Computer Vision II

Numpy/PyTorch Introduction

05.06.2019

TECHNISCHE UNIVERSITÄT DARMSTADT

visual inference

Why Numpy?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Fundamental package for scientific computing with Python
  - Powerful **N-dimensional array** objects (`ndarray`)
  - Sophisticated broadcasting (similar, but more intuitive than Julia)
  - Tools for integrating C/C++ code
  - Linear algebra, Fourier transforms, random number generation

- Shared by Python deep learning framework to **communicate with CPU hosts**
  - Cuda/GPU computations –> TensorFlow/PyTorch/MXNet datatypes
  - Transfer between GPU and CPU –> Numpy.ndarray

```
import numpy as np
```

Creating Vectors

MATLAB	PYTHON	JULIA
Row vector: size (1, n)		
`A = [1 2 3]`	`A = np.array([1, 2, 3]).reshape(1, 3)`	`A = [1 2 3]`
Column vector: size (n, 1)		
`A = [1; 2; 3]`	`A = np.array([1, 2, 3]).reshape(3, 1)`	`A = [1 2 3]'`
1d array: size (n,)		

Not possible	`A = np.array([1, 2, 3])`	`A = [1; 2; 3]`
		or
		`A = [1, 2, 3]`

Integers from j to n with step size k		
`A = j:k:n`	`A = np.arange(j, n+1, k)`	`A = j:k:n`
Linearly spaced vector of k points		
`A = linspace(1, 5, k)`	`A = np.linspace(1, 5, k)`	`A = range(1, 5, length = k)`

Creating Matrices

MATLAB	PYTHON	JULIA
Create a matrix		
`A = [1 2; 3 4]`	`A = np.array([[1, 2], [3, 4]])`	`A = [1 2; 3 4]`
2 x 2 matrix of zeros		
`A = zeros(2, 2)`	`A = np.zeros((2, 2))`	`A = zeros(2, 2)`
2 x 2 matrix of ones		
`A = ones(2, 2)`	`A = np.ones((2, 2))`	`A = ones(2, 2)`
2 x 2 identity matrix		

`A = eye(2, 2)`	`A = np.eye(2)`	`A = I # will adopt`
		`# 2x2 dims if demanded by`
		`# neighboring matrices`

Diagonal matrix		
`A = diag([1 2 3])`	`A = np.diag([1, 2, 3])`	`A = Diagonal([1, 2,` `3])`
Uniform random numbers		
`A = rand(2, 2)`	`A = np.random.rand(2, 2)`	`A = rand(2, 2)`

Accessing Vector/Matrix Elements

MATLAB	PYTHON	JULIA
Access one element		
`A(2, 2)`	`A[1, 1]`	`A[2, 2]`
Access specific rows		
`A(1:4, :)`	`A[0:4, :]`	`A[1:4, :]`
Access specific columns		
`A(:, 1:4)`	`A[:, 0:4]`	`A[:, 1:4]`
Remove a row		
`A([1 2 4], :)`	`A[[0, 1, 3], :]`	`A[[1, 2, 4], :]`
Diagonals of matrix		
`diag(A)`	`np.diag(A)`	`diag(A)`
Get dimensions of matrix		
`[nrow ncol] = size(A)`	`nrow, ncol = np.shape(A)`	`nrow, ncol = size(A)`

- In Python/Numpy indices start with 0 !!!

Manipulating Vectors and Matrices

Transpose

MATLAB	PYTHON	JULIA
`A.'`	`A.T`	`transpose(A)`

Complex conjugate transpose (Adjoint)

MATLAB	PYTHON	JULIA
`A'`	`A.conj()`	`A'`

Concatenate horizontally

MATLAB	PYTHON	JULIA
`A = [[1 2] [1 2]]`	`B = np.array([1, 2])` `A = np.hstack((B, B))`	`A = [[1 2] [1 2]]`
or		or
`A = horzcat([1 2], [1 2])`		`A = hcat([1 2], [1 2])`

Concatenate vertically

MATLAB	PYTHON	JULIA
`A = [[1 2]; [1 2]]`	`B = np.array([1, 2])` `A = np.vstack((B, B))`	`A = [[1 2]; [1 2]]`
or		or
`A = vertcat([1 2], [1 2])`		`A = vcat([1 2], [1 2])`

Reshape (to 5 rows, 2 columns)

MATLAB	PYTHON	JULIA
`A = reshape(1:10, 5, 2)`	`A = A.reshape(5, 2)`	`A = reshape(1:10, 5, 2)`

Convert matrix to vector

MATLAB	PYTHON	JULIA
`A(:)`	`A = A.flatten()`	`A[:]`

Programming

Comment one line

MATLAB
```
% This is a comment
```

PYTHON
```
# This is a comment
```

JULIA
```
# This is a comment
```

Comment block

MATLAB
```
%{
Comment block
%}
```

PYTHON
```
# Block
# comment
# following PEP8
```

JULIA
```
#=
Comment block
=#
```

For loop

MATLAB
```
for i = 1:N
    % do something
end
```

PYTHON
```
for i in range(n):
    # do something
```

JULIA
```
for i in 1:N
    # do something
end
```

While loop

MATLAB
```
while i <= N
    % do something
end
```

PYTHON
```
while i <= N:
    # do something
```

JULIA
```
while i <= N
    # do something
end
```

If

MATLAB
```
if i <= N
    % do something
end
```

PYTHON
```
if i <= N:
    # do something
```

JULIA
```
if i <= N
    # do something
end
```

If / else

MATLAB
```
if i <= N
    % do something
else
    % do something else
end
```

PYTHON
```
if i <= N:
    # do something
else:
    # so something else
```

JULIA
```
if i <= N
    # do something
else
    # do something else
end
```

Print text and variable

```
x = 10
fprintf('x = %d \n', x)
```

```
x = 10
print(f'x = {x}')
```

```
x = 10
println("x = $x")
```

Function: anonymous

```
f = @(x) x^2
```

```
f = lambda x: x**2
```

```
f = x -> x^2
# can be rebound
```

Function

```
function out  = f(x)
   out = x^2
end
```

```
def f(x):
    return x**2
```

```
function f(x)
   return x^2
end

f(x) = x^2 # not anon!
```

Tuples

```
t = {1 2.0 "test"}
t{1}
```

```
t = (1, 2.0, "test")
t[0]
```

```
t = (1, 2.0, "test")
t[1]
```

Can use cells but watch performance

Closures

```
a = 2.0
f = @(x) a + x
f(1.0)
```

```
a = 2.0
def f(x):
    return a + x
f(1.0)
```

```
a = 2.0
f(x) = a + x
f(1.0)
```

Inplace Modification

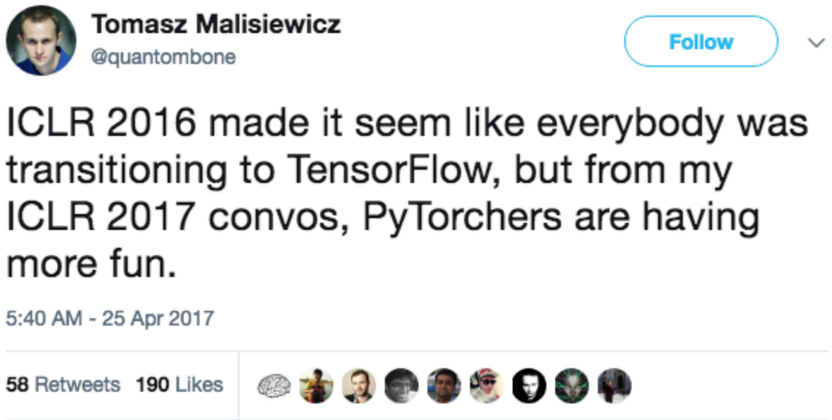
```
function f(out, x)
     out = x.^2
end
x = rand(10)
y = zeros(length(x), 1)
f(y, x)
```

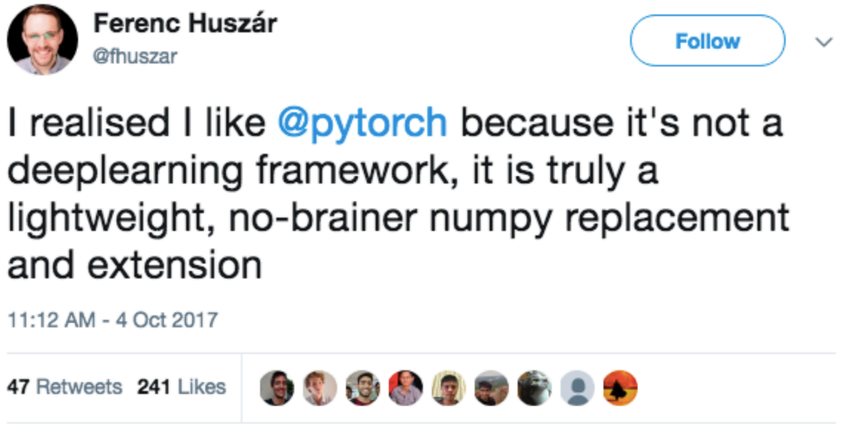
```
def f(x):
    x **=2
    return

x = np.random.rand(10)
f(x)
```

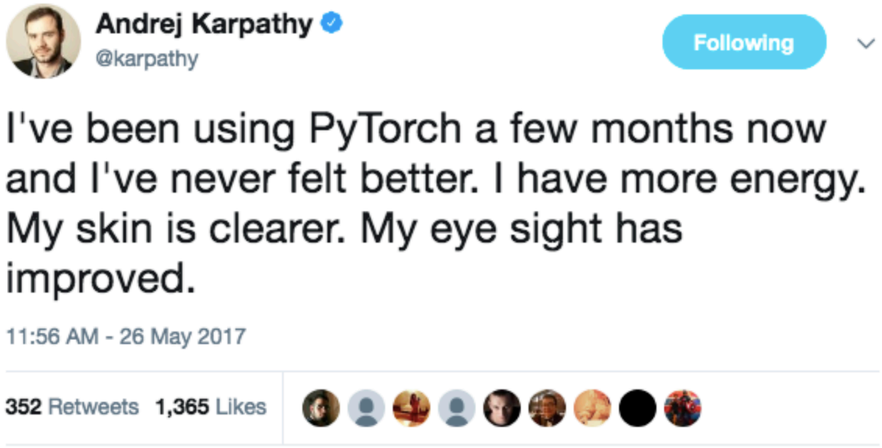
```
function f!(out, x)
    out .= x.^2
end
x = rand(10)
y = similar(x)
f!(y, x)
```

Why PyTorch?

TECHNISCHE UNIVERSITÄT DARMSTADT







Goals of deep learning frameworks

- Easily build **big computational graphs**
- Easily **compute gradients** in computational graphs
  - PyTorch utilizes dynamic computation graphs
- Run it all **efficiently on GPU**
  - Wrap thirdparty accelerators such as cuDNN, cuBLAS, etc

PyTorch ~ Python/Numpy

TECHNISCHE UNIVERSITÄT DARMSTADT

- PyTorch is engineered to be fairly close to Numpy
  - Data types, operation names, arguments etc

Array creation

Numpy	PyTorch	Notes
`np.empty((2, 2))`	`torch.empty(5, 3)`	empty array
`np.random.rand(3,2)`	`torch.rand(5, 3)`	random
`np.zeros((5,3))`	`torch.zeros(5, 3)`	zeros
`np.array([5.3, 3])`	`torch.tensor([5.3, 3])`	from list
`np.random.randn(*a.shape)`	`torch.randn_like(a)`	
`np.arange(16)`	`torch.range(0,15)`	array starting from 0 ending at 15 (inclusive)

Math operations

Numpy	PyTorch	Notes

`x+y`	`x+y`
	`y.add_(x)`
	`torch.add(x,y)`
		addition

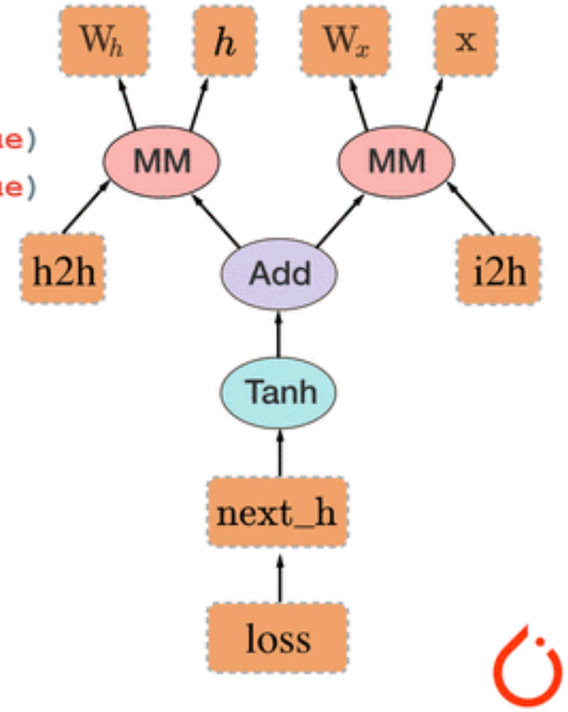
`np.dot(x,y)` `np.matmul(x,y)`	`torch.mm(x,y)` `x.mm(y)`	matrix multiplication
`x*y`	`x*y`	element-wise multiplication
`np.max(x)`	`torch.max(x)`	
`np.argmax(x)`	`torch.argmax(x)`	
`x**2`	`x**2`	Element-wise powers

What is a dynamic computation graph?

Back-propagation

uses the dynamically created graph

```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
next_h = next_h.tanh()

loss = next_h.sum()
loss.backward()  # compute gradients!
```



How does this work?

TECHNISCHE UNIVERSITÄT DARMSTADT

- The key is **automatic differentiation** (Autograd or AD)
- Since we can easily derive derivatives for elementary arithmetic operations – we can also obtain the derivatives for **more complex expressions constructed from elementary operations**
- The mechanics behind this root in the repeated **application of the chain rule**
- Let's look at an example…

Automatic differentiation – An example

TECHNISCHE UNIVERSITÄT DARMSTADT

- A given scalar function:

$$f = \exp\left(\exp(x) + \exp(x)^2\right) + \sin\left(\exp(x) + \exp(x)^2\right)$$

- Let's write down the derivative of this:

$$\begin{aligned}\frac{df}{dx} = \quad & \exp\left(\exp(x) + \exp(x)^2\right)\left(\exp(x) + 2\exp(x)^2\right) \\ & + \cos\left(\exp(x) + \exp(x)^2\right)\left(\exp(x) + 2\exp(x)^2\right)\end{aligned}$$

Automatic differentiation – An example

TECHNISCHE UNIVERSITÄT DARMSTADT

- A given scalar function:

$$f = \exp\left(\exp(x) + \exp(x)^2\right) + \sin\left(\exp(x) + \exp(x)^2\right)$$

- Alternative way is to “program” the expression:

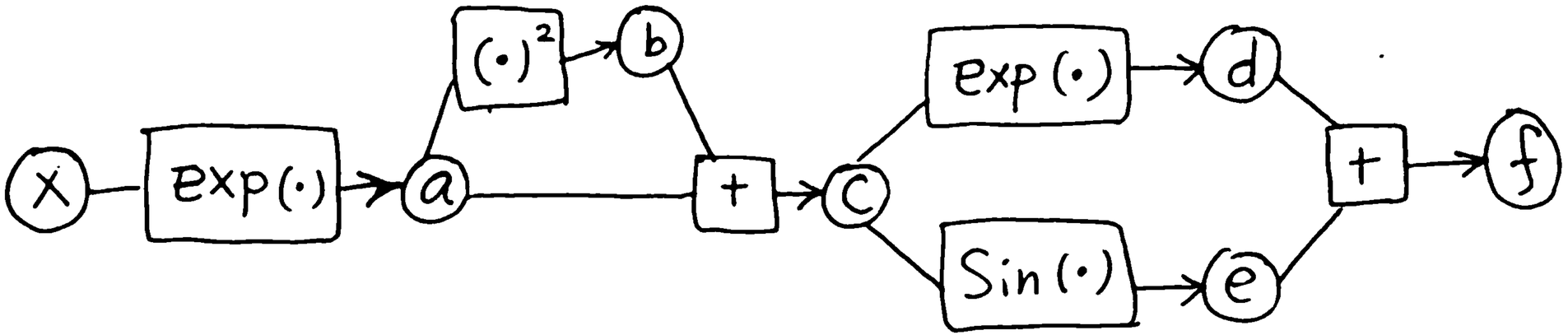
$$\begin{aligned} a &= \exp(x) \\ b &= a^2 \\ c &= a + b \\ d &= \exp(c) \\ e &= \sin(c) \\ f &= d + e \end{aligned}$$

Automatic differentiation – An example

$$f = \exp\left(\exp(x) + \exp(x)^2\right) + \sin\left(\exp(x) + \exp(x)^2\right)$$

$$a = \exp(x)$$
$$b = a^2$$
$$c = a + b$$
$$d = \exp(c)$$
$$e = \sin(c)$$
$$f = d + e$$

- Visualizing the program



Automatic differentiation – An example

- Derive by chaining derivatives of subexpressions

$$a = \exp(x)$$
$$b = a^2$$
$$c = a + b$$
$$d = \exp(c)$$
$$e = \sin(c)$$
$$f = d + e$$

Chain rules

$$\frac{df}{dd} = 1$$
$$\frac{df}{de} = 1$$
$$\frac{df}{dc} = \frac{df}{dd}\frac{dd}{dc} + \frac{df}{de}\frac{de}{dc}$$
$$\frac{df}{db} = \frac{df}{dc}\frac{dc}{db}$$
$$\frac{df}{da} = \frac{df}{dc}\frac{dc}{da} + \frac{df}{db}\frac{db}{da}$$
$$\frac{df}{dx} = \frac{df}{da}\frac{da}{dx}$$

→

Chain rules

$$\frac{df}{dd} = 1$$
$$\frac{df}{de} = 1$$
$$\frac{df}{dc} = \frac{df}{dd}\exp(c) + \frac{df}{de}\cos(c)$$
$$\frac{df}{db} = \frac{df}{dc}$$
$$\frac{df}{da} = \frac{df}{dc} + \frac{df}{db}2a$$
$$\frac{df}{dx} = \frac{df}{da}\exp(x).$$

← Final expression

Automatic differentiation

TECHNISCHE UNIVERSITÄT DARMSTADT

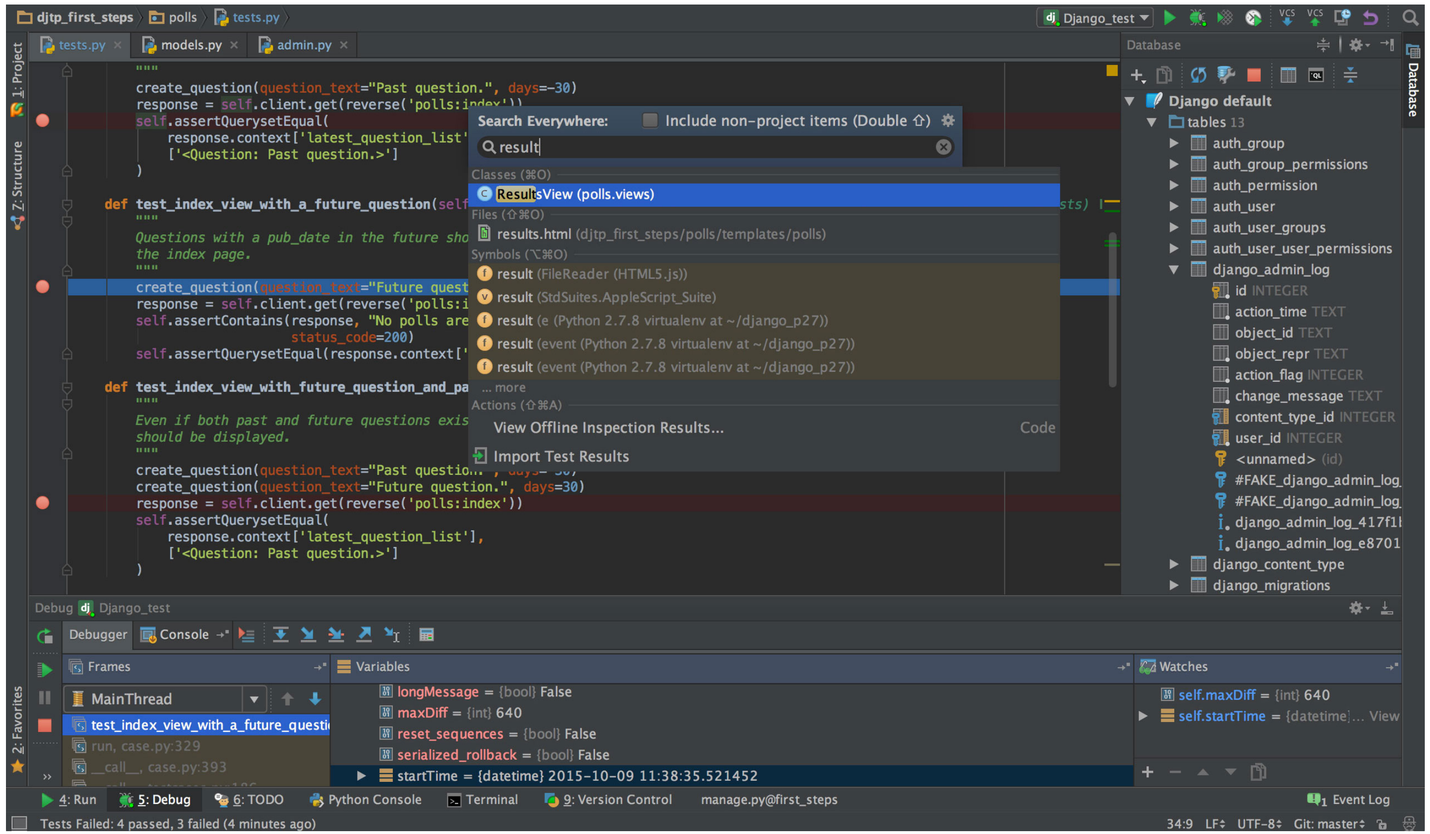
- Works for any differential function that can be phrased as an expression graph
- Most important:
  - **Abstracts away difficulties** in gradient computations
  - In turn, this frees you up to **think about the forward computation** and problem at hand

- Pitfalls:
  - Fails if program is not differentiable
  - Difficult to handle if-else blocks
  - Can get rather complicated for vector and matrix expressions
    - e.g. how to compute the derivative of $\boldsymbol{f} = \boldsymbol{A}^{-1}\boldsymbol{x}$?

Developing with Python/PyTorch

TECHNISCHE UNIVERSITÄT DARMSTADT

- Most Popular Python IDE: https://www.jetbrains.com/pycharm/
  - Code completions
  - Code inspections
  - Visual debugger



- It is usually easiest to tell PyCharm to use an existing Python distribution!

How to install Numpy and PyTorch?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Install an Anaconda Python3 distribution from:
  - https://www.anaconda.com/distribution/
  - Includes comprehensive set of scientific packages (Numpy, Matplotlib, etc)

- Follow instructions form PyTorch webpage:
  - https://pytorch.org/get-started/locally/

- This usually just means running:
  - Linux: `$conda install pytorch-cpu torchvision-cpu -c pytorch`
  - Windows: `$conda install pytorch-cpu torchvision-cpu -c pytorch`
  - MacOS: `$conda install pytorch torchvision -c pytorch`

Helpful links

TECHNISCHE UNIVERSITÄT DARMSTADT

- Good Numpy tutorial:
  - http://cs231n.github.io/python-numpy-tutorial/
- PyTorch homepage:
  - https://pytorch.org/
- Get started guide
  - https://pytorch.org/get-started/locally/
- PyTorch Tutorials
  - https://pytorch.org/tutorials/
- PyTorch API:
  - https://pytorch.org/docs/stable/index.html

We will now look into the Tutorial notebooks

TECHNISCHE UNIVERSITÄT DARMSTADT

- Numpy Tutorial:
  - http://cs231n.github.io/python-numpy-tutorial/
- What is PyTorch Tutorial
  - https://pytorch.org/tutorials/beginner/blitz/tensor_tutorial.html#sphx-glr-beginner-blitz-tensor-tutorial-py
- Autograd Tutorial:
  - https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html#sphx-glr-beginner-blitz-autograd-tutorial-pyet started guide

References

TECHNISCHE UNIVERSITÄT DARMSTADT

- These slides have been compiled from a variety of resources:
  - https://cheatsheets.quantecon.org
  - https://people.cs.umass.edu/~domke/courses/sml/09autodiff_nnets.pdf
  - https://cise.ufl.edu/~xiaoyong/materials/pytorch_tutorial.pdf
  - https://mldublin.github.io/assets/slides/meetup_19/Intro_to_pytorch.html