Japanese Language Analysis by GPU Ready Open Source NLP Frameworks

Hiroshi Matsuda

GPU Technology Conference 2020

Contents

発表者略歴

松田寛 （言語処理学会理事）

1998-2000 奈良先端科学技術大学院大学 博士前期課程

- 形態素解析器 茶筌 version 2.0 のリリースを担当

2000-2012 株式会社ジャストシステム

- エンジニア → NLP技術部門ディレクタ → 研究開発部長

2012-2018 スタートアップ数社をサポート

- プログラマ+データサイエンティスト

2018-現在 株式会社リクルート Megagon Labs

- 2019年に日本語NLPライブラリ GiNZA をリリース

趣味: Viola演奏・珈琲焙煎・離島巡り

GitHub: hiroshi-matsuda-rit, Twitter: (at) hmtd223

Megagon Labsのご紹介

Megagon Labs

Mountain View Research Office

Experiential Analytics + Data Management

- よりよい情報により、人々の最善の意思決定を支援していきます。

Vision

- 人々のニーズや要望が、最も有益なサービスや情報に瞬時にアクセスしてかなえられる。先進的コンピューティング研究とテクノロジーを駆使し、この実現をめざします。

About us

- Megagon Labsはリクルートグループのイノベーション・センターです。リクルートグループは、多くのサービス分野において情報を広く公開し、人々が人生ステージで最良の機会を発見することを支援してきました。
- リクルートグローバルネットワークは、IndeedやGlassdoorを含む世界中に広がっています。Megagon Labsは世界トップクラスの研究者、有名大学やリクルートグループ内企業との連携を含むグローバルなイノベーションネットワークを築いています。

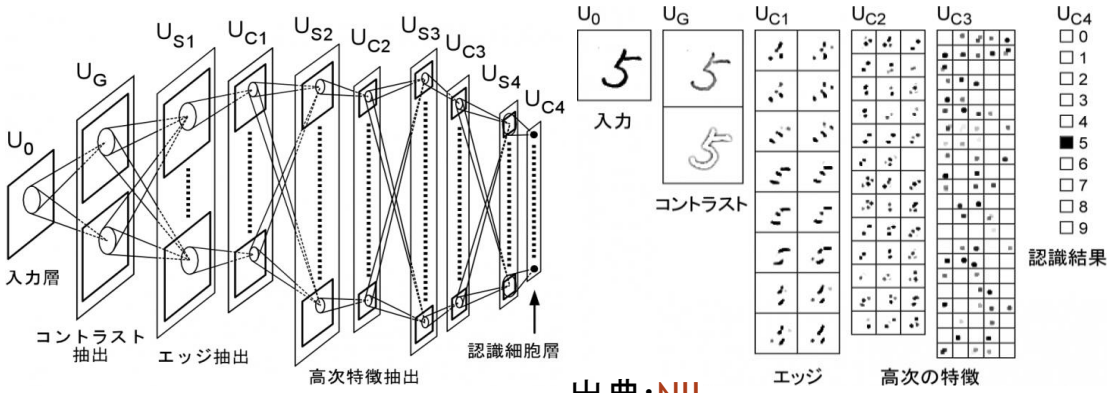
自然言語処理技術の進歩とGPUが与えた影響

ルールベースからTransformersまで

日本語NLP技術の萌芽
1978年

東芝が日本初のワードプロセッサを発表

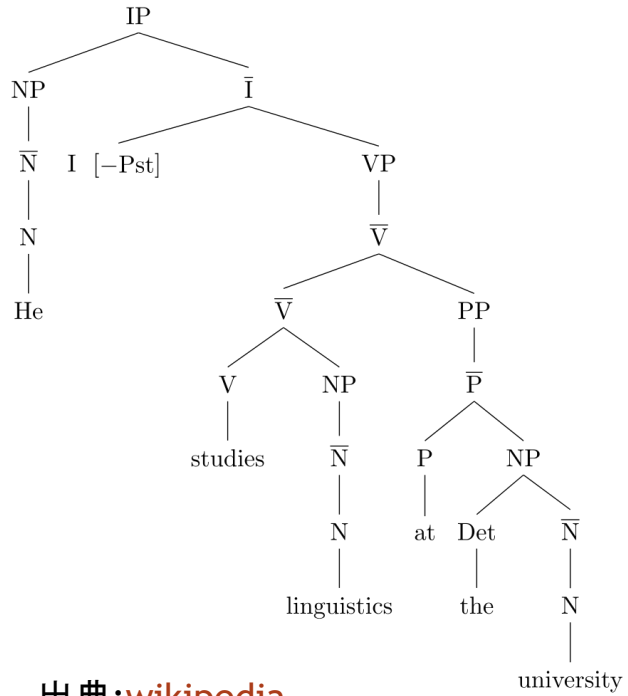
- 日本語入力で初めて形態素解析を導入
- 同研究所ではCNNの原型が研究されていた
- 日本語文字コード JIS C 6226 制定



出典:NII

世界で言語学・認知科学研究が進む

- 生成文法理論の発展と計算機上の実証実験
- アンチテーゼとしての認知言語学の勃興
- メタファー・カテゴリ化などの認知能力の研究



出典:wikipedia

経験則的NLP技術の発展
1985年

国内で16bit PCとワープロソフトが普及

- JIS第二水準漢字ROM搭載が標準化
- 日本語入力技術の開発競争激化
- 初のPC向け日英自動翻訳システム発売

ルールベースNLPの飛躍的な発展

- 哲学・言語学・計算機科学の融合が進む
- 日本は自然言語処理先進国でもあった

言語学は文脈を考慮した記号処理に移行

- 語の静的意味と文脈依存（言語使用上）の意味の分離 → 現代の分散表現の進歩に符合

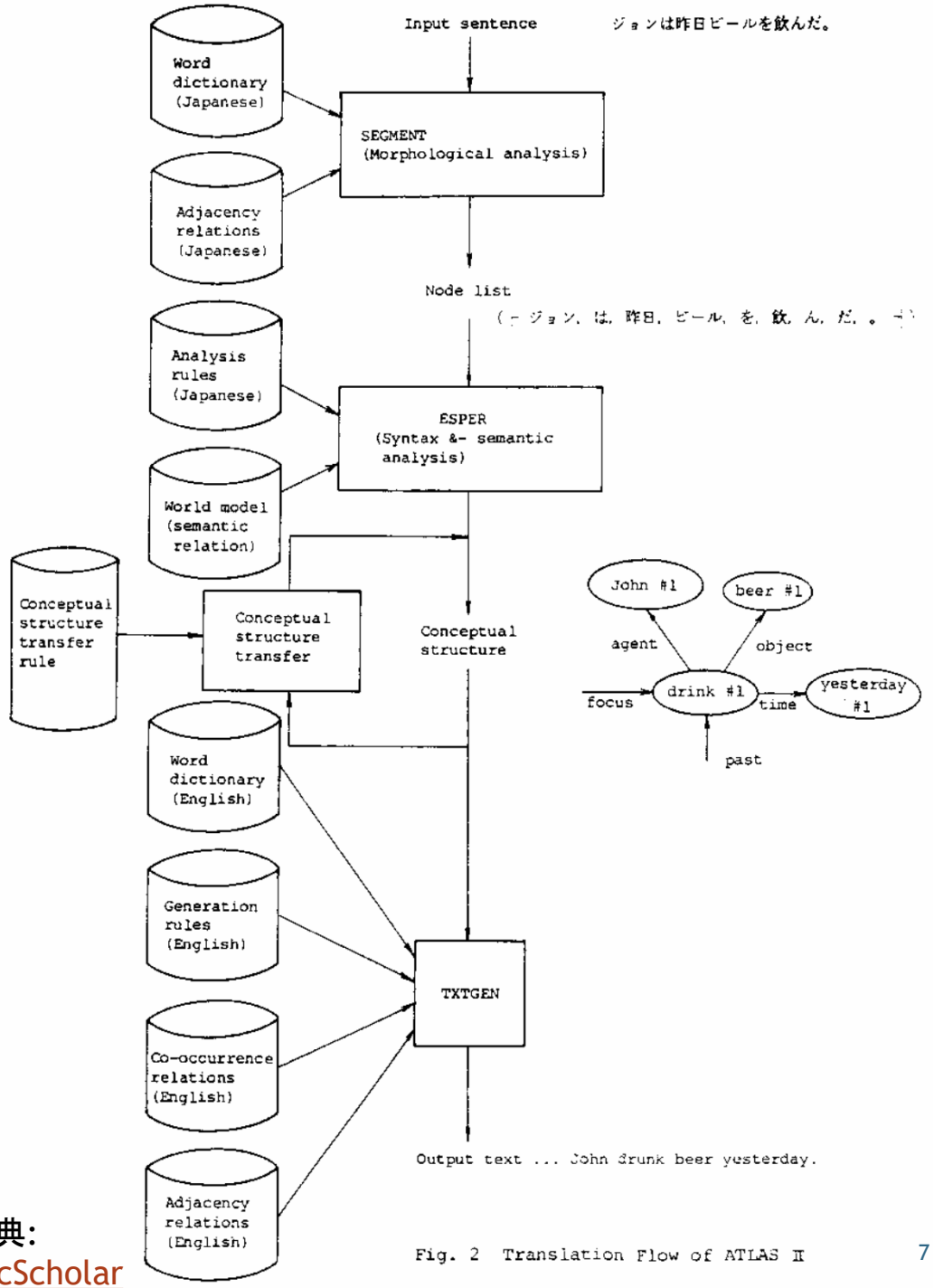


Fig. 2 Translation Flow of ATLAS II

出典:
SemanticScholar

統計的NLP技術の発展
1993年

32bit PC & Windowsの普及

- WebブラウザMosaicの登場
- IBMが統計的機械翻訳技術を論文発表
- 音声認識ソフトウェア発売(1994)

NLPに立ち塞がる二つの壁の顕在化

- 文法カバー率の限界(知識記述コストの壁)
- データスパースネス問題(共起空間の壁)

話し言葉への注目

- 話し言葉と書き言葉を統合する文法は作れるか？

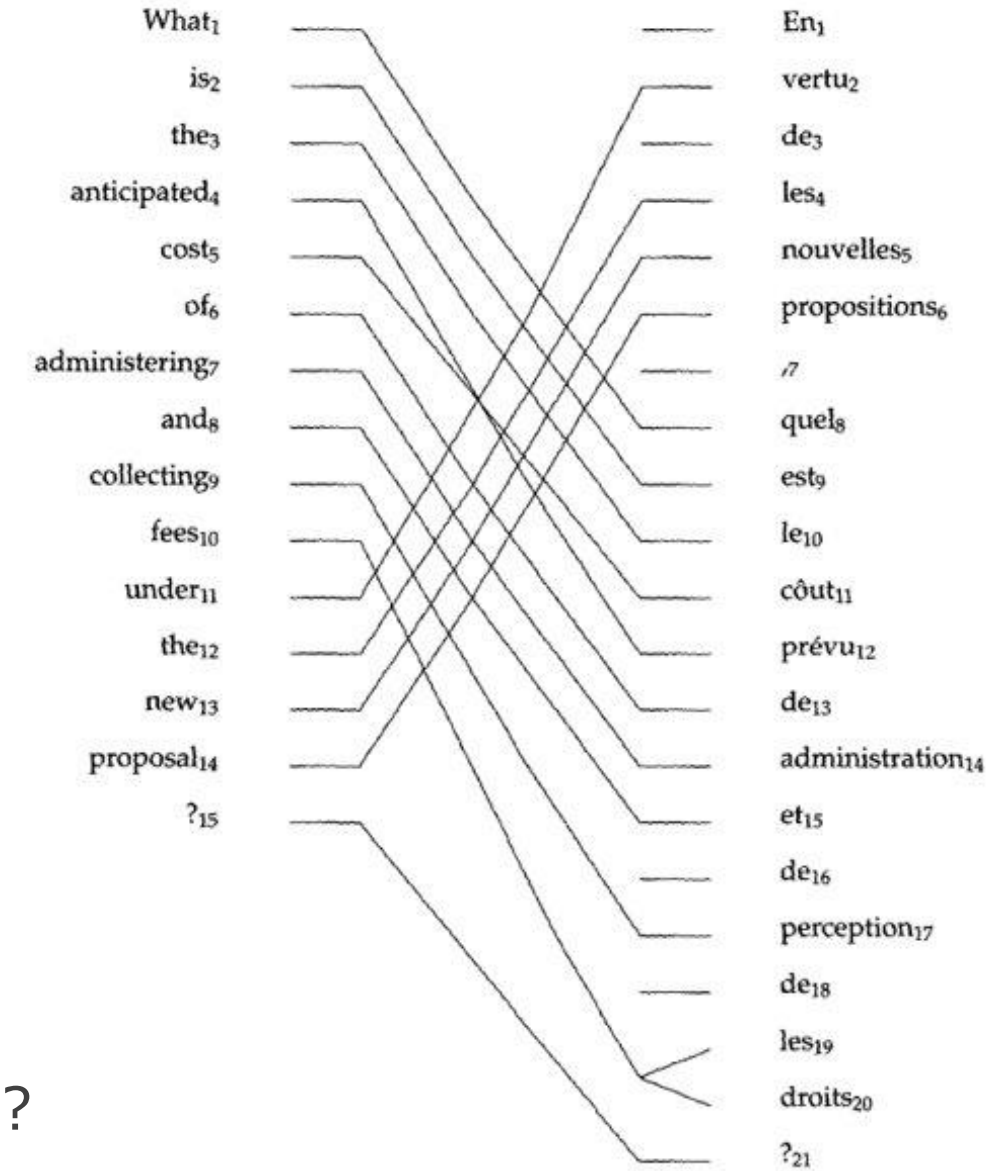


Figure 16
The best of 1.9×10^{25} alignments.

出典:ResearchGate

機械学習技術のNLP応用
1999年

Internet普及・iモード提供開始

- Unicode 3.0公開・Webの国際化が一気に進む
- 掲示板・メールで独自の文体が生まれる
- SVMやCRF(2003)など機械学習手法がルールベースシステムの精度を超える

表 1　使用した静的素性
Table 1　List of static features.

前/後 文節	主辞見出し，主辞品詞，主辞品詞細分類，主辞活用，主辞活用形，語形見出し，語形品詞，語形品詞細分類，語形活用，語形活用形，括弧の有無，句読点の有無，文節の位置（文頭，文末）
文節間	距離（1,2-5,6 以上），すべての助詞（は，が，を，に …），括弧，句読点の有無

出典:chasen.org

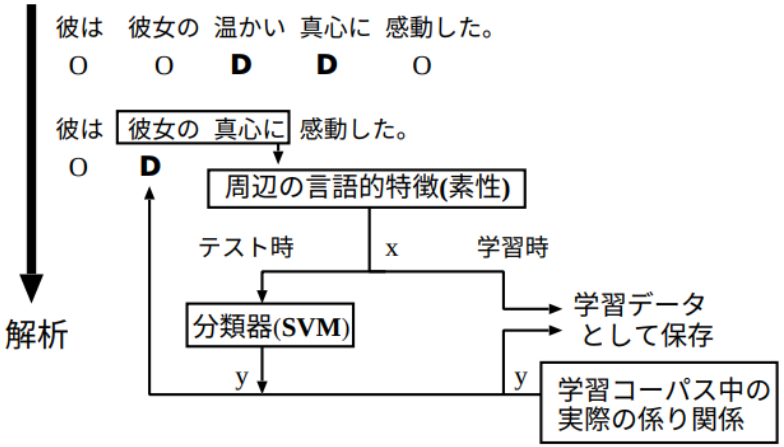


図 2　学習とテストの処理
Fig. 2　Process of training and testing.

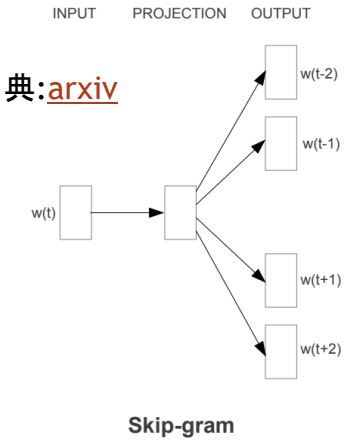
Table 2: Feature templates: $f_k(\langle w', t' \rangle, \langle w, t \rangle)$
$t' = \langle p1', p2', cf', ct, bw' \rangle$, $t = \langle p1, p2, cf, ct, bw \rangle$, where $p1'/p1$ and $p2'/p2$ are the top and sub categories of POS. cf'/cf and ct'/ct are the cfrom and ctype respectively. bw'/bw are the base form of the words w'/w.

type	template
Unigram basic features	$\langle p1 \rangle$ $\langle p1, p2 \rangle$
w is known	$\langle bw \rangle$ $\langle bw, p1 \rangle$ $\langle bw, p1, p2 \rangle$
w is unknown	length of the word w up to 2 suffixes $\times$ $\{\phi, \langle p1 \rangle, \langle p1, p2 \rangle\}$ up to 2 prefixes $\times$ $\{\phi, \langle p1 \rangle, \langle p1, p2 \rangle\}$ character type $\times$ $\{\phi, \langle p1 \rangle, \langle p1, p2 \rangle\}$
Bigram basic features	$\langle p1', p1 \rangle$ $\langle p1', p1, p2 \rangle$ $\langle p1', p2', p1 \rangle$ $\langle p1', p2', p1, p2 \rangle$ $\langle p1', p2', cf', p1, p2 \rangle$ $\langle p1', p2', ct', p1, p2 \rangle$ $\langle p1', p2', cf', ct', p1, p2 \rangle$ $\langle p1', p2', p1, p2, cf \rangle$ $\langle p1', p2', p1, p2, ct \rangle$ $\langle p1', p2', p1, p2, cf, ct \rangle$ $\langle p1', p2', cf', p1, p2, cf \rangle$ $\langle p1', p2', ct, p1, p2, ct \rangle$ $\langle p1', p2', cf', p1, p2, ct \rangle$ $\langle p1', p2', ct', p1, p2, cf \rangle$ $\langle p1', p2', cf', ct', p1, p2, cf, ct \rangle$
w' is lexicalized	$\langle p1', p2', cf', ct', bw', p1, p2 \rangle$ $\langle p1', p2', cf', ct', bw', p1, p2, cf \rangle$ $\langle p1', p2', cf', ct', bw', p1, p2, ct \rangle$ $\langle p1', p2', cf', ct', bw', p1, p2, cf, ct \rangle$
w is lexicalized	$\langle p1', p2', p1, p2, cf, ct, bw \rangle$ $\langle p1', p2', cf', p1, p2, cf, ct, bw \rangle$ $\langle p1', p2', ct', p1, p2, cf, ct, bw \rangle$ $\langle p1', p2', cf', ct', p1, p2, cf, ct, bw \rangle$
w'/w are lexicalized	$\langle p1', p2', cf', ct', bw', p1, p2, cf, ct, bw \rangle$

出典:ACL anthology

深層学習技術のNLP応用
2010年

出典:arxiv



Skip-gram

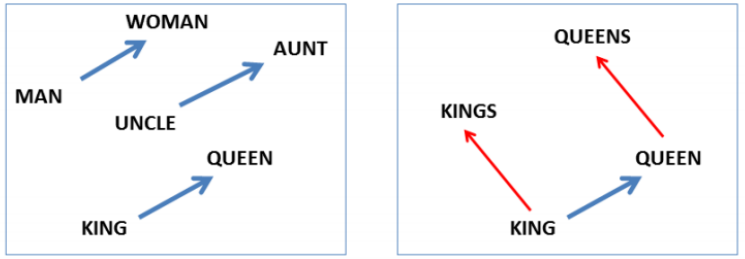


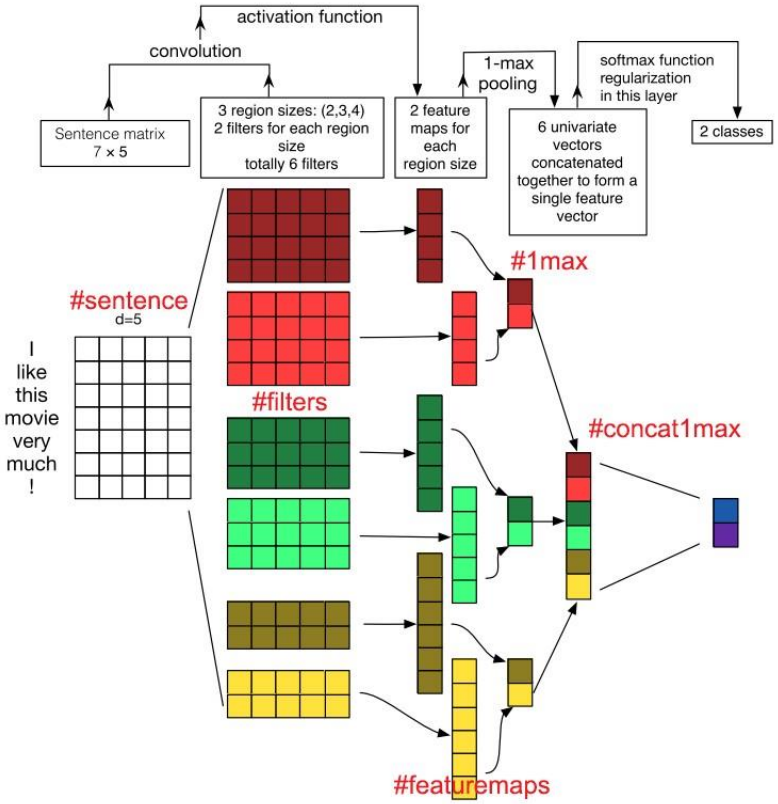
Figure 2: Left panel shows vector offsets for three word pairs illustrating the gender relation. Right panel shows a different projection, and the singular/plural relation for two words. In high-dimensional space, multiple relations can be embedded for a single word. 出典:ACL anthology

スマートフォンの急速な普及

- 行動履歴の蓄積・マイニング・リコメンデーション

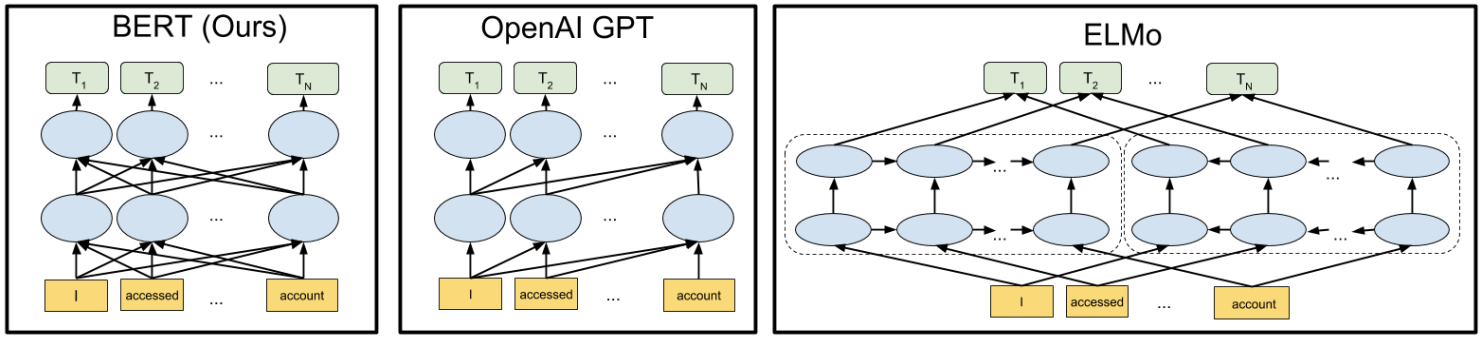
GPUを使ったNLP研究の萌芽

- 「GPGPUによる超並列処理の文書分類への適用」
  - 言語処理学会年次大会発表論文(2011)
    豊田工大 佐々木裕氏 → CUDAベースのSVM分類器
- 画像・音声分野でCNN/RNN系モデルの発展
  - NLPタスクへの応用が始まる
  - Googleが発表したword2vec(2013)により
    流れが一気に加速
  - 単語の意味をベクトルで表現するEmbeddingの
    時代に突入していく



出典:arxiv

TransformerのNLP適用 2018年



LSTM → 双方向Transformers

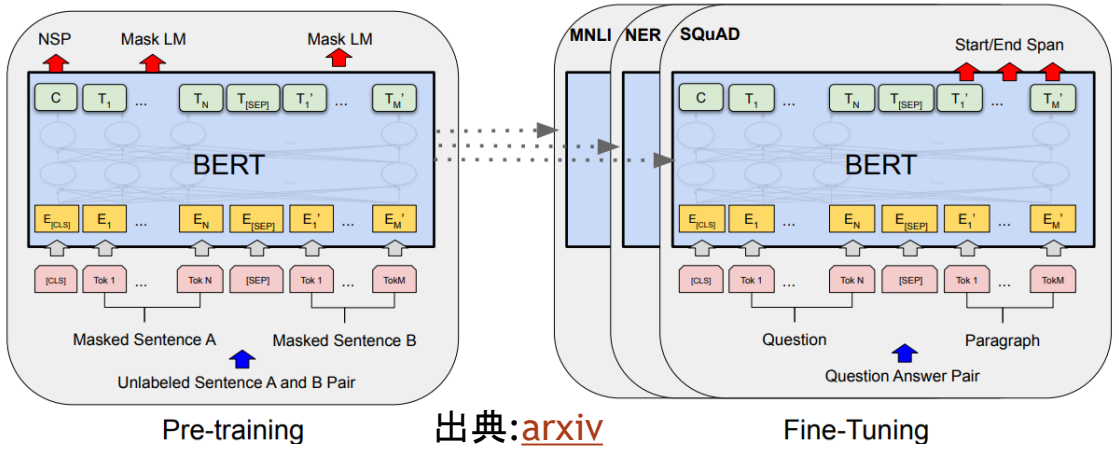
- Pre-training & Fine-Tuning型の学習
- BERTologyと呼ばれる一連の研究に発展

文脈を考慮した単語分散表現の一般化

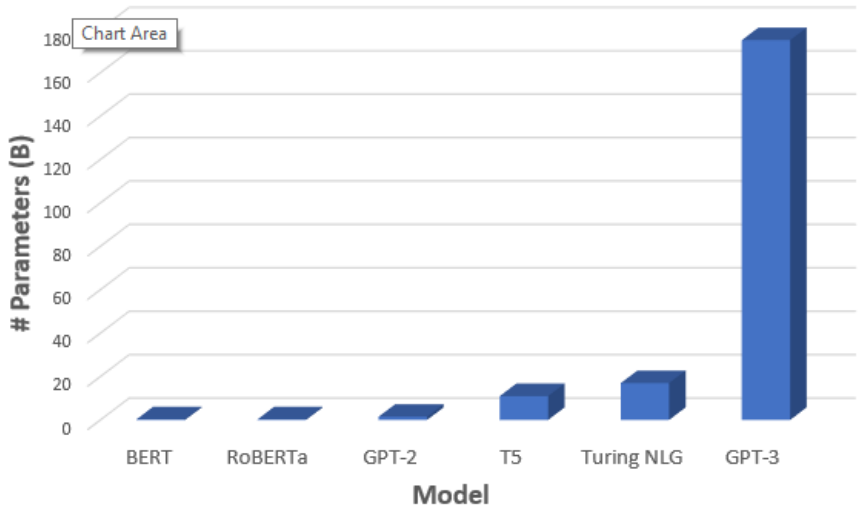
- 文分散表現の表現力も格段に向上
  - SIF派(全トークン重み加重平均－第1PC)
  - CLS派(Fine-Tuning後のCLS出力層の重み)

学習モデルの超大規模化と文生成への応用

- BERT-Large: 340M parameters
- GPT-3: 175G parameters (V100 x 1万台)



出典:arxiv



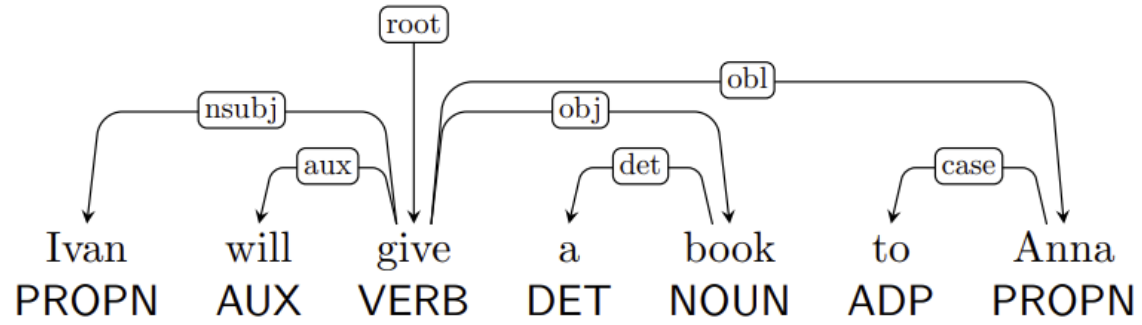
出典:arxiv

NLP Frameworkを用いた日本語の解析

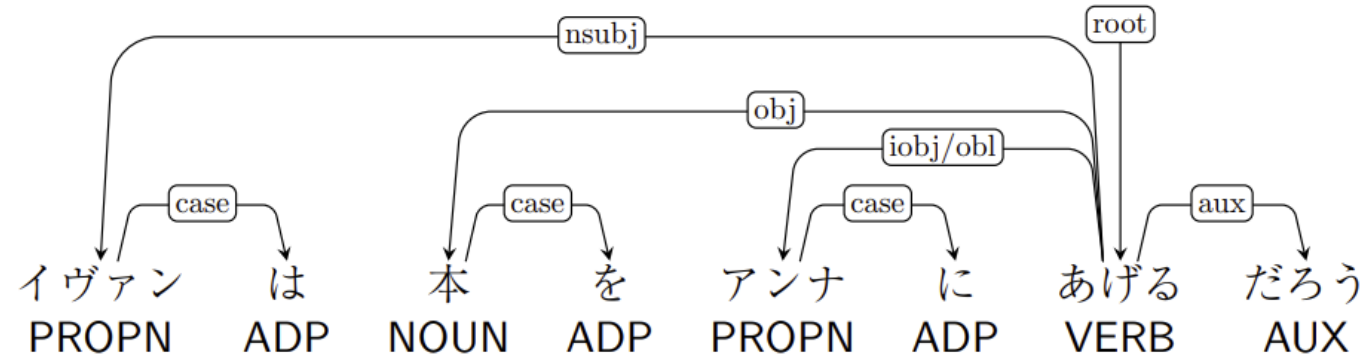
Universal DependenciesとGiNZAの文節API

Universal Dependenciesとは？

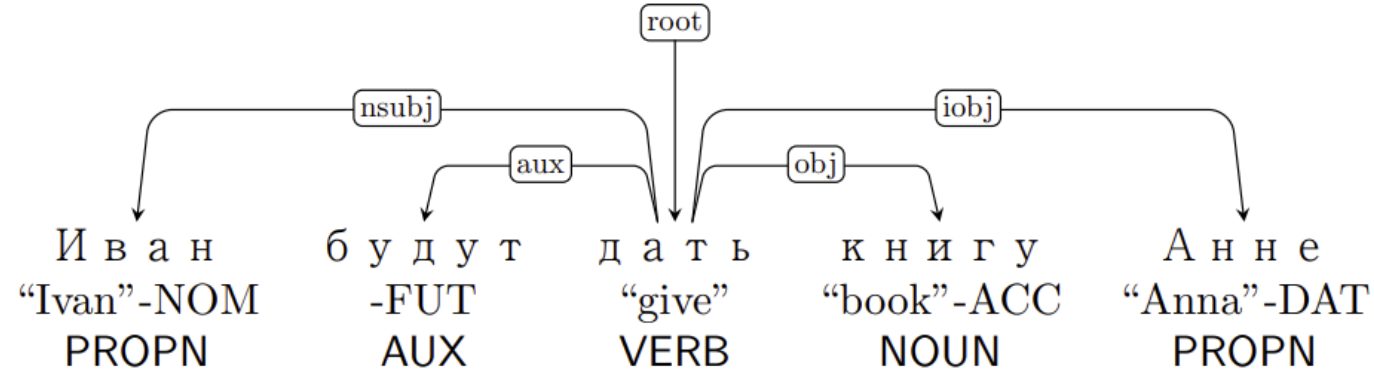
(4)



(5)



(6)



出典:jstage

Universal Dependenciesとは？

目的

Universal Dependencies (UD)は、構文解析の後段の処理の共通化や、他の言語のコーパスを用いた言語横断的な学習、言語間の定量的な比較などを可能にするための土台を目指して、多言語で一貫した構文構造とタグセットを定義するという活動である。[金山2015]

歴史

- 2014年頃にUDオープンコミュニティが日本を含め全世界で立ち上がる
- UDの多言語データセットを学習対象とするNLP Frameworkのリリースが続く

内容

- 世界共通の文法体系で様々な言語のテキストに依存関係情報を付与
- 日本語データセットはPUD・GSD・BCCWJ・KTC等が公開済み
- PUDの1000文は各言語に翻訳される（パラレルコーパス）

主要な日本語UDデータセット

UD_Japanese-GSD - 約8,000文

- v2.5から商用利用可能なライセンス(CC BY-SA 4.0)に変更
- v2.6に固有表現ラベルを追加した v2.6-NE をMegagon Labsが公開

UD_Japanese-BCCWJ – 約57,000文 (新聞由来の約16,000文を含む)

- 国立国語研究所が開発した現代日本語書き言葉均衡コーパス(BCCWJ)がベース
- ライセンスは有償(GiNZAは国語研の許諾を受けて学習済みモデルを配布)

UD_Japanese-PUD – 1,000文

- 全言語で共通の内容を持つパラレルコーパス
- 原文は750文が英語で残りはドイツ語・フランス語・イタリア語・スペイン語
- プロの翻訳家が各国語に翻訳

CoNLL-U Data Format (miscに言語固有情報を追加)

```
# text = 銀座でランチをご一緒しましょう。
1   銀座      銀座    PROPN   名詞-固有名詞-地名-一般   _   6   obl     _   BunsetuBILabel=B|NE=B-City
2   で        で      ADP     助詞-格助詞               _   1   case    _   BunsetuBILabel=I
3   ランチ    ランチ  NOUN    名詞-普通名詞-一般        _   6   obj     _   BunsetuBILabel=B
4   を        を      ADP     助詞-格助詞               _   3   case    _   BunsetuBILabel=I
5   ご        ご      NOUN    接頭辞                    _   6   compound _  BunsetuBILabel=B
6   一緒      一緒    VERB    名詞-普通名詞-サ変可能    _   0   root    _   BunsetuBILabel=I
7   し        する    AUX     動詞-非自立可能           _   6   advcl   _   BunsetuBILabel=I
8   ましょう  ます    AUX     助動詞                    _   6   aux     _   BunsetuBILabel=I
9   。        。      PUNCT   補助記号-句点             _   6   punct   _   BunsetuBILabel=I
```

Token Fields (tab separated values)

1. ID: Word index, integer starting at 1
2. FORM: Word form or punctuation symbol
3. LEMMA: Lemma or stem of word form
4. UPOS: Universal part-of-speech tag
5. XPOS: Language-specific part-of-speech tag
6. FEATS: List of morphological features
7. HEAD: Head of the current word
8. DEPREL: Universal dependency relation
9. DEPS: Enhanced dependency graph
10. MISC: Any other annotation

日本語NLPライブラリ GiNZA

Universal Dependenciesをベースとする初の日本語NLPライブラリ

- Megagon Labsと国立国語研究所の共同研究成果として2019年4月にリリース
  - 2020年1月から8ヶ月間のダウンロード数は3万回以上

分析の現場ですぐに使える実用性重視の設計

- 商用利用可能なMITライセンスで配布
  - Python 3.6以上を導入済みの環境であればpipコマンド一行で導入が完了
- 産業利用を想定し高速・軽量化されたspaCyをフレームワークに採用
  - 日本語以外の言語にもリソース変更のみで対応可能
- UD_Japanese-BCCWJで学習した高精度な解析モデルを搭載
  - トークナイザとしてSudachiPyを採用

出典 https://ja.wikipedia.org/wiki/海_(ドビュッシー)

GiNZAの名称について

日本文化を世界に発信し続けてきた街「銀座」

- 文明開化以降、時代の最先端で常に世界と影響を与え合ってきた商業・文化・芸術の街
- Megagon Labs Japanの拠点

全世界のビジネスで手軽に使える日本語NLPの提供

- 商用利用可能+フリーな日本語依存構造解析器の公開
- 日本語を使用する世界の産業の活性化に貢献

GiNZA - 使い方

インストール (python 3.6以上)

```
$ pip install -U ginza
```

CLIツールによる解析 - CoNLL-U出力

```
$ ginza sentence_per_line.txt
```

コーディング例

```
$ python
import spacy
nlp = spacy.load("ja_ginza")
doc = nlp("赤い車を持っている")
for tkn in doc:
    print(tkn.i, tkn.orth_, tkn.lemma_, tkn.pos_, tkn.tag_, tkn.dep_, tkn.head.i)
```

```
0 赤い 赤い ADJ 形容詞-一般 acl 1
1 車 車 NOUN 名詞-普通名詞-一般 obj 3
2 を を ADP 助詞-格助詞 case 1
3 持っ 持つ VERB 動詞-一般 ROOT 3
4 て て SCONJ 助詞-接続助詞 mark 3
5 いる いる AUX 動詞-非自立可能 aux 3
```

GiNZA - GPUの有効化

GPUライブラリthincを利用

- ginzaの後にthincを上書きインストールする (CUDA 10.2の例)

```
$ pip install -U ginza
$ pip install -U thinc[cuda102]
```

CLI

```
$ ginza -g sentence_per_line.txt
```

コーディング例

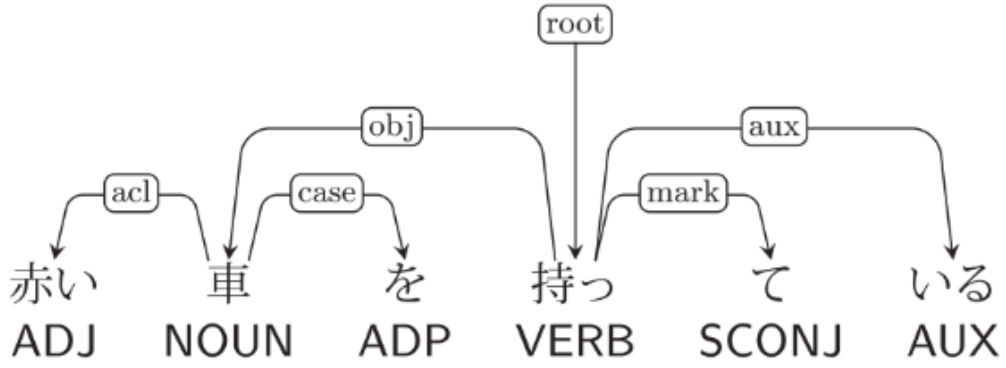
```
$ python
import spacy
spacy.require_gpu()
nlp = spacy.load("ja_ginza")
```

GiNZA - 文節API

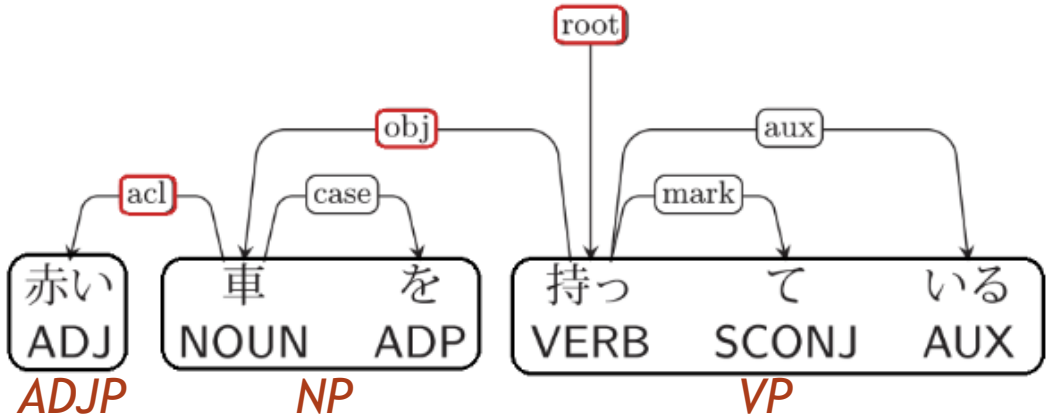
- Universal Dependenciesの枠組みの中に日本語に特徴的な文節構造を組み込む

```
import ginza
for bs in ginza.bunsetu_spans(doc):
    print(bs.root.i, bs.text, bs.label_, bs.root.dep_, bs.root.head.i)
```

```
0 赤い ADJP acl 1
1 車を NP obj 3
3 持っている VP ROOT 3
```



Universal Dependencies 体系は単語(トークン)を単位とする依存関係(係り受け)を定義している。



GiNZAでは日本語の文節の主辞となる単語への依存関係を区別して学習することで文節を単位とする依存構造を捉えることができる。

GPU Ready OSS NLP Frameworks

spaCy v2 - v3 and Stanza

spaCy

spaCy

ExplosionAI GmbHが開発するPythonベースの多言語NLPフレームワーク

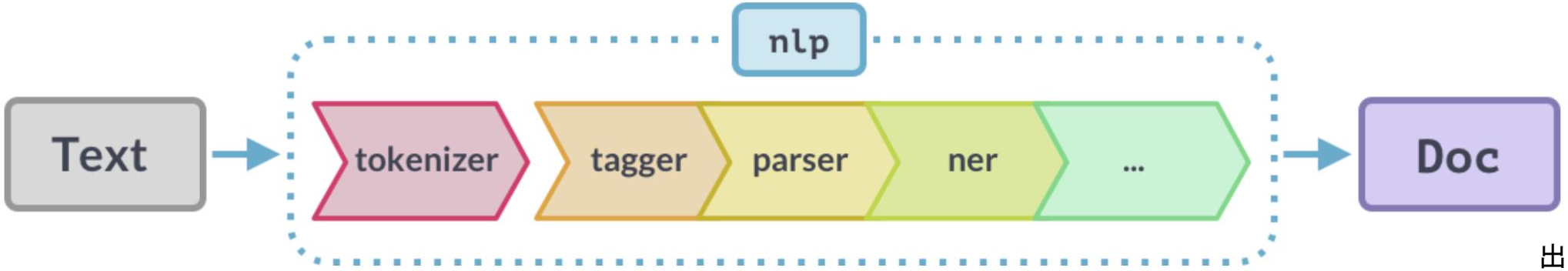
- 産業利用を前提とする設計・ライセンス
  - Python 2.7, 3.5+に対応 (only 64 bit)
  - Cythonを用いた高速軽量化・高カバレッジのテストコードによる品質維持
  - 充実したドキュメントとオンラインコース・商用利用可能なMITライセンス
  - 可視化ツールdisplaCyやアノテーションツールProdigyなど周辺機能も充実
- 公式対応言語
  - 英,西,独,仏,伊,蘭,葡,ギリシャ,ポーランド,ノルウェー,デンマーク,ルーマニア,リトアニア,日本語,中国語（日本語は2020年6月リリース）
  - トークナイズ・UD品詞付与・依存構造解析・固有表現抽出の全機能を提供
  - 学習データはUniversal Dependenciesが中心
  - 他45言語はトークナイズのみに対応

spaCy v2 - パイプラインモデル

spaCy

CNNマルチタスク学習器を用いたパイプライン設計

- 学習モデルについてはこちらの解説記事に詳しい



出典: spacy.io

日本語モデルはtokenizerとしてWork ApplicationsのSudachiPyを使用

- 単語ベクトルは同社のchiVe mc90を使用

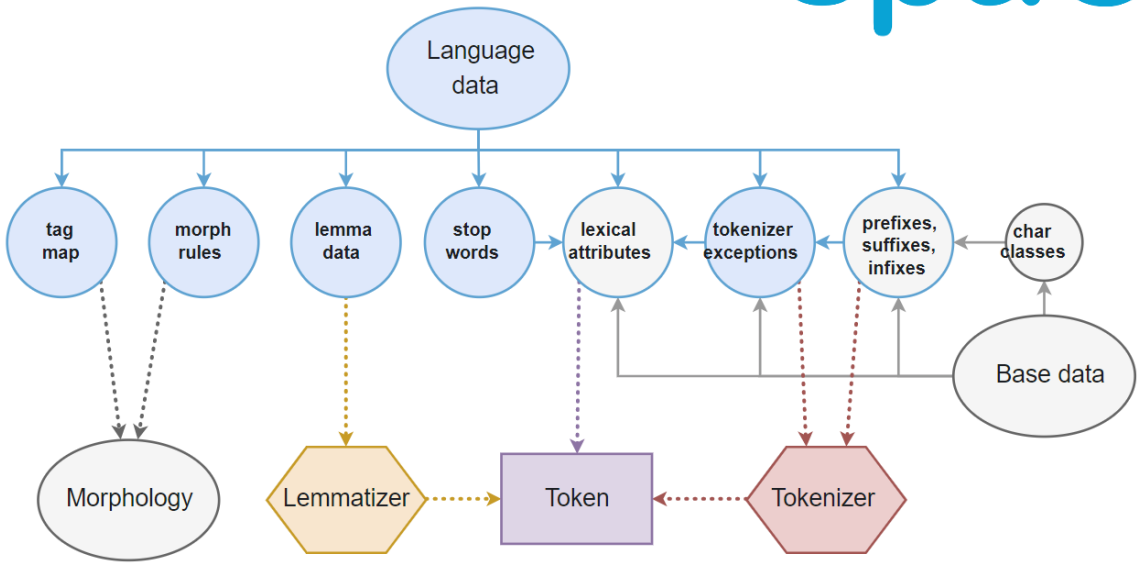
学習データはUD_Japanese-GSD v2.6-NEを使用

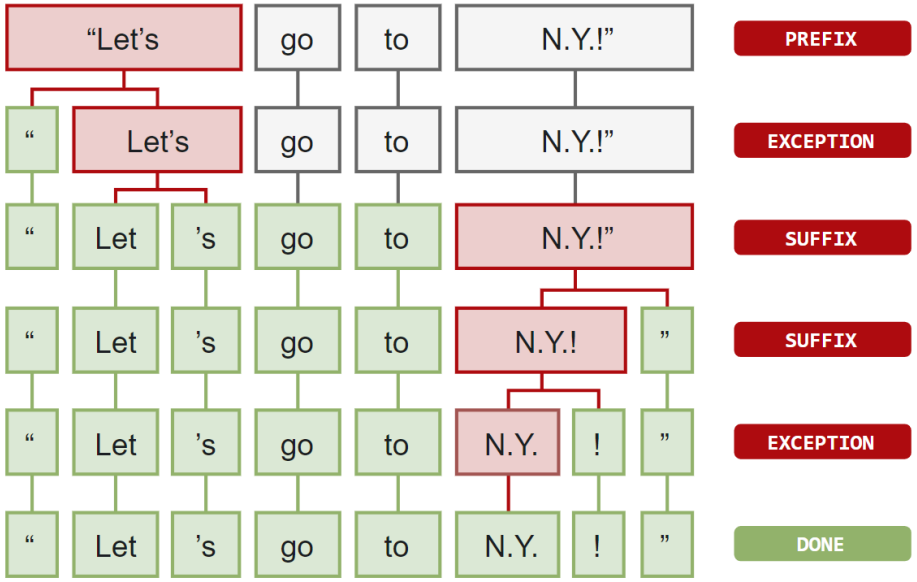
- parser+nerのマルチタスク学習

英語モデルのtokenize

spaCy

- 空白区切りを起点にルールを適用
- トークン分割ルール
  - Prefix, Exception, Suffix …
- 表記正規化ルール
- 単語レベル正規化ルール
  - 's = us (let usの場合)
  - N.Y. = New York
- 機械学習ベースでの品詞付与
  - 英語モデルではtagger/parser/nerがCNNでマルチタスク学習される





出典: spacy.io

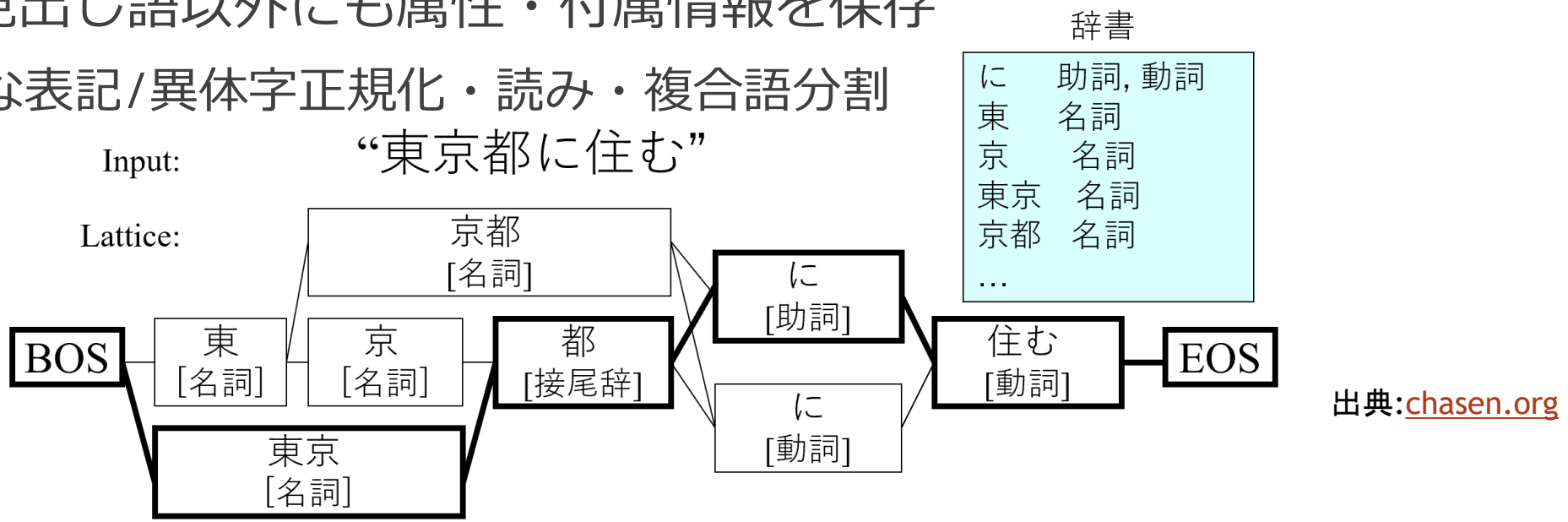
形態素解析器による日本語のtokenize/tagging

形態素解析：単語分割(tokenize)と品詞付与を同時実行

- 一般に品詞付与=taggingだが日本語の品詞体系をUD品詞にマップする必要がある
  - 「一緒」→ NOUN=名詞, VERB=動詞(ご一緒する), ADJ=形容詞(一緒だ) を用法別に解釈

形態素辞書：見出し語以外にも属性・付属情報を保存

- 活用形/かな表記/異体字正規化・読み・複合語分割



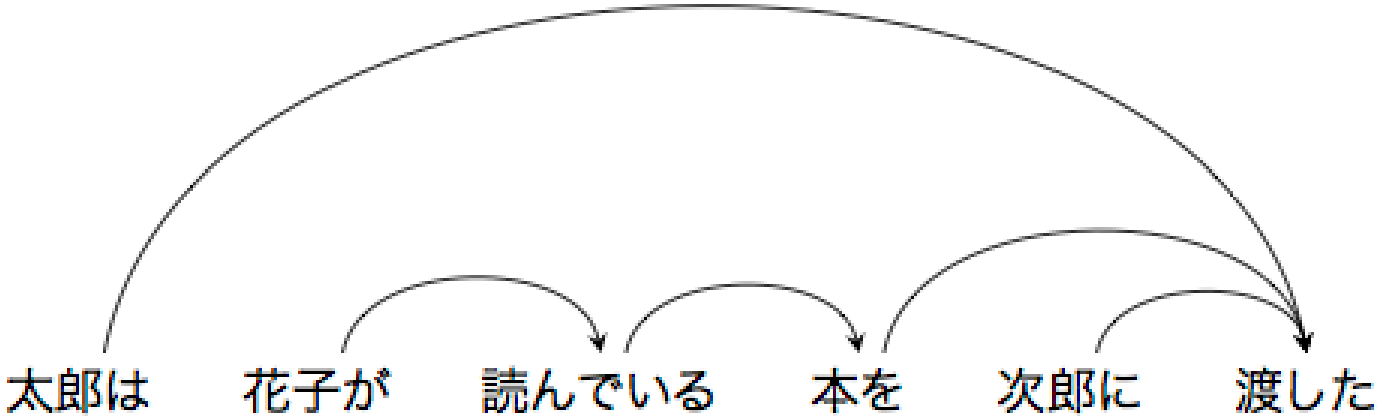
出典:chasen.org

形態素解析=形態素ラティスから正しい系列を１つ選ぶタスク

係り受け解析と依存関係ラベリングの違い

係り受け解析は単語または文節の係り先の判定まで

- 日本語の文節係り受けは一般に非交差かつ後方への単方向参照



出典: github.com/tetsuok

依存関係ラベリングはnsubj(主語)やobj(目的語)など依存関係分類まで行う

- UDのSyntactic Relationsは全37種類
- spaCyは非交差制約を前提に学習・解析を行う

spaCy v2 - 使い方

インストール (python 3.6以上をおすすめ)

```
$ pip install -U spacy
$ python -m spacy download ja_core_news_md
```

CLIツールによる解析

(CoNLL-U出力コマンドはなし)

コーディング例

```
$ python
import spacy
nlp = spacy.load("ja_core_news_md")
doc = nlp("赤い車を持っている")
for tkn in doc:
    print(tkn.i, tkn.orth_, tkn.lemma_, tkn.pos_, tkn.tag_, tkn.dep_, tkn.head.i)
```

```
0 赤い 赤い ADJ 形容詞-一般 acl 1
1 車 車 NOUN 名詞-普通名詞-一般 obj 3
2 を を ADP 助詞-格助詞 case 1
3 持っ 持つ VERB 動詞-一般 ROOT 3
4 て て SCONJ 助詞-接続助詞 mark 3
5 いる いる AUX 動詞-非自立可能 aux 3
```

spaCy v2 - 解析モデルの学習

標準trainコマンドはjson形式のコーパスが必要

- UD_Japanese-GSD v2.6-NE の ja_gsd-ud-(train|dev|test).ne.json を使用

単語ベクトルの準備

- chiVe mc90 をダウンロードして次のコマンドを実行

```
$ python -m spacy init-model ja ja_vectors_chive_mc90_100k
 --vectors-loc chive-1.1-mc90-20200318.txt.gz -t 100000
```

学習の実行 (v3で大きく変更される予定)

```
$ python -m spacy train ja ja_gtc-2.3.0 ja_gsd-ud-train.ne.json ja_gsd-ud-dev.ne.json
 -v ja_vectors_chive_mc90_100k/ -ovl 0.3 -n 30 -V 2.3.0
```

※ v2.3ではGPU有効化(-g)オプションの効果は乏しい

spaCy vs GiNZA - パイプラインの比較

SudachiPyの複合語分割モードの違い

- spaCy = mode A(最も短い)
  選挙 / 管理 / 委員 / 会
- GiNZA = mode C(最も長い)
  選挙管理委員会

CompoundSplitter

- 複合語をmode=A or Bの短単位に分割可能

BunsetuRecognizer

- 文節＋主辞区間認定
- 主辞品詞分類

※文節拡張モデルが必要なためspaCy日本語モデルとの組み合わせは不可

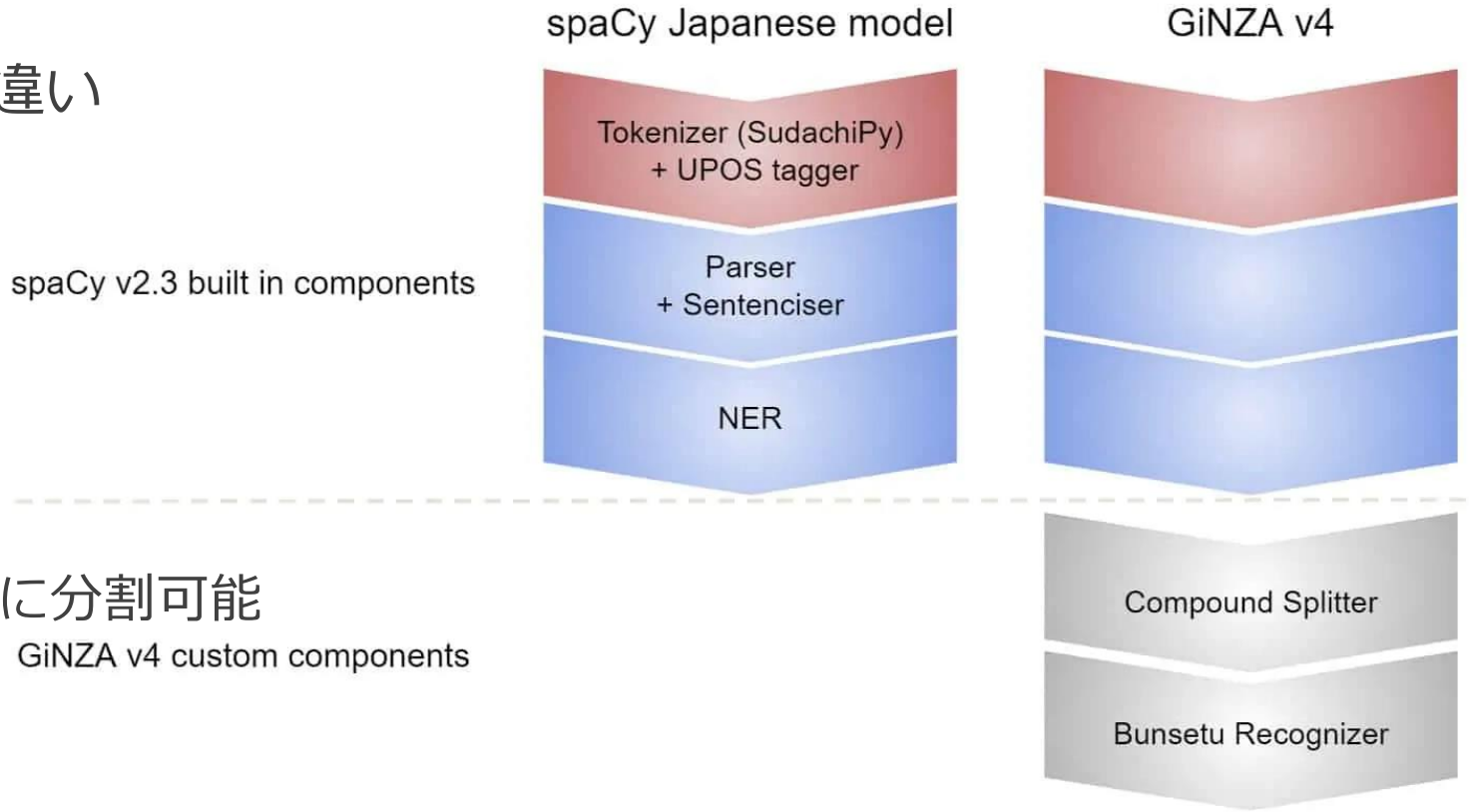
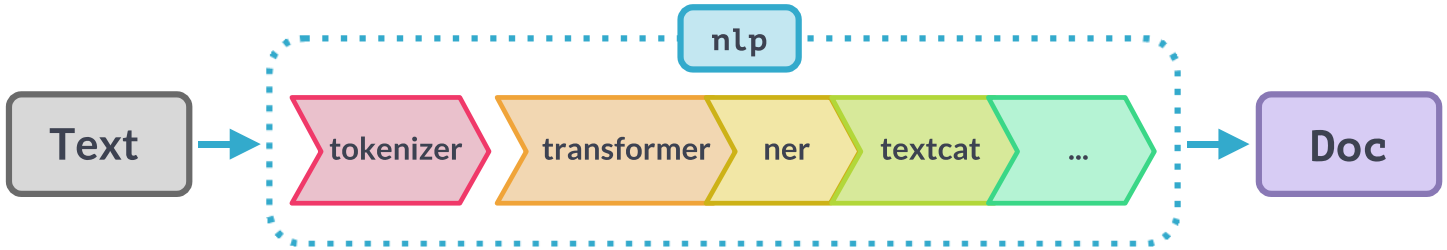
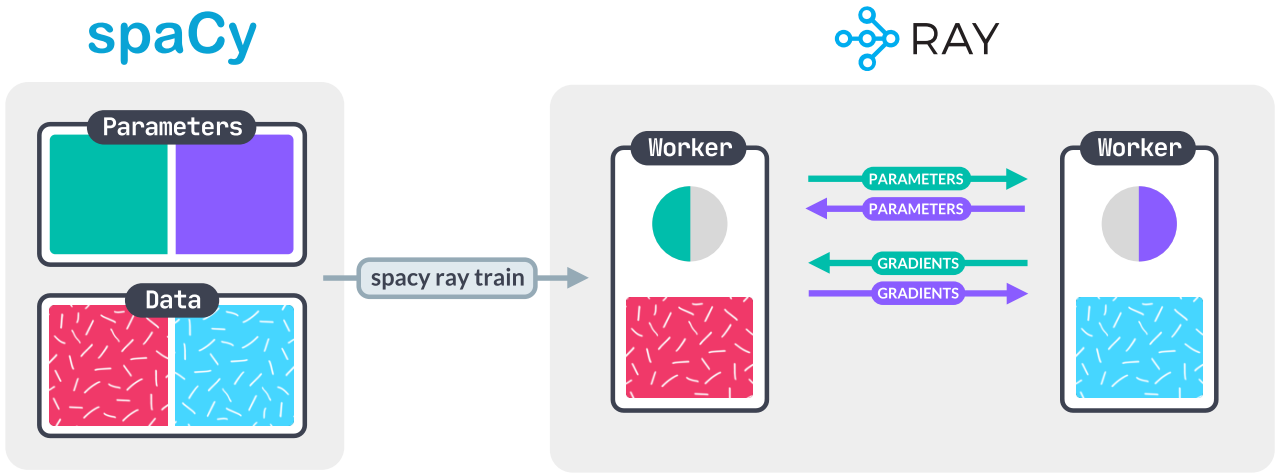


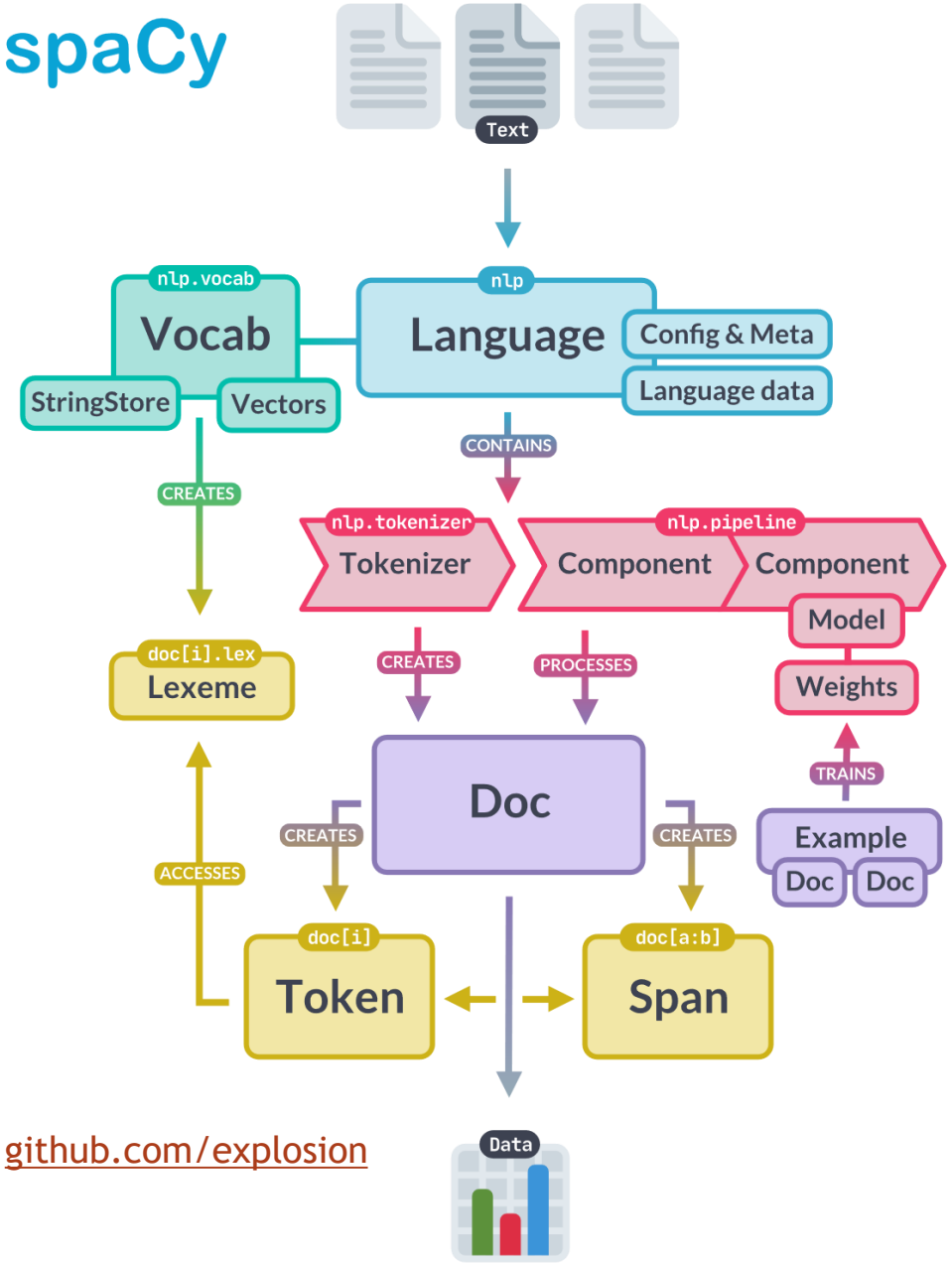
図1 spaCy日本語モデルとGiNZA v4のPipelineの比較

spaCy v3 is coming (Python 3.6+)

- PyTorch, TensorFlow, MXNetの組み込みが可能
- HuggingFace transformersのpretrainモデルに公式対応(英西独仏中の5言語でスタート)
- Rayによる並列分散学習



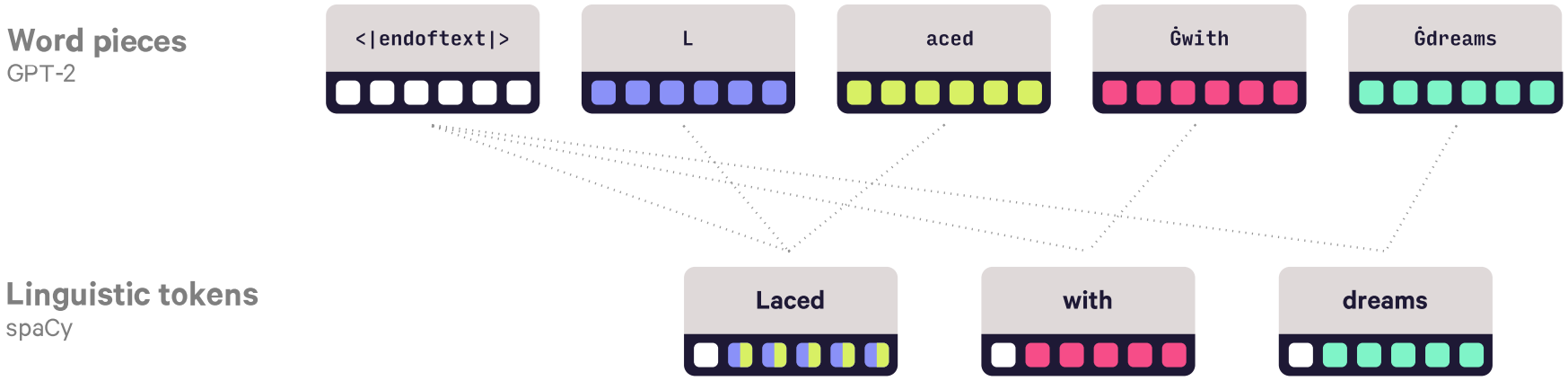




出典: github.com/explosion

spaCy v3 with transformers

- transformersとspaCyのToken Alignment
  - vocab<=32k制限によりtransformersのTokenは細かく切られる
  - 英語でもWordPieceとspaCy Tokenizerの出力間に多対多の対応関係が生じる



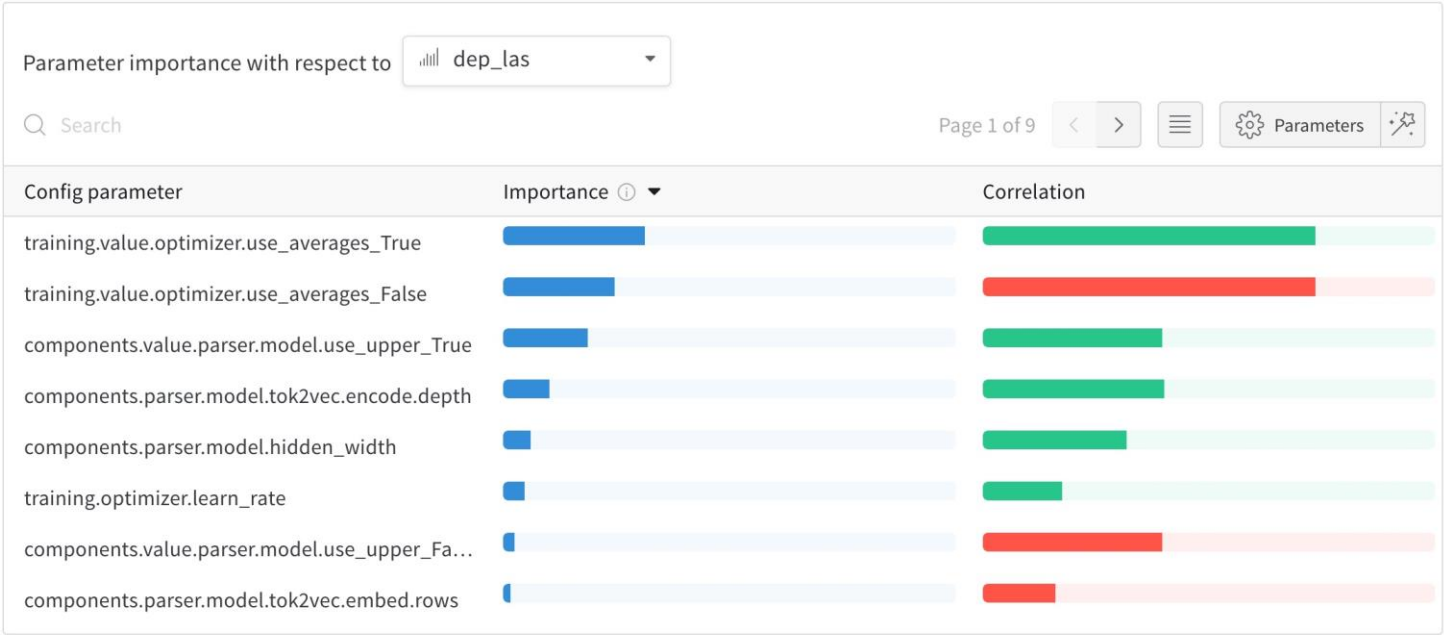
出典: explosion.ai

  - transformers日本語モデルのTokenizerは多様(Juman++, MeCab, SentencePiece)
    - GiNZAのSudachiPy mode Cは長単位のため多対1の対応が多くなり有利か

spaCy v3 - wandb: Weight & Biases

学習ログのトラッキングに WandbLoggerを利用可能

- GridSearchで各種ハイパーパラメータが与える影響を確認できる



出典: github.com/explosion

Stanza - 使い方

インストール (python>=3.6・PyTorch>=1.3.0)

```
$ pip install stanza
$ python
import stanza
stanza.download("ja")
```

解析

```
nlp = stanza.Pipeline(lang="ja")
doc = nlp("赤い車を持っている")
doc_conll = stanza.utils.conll.CoNLL.convert_dict(doc.to_dict())
print(stanza.utils.conll.CoNLL.conll_as_string(doc_conll))
```

Stanza - GPUの有効化

Stanzaより先にpytorch CUDA環境を導入する (CUDA10.1の場合)

```
$ pip install torch==1.6.0+cu101 -f https://download.pytorch.org/whl/torch_stable.html
$ pip install stanza
$ python
import stanza
stanza.download("ja")
```

解析

```
nlp = stanza.Pipeline(lang="ja", use_gpu=True)
```

Stanza - 解析モデルの学習

基本的に公式ドキュメントにある手順に従う

※ただしMWTなど日本語で指定不要なコンポーネントがあるので注意

- UD_Japanese-GSD v2.6-NE の下記ファイルを ./UD_Japanese-GSD/ に配備
  - ja_gsd-ud-*.ne.conllu → ja_gsd-ud-*.conllu にリネームする(ne.をトル)
  - ja_gsd-ud-*.txt
  - (train|dev|test).bio
- chiVe mc90 をxz形式で再圧縮して次のパスに配備

  ./extern_data/word2vec/word2vec/Japanese/ja.vectors.xz

Stanza - 学習の実行手順

スクリプト実行手順 (GPU使用を推奨) - 学習済みモデルを公開中

```
scripts/config.sh
export UDBASE=$(pwd)
export NERBASE=$(pwd)
scripts/run_tokenize.sh UD_Japanese-GSD
scripts/run_pos.sh UD_Japanese-GSD --max_steps 5000
scripts/run_lemma.sh UD_Japanese-GSD --num_epoch 20
scripts/run_depparse.sh UD_Japanese-GSD gold --max_steps 5000
scripts/run_ner.sh UD_Japanese-GSD --word_emb_dim 300 --max_steps 10000
scripts/run_ete.sh UD_Japanese-GSD test
```

Stanza - パイプラインの設定・解析実行

Build Pipeline from a Config Dictionaryを参考にtokenizerにsudachipyを設定

```
import stanza
nlp = stanza.Pipeline(
    lang="ja",
    processors={"tokenize": "sudachipy", "pos": "default", "lemma": "default", "depparse": "default", "ner": "default"},
    # tokenize_model_path="./saved_models/tokenize/ja_gsd_tokenizer.pt",
    pos_model_path="./saved_models/pos/ja_gsd_tagger.pt",
    pos_pretrain_path="./saved_models/pos/ja_gsd.pretrain.pt",
    lemma_model_path="./saved_models/lemma/ja_gsd_lemmatizer.pt",
    parser_model_path="./saved_models/depparse/ja_gsd_parser.pt",
    parser_pretrain_model_path="./saved_models/depparse/ja_gsd.pretrain.pt",
    ner_model_path="./saved_models/ner/ja_gsd_nertagger.pt",
    ner_forward_charlm_path=None,
    ner_backward_charlm_path=None,
)
print(nlp("赤い車に乗っている"))
```

Benchmark: spaCy vs GiNZA vs Stanza

UD_Japanese-GSD v2.6-NEを用いた精度と速度の比較評価結果

- 言語処理学会論文誌「自然言語処理」 Volume 27 Number 3
  GiNZA - Universal Dependenciesによる実用的日本語解析 から抜粋
- GiNZA・Stanzaはv2.6-NEで学習モデルを独自に構築

表 3　日本語 GSD v2.6-NE を用いた日本語 UD 解析系のベンチマーク

	Tokenize	TAG	UPOS	UAS	LAS	MLAS	Ent	Load	CPU	GPU
spaCy v2.3.0	**0.981**	**0.970**	**0.953**	0.894	0.875	0.856	0.765	2.6	**183**	**212**
GiNZA v4.0.0	**0.981**	**0.970**	**0.953**	**0.908**	**0.888**	**0.868**	0.714	**1.3**	164	180
Stanza v1.0.1	0.970	-	0.952	0.890	0.877	0.801	**0.805**	6.5	11	37

Load:loading(library+model)[sec], CPU,GPU:speeds[sent/sec], Environment: Windows10 Pro Update1909 (CPU:WSL, GPU:CUDA10.0), Python version 3.8.3, Core i9-8950H, DDR4 64GB, Intel 760p, RTX2080

Python GPU Applicationの頑張りどころ

Cythonで高速化したいがコンパイル済みWheelを配布するのに一苦労

- コンパイル対象はOSバリエーション x アプリが対応するPythonバージョン
- manylinux等を駆使してGitHub ActionsでCI化は可能
- WheelにDLL等のランタイムを追加する必要があったりで結局かなり面倒

PyPIから配布可能なインストールイメージは60MBに制限されている

- pip install -r にはPyPI以外のロケーションを含めることができない
- 60MB超のイメージを置くための申請にはかなりの手間と時間がかかる
  - 大容量の学習済みモデルを置くことは拒否される
- 数GBに及ぶ学習モデルをCDN経由で配布できる枠組みが望まれる


Megagon Labs


GiNZA