators, data types, built-in functions
   - Functions, modules and classes
   - Control structures (conditionals and loops)
   - Pythonians: args/kwargs, list comprehensions and iterators

3. Vector / matrix calculations with numpy

4. Plotting with pyplot

INSTITUT FÜR NEUROINFORMATIK
RUHR UNIVERSITÄT BOCHUM

SYSTEM SETUP

Version information

We will be using Python 3.6.8 with the following packages:

numpy 1.15.4
scipy 0.19.1
matplotlib 2.0.2
sklearn (scikit-learn) 0.19.1
tensorflow 1.12.0
keras 2.2.4

Code will most probably work on other versions as well, but we cannot guarantee

You can use the **virtual environment from Moodle**. Instructions are provided below.

Using the Anaconda environment

- Download *Miniconda* (https://docs.conda.io/en/latest/miniconda.html, Python 3.* version) or *Anaconda* (*https://www.anaconda.com/distribution/*, Python 3.* version). In order to save time and disk space, *Miniconda* is recommended.
- Install Anaconda.
  Windows: You do not need to add python to your system path in the process.
- Download the file **annvenv.yml** from Moodle
- Open a terminal (Windows: the Anaconda Prompt)
- In the terminal / Anaconda Prompt, navigate to the directory in which **annvenv.yml** is located (use `cd dir/to/file`)
- Create the environment with
  ```
  conda env create -f annvenv.yml
  ```
- Activate it with
  ```
  conda activate annvenv
  ```

Next time you only need to do the last step (activation)

INSTITUT FÜR NEUROINFORMATIK
RUHR UNIVERSITÄT BOCHUM

Alternative

Of course you can install python and the packages without using the Anaconda environment.

For instance on Linux systems:

- Use your package manager to install python
  (e.g. on Ubuntu `apt-get install python3.6`)
- Use python's package manager *pip* to install the python packages
  (`pip install [package]==[version]`)

Multiple python versions

If you already have a different version of python installed, just installing python3.6 might not work, or make package management difficult.
Consider the following options

- Use your installed versions – as long as they are newer than the ones we use in this course it will work most probably (and as long as the primary version number is identical)
- Use pyenv to manage multiple python versions. It works on linux and mac. Information on installation can be found here: https://github.com/pyenv/pyenv-installer,
  for usage see here https://github.com/pyenv/pyenv#table-of-contents and here https://github.com/pyenv/pyenv/blob/master/COMMANDS.md

IDE: Spyder

Spyder is recommended as an IDE.
Spyder 3.3.6 is included in the Anaconda environment.

HOW TO PYTHON

Operators, data types, built-in functions

Operators

Arithmetic

`+    -    *    /    //    **    %`

Relational

`==    !=    <    <=    >    >=    is`

Assignment

`=    +=    -=    /=    [...]`

Logical

`and    or    not`

Data types

are implicit in python

Numbers

int: `42` float: `42.0` (can be `inf` or `nan`)

Lists and strings

`[1,2,3]` `'Hello'` `"Hello"`

String and list concatenation

`'Hello ' + 'world'` `[1,2,3] + [4,5,6]`

Others

bool: **`True`** / **`False`** NoneType: **`None`** **`Inf, NaN`**

Accessing list elements

lst = `[1, 2, 3, 4, 'potato']`

First element / last element

```
lst[0]          lst[-1]
```

List slicing

```
lst[start, stop, step]
```

start: first element (index) to include
stop: first element (index) to exclude
step: step size

Examples

```
lst[4:1:-2]
lst[2:]
lst[:-1]
lst[::-1]
```

More on lists

```
lst1 = [1, 2, 3]
```

Test if item is contained

```
3 in lst1
4 in lst1
```

In-place manipulation: extend / append

```
lst1.append([4, 'potato'])
lst1.extend([4, 'potato'])
```

Other useful functions

```
lst1.pop()   or  lst1.pop(2)
lst1.index('potato')
```
etc.

Dictionaries

`dct = {'one': 1, 'two': 2}` or `dct = dict(one=1, two=2)`

Dictionary access

```
dct['one']
```

`dct.keys()` `dct.values()`

Note:
You can also use numbers or tuples, e.g. `(1, 'a')` as keys

Built-in functions

<u>Console output</u>

`print()`

<u>Type conversion</u>

`str()`		`int()`		`float()`

<u>Numerical</u>

`abs()`

<u>List properties</u>

`len()`		`min()`		`max()`		`sum()`

<u>Other</u>

`range(start, stop, step)`

HOW TO PYTHON

Functions, modules and classes

Scripts

File **hello.py**:

```python
print("Hello World")
```

Navigate to the directory containing your script

```
python hello.py
```

Defining functions

File **fun.py**:

```
def inc(x=0):                    (var=default value)
    y = x + 1
    return y
```

Indentation is used to define hierarchical blocks. No start / end markers.
Best practice is to use one **tab**.

Start python from the directory containing your file

```
import fun
fun.inc(1337)
fun.inc()
```

Modules

Directory structure:
fun.py
mod/__init__.py *(double underscore)*
mod/fun2.py

Script **fun2.py**:

```
def dec(x):
    y = x - 1
    return y
```

Now you can do:

```
import fun
from mod import fun2
a = fun.inc(1337)
fun2.dec(a)
```

Classes

File **potato.py**

```
class Potato():                (inheritance)

    def __init__(self, weight=100):
        self.weight = weight

    def report(self):
        print("This potato weighs " + self.weight + " grams.")
```

Now you can do:

```
import potato
p1 = potato.Potato()
p2 = potato.Potato(250)
p1.report()
p2.report()
```

HOW TO PYTHON

Control structures

if statements

```python
def sign(x):
    if x < 0:
        return -1
    elif x > 0:
        return 1
    else:
        return 0
```

for loops

```
for w in ['cat', 'window', 'defenestrate']:
    print(w)
```

or

```
for i in range(5):
     print(i)
```

enumerate can be useful

```
for i, w in enumerate(['one', 'two', 'three']):
    print(i, w)
```

while loops

```python
i = 0
while i < 10:
    print(i)
    i += 1
```

Note:
For both types of loops, the **`break`** and **`continue`** statements are available.

HOW TO PYTHON

Pythonians

*args, **kwargs

You can use pointers to lists or dictionaries to set function arguments

```
def add(x, y, z=1):
    return x + y + z

lst = [3, 4, 5]
dct = dict(x=3, a=10, y=4, z=5)
add(*lst)
add(**dct)
```

Lists provide arguments without keywords (*args)
while dicts provide arguments with keywords (**kwargs)

List comprehensions

Make a new list from a list

```
lst = list(range(10))
[abs(n-5) for n in lst]
```

We can also use conditionals

```
[abs(n-5) for n in lst if n != 5]
```

Iterators

Make an iterator from a list

```
it = iter([1, 8, 42, 1337, "over 9000"])
next(it)
next(it)
```

...

Generators work similarly, but programatically:

```
def gen():
    i = 0
    while True:
        yield i
        i += 1

g = gen()
next(g)
next(g)
...
```

MATRIX CALCULATIONS WITH NUMPY

What is numpy?

Package containing functions for vector calculations.

```
import numpy as np
```

Basic data structure: array

```
arr = np.array([2, 3, 5, 7, 11, 13, 17, 19])
arr
type(arr)
```

Access works the same as for lists

```
arr[0]
```

Number type and conversion

```
arr.dtype
arrF = arr.astype(float)
```

Defining arrays

```
arr1 = np.arange(5)
arr2 = np.zeros(5)
arr3 = np.ones(5)
```

<u>Arrays can have more than one dimension</u>

```
arr11 = np.array([[1,3,5,7,9], [2,4,6,8,10]])
arr22 = np.zeros((2,5))
arr33 = np.ones((2,5))
arrEy = np.eye(5)
```

unit matrix

<u>Shape</u>

```
arr10 = np.arange(10)
arr10.shape
arr10.reshape((2, 5))
```

Random numbers

2-by-5 matrix containing random floats between 0 and 1:

```
rand1 = np.random.rand(2,5)
```

10-element vector containing floats drawn from a "standard normal" distribution:

```
rand2 = np.random.randn(10)
```

Many more options, see the documentation!
Google: "numpy random"

Array indexing

Give indices for both dimensions

```
arr11[0, 4]
```

Access an entire column using a colon

```
arr11[:, 2]
```

Fancy indexing

Index an array with an array.

```
arr = np.array([2, 3, 5, 7, 11, 13, 17, 19])
ind = np.array([0,2,4,6])
arr[ind]
```

You can also edit elements accessed that way

```
arr[ind] = 0
```

Operations on arrays

Operators are element-wise:

```
a = np.array([[1,3,5], [-5,-7,-9]])
b = np.array([[2,4,6], [6,8,10]])
a+b
a*b
a<b
b-5
```

etc.

numpy element-wise functions (examples)

```
np.abs(a)
np.exp(b)
```

Basic data processing

```
a = np.array([[1,3,5], [-5,-7,-9]])
np.max(a)
np.min(a)
np.mean(a)
np.std(a)
```

Note that this is population std. For sample std use np.std(a, dof=1)

Select axis along which to process

```
np.mean(a, axis=0)
np.mean(a, axis=1)
```

Load and save arrays

```
np.save("a.npy", a)
a1 = np.load("a.npy")
```

np.nonzero()

Get indices of non-zero elements

```
arr0 = np.array([[0,1,2,0,3,0],[0,4,5,0,0,6]])
nnz = np.nonzero(arr0)
nnz
arr0[nnz]
```

Use it to find where condition is met

```
arr10 = np.arange(10)
nnz10 = np.nonzero(np.abs(arr10-5) > 2)
nnz10
arr10[nnz10]
```

Also useful

Number of elements that meet condition

```
np.sum(np.abs(arr10-5) > 2)
```

Compare entire arrays

```
np.array_equal(a, b)
```

Check for invalid data

```
t1 = np.isnan(a)
t2 = np.isinf(b)
```

any / all

```
np.any(t1)
np.all(t2)
```

Inner and outer product

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^{\mathrm{T}} \mathbf{b} = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n = \sum^{n} a_i b_i$$

(inner product)

$$\mathbf{u} \otimes \mathbf{v} = \mathbf{u}\mathbf{v}^{\mathrm{T}} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} = \begin{bmatrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_2 & u_2 v_3 \\ u_3 v_1 & u_3 v_2 & u_3 v_3 \\ u_4 v_1 & u_4 v_2 & u_4 v_3 \end{bmatrix}$$

(outer product)

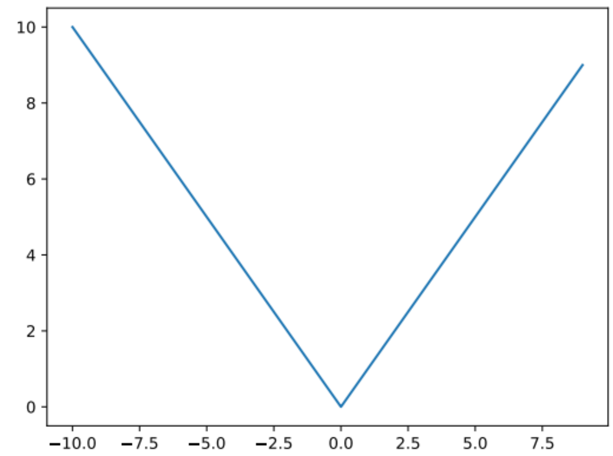
`np.inner()` / `np.dot()` and `np.outer()`

For matrix multiplication, use `np.matmul(A,B)` or `A @ B`

PLOTTING WITH PYPLOT

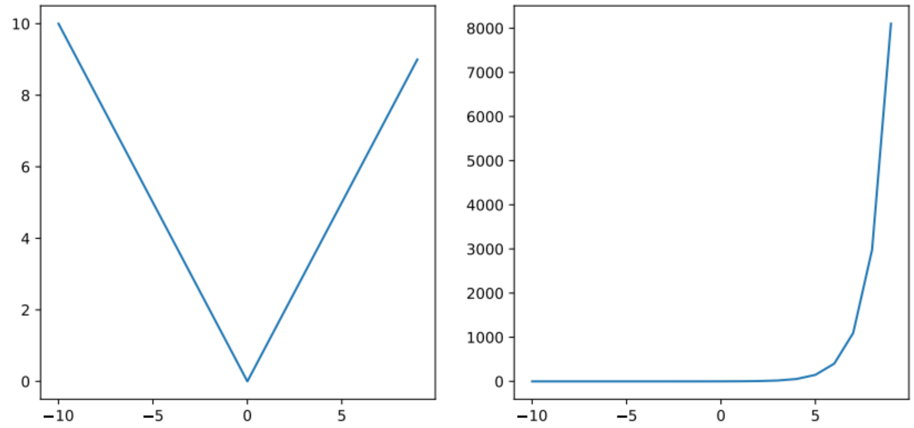
Plot

```
import matplotlib.pyplot as plt
x = np.arange(20)-10
y = np.abs(x)
plt.plot(x,y)
plt.show()
```



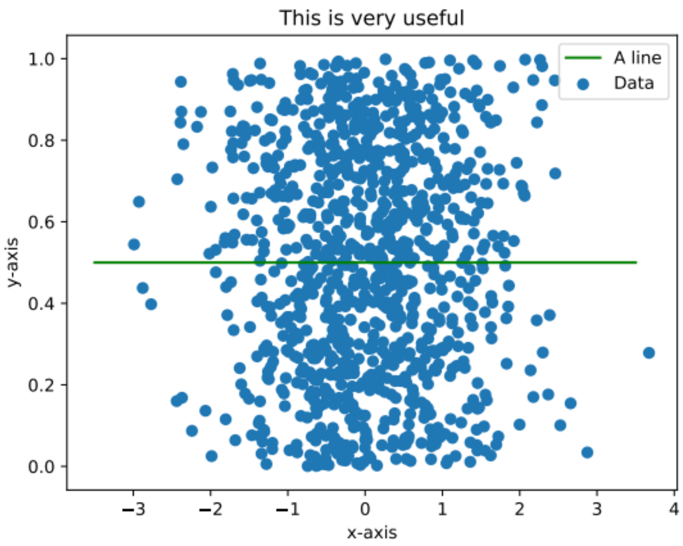
Subplot

```
plt.subplot(1,2,1)
plt.plot(x,y)
y2 = np.exp(x)
plt.subplot(1,2,2)
plt.plot(x,y2)
plt.show()
```



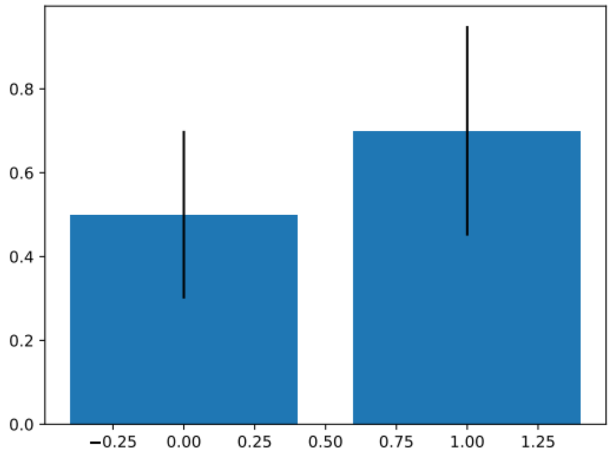
Scatter (and more...)

```
xr = np.random.randn(1000)
yr = np.random.rand(1000)
plt.scatter(xr,yr,label="Data")

plt.plot([-3.5, 3.5], [0.5, 0.5],
   color='g', label="A line")

plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("This is very useful")

plt.legend()

plt.savefig("example.pdf")
plt.show()
```



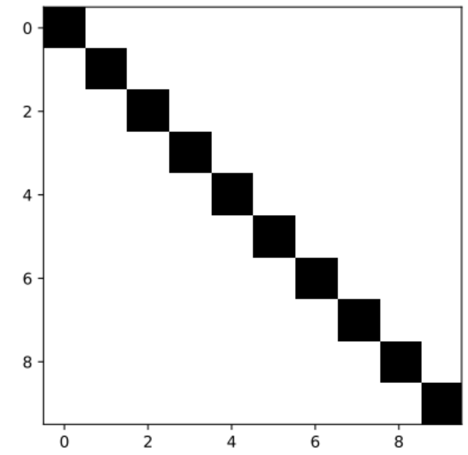
Bar

```
x = [0, 1]
y = [0.5, 0.7]
std = [0.2, 0.25]
plt.bar(x, y, width=0.8, yerr=std)
plt.show()
```



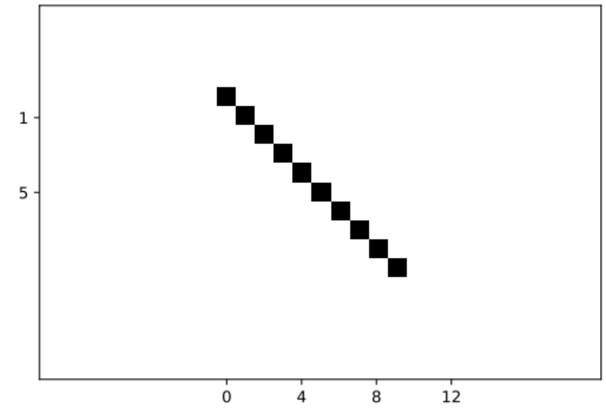
Imshow

```
m = np.eye(10)
plt.imshow(m, interpolation='none', cmap="Greys")
plt.show()
```

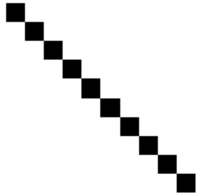


More options

```
plt.imshow(m, interpolation='none',
                       cmap="Greys")
plt.xlim([-10,20])
plt.ylim([15,-5])
plt.xticks([0,4,8,12])
plt.yticks([1,5])
plt.show()
```




```
plt.axis('off')
```



DONE!