H2020-Astronomy ESFRI and Research Infrastructure Cluster
(Grant Agreement number: 653477).

Accelerating Science by Repurposing Machine Learning Software

Bojan Nikolic, Vladislav Stolyarov

3nd ASTERICS-OBELICS Workshop

23-25 October 2018, Cambridge, UK.

Machine Learning Setting

Machine Learning is seen as valuable in commerce

Large, sustained commercial investment in Software

and Hardware

Commercial + Sociological factors:

Open source (cf. Java in early days)

Open competition

Machine Learning in Science ??

Alternative: traditional science by repurposing ML Software

Goals

Science	• Hypothesis based • Exact Models • Driven by the Scientist
Easily	• Familiar interface • Short development cycle • Reduced opportunity for error
Quickly	• Short wall clock time • Efficient (cap ex, op ex)

What can ML software offer?

- Acceleration
- NUMA
- Non-Uniform Interconnect
- Distributed Memory Cluster Scaling
- High Input Data Throughput
- High Quality Programming Interfaces
- Automatic Differentiation
- Task Segmentation
- Working Memory Management
- Efficient large scale minimisers
- +++

What can ML software offer -- example

- Acceleration
- NUMA
- Non-Uniform Interconnect
- Distributed Memory Cluster Scaling
- High Input Data Throughput
- High Quality Programming Interfaces
- Automatic Differentiation
- Task Segmentation
- Working Memory Management
- Efficient large scale minimisers
- +++

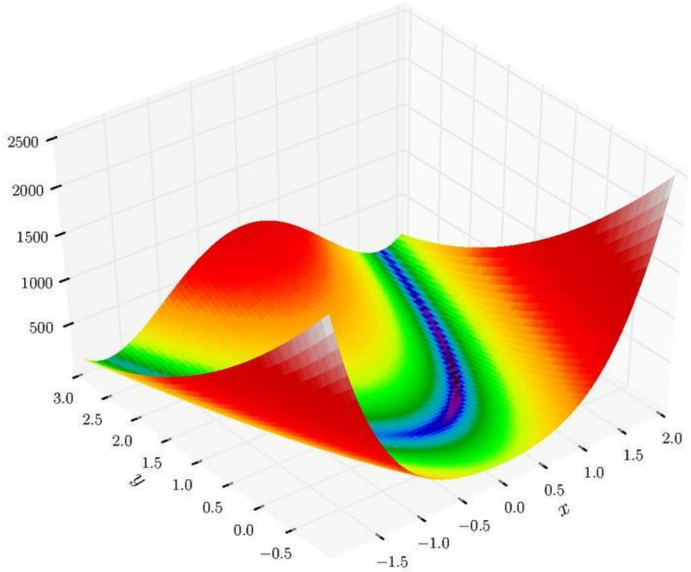
Example: Non-Linear Function minimisation

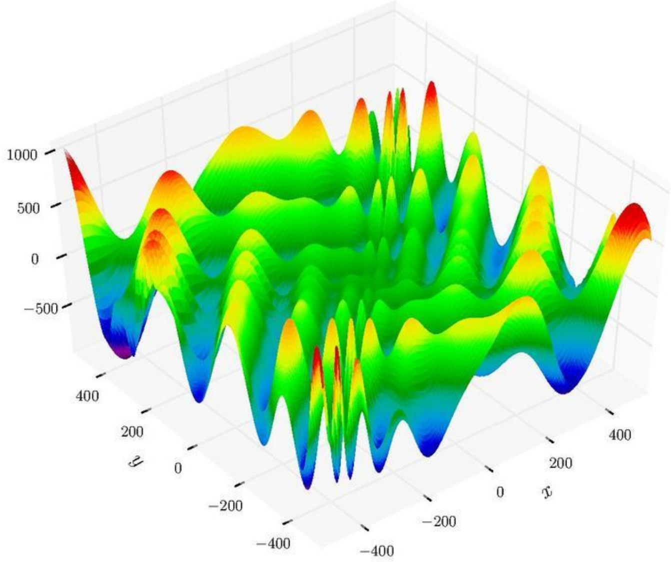
Find:

$$\underset{\vec{x} \in R^n}{\operatorname{argmin}} \ f(\vec{x})$$

$f(\vec{x})$ single valued function

Example algorithms: Nelder-Mead downhill simplex, gradient descent, Broyden Fletcher Goldfarb Shanno (BFGS), MCMC etc. Function gradient often used





Graphics: wikipedia

Focus on function and gradient evaluation

How to calculate:

$$f(\boldsymbol{x}) \text{ and } \nabla f(\boldsymbol{x}) = \frac{\partial f}{\partial x_i}$$

quickly & easily

- Best results for $f(\boldsymbol{x})$ that takes lots of data, uses arrays and has few iterations
- Will say nothing about the minimisation algorithms themselves!
- Useful for function minimisation/maximisation but presumably in other areas too

Machine Learning as Function Minimisation

$\mathcal{D}_{\text{train}}$: $\{(X_i, \widehat{Y}_i)\}$, the training data set

$M(X;\ \Theta) \rightarrow Y$: Predictor

X : An observation (e.g., pixelated image)

Y : Prediction/classification/etc

Θ : Predictor parameters (to be learned) – e.g., weights, biases of a neural network

$\mathcal{L}(\Theta; M, \mathcal{D}_{\text{train}}) = \sum_i L\left(\widehat{Y}_i - M(X;\ \Theta)\right)$: The total “loss” function

L : individual loss function, could be $L1$, $L2$ or something more tailored

“Learning” is (approximately) minimising $\mathcal{L}$ with respect to Θ

How? (With PyTorch)

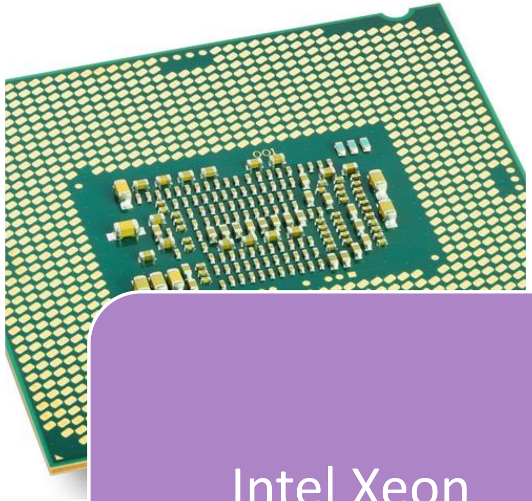
Acceleration	Automatic Differentiation	NumPy-like Interface
• Fast, efficient, high-throughput computation • Using GPUs (or multithreaded CPUs)	• Efficient computation of gradients of functions with many parameters	• Standard imperative semantics, in standard Python • Familiar to most scientists • Right abstraction level (APL)

Why (what is) acceleration ?

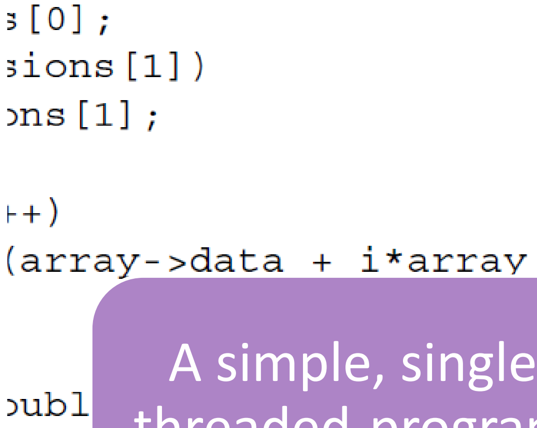
Intel 8087 (Wikipedia, by Rautakorbi)

One Floating Point Unit needs around 20k transistors

Complete FP Co-proc 45k transistors



Intel Xeon Broadwell E5 V4 – 7 billion transistors

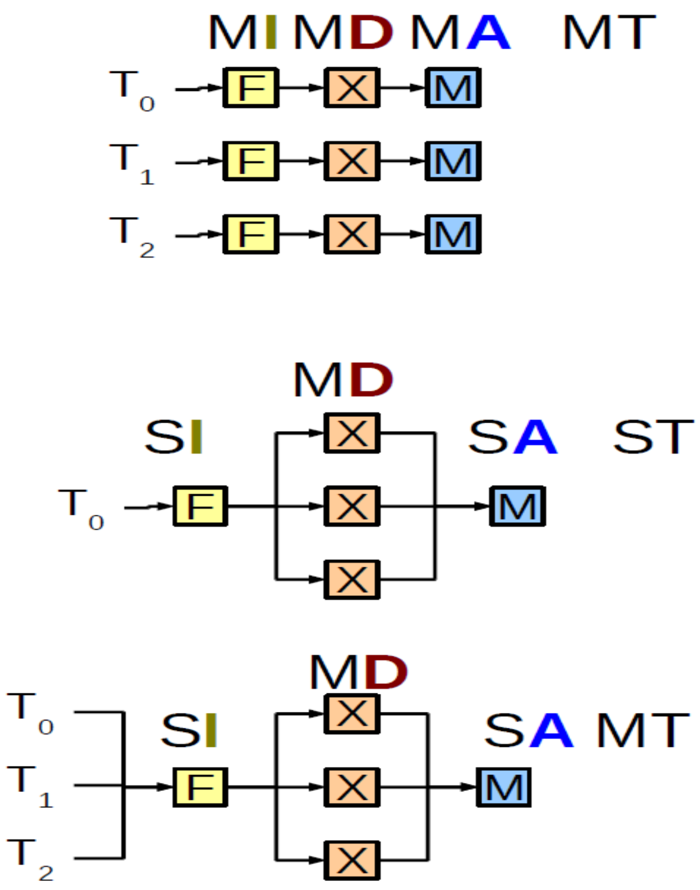


A simple, single threaded-program will “retire” ~1-2 floating point instructions / cycle

on both

GPU Acceleration

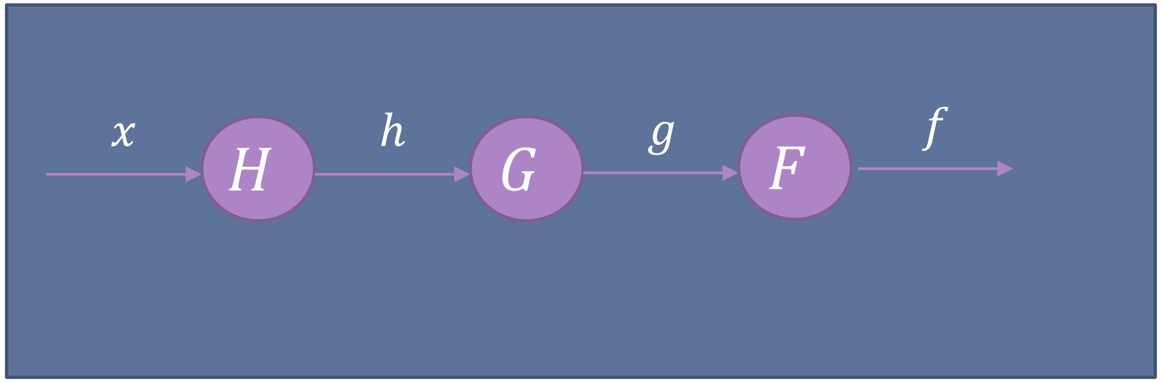
- Multi-core
  - MIMD(MAMT)
- Short-vector SIMD
  - SIMD(SAST)
- GPU
  - SI(MDSA)MT



From very nice slide deck by Sylvain Collange (2011)

Automatic Differentiation

$$h = H(x) \qquad f = F(G(H(x)))$$
$$g = G(h) \qquad h_0 = H(x_0)$$
$$f = F(g) \qquad g_0 = G(h_0)$$



Symbolic Differentiation

- $\left.\frac{df}{dx}\right|_{x_0} = \left.\left(\frac{dF}{dg}\frac{dG}{dh}\frac{dH}{dx}\right)\right|_{x_0}$

Automatic differentiation

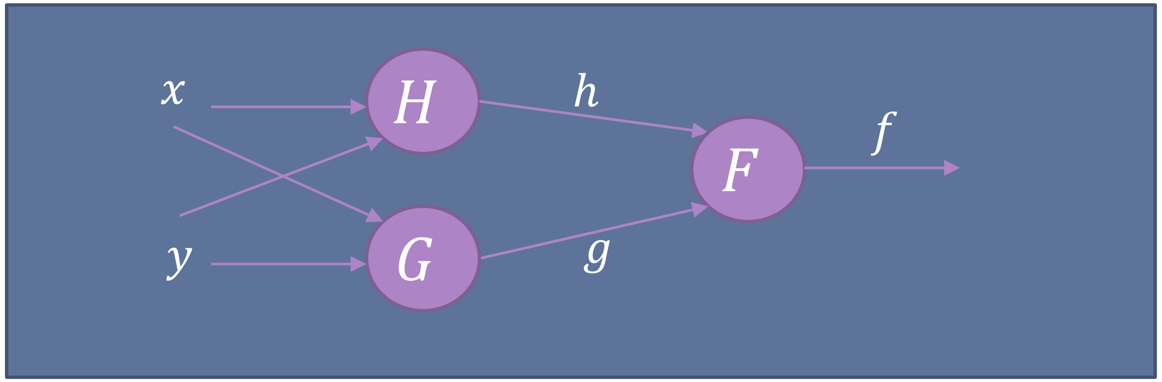
- $\left.\frac{df}{dx}\right|_{x_0} = \left.\frac{dF}{dg}\right|_{g_0} \left.\frac{dG}{dh}\right|_{h_0} \left.\frac{dH}{dx}\right|_{x_0}$

Reverse-Mode Automatic Differentiation

$$h = H(x, y)$$

$$g = G(x, y)$$

$$f = F(h, g)$$



$$\left.\frac{\partial f}{\partial x}\right|_{x_0,y_0} = \left.\frac{\partial F}{\partial h}\right|_{h_0,g_0} \left.\frac{\partial h}{\partial x}\right|_{x_0,y_0} + \left.\frac{\partial F}{\partial g}\right|_{h_0,g_0} \left.\frac{\partial g}{\partial x}\right|_{x_0,y_0}$$

$$\left.\frac{\partial f}{\partial y}\right|_{x_0,y_0} = \left.\frac{\partial F}{\partial h}\right|_{h_0,g_0} \left.\frac{\partial h}{\partial y}\right|_{x_0,y_0} + \left.\frac{\partial F}{\partial g}\right|_{h_0,g_0} \left.\frac{\partial g}{\partial y}\right|_{x_0,y_0}$$

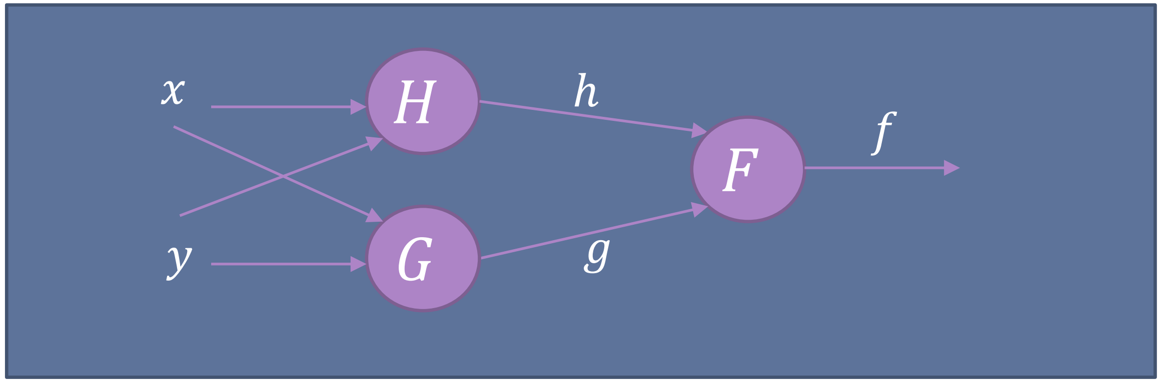
Standard Chain Rule

Reverse-Mode Automatic Differentiation

$$h = H(x, y)$$

$$g = G(x, y)$$

$$f = F(h, g)$$



$$\left.\frac{\partial f}{\partial x}\right|_{x_0,y_0} = \left.\frac{\partial F}{\partial h}\right|_{h_0,g_0} \left.\frac{\partial h}{\partial x}\right|_{x_0,y_0} + \left.\frac{\partial F}{\partial g}\right|_{h_0,g_0} \left.\frac{\partial g}{\partial x}\right|_{x_0,y_0}$$

$$\left.\frac{\partial f}{\partial y}\right|_{x_0,y_0} = \left.\frac{\partial F}{\partial h}\right|_{h_0,g_0} \left.\frac{\partial h}{\partial y}\right|_{x_0,y_0} + \left.\frac{\partial F}{\partial g}\right|_{h_0,g_0} \left.\frac{\partial g}{\partial y}\right|_{x_0,y_0}$$

Standard Chain Rule

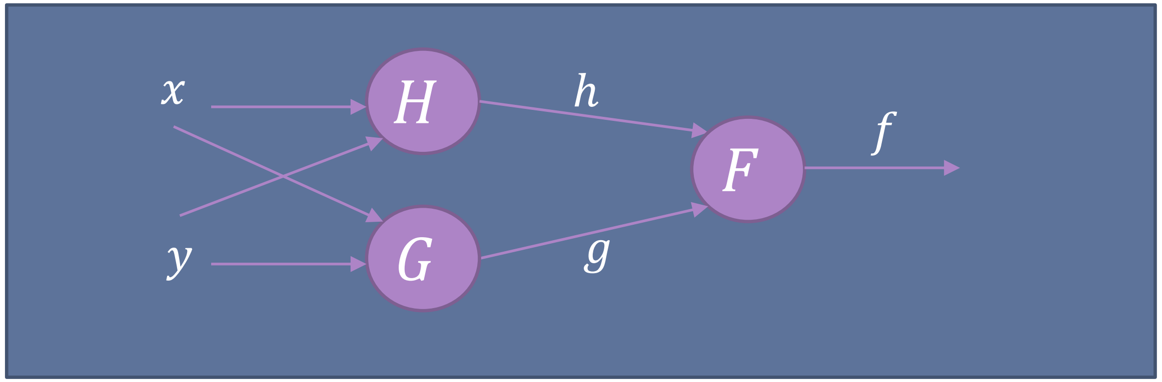
Huge efficiency gain by re-using these (scalar) values

Why reverse?

$$h = H(x, y)$$

$$g = G(x, y)$$

$$f = F(h, g)$$



$$\left.\frac{\partial f}{\partial x}\right|_{x_0, y_0} = \left.\frac{\partial h}{\partial x}\right|_{x_0, y_0} \left.\frac{\partial F}{\partial h}\right|_{h_0, g_0} + \left.\frac{\partial g}{\partial x}\right|_{x_0, y_0} \left.\frac{\partial F}{\partial g}\right|_{h_0, g_0}$$

$$\left.\frac{\partial f}{\partial y}\right|_{x_0, y_0} = \left.\frac{\partial h}{\partial y}\right|_{x_0, y_0} \left.\frac{\partial F}{\partial h}\right|_{h_0, g_0} + \left.\frac{\partial g}{\partial y}\right|_{x_0, y_0} \left.\frac{\partial F}{\partial g}\right|_{h_0, g_0}$$

Standard Chain Rule

Need to evaluate gradient in reverse order compared to program flow

Ease of use: NumPy-like

The languages which were used to guide the development of NumPy include the ***infamous APL*** *family of languages, Basis, MATLAB, FORTRAN, S and S+, and others.*

Numerical Python, Ascher et al, Sept 2001, LANL

1979
Turing
Award
Lecture

Write numerical programs in a way which is easy to think about:

- Benefits both the human user and the language compiler

Notation as a Tool of Thought

KENNETH E. IVERSON
IBM Thomas J. Watson Research Center

PyTorch

PyTorch

Tensors and Dynamic neural networks in Python with strong GPU acceleration

Installation:

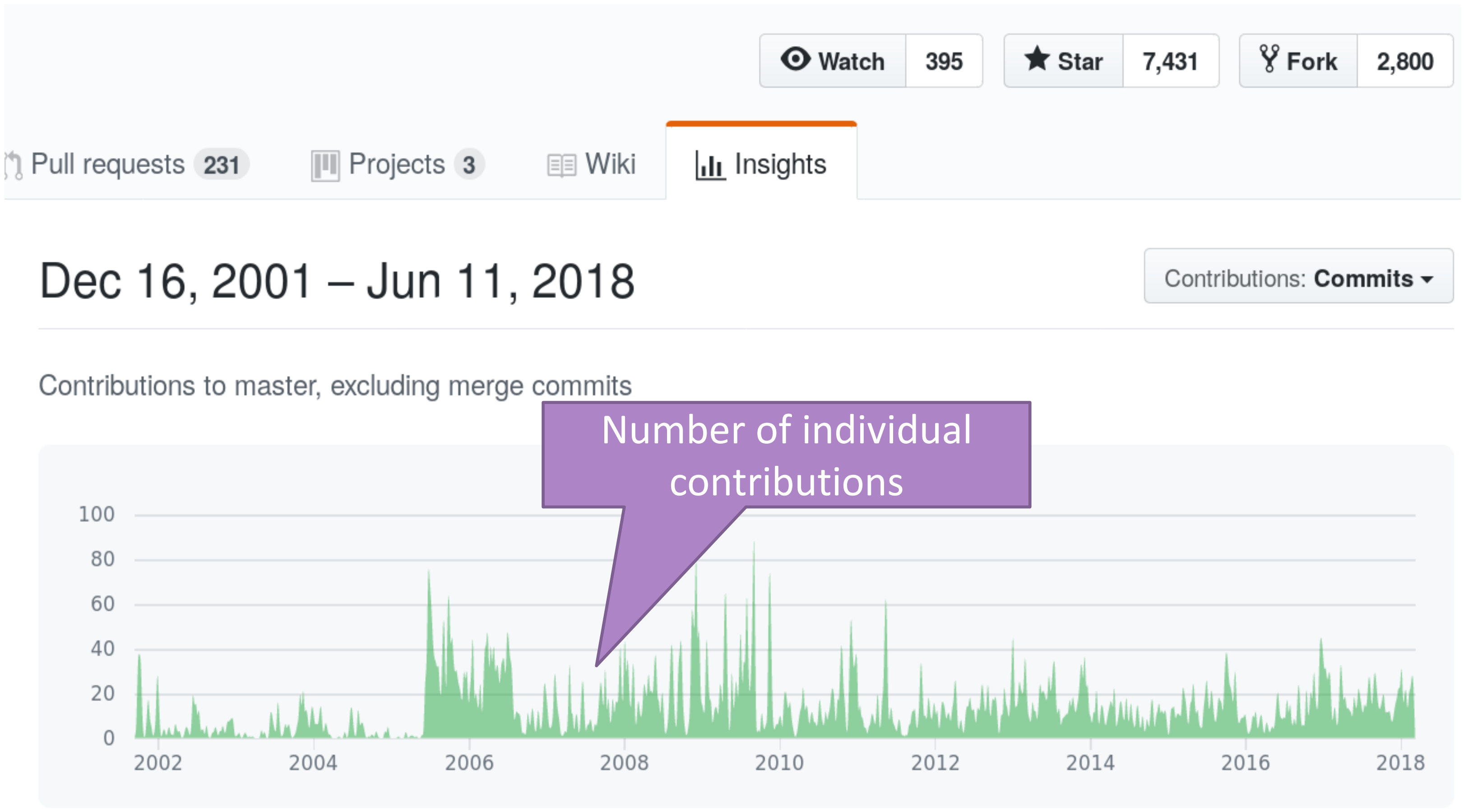
conda install pytorch cuda91 –c pytorch

Automatic differentiation among other features
Trivially easy to offload to GPUs
(PyTorch 0.3.0 supports old GPUs like GeForce 6xx)
OS: Windows/Linux/Mac
Python version: 2.7, 3.5 – 3.7
CUDA version: 8.0 – 9.2
Installer: PIP, Conda, source
C++/LibTorch version
See https://pytorch.org/ for details, tutorial, etc .

NumPy Contributions

Plot on GitHub of contribution frequency over lifetime of the project

NumPy is the main workhorse of numerical data analysis in Python. It is evolution of a library starting in 1996 (numeric, numarrays, etc)



PyTorch Contributions

Plot on GitHub of contribution frequency over lifetime of the project

Sustained, intense contribution

Not usually seen in community-led sw

UNIVERSITY OF CAMBRIDGE

Example application

“OUT-OF-FOCUS” (OOF) HOLOGRAPHY

"Out-of-focus" (OOF) Holography

- Roughly the radio equivalent of optical "active optics"
- A technique to measure the shape of single-dish radio telescope from observations of astronomical sources
  - In "near-real-time" use at the GBT 100m
- An inverse problem adequately solved by local maximum likelihood solution
- Similar problems at optical wavelengths, in electron microscopy, etc.
- See https://www.mrao.cam.ac.uk/~bn204/oof/index.html

Likelihood function

TRADITIONAL

$$P(y|\hat{y}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(y-\hat{y})^2}{2\sigma^2}}$$

Likelihood for a set of observations

$$P(\{y_i\}) = \prod_i P(y_i|\hat{y}_i)$$

Log-likelihood is equivalent to **least-squares** problem

<u>Not efficient</u> to reduce likelihood to a single valued function!

"ROBUST"

E.g. Cauchy distribution:

$$P(y|\hat{y}) = \frac{1}{\pi\gamma} \frac{\gamma^2}{(y-\hat{y})^2+\gamma^2}$$

Captures the possibility of outliers (glitches in read-out, short term pointing instability in telescope, atmospheric disturbance)

Log-likelihood **does not** factor into least-squares

Porting to PyTorch

- Most NumPy operations are supported by PyTorch arrays
  - Missing ops easily implemented
- Array slicing etc works in the same way
- Normal "imperative" Python semantics
- Only performance critical code needs to be ported
  - Zero-overhead sharing of data between PyTorch and NumPy arrays, so incremental porting very easy and efficient
- Can use standard Python/Numpy/Scipy optimisers

NumPy vs PyTorch code comparison

NumPy:

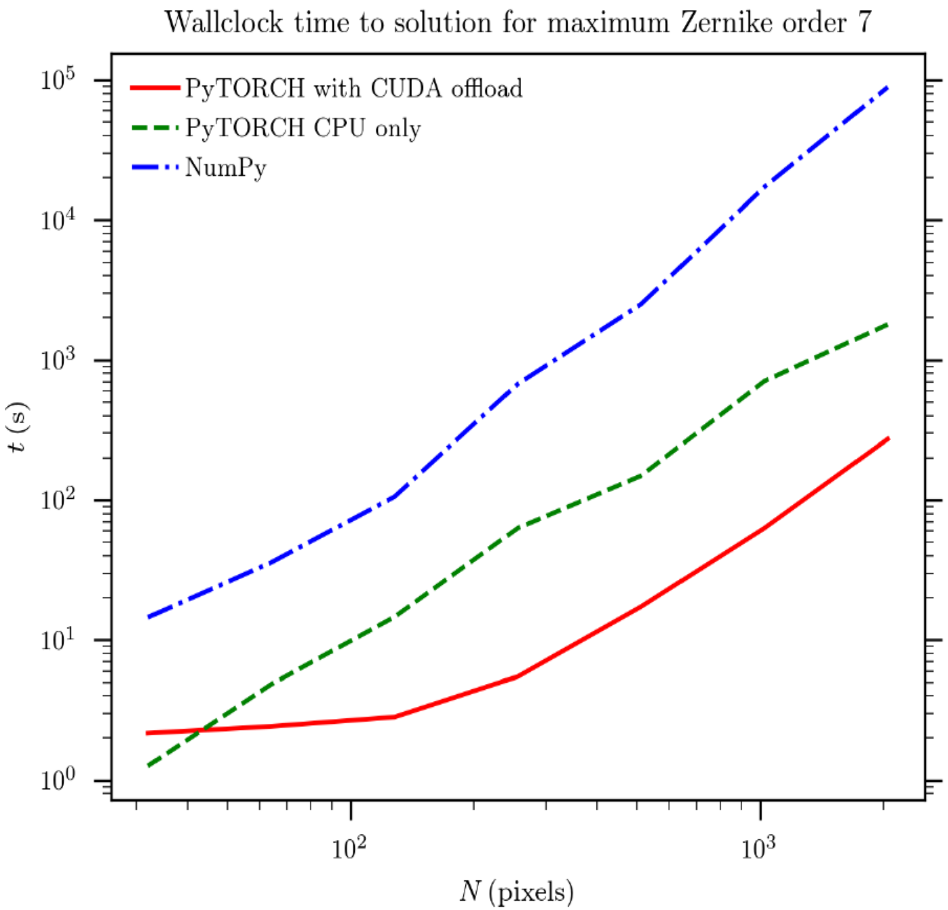
```
def gauss(x0, y0,
          amp,
          sigma, rho, diff,
          a):
    dx=a[...,0]-x0
    dy= a[...,1]-y0
    r=numpy.hypot(dx, dy)
    R2= (r**2 +
         rho*(dx*dy)+
         diff*(dx**2-dy**2))
    E=numpy.exp(-1.0/
         (2*sigma**2)*R2)
    return amp*E
```

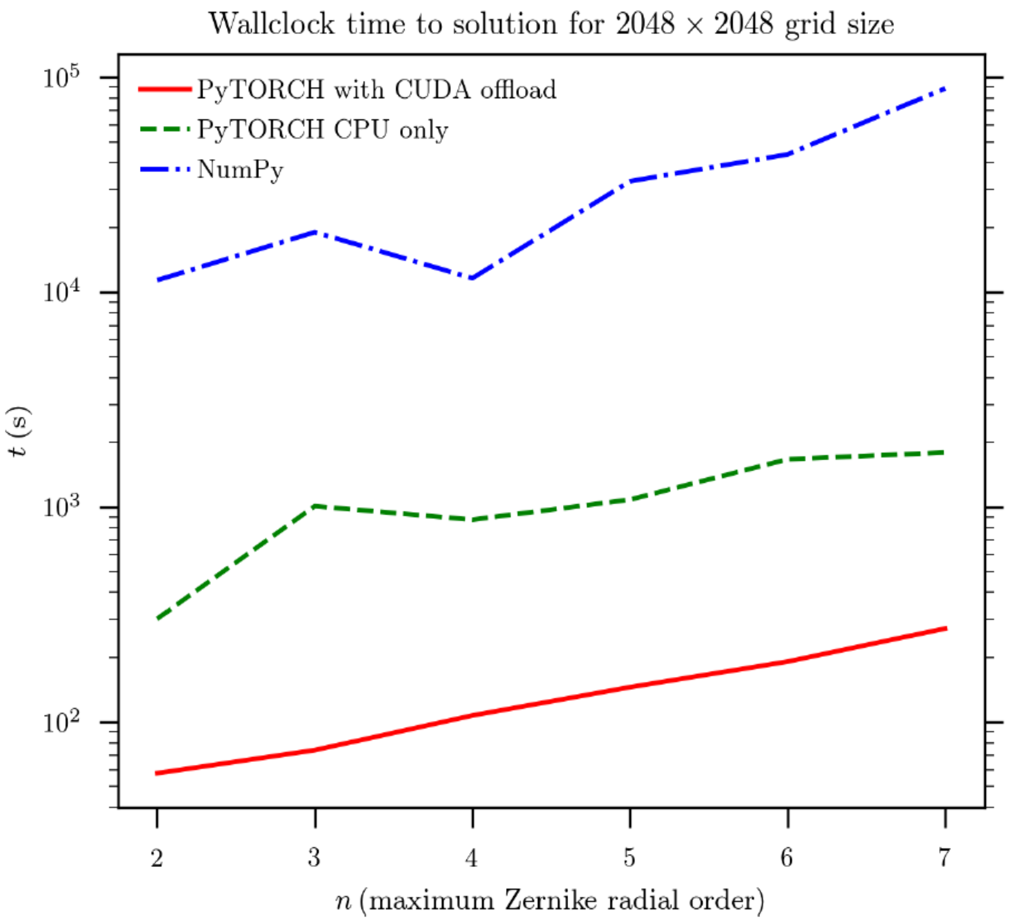
PyTorch:

```
import torch as T
def hypot(x, y):
    return T.sqrt(x**2 + y**2)

def gauss(x0, y0,
          amp,
          sigma, rho, diff,
          a):
    dx=a[...,0]-x0
    dy= a[...,1]-y0
    r=hypot(dx, dy)
    R2=(r**2 +
        OO(rho*(dx*dy))+
        OO(diff*(dx**2-dy**2)))
    E=T.exp(-1.0/(2*sigma**2)*R2)
    return OO(amp*E)
```

Performance comparison





Why?

- 300x speed-up brings new possibilities (~1 year -> 1 day):
  - More complex models
  - More data
  - Difficult fn’s (e.g., near-field diffraction)
  - Realistic maximum likelihood statistics that take into account telescope errors
- New regime: real-time maximum-likelihood or maximum-a-posteriori solutions:
  - 5 minutes - > 1 second
  - Human in the loop and control system applications

Summary

>100x performance improvement in minimising functions

Small, contained, software effort needed

- Perfect integration with standard Python environment

Out-of-box support for GPUs and multi-threaded CPUs

Easy to use (& install!)

More details: arXiv:1805.07439

Acknowledgement

H2020-Astronomy ESFRI and Research Infrastructure Cluster (Grant Agreement number: 653477).