Dynamic Programming with Google JAX

CEF 2024

John Stachurski

June 2024

Slides, code:

https://github.com/QuantEcon/cef_2024_singapore

Quick poll:

- Python programmers?
  - NumPy? Numba? PyTorch? JAX?
- Julia?
- MATLAB?
- C?
- Fortran?

Regular GPU users?

Topics

- Types of programming languages
- Vectorization vs JIT compilers
- Motivation for JAX

Old school: static types & AOT compilers

Typical languages: Fortran / C / C++

Example. Suppose we want to compute the sequence

$$k_{t+1} = s k_t^{\alpha} + (1 - \delta) k_t$$

from some given k_0

Let's write a function that returns the last k_t

```c
#include <stdio.h>
#include <math.h>

int main() {
    double k = 0.2;
    double alpha = 0.4;
    double s = 0.3;
    double delta = 0.1;
    int i;
    int n = 1000;
    for (i = 0; i < n; i++) {
        k = s * pow(k, alpha) + (1 - delta) * k;
    }
    printf("k = %f\n", k);
}
```

```fortran
program main
 implicit none
 integer, parameter :: dp=kind(0.d0)
 integer :: n=1000
 real(dp) :: s=0.3_dp
 real(dp) :: a=1.0_dp
 real(dp) :: delta=0.1_dp
 real(dp) :: alpha=0.4_dp
 real(dp) :: k=0.2_dp
 integer :: i
 do i = 1, n - 1
  k = a * s * k**alpha + (1 - delta) * k
 end do
 print *,'k = ', k
end program main
```

Pros of low-level languages

- fast loops / arithmetic

Cons

- low interactivity
- time consuming to write / read / debug
- hard to parallelize

For comparison, the same operation in Python:

```python
α = 0.4
s = 0.3
δ = 0.1
n = 1_000
k = 0.2

for i in range(n):
    k = s * k**α + (1 - δ) * k

print(k)
```

Pros

- high interactivity
- easy to write / read / debug

Cons

- slow loops / arithmetic

Why is pure Python slow?

Pros

- high interactivity
- easy to write / read / debug

Cons

- slow loops / arithmetic

Why is pure Python slow?

Problem 1: Type checking

```
# Ints
x, y = 1, 2
z = x + y        # z = 3

# Floats
x, y = 1.0, 2.0
z = x + y        # z = 3.0

# Strings
x, y = 'foo', 'bar'
z = x + y        # z = 'foobar'
```

How does Python know which operation to perform?

Answer: Python checks the type of the objects first

```
>> x = 1
>> type(x)
int
```

```
>> x = 'foo'
>> type(x)
str
```

In a large loop, this type checking generates massive overhead

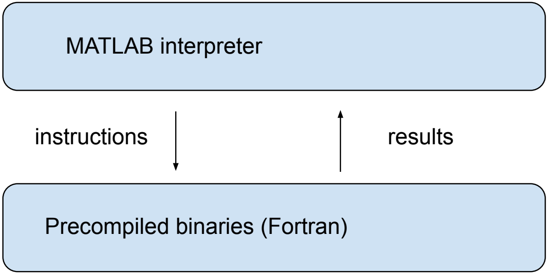
Problem 2: Memory management

```
>>> import sys
>>> x = [2.56, 3.21]
>>> sys.getsizeof(x) * 8        # number of bits
576                             # whaaaat???
```

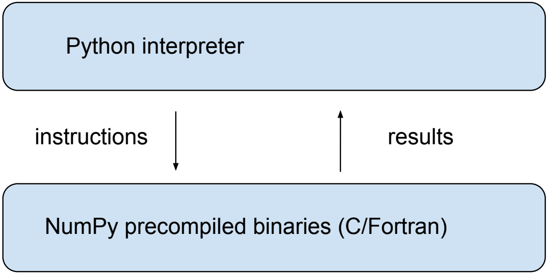
So how can we get

good execution speeds **and** high productivity / interactivity?

MATLAB's vectorization trick



Python + NumPy – stealing MATLAB's idea



Vectorization: pros and cons

Pros

- high interactivity

Cons

- some tasks cannot be efficiently vectorized
- cannot adapt flexibly to function arguments / hardware

Julia — rise of the JIT compilers

```
function solow(k0, α=0.4, δ=0.1, n=1_000)
    k = k0
    for i in 1:(n-1)
        k = s * k^α + (1 - δ) * k
    end
    return k
end

solow(0.2)
```

Python + Numba copy Julia

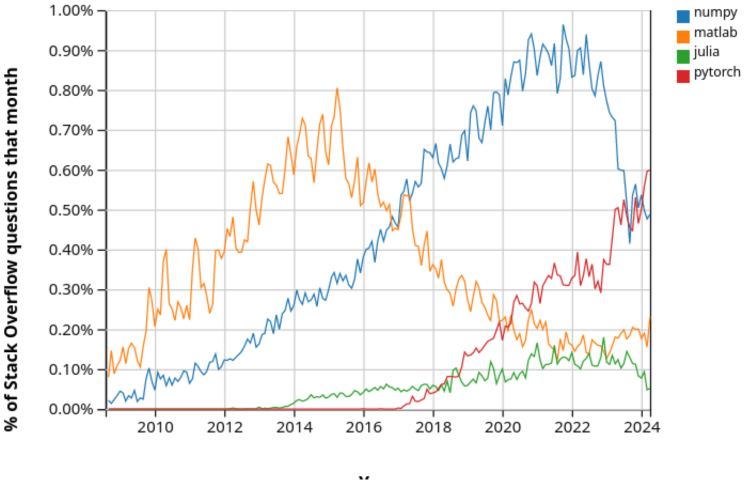
```
from numba import jit

@jit
def solow(k0, α=0.4, δ=0.1, n=1_000):
    k = k0
    for i in range(n-1):
        k = s * k**α + (1 - δ) * k
    return k

solow(0.2)
```

Runs at same speed as Julia / C / Fortran

Some trends:


% of Stack Overflow questions that month
1.00%
0.90%
0.80%
0.70%
0.60%
0.50%
0.40%
0.30%
0.20%
0.10%
0.00%
2010
2012
2014
2016
2018
2020
2022
2024
x
numpy
matlab
julia
pytorch

So where does JAX fit in?

Let's start with some motivation and background

AI-driven scientific computing

AI is changing the world

- image processing / computer vision
- speech recognition, translation
- scientific knowledge discovery
- forecasting and prediction
- generative AI

Plus killer drones, skynet, etc....

AI-driven scientific computing

AI is changing the world

- image processing / computer vision
- speech recognition, translation
- scientific knowledge discovery
- forecasting and prediction
- generative AI

Plus killer drones, skynet, etc....

Projected spending on AI in 2024:

- Google: \$48 billion
- Microsoft: \$60 billion
- Meta: \$40 billion
- etc.

Key point: vast investment in AI is changing the production possibility frontier for **all** scientific coders

Projected spending on AI in 2024:

- Google: \$48 billion
- Microsoft: \$60 billion
- Meta: \$40 billion
- etc.

Key point: vast investment in AI is changing the production possibility frontier for **all** scientific coders

Platforms / libraries

- PyTorch (ChatGPT, LLaMA 3, Github Copilot)
- Google JAX (Gemini)
- Tensorflow?
- Mojo (by Modular)?

Deep learning in two slides

Supervised deep learning: find a good approximation to an unknown functional relationship

$$y = f(x) \qquad (x \in \mathbb{R}^d,\ y \in \mathbb{R})$$

Examples.

- $x =$ sequence of words, $y =$ next word
- $x =$ weather sensor data, $y =$ max temp tomorrow

Problem:

- observe $(x_i, y_i)_{i=1}^n$ and seek f such that $y_{n+1} \approx f(x_{n+1})$

Training: minimize the empirical loss

$$\ell(\theta) := \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

But what is $\{f_\theta\}_{\theta \in \Theta}$?

In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_1 \circ \cdots \circ \sigma \circ A_{k-1} \circ \sigma \circ A_k$$

where

- $A_i x = W_i x + b_i$ is an affine map
- σ is a nonlinear "activation" function

Training: minimize the empirical loss

$$\ell(\theta) := \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

But what is $\{f_\theta\}_{\theta\in\Theta}$?

In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_1 \circ \cdots \circ \sigma \circ A_{k-1} \circ \sigma \circ A_k$$

where

- $A_i x = W_i x + b_i$ is an affine map
- σ is a nonlinear "activation" function

Training: minimize the empirical loss

$$\ell(\theta) := \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2 \quad \text{s.t.} \quad \theta \in \Theta$$

But what is $\{f_\theta\}_{\theta \in \Theta}$?

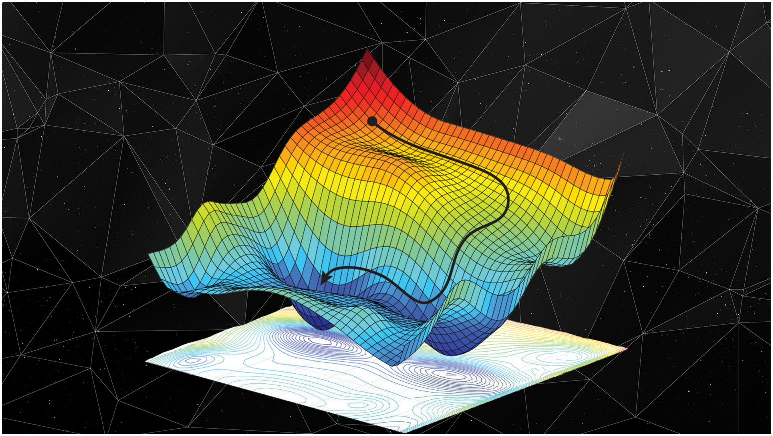
In the case of ANNs, we consider all f_θ having the form

$$f_\theta = \sigma \circ A_1 \circ \cdots \circ \sigma \circ A_{k-1} \circ \sigma \circ A_k$$

where

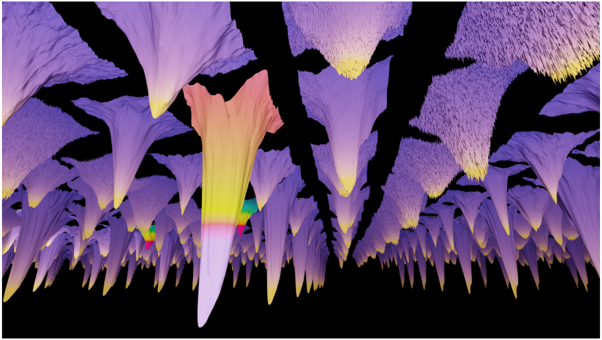
- $A_i x = W_i x + b_i$ is an affine map
- σ is a nonlinear "activation" function

Minimizing a smooth loss functions – what algorithm?



Source: https://danielkhv.com/

Deep learning: $\theta \in \mathbb{R}^d$ where $d = ?$



Source: https://losslandscape.com/gallery/

Hardware

"NVIDIA supercomputers are the factories of the AI industrial revolution." – Jensen Huang

Software

Core elements

- automatic differentiation (for gradient descent)
- parallelization (GPUs! — how many?)
- Compilers / JIT-compilers

Crucially, these components must be well integrated

```python
import jax.numpy as jnp
from jax import grad, jit

def f(θ, x):
  for W, b in θ:
    w = W @ x + b
    x = jnp.tanh(w)
  return x

def loss(θ, x, y):
  return jnp.sum((y - f(θ, x))**2)

grad_loss = jit(grad(loss))  # Now use gradient descent
```

Source: JAX readthedocs

What I need: software that will parallelize **for me**

- automated intelligent parallelization
- JIT compiled — flexible
- integrated autodiff
- automatically supports most CPUs / GPUs / TPUs / etc.

JAX ticks these boxes