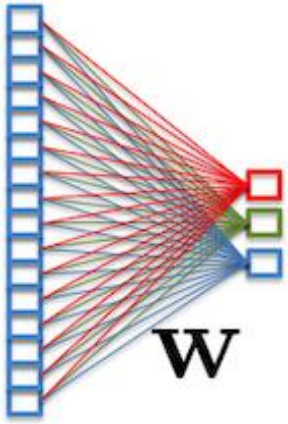
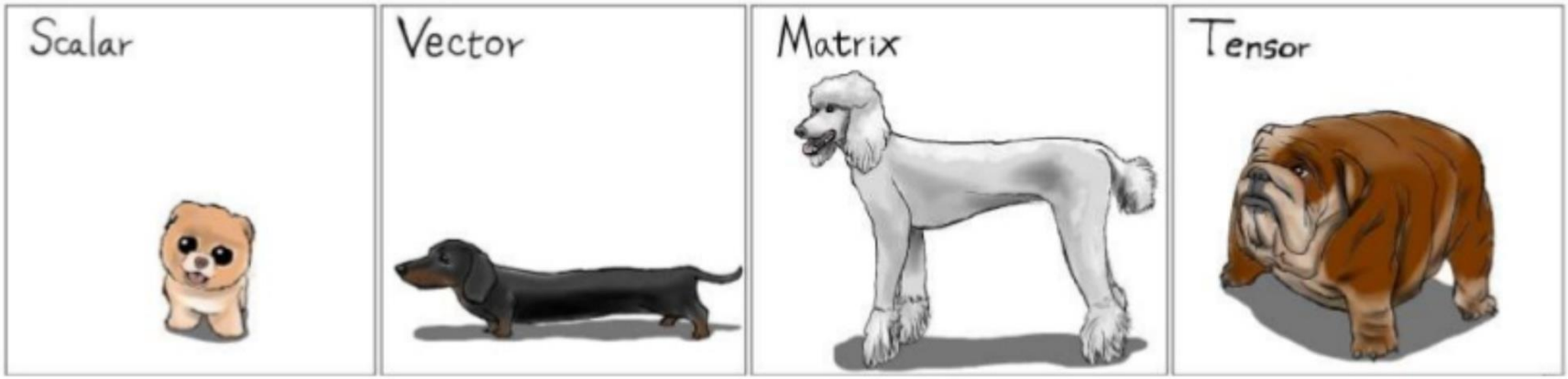
AI and Data Analysis

2.1 Frameworks and tensors



Christian Wolf

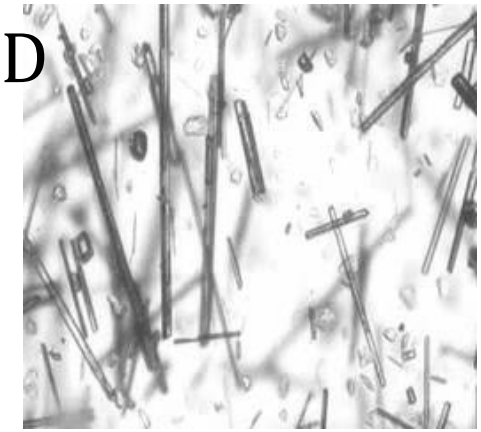
We manipulate tensors

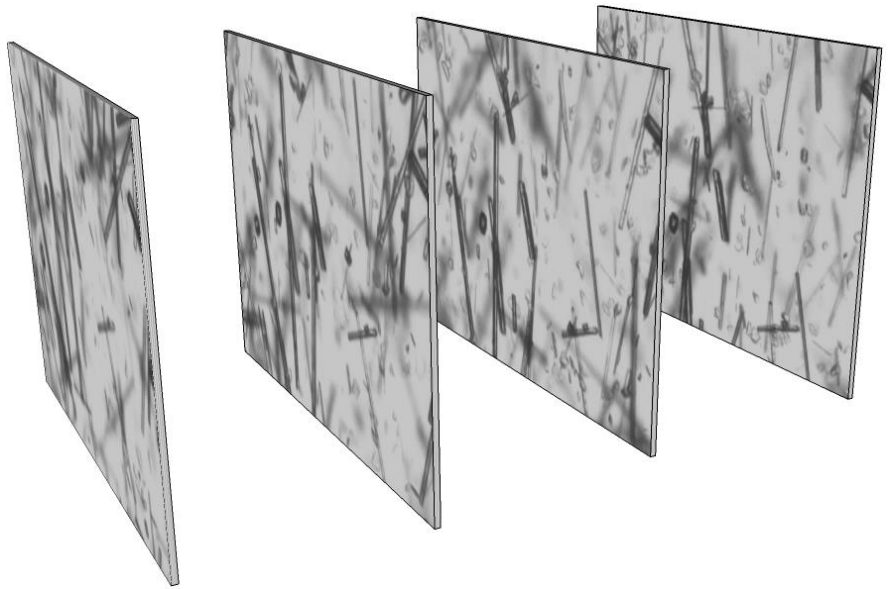


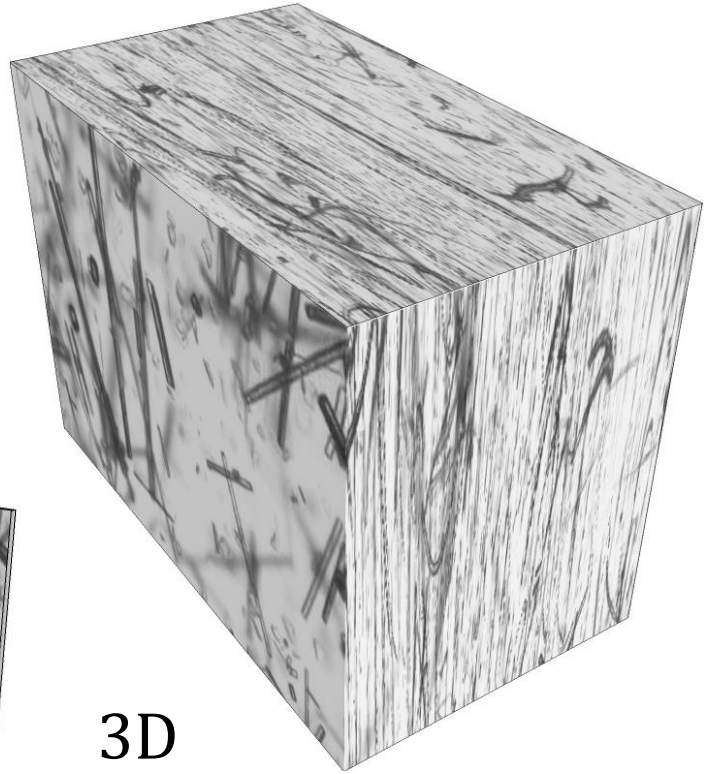
[Figure: Anima Anandkumar]

High dimensional tensors

2D





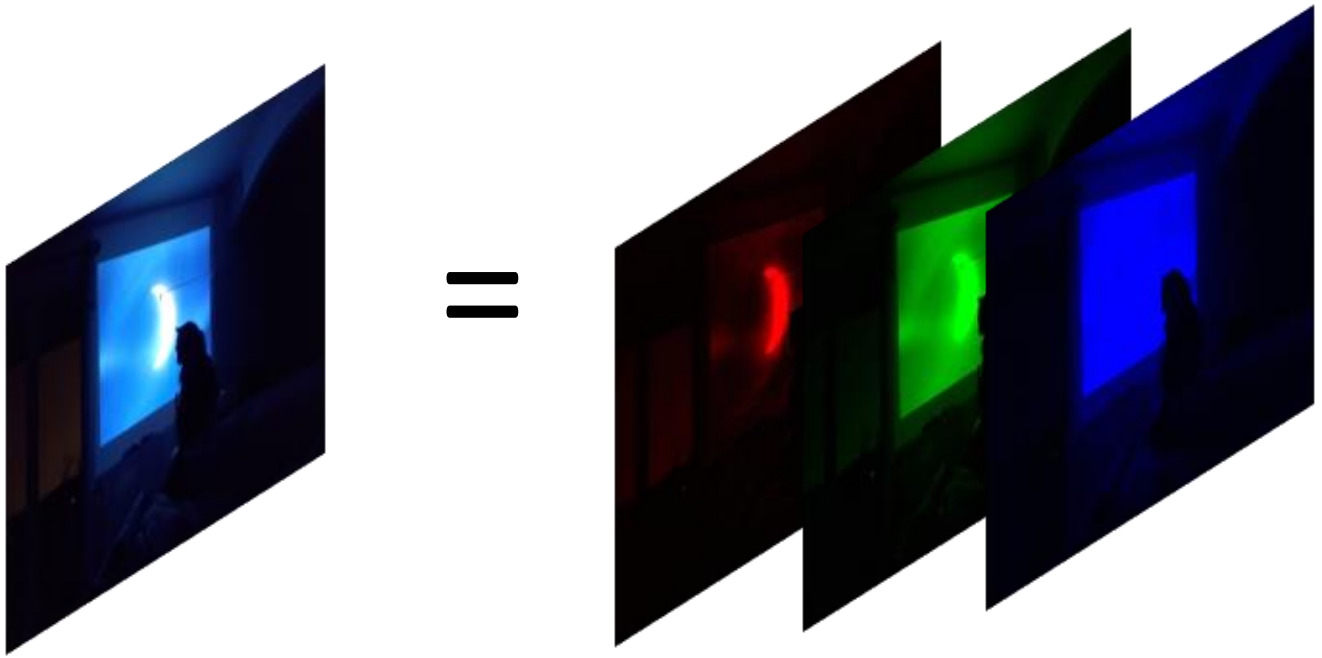


3D
(2D+t)

[Data: LAGEP]

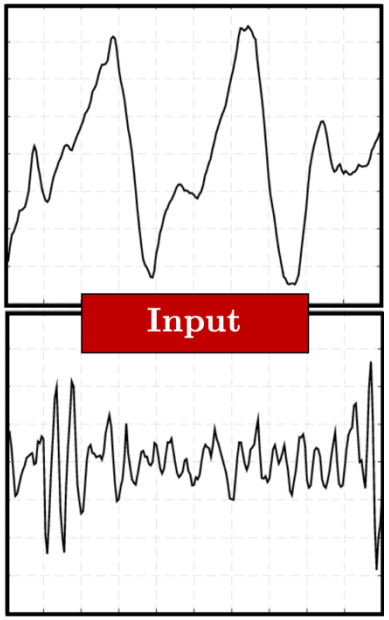
Images as tensors

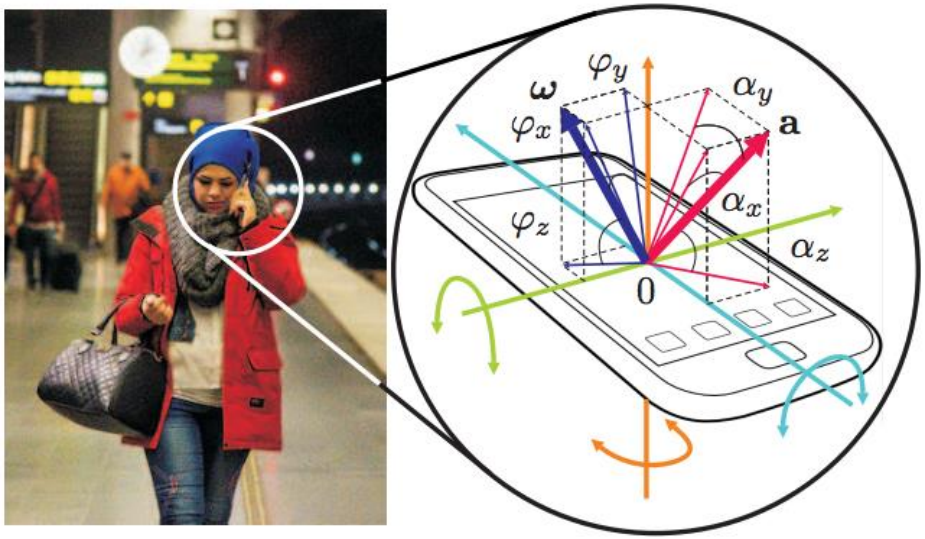
A color image has 3 color channels (red, green, blue) and is therefore a 3D tensor.



(Multiple) 1D signals

- 1500 volunteers, 1500 Nexus 5 smartphones
- Several months of natural daily usage, 27.6 TB of data
- Multiple sensors: camera, touchscreen, GPS, bluetooth, wifi, cell antenna, inertial, magnetometer





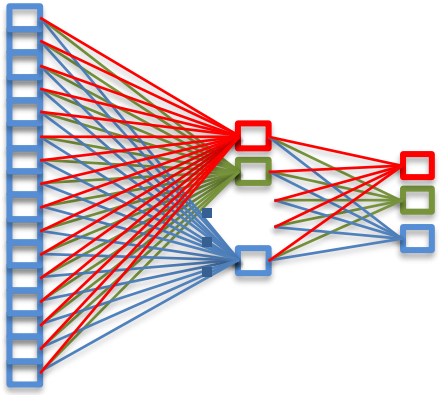
Google

[Neverova, Wolf, Lacey, Fridmann, Chandra, Barbello, Taylor, *IEEE Access 2016*]

Tensors: examples

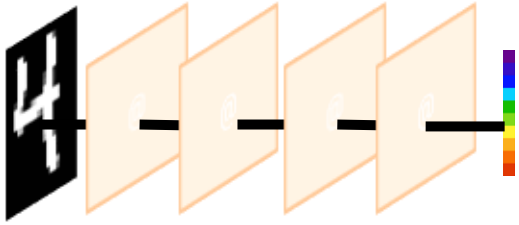
- Example of a tensor of dim 2 (input data, 1D signal)
  - Batch dimension (multipe samples)
  - Signal dimension

- Example of a tensor of dim 2 (output data, classification)
  - Batch dimension (multipe samples)
  - Prediction for different classes



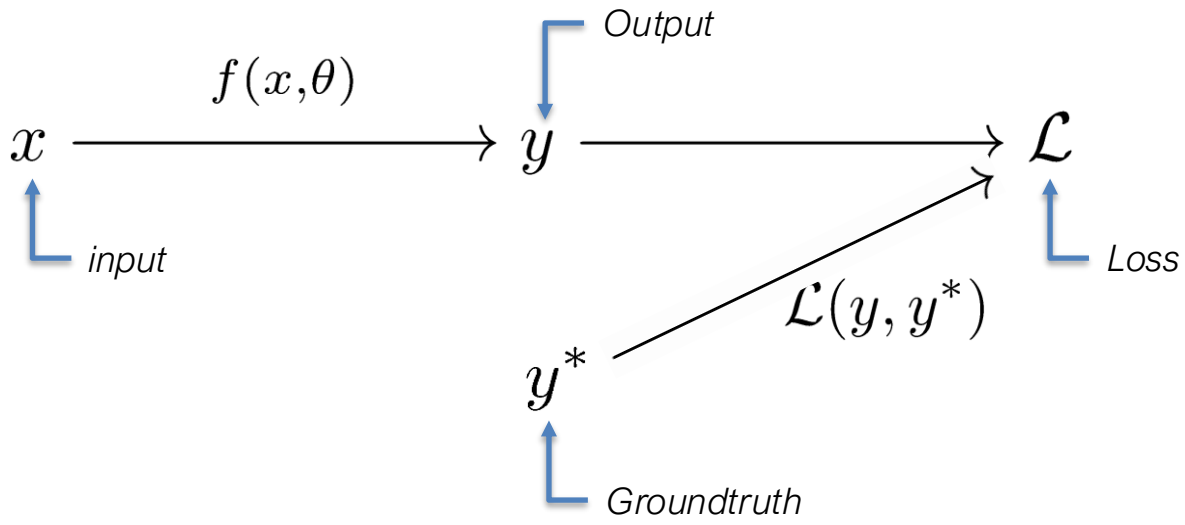
- Example of a tensor of dim 3 (layer activation, 1D signal)
  - Batch dimension (multipe samples)
  - Signal dimension
  - Feature dimension

- Example of a tensor of dim 4 (layer activation, 2D image)
  - Batch dimension (multipe samples)
  - Spatial X dimension
  - Spatial Y dimension
  - Feature dimension



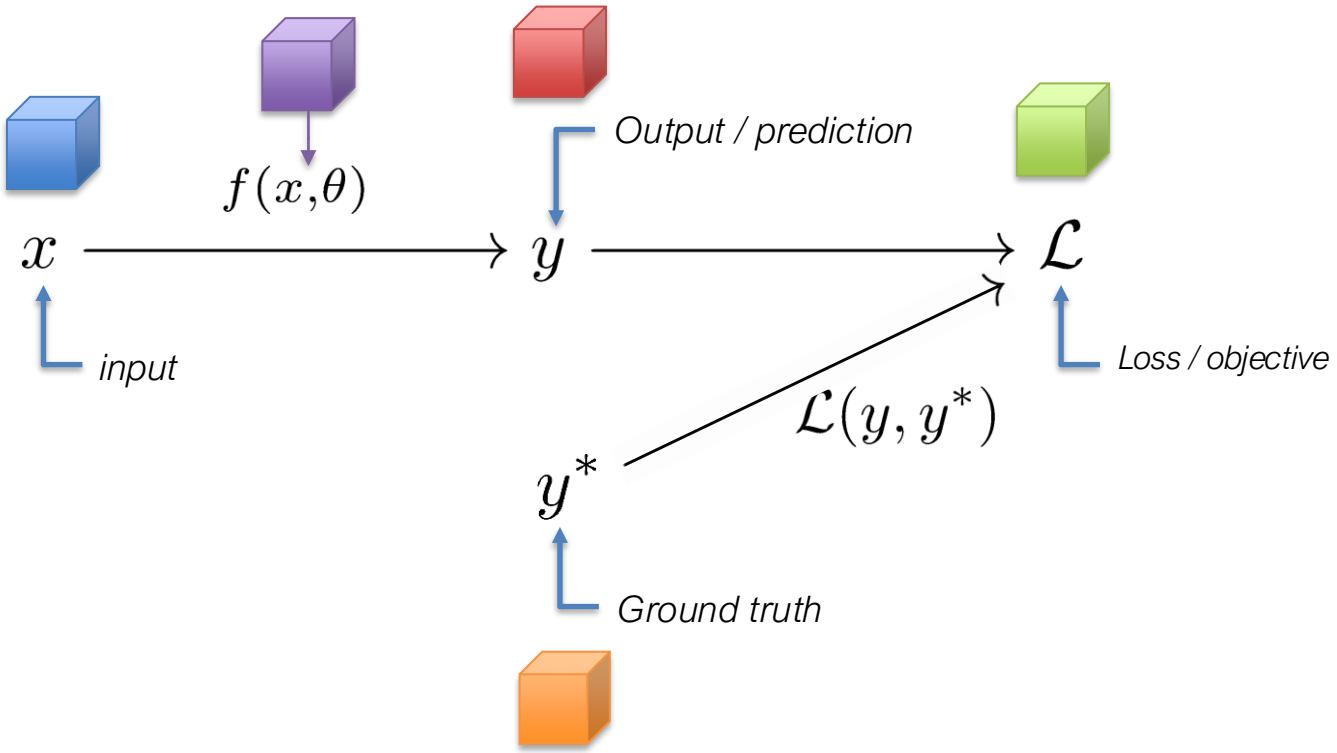
- Example of a tensor of dim 5 (input data, 2D+t video)
  - Batch dimension (multipe samples)
  - Spatial X dimension
  - Spatial Y dimension
  - Color channel dimension
  - Time dimension

Functional mappings



How do we code all this?

Inputs, outputs, layer activations, weights are tensors of different dimensions.

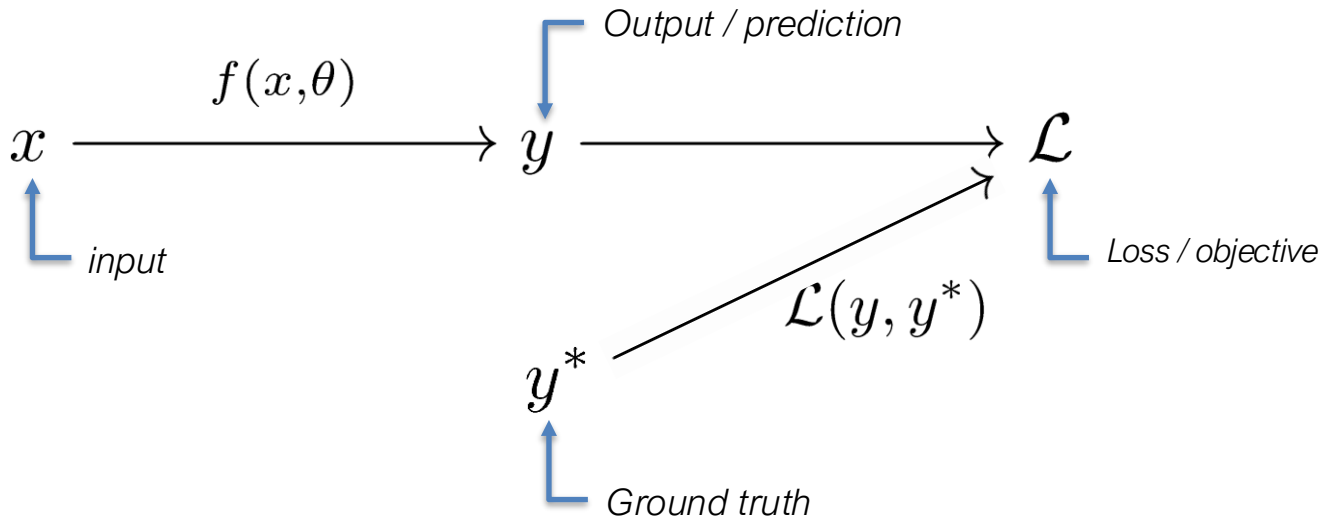


Autograd!

The basic operation is differentiation:

How can we derive … computer science code???

Deep Learning is "differentiable programming"



Deep Learning Frameworks

Before 2013 — The wild west. Models are handcoded in Matlab, C++ by a small minority of people doing deep learning.

2002  — Initial release of Torch by IDIAP Research Institute, later Facebook AI Research. Interfaces: Lua, C++. It uses Autograd (automatic dynamic differentiation) [**Replaced by PyTorch.**]

2010 theano — Theano introduced by Montreal Institute for Learning Algorithms (MILA), University of Montreal. Autograd. [**Discontinued**]

2013 Caffe — CAFFE introduced by University of California, Berkeley. Interfaces: C++ and shell+text files.

2015  — Tensorflow introduced by Google, with immediate success. Interaces: Python, C++ (less supported). Uses a static calculation graph, not Autograd.

2015  — MXNet (Appache with researchers from CMU, NYU, NUS, MIT)

Deep Learning Frameworks

2015 

Keras is a meta-language developed by Google Engineer François Chollet, simplifying coding, initially developed to run on top of Theano. Now also runs on Tensorflow, CNTK. Part of Tensorflow now.

2016 

PyTorch introduced by Facebook, with immediate success. Interaces: Python, C++ (less supported). Uses a static calculation graph, not Autograd.

2016 

CNTK by Microsoft

2017 

Tensorflow introduces eager mode, a dynamic graph calculation mode based on Autograd to respond to the success of PyTorch & Co.

2018 

Jax is introduced by Google and directly works with numpy tensors.

2019 

Tensorflow makes the eager mode the default mode in version 2.0.

Main frameworks

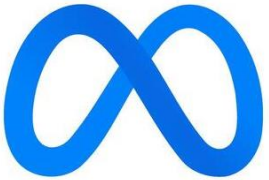
Tensorflow

Jax

PyTorch

Google

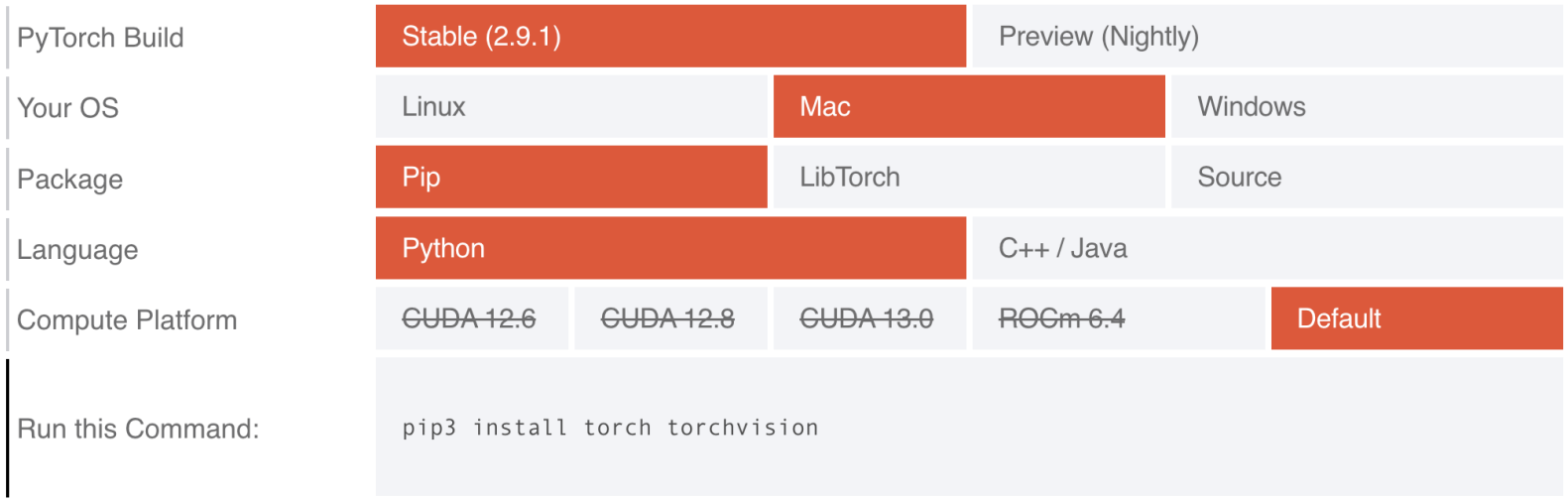
Google



- Support execution and training on CPUs, GPUs, TPUs (google's machine learning hardware)
- All use python.
- Tensorflow also supports C++, Swift.

PyTorch
Tensor cheat sheet

PyTorch Live Install Party



https://pytorch.org/get-started/locally/

Installing PyTorch

Create a virtual environment where we are "safe". Use Python 3.7.

```
cd
virtualenv --system-site-packages ailecture
```

Activate the environment:

```
source ~/ailecture/bin/activate
```

Now all installs well be done in the virtual env.
Install PyTorch itself and the vision package

```
pip3 install torch torchvision
pip3 install scikit-image
```

Creating tensors

```
# loading PyTorch
import torch

# create with given shape
torch.full((shape), value)
torch.full_like(other_tensor,value)

# create with given values
torch.tensor((values))
torch.tensor((values), dtype=torch.int16)

# create from numpy array
torch.from_numpy(numpyArray)

# Create zeros or ones
torch.zeros((shape))
torch.zeros_like(other_tensor)
torch.ones((shape))
torch.ones_like(other_tensor)
```

Creating tensors, tensor I/O

```
# Random tensors
torch.randn(3, 4)

# Tensor I/O
A = torch.load ("A.tensor")
torch.save (A, "A.tensor")
```

Tensor slicing

Slicing is similar to python (NumPy) Slicing or Matlab notation. Example for a 2D tensor:

```
A[1,5]                      # access an element (row, col)
A[:,5]                      # column access
A[1,:]                      # row access

A[1:6,:]    A[1:,:]         # range access (1:6 = 1,2,3,4,5)
A[:,0:-1]                   # Negative index count backwards
                            # -1 = last col/row

A==3                        # provides a tensor of logical
                            # results

A[4:17,3] = B               # replace a slice

A[B==3]=4                   # Set values in A to 4 at pos
                            # where there is a 3 in B
```

Manipulating tensors

```
# concatenate tensors
torch.cat((tensors), axis)

# split tensors into chunks of equal size
torch.split(tensor, splitSize, dim=0)

# reshape tensor w/o changing the data
torch.view(tensor, shape)

# Repeat along a given dimension
X.repeat(4,2)

# transpose tensor
torch.t(tensor) # 1D and 2D tensors
torch.transpose(tensor, dim0, dim1)

# Sorting
torch.sort(input, dim=-1)
```

Tensor math

```
# Overloaded operators
x = A+Y*Z-B                 # * is elementwise mul

# Sum, product,  min, max  of all elements
torch.sum(tensor)    torch.min(tensor)
torch.prod(tensor)   torch.max(tensor)

# Linear algebra
torch.mm(A, B)              # Matrix multiplication
torch.inverse(tensor)       # Matrix inversion
torch.det(tensor)           # Determinant
```

Elementwise operations

```
torch.exp(tensor)    torch.log(tensor)
torch.cos(tensor)    torch.cosh(tensor)
torch.sin(tensor)    torch.sinh(tensor)
torch.tan(tensor)    torch.tanh(tensor)

torch.add(tensor, tensor2) # or tensor+scalar
torch.div(tensor, tensor2) # or tensor/scalar
torch.mult(tensor,tensor2) # or tensor*scalar
torch.sub(tensor, tensor2) # or tensor-scalar
```

Broadcasting

If a PyTorch operation supports broadcast, then its Tensor arguments can be automatically expanded to be of equal sizes (without making copies of the data).

```
x=torch.empty(5,1,4,1)
y=torch.empty(  3,1,1)
(x+y).size()
```

```
torch.Size([5, 3, 4, 1])
```

```
x=torch.empty(5,2,4,1)
y=torch.empty(3,1,1)
(x+y).size()
```

```
RuntimeError: The size of tensor a (2) must match the size
    of tensor b (3) at non-singleton dimension 1
```

https://pytorch.org/docs/stable/notes/broadcasting.html

Broadcasting: rules

Two tensors are "broadcastable" if the following rules hold:

- Each tensor has at least one dimension.
- When iterating over the dimension sizes, starting at the trailing dimension, the dimension sizes must either be equal, one of them is 1, or one of them does not exist.

If two tensors x, y are "broadcastable", the resulting tensor size is calculated as follows:

- If the number of dimensions of x and y are not equal, prepend 1 to the dimensions of the tensor with fewer dimensions to make them equal length.
- Then, for each dimension size, the resulting dimension size is the max of the sizes of x and y along that dimension.

`https://pytorch.org/docs/stable/notes/broadcasting.html`

Images as tensors

```
pip install scikit-image
```

```
import torch
from skimage import io

image = io.imread("blue_tigrou.jpg")

io.imsave("tigre_saved.jpg", image)

G=image[:,:,0]
G=image[:,:,1]
B=image[:,:,2]
io.imsave("tigre_r.jpg", R)
io.imsave("tigre_g.jpg", G)
io.imsave("tigre_b.jpg", B)
```

Tensors from .csv files

```
from numpy import genfromtxt

# Import the text file into a numpy array
n = genfromtxt('file.csv', delimiter=';')

# Convert to torch tensor
D = torch.tensor(n, dtype=torch.float32)
```