Natural Language Processing

Tutorial 3: Log-Linear Models for Tagging (MEMM)

Spring 2025

Eilam Shapira & Shani Pais

Based on slides by Nadav Oved, Ira Leviant, Yftah Ziser, Tomer Volk

Logistics

- HW1 - Tuesday 15:30!
- Machines should be available soon

Last Week

סיבוכיות של HMM בלי Viterbi
היא אקספוננציאלית ועכשיו -לו בזמן סביר

- POS tagging
- HMM
- Viterbi
- Beam search

- This week – MEMM

Agenda:

- POS Tagging Recap
- Log Linear Models
- Log Linear POS Tagger
- Constructing features for Log Linear POS Tagger
- Training Log Linear Models
- Inference for Log Linear Models with Viterbi Algorithm

Part-of-Speech Tagging

INPUT:

Profits soared at Boeing Co., easily topping forecasts on Wall Street, as their CEO Alan Mulally announced first quarter results.

OUTPUT:

Profits/N soared/V at/P Boeing/N Co./N ,/, easily/ADV topping/V forecasts/N on/P Wall/N Street/N ,/, as/P their/POSS CEO/N Alan/N Mulally/N announced/V first/ADJ quarter/N results/N ./.

N = Noun
V = Verb
P = Preposition
Adv = Adverb
Adj = Adjective
...

Recall - Trigram Hidden Markov Models

$$p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i | y_i)$$

Parameters of the model:

- $q(s|u, v)$ for any $s \in \mathcal{S} \cup \{\text{STOP}\}$, $u, v \in \mathcal{S} \cup \{*\}$
- $e(x|s)$ for any $s \in \mathcal{S}$, $x \in \mathcal{V}$

Recall: How do we apply a generative model to a new test example?

$$f(x) = \arg\max_y p(y \mid x) = \arg\max_y \frac{p(\mathrm{x}, \mathrm{y})}{p(x)} = \arg\max_y \frac{p(\mathrm{y})\, p(\mathrm{x} \mid \mathrm{y})}{p(x)} = \arg\max_y p(\mathrm{y})\, p(\mathrm{x} \mid \mathrm{y})$$

HMM is a generative model:

The **q** parameters are the prior probability of the tags, i.e. **p(y)** (prior).
The **e** parameters are the conditional probabilities, i.e. **p(x|y)** (likelihood).

Log-Linear Models for Tagging

- We have an input sentence $w_{[1:n]} = w_1, w_2, \ldots, w_n$ (w_i is the i'th word in the sentence)

- We have a tag sequence $t_{[1:n]} = t_1, t_2, \ldots, t_n$ (t_i is the i'th tag in the sentence)

- We'll use an log-linear model to define

  $$p(t_1, t_2, \ldots, t_n | w_1, w_2, \ldots, w_n)$$

  for any sentence $w_{[1:n]}$ and tag sequence $t_{[1:n]}$ of the same length. (Note: contrast with HMM that defines $p(t_1 \ldots t_n, w_1 \ldots w_n)$)

- Then the most likely tag sequence for $w_{[1:n]}$ is

  $$t^*_{[1:n]} = \text{argmax}_{t_{[1:n]}} p(t_{[1:n]} | w_{[1:n]})$$

Trigram Log Linear Tagger - Definition

A Trigram Log-Linear Tagger:

$$p(t_{[1:n]}|w_{[1:n]}) = \prod_{j=1}^{n} p(t_j \mid w_1 \ldots w_n, t_1 \ldots t_{j-1})$$ Chain rule

$$= \prod_{j=1}^{n} p(t_j \mid w_1, \ldots, w_n, t_{j-2}, t_{j-1})$$

Trigram Independence assumptions

- We take $t_0 = t_{-1} = *$
- Independence assumption: each tag only depends on previous two tags

$$p(t_j|w_1, \ldots, w_n, t_1, \ldots, t_{j-1}) = p(t_j|w_1, \ldots, w_n, t_{j-2}, t_{j-1})$$

Log-Linear Models – Formal Definition

- We have some input domain $\mathcal{X}$, and a finite label set $\mathcal{Y}$. Aim is to provide a conditional probability $p(y \mid x)$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- A feature is a function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$
  (Often binary features or indicator functions
  $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$).
- Say we have m features f_k for $k = 1 \ldots m$
  $\Rightarrow$ A feature vector $f(x, y) \in \mathbb{R}^m$ for any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.
- We also have a **parameter vector** $v \in \mathbb{R}^m$
- We define

$$\boxed{p(y \mid x; v) = \frac{e^{v \cdot f(x,y)}}{\sum_{y' \in \mathcal{Y}} e^{v \cdot f(x,y')}}}$$

History Tuple - 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 .

- $t_{-2}, t_{-1} =$ DT, JJ
- $w_{[1:n]} = \langle Hispaniola, quickly, became, \ldots, Hemisphere, . \rangle$
- $i = 6$

- A **history** is a 4-tuple $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$
- t_{-2}, t_{-1} are the previous two tags.
- $w_{[1:n]}$ are the n words in the input sentence.
- i is the index of the word being tagged

Feature Functions - Example

- $\mathcal{X}$ is the set of all possible histories of form $\langle t_{-2}, t_{-1}, w_{[1:n]}, i \rangle$
- $\mathcal{Y} = \{$NN, NNS, Vt, Vi, IN, DT, $\ldots\}$
- We have m features $f_k : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ for $k = 1 \ldots m$

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

$$
\begin{aligned}
f_1(h, t) &= \begin{cases} 1 & \text{if current word } w_i \text{ is base and } t = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases} \\
f_2(h, t) &= \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \texttt{ing} \text{ and } t = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases} \\
&\ldots
\end{aligned}
$$

$f_1(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle\text{, } 6 \rangle, \text{Vt}) = 1$
$f_2(\langle \text{JJ, DT, } \langle \text{ Hispaniola, } \ldots \rangle\text{, } 6 \rangle, \text{Vt}) = 0$

The Full Set of Features in [(Ratnaparkhi, 96)]

- Word/tag features for all word/tag pairs, e.g.,

$$f_{100}(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ is base and } t = \texttt{Vt} \\ 0 & \text{otherwise} \end{cases}$$

- Spelling features for all prefixes/suffixes of length ≤ 4, e.g.,

$$f_{101}(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \texttt{ing} \text{ and } t = \texttt{VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{102}(h,t) = \begin{cases} 1 & \text{if current word } w_i \text{ starts with } \texttt{pre} \text{ and } t = \texttt{NN} \\ 0 & \text{otherwise} \end{cases}$$

The Full Set of Features in [(Ratnaparkhi, 96)]

- Contextual Features, e.g.,

$$f_{103}(h,t) = \begin{cases} 1 & \text{if } \langle t_{-2}, t_{-1}, t\rangle = \langle \text{DT, JJ, Vt}\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{104}(h,t) = \begin{cases} 1 & \text{if } \langle t_{-1}, t\rangle = \langle \text{JJ, Vt}\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{105}(h,t) = \begin{cases} 1 & \text{if } \langle t\rangle = \langle \text{Vt}\rangle \\ 0 & \text{otherwise} \end{cases}$$

$$f_{106}(h,t) = \begin{cases} 1 & \text{if previous word } w_{i-1} = \textit{the} \text{ and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{107}(h,t) = \begin{cases} 1 & \text{if next word } w_{i+1} = \textit{the} \text{ and } t = \text{Vt} \\ 0 & \text{otherwise} \end{cases}$$

Log-Linear Models - MEMM

$$p(y \mid x; v) = \frac{e^{v \cdot f(x,y)}}{\sum_{y' \in \mathcal{Y}} e^{v \cdot f(x,y')}}$$

$$\log p(y \mid x; v) = \underbrace{v \cdot f(x, y)}_{\text{Linear term}} - \underbrace{\log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x,y')}}_{\text{Normalization term}}$$

<u>Define (log) likelihood function:</u>

$$L(v) = \sum_{i=1}^{n} \log p(y^{(i)} \mid x^{(i)}; v) = \sum_{i=1}^{n} v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)},y')}$$

Log-Linear Models - MEMM

- Need to maximize:

$$L(v) = \sum_{i=1}^{n} v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')}$$

- Calculating gradients:

$$\begin{aligned}
\frac{dL(v)}{dv_k} &= \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \frac{\sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') e^{v \cdot f(x^{(i)}, y')}}{\sum_{z' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, z')}} \\
&= \sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') \frac{e^{v \cdot f(x^{(i)}, y')}}{\sum_{z' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, z')}} \\
&= \underbrace{\sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)})}_{\text{Empirical counts}} - \underbrace{\sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' \mid x^{(i)}; v)}_{\text{Expected counts}}
\end{aligned}$$

Log-Linear Model – L2 Regularization

- Modified loss function

$$L(v) = \sum_{i=1}^{n} v \cdot f(x^{(i)}, y^{(i)}) - \sum_{i=1}^{n} \log \sum_{y' \in \mathcal{Y}} e^{v \cdot f(x^{(i)}, y')} - \frac{\lambda}{2} \sum_{k=1}^{m} v_k^2$$

- Calculating gradients:

$$\frac{dL(v)}{dv_k} = \underbrace{\sum_{i=1}^{n} f_k(x^{(i)}, y^{(i)})}_{\text{Empirical counts}} - \underbrace{\sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} f_k(x^{(i)}, y') p(y' \mid x^{(i)}; v)}_{\text{Expected counts}} - \lambda v_k$$

- Adds a penalty for large weights

Training the Log-Linear Model - MEMM

- To train a log-linear model, we need a training set (x_i, y_i) for $i = 1 \ldots n$. Then search for

$$v^* = \text{argmax}_v \left(\underbrace{\sum_i \log p(y_i | x_i; v)}_{Log-Likelihood} - \underbrace{\frac{\lambda}{2} \sum_k v_k^2}_{Regularizer} \right)$$

Training the Log-Linear Model - Gradient Ascent

Initialization: $v = 0$ (in practice, small random values round 0)

Iterate until convergence:

Gradient ▸ Calculate $\Delta = \frac{dL(v)}{dv}$

Step Size ▸ Calculate $\beta_* = \text{argmax}_\beta L(v + \beta\Delta)$ (Line Search)

Update ▸ Set $v \leftarrow v + \beta_*\Delta$

Convergence (many options for stop condition, depends on implementation)

- Set a fixed maximum number of iterations *T*
- Stop if $||\Delta_t|| < \varepsilon$ (small norm of gradient)
- Stop if $|\beta_t| < \varepsilon$ or if $||\beta_t\Delta_t|| < \varepsilon$ (small step size)
- Stop if $||v_{t-1}-v_t|| < \varepsilon$ or if $||\Delta_{t-1}-\Delta_t|| < \varepsilon$ (small change between iterations)

Gradient Descent:

Equivalent to Gradient Ascent but in opposite direction ($\min\{f(x)\} = \max\{-f(x)\}$)

$$\beta_* = \text{argmin}_\beta\{L(v-\beta\Delta)\}$$
$$v = v - \beta_*\Delta$$

Inference - MEMM

Problem: for an input $w_1 \ldots w_n$, find

$$\arg\max_{t_1 \ldots t_n} p(t_1 \ldots t_n \mid w_1 \ldots w_n)$$

We assume that p takes the form

$$p(t_1 \ldots t_n \mid w_1 \ldots w_n) = \prod_{i=1}^{n} q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

(In our case $q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$ is the estimate from a log-linear model.)

The Viterbi Algorithm - MEMM

- Define n to be the length of the sentence
- Define

$$r(t_1 \ldots t_k) = \prod_{i=1}^{k} q(t_i | t_{i-2}, t_{i-1}, w_{[1:n]}, i)$$

- Define a dynamic programming table

$\pi(k, u, v) \quad = \quad$ maximum probability of a tag sequence ending in tags u, v at position k

that is,

$$\pi(k, u, v) = \max_{\langle t_1, \ldots, t_{k-2} \rangle} r(t_1 \ldots t_{k-2}, u, v)$$

A Recursive Definition - MEMM

Base case:

$$\pi(0, *, *) = 1$$

Recursive definition:

For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{S}_{k-1}$ and $v \in \mathcal{S}_k$:

$$\pi(k, u, v) = \max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k)\right)$$

where $\mathcal{S}_k$ is the set of possible tags at position k

The Viterbi Algorithm with Backpointers

Input: a sentence $w_1 \ldots w_n$, log-linear model that provides $q(v|t, u, w_{[1:n]}, i)$ for any tag-trigram t, u, v, for any $i \in \{1 \ldots n\}$
Initialization: Set $\pi(0, *, *) = 1$.
Algorithm:

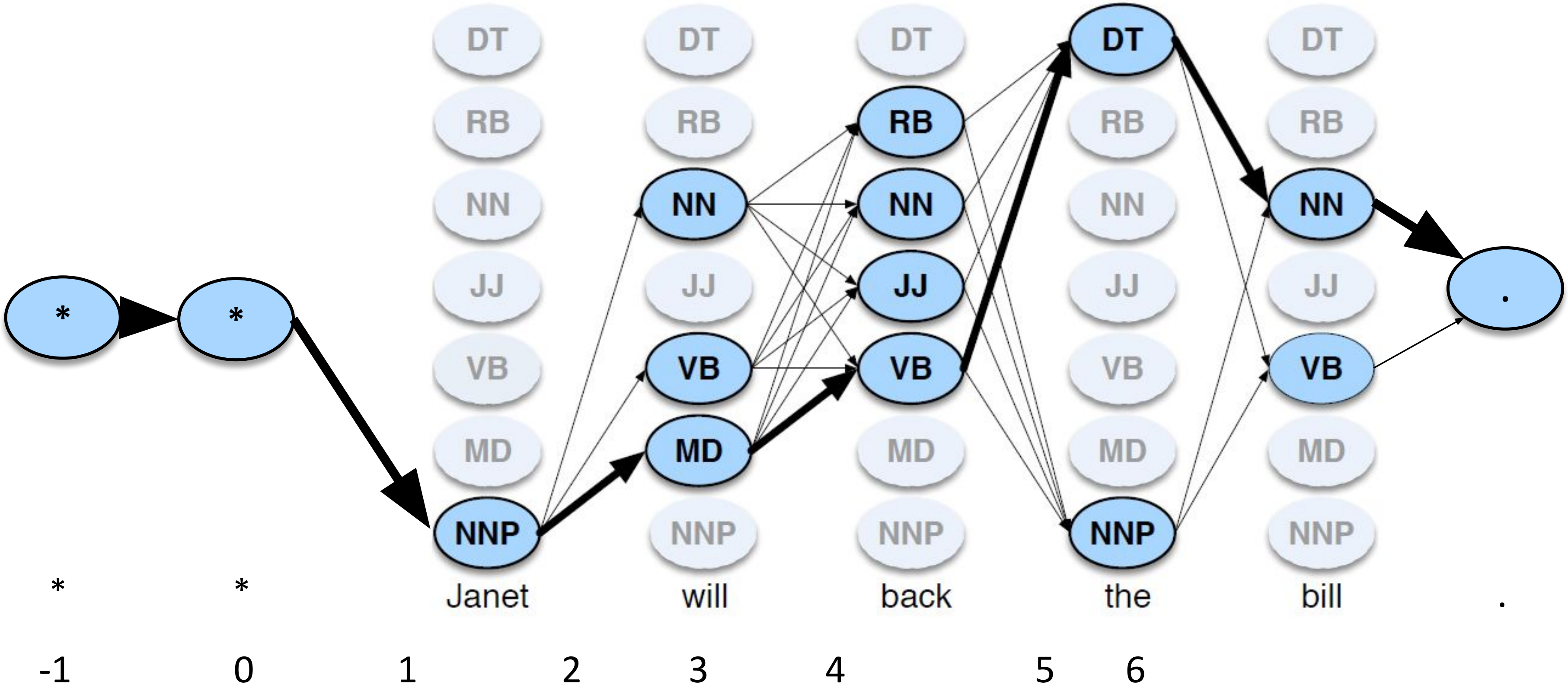
- For $k = 1 \ldots n$,
  - For $u \in \mathcal{S}_{k-1}$, $v \in \mathcal{S}_k$,

$$\begin{aligned} \pi(k, u, v) &= \max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k)\right) \\ bp(k, u, v) &= \arg\max_{t \in \mathcal{S}_{k-2}} \left(\pi(k-1, t, u) \times q(v|t, u, w_{[1:n]}, k)\right) \end{aligned}$$

- Set $(t_{n-1}, t_n) = \arg\max_{(u,v)} \pi(n, u, v)$
- For $k = (n-2) \ldots 1$, $t_k = bp(k+2, t_{k+1}, t_{k+2})$
- **Return** the tag sequence $t_1 \ldots t_n$

The Viterbi Algorithm - Example

$$\pi(k,u,v) = \max_{t \in S_{k-2}} \{\pi(k-1,t,u) \cdot q(v|t,u,w_{[1:n]},k)\}, \qquad \pi(0,*,*) = 1$$



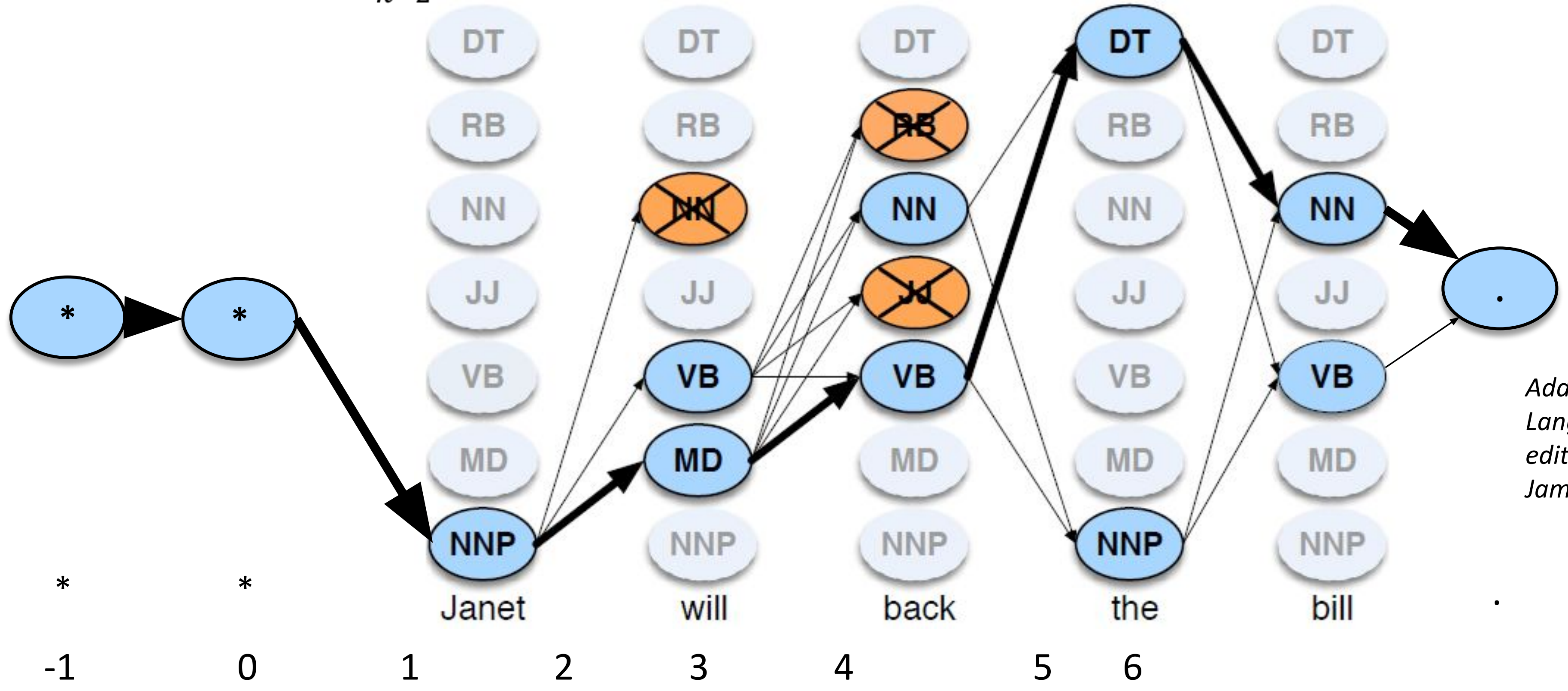
Adapted from "Speech and Language Processing" (3rd edition) by Dan Jurafsky and James H. Martin, page 154.

The Viterbi Algorithm + Beam Search

<u>Beam search (heuristic):</u>

At each time step (k), calculate scores for all possible states (π), sort them according to their score values and pass only the top B (beam width) possible states to the next time step.

$$\pi(k,u,v) = \max_{t \in S_{k-2}} \{\pi(k-1,t,u) \cdot q(v|t,u,w_{[1:n]},k)\}, \qquad \pi(0,*,*) = 1, \qquad B = 2$$



Adapted from "Speech and Language Processing" (3rd edition) by Dan Jurafsky and James H. Martin, page 158.

Evaluating POS Taggers

Ground Truth: The_DT boy_NNP walked_VBD

Our Tagger Predicted: The_DT boy_DT walked_NNP

The Confusion matrix:

Truth \ Predicted	***DT***	***NNP***	***VBD***
DT	1	0	0
NNP	1	0	0
VBD	0	1	0

Accuracy:

$$\frac{\#\ correct}{\#\ total} = \frac{sum(diagonal)}{sum(matrix)} = \frac{\sum_{i=1}^{|S|} a_{i,i}}{\sum_{i=1}^{|S|} \sum_{j=1}^{|S|} a_{i,j}} = \frac{1}{3}$$

Language is the Key