Word Embeddings in Feedforward Networks; Tagging and Dependency Parsing using Feedforward Networks

Michael Collins, Columbia University

Overview

- Introduction
- Multi-layer feedforward networks
- Representing words as vectors (“word embeddings”)
- The dependency parsing problem
- Dependency parsing using a shift-reduce neural-network model

Multi-Layer Feedforward Networks

- An integer d specifying the input dimension. A set $\mathcal{Y}$ of output labels with $|\mathcal{Y}| = K$.
- An integer J specifying the number of hidden layers in the network.
- An integer m_j for $j \in \{1 \ldots J\}$ specifying the number of hidden units in the j'th layer.
- A matrix $W^1 \in \mathbb{R}^{m_1 \times d}$ and a vector $b^1 \in \mathbb{R}^{m_1}$ associated with the first layer.
- For each $j \in \{2 \ldots J\}$, a matrix $W^j \in \mathbb{R}^{m_j \times m_{j-1}}$ and a vector $b^j \in \mathbb{R}^{m_j}$ associated with the j'th layer.
- For each $j \in \{1 \ldots J\}$, a transfer function $g^j : \mathbb{R}^{m_j} \to \mathbb{R}^{m_j}$ associated with the j'th layer.
- A matrix $V \in \mathbb{R}^{K \times m_J}$ and a vector $\gamma \in \mathbb{R}^K$ specifying the parameters in the output layer.

Multi-Layer Feedforward Networks (continued)

- Calculate output of first layer:

$$\begin{aligned} z^1 \in \mathbb{R}^{m_1} &= W^1 x^i + b^1 \\ h^1 \in \mathbb{R}^{m_1} &= g^1(z^1) \end{aligned}$$

- Calculate outputs of layers $2 \ldots J$:
  For $j = 2 \ldots J$:

$$\begin{aligned} z^j \in \mathbb{R}^{m_j} &= W^j h^{j-1} + b^j \\ h^j \in \mathbb{R}^{m_j} &= g^j(z^j) \end{aligned}$$

- Calculate output value:

$$\begin{aligned} l \in \mathbb{R}^K &= V h^J + b^J \\ q \in \mathbb{R}^K &= \mathsf{LS}(l) \\ o \in \mathbb{R} &= -\log q_{y^i} \end{aligned}$$

Overview

- Introduction
- Multi-layer feedforward networks
- Representing words as vectors (“word embeddings”)
- The dependency parsing problem
- Dependency parsing using a shift-reduce neural-network model

An Example: Part-of-Speech Tagging

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere .

- There are many possible tags in the position ??
  {NN, NNS, Vt, Vi, IN, DT, . . . }

- The task: model the distribution $p(t_j|t_1, \ldots, t_{j-1}, w_1 \ldots w_n)$
  where t_j is the j'th tag in the sequence, w_j is the j'th word

- The input to the neural network will be $\langle t_1 \ldots t_{j-1}, w_1 \ldots w_n, j\rangle$

One-Hot Encodings of Words, Tags etc.

- A dictionary D with size $s(D)$ maps each word w in the vocabulary to an integer $\text{Index}(w, D)$ in the range $1 \ldots s(D)$.

$$
\begin{aligned}
\text{Index}(\textit{the}, D) &= 1 \\
\text{Index}(\textit{dog}, D) &= 2 \\
\text{Index}(\textit{cat}, D) &= 3 \\
\text{Index}(\textit{saw}, D) &= 4 \\
&\ldots
\end{aligned}
$$

- For any word w, dictionary D, $\text{Onehot}(w, D)$ maps a word w to a "one-hot vector" $u = \text{Onehot}(w, D) \in \mathbb{R}^{s(D)}$. We have

$$
\begin{aligned}
u_j &= 1 \quad \text{for } j = \text{Index}(w, D) \\
u_j &= 0 \quad \text{otherwise}
\end{aligned}
$$

One-Hot Encodings of Words, Tags etc. (continued)

- A dictionary D with size $s(D)$ maps each word w in the vocabulary to an integer in the range $1 \ldots s(D)$.

$$
\begin{aligned}
\text{Index}(\textit{the}, D) &= 1 \\
\text{Index}(\textit{dog}, D) &= 2 \\
\text{Index}(\textit{cat}, D) &= 3 \\
&\ldots
\end{aligned}
$$

$$
\begin{aligned}
\text{Onehot}(\textit{the}, D) &= [1, 0, 0, \ldots] \\
\text{Onehot}(\textit{dog}, D) &= [0, 1, 0, \ldots] \\
\text{Onehot}(\textit{cat}, D) &= [0, 0, 1, \ldots] \\
&\ldots
\end{aligned}
$$

The Concatenation Operation

- Given column vectors $v^i \in \mathbb{R}^{d_i}$ for $i = 1 \ldots n$,

$$z \in \mathbb{R}^d = \text{Concat}(v^1, v^2, \ldots v^n)$$

  where $d = \sum_{i=1}^{n} d_i$
- z is a vector formed by concatenating the vectors $v^1 \ldots v^n$
- z is a column vector of dimension $\sum_i d_i$

The Concatenation Operation (continued)

- Given vectors $v^i \in \mathbb{R}^{d_i}$ for $i = 1 \dots n$,

$$z \in \mathbb{R}^d = \text{Concat}(v^1, v^2, \dots v^n)$$

  where $d = \sum_{i=1}^n d_i$
- The Jacobians:

$$\frac{\partial z}{\partial v^i} \in \mathbb{R}^{d \times d_i}$$

  have entries

$$\left[\frac{\partial z}{\partial v^i}\right]_{j,k} = 1$$

  if $j = k + \sum_{i' < i} d_{i'}$,

$$\left[\frac{\partial z}{\partial v^i}\right]_{j,k} = 0$$

  otherwise

A Single-Layer Computational Network for Tagging

Inputs: A training example $x^i = \langle t_1 \ldots t_{j-1}, w_1 \ldots w_n, j \rangle$, $y^i \in \mathcal{Y}$. A word dictionary D with size $s(D)$, a tag dictionary T with size $s(T)$. Parameters of a single-layer feedforward network.

Computational Graph:

$$
\begin{aligned}
t'_{-2} \in \mathbb{R}^{s(T)} &= \text{Onehot}(t_{j-2}, T) \\
t'_{-1} \in \mathbb{R}^{s(T)} &= \text{Onehot}(t_{j-1}, T) \\
w'_{-1} \in \mathbb{R}^{s(D)} &= \text{Onehot}(w_{j-1}, D) \\
w'_{0} \in \mathbb{R}^{s(D)} &= \text{Onehot}(w_{j}, D) \\
w'_{+1} \in \mathbb{R}^{s(D)} &= \text{Onehot}(w_{j+1}, D) \\
u \in \mathbb{R}^{2s(T)+3s(D)} &= \text{Concat}(t'_{-2}, t'_{-1}, w'_{-1}, w'_{0}, w'_{+1}) \\
z &= Wu + b, \quad h = g(z), \quad l = Vh + \gamma, \quad q = \text{LS}(l) \\
o &= q_{y^i}
\end{aligned}
$$

The Number of Parameters

$$
\begin{array}{rcl}
t'_{-2} \in \mathbb{R}^{s(T)} & = & \text{Onehot}(t_{j-2}, T) \\
\ldots & & \\
w'_{+1} \in \mathbb{R}^{s(D)} & = & \text{Onehot}(w_{j+1}, D) \\
u & = & \text{Concat}(t'_{-2}, t'_{-1}, w'_{-1}, w'_0, w'_{+1}) \\
z \in \mathbb{R}^m & = & Wu + b \\
\ldots & &
\end{array}
$$

- An example: $s(T) = 50$ (50 tags), $s(D) = 10,000$ (10,000 words), $m = 1000$ (1000 neurons in the single layer)
- Then

  $$W \in \mathbb{R}^{m \times (2s(T)+3s(D))}$$

  and $m = 1000$, $2s(T) + 3s(D) = 30,100$, so there are $m \times (2s(T) + 3s(D)) = 30,100,000$ parameters in the matrix W

An Example

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere .

$$
\begin{aligned}
t'_{-2} \in \mathbb{R}^{s(T)} &= \text{Onehot}(t_{j-2}, T) \\
t'_{-1} \in \mathbb{R}^{s(T)} &= \text{Onehot}(t_{j-1}, T) \\
w'_{-1} \in \mathbb{R}^{s(D)} &= \text{Onehot}(w_{j-1}, D) \\
w'_{0} \in \mathbb{R}^{s(D)} &= \text{Onehot}(w_{j}, D) \\
w'_{+1} \in \mathbb{R}^{s(D)} &= \text{Onehot}(w_{j+1}, D) \\
u &= \text{Concat}(t'_{-2}, t'_{-1}, w'_{-1}, w'_{0}, w'_{+1}) \\
\ldots &
\end{aligned}
$$

Embedding Matrices

- Given a word w, a word dictionary D we can map w to a one-hot representation

$$w' \in \mathbb{R}^{s(D)\times 1} = \text{Onehot}(w, D)$$

- Now assume we have an *embedding dictionary* $E \in \mathbb{R}^{e\times s(D)}$ where e is some integer. Typical values of e are $e = 100$ or $e = 200$
- We can now map the one-hot representation w' to

$$\underbrace{w''}_{e\times 1} = \underbrace{E}_{e\times s(D)} \underbrace{w'}_{s(D)\times 1} = E \times \text{Onehot}(w, D)$$

- Equivalently, a word w is mapped to a vector $E(:j) \in \mathbb{R}^e$ where $j = \text{Index}(w, D)$ is the integer that word w is mapped to, and $E(:j)$ is the j'th column in the matrix.

Embedding Matrices vs. One-hot Vectors

- One-hot representation:

$$w' \in \mathbb{R}^{s(D)\times 1} = \text{Onehot}(w, D)$$

  This representation is **high-dimensional, sparse**
- Embedding representation:

$$\underbrace{w''}_{e\times 1} = \underbrace{E}_{e\times s(D)} \underbrace{w'}_{s(D)\times 1} = E \times \text{Onehot}(w, D)$$

  This representation is **low-dimensional, dense**
- The embedding matrices can be learned using stochastic gradient descent and backpropagation (each entry of E is a new parameter in the model)
- Critically, embeddings allow *shared information* between words: e.g., words with similar meaning or syntax get mapped to “similar” embeddings

A Single-Layer Computational Network for Tagging

Inputs: A training example $x^i = \langle t_1 \dots t_{j-1}, w_1 \dots w_n, j \rangle$, $y^i \in \mathcal{Y}$. A word dictionary D with size $s(D)$, a tag dictionary T with size $s(T)$. A word embedding matrix $E \in \mathbb{R}^{e \times s(D)}$. A tag embedding matrix $A \in \mathbb{R}^{a \times s(D)}$. Parameters of a single-layer feedforward network.

Computational Graph:

$$
\begin{aligned}
t'_{-2} \in \mathbb{R}^a &= A \times \text{Onehot}(t_{j-2}, T) \\
t'_{-1} \in \mathbb{R}^a &= A \times \text{Onehot}(t_{j-1}, T) \\
w'_{-1} \in \mathbb{R}^e &= E \times \text{Onehot}(w_{j-1}, D) \\
w'_{0} \in \mathbb{R}^e &= E \times \text{Onehot}(w_{j}, D) \\
w'_{+1} \in \mathbb{R}^e &= E \times \text{Onehot}(w_{j+1}, D) \\
u \in \mathbb{R}^{2a+3e} &= \text{Concat}(t'_{-2}, t'_{-1}, w'_{-1}, w'_0, w'_{+1}) \\
z &= Wu + b, \quad h = g(z), \quad l = Vh + \gamma, \quad q = \text{LS}(l) \\
o &= q_{y^i}
\end{aligned}
$$

An Example

Hispaniola/NNP quickly/RB became/VB an/DT important/JJ base/?? from which Spain expanded its empire into the rest of the Western Hemisphere .

$$
\begin{aligned}
t'_{-2} \in \mathbb{R}^{a} &= A \times \text{Onehot}(t_{j-2}, T) \\
t'_{-1} \in \mathbb{R}^{a} &= A \times \text{Onehot}(t_{j-1}, T) \\
w'_{-1} \in \mathbb{R}^{e} &= E \times \text{Onehot}(w_{j-1}, D) \\
w'_{0} \in \mathbb{R}^{e} &= E \times \text{Onehot}(w_{j}, D) \\
w'_{+1} \in \mathbb{R}^{e} &= E \times \text{Onehot}(w_{j+1}, D) \\
u \in \mathbb{R}^{2a+3e} &= \text{Concat}(t'_{-2}, t'_{-1}, w'_{-1}, w'_{0}, w'_{+1})
\end{aligned}
$$

Calculating Jacobians

$$w'_0 \in \mathbb{R}^e \quad = \quad E \times \text{Onehot}(w, D)$$

Equivalently:

$$(w'_0)_j = \sum_k E_{j,k} \times \text{Onehot}_k(w, D)$$

- Need to calculate the Jacobian

$$\frac{\partial w'_0}{E}$$

This has entries

$$\left[\frac{\partial w'_0}{E}\right]_{j,(j',k)} \quad = \quad 1 \text{ if } j = j' \text{ and } \text{Onehot}_k(w, E) = 1, \; 0 \text{ otherwise}$$

An Additional Perspective

$$
\begin{aligned}
t'_{-2} \in \mathbb{R}^a &= \text{Onehot}(t_{j-2}, T) \\
\ldots & \\
w'_{+1} \in \mathbb{R}^e &= \text{Onehot}(w_{j+1}, D) \\
u &= \text{Concat}(t'_{-2} \ldots w'_{+1}) \\
z \in \mathbb{R}^m &= Wu + b
\end{aligned}
$$

$$
\begin{aligned}
t'_{-2} \in \mathbb{R}^a &= A \times \text{Onehot}(t_{j-2}, T) \\
\ldots & \\
w'_{+1} \in \mathbb{R}^e &= E \times \text{Onehot}(w_{j+1}, D) \\
\bar{u} &= \text{Concat}(t'_{-2} \ldots w'_{+1}) \\
\bar{z} \in \mathbb{R}^m &= \bar{W}\bar{u} + b
\end{aligned}
$$

- If we set

$$
\underbrace{W}_{m\times(2s(T)+3s(E))} = \underbrace{\bar{W}}_{m\times(2a+3e)} \times \underbrace{\text{Diag}(A, A, E, E, E)}_{(2a+3e)\times(2s(T)+3s(D))}
$$

then $Wu + b = \bar{W}\bar{u} + b$ hence $z = \bar{z}$

An Additional Perspective (continued)

- If we set

$$\underbrace{W}_{m\times(2s(T)+3s(E))} = \underbrace{\bar{W}}_{m\times(2a+3e)} \times \underbrace{\text{Diag}(A, A, E, E, E)}_{(2a+3e)\times(2s(T)+3s(D))}$$

  then $Wu + b = \bar{W}\bar{u} + b$ hence $z = \bar{z}$

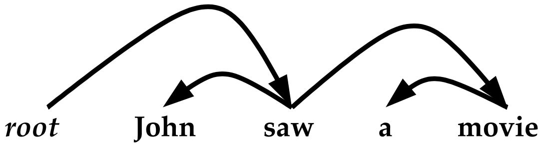
- An example: $s(T) = 50$ (50 tags), $s(D) = 10,000$ (10,000 words), $a = e = 100$ (recall a, e are size of embeddings for tags and words respectively), $m = 1000$ (1000 neurons)
- Then we have parameters

$$\underbrace{W}_{1000\times 30,100} \quad \text{vs.} \quad \underbrace{\bar{W}}_{1000\times 500} \quad \underbrace{A}_{100\times 50} \quad \underbrace{E}_{100\times 10,000}$$

Overview

- Introduction
- Multi-layer feedforward networks
- Representing words as vectors (“word embeddings”)
- The dependency parsing problem
- Dependency parsing using a shift-reduce neural-network model

Unlabeled Dependency Parses

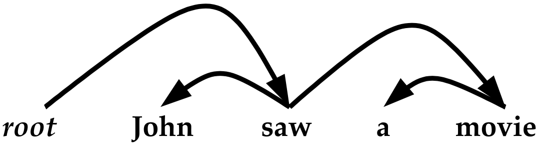


- `root` is a special *root* symbol
- Each dependency is a pair (h, m) where h is the index of a head word, m is the index of a modifier word. In the figures, we represent a dependency (h, m) by a directed edge from h to m.
- Dependencies in the above example are $(0, 2)$, $(2, 1)$, $(2, 4)$, and $(4, 3)$. (We take 0 to be the root symbol.)

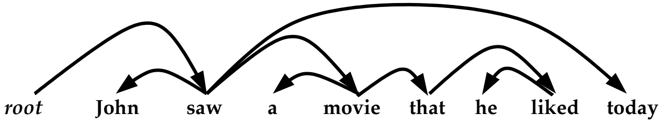
The (Unlabeled) Dependency Parsing Problem

John saw a movie

$\Downarrow$



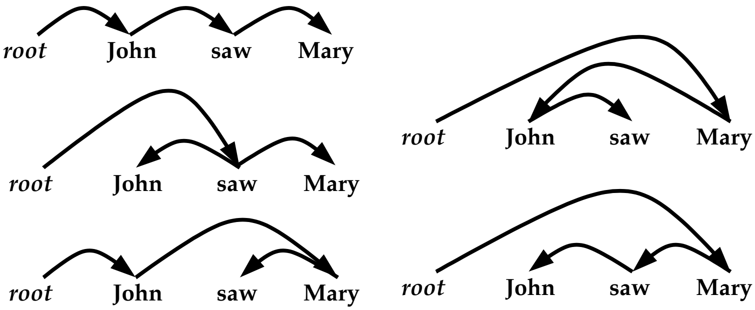
Conditions on Dependency Structures



- The dependency arcs form a *directed tree*, with the `root` symbol at the root of the tree.
  (Definition: A directed tree rooted at *root* is a tree, where for every word w other than the root, there is a directed path from *root* to w.)

- There are no "crossing dependencies".
  Dependency structures with no crossing dependencies are sometimes referred to as **projective** structures.

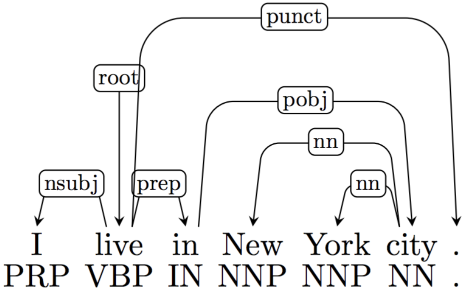
All Dependency Parses for *John saw Mary*



The Labeled Dependency Parsing Problem

I live in New York city .

$\Downarrow$



Overview

- Introduction
- Multi-layer feedforward networks
- Representing words as vectors ("word embeddings")
- The dependency parsing problem
- Dependency parsing using a shift-reduce neural-network model

Shift-Reduce Dependency Parsing: Configurations

- A configuration consists of:
  1. A *stack* σ consisting of a sequence of words, e.g.,

  $$\sigma = [\text{root}_0, \text{I}_1, \text{live}_2]$$

  2. A *buffer* β consisting of a sequence of words, e.g.,

  $$\beta = [\text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7]$$

  3. A set α of *labeled dependencies*, e.g.,

  $$\alpha = \{\{1 \to^{nsubj} 2\}, \{6 \to^{nn} 5\}$$

The Initial Configuration

$$\sigma = [\textsf{root}_0], \quad \beta = [\textsf{I}_1, \textsf{live}_2, \textsf{in}_3, \textsf{New}_4, \textsf{York}_5, \textsf{city}_6, \textsf{.}_7], \quad \alpha = \{\}$$

Shift-Reduce Actions: The Shift Action

The shift action takes the first word in the buffer, and adds it to the end of the stack.

$$\sigma = [\text{root}_0], \quad \beta = [\text{I}_1, \text{live}_2, \text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7], \quad \alpha = \{\}$$

SHIFT

$$\Downarrow$$

$$\sigma = [\text{root}_0, \text{I}_1], \quad \beta = [\text{live}_2, \text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7], \quad \alpha = \{\}$$

Shift-Reduce Actions: The Shift Action

The shift action takes the first word in the buffer, and adds it to the end of the stack.

$$\sigma = [\text{root}_0, \text{I}_1], \quad \beta = [\text{live}_2, \text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7], \quad \alpha = \{\}$$

$$\text{SHIFT}$$

$$\Downarrow$$

$$\sigma = [\text{root}_0, \text{I}_1, \text{live}_2], \quad \beta = [\text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7], \quad \alpha = \{\}$$

Shift-Reduce Actions: The Left-Arc Action

The LEFT-ARC^{nsubj} action takes the top two words on the stack, adds a dependency between them in the left direction with label *nsubj*, and removes the modifier word from the stack. There is a LEFT-ARC^{l} action for each possible dependency label l.

$$\sigma = [\text{root}_0, \text{I}_1, \text{live}_2], \quad \beta = [\text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7], \quad \alpha = \{\}$$

$$\text{LEFT-ARC}^{nsubj}$$

$$\Downarrow$$

$$\sigma = [\text{root}_0, \text{live}_2], \quad \beta = [\text{in}_3, \text{New}_4, \text{York}_5, \text{city}_6, ._7], \quad \alpha = \{\{2 \rightarrow^{nsubj} 1\}\}$$

Shift-Reduce Actions: The Right-Arc Action

The RIGHT-ARC^{prep} action takes the top two words on the stack, adds a dependency between them in the right direction with label *prep*, and removes the modifier word from the stack. There is a RIGHT-ARC^{l} action for each possible dependency label l.

$$\sigma = [\text{root}_0, \text{live}_2, \text{in}_3], \quad \beta = [._7], \quad \alpha = \{\{2 \rightarrow^{nsubj} 1\},\}$$

$$\text{RIGHT-ARC}^{prep}$$

$$\Downarrow$$

$$\sigma = [\text{root}_0, \text{live}_2], \quad \beta = [._7], \quad \alpha = \{\{2 \rightarrow^{nsubj} 1\}, \{2 \rightarrow^{prep} 3\}\}$$

Each Dependency Parse is Mapped to a Sequence of Actions

Action	σ	β	$h \xrightarrow{l} d$
Shift	[root$_0$]	[I$_1$,live$_2$, in$_3$, New$_4$, York$_5$, city$_6$, .$_7$]	
Shift	[root$_0$, I$_1$]	[live$_2$, in$_3$, New$_4$, York$_5$, city$_6$, .$_7$]	
Left-Arc^{nsubj}	[root$_0$, I$_1$, live$_2$]	[in$_3$, New$_4$, York$_5$, city$_6$, .$_7$]	$2 \xrightarrow{nsubj} 1$
Shift	[root$_0$, live$_2$]	[in$_3$, New$_4$, York$_5$, city$_6$, .$_7$]	
Shift	[root$_0$, live$_2$, in$_3$]	[New$_4$, York$_5$, city$_6$, .$_7$]	
Shift	[root$_0$, live$_2$, in$_3$, New$_4$]	[York$_5$, city$_6$, .$_7$]	
Shift	[root$_0$, live$_2$, in$_3$, New$_4$, York$_5$]	[city$_6$, .$_7$]	
Left-Arc^{nn}	[root$_0$, live$_2$, in$_3$, New$_4$, York$_5$, city$_6$]	[.$_7$]	$6 \xrightarrow{nn} 5$
Left-Arc^{nn}	[root$_0$, live$_2$, in$_3$, New$_4$, city$_6$]	[.$_7$]	$6 \xrightarrow{nn} 4$
Right-Arc^{pobj}	[root$_0$, live$_2$, in$_3$, city$_6$]	[.$_7$]	$3 \xrightarrow{pobj} 6$
Right-Arc^{prep}	[root$_0$, live$_2$, in$_3$]	[.$_7$]	$2 \xrightarrow{prep} 3$
Shift	[root$_0$, live$_2$]	[.$_7$]	
Right-Arc^{punct}	[root$_0$, live$_2$, .$_7$]	[]	$2 \xrightarrow{punct} 7$
Right-Arc^{root}	[root$_0$, live$_2$]	[]	$0 \xrightarrow{root} 2$
Terminal	[root$_0$]	[]	

Each Dependency Parse is Mapped to a Sequence of Actions

- Input $w_1 \ldots w_n$ = *I live in New York city .*
- Dependency parse requires actions $a_1 \ldots a_m$, e.g.,

$$\begin{aligned} a_1 \ldots a_m \quad = \quad & \langle \text{Shift}, \text{Shift}, \text{LEFT-ARC}^{nsubj}, \text{Shift}, \text{Shift}, \text{Shift}, \text{Shift}, \\ & \text{LEFT-ARC}^{nn}, \text{LEFT-ARC}^{nn}, \text{RIGHT-ARC}^{pobj}, \text{RIGHT-ARC}^{prep}, \\ & \text{Shift}, \text{RIGHT-ARC}^{punc}, \text{RIGHT-ARC}^{root} \rangle \end{aligned}$$

- We use a feedforward neural network to model

$$p(a_1 \ldots a_m | w_1 \ldots w_n) = \prod_{i=1}^{m} p(a_i | a_1 \ldots a_{i-1}, w_1 \ldots w_n)$$

Feature Extractors

- We use a feedforward neural network to model

$$p(a_1 \dots a_m | w_1 \dots w_n) = \prod_{i=1}^{m} p(a_i | a_1 \dots a_{i-1}, w_1 \dots w_n)$$

- Note that the action sequence $a_1 \dots a_{i-1}$ maps to a *configuration* $c_i = \langle \sigma_i, \beta_i, \alpha_i \rangle$
- A *feature extractor* maps a $(c_i, w_1 \dots w_n)$ pair to either a word, part-of-speech tag, or dependency label
- Weiss et al. 2015 (see also Chen and Manning 2014) have 20 word-based feature extractors, 20 tag-based feature extractors, 12 dependency label feature extractors
- This gives $20 + 20 + 12 = 52$ one-hot vectors as input to a neural network that estimates $p(a | c, w_1 \dots w_n)$

Word-Based Feature Extractors

- A *feature extractor* maps a $(c_i, w_1 \ldots w_n)$ pair to either a word, part-of-speech tag, or dependency label
- s_i for $i = 1 \ldots 4$ is the index of the i'th element on the stack. b_i for $i = 1 \ldots 4$ is the index of the i'th element on the buffer. $lc1(s_i)$ is the first left-child of word s_i, $lc2(s_i)$ is the second left-child. $rc1(s_i)$ and $rc2(s_i)$ are the first and second right-children of s_i.
- We then have features:

  word(s1) word(s2) word(s3) word(s4) word(b1) word(b2) word(b3)
  word(b4) word(lc1(s1)) word(lc1(s2)) word(lc2(s1)) word(lc2(s2))
  word(rc1(s1)) word(rc1(s2)) word(rc2(s1)) word(rc2(s2))
  word(lc1(lc1(s1)) word(lc1(lc1(s2)) word(rc1(rc1(s1)) word(rc1(rc1(s2))

Some Results

Method	Unlabeled Dep. Accuracy
Global linear model[1]	92.9%
Neural network, greedy[2]	93.0%
Neural network, beam[3]	93.6%
Neural network, beam, global training[4]	94.6%

1. Hand-constructed features very similar to features in log-linear models. Uses beam search in conjunction with a global linear model. *Transition-based Dependency Parsing with Rich Non-local Features, Zhang and Nivre 2011.*

2, 3: feedforward neural network with greedy search, or beam search. *Globally normalized transition-based neural networks. Andor et al., ACL 2016.* See also *A Fast and Accurate Dependency Parser using Neural Network Chen and Manning, ACL 2014.*

4: Neural network with global training, related to training of global linear models (but with word embeddings, and non-linearities from a neural network). See Andor et al. 2016.