Natural Language Processing with Julia

Pontus Stenetorp

Post-doctoral Research Fellow, University of Tokyo

http://pontus.stenetorp.se/

Natural Language Processing (NLP)

A brief introduction

What is NLP?

- Artificial Intelligence
- Process natural language using computers
- Language understanding

Why is NLP?

Theoretical:

- Human to human communication interface
- Our storage format
- Yet, largely a "mystery"

Practical:

- A window into the mind.
- "The Hathaway Effect"

NLP History

1930s:

- Word-to-word translation, by punch cards

1940s to 1950s:

- First computer systems

1960s to 1980s:

- Large-scale rule-based systems

1990s to present:

- Statistical revolution

It has been over 50 years:

- Where is HAL 9000? Flying cars? Conversational robots?

Machine Translation in the 60s

"Out of sight, out of mind."

▼

"Invisible Idiot"

"The spirit is willing, but the flesh is weak."

▼

"The liqour is holding out all right, but the meat has spoiled."

Why is language difficult?

- Context, context, context...
- Local context: "I stood at the bank"
- Social context
- Historical context
- Compression, not decryption
- Effectively AI-complete

The current state of NLP

"Fundamental" tasks:

- Parsing
- Word sense disambiguation

Practical tasks:

- Sentiment classification
- Voice recognition

How I met Julia

Meet the candidates

- C
- C (OpenBLAS)
- Python (NumPy/OpenBLAS)
- Matlab
- Octave
- Julia

Recursive Neural Network

Input: $x_{1,2} \in \mathbb{R}^d$

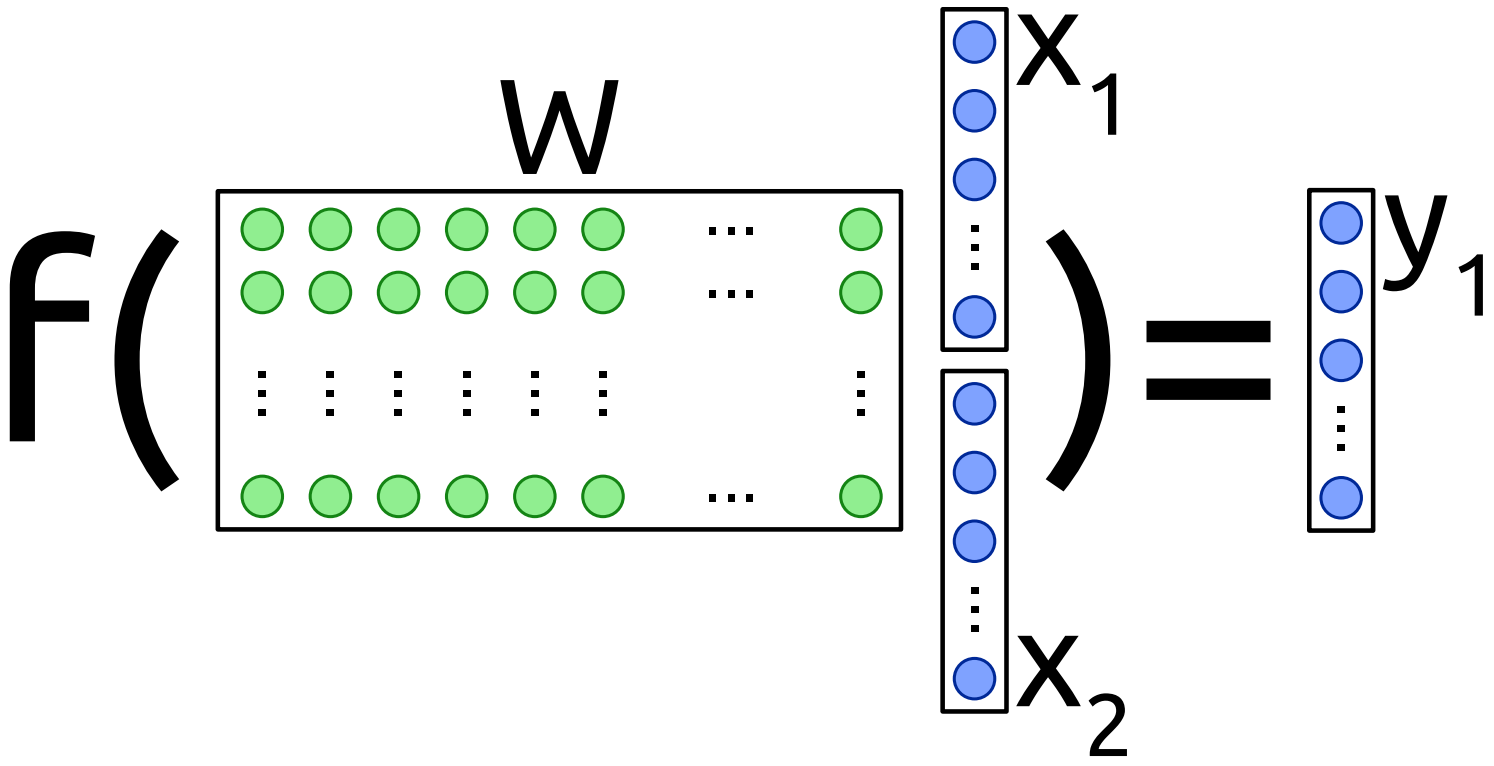
Weights: $W \in \mathbb{R}^{d \times 2d}$

Non-linearity: f

Output: $y_1 \in \mathbb{R}^d$

$$\mathrm{forward}(x_1, x_2) = \mathrm{f}(W[x_1; x_2])$$

Or... visually



Forward pass

$$\text{forward}(x_1, x_2) = \text{f}(W[x_1; x_2])$$

```
function forward(W, x, y)
    for row in 1:size(W, 1)
        y[row] = 0
        for col in 1:size(W, 2)
            y[row] += W[row, col] * x[col]
        end
        y[row] = tanh(y[row])
    end
end
```

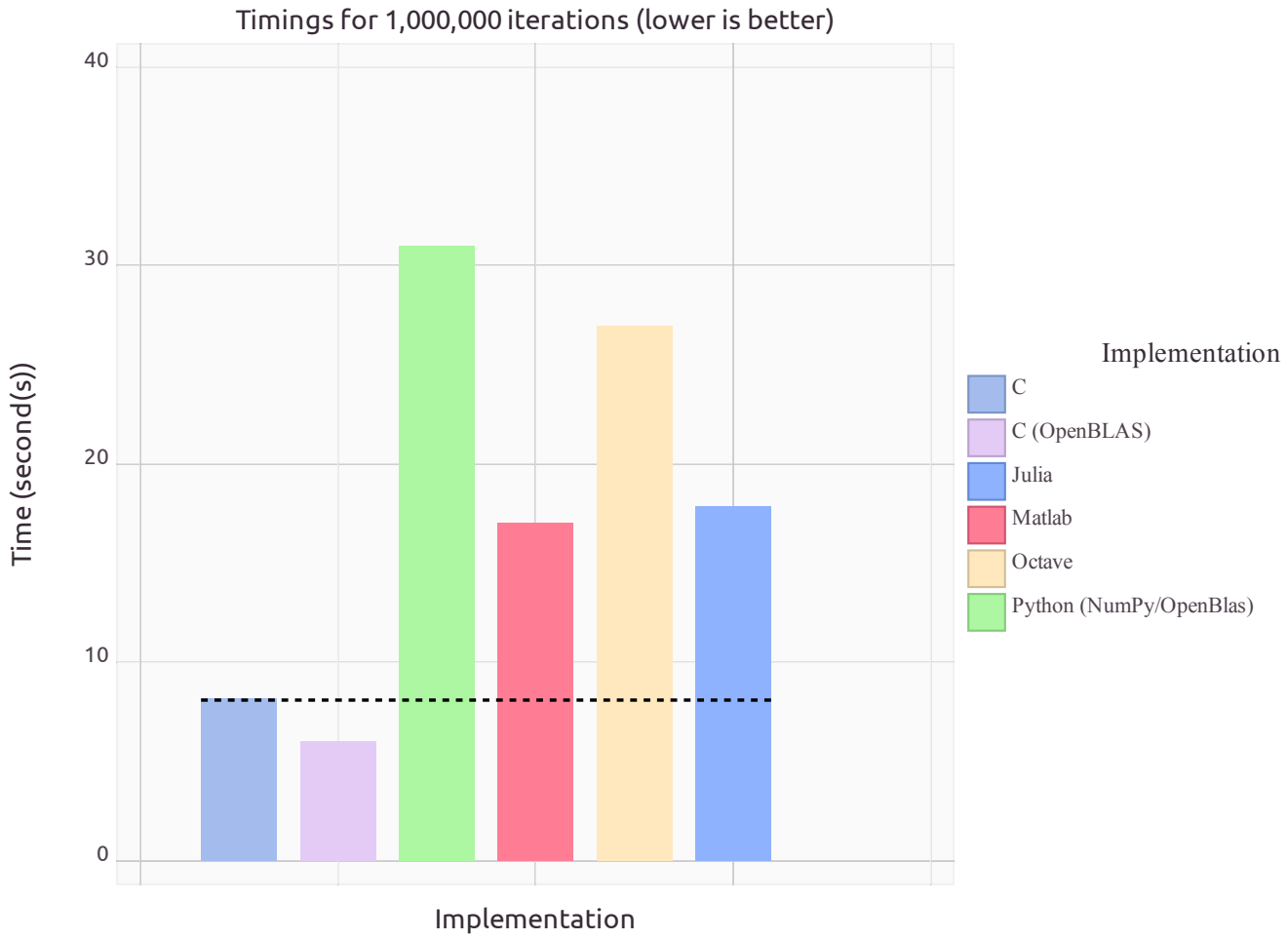
Backward pass

Error: $e \in \mathbb{R}^d$

$$\text{backward}(x_1, x_2, e) = (\text{f}'(\text{forward}(x_1, x_2)) \cdot e)[x_1; x_2]^\intercal$$

```
function backward(x, y, e, b)
    for row in 1:size(b, 1)
        yd = (1.0 - y[row] ^ 2) * e[row]
        for col in 1:size(b, 2)
            b[row, col] = yd * x[col]
        end
    end
end
```

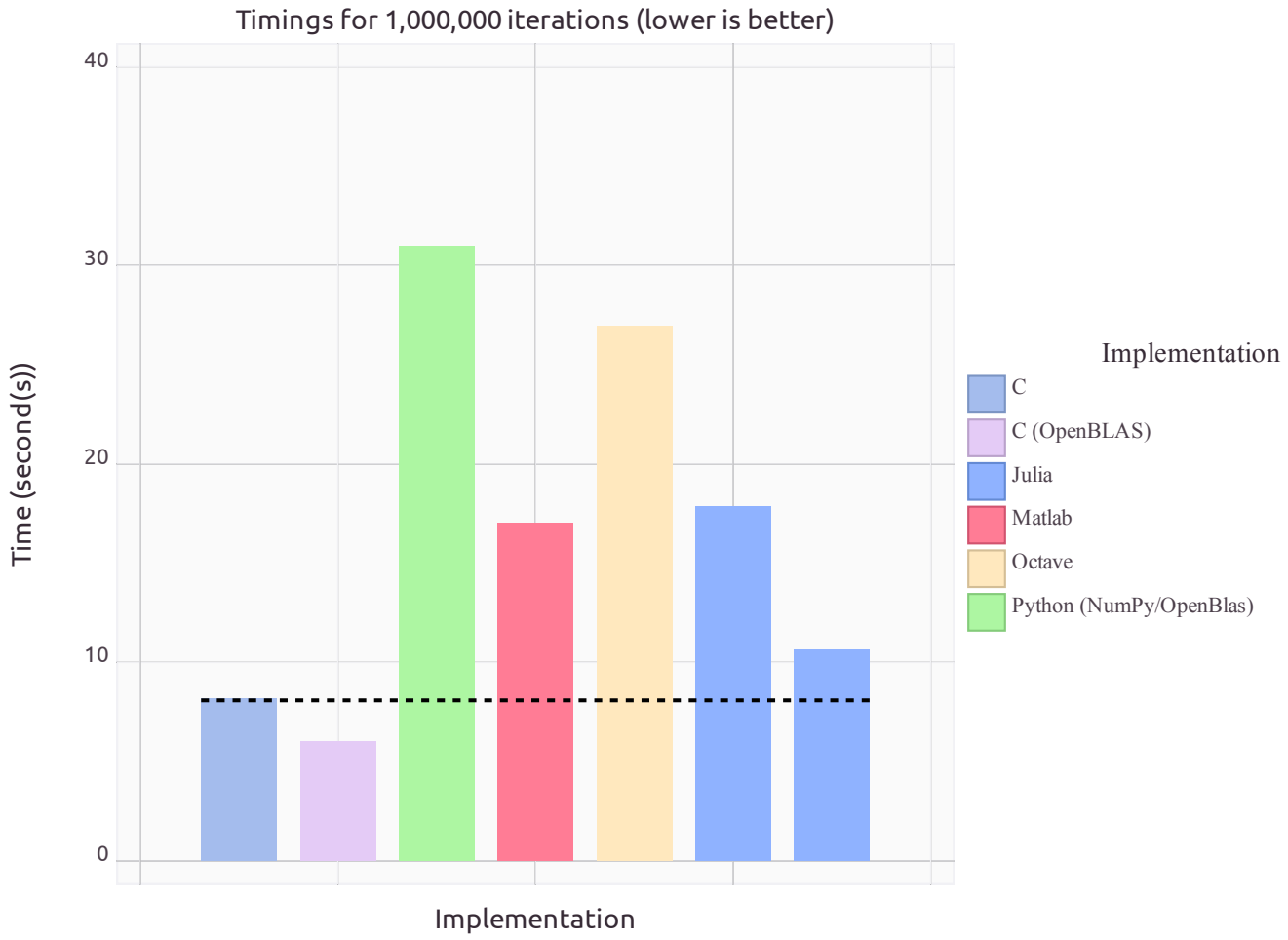
Results


Timings for 1,000,000 iterations (lower is better)
Time (second(s))
40
30
20
10
0
Implementation
Implementation
C
C (OpenBLAS)
Julia
Matlab
Octave
Python (NumPy/OpenBlas)

Saved by a Kiwi?

- Not better than Matlab...
- Back to Python...
- Moping on Hacker News
- A miracle occurs, `@inbounds`!

Results (again)



Julia, a perfect match?

"Feature Engineer"

1. Acquire Data
2. Preprocessing
3. Feature generation
4. Training (LIBSVM, LIBLINEAR, etc.)
5. Analysis
6. `if (publishable) GOTO 9`
7. Feature engineering
8. `GOTO 3`
9. Profit!

Their needs

- Excellent string handling ✓
- Language processing support ✗
- Learning library bindings ≈
- Productivity ✓

"Machine Learner"

1. Acquire data
2. Preprocessing
3. Feature generation
4. Training
5. Analysis
6. `if (publishable) GOTO 9`
7. Modify training algorithm
8. `GOTO 4`
9. Profit!

Their needs

- Excellent maths support ✓
- Performance ✓
- C/C++/Fortran inter-op ✓
- Productivity ✓

Researcher feedback

I/O performance

```
function f()
    tic()
    for _ in eachline(STDIN)
        pass
    end
    toc()
end

f()
```

```
> wc -l holmes/*.txt
   7728 holmes/hound.txt
   4945 holmes/sign.txt
   5151 holmes/study.txt
   7255 holmes/valley.txt
  25079 total
```

```
> cat holmes/*.txt | julia read.jl
elapsed time: 0.104006909 seconds
```

I/O performance

```
> cat holmes/*.txt | julia read.jl
elapsed time: 0.104006909 seconds
```

```
from sys import stdin
from time import time

start = time()
for _ in stdin:
    pass
print time() - start
```

```
> cat holmes/*.txt | python read.py
0.00209999084473
```

Dictionary performance

```
function f()
    tokens = split(readall(STDIN))
    counts = Dict{UTF8String,Int}()
    tic()
    for token in tokens
        counts[token] = get(counts, token, 1)
    end
    toc()
end

f()
```

```
> cat holmes/*.ss.tok | julia count.jl
elapsed time: 0.290906006 seconds
```

Dictionary performance

```
> cat holmes/*.ss.tok | julia count.jl
elapsed time: 0.290906006 seconds
```

```
from sys import stdin
from time import time

tokens = stdin.read().split()
counts = {}
start = time()
for token in tokens:
    try:
        counts[token] += 1
    except KeyError:
        counts[token] = 0
print time() - start
```

```
> cat holmes/*.ss.tok | python count.py
0.0732431411743
```

Start-up/compilation

```
println("Hello!")
```

```
> time julia noargs.jl
Hello!

real    0m0.296s
user    0m0.264s
sys     0m0.032s
```

Start-up/compilation

```
using ArgParse # Not in the standard library.

argparse = ArgParseSettings()
@add_arg_table argparse begin
    "name"
        required = true
end

pargs = parse_args(argparse)

println("Hello $(pargs["name"])!")
```

```
> time julia args.jl Julia
Hello Julia!

real    0m4.280s
user    0m4.185s
sys     0m0.088s
```

Regular Expressions

```
function f()
    tokens = split(readall(STDIN))
    tic()
    for token in tokens
        if match(r"^[a-z].*ion$", token) == nothing
            continue
        end
        #println(token)
    end
    toc()
end

f()
```

```
> cat holmes/*.ss.tok | julia2 ion.jl
elapsed time: 0.266435235 seconds
```

```
> cat holmes/*.ss.tok | julia ion.jl
elapsed time: 0.091687288 seconds
```

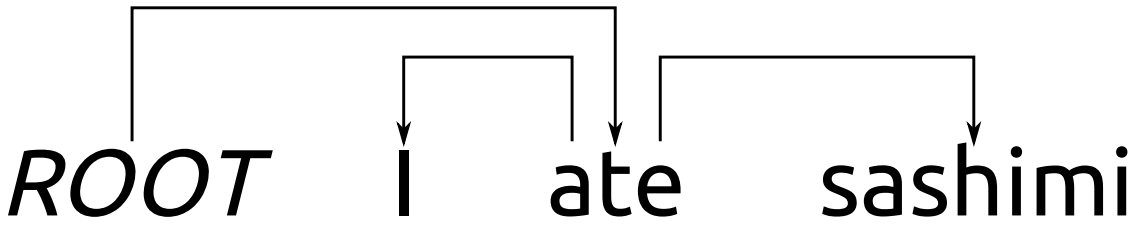
```
> cat holmes/*.ss.tok | julia ion.jl | sort | uniq -c | sort -n -r | head -n
     86 companion
     61 question
     50 expression
```

The Good

- "Wow, fast numerical code!"
- "Trivial to compile/deploy."
- "Extending data structures is easy."
- "Almost too familiar syntax."

Allén

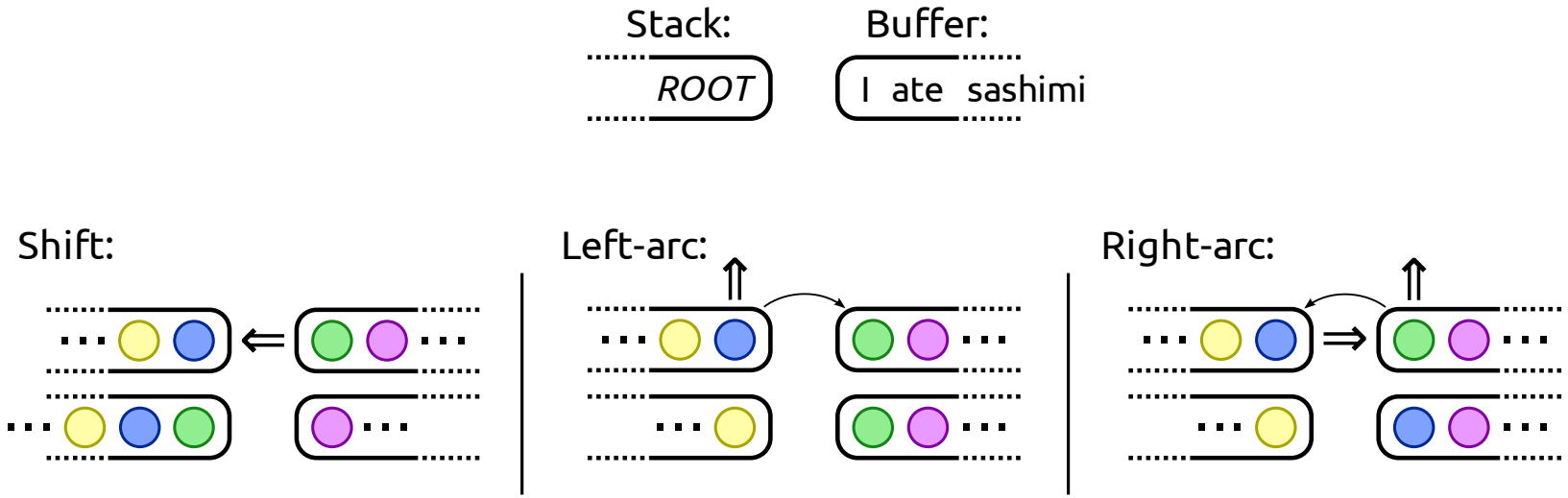
Dependency Parsing



Approaches

Graph-based	$\mathcal{O}(n^3)$
Transition-based	$\mathcal{O}(n)$

Transition-based



Featurisation

```
# 50 to 200 lines like these.
append("s0s1_%s_%s" % (s0,s1))
append("Ts0Ts1_%s_%s" % (Ts0,Ts1))
append("Ts0Tw0_%s_%s" % (Ts0,Tw0))
append("s0Ts0Ts1_%s_%s_%s" % (s0,Ts0,Ts1))
append("Ts0s1Ts1_%s_%s_%s" % (Ts0,s1,Ts1))
append("s0s1Ts1_%s_%s_%s" % (s0,s1,Ts1))
append("s0Ts0s1_%s_%s_%s" % (s0,Ts0,s1))
append("s0Ts0Ts1_%s_%s_%s" % (s0,Ts0,Ts1))
append("s0w0_%s_%s" % (s0,w0))
append("Ts0Tw0_%s_%s" % (Ts0,Tw0))
append("Ts0Tw1_%s_%s" % (Ts0,Tw1))
append("s0Ts0Tw0_%s_%s_%s" % (s0,Ts0,Tw0))
append("Ts0w0Tw0_%s_%s_%s" % (Ts0,w0,Tw0))
append("s0w0Tw0_%s_%s_%s" % (s0,w0,Tw0))
```

Featurisation (with macros)

```
# Still 50 to 200 lines like these.
(s0.form,   s1.form)
(s0.postag, s1.postag)
(s0.postag, b0.postag)

(s0.form,   s0.postag,  s1.postag)
(s0.postag, s1.form,    s1.postag)
(s0.form,   s1.form,    s1.postag)
(s0.form,   s0.postag,  s1.form)
(s1.form,   s1.postag,  b0.postag)

(s0.form,   b0.form)
(s0.postag, b1.postag)

(s0.postag, b0.form,    b0.postag)
(s0.form,   b0.form,    b0.postag)
```

Feature/index mapping

```
type Identifier{T,U<:Integer}
    d::Dict{T,U}
end

Identifier(T::Type=Any,U::Type=UInt) = Identifier(Dict{T,U}())

import Base: length
length(c::Identifier) = length(c.d)
id(c::Identifier, o) = get!(c.d, o, length(c) + 1)
```

Feature/index mapping

```
type Identities{T,U<:Integer}
    d::Dict{T,U}
    default::U
end

function Identities{T,U<:Integer}(c::Identifier{T,U})
    Identities(c.d, zero(U))
end

function Identities{T,U<:Integer}(c::Identifier{T,U}, default::U)
    Identities(c.d, default)
end

import Base: length
length(c::Identities) = length(c.d)
id(c::Identities, o) = get(c.d, o, c.default)
```

Perceptron Learning

Configuration ("State"): c

Featurisation: $\phi(c) \in \mathbb{R}^d$

Weights: $w \in \mathbb{R}^d$

$$\text{score}(c) = \phi(c) w^\intercal$$

Sparse: $d = 4,883,498$

Active: $\approx 0.0001\%$

Perceptron Learning

Python:

- Requires loop
- Implement in Cython/C

Julia:

```
function score(fidxs, weights)
    score = 0
    for fidx in fidxs
        score += weights[fidx]
    end
    return score
end
```

What about speed?

System	*Sentences/second*
State-of-the-art (Cython)	$\approx 1,000$
Allén	≈ 600

```
> wc -l src/*.jl
  164 src/conllx.jl
  215 src/dep.jl
  417 src/hybrid.jl
  326 src/parse.jl
  127 src/percep.jl
   41 src/structs.jl
 1290 total
```

https://github.com/ninjin/allen

Thank you for your attention

ご清聴(せいちょう)ありがとうございました

Tack för er uppmärksamhet