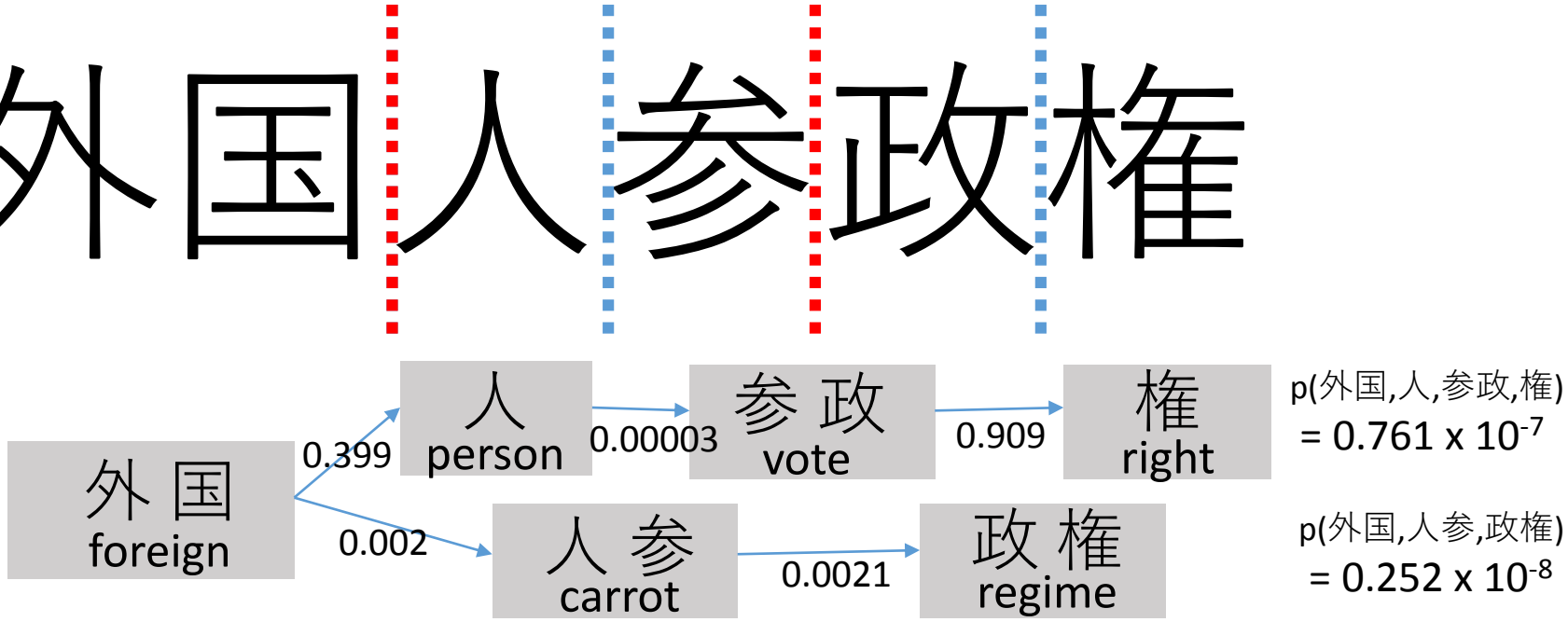
Juman++ v2: A Practical and Modern Morphological Analyzer

Arseny Tolmachev
Kyoto University
Kurohashi-Kawahara lab
2018-03-14

What is Juman++

[Morita+ EMNLP2015]



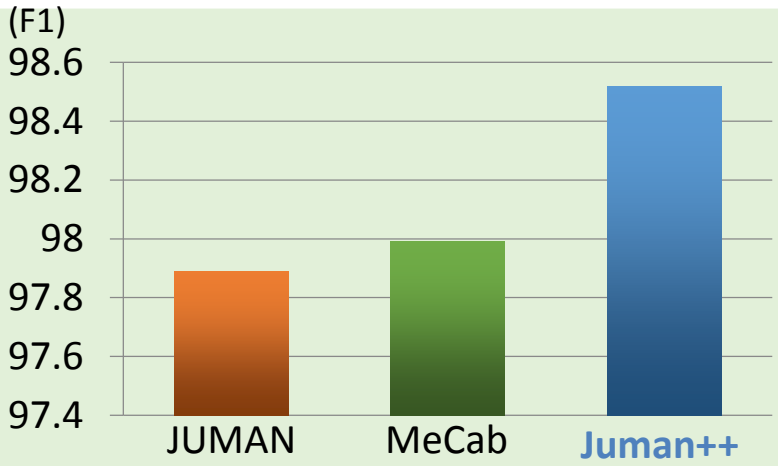
Main idea: Use a Recurrent Neural Network Language Model to consider semantic plausibility in addition to usual model score

Why Juman++? Accuracy!

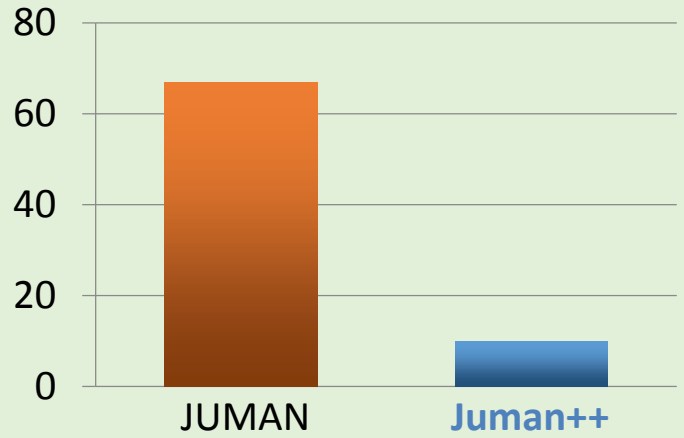
Setting

Dataset for training RNNLM:
10 million raw sentences
crawled from the web

Dataset for training base model and evaluation:
Kyoto University Text Corpus (NEWS),
Kyoto University Web Document Leads Corpus (WEB)



Word segmentation & POS tags



Number of fatal errors in 1,000 sentences

✗ **JUMAN**

感想|やご|要望

a larva of a dragonfly

✓ **Juman++**

感想 | や | ご | 要望

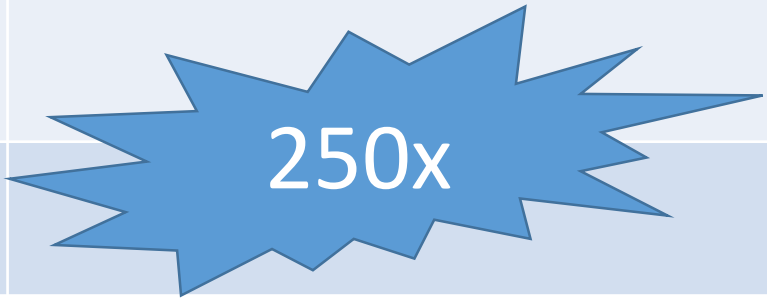
impression and honorific prefix request

Why not Juman++? Speed!

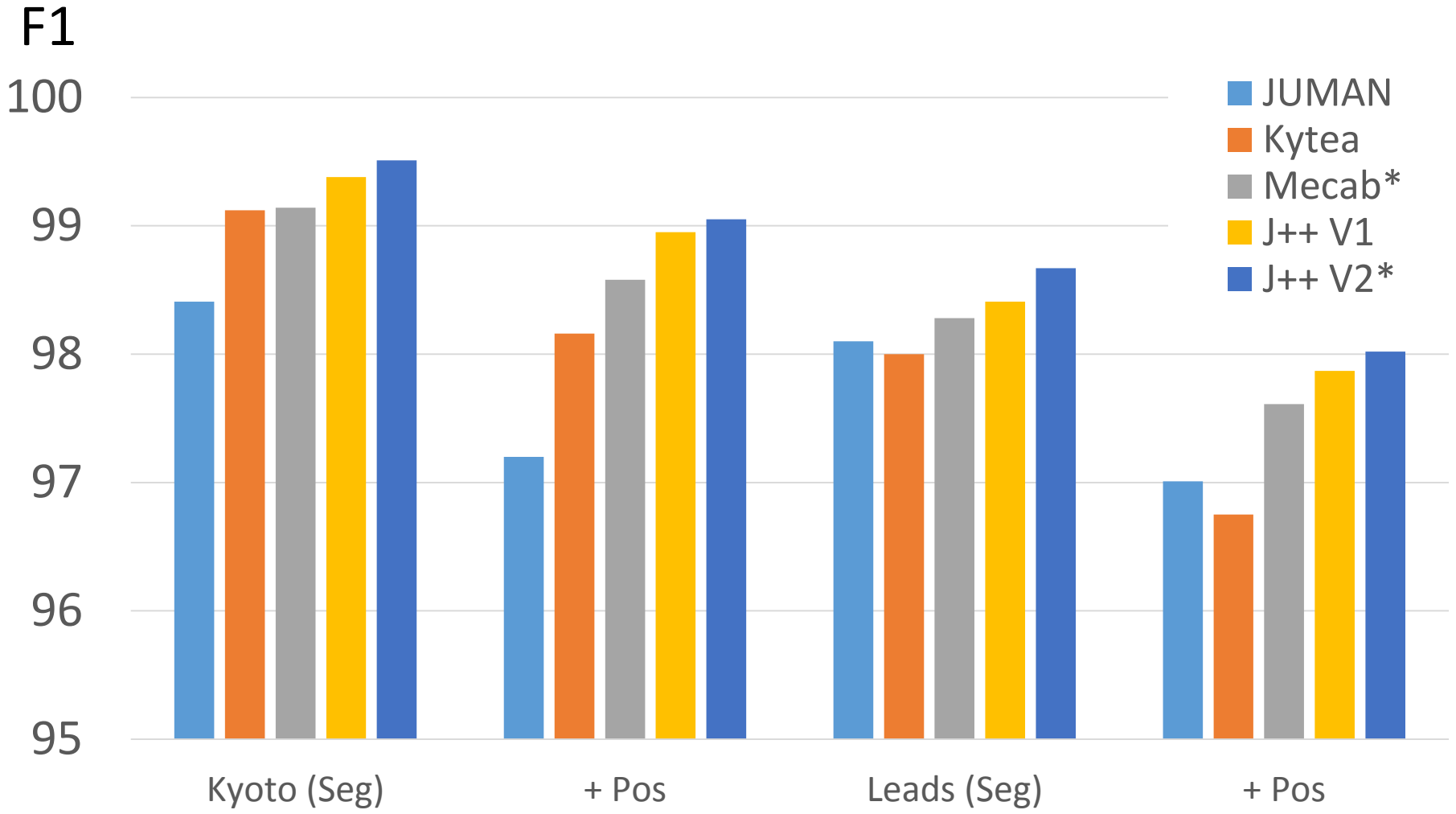
Analyzer	Analysis speed (Sentences/Second)
Mecab	53,000
KyTea	2,000
JUMAN	8,800
Juman++	16

Why not Juman++ V1? Speed!

Analyzer	Analysis speed (Sentences/Second)
Mecab	53,000
KyTea	2,000
JUMAN	8,800
Juman++ V1	16
Juman++ V2	4,800



Why Juman++ V2? Accuracy!!!



* = Optimized hyper-parameters on 10-fold cross-validation
Using the same Jumandic + concatenation of Kyoto/KWDLC corpora for training

What is Juman++ v2 (in its core)

A dictionary independent
thread-safe
library (not just a binary program)
for morphological analysis
using lattice-based segmentation
optimized for n-best output

Reasons of speedup

Algorithmic

- Dictionary representation
- No non-necessary computations
- Reduced search space

(Micro) architectural

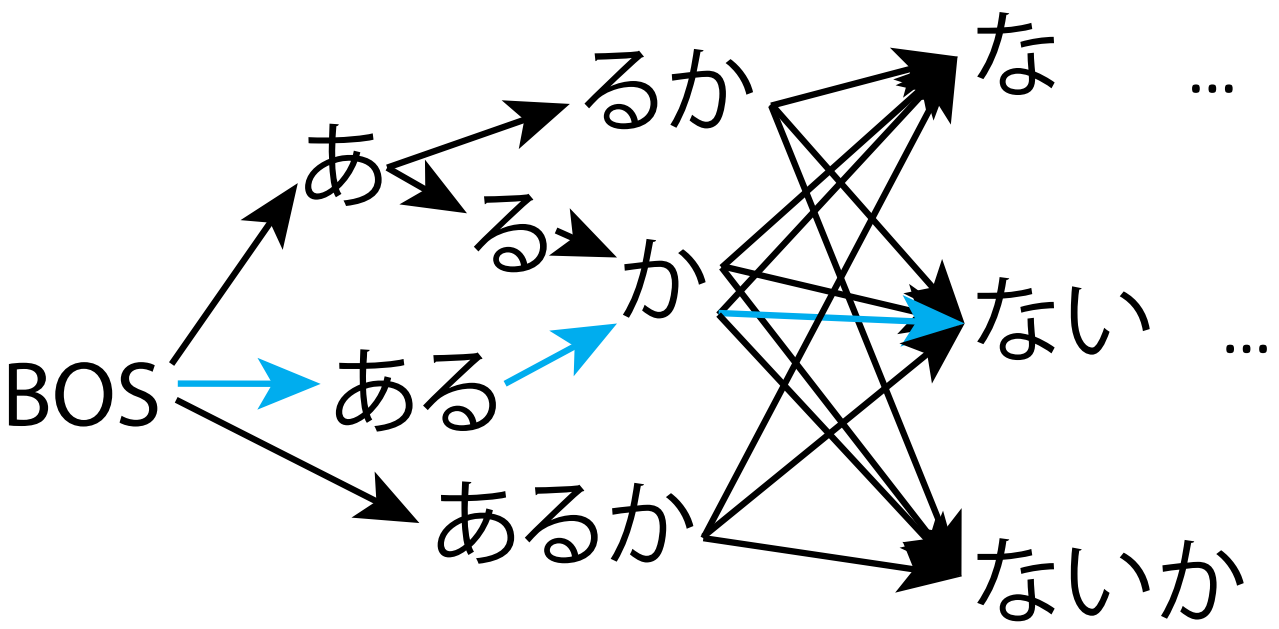
- Cache-friendly data structures
  - Lattice => Struct of arrays
- Code generation
  - Mostly branch-free feature extraction
- Weight prefetching
- RNN-specific: vectorization, batching

Juman++ Model

For an input sentence:

Build a lattice

Assign a score to each path through the lattice



Linear Model

$$s = \sum w_i \phi_i + \alpha(s_{RNN} + \beta)$$

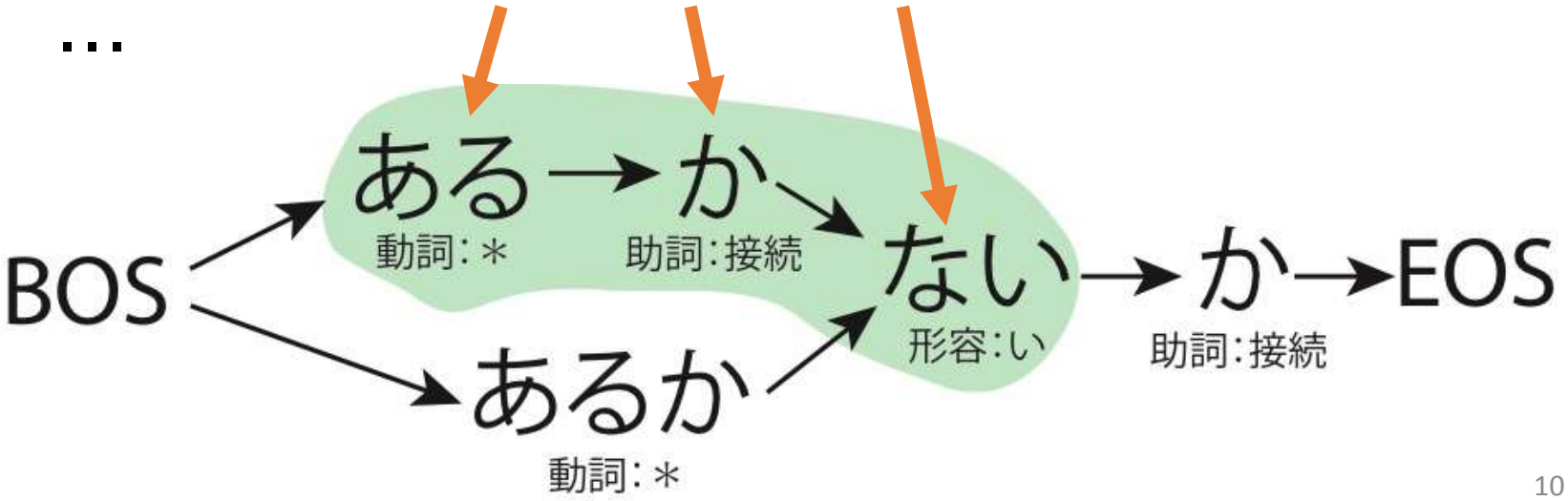
Weights: w_i — Features: ϕ_i

Linear Model Features

Created based on ngram feature templates
Use dictionary info, surface characters, character type
1,2,3-grams
67 ngram feature templates for Jumandic, like:

- BIGRAM (品詞)(品詞)
- BIGRAM (品詞-細分類)(品詞-細分類)
- TRIGRAM (品詞)(品詞)(品詞)

…



Dictionary as column database

Dictionary field values become pointers

ある	動詞	基本形
あるか	動詞	未然形
か	助詞(接)	*
ない	接尾	基本形
か	助詞(終)	*

9	0	0
2	0	6
0	3	4
6	15	0
0	9	4

Field data is deduplicated and each field is stored separately

1か3ある か2ない2ある
2動詞5助 詞(接)5 助詞(終)2接尾
3基本形1＊3未然形

Length

0 5 10 15

Dictionary: Details

- Dictionary is mmap-able
  - Loading from disk is almost free and cacheable by OS
- What dictionary gives us
  - Handle strings as integers (data pointers)
  - Use data pointers as primitive features

- Compute ngram features by hashing components together
- Dictionary + smart hashing implementation = 8x speedup

Dic/model size on Jumandic

	Kytea	Mecab	Juman++ V1	Juman++ V2
Dictionary	-	311M	445M	158M
Model	200M	7.7M	135M	16M

Raw dictionary (CSV with expanded inflections) is 256MB

Kytea doesn't store all dictionary information:
(It uses only surface, pos, subpos information)

Note: 44% of V2 dictionary is DARTS trie

Quiz: Where is the bottleneck?

1. Dictionary lookup/lattice construction
   - Trie lookup, dictionary access
2. Feature computation
   - Hashing, many conditionals
3. Score computation
   - Score += weights[feature & (length – 1)];
4. Output
   - Formatting, string concatenation

Quiz: Where is the bottleneck?

1. Dictionary lookup
   - Trie lookup
2. Feature computat
   - Hashing, many con
3. Score computation
   - Score += weights[feature & (length – 1)];
4. Output
   - Formatting, string concatenation

Array access (not sum) was taking ~80% of all time. Reason: L2 cache/dTLB misses.

Latency numbers every programmer should know

Action	Time	Comments
L1 cache reference	0.5 ns	
Branch mispredict	5 ns	
L2 cache reference	7 ns	14x L1 cache
Main memory reference	100 ns	20x L2 cache, 200x L1 cache

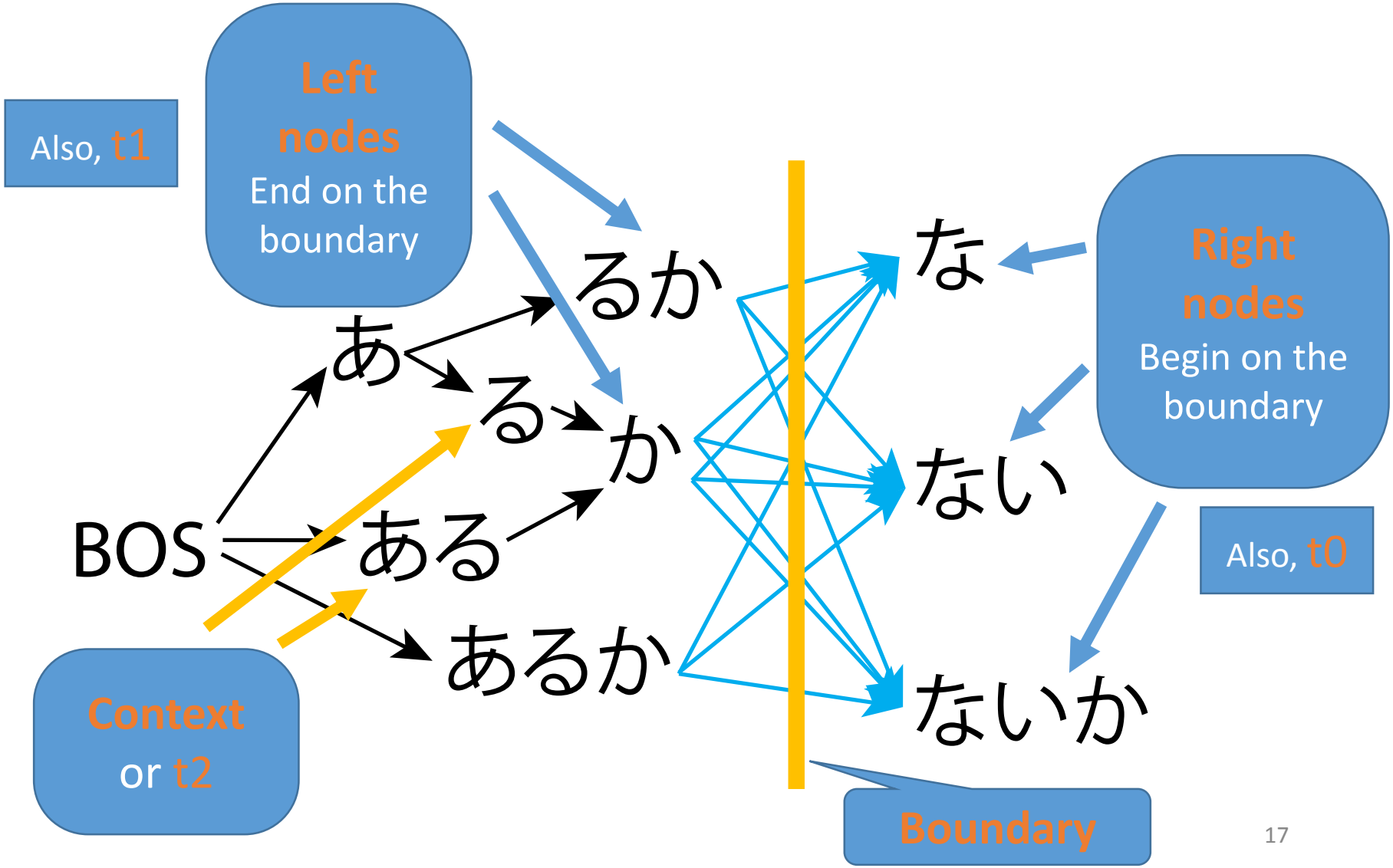
Cache misses are really slow!

Random memory access is almost guaranteed to be a cache miss!

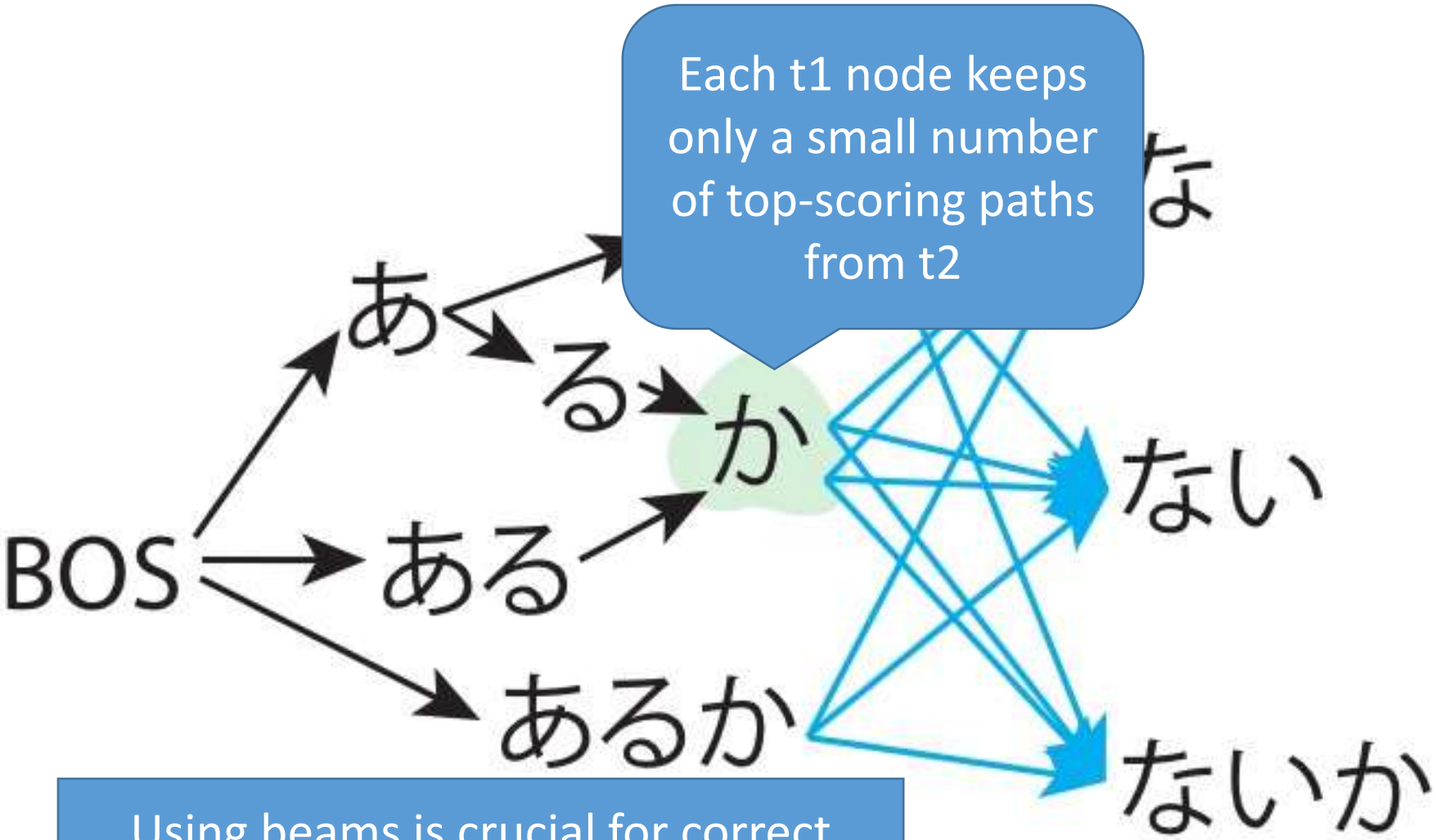
Direction: Reduce number of memory accesses

-> **reduce number of score computations**

Lattice and terms



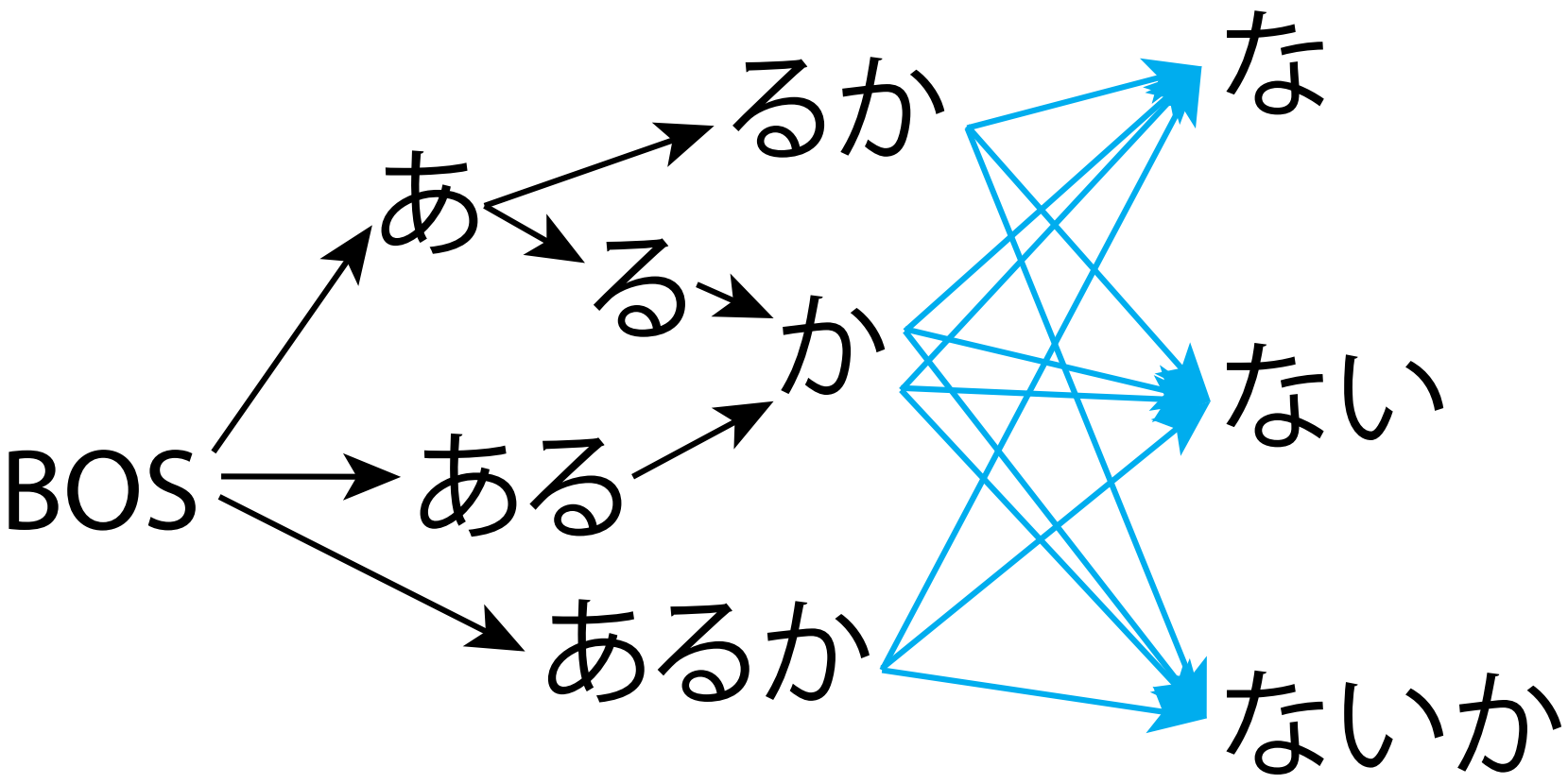
Beams in Juman++



Using beams is crucial for correct usage of high-order features

New in V2: Search space trimming

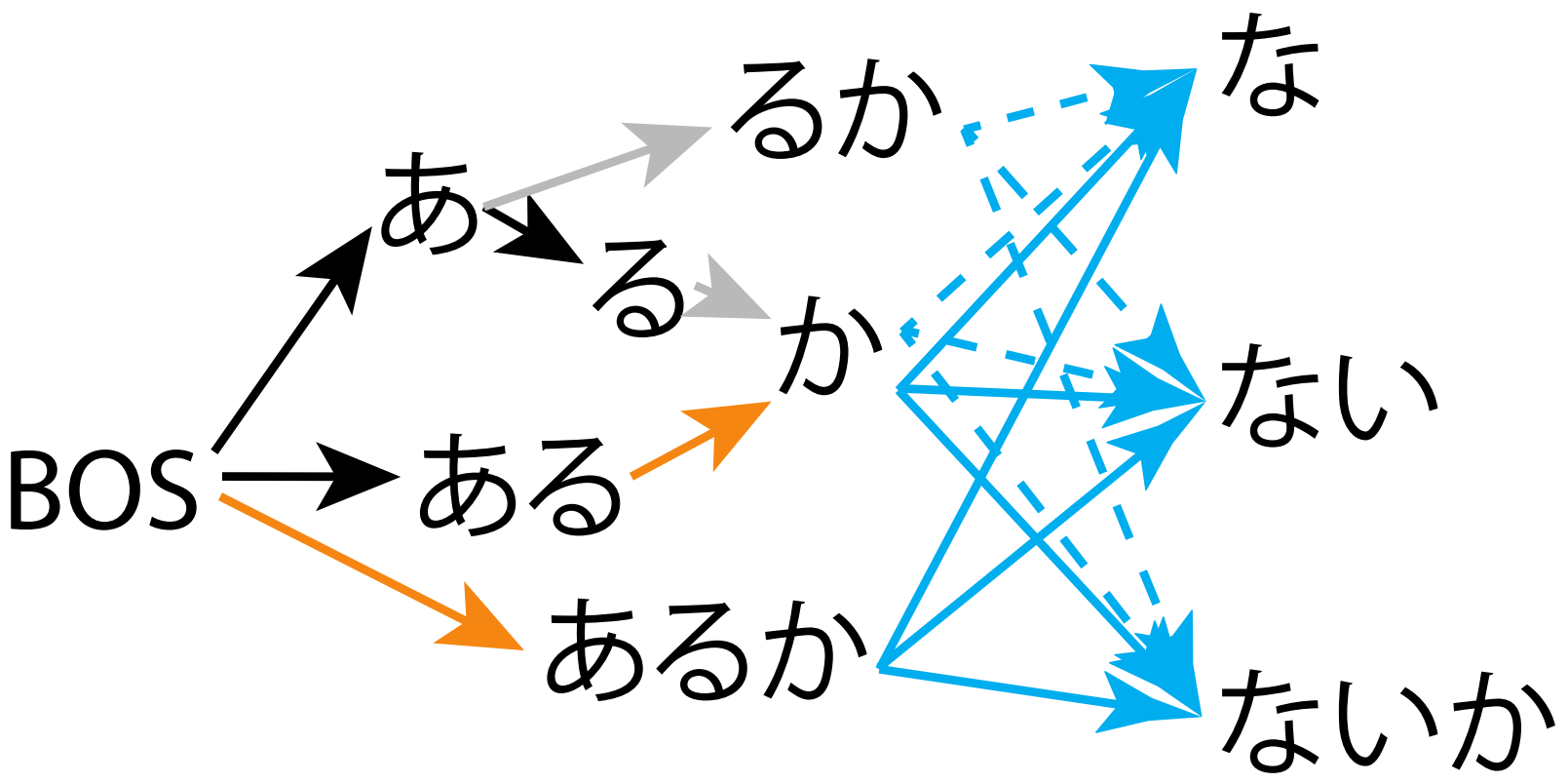
There could be a lot of paths going through the lattice



Most of the candidate paths can't be even remotely correct

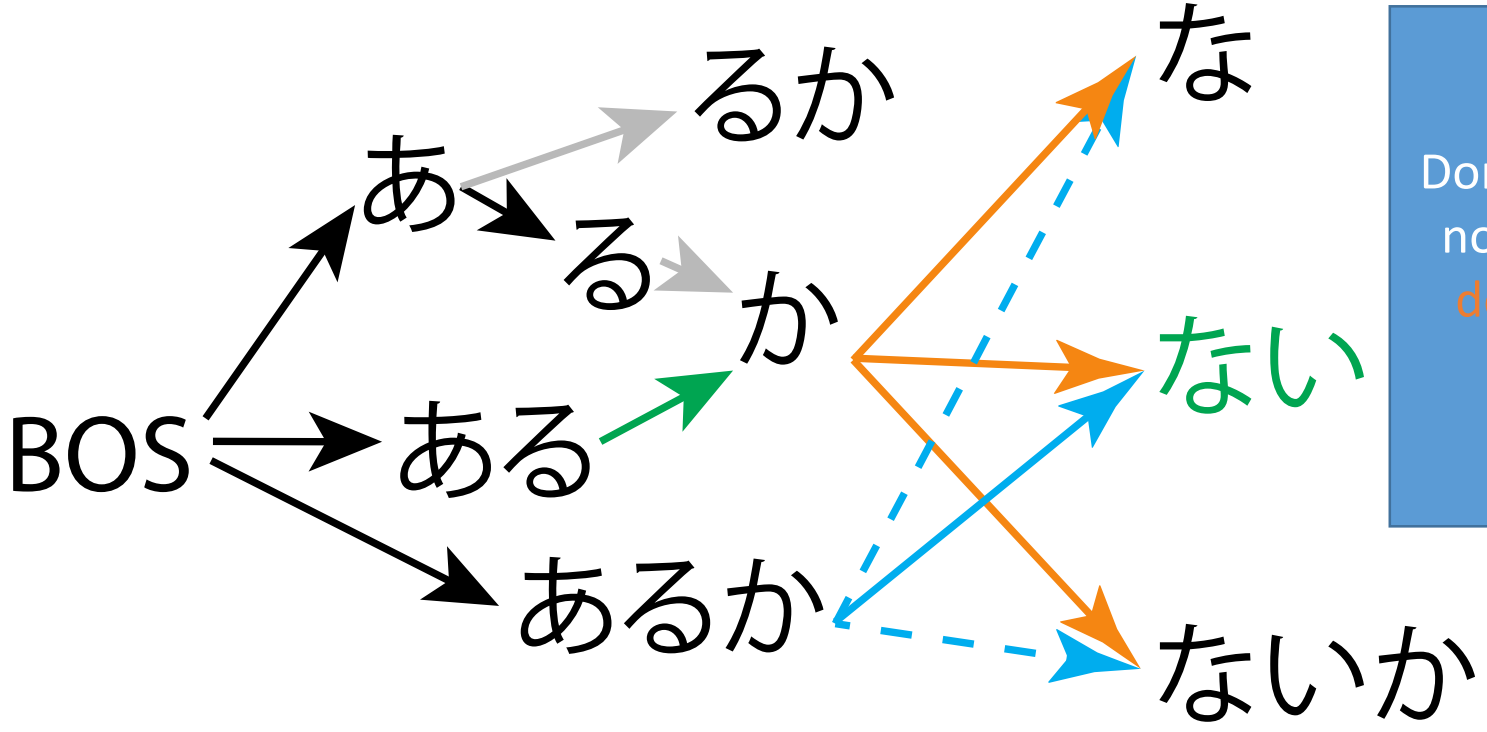
Global Beam: Left

Instead of considering all paths ending on a boundary, consider only ***top k*** paths ending on the current boundary



Global Beam: Right

Use top *m* left paths to compute scores of right nodes. Compute the connections to remaining (*k*-*m*) left paths only for top / right nodes. (m=1, l=1 on the figure)



Idea:
Don't consider nodes which don't make sense in context

Effects of global beams

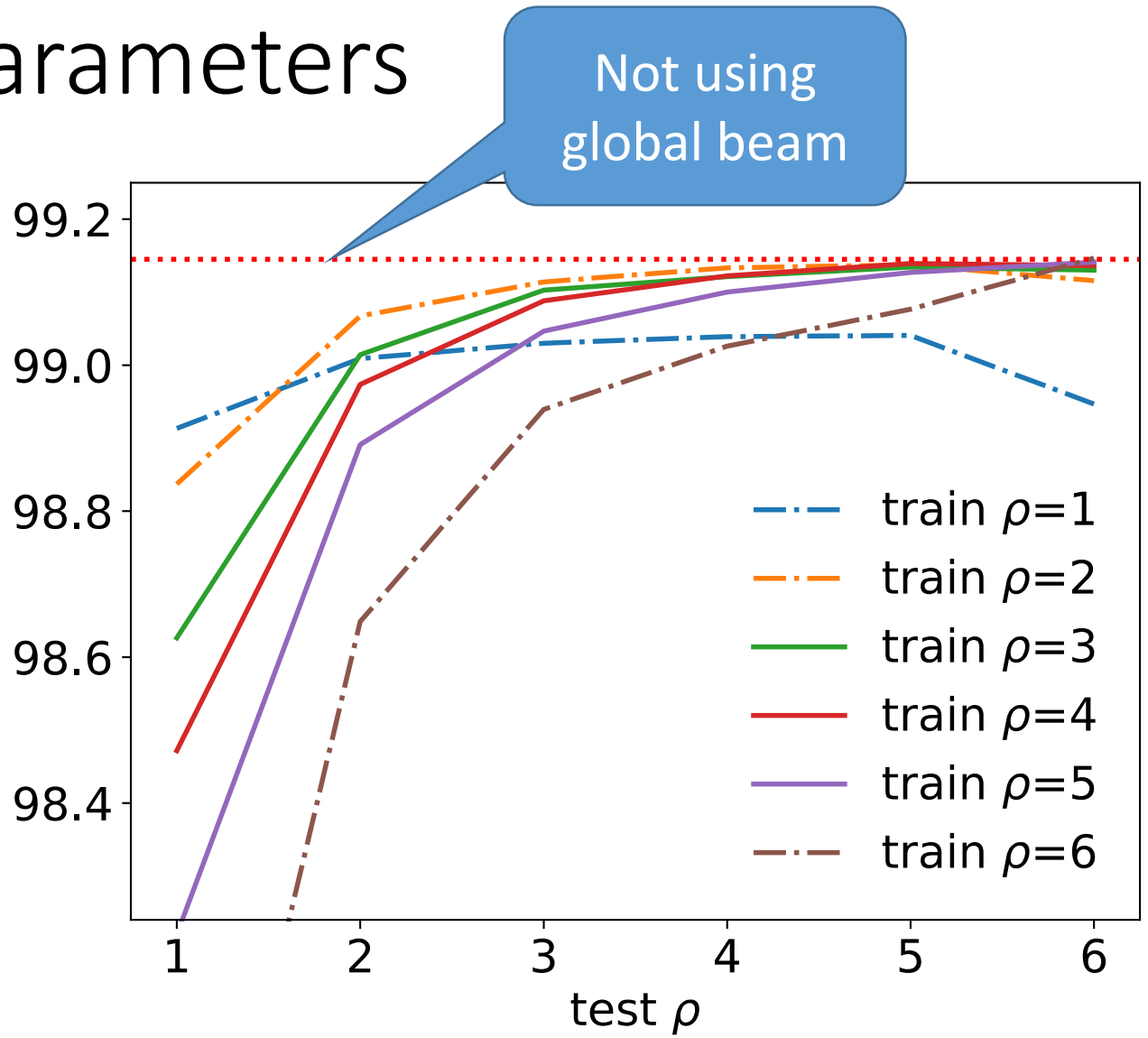
- Almost every sentence has situations when there are 20-30 left/right nodes
- Juman++v2 default settings
  - Local beam size = 5
  - Left beam size = 6
  - Right beam size = 1/5
- Considering much less candidates
  - In total, ~4x speedup
- Accuracy considerations
  - Use surface characters after the current token as features
  - During training use smaller global beam sizes

Ranking procedure should (at least) keep sensible paths in the right beam

Seg+POS F1 on different global beam parameters

Train the model on one set of global beam parameters, test on another

ρ here is the number of both left and right beams

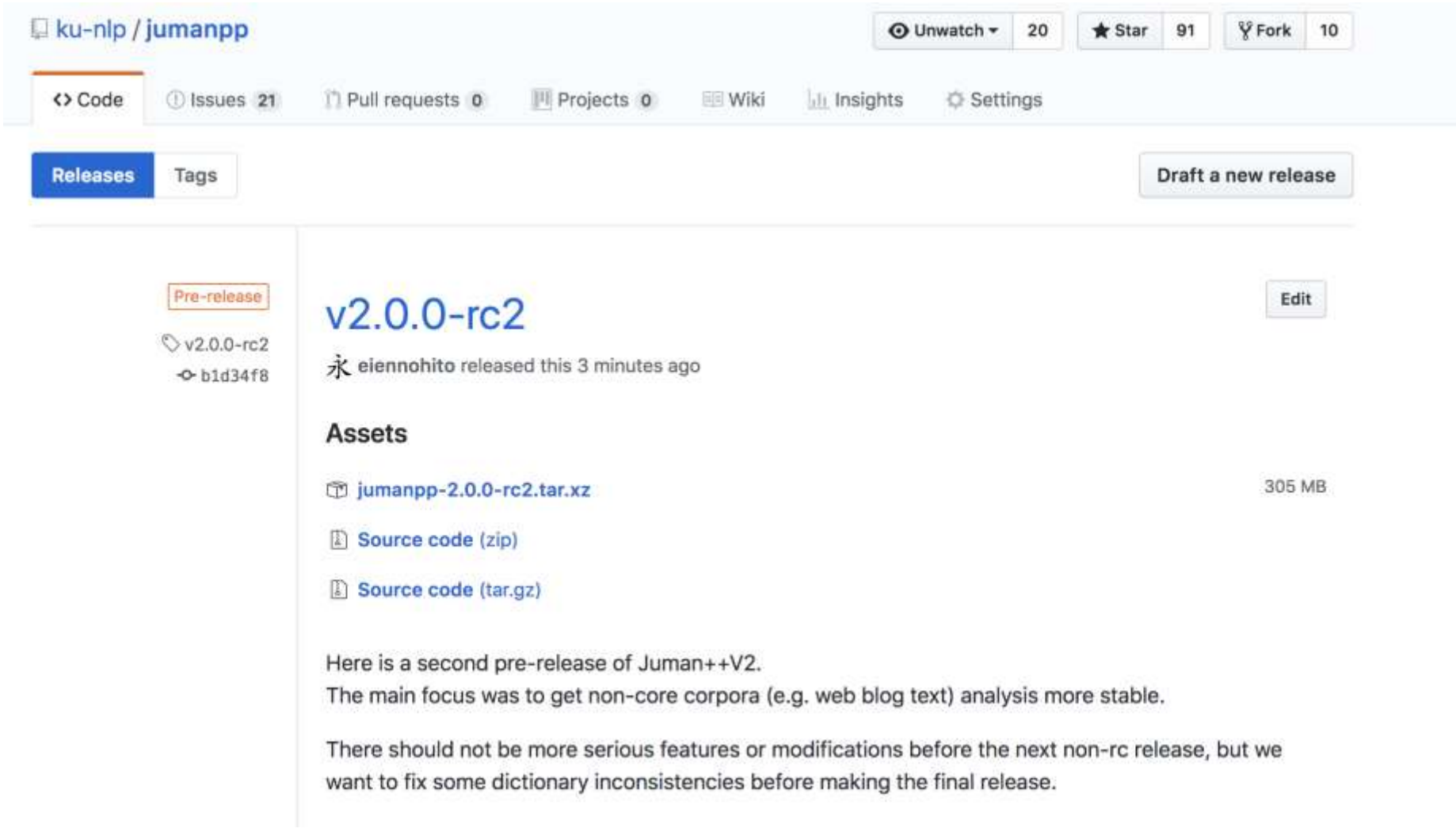


Juman++ V2 RNNLM

- V1 RNNLM implementation was non-batched and non-vectorizable
- V2 RNNLM
  - Uses Eigen for vectorization
  - Batches all computations
  - Deduplicates paths with identical states
- Finally, we evaluate only paths which have remained in EOS beam
  - Basically, RNN is used to reorder paths
  - Opinion: Evaluating the whole lattice is a waste of energy
- Help wanted for LSTM language model implementation (Mikolov's RNNLM now)

Future work

- Improve robustness on informal Japanese (web)
- Create Jumanpp-Unidic
- Use it to bootstrap correct readings in our annotated corpora:
  - Kyoto Corpus
  - Kyoto Web Document Leads Corpus


ku-nlp / jumanpp
Unwatch 20
Star 91
Fork 10
Code
Issues 21
Pull requests 0
Projects 0
Wiki
Insights
Settings
Releases
Tags
Draft a new release
Pre-release
v2.0.0-rc2
b1d34f8
v2.0.0-rc2
Edit
永 eiennohito released this 3 minutes ago
Assets
jumanpp-2.0.0-rc2.tar.xz
305 MB
Source code (zip)
Source code (tar.gz)
Here is a second pre-release of Juman++V2.
The main focus was to get non-core corpora (e.g. web blog text) analysis more stable.
There should not be more serious features or modifications before the next non-rc release, but we want to fix some dictionary inconsistencies before making the final release.

 github.com/ku-nlp/jumanpp

 #jumanpp

Conclusion

- Fast trigram feature/RNNLM-based morphological analyzer
- Usable as library in multithreaded environment
- Not hardwired to Jumandic!
  - Can be used with different dictionaries/standards
- SOTA on Jumandic F1
- Smallest model size (when without RNN)
- Can use surface features (character/character type)
- Can train with partially annotated data