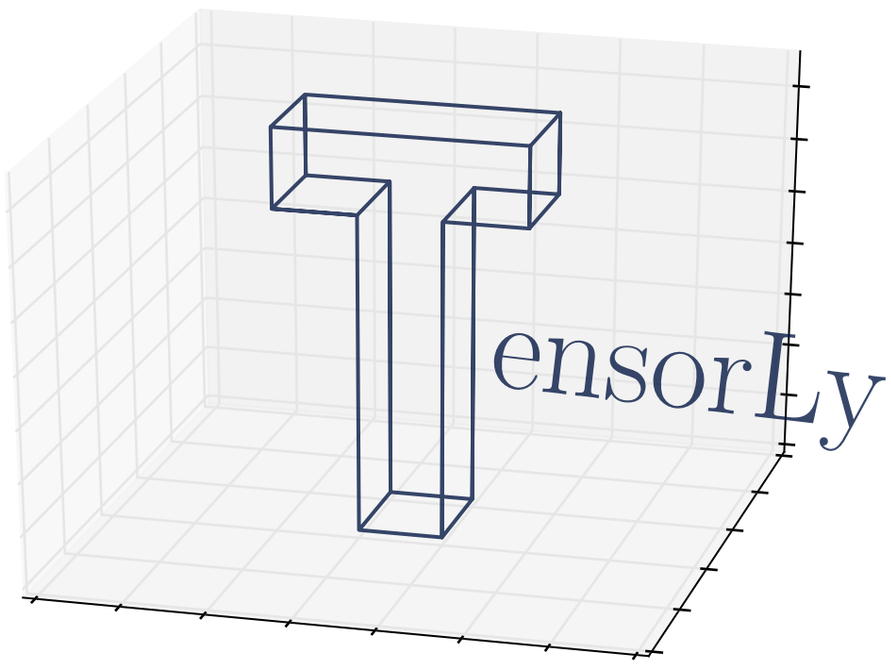
Tensor Learning in python with TensorLy

Jean Kossaifi



https://tensorly.github.io/dev/
https://github.com/tensorly/tensorly

What is it?

- High-level API for tensor methods and deep tensorized neural networks in Python

- Backend system allows users to perform computations with NumPy, MXNet, PyTorch, TensorFlow and CuPy

- Operations and algorithms can be scaled on multiple CPU or GPU machines

Open-source

- Open source, on Github
  - https://github.com/tensorly/tensorly
- BSD licensed
  - Suitable for academic / industrial applications
- Contributions welcome!

Reliable and easy to use

- Depends only on NumPy, SciPy
  [Optionally Matplotlib, MXNet, PyTorch, TensorFlow, CuPy]
- Exhaustive documentation, Unit-testing for all functions
- Fast
- User-friendly API

User-Friendly API

TensorLy

Tensor decomposition | **Tensor regression** | **Tensor + Deep**

Basic tensor operations

Unified backend

NumPy | SciPy | mxnet | PYTORCH

Backend

- Unify the syntax by abstracting away the backend
- tl.set_backend(‘numpy’) # or ‘mxnet’ or ‘pytorch’

```
import tensorly as tl

T = tl.tensor([[1, 2, 3], [4, 5, 6]])
tl.tenalg.kronecker([T, T])
tl.clip(T, a_min=2, a_max=5)

tl.set_backend('mxnet')
T = tl.tensor([[1, 2, 3], [4, 5, 6]])

tl.set_backend('pytorch')
T = tl.tensor([[1, 2, 3], [4, 5, 6]])
```

NumPy ndarray

MXNet NDArray

PyTorch FloatTensor

Tensor operations

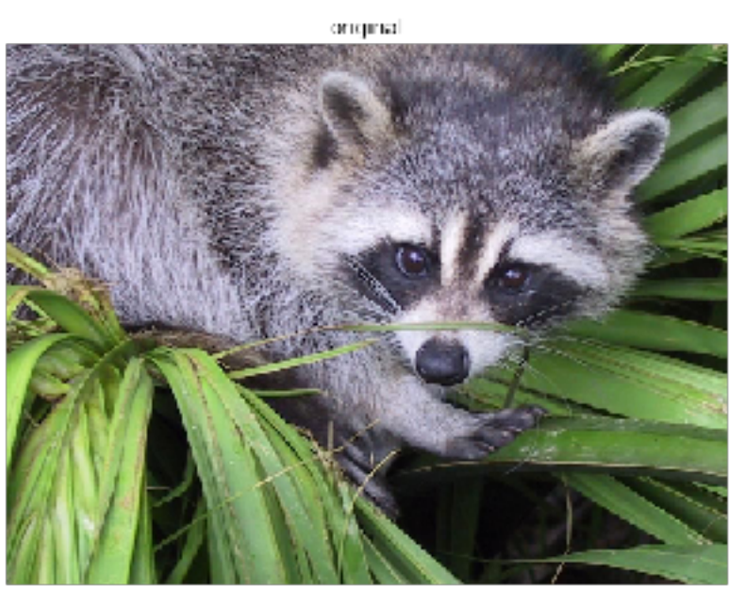
- Kronecker, Khatri-rao, Hadamard products
- Tensor unfolding/folding/vectorization
- n-mode product
- Kruskal and Tucker tensors
- Proximal operators
- …

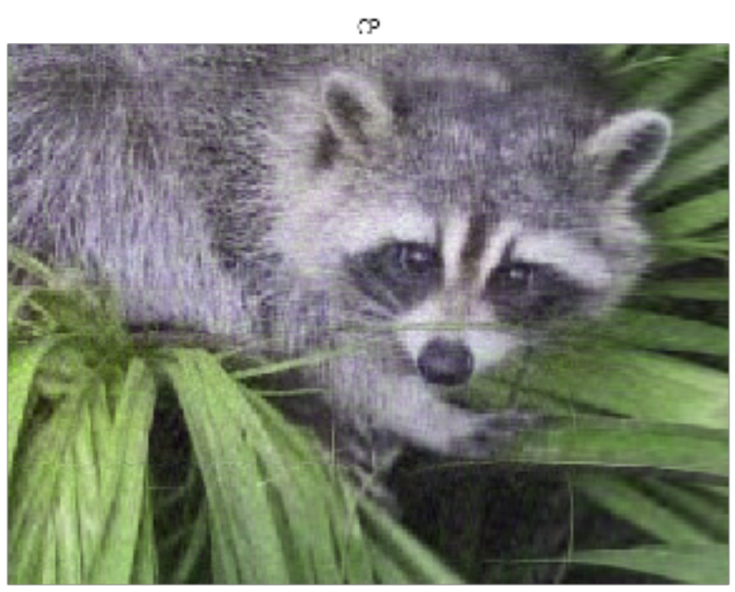
Tensor decomposition

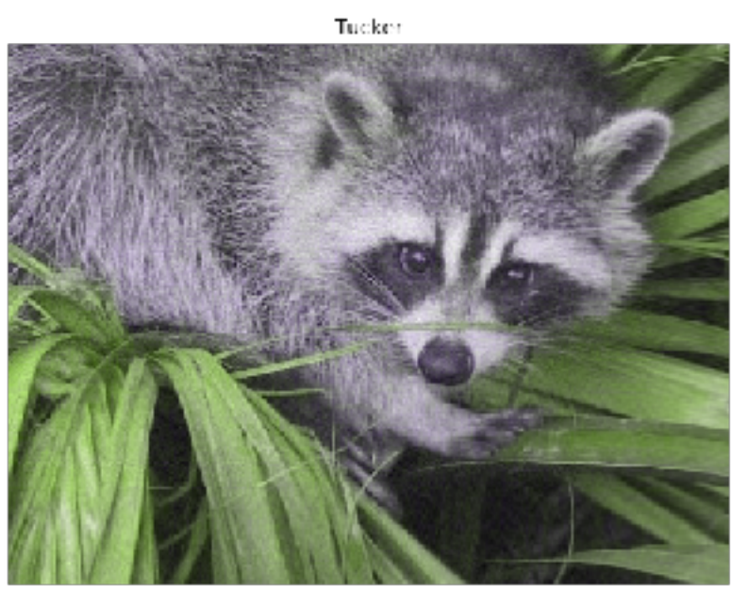
- CANONICAL-POLYADIC (CP)
- Non-negative CP
- Randomised CP
- Tucker (HO-SVD)
- Non-negative Tucker
- Robust Tensor PCA

Tensor decomposition

```
from tensorly.decomposition import parafac

factors = parafac(image, rank=50, init='random')
cp_reconstruction = tl.kruskal_to_tensor(factors)
```



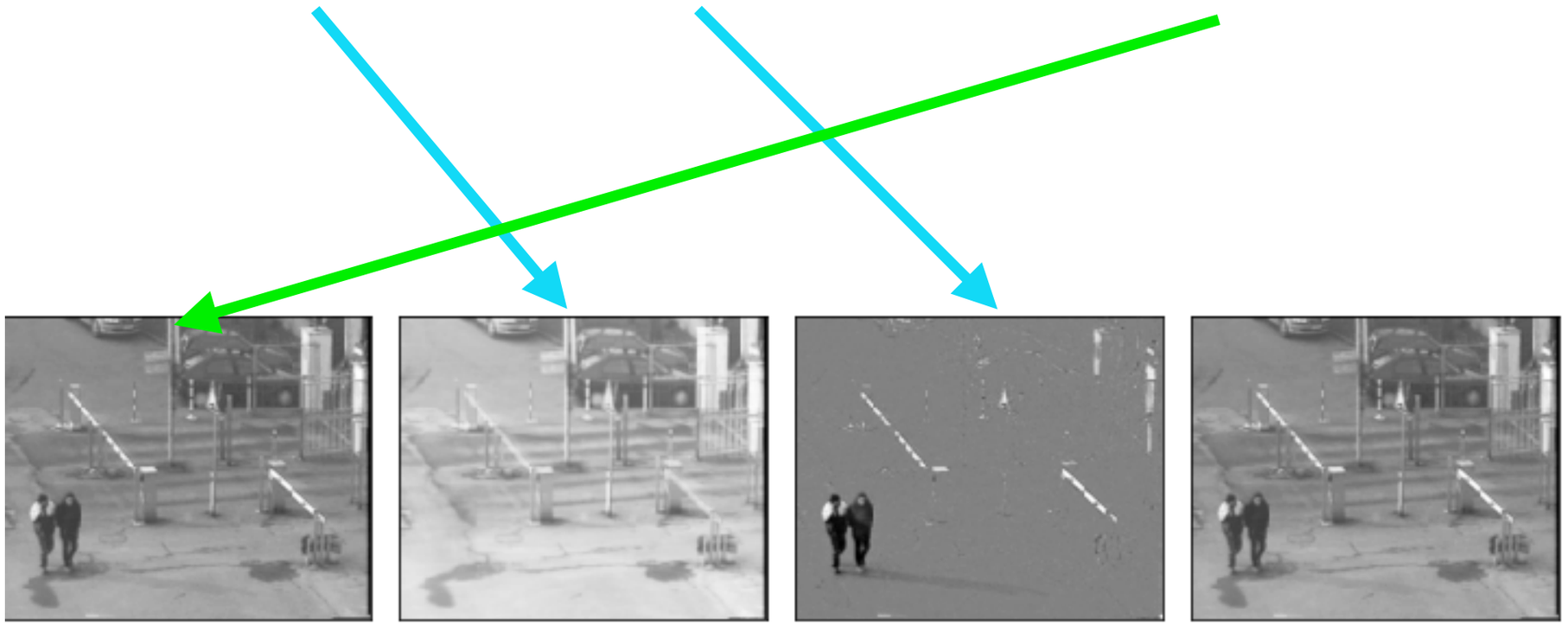




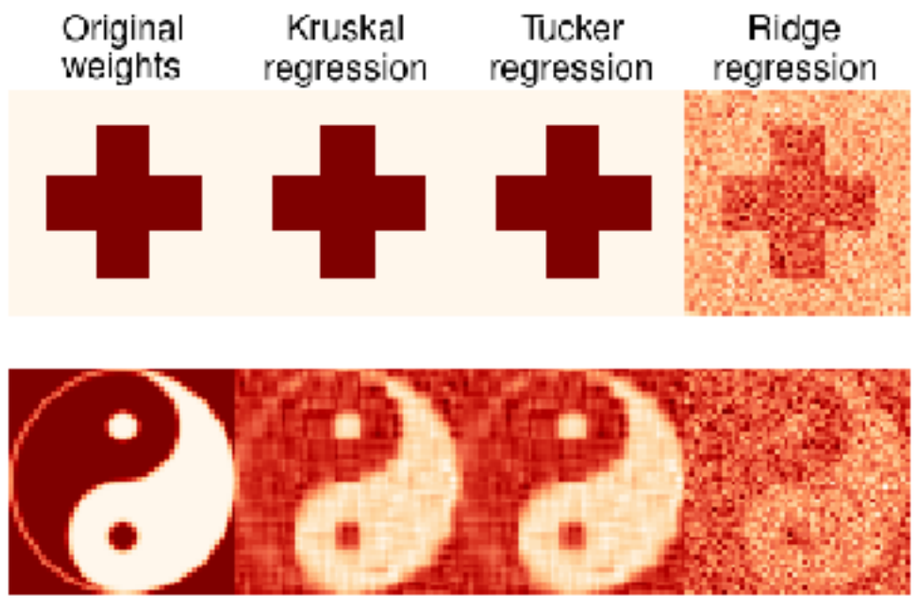
```
from tensorly.decomposition import tucker

core, factors = tucker(image, ranks=(50, 50, 3), init='random')
tucker_reconstruction = tl.tucker_to_tensor(core, factors)
```

Tensor decomposition

low_rank_tensor, sparse_tensor = robust_pca(data_tensor)

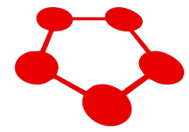


Tensor regression




```
from tensorly.regression.kruskal_regression import KruskalRegressor

estimator = KruskalRegressor(weight_rank=25)

estimator.fit(X, y)

estimator.predict(X)
```

Deep learning + tensor methods = <3



Chainer

PYTORCH

imperative

symbolic

mxnet

gluon

theano

Caffe

Caffe2

Microsoft CNTK

TensorFlow

before 2012 2013 2014 2015 2016 2017

TENSORLY WITH PYTORCH BACKEND

```
import tensorly as tl
from tensorly.random import tucker_tensor

tl.set_backend('pytorch')
core, factors = tucker_tensor((5, 5, 5),
                              rank=(3, 3, 3))
core = Variable(core, requires_grad=True)
factors = [Variable(f, requires_grad=True) for f in factors]

optimiser = torch.optim.Adam([core]+factors, lr=lr)

for i in range(1, n_iter):
    optimiser.zero_grad()
    rec = tucker_to_tensor(core, factors)
    loss = (rec - tensor).pow(2).sum()
    for f in factors:
        loss = loss + 0.01*f.pow(2).sum()

    loss.backward()
    optimiser.step()
```

Any questions?

@JeanKossaifi

jean.kossaifi@gmail.com