# Structured Training for Neural Network Transition-Based Parsing

渡辺有祐

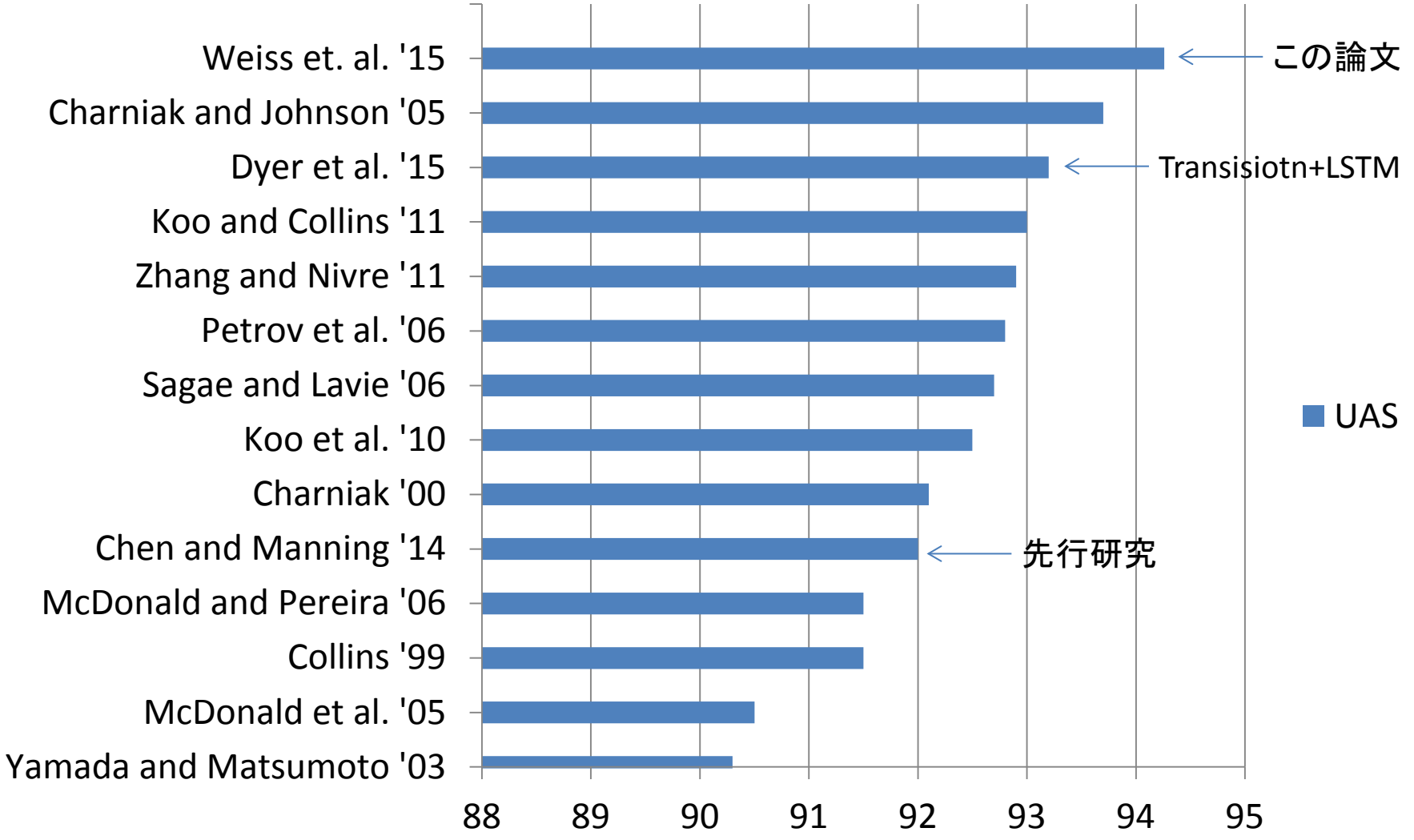
SONY

# 論文 概要

- transition-based dependency parsingをNeural Netを用いて行った

- PennTreebankデータセットで最高性能を達成
  - UAS: 94.26%
  - LAS: 92.41%

# 性能比較 (PTB, UAS)

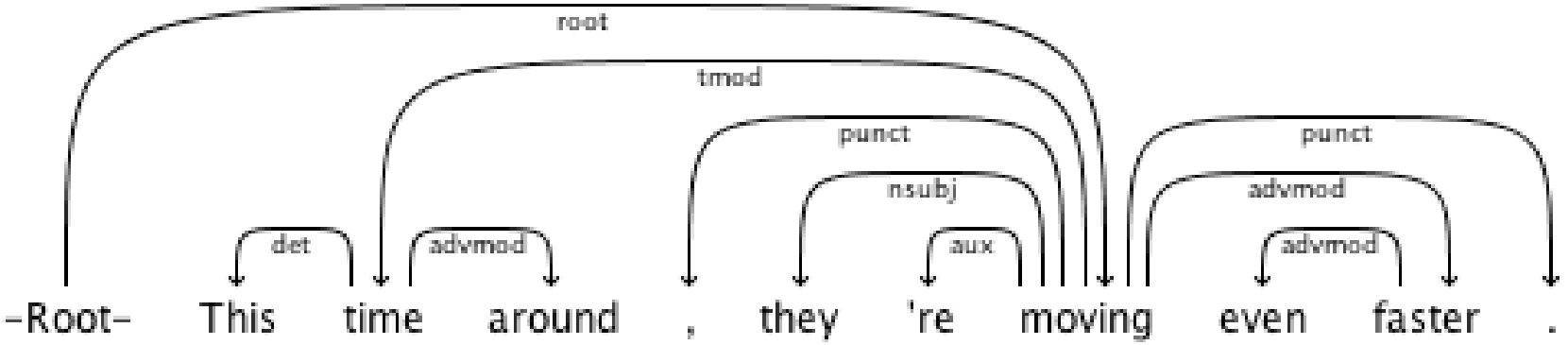



http://nodalida2011.lumii.lv/slides/nivre.pdf

# 関連研究

- 先行研究
  - [Chen and Manning EMNLP2014] A fast and accurate dependency parser using neural networks
- 関連研究
  - [Dyer et al. ACL2015] Transition-Based Dependency Parsing with Stack Long Short-Term Memory
  - [Alberti et al. EMNLP2015] Improved Transition-Based Parsing and Tagging with Neural Networks

# dependency parsing




- 文から dependency parsing を計算する手法は主に **Graph-based** と **Transition-based** がある。
- **この論文では Transition-basedなアプローチを採用する。**

# Transition-based parsing (arc-standard)

能地さんスライド

## Arc-standard

People write Java with tears

Stack

Buffer (Input)

People write Java with tears

SH

Actions

SH Shift: Bufferの先頭をStackに移動

L LeftReduce: スタックの先頭２つをくっつける（右がhead）

R RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears

| Stack | Buffer (Input) |
| --- | --- |
| People | write Java with tears |

**SH SH**

Actions

**SH** Shift: Bufferの先頭をStackに移動
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）
**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears

| Stack | Buffer (Input) |
| --- | --- |
| People write | Java with tears |

**SH SH L**

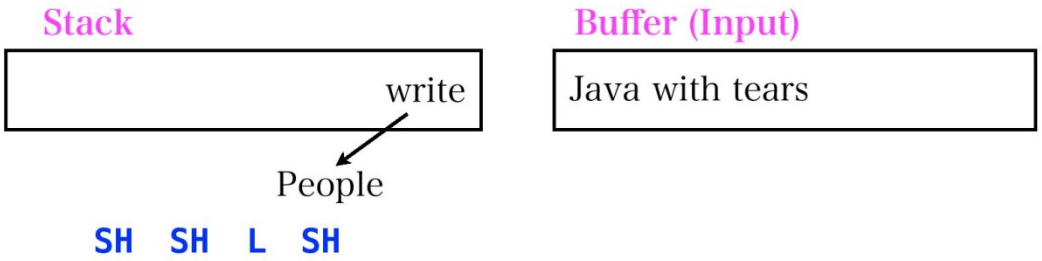
Actions

**SH** Shift: Bufferの先頭をStackに移動
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）
**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears




**SH SH L SH**

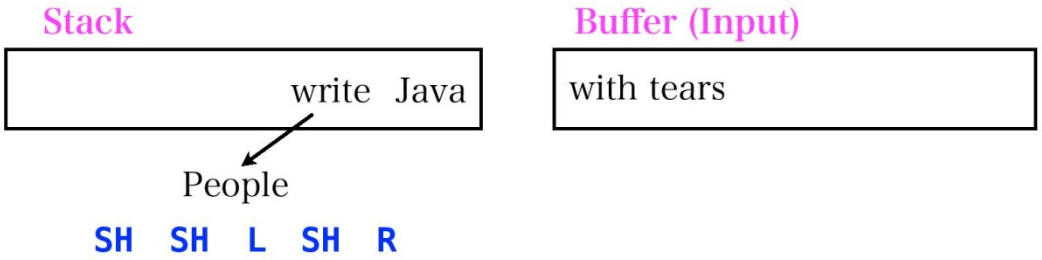
Actions

**SH** Shift: Bufferの先頭をStackに移動
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）
**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears




Actions

**SH** Shift: Bufferの先頭をStackに移動

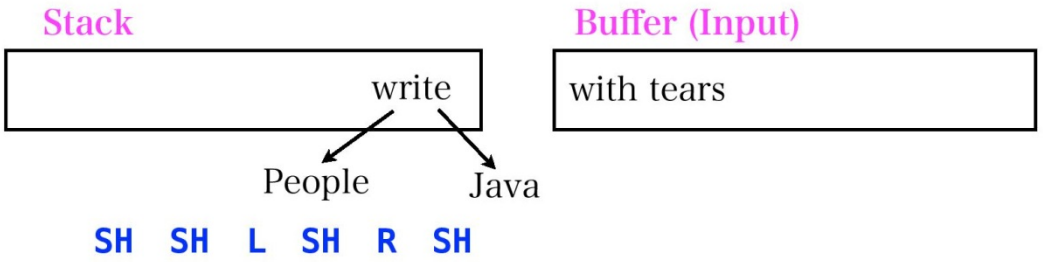
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）

**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears




Actions

**SH** Shift: Bufferの先頭をStackに移動

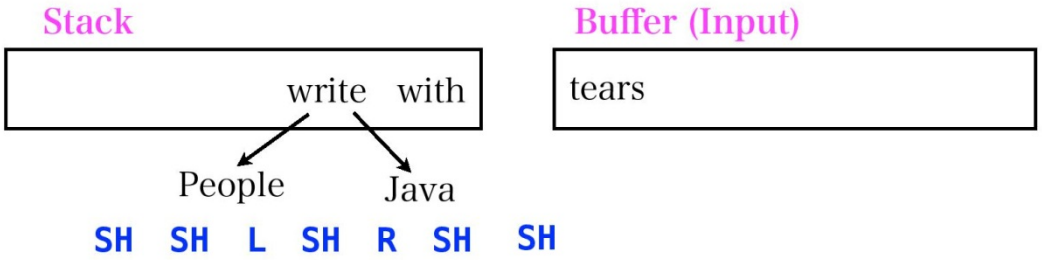
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）

**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears




Actions

**SH** Shift: Bufferの先頭をStackに移動

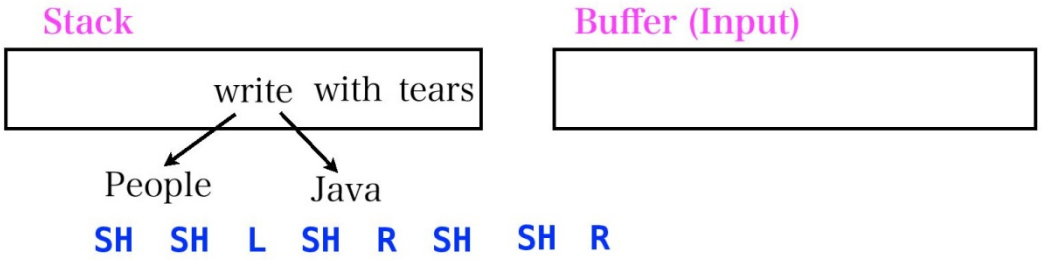
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）

**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears




Actions

**SH** Shift: Bufferの先頭をStackに移動

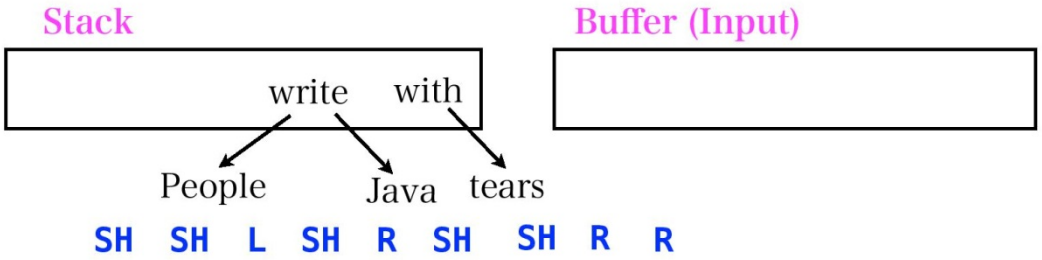
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）

**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears




Actions

**SH** Shift: Bufferの先頭をStackに移動

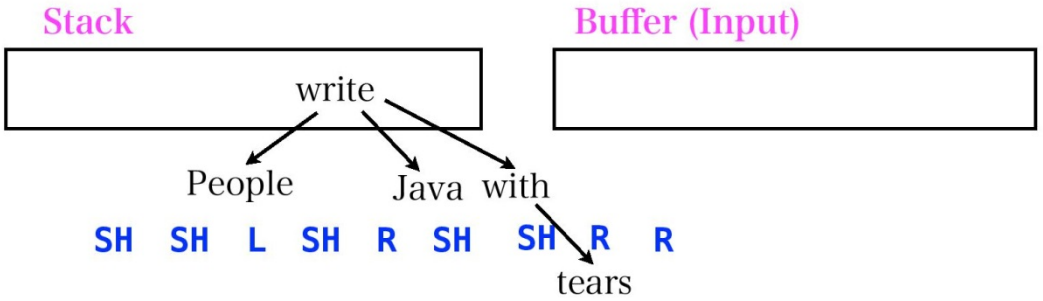
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）

**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

# Transition-based parsing (arc-standard)

## Arc-standard

People write Java with tears




Actions

**SH** Shift: Bufferの先頭をStackに移動

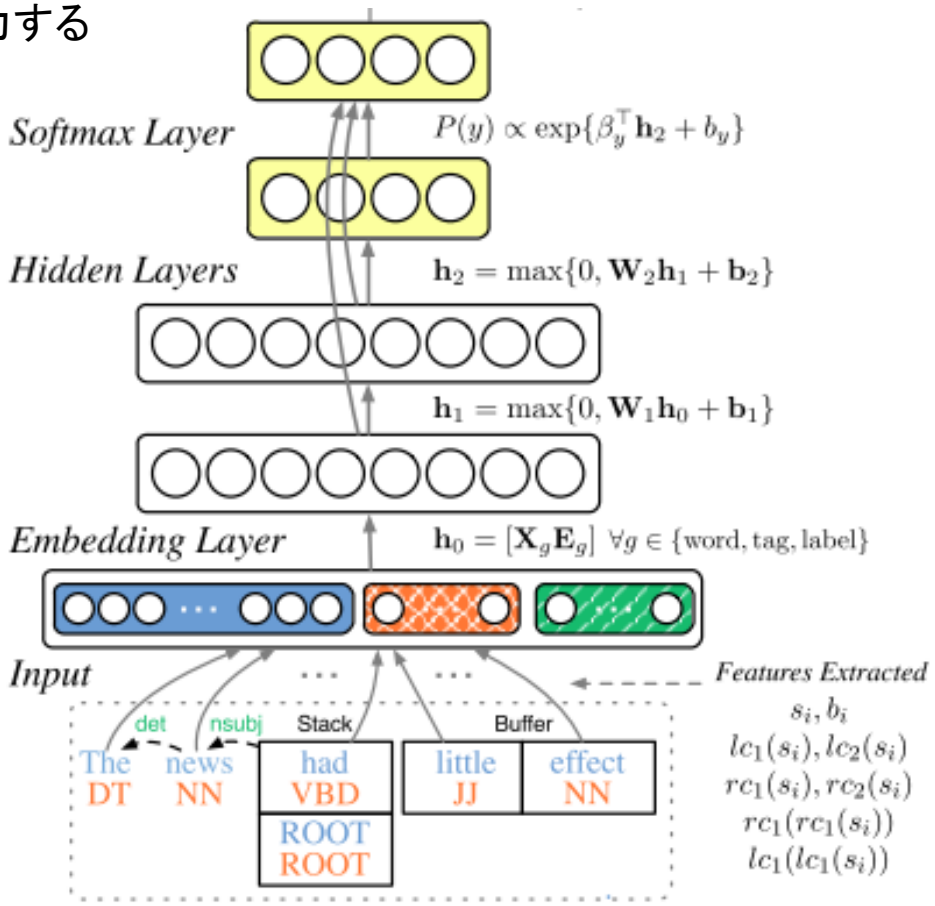
**L** LeftReduce: スタックの先頭２つをくっつける（右がhead）

**R** RightReduce: スタックの先頭２つをくっつける（左がhead）

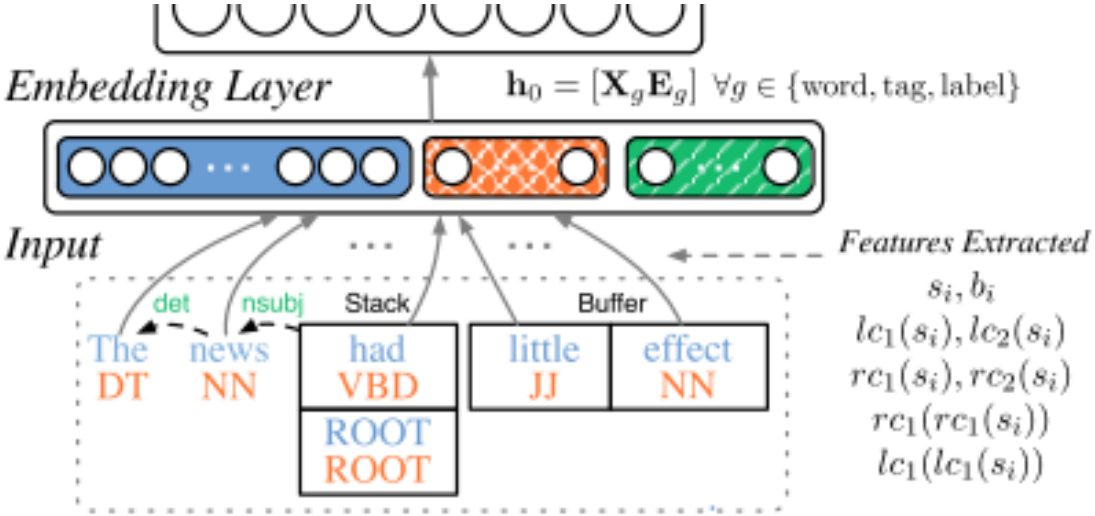
# Transitionの学習

- ある”状態”で、{Shift, Left, Right} のどのtransitionを行うのかの判別器を学習する
  - “状態” を表現する特徴ベクトルを入力する
  - 判別器=NN

$$L(\Theta) = -\sum_j \log P(y_j \mid c_j, \Theta) + \lambda \sum_i \|\mathbf{W}_i\|_2^2,$$

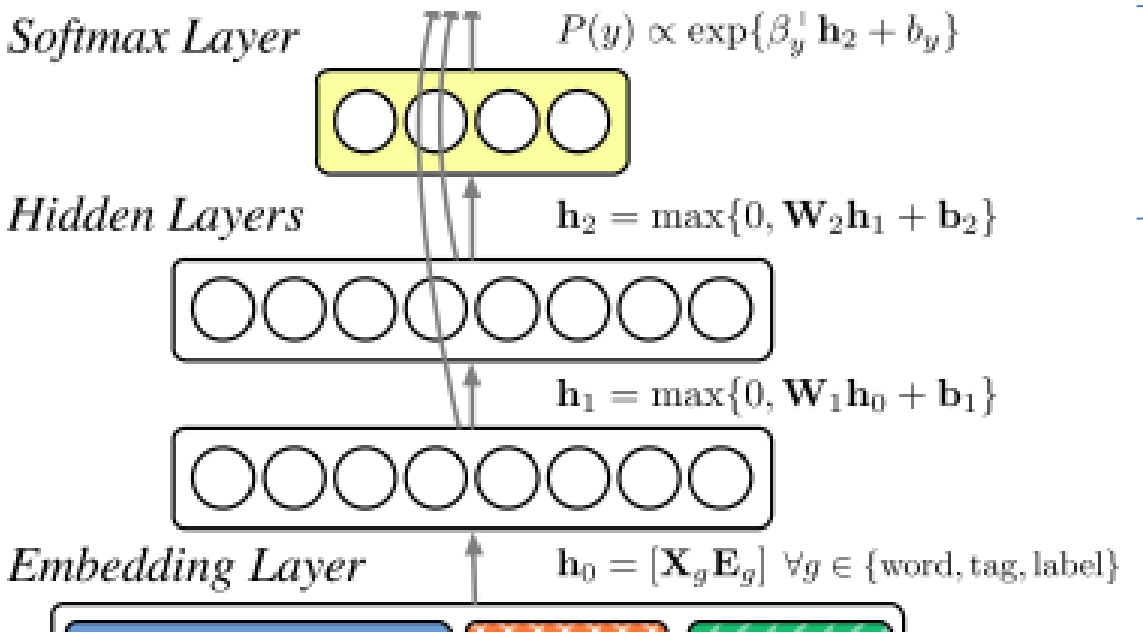


# Neural Network Model (入力部)




- 入力は0,1ベクトル
  - スタック内の単語、POSタグ、ラベルなどのフラグ
- これが、denseなベクトルになる

# Neural Network Model




最後はSoftmax

全結合 with Relu

# Structured Perceptron (1/2)

- 系列全体を見て、Transitionを決めたほうが良さそう ⇒ モデルの最後段に Structured Perceptron を付け加えよう!
  - これにより、0.6%程度性能向上

x: 文
yi: i番目のtransition

$$\underset{y \in \mathrm{GEN}(x)}{\operatorname{argmax}} \sum_{j=1}^{m} \mathbf{v}(y_j) \cdot \phi(x, y_1 \ldots y_{j-1})$$

文xに対する
transitionの全系列

j番目までの
transitionの系列

argmaxは動的計画法で計算出来ない。
Beam Searchで良い物を選ぶ。

# Structured Perceptron (2/2)

$$\underset{y \in \mathrm{GEN}(x)}{\operatorname{argmax}} \sum_{j=1}^{m} \mathbf{v}(y_j) \cdot \underbrace{\phi(x, y_1 \ldots y_{j-1})}_{[\mathbf{h}_1\ \mathbf{h}_2\ P(y)]}$$

この特徴量に、先ほどのNNの出力と中間層の発火を用いる。

- 係数v(y)を学習する
- 学習方法
  - ビームサイズをBとする
  - 各訓練サンプル文に対し、beam searchを実行していき、正解の系列がB個の中に入らなくなった時点jで止める
  - B個の中でスコアが最高のものをnegative sampleとして学習

# 実験結果 (PTB)

| Method | UAS | LAS | Beam |
|---|---|---|---|
| *Graph-based* | | | |
| Bohnet (2010) | 92.88 | 90.71 | n/a |
| Martins et al. (2013) | 92.89 | 90.55 | n/a |
| Zhang and McDonald (2014) | 93.22 | 91.02 | n/a |
| *Transition-based* | | | |
| ⋆Zhang and Nivre (2011) | 93.00 | 90.95 | 32 |
| Bohnet and Kuhn (2012) | 93.27 | 91.19 | 40 |
| Chen and Manning (2014) | 91.80 | 89.60 | 1 |
| S-LSTM (Dyer et al., 2015) | 93.20 | 90.90 | 1 |
| Our Greedy | 93.19 | 91.18 | 1 |
| Our Perceptron | **93.99** | **92.05** | 8 |
| *Tri-training* | | | |
| ⋆Zhang and Nivre (2011) | 92.92 | 90.88 | 32 |
| Our Greedy | 93.46 | 91.49 | 1 |
| Our Perceptron | **94.26** | **92.41** | 8 |

# まとめ

- NNを使って、高性能なdependency parserを作ることが出来た。
- 手法は、[Chen and Manning 2014]を改良したもの
  - NNの構成を改良
  - 最後にStructured Perceptronを付け加えた
  - “Tri-training”

- 感想
  - 本気でチューニングすればもう少し性能が上がりそう