Introduction to Julia

August 2018

Julia: What

High-level dynamic programming language designed to address the needs of high-performance numerical analysis and computational science

Started in 2009 by academics Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman

No official reason for the name "Julia"

Current version: 0.7(as of June 2018)

Julia History

Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman

> *Soon the team was building their dream language. MIT, where Bezanson is a graduate student, became an anchor for the project, with much of the work being done within computer scientist and mathematician Alan Edelman's research group. But development of the language remained completely distributed. "Jeff and I didn't actually meet until we'd been working on it for over a year, and Viral was in India the entire time," Karpinski says. "So the whole language was designed over email."*
>
> — "Out in the Open: Man Creates One Programming Language to Rule Them All" - http://www.wired.com/2014/02/julia/

Julia: Why

Designed for high performance

Dynamically-typed and optionally typed, feels like a scripting language

Built-in asynchronous I/O, process control, logging, profiling, a package manager, and more.

Downloaded over 2 million times and the Julia community has developed over 1,900 Julia packages

Integration with MySQL, JDBC, ODBC, HDFS, Hive, etc.

Julia Syntax

Types

AbstractString
UTF8String
ASCIIString
Char

Simple Usage Examples

Using a char

```
a = 'A'
```

Simple string

```
b = "for Data Science"
```

Interporlation

```
println("Julia is $b")
```

Regular Expression

```
match(r"(\W\w+){,2}", b)
```

Unicode usage

```
println("\u2200 x \u2203 y")
```

Triple Quote

```
json = """{
    "Id": 10232
}"""
```

Concat

```
"Lets" * " code"
```

Repeat sentence

```
repeat("Julia", 10)
```

Julia Syntax

Functions

Basic function definition

```
function f(x,y)
        x + y
end
```

Terse function definition

```
f(x,y) = x+y
```

Optional and keywords args
x: optional; a: keyword

```
f(x, y=1; a=3) = 1
```

Control Flow

Compound expressions

```
Z = begin f(x,y)
        x+ 1
        x + y
end
```

Repeated eval loops

```
while x < 1
for x=1:10
```

Short-circuit evaluation

```
&&, ||
```

Conditional evaluations

```
If x < 10
   x += 1
elseif 10 <= x < 12
   x+= 2
else
    x += 3
end
```

Exception Handling

```
try - catch
```

Tasks (coroutines)

```
yieldto
```

Julia Syntax

Types

Abstract type

```
abstract Integer <: Real
```

Create a composite type

```
type Point
      x::Float64
            y::Float64
end
```

Parallel

Execute parallel command

```
nheads = @parallel (+) for  i=1:200000000
     Int(rand(Bool))
end
```

Packages

Show status

```
Pkg.status()
```

Install a new package

```
Pkg.add("<Package Name>")
```

Remove Package

```
Pkg.rm("Package")
```

Install from GitHub

```
Pkg.clone("Package")
```

Update packages

```
Pkg.update()
```

Julia Syntax

Example Variables

A variable
`x = 10`

Float variable
`y = x + 2.0`

UTF-8 variable
`σ = 1`

Built-in Types

Int8, Int16, Int32, Int64,
Int12, UInt32, UInt64,
UInt128 `103`
Bool `false, true`
AbstractString `“Data!”`
Char `‘Z’`
Float16, Float32, Float64
Complex `1 + 2im`
Rational
`5//6`

Some math functions

round(Int, 76.0)

floor, ceil, trunc, eps, ...
div, rem, mod, gcd, lcm, ...
abs, sqrt, cbrt, exp, log, log2,
sin, cos, tan, cot, sec, hypot,
beta, gamma, eta, zeta, ...

Julia: Metaprogramming

- Julia can represent its own program code as a data structure (`Expr`).
- Three metaprogramming components in Julia:
  - Macros
    - generate an expression from expressions.
    - `Expr ↦ Expr`
  - Generated functions
    - generate an expression from types.
    - `Types ↦ Expr`
  - Non-standard string literals
    - generate an expression from a string.
    - `String ↦ Expr`

Multiple Dispatch

Providing ability to define function behavior across many combinations of argument types

All concrete types are subtypes of abstract types, directly or indirectly subtypes of the Any type, which is the top of the type hierarchy. Concrete types can not be subtyped, but composition is used over inheritance, that is used by traditional object-oriented languages

```
collide_with(x::Asteroid,  y::Asteroid)  = ... # deal with asteroid hitting asteroid
collide_with(x::Asteroid,  y::Spaceship) = ... # deal with asteroid hitting spaceship
collide_with(x::Spaceship, y::Asteroid)  = ... # deal with spaceship hitting asteroid
collide_with(x::Spaceship, y::Spaceship) = ... # deal with spaceship hitting spaceship
```

Development Environment

Juno-Atom Plugin

VSCode- VS Code Extension

Jupyter-Jupyter kernel

Vim-vim plugin

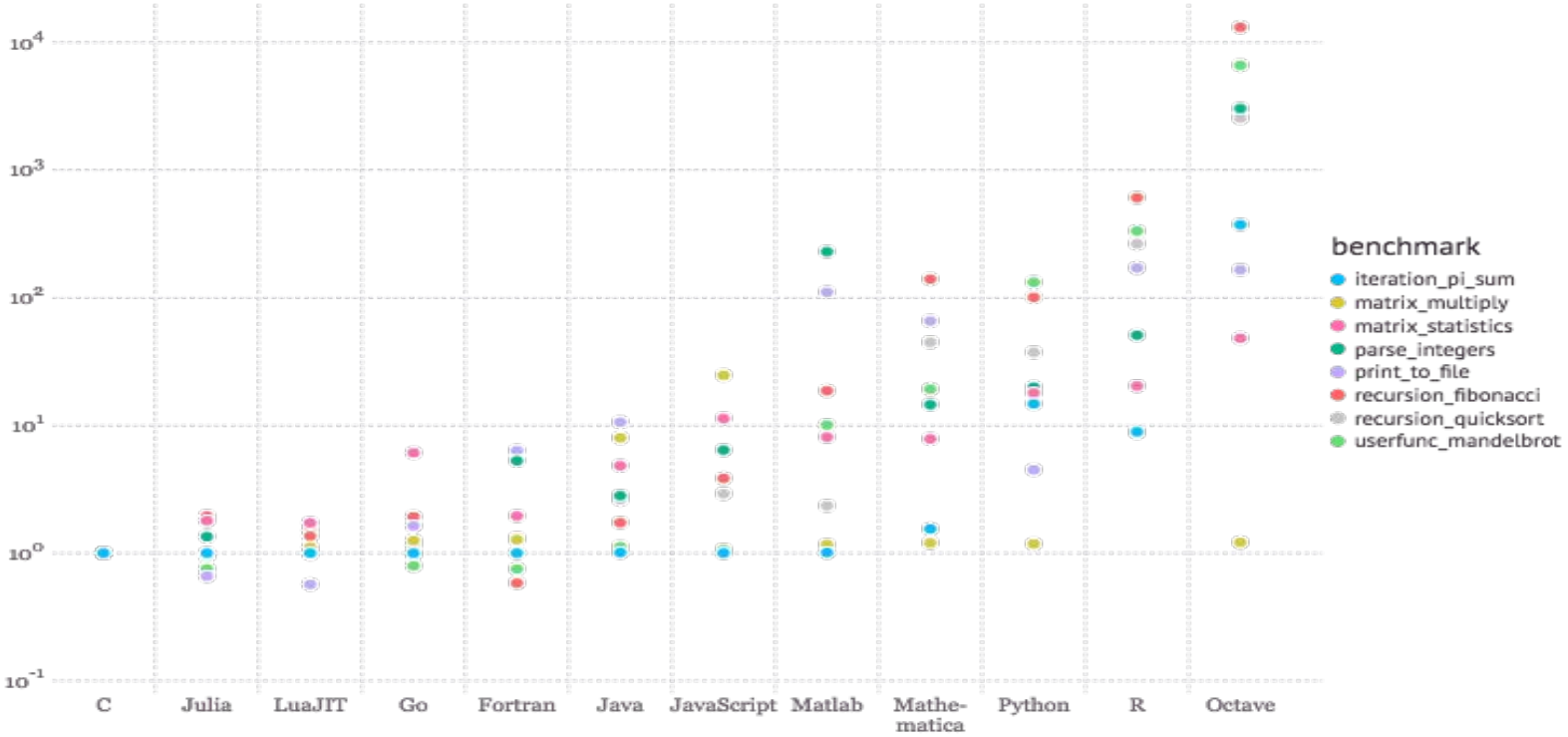
Emacs-Emacs plugin

Sublime-Sublime plugin

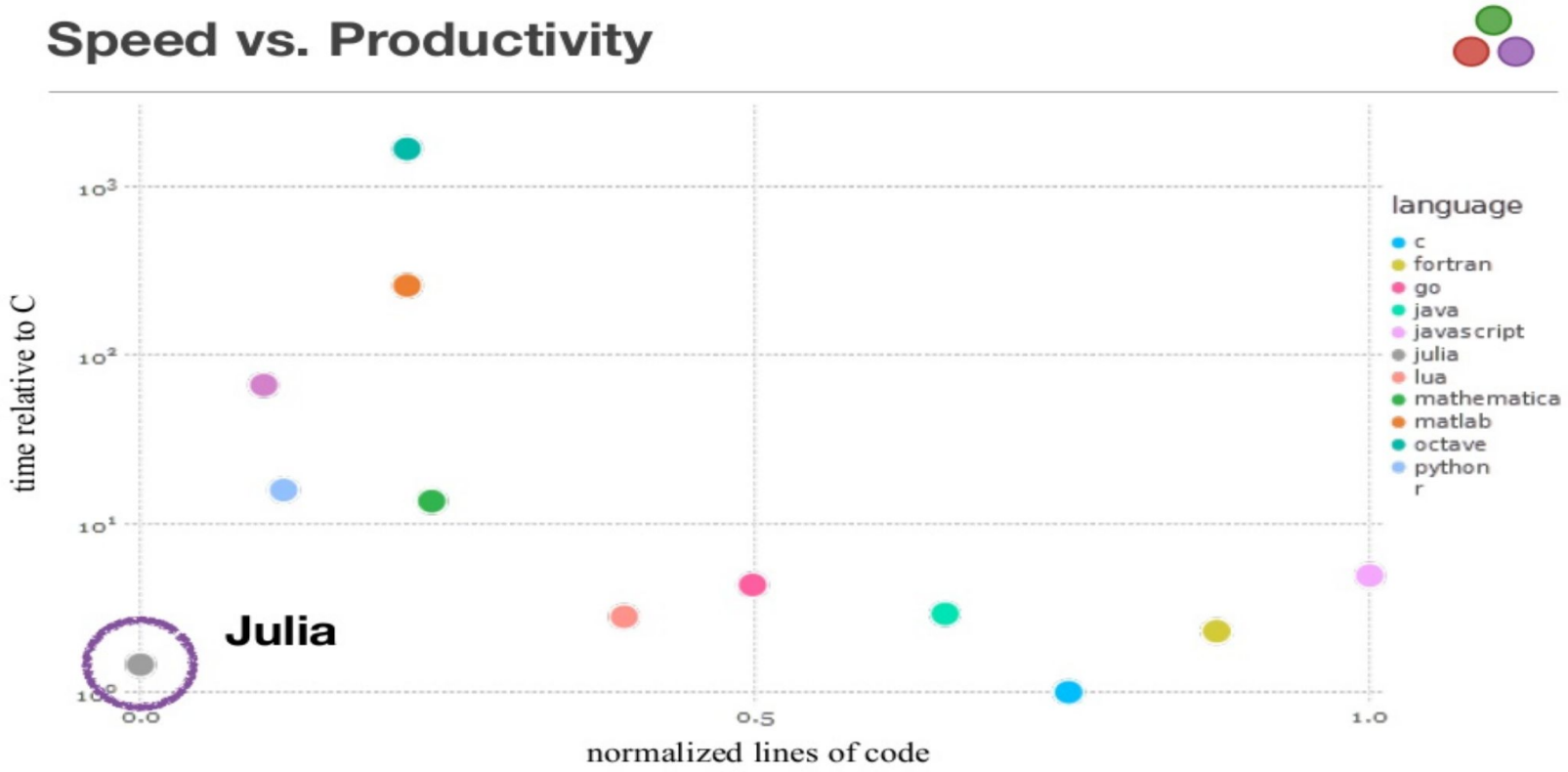
Kaggle Kernels

Julia REPL

Advertised benchmarks



Advertised productivity



Notable Packages

Julia GPU: https://github.com/FluxML/Flux.jl

Julia pre-trained CV models: https://github.com/FluxML/Metalhead.jl

Julia Deep Learning: https://github.com/denizyuret/Knet.jl

Bioinformatics - Bio.jl: https://github.com/BioJulia/Bio.jl

Database - JuliaDB: https://github.com/JuliaDB

Celeste: Used by academic consortium (MIT, UCB, LBNL, Harvard, ...) for petaflop computation of astronomy data

Reviews:

Andre Pemmelaar, 2015, Finance@Quantix Research: Used for LSTMs and RNNS. R:15 minutes, Java: 20 seconds, Julia: 4.3 seconds. Missing documents and garbage collection can be heavier than anticipated.

Pedro Serrano, Actuary, January 2017: Naive Bayes Package has two examples with no comments and is no longer maintained. Data should be in vector or matrix form (not data frame) for consumption by Julia packages. Decision tree package returns single metric rather than confusion matrix, be prepared to modify yourself. Packages can be sensitive to Julia language version.

Reviews:

- Victor Zverovich, Software Engineer, Facebook, 2016: Performance issues including long startup time and JIT lags. Poor text formatting facilities in the language and lack of good unit testing frameworks. Unsafe interface to native APIs by default. Unnecessarily complicated codebase and insufficient attention to bug fixing
- Dan Luu, Software Engineer, Microsoft, 2015: Ran into bugs involving bogus exceptions when processing Unicode strings. Lack of a good story for error handling across packages. Clunky exception handling. “I know one of the co-authors of the O'Reilly Learning Julia book and they have to rewrite examples to work around core bugs all the time.”

Julia: News

Julia 0.7 just released

- Compiler improvements
- Library improvements
- Some breaking changes

JuliaCon (http://juliacon.org/2018/) going on right now in London, videos should be posted soon!

Learn more

Julia Lab: https://julia.mit.edu/

Julia: A Fresh Approach to Numerical Computing: https://julialang.org/publications/julia-fresh-approach-BEKS.pdf

Cheat Sheet: https://juliadocs.github.io/Julia-Cheat-Sheet/

Coursera: https://www.coursera.org/learn/julia-programming

EdX: https://www.edx.org/course/optimization-methods-business-analytics-mitx-15-053x

Getting involved

Github: https://github.com/JuliaLang/julia

Slack: https://julialang.slack.com/

JuliaCon Videos: http://juliacon.org/

Discourse forum: https://discourse.julialang.org/

Introduction to spaCy

August 2018

spaCy: What

An open-source production ready (v2.0) software library for advanced Natural Language Processing, written in the programming languages Python and Cython

Published under MIT license and currently offers statistical neural network models for English, German, Spanish, Portuguese, French, Italian, Dutch and multi-language NER

Supports deep learning workflows that allow connecting statistical models trained by popular machine learning libraries like TensorFlow, Keras, Scikit-learn or PyTorch

spaCy: Who

Matthew Honnibal

CO-FOUNDER

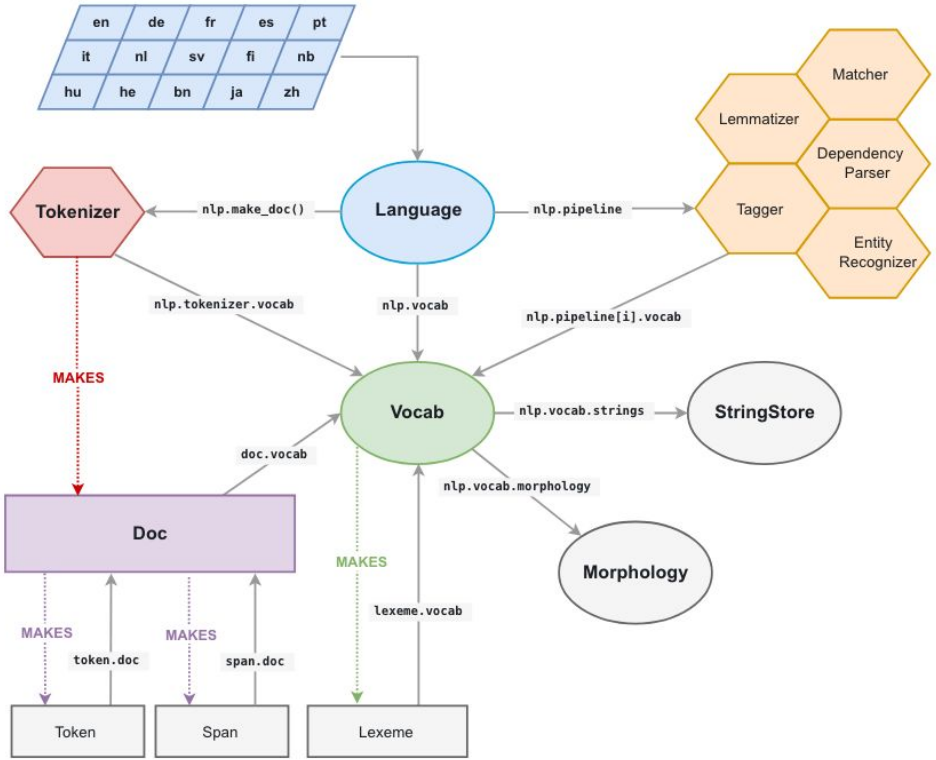
PhD in Computer Science in 2009.
10 years publishing research on state-of-the-art natural language understanding systems.
Left academia in 2014 to develop spaCy.

Ines Montani

CO-FOUNDER

Programmer and front-end developer with degree in media science and linguistics.
Has been working on spaCy since its first release. Lead developer of Prodigy.

spaCy: Architecture



spaCy: Example Workflow

Get things done

Installation:

```
$ pip install spacy
$ python -m spacy download en
```

Load model and process text:

```
import spacy
nlp = spacy.load('en')
doc = nlp('Can you process this text?')
```

Get things done

POS tagging:

```
for token in doc:
    print(token, token.pos_)
```

```
Can VERB
you PRON
process VERB
this DET
text NOUN
? PUNCT
```

spaCy: Example Workflow

Get things done

Named entity recognition:

```
doc = nlp('The current capital of Japan is Tokyo.')
print(doc.ents)
```

```
(Japan, Tokyo)
```

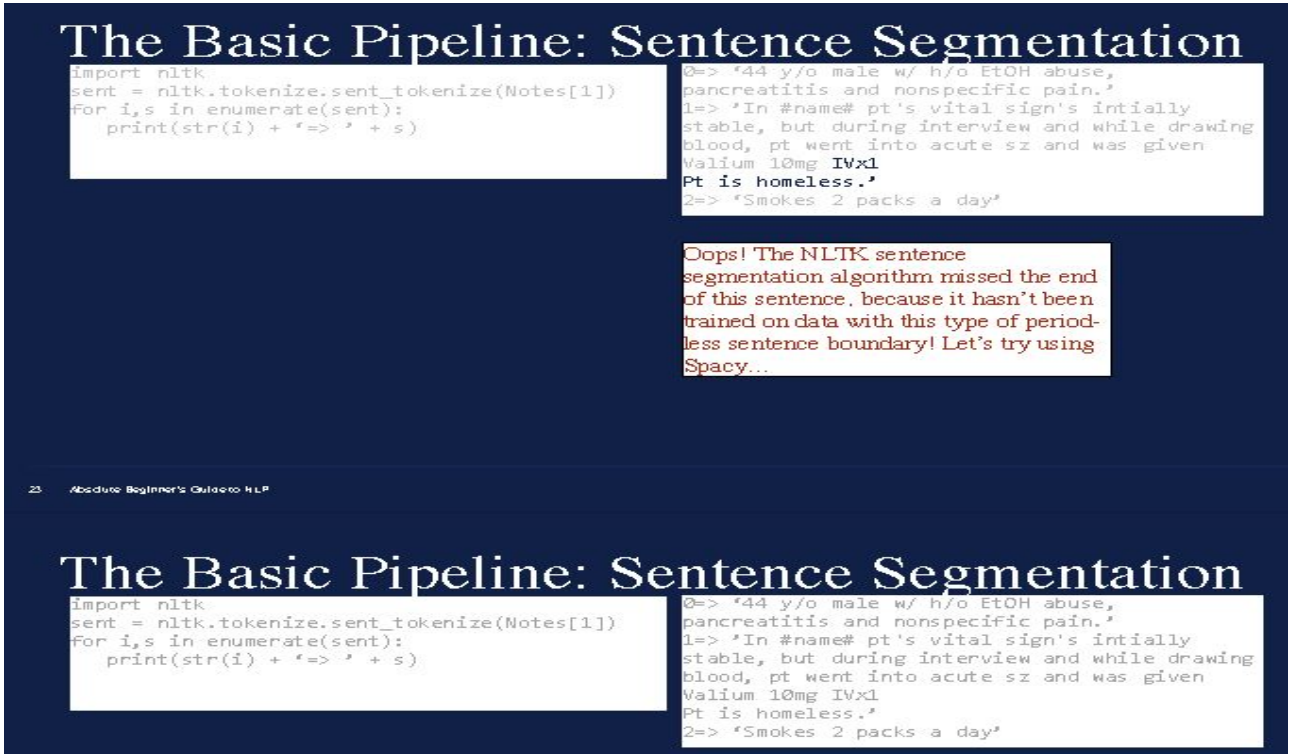
Get things done

Dependency parsing:

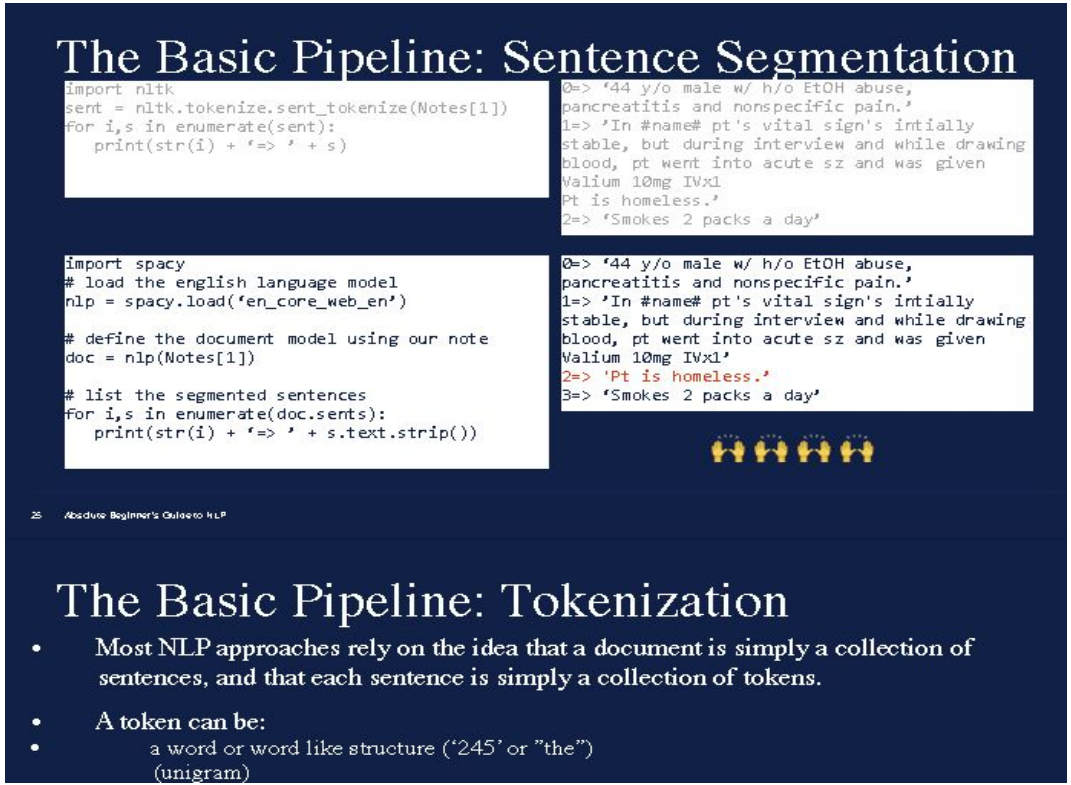
```
for token in doc:
    print('{} -({})-> {}'.format(token.head, token.dep_,
```

```
process -(aux)-> Can
process -(nsubj)-> you
process -(ROOT)-> process
text -(det)-> this
process -(dobj)-> text
process -(punct)-> ?
```

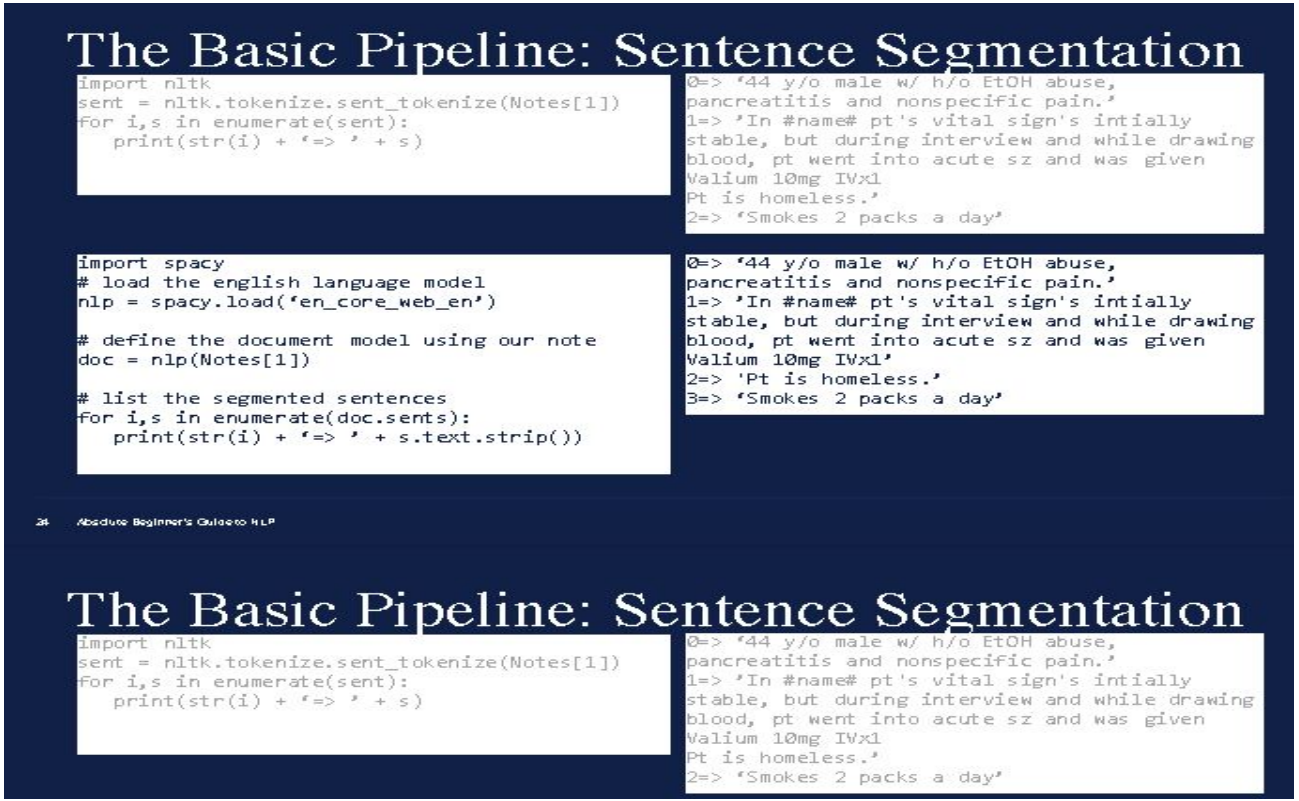
spaCy: Where NLTK fails



spaCy: Where NLTK fails



spaCy: Where NLTK fails



Advertised accuracy

NER accuracy (OntoNotes 5, no pre-process)

This is the evaluation we use to tune spaCy's parameters to decide which algorithms are better than the others. It's reasonably close to actual usage, because it requires the parses to be produced from raw text, without any pre-processing.

SYSTEM	YEAR	TYPE	ACCURACY
spaCy `en_core_web_lg` v2.0.0a3	2017	neural	85.85
Strubell et al.	2017	neural	**86.81**
Chiu and Nichols	2016	neural	86.19
Durrett and Klein	2014	neural	84.04
Ratinov and Roth	2009	linear	83.45

Advertised accuracy

Parse accuracy (Penn Treebank / Wall Street Journal)

This is the "classic" evaluation, so it's the number parsing researchers are most easily able to put in context. However, it's quite far removed from actual usage: it uses sentences with gold-standard segmentation and tokenization, from a pretty specific type of text (articles from a single newspaper, 1984-1989).

SYSTEM	YEAR	TYPE	ACCURACY
spaCy v2.0.0	2017	neural	94.48
spaCy v1.1.0	2016	linear	92.80
Dozat and Manning	2017	neural	**95.75**
Andor et al.	2016	neural	94.44
SyntaxNet Parsey McParseface	2016	neural	94.15
Weiss et al.	2015	neural	93.91
Zhang and McDonald	2014	linear	93.32
Martins et al.	2013	linear	93.10

spaCy: Advertised speed

Two peer-reviewed papers in 2015 confirm that spaCy offers the **fastest syntactic parser in the world** and that **its accuracy is within 1% of the best** available. The few systems that are more accurate are 20× slower or more.

SYSTEM	YEAR	LANGUAGE	ACCURACY	SPEED (WPS)
spaCy v2.x	2017	Python / Cython	**92.6**	*n/a* ⓘ
spaCy v1.x	2015	Python / Cython	91.8	13,963
ClearNLP	2015	Java	91.7	10,271
CoreNLP	2015	Java	89.6	8,602
MATE	2015	Java	92.5	550
Turbo	2015	C++	92.4	349

spaCy: Advertised feature set

	SPACY	SYNTAXNET	NLTK	CORENLP
Programming language	Python	C++	Python	Java
Neural network models	✓	✓	✗	✓
Integrated word vectors	✓	✗	✗	✗
Multi-language support	✓	✓	✓	✓
Tokenization	✓	✓	✓	✓
Part-of-speech tagging	✓	✓	✓	✓
Sentence segmentation	✓	✓	✓	✓
Dependency parsing	✓	✓	✗	✓
Entity recognition	✓	✗	✓	✓
Coreference resolution	✗	✗	✗	✓

textaCy


textacy
higher-level NLP built on spaCy
Documentation / GitHub / API Reference
textacy is a Python library for performing higher-level natural la nguage processing (NLP) tasks, built on the high-performance s paCy library.
textacy focuses on tasks facilitated by the ready availability of t okenized, POS-tagged, and parsed text.
Features
• Stream text, json, csv, and spaCy binary data to and from disk
• Clean and normalize raw text, before analyzing it
• Explore a variety of included datasets, with both text data and metadata from Congressional speeches to historical literature to Reddit comments
• Access and filter basic linguistic elements, such as words and ngrams, noun chunks and sentences
• Extract named entities, acronyms and their definitions, direct quotations, key terms, and more from documents
• Compare strings, sets, and documents by a variety of similarity metrics
• Transform documents and corpora into vectorized and semantic network representations
• Train, interpret, visualize, and save sklearn -style topic models using LSA, LDA, or NMF methods
• Identify a text's language, display key words in context (KWIC), true-case words, and navigate a parse tree
https://github.com/chartbeat-labs/textacy

spaCy: Other notable projects

thinc

spaCy's Machine Learning library for NLP in Python

Assembly ★ 716 85

spacy-models

Models for the spaCy Natural Language Processing (NLP) library

Python ★ 195 36

Berkeley Neural Parser

Constituency Parsing with a Self-Attentive Encoder (ACL 2018)

neuralcoref

State-of-the-art coreference resolution based on neural nets and spaCy

Kindred

Biomedical relation extraction using spaCy

ADAM: Question Answering System

A question answering system that extracts answers from Wikipedia to questions posed in natural language.

Introduction to pyTorch

August 2018

pyTorch: What

Torch: open source machine learning library, a scientific computing framework, and a script language based on Lua (programming language designed primarily for embedded use in applications)

pyTorch: an open source imperative machine learning library for Python, based on Torch

Primarily developed by Facebook's artificial-intelligence research group with two features:

- Tensor computation (like numpy) with strong GPU acceleration
- Deep Neural Networks built on a tape-based autodiff system

Torch: What

- An open source machine learning library, a scientific computing framework based on the *Lua programming language*(similar to javascript) with strong CPU and CUDA backends.
- **Goal**: to have maximum flexibility and speed in building your scientific algorithms while making the process extremely simple.
- **Original author**: Ronan Collobert, Koray Kavukcuoglu, Clement Farabet
- **Initial release**: October 2002(**NYU/Facebook**)
- **Operating system**: Linux, Android, Mac OS X, iOS
- **Applications**
  - It is used by the **Facebook** AI research group, IBM, Yandex, and the Idiap Research Institutes.

Torch: Why

Strong points of Torch

- *It's simple to use, while having maximum flexibility in implementing complex neural network topologies.*
- Efficient Tensor library with an efficient CUDA backend
- Good community and industry support(several hundred community) built and maintained packages
- We can make arbitrary graphs of neural networks, and parallelize them over CPUs and GPUs in an efficient manner.
- It has tons of built-in modules and loss functions

Torch Structures

1] Tensors

Tensor is a general name of multi-way array data

It can easily change data type like numpy

It's easy to use GPU unlike numpy

2] Modules

Modules are classes written in Lua using tensor methods

It's easy to read and write

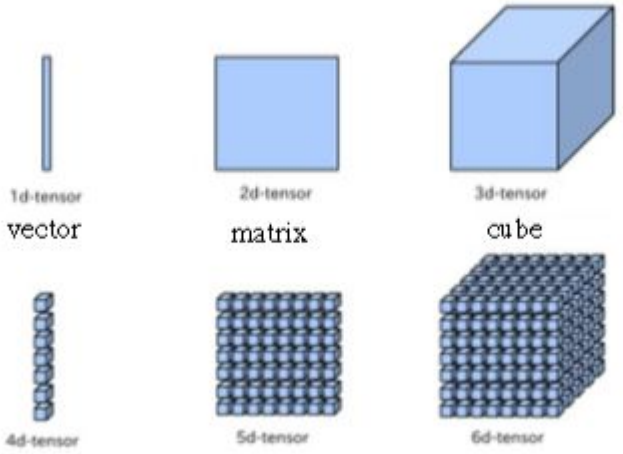
Lots of package

nn: It lets you easily build and train neural networks

cunn: Running on GPU is easy

Torch: Tensors

- Tensor is a general name of multi-way array data
- It is defined mathematically, are simply arrays of numbers, or functions, that transform according to certain rules under a change of coordinates.



Torch: torch.nn

Neural networks can be constructed using the torch.nn package.

Forward

An nn.Module contains layers, and a method forward(input) that returns the output

You can use any of the Tensor operations in the forward function

Backward

nn depends on autograd to define models and differentiate them

You just have to define the forward function, and the backward function (where gradients are computed) is automatically defined for you using autograd

Torch vs. pyTorch

Torch	PyTorch
Lua	Python
No autograd	Autograd
More stable	Newer, still changing
Lots of existing code	Less existing code
Fast	Fast
1st generation(developed by academia)	next generation(originated from industry)

pyTorch: Who


Developers
Facebook Open Source
NVIDIA
salesforce
ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIE PARIS INSTITUTE OF TECHNOLOGY
Carnegie Mellon University
UNIVERSITE PIERRE & MARIE CURIE
UBER
Stanford University
UNIVERSITY OF OXFORD
NYU
Inria
ENS
ÉCOLE NORMALE SUPÉRIEURE
VisionLabs
MACHINES CAN SEE
Digital Reasoning

pyTorch: Levels of abstraction

[1] Tensor

- Imperative N-dimensional array like numpy
- But it runs on GPU unlike numpy

[2] Variable

- Node in a computational graph
- It stores data and gradient

[3] Module

- A neural network layer
- It may store state or learnable weights

pyTorch: Example code

Matrix Multiplication in PyTorch

```
import torch

mat1=torch.randn(2,3)
mat2=torch.randn(3,3)
res=torch.mm(mat1,mat2)

print res.size()
```

Output:

(2L, 3L)

pyTorch: Operations

Many Tensor operations in PyTorch……

torch.mm
- Matrix multiplication

torch.bmm
- Batch matrix multiplication

torch.cat
- Tensor Concatenation

torch.sqeueeze/torch.unsqueeze
- Change Tensor dimensions

…..

…..

Check documentation at http://pytorch.org/docs/master/torch.html#tensors

pyTorch: Variables

A PyTorch Variable is a wrapper around a PyTorch Tensor, a
node in a computational graph

```
import torch
from torch.autograd import Variable

#PyTorch Tensor
x = torch.ones(2,2)
y = torch.ones(2,1)
w = torch.randn(2,1)
b = torch.randn(1)

#PyTorch Variable
x = Variable(x, requires_grad=False)
y = Variable(y, requires_grad=False)
w = Variable(w, requires_grad=True)
b = Variable(b, requires_grad=True)
```

pyTorch: Example

Define a Network Class

```
import torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # create layers

    def forward(self, x):
        # define your feed-forward function

    # define your helper functions
    def helper_function_1():


    def helper_function_1():
```

pyTorch: Example

Compute Loss

input is a random image
target is a dummy label

```
input = Variable(torch.randn(1, 1, 32, 32))
output = net(input)
target = Variable(torch.arange(1, 11))  # a dummy target, for example
criterion = nn.MSELoss()

loss = criterion(output, target)
print(loss)
```

pyTorch: Example

Backpropagation

Use torch.optim package to do backpropagation

```
import torch.optim as optim

# create your optimizer
optimizer = optim.SGD(net.parameters(), lr=0.01)

# in your training loop:
optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
loss.backward()
optimizer.step()    # Does the update
```

pyTorch: Example

Define a CNN Network Class

```
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square convolutio
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = x.view(-1, self.num_flat_features(x))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def num_flat_features(self, x):
        size = x.size()[1:]  # all dimensions except the batch dimension
        num_features = 1
        for s in size:
            num_features *= s
        return num_features
```

pyTorch: Notable projects

translate

Translate - a PyTorch Language Library

machine-learning · pytorch · artificial-intelligence · onnx

Python · ★ 268 · 41 · BSD-3-Clause · Updated 18 hours ago

ignite

High-level library to help with training neural networks in PyTorch

python · machine-learning · deep-learning · neural-network · pytorch

Python · ★ 587 · 63 · BSD-3-Clause · 5 issues need help · Updated 21 hours ago

examples

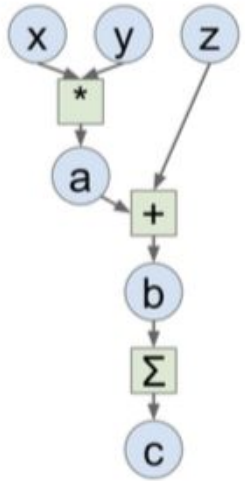
A set of examples around pytorch in Vision, Text, Reinforcement Learning, etc.

Python · ★ 5,556 · 2,528 · BSD-3-Clause · Updated a day ago

tutorials

Python · ★ 1,563 · 579 · BSD-3-Clause · Updated 3 days ago

pyTorch vs. numPy



Computational graphs

	Numpy	PyTorch

Define variable	`import numpy as np`
	`np.random.seed(0)`
	
	`N, D = 3, 4`
	
	`x = np.random.randn(N, D)`
	`y = np.random.randn(N, D)`
	`z = np.random.randn(N, D)`
		`import torch`
		`from torch.autograd import Variable`
		
		`N, D = 3, 4`
		
		`x = Variable(torch.randn(N, D),`
		`requires_grad=True)`
		`y = Variable(torch.randn(N, D),`
		`requires_grad=True)`
		`z = Variable(torch.randn(N, D),`
		`requires_grad=True)`

Forward pass	`a = x * y`
	`b = a + z`
	`c = np.sum(b)`
		`a = x * y`
		`b = a + z`
		`c = torch.sum(b)`

Compute the gradients	`grad_c = 1.0`
	`grad_b = grad_c * np.ones((N, D))`
	`grad_a = grad_b.copy()`
	`grad_z = grad_b.copy()`
	`grad_x = grad_a * y`
	`grad_y = grad_a * x`
		`c.backward()`
		
		`print(x.grad.data)`
		`print(y.grad.data)`
		`print(z.grad.data)`

* Extracted image from cs231n Spring 2017: Lecture 8

pyTorch: Comparison to other frameworks

Software	Creator	Software license[4]	Open source	Platform	Written in	Interface	OpenMP support	OpenCL support	CUDA support	Parallel execution (multi node)	Automatic differentiation[1]	Has pretrained models	Recurrent nets	Convolutional nets	RBM/DBNs
Caffe	Berkeley Vision and Learning Center	BSD license	Yes	Linux, macOS, Windows[2]	C++	Python, MATLAB, C++	Yes	Under development[3]	Yes	Yes	Yes[4]	Yes	Yes	Yes	?
Deeplearning4j	Skymind engineering team; Deeplearning4j community; originally Adam Gibson	Apache 2.0	Yes	Linux, macOS, Windows, Android (Cross-platform)	C++, Java	Java, Scala, Clojure, Python (Keras), Kotlin	Yes	On roadmap[5]	Yes[6][7]	Computational Graph	Yes[8]	Yes	Yes	Yes	Yes[9]
Keras	François Chollet	MIT license	Yes	Linux, macOS, Windows	Python	Python, R	Only if using Theano as backend	Under development for the Theano backend (and on roadmap for the TensorFlow backend)	Yes	Yes	Yes[10]	Yes	Yes	Yes	Yes[11]
MATLAB + Neural Network Toolbox	MathWorks	Proprietary	No	Linux, macOS, Windows	C, C++, Java, MATLAB	MATLAB	No	No	Train with Parallel Computing Toolbox and generate CUDA code with GPU Coder[12]	No	Yes[13][14]	Yes[13]	Yes[13]	Yes[13]	With Parallel Computing Toolbox[15]
Microsoft Cognitive Toolkit	Microsoft Research	MIT license[16]	Yes	Windows, Linux[17] (macOS via Docker on roadmap)	C++	Python (Keras), C++, Command line,[18] BrainScript[19] (.NET on roadmap[20])	Yes[21]	No	Yes	Yes	Yes[22]	Yes[23]	Yes[23]	Yes	Yes[24]
Apache MXNet	Apache Software Foundation	Apache 2.0	Yes	Linux, macOS, Windows,[25][26] AWS, Android,[27] iOS, JavaScript[28]	Small C++ core library	C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl	Yes	On roadmap[29]	Yes	Yes[30]	Yes[31]	Yes	Yes	Yes	Yes[32]
PyTorch	Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan	BSD license	Yes	Linux, macOS, Windows	Python, C, CUDA	Python	Yes	Via separately maintained package[33][34][35]	Yes	Yes	Yes	Yes	Yes		Yes
TensorFlow	Google Brain team	Apache 2.0	Yes	Linux, macOS, Windows,[36] Android	C++, Python, CUDA	Python (Keras), C/C++, Java, Go, R[37], Julia	No	On roadmap[38] but already with SYCL[39] support	Yes	Yes[40]	Yes[41]	Yes	Yes	Yes	Yes
Theano	Université de Montréal	BSD license	Yes	Cross-platform	Python	Python (Keras)	Yes	Under development[42]	Yes	Yes[43][44]	Through Lasagne's model zoo[45]	Yes	Yes	Yes	Yes[46]
Torch	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet	BSD license	Yes	Linux, macOS, Windows,[47] Android,[48] iOS	C, Lua	Lua, LuaJIT,[49] C, utility library for C++/OpenCL[50]	Yes	Third party implementations[51][52]	Yes[53][54]	Through Twitter's Autograd[55]	Yes[56]	Yes	Yes	Yes	Yes[57]

pyTorch: News

Facebook announced Pytorch 1.0, an updated version of the popular AI framework Pytorch, that aims to make it easier for developers to use neural network systems in production.

On the second day of its developer conference F8 in San Jose, California, CTO Mike Schroepfer, introduced Pytorch 1.0, and said it combines Pytorch, Caffe 2, with Open Neural Network Exchange (ONNX).

pyTorch: Resources

AAN: search engine for resources and papers

– http://tangra.cs.yale.edu/newaan/

• Richard Socher's Stanford class

– http://cs224d.stanford.edu/