CS 533: Natural Language Processing

Transition-Based Dependency Parsing

Karl Stratos

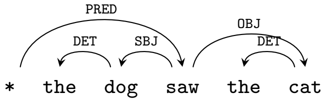
Rutgers University

Transition-Based Dependency Parsing

- Dependency parsing framed as a sequence of **transitions**

$$c_0 \xrightarrow{t_0} c_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{T-1}} c_T$$

$$\Downarrow$$



- Runtime **linear** in sentence length!
  - Major advantage over graph-based dependency parsing

Overview

Dependency Parsing

Transition-Based Framework

- Configuration
- Transitions

Transition Systems

- Arc-Standard
- Arc-Eager

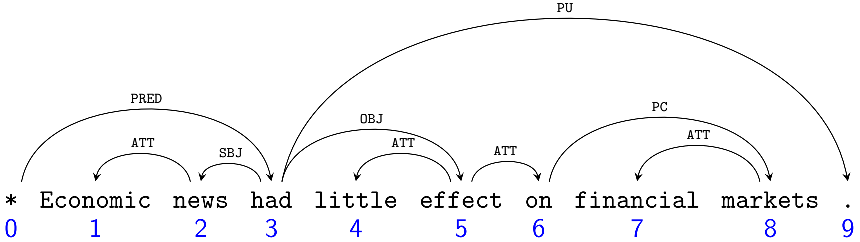
Implementation

- Training
- Greedy Parser
- Beam Search Parser
- Evaluation

Example Dependency Tree (Nivre 2013)

`Economic news had little effect on financial markets.`

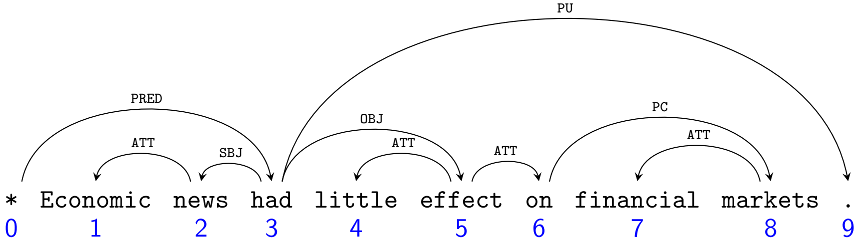
$\Downarrow$



Example Dependency Tree (Nivre 2013)

`Economic news had little effect on financial markets.`

$\Downarrow$



$$A = \{(0, \texttt{PRED}, 3), (3, \texttt{SBJ}, 2), (2, \texttt{ATT}, 1), (3, \texttt{OBJ}, 5),$$
$$(3, \texttt{PU}, 9), (5, \texttt{ATT}, 4), (5, \texttt{ATT}, 6), (6, \texttt{PC}, 8), (8, \texttt{ATT}, 7)\}$$

Dependency Parsing = Arc Finding

- Sentence: $x_1 \dots x_m$
- Associated nodes: $\mathcal{N} = \{0, 1, \dots, m\}$
  - Convention: leftmost root 0
- Labels: $L = \{\texttt{PRED}, \texttt{SBJ}, \dots\}$

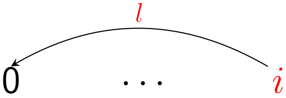
Goal. Find a set of **labeled, directed arcs**

$$A \subseteq \mathcal{N} \times L \times \mathcal{N}$$

that corresponds to a **correct dependency tree** for $x_1 \dots x_m$.

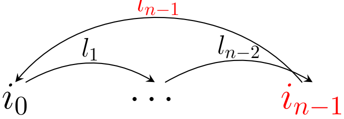
Valid Dependency Tree

1. (Root): 0 must not have a parent.



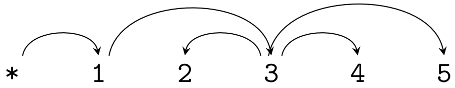
2. (Connected): There must be a path from 0 to every $i \in \mathcal{N}$.
3. (Tree): A node must not have more than one parent.

4. (Acyclic): Nodes must not form a cycle.

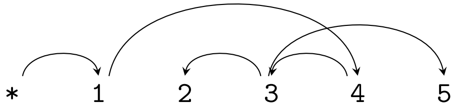


Projective

- A valid dependency tree is **projective** if for every arc (i, l, j) there is a path from i to k for all $i < k < j$.



- Valid but non-projective



We will focus on projective trees only!

Overview

Parser Configuration

Triple $c = (\sigma, \beta, A)$ where

- $\sigma = [\ldots\ i]$: "stack" of $\mathcal{N}$ with i at the top
- $\beta = [i\ \ldots]$: "buffer" of $\mathcal{N}$ with i at the front
- $A \subseteq \mathcal{N} \times L \times \mathcal{N}$: arcs

Notation

- $\mathcal{C}$ denotes the space of all possible configurations.
- $c.\sigma$, $c.\beta$, $c.A$ denote stack, buffer, arcs of $c \in \mathcal{C}$.

Configuration-Based Parsing Scheme

Initial configuration

$$c_0 := ([0], [1 \ldots m], \{\ \})$$

Apply "transitions" until we reach **terminal** c_T (defined later)

$$c_0 \xrightarrow{t_0} c_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{T-1}} c_T$$

and return as a parse

$$c_T.A$$

Overview

Dependency Parsing

Transition-Based Framework
- Configuration
- Transitions

Transition Systems
- Arc-Standard
- Arc-Eager

Implementation
- Training
- Greedy Parser
- Beam Search Parser
- Evaluation

Shift and Reduce

SHIFT $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$

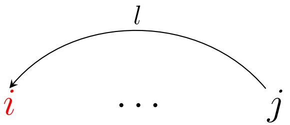
Illegal if β is empty.

REDUCE $(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)$

Illegal if i does not have a parent.

Left-Arc

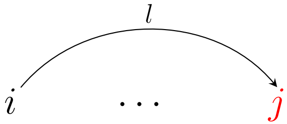
$$\textbf{LEFT}_l \quad (\sigma|i|j, \beta, A) \Rightarrow (\sigma|j, \beta, A \cup \{(j, l, i)\})$$



Illegal if either $i = 0$ or i already has a parent.

Right-Arc

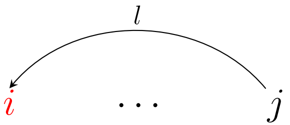
$$\textbf{RIGHT}_l \quad (\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i, l, j)\})$$



Illegal if j already has a parent.

“Eager” Left-Arc

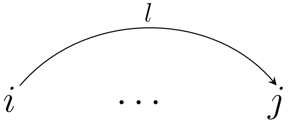
$$\textbf{LEFT}_l^e \quad (\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, l, i)\})$$



Illegal if either $i = 0$ or i already has a parent.

"Eager" Right-Arc

$$\textbf{RIGHT}^{e}_{l} \quad (\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i, l, j)\})$$



Illegal if j already has a parent.

Legal transitions

- Certain transitions are illegal depending on $c \in \mathcal{C}$.

- We will denote the **set of legal actions at** c by $\mathrm{LEGAL}(c)$.

Overview

Definition

$2\,|L| + 1$ possible transitions $\mathcal{T}^{\text{std}}$

- **SHIFT**: $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$
- **LEFT**$_l$ for each $l \in L$:

$$(\sigma|i|j, \beta, A) \Rightarrow (\sigma|j, \beta, A \cup \{(j, l, i)\})$$

- **RIGHT**$_l$ for each $l \in L$:

$$(\sigma|i|j, \beta, A) \Rightarrow (\sigma|i, \beta, A \cup \{(i, l, j)\})$$

Terminal condition: $c.\sigma = [0]$ and $c.\beta = [\,]$

Properties

- Makes **exactly** $2m$ transitions to parse $x_1 \dots x_m$. Why?
- **Bottom-up**: a node must collect all its children before getting a parent. Why?
- **Sound**: if c is terminal, $c.A$ forms a valid projective tree.
- **Complete**: every valid projective tree A can be produced from c_0 by some sequence of transitions $t_0 \dots t_{T-1} \in \mathcal{T}^{\text{std}}$.

$$\begin{aligned} t_i &= \text{Oracle}^{\text{std}}(c_i) \\ c_{i+1} &= t_i(c_i) \end{aligned}$$

Oracle$^{\text{std}}$

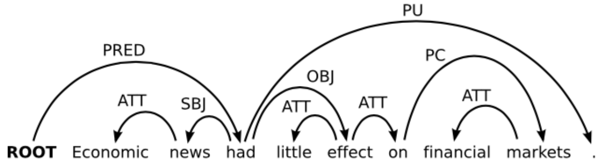
Input: gold arcs A^{gold}, non-terminal configuration $c = (\sigma, \beta, A)$
Output: transition $t \in \mathcal{T}^{\text{std}}$ to apply on c

1. Return **SHIFT** if $|\sigma| = 1$.
2. Otherwise $\sigma = [\ldots\ i\ j]$ for some $i < j$:
   2.1 Return **LEFT**$_l$ if $(j, l, i) \in A^{\text{gold}}$.
   2.2 Return **RIGHT**$_l$ if $(i, l, j) \in A^{\text{gold}}$ and for all $l' \in L, j' \in \mathcal{N}$,

   $$(j, l', j') \in A^{\text{gold}} \qquad \Rightarrow \qquad (j, l', j') \in A$$

   2.3 Return **SHIFT** otherwise.

Example Parse (Nivre 2013)



Transition	Configuration			
$c_s(x) =$	($[0]$,	$[1, \ldots, 9]$,	$\emptyset$	)
$\textsc{Shift} \Longrightarrow$	($[0, 1]$,	$[2, \ldots, 9]$,	$\emptyset$	)
$\textsc{Shift} \Longrightarrow$	($[0, 1, 2]$,	$[3, \ldots, 9]$,	$\emptyset$	)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	($[0, 2]$,	$[3, \ldots, 9]$,	$A_1 = \{(2, \textsc{att}, 1)\}$	)
$\textsc{Shift} \Longrightarrow$	($[0, 2, 3]$,	$[4, \ldots, 9]$,	A_1	)
$\textsc{Left-Arc}_{\textsc{sbj}} \Longrightarrow$	($[0, 3]$,	$[4, \ldots, 9]$,	$A_2 = A_1 \cup \{(3, \textsc{sbj}, 2)\}$	)
$\textsc{Shift} \Longrightarrow$	($[0, 3, 4]$,	$[5, \ldots, 9]$,	A_2	)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots, 5]$,	$[6, \ldots, 9]$,	A_2	)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	($[0, 3, 5]$,	$[6, \ldots, 9]$,	$A_3 = A_2 \cup \{(5, \textsc{att}, 4)\}$	)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots, 6]$,	$[7, 8, 9]$,	A_3	)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots 7]$,	$[8, 9]$,	A_3	)
$\textsc{Shift} \Longrightarrow$	($[0, \ldots, 8]$,	$[9]$,	A_3	)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	($[0, \ldots 8]$,	$[9]$,	$A_4 = A_3 \cup \{(8, \textsc{att}, 7)\}$	)
$\textsc{Right-Arc}_{\textsc{pc}} \Longrightarrow$	($[0, \ldots, 6]$,	$[9]$,	$A_5 = A_4 \cup \{(6, \textsc{pc}, 8)\}$	)
$\textsc{Right-Arc}_{\textsc{att}} \Longrightarrow$	($[0, 3, 5]$,	$[9]$,	$A_6 = A_5 \cup \{(5, \textsc{att}, 6)\}$	)
$\textsc{Right-Arc}_{\textsc{obj}} \Longrightarrow$	($[0, 3]$,	$[9]$,	$A_7 = A_6 \cup \{(3, \textsc{obj}, 5)\}$	)
$\textsc{Shift} \Longrightarrow$	($[0, 3, 9]$,	$[\,]$,	A_7	)
$\textsc{Right-Arc}_{\textsc{pu}} \Longrightarrow$	($[0, 3]$,	$[\,]$,	$A_8 = A_7 \cup \{(3, \textsc{pu}, 9)\}$	)
$\textsc{Right-Arc}_{\textsc{root}} \Longrightarrow$	($[0]$,	$[\,]$,	$A_9 = A_8 \cup \{(0, \textsc{root}, 3)\}$	)

Overview

Dependency Parsing

Transition-Based Framework
- Configuration
- Transitions

Transition Systems
- Arc-Standard
- Arc-Eager

Implementation
- Training
- Greedy Parser
- Beam Search Parser
- Evaluation

Definition

$2|L| + 2$ possible transitions $\mathcal{T}^{\text{eag}}$

SHIFT: $(\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)$

REDUCE: $(\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)$

LEFT$_l^e$ for each $l \in L$:

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, l, i)\})$$

RIGHT$_l^e$ for each $l \in L$:

$$(\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i, l, j)\})$$

Terminal condition: $c.\beta = [\,]$

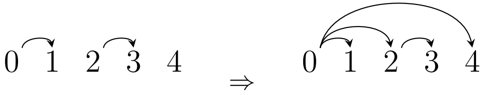
- Stop as soon as the buffer is empty.

Properties

- Makes **at most** $2m$ transitions to parse $x_1 \ldots x_m$. Why?

- **Partially top-down**: but a node must collect all its **left** children before right children. Why?

- **Not sound**: even if c is terminal, $c.A$ may form unconnected projective trees ("dependency forest").
  - But can be manually corrected by connecting to the root.



- **Complete**: every valid projective tree A can be produced from c_0 by some sequence of transitions $t_0 \ldots t_{T-1} \in \mathcal{T}^{\text{eag}}$.

$$t_i = \mathsf{Oracle}^{\text{eag}}(c_i)$$
$$c_{i+1} = t_i(c_i)$$

Oracle$^{\text{eag}}$

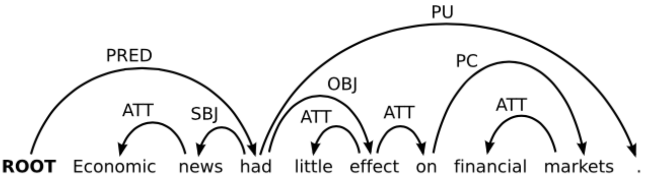
Input: gold arcs A^{gold}, non-terminal configuration

$$c = (\sigma = [\ldots\ i], \beta = [j\ \ldots], A)$$

Output: transition $t \in \mathcal{T}^{\text{eag}}$ to apply on c

1. Return $\textbf{LEFT}^{e}_{l}$ if $(j, l, i) \in A^{\text{gold}}$.
2. Return $\textbf{RIGHT}^{e}_{l}$ if $(i, l, j) \in A^{\text{gold}}$.
3. Return **REDUCE** if there is some $k < i$ such that $(k, l, j) \in A^{\text{gold}}$ or $(j, l, k) \in A^{\text{gold}}$ for some l.
4. Return **SHIFT** otherwise.

Example Parse (Nivre 2013)



Transition	Configuration			
	(	$[0]$,	$[1, \ldots, 9]$,	$\emptyset$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 1]$,	$[2, \ldots, 9]$,	$\emptyset$)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	(	$[0]$,	$[2, \ldots, 9]$,	$A_1 = \{(2, \textsc{att}, 1)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 2]$,	$[3, \ldots, 9]$,	A_1)
$\textsc{Left-Arc}_{\textsc{sbj}} \Longrightarrow$	(	$[0]$,	$[3, \ldots, 9]$,	$A_2 = A_1 \cup \{(3, \textsc{sbj}, 2)\}$)
$\textsc{Right-Arc}_{\textsc{root}} \Longrightarrow$	(	$[0, 3]$,	$[4, \ldots, 9]$,	$A_3 = A_2 \cup \{(0, \textsc{root}, 3)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0, 3, 4]$,	$[5, \ldots, 9]$,	A_3)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	(	$[0, 3]$,	$[5, \ldots, 9]$,	$A_4 = A_3 \cup \{(5, \textsc{att}, 4)\}$)
$\textsc{Right-Arc}_{\textsc{obj}} \Longrightarrow$	(	$[0, 3, 5]$,	$[6, \ldots, 9]$,	$A_5 = A_4 \cup \{(3, \textsc{obj}, 5)\}$)
$\textsc{Right-Arc}_{\textsc{att}} \Longrightarrow$	(	$[0, \ldots, 6]$,	$[7, 8, 9]$,	$A_6 = A_5 \cup \{(5, \textsc{att}, 6)\}$)
$\textsc{Shift} \Longrightarrow$	(	$[0, \ldots, 7]$,	$[8, 9]$,	A_6)
$\textsc{Left-Arc}_{\textsc{att}} \Longrightarrow$	(	$[0, \ldots 6]$,	$[8, 9]$,	$A_7 = A_6 \cup \{(8, \textsc{att}, 7)\}$)
$\textsc{Right-Arc}_{\textsc{pc}} \Longrightarrow$	(	$[0, \ldots, 8]$,	$[9]$,	$A_8 = A_7 \cup \{(6, \textsc{pc}, 8)\}$)
$\textsc{Reduce} \Longrightarrow$	(	$[0, \ldots, 6]$,	$[9]$,	A_8)
$\textsc{Reduce} \Longrightarrow$	(	$[0, 3, 5]$,	$[9]$,	A_8)
$\textsc{Reduce} \Longrightarrow$	(	$[0, 3]$,	$[9]$,	A_8)
$\textsc{Right-Arc}_{\textsc{pu}} \Longrightarrow$	(	$[0, 3, 9]$,	$[]$,	$A_9 = A_8 \cup \{(3, \textsc{pu}, 9)\}$)

Overview

Getting Training Data

- **Treebank**: sentence-tree pairs $(x^{(1)}, A^{(1)}) \ldots (x^{(M)}, A^{(M)})$
  - Assume all projective

- For each $A^{(j)}$, use an oracle to extract

$$(c_0^{(j)}, t_0^{(j)}) \ldots (c_{T-1}^{(j)}, t_{T-1}^{(j)})$$

where $t_{T-1}^{(j)}(c_{T-1}^{(j)}).A = A^{(j)}$.

- We can now use this to train a **classifier**

$$(x^{(j)}, c_i^{(j)}) \mapsto t_i^{(j)}$$

Linear Classifier

- Parameters: $w_t \in \mathbb{R}^d$ for each $t \in \mathcal{T}$
- Each $c \in \mathcal{C}$ for sentence x is "featurized" as $\phi^x(c) \in \mathbb{R}^d$.
  - Classical approach: **binary features** providing useful signals
  - Assumes we have access to POS tags of $x_1 \ldots x_m$.

$$\phi^x_{20134}(c) := \begin{cases} 1 & \text{if } x_{c.\sigma[0]}.\text{POS} = \texttt{NN} \text{ and } x_{c.\beta[0]}.\text{POS} = \texttt{VBD} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi^x_{1988}(c) := \begin{cases} 1 & \text{if } x_{c.\sigma[0]}.\text{POS} = \texttt{VBD} \text{ with leftmost arc } \texttt{SUBJ} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi^x_{42}(c) := \begin{cases} 1 & \text{if } x_{c.\beta[1]} = \texttt{cat} \\ 0 & \text{otherwise} \end{cases}$$

Linear Classifier (Continued)

- **Score** of $t \in \mathcal{T}$ at $c \in \mathcal{C}$ for x:

$$\mathsf{score}_x(t|c) := w_t \cdot \phi^x(c) = \sum_{i=1:\ \phi_i^x(c)=1}^{d} [w_t]_i$$

- From here on, we assume $\{w_t\}_{t \in \mathcal{T}}$ trained from data.

Important Aside

Each c_i is computed from **past decisions** $t_0 \ldots t_{i-1}$.

$$c_i = t_{i-1}(t_{i-2}(\cdots t_0(c_0)))$$

So the score function on c_i is really a **function of** $t_0 \ldots t_{i-1}$.

$$\text{score}_x(t|c) = \text{score}_x(t|t_1 \ldots t_{i-1})$$

Will use c_i and $t_0 \ldots t_{i-1}$ interchangeably.

Overview

Dependency Parsing

Transition-Based Framework
- Configuration
- Transitions

Transition Systems
- Arc-Standard
- Arc-Eager

Implementation
- Training
- Greedy Parser
- Beam Search Parser
- Evaluation

Greedy

At each configuration c_i, pick

$$t_i \leftarrow \underset{t \in \ \texttt{LEGAL}(c_i)}{\arg\max} \ \ \textsf{score}_x(t | t_0 \ldots t_{i-1})$$

Parsing Algorithm

Input: $\{w_t\}_{t \in \mathcal{T}}$, sentence x of length m
Output: arcs representing a dependency tree for x

1. $c \leftarrow c_0$
2. While $c.\beta \neq [\,]$,

    2.1 Select

    $$\hat{t} \leftarrow \operatorname*{arg\,max}_{t \in \texttt{LEGAL}(c)} \ \mathsf{score}_x(t|c)$$

    2.2 Make a transition: $c \leftarrow \hat{t}(c)$.
3. Return $c.A$.

Overview

Dependency Parsing

Transition-Based Framework
- Configuration
- Transitions

Transition Systems
- Arc-Standard
- Arc-Eager

Implementation
- Training
- Greedy Parser
- Beam Search Parser
- Evaluation

Beam Search

Approximate the **optimal sequence of transitions**:

$$t_0^* \dots t_{T-1}^* = \mathop{\arg\max}_{\substack{t_0 \dots t_{T-1}: \\ t_i \in \texttt{LEGAL}(c_i) \\ c_T.\beta = [\,]}} \sum_{i=0}^{T-1} \mathsf{score}_x(t_i | t_0 \dots t_{i-1})$$

Parsing Algorithm

Input: $\{w_t\}_{t \in \mathcal{T}}$, sentence x of length m, beam width K
Beam: $\langle c, s \rangle \in \mathcal{C} \times \mathbb{R}$ organized by second argument (score)
Output: arcs representing a dependency tree for x

1. $\mathcal{B} \leftarrow \texttt{Beam}(\{\langle c_0, 0 \rangle\}, K)$
2. While $c.\beta \neq [\,]$ for some $\langle c, s \rangle \in \mathcal{B}$,
   2.1 $\mathcal{B}' \leftarrow \texttt{Beam}(\{\,\}, K)$

   2.2 For $\langle c, s \rangle \in \mathcal{B}$, for $t \in \texttt{LEGAL}(c)$,

   $$\mathcal{B}'.\text{push}\langle t(c), s + \text{score}_x(t|c) \rangle$$

   2.3 $\mathcal{B} \leftarrow \mathcal{B}'$
3. Return $c^*.A$ where $c^* \leftarrow \mathcal{B}.\text{pop}()$.

Overview

UAS/LAS

- Unlabeled Attachment Score (UAS):

$$\frac{\#\ \text{words with correct parent}}{\#\ \text{words}}$$

- Labeled Attachment Score (LAS):

$$\frac{\#\ \text{words with correct parent and label}}{\#\ \text{words}}$$

Current state-of-the-art: 93-95 UAS, 91-93 LAS!

Parting Remarks

- There are better ways to train model $\{w_t\}_{t \in \mathcal{T}}$.
  - Online learning, "dynamic" oracles, etc.

- Today, state-of-the-art parsers are obtained by just replacing

$$\text{score}_x(t|c) = \overbrace{w_t}^{\text{linear}} \cdot \underbrace{\phi^x(c)}_{\text{hand-engineered}}$$

  with a neural network (Kiperwasser and Goldberg, 2016).

- **Graph-based** dependency parsing (Eisner, 1996): similar to CKY in constituency parsing