ニューラルグラフ型係り受け解析器のための文節ベクトル表現

松野 智紀　　能地 宏　　松本 裕治
奈良先端科学技術大学院大学 情報科学研究科
{matsuno.tomoki.mr1, noji, matsu}@is.naist.jp

1 はじめに

近年, 自然言語処理の様々な基礎解析タスクにおいて, リカレントニューラルネットワーク, 特に双方向 LSTM による特徴抽出に基づくモデルが大きな成功を収めている. この状況は構文解析でも同様である. 特に LSTM の出力ベクトルから各単語間の依存構造の尤もらしさを独立に計算する単純なグラフ型の係り受け解析器 [6, 1] が英語の Penn Treebank 上で最高精度を達成した他, 2017 年の CoNLL shared task でも多言語構文解析部門でトップとなった [2]. 単純ながら, LSTM が文中の大域的な文脈を捉えた表現を得ることを可能にし, 構文解析に有効な素性が得られているのだと考えられる.

本稿では, このグラフ型構文解析を日本語の文節係り受け解析に適用する際の問題点について議論し, 日本語に適したネットワーク構造を探求する. Universal Dependencies を始めとし, 単語単位の日本語係り受け解析も研究が進んできているが, 現在も日本語では文節単位の係り受け解析が主流である. CaboCha など従来の手法は疎な素性に基づくが, これらは二つの文節間の係りやすさを計算する際, 各文節から主辞など重要な単語を抜き出し, それらの組み合わせを素性として利用する. 対して上記ニューラルネットワークに基づく手法では, 密なベクトル間の双線型変換などにより要素間の係りやすさを計算するため, まず文節に対して適切な密ベクトルを割り当てる必要がある.

本稿で最も焦点を当てるのは, この文節ベクトルの有効な構築法である. 京大コーパス上でいくつかの異なるモデルを比較した結果, 以下のことが明らかになった. まず, 文節間の係りやすさを計算する際の, 文節単位の LSTM 層を用意することの重要性を確認した. また, この文節単位の LSTM 層への入力として, 単語単位 LSTM 層の状態の差分を用いる方法の有効性を確認した. この差分に基づく方法は, 近年句構造解析におけるスパンの表現を得るために用いられているものである [5]. 提案法は京大コーパスにおける実験で 91.23%を記録し, 注意深く設計された既存の解析器に迫る精度を得ることができた.

2 Deep Biaffine parser

本稿で考えるモデルは, Dozat らの Deep Biaffine parser [1] を基礎とする. 関連するモデルはいくつか発表されているが, これは CoNLL 2017 shared task において最高精度を達成したものである.

このモデルは以下の構造になっている.

(1) まず, 入力文の単語列と品詞列から各単語のベクトル表現を作り双方向 LSTM に入力して新たな単語ベクトル表現にエンコードする. i 番目の単語 w_i に対する双方向 LSTM の出力ベクトルを $\mathbf{y}_i$ とする.

(2) 各単語ベクトル表現のペアを, 係り元と係り先で異なる多層パーセプトロンを使って次元圧縮する.

$$\mathbf{h}_i^{head} = \mathrm{MLP}^{head}(\mathbf{y}_i),$$
$$\mathbf{h}_j^{dep} = \mathrm{MLP}^{dep}(\mathbf{y}_j).$$

(3) 最後に, Biaffine 変換によって各単語ペアの係り受け関係の尤度を計算する.

$$s_{i,j} = \mathbf{h}_i^{head^{\mathrm{T}}} U \mathbf{h}_j^{dep} + \mathbf{h}_i^{head^{\mathrm{T}}} \mathbf{u}.$$

$s_{i,j}$ は, 単語 w_j の w_i への係りやすさのスコアである. Biaffine 変換は二つの項からなっている. 最初の項は, パラメータ U による双線型変換で, 二単語の係りやすさを計算する. それに加え, 二番目の項が, w_i が主辞になりやすいかどうかを表現する働きを持つ.

以上の操作により, 各単語ペア毎にスコア $s_{i,j}$ が得られる. 最後にこれらを入力として Chu-Liu/Edmonds アルゴリズムを走らせることで, 合計スコアが最大となる木構造を得ることができる.

3 提案手法

ここでは Deep Biaffine parser の日本語の文節係り受け解析への適用について論じる．1 節で述べたように, 文節係り受けでは文節間の係りやすさを計算するために, 文節に対する適切なベクトル表現を得なくてはならない．それらが得られれば, 前節の (2) 以降の手順により, i 番目と j 番目の文節間の係りやすさを $s_{i,j}$ として計算することができる．

以下 3.1 節でモデルを説明するための表記についてまとめた後, 3.2 節で文節ベクトルの構成法について述べる．ここでは 3 つの方法を述べた後, 4 節でそれらの比較実験を行う．

3.1 表記

本研究では文節区切りおよび形態素解析の行われた文を入力として仮定する．文中の単語数を n, 文節数を m で表す．以下, 単語層と文節層にそれぞれ異なる LSTM を導入するが, どちらに対しても, $\overrightarrow{\mathbf{x}_i}$ で i 番目の要素 (単語または文節) の右方向 LSTM への入力, $\overrightarrow{\mathbf{y}_i}$ でその出力を表す．また逆に, $\overleftarrow{\mathbf{x}_i}$ で対応する左方向 LSTM への入力, $\overleftarrow{\mathbf{y}_i}$ でその出力を表すものとする．単語層 LSTM の入力または出力であることを示す必要があるときは右肩に $word$, 文節層の場合は $chunk$ と記して表記する．また文中の i 番目から j 番目の語を含む範囲を (i, j) と表す．

3.2 文節表現モデル

入力　モデルはまず各単語 w_t を表すベクトル $\mathbf{x}_t$ を入力として受けとる．このベクトルは, 単語や品詞などの埋め込みベクトルを結合したもので, 本研究ではこれを, (単語表記, 活用型, 活用形, 品詞, 品詞細分類) を表す各埋め込みベクトルと, 単語の文字表現を表すベクトルとの結合により生成する．単語の文字表現は, 文字用の双方向 LSTM を用い, その始点と終点の文字に当たる出力を結合することで得る．また, 文頭と文末には BOS, EOS という特別な記号を用意した．これらは文節のベクトルを単語 LSTM の差分で表すときに必要となるものである．

LSTM-Minus による文節表現　以下で紹介するいくつかのモデルは, 内部で LSTM-Minus という仕組みを利用する．これは, 単語単位の LSTM から複数単語にまたがるスパンに対する表現を, LSTM の出力間の差分により表現しようというもので, 最初にグラフ型構文解析の素性として導入された [6] 後, 最近句構造解析でも各句に対応する表現を得る目的で利用され [5], 有効性が示されているものである．具体的には, 単語層 LSTM の出力 $\{\overrightarrow{\mathbf{y}}_i^{word}, \overleftarrow{\mathbf{y}}_i^{word}\}$ が得られているとき, (i, j) の範囲に当たる文節の表現を, 右方向の差分 $\overrightarrow{\mathbf{y}}_j^{word} - \overrightarrow{\mathbf{y}}_{i-1}^{word}$ と, 左方向の差分 $\overleftarrow{\mathbf{y}}_i^{word} - \overleftarrow{\mathbf{y}}_{j+1}^{word}$ によって表現する．これらは, 単語層 LSTM へ文節に含まれる語が入力される前の時刻の状態と入力された後の時刻の状態との差分をとることを意味しており, 自然に文節の表現を得ることができると考えられる．

モデルの構造　次に, これらを用いた具体的なモデルの構成について述べる．文節を入力とした Biaffine parser を構築する際の最も単純な方法は, 単語層 LSTM を用いずに, 文節内の単語ベクトルの平均等で文節を表現し, それらを文節層 LSTM の入力とするものである．しかしながら, この方法では単語同士の統語的なやりとりを直接モデル化することができず, Biaffine parer が単語単位の係り受け解析で成功していることを鑑みると, 単語に対する LSTM でのモデル化は重要であることが予測される．そこで本研究では, 単語層 LSTM と文節層 LSTM の二つを上記 LSTM-Minus により結合したモデルを主な提案手法として, それよりも単純な二つのモデルと比較することで, 単語層 LSTM や LSTM-Minus の効果について議論することとする．

- **LSTM-Minus + 文節層 LSTM**: この方法ではまず, 単語層 LSTM により入力単語ベクトル $\mathbf{x}_i$ を二つの出力ベクトル $\overleftarrow{\mathbf{y}}_i^{word}$ と $\overrightarrow{\mathbf{y}}_i^{word}$ に変換する．

$$\overrightarrow{\mathbf{y}}_i^{word} = \mathrm{LSTM}_l^{word}(\mathbf{x}_i),$$
$$\overleftarrow{\mathbf{y}}_i^{word} = \mathrm{LSTM}_r^{word}(\mathbf{x}_i).$$

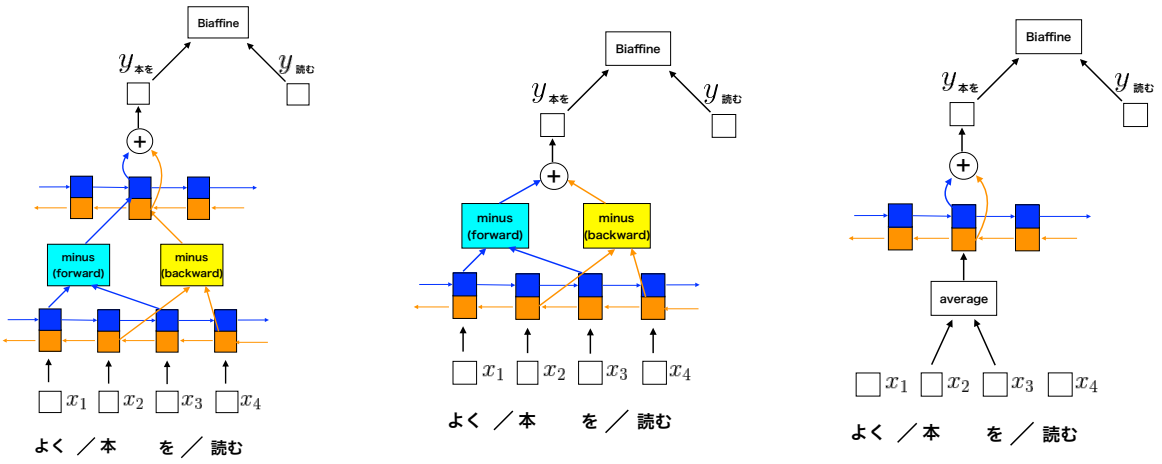
  ここで LSTM_l^{word} と LSTM_r^{word} はそれぞれ左方向, 右方向 LSTM であり, 入力列 $\{\mathbf{x}_i\}$ を受け取る．そして, t 番目の文節の範囲が (i, j) であるとき, 文節層 LSTM の入力を, LSTM-Minus を用いて以下のように求める．

$$\overrightarrow{\mathbf{x}}_t^{chunk} = \overrightarrow{\mathbf{y}}_j^{word} - \overrightarrow{\mathbf{y}}_{i-1}^{word},$$
$$\overleftarrow{\mathbf{x}}_t^{chunk} = \overleftarrow{\mathbf{y}}_i^{word} - \overleftarrow{\mathbf{y}}_{j+1}^{word}.$$

  最後にこれらを入力とする文節層 LSTM により出力 $\{\overrightarrow{\mathbf{y}}_t^{chunk}, \overleftarrow{\mathbf{y}}_t^{chunk}\}$ を得て, これらを結合することで, Biaffine 変換への入力 $\mathbf{y}_t$ を得る．

$$\mathbf{y}_t = \overrightarrow{\mathbf{y}}_t^{chunk} \oplus \overleftarrow{\mathbf{y}}_t^{chunk}$$

  このモデルの概略を図 1a に示す．なお実験では単語層に 2 層の LSTM を, 文節層に 1 層の LSTM を用いる．



(a) LSTM-Minus+文節層 LSTM　(b) LSTM-Minus　(c) 文節層 LSTM のみ

図 1: 比較する三つのモデル．$y_{本を}$ などは各文節を表現するベクトルで，Biaffine 変換への入力となる．ここでは $y_{本を}$ の生成のみを示すが，$y_{読む}$ なども同様に作られ，スコア計算に利用される．

- **LSTM-Minus**: より単純なモデルとして，上のモデルから文節層の LSTM を除き，単語層からの LSTM-Minus の結果を文節表現としたものを比較対象とする．すなわち，上記 $\overrightarrow{\mathbf{x}}_t^{chunk}, \overleftarrow{\mathbf{x}}_t^{chunk}$ を用いて，

$$\mathbf{y}_t = \overrightarrow{\mathbf{x}}_t^{chunk} \oplus \overleftarrow{\mathbf{x}}_t^{chunk}$$

  となる．このモデルを図 1b に示す．なお，上記のモデルと LSTM の層数を合わせるため，本モデルでは 3 層の単語層 LSTM を用いる．

- **文節層 LSTM のみ**: 最後に単語層 LSTM の必要性を調べる目的で，文節層 LSTM のみを用いるモデルを用意する．ここでは単純な方法として，文節内の単語の平均により文節層 LSTM の入力とする方法を採用した．つまり，

$$\overrightarrow{\mathbf{x}}_t^{chunk} = \overleftarrow{\mathbf{x}}_t^{chunk} = average(\mathbf{x}_i, \cdots, \mathbf{x}_j)$$

  とし，これらを文節層 LSTM への入力とする．ここでは文節層に 3 層の LSTM を用いる．モデルの概略を図 1c に示す．

4 実験

これまで述べたモデル間の比較および，日本語の既存の構文解析器との比較を行う．提案モデルの実装には，深層学習ライブラリ dynet [4] を利用した．

4.1 実装詳細

データ　実験には京大コーパス 4.0 を用いた．Yoshinaga ら [7] に従い，次に示す分割でモデルの訓練・評価を行った．

- **訓練**: 1 月 1-11 日の記事と，1 月から 8 月の社説．
- **開発**: 1 月 12-13 日の記事と，9 月の社説．
- **評価**: 1 月 14-17 日の記事と，10 月から 12 月の社説．

実験で用いる単語境界ならびに文節境界には京大コーパスの正解アノテーションを利用する．単語の埋め込みベクトルも含め，モデルの学習に外部資源は一切利用しない．

パラメータ選択　パラメータの最適化は確率的勾配降下法 (SGD) で行う．学習係数は焼きなまし法を用いた Adam [3] で自動調整する．焼きなまし法の設定について，学習係数の減衰率を 0.9 に設定した以外は Dozat ら [1] の設定に従った．

次にハイパーパラメータとモデルの初期値について述べる．表 1 に，LSTM 隠れ層の次元を含めたモデル全体の大きさをまとめた．単語や品詞などの埋め込みベクトルは，dynet のデフォルトの設定に従って初期化を行った．また LSTM の重みパラメータ行列は，直交行列により初期化した．最後に，単語層 LSTM，文節層 LSTM，多層パーセプトロンの各層にドロップアウト率 0.1 を設定した．

パラメータ名	次元	パラメータ名	次元
単語層 LSTM 隠れ層	400	表記素性	100
文節層 LSTM 隠れ層	400	活用型素性	50
文字層 LSTM 隠れ層	50	活用形素性	50
多層パーセプトロン層	500	品詞	100
文字埋め込み	100	品詞細分類	100

表 1: 実験に用いたパラメータの次元

	単語層	文節層	精度
提案法	2	1	91.23
LSTM-Minus	3	0	90.13
平均 + 文節層 LSTM	0	3	90.52
J.DepP			92.29
CaboCha			91.84

表 2: 文節係り受けの精度. 比較手法の評価基準に習い, 最も右の文節については係り受けの評価から除外した.

4.2 比較手法

3 つの提案法のほか, 既存の日本語構文解析器と比較するため, 代表的な二つの解析器との比較を行った. 一つは CaboCha [8] で, 上で述べた訓練データ上でモデルの学習を行い評価した. もう一つは J.DepP [7] で, こちらはホームページ上[1]に掲載されている同じ訓練データを用いた場合の数値を引用した. これらはいずれも決定的な Shift-reduce 型の構文解析器で, 各動作の選択を文節内の単語から抽出した組み合わせ素性を用いて行う.

4.3 結果

表 2 に解析結果をまとめた. 単語層もしくは文節層のみを用いるモデルと比較し, 両方を用いるモデルで約 1 ポイントの精度向上が見られた. これは, 2 種類の LSTM を用意することの有効性, 及び LSTM-Minus による素性抽出が文節表現にも有効であることを示すものである. どちらかの LSTM を用いた場合, 文節 LSTM のみを用いる手法の方が上回った. これは, Biaffine 変換により係りやすさを計算する際, その入力は直接 LSTM の出力であった方が良いことを示唆している. 精度 91.23% は高精度であるが, 注意深く設計された素性を用いる既存の手法には若干劣る数値である. 単語単位の係り受け解析では, ニューラルネットワークに基づく手法が他を圧倒し始めていることを考えると, 日本語文節係り受け解析に置いて, この差を埋めることができるのかを明らかにすることは今後の課題である.

[1] http://www.tkl.iis.u-tokyo.ac.jp/ ynaga/jdepp/

5 おわりに

本稿では, 双方向 LSTM を用いたグラフ型構文解析器の日本語文節係り受けへの適用について議論した. 文節をエンコードする際, 双方向 LSTM に単語層と文節層の 2 種類を用意することでタスクの精度が向上することが分かった. また, LSTM 出力の差によって文中のスパンの表現を得る LSTM-Minus が, 日本語の文節に対する表現にとっても有用であることを示した. 今後の課題として, (i) 単語・文節分割や品詞タグ付けも含めた end-to-end の文節単位構文解析タスクへの拡張や, (ii) 話し言葉など比較的文節の並びが自由である場合の構文解析タスクへの応用などが考えられる.

参考文献

[1] Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734, 2016.

[2] Timothy Dozat, Peng Qi, and Christopher D. Manning. Stanford's graph-based neural dependency parser at the conll 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[3] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv: 1412.6980*, 2014.

[4] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gopalakrishnan Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. Dynet: The dynamic neural network toolkit. *CoRR*, abs/1701.03980, 2017.

[5] Mitchell Stern, Jacob Andreas, and Dan Klein. A minimal span-based neural constituency parser. In *Proc. of ACL*, July 2017.

[6] Wenhui Wang and Baobao Chang. Graph-based dependency parsing with bidirectional lstm. In *ACL*, 2016.

[7] Naoki Yoshinaga and Masaru Kitsuregawa. Polynomial to linear: Efficient classification with conjunctive features. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3*, EMNLP '09, pages 1542–1551, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[8] 工藤 拓 and 松本 裕治. チャンキングの段階適用による日本語係り受け解析. 43(6):1834–1842, 2002.