Why Python's the best language for AI (and how to make it even better)

Matthew Honnibal
Explosion AI

About me

- **Matthew Honnibal**, @honnibal
- BA in **Linguistics**, PhD in **Computer Science**
- Half of **Explosion AI**, together with **Ines Montani**
- We're the makers of **spaCy** & other **Python AI** libraries

spaCy — Open-source library for industrial-strength Natural Language Processing

THINC — spaCy's next-generation Machine Learning library for deep learning with text

spaCy

- **Fastest** Natural Language Processing (NLP) library **in the world**
- Known for **ease of use** and **performance**
- Implemented in **Cython**
- **Lesson:** upfront optimisation is not necessarily premature

Python is the best language for AI – but it could be even better.

Python is dominant in data science and AI

- More **general-purpose** and **practical** than R
- More **productive** and **flexible** than Java and C++
- Better **ecosystem** than Ruby and JavaScript
- Python is a good all-rounder, but it's only #1 in **data science**, **machine learning** and **AI**

How did Python get the best ecosystem?

- Languages “win” by being **good enough** and in the **right place** at the **right time**
- ~2004 Java and C++ weren’t productive enough
- **C extensions** were harder for Perl and Ruby

C extensions are our edge. We can't exploit that if we don't understand it.

Python vs. Cython

19 seconds

```
def relu(X):
    for i in range(X.shape[0]):
        if X[i] < 0:
            X[i] = 0
```

1.3 seconds

```
cdef void relu(float* X, int n) nogil:
    for i in range(n):
        if X[i] < 0:
            X[i] = 0
```

1.35 seconds

```
def relu(X):
    numpy.maximum(X, 0, X)
```

1.4 seconds

```
def relu(X):
    X *= X > 0
```

What should we do differently?

- Individual developers should get comfortable **writing Cython** and reach for it sooner
- The community should invest more in making this **easy** and **obvious**
- **Performance** is a big problem for Python code, but we can’t fix it as an afterthought


```
from libcpp.pair cimport pair
from libcpp.queue cimport priority_queue
from lib.stdint cimport uint64_t

cdef struct Rating:
     uint64_t user
     uint64_t product
     float score

cdef top3_ratings(Rating* ratings, int n):
     cdef priority_queue[pair[float, int]] queue
     cdef pair[float, int] item
     for i in range(n):
          item.first = ratinga[i].score
          item.second = i
          queue.push(item)
     top3 = []
     for i in range(3):
          top3.append(ratings[queue.pop().second])
     return top3
```

each rating: 160 bits of memory
one billion ratings: 20 GB

find top 3 without sorting whole list

easily output to Python

Python is the best language for AI.

Python's most popular...

What programming language do you mostly use for AI?

thestateofai.com

78%

PYTHON

Most Popular Technologies per Dev Type: Math & Data

insights.stackoverflow.com

55.5%

PYTHON

Python's ecosystem makes it the best choice for AI...

"what really sets the two languages [Python and Ruby] apart are their **surrounding ecosystems of frameworks** and libraries."

"scikit-learn: machine learning in Python is one of the easiest and most advanced library used for this purpose. "

"Because of the maturity and **breadth of its package library,** Python excels in providing a playground for working with

"As far as I know it's purely library based."

"Another important reason for its popularity is availability of many **open source projects related to ML** (scikit-learn, scipy, numpy etc.). These packages are freely available and anyone can tweak it according

Source: Why is Python so popular in Machine Learning? (Quora)

A normal approach to performant Python

1. Implement it.
2. Make it a bit faster.
3. Can we use PyPy here?
4. More cores!
5. Crap, why isn't any of this helping?

The parable of the tree kangaroo

Incremental improvements don't always lead to the best solution.

- You can make your Python code faster, bit by bit
- Kangaroos can get better at climbing, bit by bit
- But if you start off in the **wrong part of the solution space**, you won't get to the best solution

Incremental solutions aren’t always competitive.

A better approach to performant Python

1. **Plan your data structures.** Don't be afraid of pointers and structs.
2. Write the **simple, obvious** and **approximately optimal** solution.
3. Chant strange incantations into setuptools to fix some weird compiler error that makes no sense.
4. Wonder why there aren't more docs for this thing.
5. Reconsider your life choices.
6. Eventually, **benefit greatly.**

AI in Python could be much better.

Let's make Cython the accepted answer.

- Provide default **support in setuptools**
- Recommend Cython in **main documentation**
- Consider **deeper integrations** into **CPython**

Let's take Cython even further.

- **Cython-CUDA** could be so good
- Libraries could have great **Cython APIs**
- It could be much easier to compile **standalone** libraries or applications

“But I’m a data scientist. I just build prototypes... ”

Are you sure your niche is secure?

“But I’m a data scientist. I just build prototypes... ”

- Don’t bet your career on that niche
- More stable to be a **specialist generalist** (T-shaped skills)
- **Collaboration** is really hard if nobody understands each others’ roles

“But why can’t we just use JIT compilation?”

- The benefits flow from **making decisions**
- That’s the part that “feels hard” — but you can’t get to a good solution without it
- If you don’t **plan out your data structures** for optimisation, JIT cannot help you

“Can’t I just call fast libraries from slow code?”

- You can – mostly! But what will you do when **you need more**?
- Learning library APIs is a **depreciating skill**
- Better to **learn fundamentals** – and reach for them sooner

Thanks!

Explosion AI

explosion.ai

📲 **Follow us on Twitter**

@explosion_ai

@honnibal

@_inesmontani