Tecnologie del Linguaggio Naturale

Parte Prima

# Lezione n. 03

# Part of Speech (PoS) Tagging

08 Marzo 2021

# PoS tagging

| | | | | |
|---|---|---|---|---|
| Input: | *Plays* | *well* | *with* | *others* |
| Ambiguity: | NNS/VBZ | UH/JJ/NN/RB | IN | NNS |
| Output: | Plays/VBZ | well/RB | with/IN | others/NNS |

il compito del PoS è rimuovere questa ambiguità

ha diversi PoS

## Uses:

- Text-to-speech (how do we pronounce "lead"?)
- Can write regexps like (Det) Adj* N+ over the output for phrases, etc.
- As input to or to speed up a full parser

- Moreover, if you know the tag, you can back off to it in other tasks

# PoS tagging performance

How many tags are correct? (**Tag accuracy**)

- About 97% currently
- But baseline is already 90%: Baseline is performance of stupidest possible method
  - Tag every word with its most frequent tag
  - Tag unknown words as nouns
- Partly easy because
  - Many words are unambiguous
  - You get points for them (the, a, etc.) and for punctuation marks!

# PoS tagging performance

How many tags are correct? (**Tag accuracy**)

- About 97% currently
- But baseline is already 90%: **Baseline** is performance of stupidest possible method
  - Tag every word with its most frequent tag
  - Tag unknown words as nouns
- Partly easy because
  - Many words are unambiguous
  - You get points for them (the, a, etc.) and for punctuation marks!

# How difficult is PoS tagging?

- About 11% of the word types in the Brown corpus are ambiguous with regard to part of speech
- But they tend to be very common words, e.g. *that*
  - *I know that he is honest* = IN
  - *Yes, that play was nice* = DT
  - *You can't go that far* = RB
- 40% of the word tokens are ambiguous

# How difficult is PoS tagging?

- About 11% of the word **types** in the Brown corpus are ambiguous with regard to part of speech
- But they tend to be very common words, e.g. that
  - *I know that he is honest* = IN
  - *Yes, that play was nice* = DT
  - *You can't go that far* = RB
- 40% of the word **tokens** are ambiguous

# Penn TreeBank POS Tagset

| Tag | Description | Example | Tag | Description | Example |
|---|---|---|---|---|---|
| CC | coordin. conjunction | *and, but, or* | SYM | symbol | +,%, & |
| CD | cardinal number | *one, two, three* | TO | "to" | *to* |
| DT | determiner | *a, the* | UH | interjection | *ah, oops* |
| EX | existential 'there' | *there* | VB | verb, base form | *eat* |
| FW | foreign word | *mea culpa* | VBD | verb, past tense | *ate* |
| IN | preposition/sub-conj | *of, in, by* | VBG | verb, gerund | *eating* |
| JJ | adjective | *yellow* | VBN | verb, past participle | *eaten* |
| JJR | adj., comparative | *bigger* | VBP | verb, non-3sg pres | *eat* |
| JJS | adj., superlative | *wildest* | VBZ | verb, 3sg pres | *eats* |
| LS | list item marker | *1, 2, One* | WDT | wh-determiner | *which, that* |
| MD | modal | *can, should* | WP | wh-pronoun | *what, who* |
| NN | noun, sing. or mass | *llama* | WP$ | possessive wh- | *whose* |
| NNS | noun, plural | *llamas* | WRB | wh-adverb | *how, where* |
| NNP | proper noun, singular | *IBM* | $ | dollar sign | *$* |
| NNPS | proper noun, plural | *Carolinas* | # | pound sign | # |
| PDT | predeterminer | *all, both* | " | left quote | ' or " |
| POS | possessive ending | *'s* | " | right quote | ' or " |
| PRP | personal pronoun | *I, you, he* | ( | left parenthesis | [, (, {, < |
| PRP$ | possessive pronoun | *your, one's* | ) | right parenthesis | ], ), }, > |
| RB | adverb | *quickly, never* | , | comma | , |
| RBR | adverb, comparative | *faster* | . | sentence-final punc | . ! ? |
| RBS | adverb, superlative | *fastest* | : | mid-sentence punc | : ; ... – - |
| RP | particle | *up, off* | | | |

# How Hard is POS Tagging? Ambiguity

| | | 87-tag Original Brown | | 45-tag Treebank Brown | |
|---|---|---|---|---|---|
| **Unambiguous (1 tag)** | | **44,019** | | **38,857** | |
| **Ambiguous (2–7 tags)** | | **5,490** | | **8844** | |
| Details: | 2 tags | 4,967 | | 6,731 | |
| | 3 tags | 411 | | 1621 | |
| | 4 tags | 91 | | 357 | |
| | 5 tags | 17 | | 90 | |
| | 6 tags | 2 | (*well, beat*) | 32 | |
| | 7 tags | 2 | (*still, down*) | 6 | (*well, set, round, open, fit, down*) |
| | 8 tags | | | 4 | (*'s, half, back, a*) |
| | 9 tags | | | 3 | (*that, more, in*) |

# Outline

- Rule-Based Tagger: ENGTWOL (ENGlish TWO Level analysis)
- Stochastic Tagger
  - HMM
  - MEMM
- PoS tagging conclusion

# Outline

- **Rule-Based Tagger: ENGTWOL (ENGlish TWO Level analysis)**
- Stochastic Tagger
  - HMM
  - MEMM
- PoS tagging conclusion

# 1 Rule-Based Tagging

## Basic Idea:

1. Assign all possible tags to words (morphological analysis!)
2. Remove tags according to set of rules of type. For example:

   **if** word+1 is an adj, adv, or quantifier and the following is a sentence boundary and word-1 is not a verb like "consider"

   - **then** eliminate non-adv
   - **else** eliminate adv.

- Typically more than 1000 hand-written rules, but it may be machine-learned.

NON rimuove totalmente le ambiguità, ma le forme non legali
↳ "la vecchia porta la sbarra" rimarrebbe con tutte e 2 le sue interpretazioni anche dopo l'algoritmo

# Sample ENGTWOL Lexicon

| Word | POS | Additional POS features |
|---|---|---|
| smaller | ADJ | COMPARATIVE |
| entire | ADJ | ABSOLUTE ATTRIBUTIVE |
| fast | ADV | SUPERLATIVE |
| that | DET | CENTRAL DEMONSTRATIVE SG |
| all | DET | PREDETERMINER SG/PL QUANTIFIER |
| dog's | N | GENITIVE SG |
| furniture | N | NOMINATIVE SG NOINDEFDETERMINER |
| one-third | NUM | SG |
| she | PRON | PERSONAL FEMININE NOMINATIVE SG3 |
| show | V | IMPERATIVE VFIN |
| show | V | PRESENT -SG3 VFIN |
| show | N | NOMINATIVE SG |
| shown | PCP2 | SVOO SVO SV |
| occurred | PCP2 | SV |
| occurred | V | PAST VFIN SV |

# Stage 1 of ENGTWOL Tagging

**Run words through a morphological analyzer to get all parts of speech.**

Example: *Pavlov had shown that salivation …*

| | |
|---|---|
| Pavlov | PAVLOV N NOM SG PROPER |
| had | HAVE V PAST VFIN SVO |
| | HAVE PCP2 SVO |
| shown | SHOW PCP2 SVOO SVO SV |
| that | ADV |
| | PRON DEM SG |
| | DET CENTRAL DEM SG |
| | CS |
| salivation | N NOM SG |

# Stage 2 of ENGTWOL Tagging

## Apply constraints in a negative way

Example: Adverbial “that” rule -> Given input: “that”

```
IF (and
    (+1 A/ADV/QUANT)          /* if next word is adj, adverb, or quantifier */
    (+2 SENT-LIM)             /* and following which is a sentence boundary, */
    (NOT -1 SVOC/A) )         /* and the previous word is not a verb like
                                 ‘consider’ which allows adjs as object
                                  complements */
THEN
     eliminate non-ADV tags
ELSE
     eliminate ADV
```

# Outline

- Rule-Based Tagger: ENGTWOL (ENGlish TWO Level analysis)
- **Stochastic Tagger**
  - **HMM**
  - MEMM
- PoS tagging conclusion

# POS Tagging as Sequence Labelling

- We are given a sentence (an "observation" or "sequence of observations")
  - *Secretariat is expected to race tomorrow*
- What is the best sequence of tags that corresponds to this sequence of observations?
- Probabilistic view
  - Consider all possible sequences of tags
  - Out of this universe of sequences, choose the tag sequence which is most probable given the observation sequence of n words $w_1 \ldots w_n$

# Terminology

1. **Modelling** -> *give a formal model*
2. **Learning** -> *an algorithm for setting the parameters of the model*
3. **Decoding** -> *algorithm for applying the model in order to compute results*

# 1a Modelling HMMs

- We want, out of all sequences of n tags $t_1 \ldots t_n$ the single tag sequence such that

$P(t_1 \ldots t_n | w_1 \ldots w_n)$ is highest.

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

- Hat ^ means "our estimate of the best one"
- $\operatorname{Argmax}_x f(x)$ means "the x such that f(x) is maximized"

# Modelling HMMs

- This equation should give us the best tag sequence

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

è una distribuz. di probabilità

- But how to make it operational? How to compute this value?
- Intuition of Bayesian inference:
  - Use Bayes rule to transform this equation into a set of probabilities that are easier to compute (and give the right answer)

si vogliono fare delle approssimazioni

# Using Bayes Rule

non c'è approssimazione!

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

"probabilità a priori di x".

"probabilità di verosimiglianza"

"probabilità x condizionata da y"

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(t_1^n|w_1^n)$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}}\, P(w_1^n|t_1^n)P(t_1^n)$$

lo posso fare xké la variabile su cui varia argmax è $t_1^n$, il PoS, non ci sono le parole. $P(w_1^n)$ non influenza la sequenza che massimizza questa prob.

al denominat. per qualsiasi sequenza di tag, avrò sempre lo stesso valore

il denominatore NON contiene i tag

sto invertendo il problema

"la probabilità di parole dato il tag"

xké è + facile da calcolare

language model: la probab. delle parole

essendo al denominatore non condiziona il massimizzare

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n|t_1^n)P(t_1^n)$$

la sequenza di Tag che massimizza questo è la stessa sequenza che massimizza questo

Questa probabilità non è sempre 1 ma è una costante: la prob. di "Paolo Paolo Paolo Paolo" è molto + bassa di "Paolo ama Francesca"

è la prob. a priori di una sequenza di parole (che interessa ad esempio nello speech recognition)

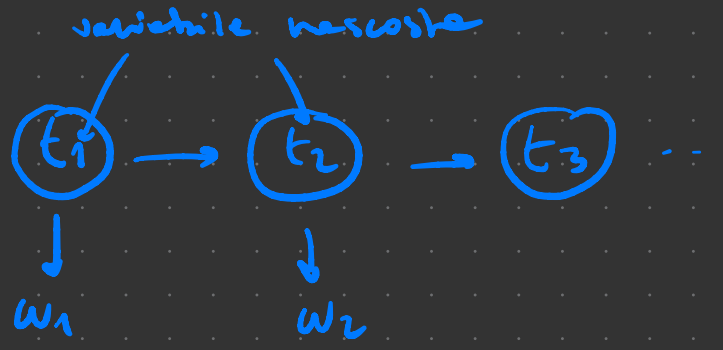
$$\frac{\dots P(\text{nome verbo nome})}{P(\text{Paolo ama Fra})} \quad \text{oppure} \quad \frac{\dots P(\text{articolo verbo articolo})}{P(\text{Paolo ama Fra})} \quad \text{oppure} \quad \frac{\dots P(\text{nome nome nome})}{P(\text{Paolo ama Fra})}$$

quindi "$P(\text{Paolo ama Fra})$" si può eliminare

Ma perché si vuole arrivare a $\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n | t_1^n) P(t_1^n)$?

Perché quelle 2 prob. sono + facili da calcolare, in particolare da approssim.

---




] modello hidden

] parole in input: osservabili

# Likelihood and Prior

ipotesi markoviane

verosimiglianza

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \overbrace{P(w_1^n|t_1^n)}^{\text{likelihood}} \overbrace{P(t_1^n)}^{\text{prior}}$$

$P(w_1 w_2 | t_1 t_2) \approx P(w_1|t_1) \times P(w_2|t_2)$

"la prob. sulle sequenze è ≈ al prodotto delle prob. sulle parole"

approssimaz.

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i) \quad \beta$$

$P(t_1 t_2 t_3) \approx P(t_3|t_2) \times P(t_2|t_1) \times P(t_1|t_\emptyset)$

tag standard di inizio sequenza

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \quad \alpha$$

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1})$$

# Learning Two Kinds of Probabilities: PoS->PoS

Tag transition probabilities $P(t_i|t_{i-1})$

- Determiners likely to precede adjs and nouns
  - That/DT flight/NN
  - The/DT yellow/JJ hat/NN
- So we expect P(NN|DT) and P(JJ|DT) to be high
- Compute P(NN|DT) by counting in a labeled corpus: **Learning**

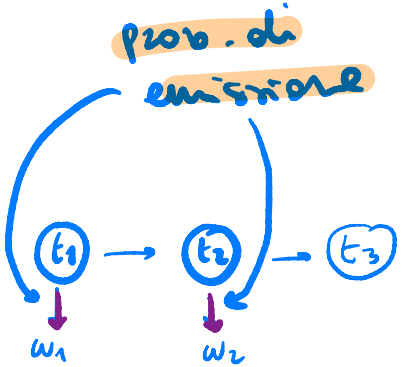
$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

$$P(NN|DT) = \frac{C(DT, NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

# Learning Two Kinds of Probabilities: PoS->word

Word likelihood probabilities $\mathbf{P(w_i|t_i)}$

- VBZ (3sg Pres Verb) likely to be "is"
- Compute P(is|VBZ) by counting in a labeled corpus: **Learning**




$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

quante volte "is" è taggato come verbo

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47$$

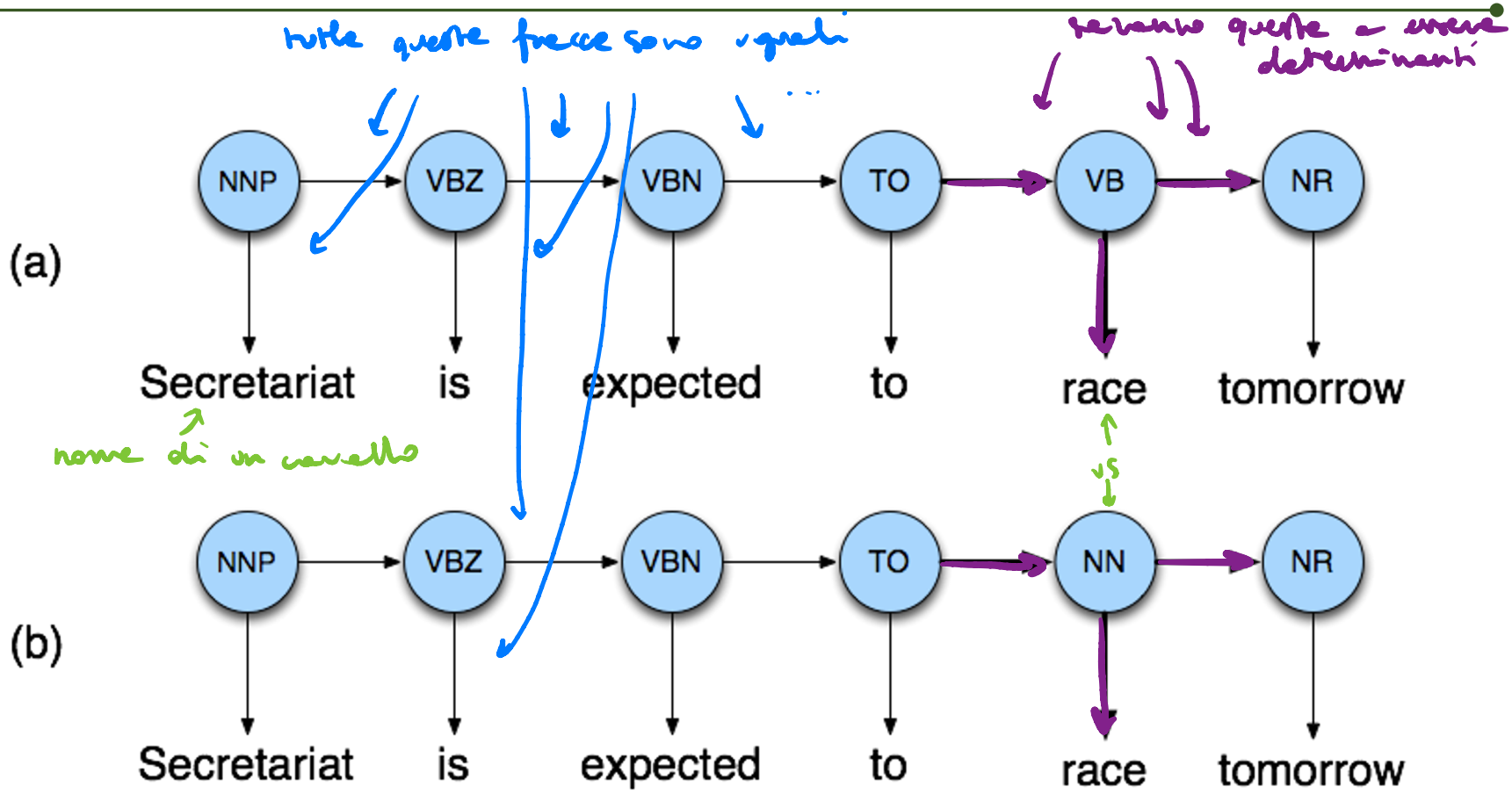
la prob. che un verbo sia "is"

# Example: The Verb “race”

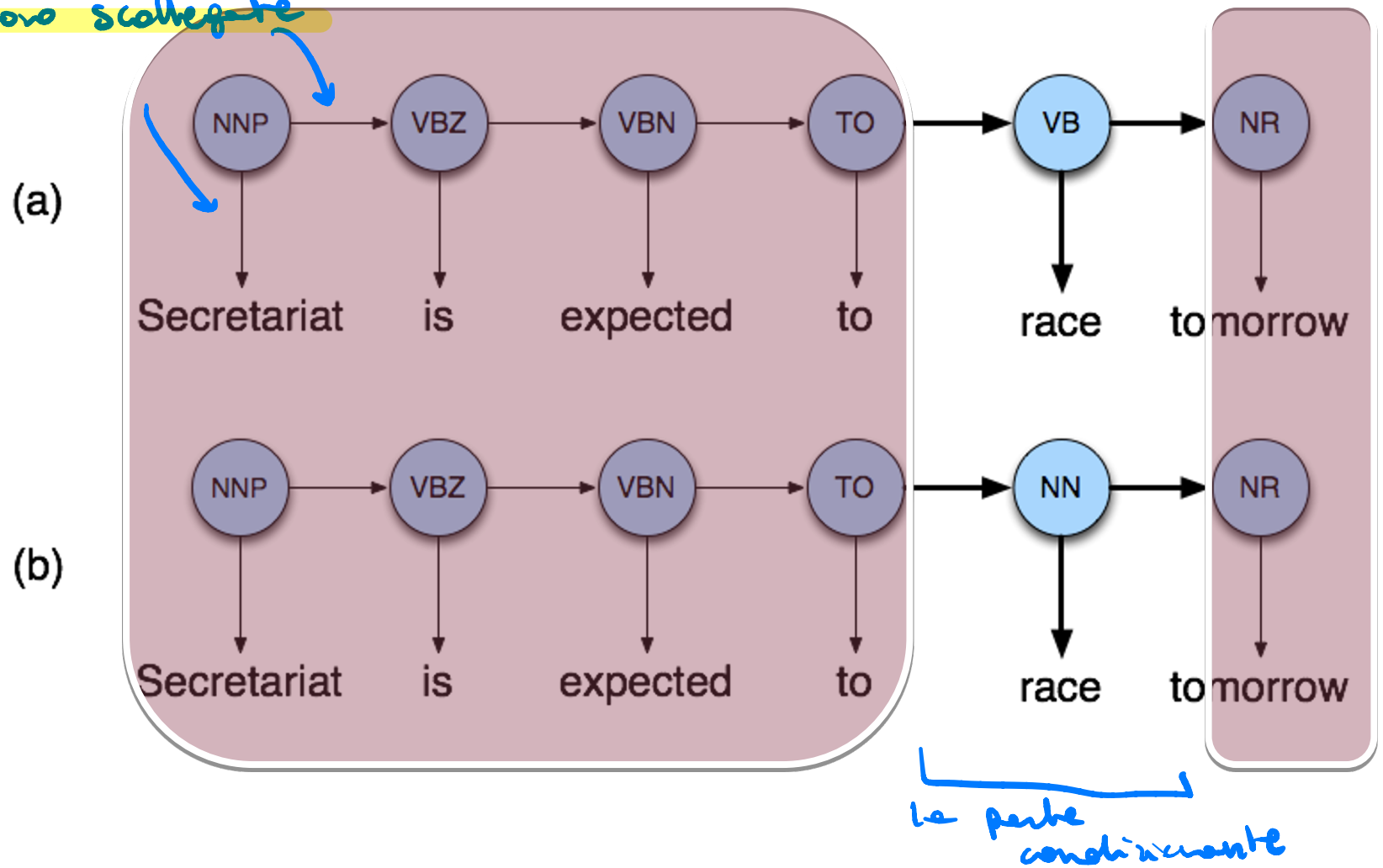
- Secretariat_NNP is_VBZ expected_VBN to_TO **race**_VB tomorrow_NR
- People_NNS continue_VB to_TO inquire_VB the_DT reason_NN for_IN the_DT **race**_NN for_IN outer_JJ space_NN

How do we pick the right tag?

# Disambiguating "race"

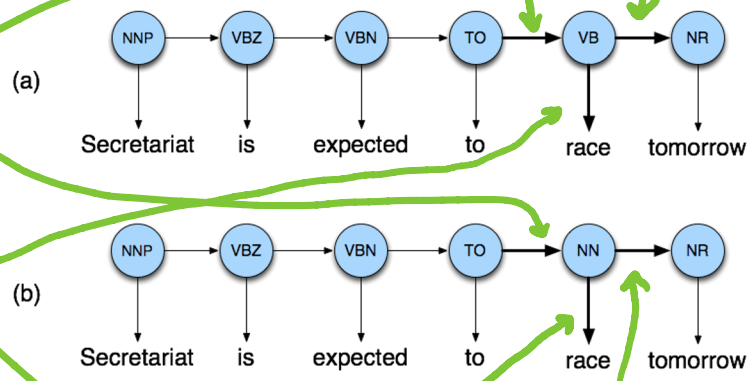


# Disambiguating “race”



# Example

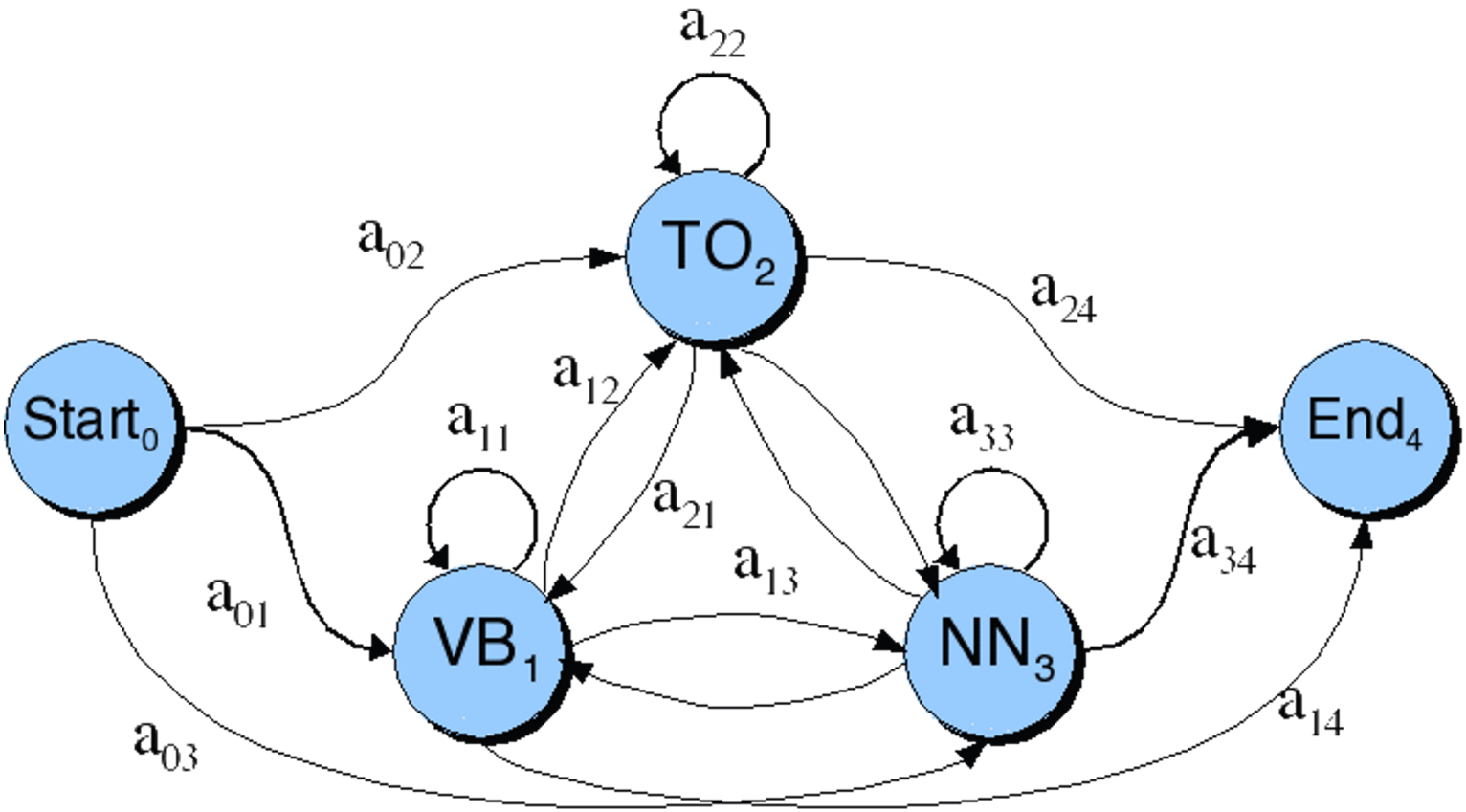
- P(NN|TO) = .00047
- P(VB|TO) = .83
- P(race|NN) = .00057
- P(race|VB) = .00012
- P(NR|VB) = .0027
- P(NR|NN) = .0012
- P(VB|TO)P(NR|VB)P(race|VB) = .00000027
- P(NN|TO)P(NR|NN)P(race|NN)=.00000000032



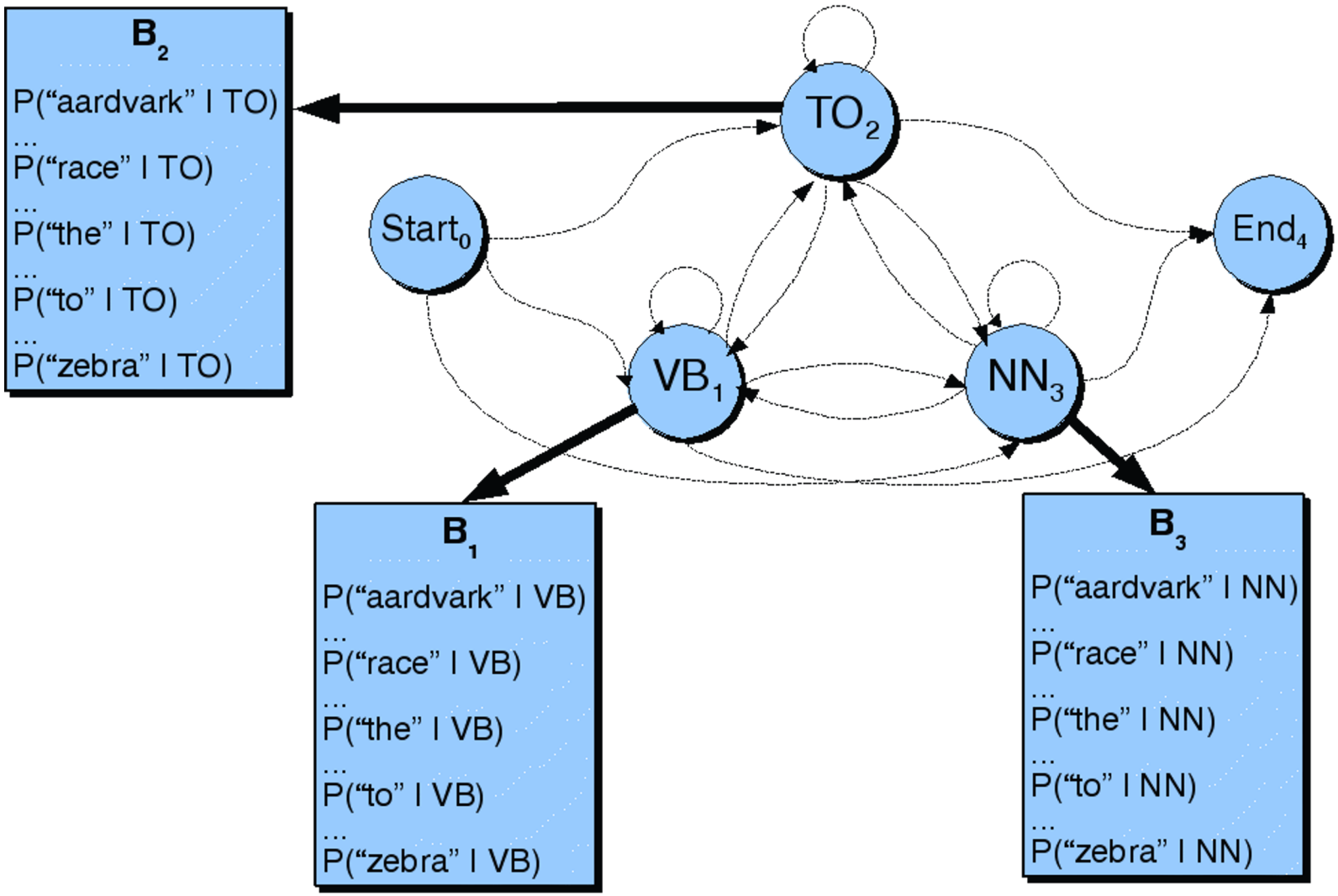

← ha una prob. + alta

So we (correctly) choose the verb tag for "race"

# POS Transition Probabilities



# Observation Likelihoods



# Decoding HMMs

- Ok, now we have a complete model that can give us what we need. Recall that we need to get

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n)$$

- We could just enumerate all paths given the input and use the model to assign probabilities to each.

# Example: Janet will back the bill

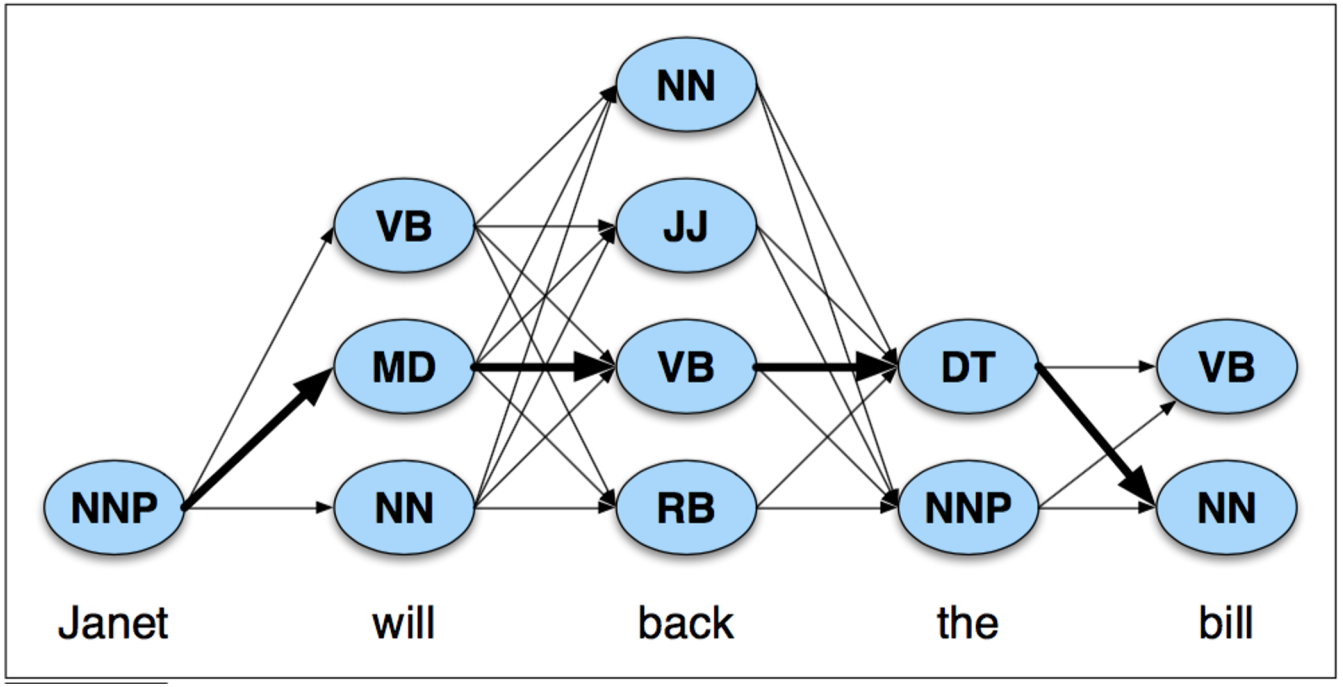
## Janet/NNP will/MD back/VB the/DT bill/NN

| | NNP | MD | VB | JJ | NN | RB | DT |
|---|---|---|---|---|---|---|---|
| <*s*> | 0.2767 | 0.0006 | 0.0031 | 0.0453 | 0.0449 | 0.0510 | 0.2026 |
| **NNP** | 0.3777 | 0.0110 | 0.0009 | 0.0084 | 0.0584 | 0.0090 | 0.0025 |
| **MD** | 0.0008 | 0.0002 | 0.7968 | 0.0005 | 0.0008 | 0.1698 | 0.0041 |
| **VB** | 0.0322 | 0.0005 | 0.0050 | 0.0837 | 0.0615 | 0.0514 | 0.2231 |
| **JJ** | 0.0366 | 0.0004 | 0.0001 | 0.0733 | 0.4509 | 0.0036 | 0.0036 |
| **NN** | 0.0096 | 0.0176 | 0.0014 | 0.0086 | 0.1216 | 0.0177 | 0.0068 |
| **RB** | 0.0068 | 0.0102 | 0.1011 | 0.1012 | 0.0120 | 0.0728 | 0.0479 |
| **DT** | 0.1147 | 0.0021 | 0.0002 | 0.2157 | 0.4744 | 0.0102 | 0.0017 |

| | Janet | will | back | the | bill |
|---|---|---|---|---|---|
| **NNP** | 0.000032 | 0 | 0 | 0.000048 | 0 |
| **MD** | 0 | 0.308431 | 0 | 0 | 0 |
| **VB** | 0 | 0.000028 | 0.000672 | 0 | 0.000028 |
| **JJ** | 0 | 0 | 0.000340 | 0.000097 | 0 |
| **NN** | 0 | 0.000200 | 0.000223 | 0.000006 | 0.002337 |
| **RB** | 0 | 0 | 0.010446 | 0 | 0 |
| **DT** | 0 | 0 | 0 | 0.506099 | 0 |

# Example: Janet will back the bill

Janet/NNP will/MD back/VB the/DT bill/NN



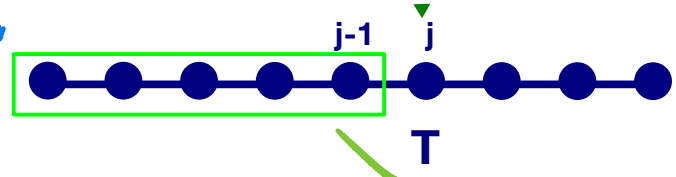
# Question

- If there are 30 or so tags in the Penn set
- And the average sentence is around 20 words...
- How many tag sequences do we have to enumerate to argmax over in the **worst case** scenario?

$30^{20}$

# Dynamic Programming idea

- Consider a state sequence (tag sequence) that ends at state **j** with a particular tag **T**.
- The probability of that tag sequence can be broken into two parts
  - The probability of the BEST tag sequence up through **j-1**
  - Multiplied by the transition probability from the tag at the end of the **j-1** sequence to **T** (and the observation probability of the word given tag **T**)



# Markov assumption

P(flies|N) P(like|V) P(N|V)

>

P(flies|V) P(like|V) P(V|V)

***follows:***

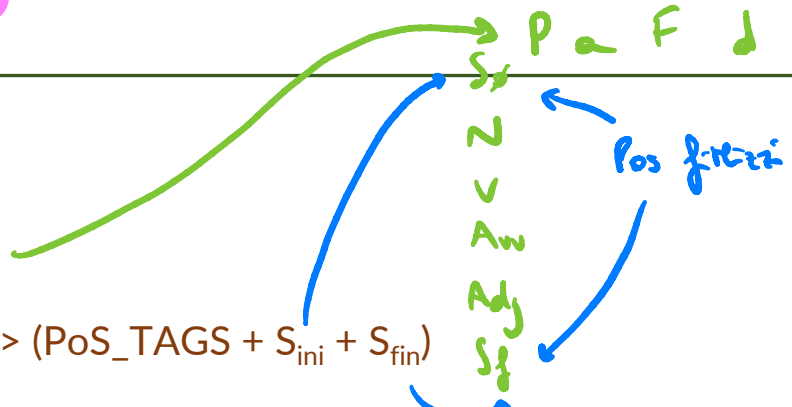
P(flies|N) P(like|V) P(N|V) P(a|ART) P(V|ART)

>

P(flies|V) P(like|V) P(V|V) P(a|ART) P(V|ART)

**Store ONLY the MAX value and its path! -> O(D x $|V|^2$)**

# Viterbi Summary




- Create an array matrix
  - With columns corresponding to inputs
  - Rows corresponding to possible states -> (PoS_TAGS + $S_{ini}$ + $S_{fin}$)
- Sweep through the array in one pass filling the columns left to right using our transition probs and observations probs
- **Dynamic programming key is that we need only store the MAX prob path to each cell, and not all paths.**

prob. di Viterbi

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

# The Viterbi Matrix

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t)$$

| | |
|---|---|
| $v_{t-1}(i)$ | the **previous Viterbi path probability** from the previous time step |
| $a_{ij}$ | the **transition probability** from previous state $q_i$ to current state $q_j$ |
| $b_j(o_t)$ | the **state observation likelihood** of the observation symbol $o_t$ given the current state $j$ |

NON mi dà la prob. di tutte le possibili sequenze (troppo oneroso), ma solo la prob. della sequenza + probabile

↑

# The Viterbi Algorithm

**function** VITERBI(*observations* of len $T$, *state-graph* of len $N$) **returns** *best-path*

create a path probability matrix *viterbi*$[N+2,T]$

**for** each state $s$ **from** 1 **to** $N$ **do** ; initialization step

$viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$

$backpointer[s,1] \leftarrow 0$

**for** each time step $t$ **from** 2 **to** $T$ **do** ; recursion step

**for** each state $s$ **from** 1 **to** $N$ **do**

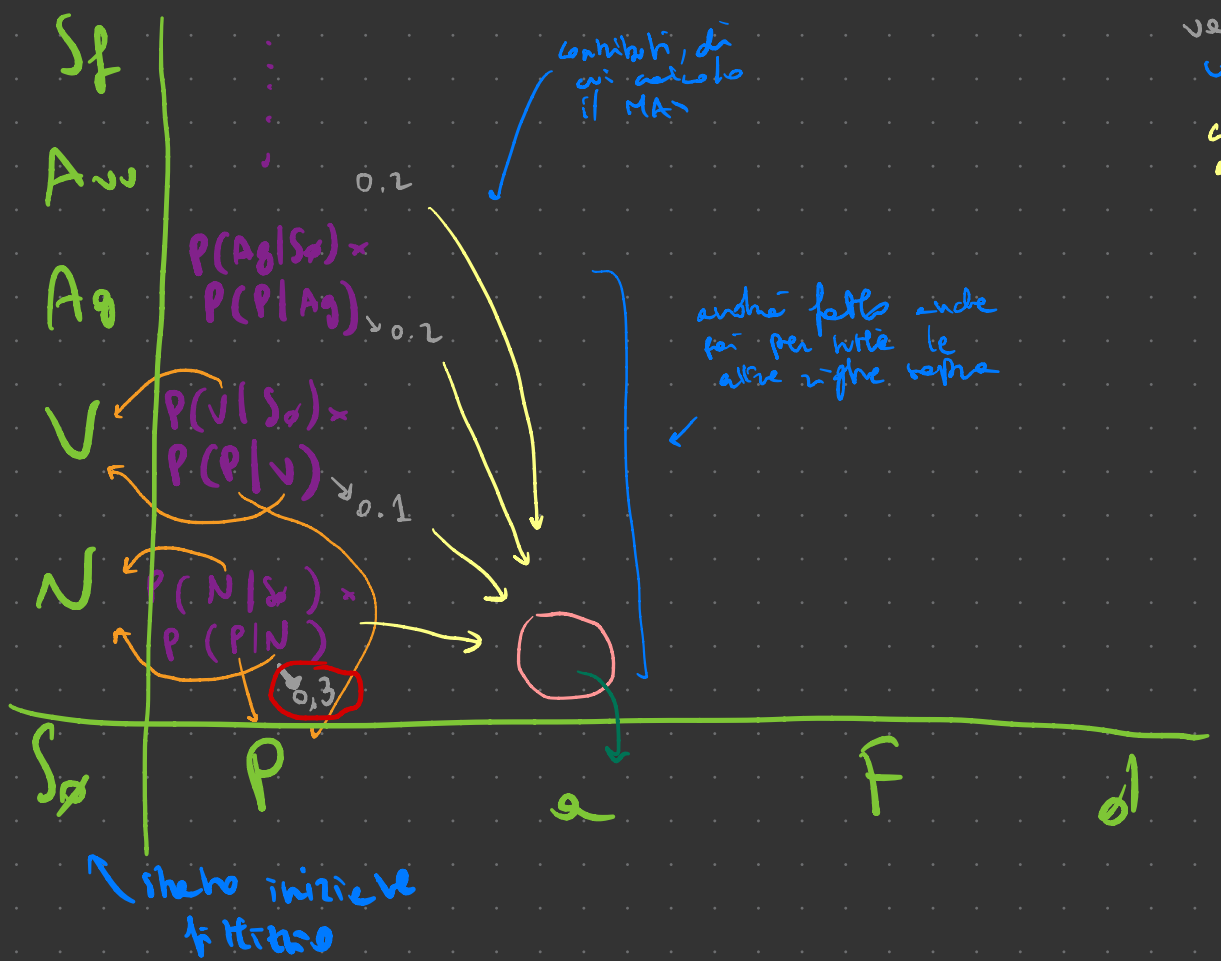
$$viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$$

$$backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$$

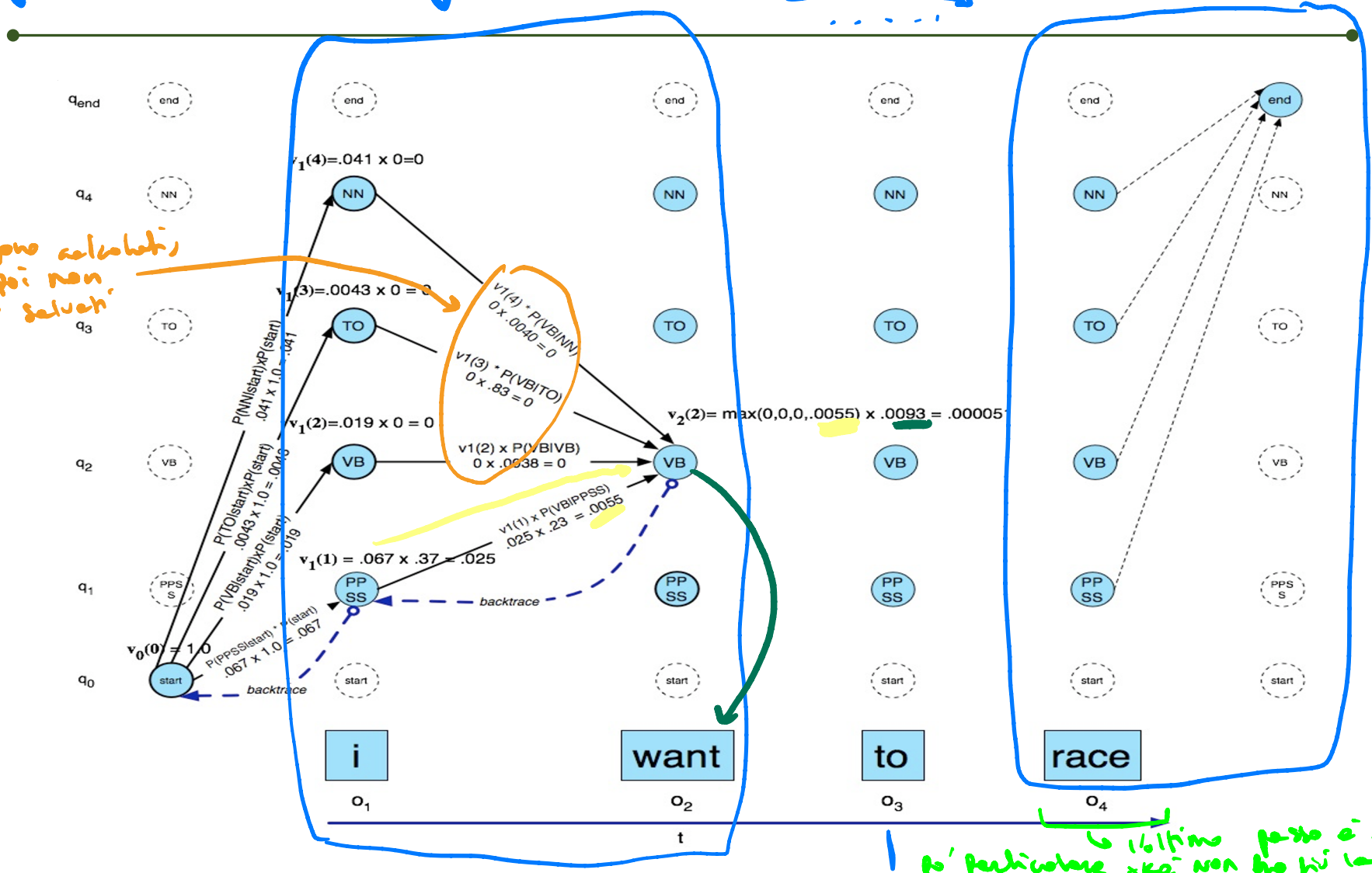
$$viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F} \quad \text{; termination step}$$

$$backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F} \quad \text{; termination step}$$

**return** the backtrace path by following backpointers to states back in time from $backpointer[q_F,T]$


Sf
Avv
Ag
V
N
Sa
P
a
F
d
0.2
P(Ag|Sa) x P(P|Ag)
0.2
P(V|Sa) x P(P|V)
0.1
P(N|Sa) x P(P|N)
0,3
contributi di cui calcolo il MAP
anche fatto anche per tutte le altre righe sopra
stato iniziale filtrato
valori numerici: prob. di un fuffisso a quella seq.
confronto i vecchi valori di viterbi, moltipl. x la prob. di transizione
Paolo alla Francesca dolcemente

# Example



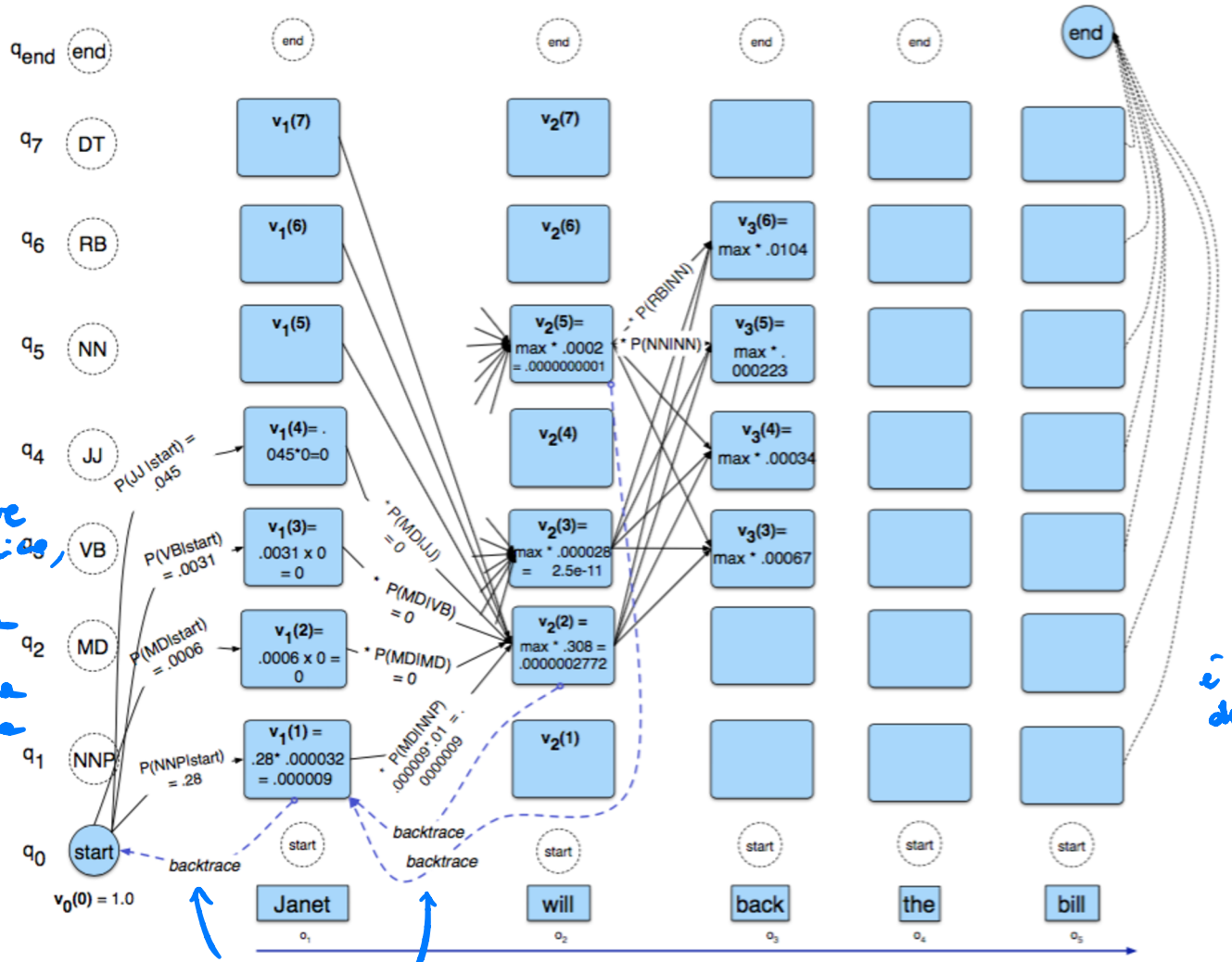

Matrice di Viterbi

finestra di Viterbi: ogni volta mi interessa solo la colonna prec., non anche quelle prima (perché qui la memoria di Markov è 1. Fosse stata 2, avrei preso in considd. fino a 2 colonne prima

vengono calcolati ma poi non sono salvati

l'ultimo passo è un po' particolare xké non ho più la prob. di emissione

# Example




(può essere anche la stessa matrice, dove x ogni nodo salvo la coppia valore - id. della riga preced. da cui arrivo)

è la "analogia" della regr. miglior

Si costruisce una matrice parallela di back pointer per risalire a quel è stata la sequenza che ha portato al (massimo) valore finale

# Viterbi Summary

- **Dynamic programming key is that we need only store the MAX prob path to each cell, and not all paths.**
- "Life can only be understood backwards, but must be lived forwards"

# Esercizio su HMM

- Fare il learning di un modello HMM sul corpus:
  - Paolo/N pesca/V
  - Giovanni/N ama/V i/D cani/N
  - Francesca/N  ama/N
  - Una/D pesca/N Francesca/A
- Fare il decoding, facendo girare a mano l'algoritmo di Viterbi, della frase:
  - **Paolo ama Francesca**

# Extending the HMM Algorithm to Trigrams

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1})$$

diversa dalla freccia = 1

modelli più complessi

memoria di lunghezza 2

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1}, t_{i-2})$$

diventa una prob. di transizione su trigrammi

# Extending the HMM Algorithm to Trigrams

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \left[ \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1}, t_{i-2}) \right] P(t_{n+1}|t_n)$$

N N V

$$P(t_i|t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})}$$

N N

# Extending the HMM Algorithm to Trigrams

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n | w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \left[ \prod_{i=1}^{n} P(w_i|t_i) P(t_i|t_{i-1}, t_{i-2}) \right] P(t_{n+1}|t_n)$$

se non lo faccio, e ad esempio sbaglio a scrivere una parola, quella non verrà trovata nel corpus, e la sua prob. di emissione, per tutti i tag, sarà 0, facendo fallire tutto

$$P(t_i|t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})}$$

**Sparsness!!!** ← anche verranno calcolate anche prob. impossibili in un corpus (V→V→V)

in realtà dipende dalla dimens. del corpus, però alcuni sono quasi impos. in ogni caso. E i corpus comunque vanno aumentati esponenzialm.

SMOOTHING → una possibilità è dare a questi casi una prob. di default / di massa

posso: • applicare una prob. equidistribuita su tutti i tag : 1/n (n = n° di tag)
• dire che la prob. è 1 per Nome, 0 per tutti gli altri tag

# Sparsity of data -> combining n-grams

$$\text{Trigrams} \quad \hat{P}(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})}$$

$$\text{Bigrams} \quad \hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$$

$$\text{Unigrams} \quad \hat{P}(t_i) = \frac{C(t_i)}{N}$$

un'altra tecnica è applicare correzioni di laplace
↳ a quegli eventi che non si verificherebbero:
gli associo le prob. dei soli eventi che lo compongono

$$P(t_i|t_{i-1}t_{i-2}) = \lambda_3\hat{P}(t_i|t_{i-1}t_{i-2}) + \lambda_2\hat{P}(t_i|t_{i-1}) + \lambda_1\hat{P}(t_i)$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

# Lambdas -> deleted interpolation algorithm

**function** DELETED-INTERPOLATION(*corpus*) **returns** $\lambda_1, \lambda_2, \lambda_3$

$\lambda_1 \leftarrow 0$
$\lambda_2 \leftarrow 0$
$\lambda_3 \leftarrow 0$
**foreach** trigram $t_1, t_2, t_3$ with $C(t_1, t_2, t_3) > 0$
  **depending** on the maximum of the following three values
    **case** $\frac{C(t_1,t_2,t_3)-1}{C(t_1,t_2)-1}$: increment $\lambda_3$ by $C(t_1, t_2, t_3)$
    **case** $\frac{C(t_2,t_3)-1}{C(t_2)-1}$: increment $\lambda_2$ by $C(t_1, t_2, t_3)$
    **case** $\frac{C(t_3)-1}{N-1}$: increment $\lambda_1$ by $C(t_1, t_2, t_3)$
  **end**
**end**
normalize $\lambda_1, \lambda_2, \lambda_3$
**return** $\lambda_1, \lambda_2, \lambda_3$

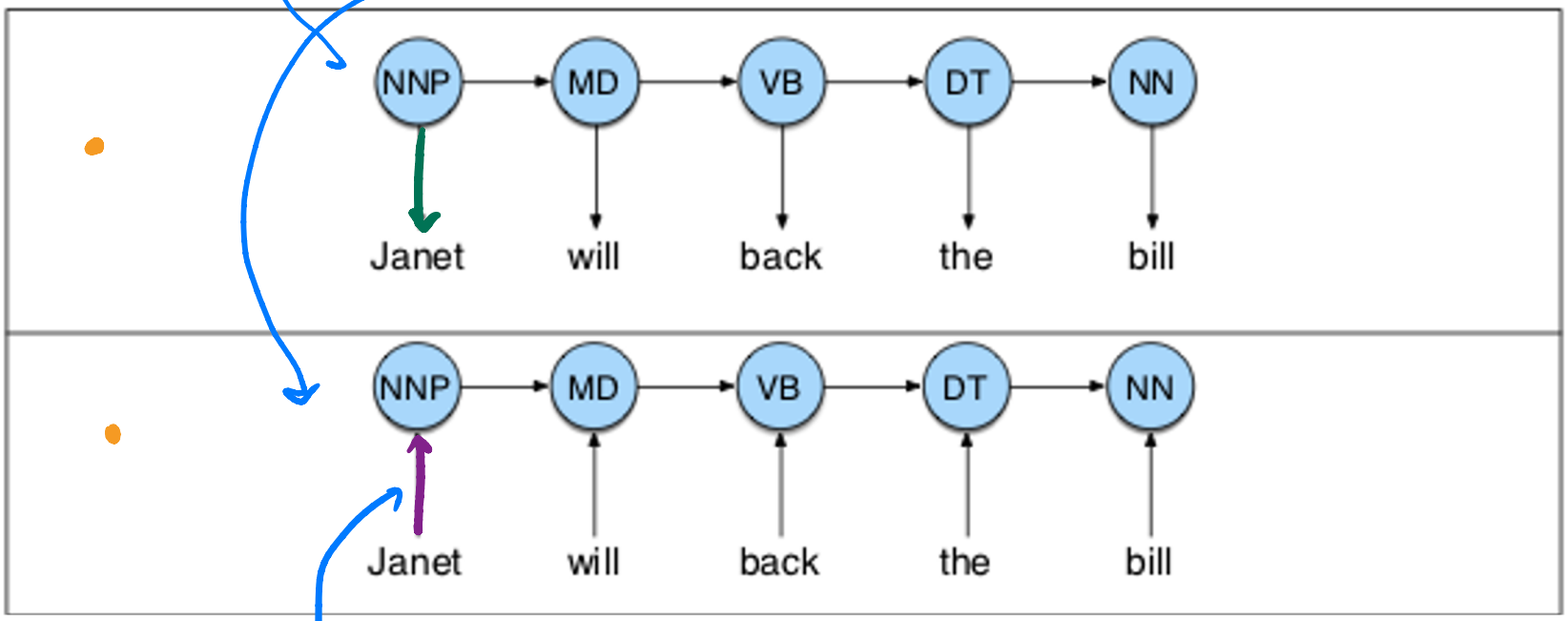
# Problems of HMM

- No Rich Feature Information
  - Rich information are required
    - When $x_k$ is complex
    - When data of $x_k$ is sparse
- Example: POS Tagging
- How to evaluate $P(w_k|t_k)$ for unknown words $w_k$ ?
- Useful features
  - Suffix, e.g., -ed, -tion, -ing, etc.
  - Capitalization

# Outline

- Rule-Based Tagger: ENGTWOL (ENGlish TWO Level analysis)
- **Stochastic Tagger**
  - HMM
  - **MEMM**
- PoS tagging conclusion

# HMM vs. MEMM (MAXIMUM ENTROPY MARKOV MODELS )




+ difficile da calcolare rispetto a quella !

# HMM vs. MEMM (MAXIMUM ENTROPY MARKOV MODELS )

$$\hat{T} = \operatorname*{argmax}_T P(T|W)$$
$$= \operatorname*{argmax}_T P(W|T)P(T)$$
$$= \operatorname*{argmax}_T \prod_i P(word_i|tag_i) \prod_i P(tag_i|tag_{i-1})$$

qui sarebbe estremam. + complesso

VS.

$$\hat{T} = \operatorname*{argmax}_T P(T|W)$$
$$= \operatorname*{argmax}_T \prod_i P(t_i|w_i, t_{i-1})$$

posso aggiungere:
- la parola finisce in "-ate"
- la parola è la stessa ma inizia con una maiusc/minusc.
...

features

## Generative vs. Discriminative (Ch. 5&8, 3rd ed.)

- *... generative model: a model that is trained to generate the data x from the class y. The likelihood term P(x|y) expresses that we are given the class y and are trying to predict which features we expect to see in the input x.*

- *A discriminative model takes this direct approach, computing P(y|x) by discriminating among the different possible values of the class y rather than first computing a likelihood:*

# MEMM

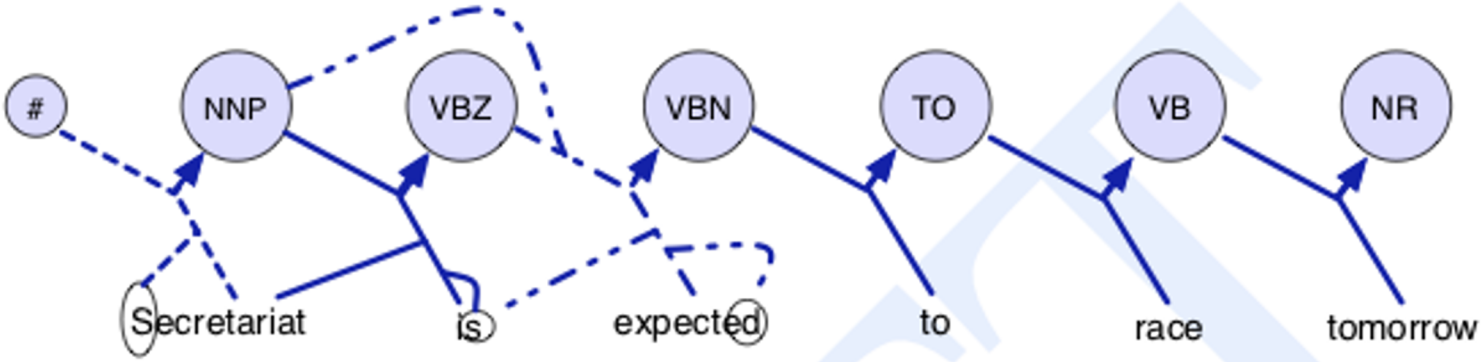



**Figure 6.21** An MEMM for part-of-speech tagging, augmenting the description in Fig. 6.20 by showing that an MEMM can condition on many features of the input, such as capitalization, morphology (ending in *-s* or *-ed*), as well as earlier words or tags. We have shown some potential additional features for the first three decisions, using different line styles for each class.

# Maximum Entropy modeling

- We may identify a more **heterogeneous set of features** which contribute in some way to the choice of the current word.
- Maxent combines these features in a **probabilistic model.** The given features provide a constraint on the model.
- We would like to have a probability distribution which, outside of these constraints, is **as uniform as possible** – has the **maximum entropy** among all models that satisfy these constraints.

$P(W_i \mid F_1 F_2 .. F_n)$ ← features

↓ dà una certa parola, che finisca in -are, ...

↓

tutte le prob. delle features che NON compaiono nella formula sono UNIFORMI

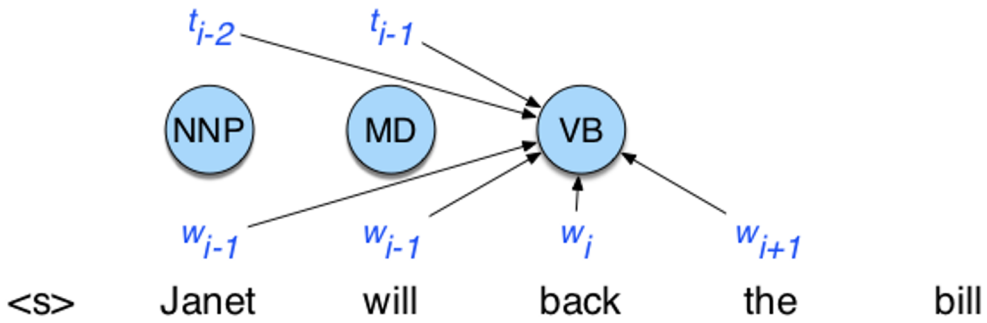
# Maxent P(tag|word)

- Can do surprisingly well just looking at a word by itself:
  - Word the: the DT
  - Prefixes unfathomable: un- JJ
  - Suffixes Importantly: -ly RB
  - Cap. Meridian: CAP NNP
  - Shapes 35-year: d-x JJ
- Then build a classifier to predict tag

  Maxent P(tag|word): 93.7% overall / 82.6% unknown

# MEMM (Maximum Entropy Markov Model)

- A sequence version of the maximum entropy classifier




$$\hat{T} = \operatorname*{argmax}_T P(T|W)$$

$$= \operatorname*{argmax}_T \prod_i P(t_i|w_i, t_{i-1})$$

# MEMM tagger

One step up: also condition on previous tags: Modelling

$$p(y_1 \dots y_n | x_1 \dots x_n) = \prod_{i=1}^{n} p(y_i | y_1 \dots y_{i-1}, x_1 \dots x_n)$$

$$= \prod_{i=1}^{n} p(y_i | y_{i-1}, x_1 \dots x_n)$$

- Learning
  - Train $p(y_i | y_{i-1}, x_1 \dots x_n)$ as a discrete log-linear (MaxEnt) model
- Decoding

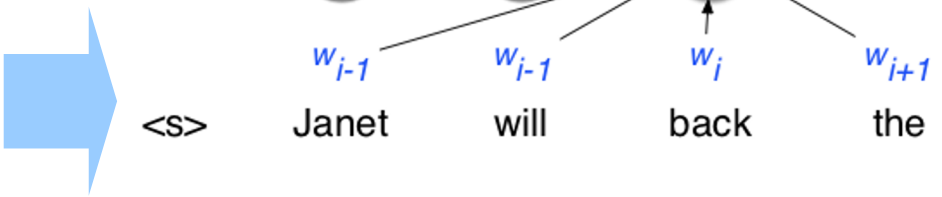
$$p(y_i | y_{i-1}, x_1 \dots x_n) = \frac{e^{w \cdot \phi(x_1 \dots x_n, i, y_{i-1}, y_i)}}{\sum_{y'} e^{w \cdot \phi(x_1 \dots x_n, i, y_{i-1}, y')}}$$

- This is referred to as an MEMM tagger [Ratnaparkhi 96]

# Features

# MEMMs

$$\langle t_i, w_{i-2}\rangle, \langle t_i, w_{i-1}\rangle, \langle t_i, w_i\rangle, \langle t_i, w_{i+1}\rangle, \langle t_i, w_{i+2}\rangle$$
$$\langle t_i, t_{i-1}\rangle, \langle t_i, t_{i-2}, t_{i-1}\rangle,$$
$$\langle t_i, t_{i-1}, w_i\rangle, \langle t_i, w_{i-1}, w_i\rangle \langle t_i, w_i, w_{i+1}\rangle,$$




$t_i$ = VB and $w_{i-2}$ = Janet
$t_i$ = VB and $w_{i-1}$ = will
$t_i$ = VB and $w_i$ = back
$t_i$ = VB and $w_{i+1}$ = the
$t_i$ = VB and $w_{i+2}$ = bill
$t_i$ = VB and $t_{i-1}$ = MD
$t_i$ = VB and $t_{i-1}$ = MD and $t_{i-2}$ = NNP
$t_i$ = VB and $w_i$ = back and $w_{i+1}$ = the

# More features

- $w_i$ contains a particular prefix (from all prefixes of length ≤ 4)
- $w_i$ contains a particular suffix (from all suffixes of length ≤ 4)
- $w_i$ contains a number
- $w_i$ contains an upper-case letter wi contains a hyphen
- $w_i$ is all upper case
- $w_i$'s word shape
- $w_i$'s short word shape
- $w_i$ is upper case and has a digit and a dash (like CFC-12)
- $w_i$ is upper case and followed within 3 words by Co., Inc., etc.

# MEMM Modelling

proporzionale al peso più il valore della feature

$$P(y|x) \quad ?= \sum_{i=1}^{N} w_i f_i$$

peso (non è + la word)

feature, normalm. booleane

$$?= \quad w \cdot f$$

"prob. che un certo output appartenga a una certa classe dati degli osservabili"

$$p(y = c|x) = p(c|x) \quad = \frac{1}{Z} \exp \sum_i w_i f_i$$

fattore di normalizzazione/partizione

# MEMM Modelling

definire le features che vogliamo

le features le abbiamo già definite. Come calcolare i giusti pesi?

$$\hat{T} = \operatorname*{argmax}_T P(T|W)$$

$$= \operatorname*{argmax}_T \prod_i P(t_i | w_{i-l}^{i+l}, t_{i-k}^{i-1})$$

$$= \operatorname*{argmax}_T \prod_i \frac{\exp\left(\sum_i w_i f_i(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}{\sum_{t' \in \text{tagset}} \exp\left(\sum_i w_i f_i(t', w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}$$

prob. che si vuole massimizzare

# MEMM Learning

- MaxEnt learning -> Multinomial Logistic Regression Learning
- **Conditional maximum likelihood estimation:** we choose the parameters w that maximize the (log) probability of the y labels in the training data given the observations x.

$$\hat{w} = \underset{w}{\operatorname{argmax}} \sum_{j} \log P(y^{(j)}|x^{(j)})$$

devo calcol. la prob. di massima verosimiglianza considerando il training data

# MEMM Learning

si tratta di calcolare il valore dei pesi che rendono massima la prob. su training set

$$L(w) = \sum_i \log P(y^{(j)}|x^{(j)})$$

$$= \sum_j \log \frac{\exp\left(\sum_{i=1}^{N} w_i f_i(y^{(j)}, x^{(j)})\right)}{\sum_{y' \in Y} \exp\left(\sum_{i=1}^{N} w_i f_i(y'^{(j)}, x^{(j)})\right)}$$

$$= \sum_j \log \exp\left(\sum_{i=1}^{N} w_i f_i(y^{(j)}, x^{(j)})\right) - \sum_j \log \sum_{y' \in Y} \exp\left(\sum_{i=1}^{N} w_i f_i(y'^{(j)}, x^{(j)})\right)$$

Convex optimization problem, so we use hill-climbing methods like stochastic gradient ascent, L-BFGS (Nocedal 1980, Byrd et al. 1995)

Qui non c'è una soluz. esatta, non può essere risolto analiticamente

# MEMM Decoding

- Simplest algorithm
  - Greedy: at each step in sequence, select tag that maximizes P(tag | nearby words, nearby tags)
- In practice
  - Viterbi algorithm
  - Beam search

$$p(y_i|y_{i-1}, x_1 \dots x_n) = \frac{e^{w \cdot \phi(x_1 \dots x_n, i, y_{i-1}, y_i)}}{\sum_{y'} e^{w \cdot \phi(x_1 \dots x_n, i, y_{i-1}, y')}}$$

più importante capire i motivi che i dettagli matematici

| | HMM | vs | MEMM |
|---|---|---|---|
| Modelling | molto semplice, ma anche debole (non si poss. aggiungere features) | | abb. semplice (si poss. agg. features qualsiasi) |
| Training | tecniche molto buone ↖ qui sta la vera diff. | | servono approssimazioni ↗ |
| Decoding | usano le stesse tecniche (complicate) | | |
| | generativo | vs | discriminativo |

Quando usare uno o l'altro?

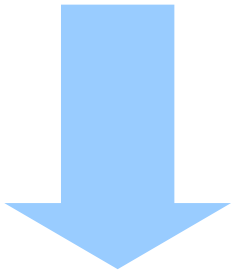
Se mi "basta" un'accuracy del ~97%, posso spendere poco tempo e usare HMM.

Se invece ho un corpus molto grosso, su cui posso modell. molte features e voglio un'accuracy del 98%, spendo molte ore macchina in learning e uso RLHMM

(è un training set molto + grosso)

# MEMM Decoding: Viterbi!

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T$$



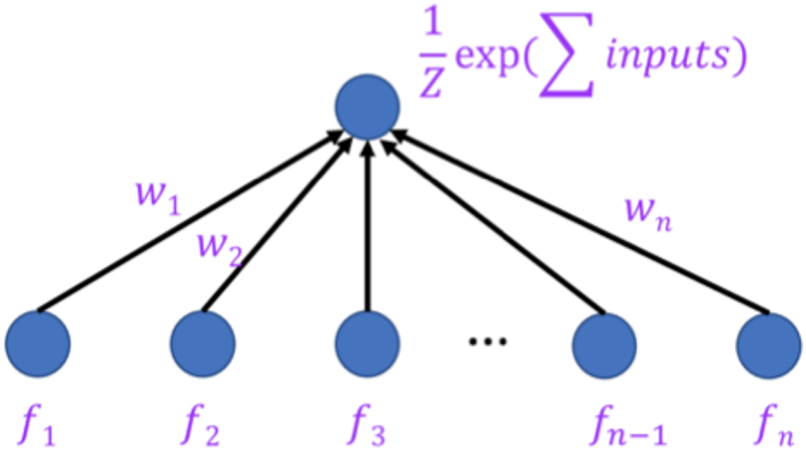
$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, P(s_j|s_i, o_t) \quad 1 \le j \le N, 1 < t \le T$$

# MEMM/LogisticRegression Summary

- model P(y|x) only, have no generative process
- can use arbitrary local/global features, including correlated ones
- can use for classification, or choosing from n-best list (reranking)
- Learning involves iteratively updating the weights, so typically slower than for generative models
- Decoding -> Greedy/Viterbi

# Extension to neural network (Sharon Goldwater)

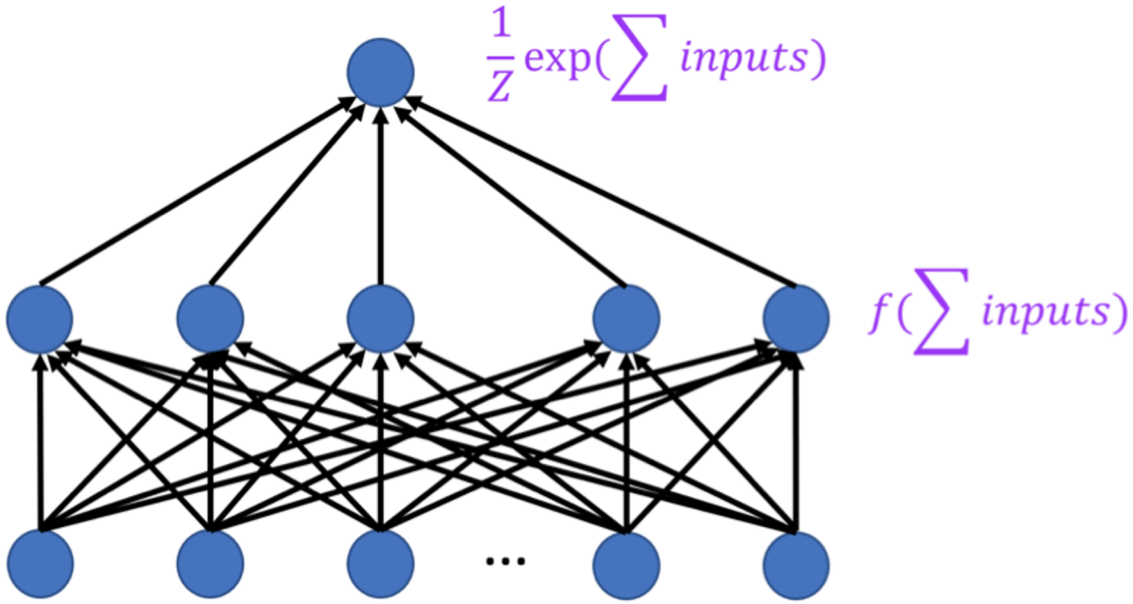
- Logistic regression can be viewed as a building block of neural networks -> **perceptron**




è un probl. che assomiglia molto a quello di una rete neurale

# Extension to neural network (Sharon Goldwater)

- Adding a fully-connected layer creates one of the simplest types of neural network: a **multi-layer perceptron** (MLP).



# Extension to neural network (Sharon Goldwater)

- Contains one or more hidden layers
  - Each node applies a non-linear function to the sum of its inputs
  - Hidden layers can be viewed as learned representations (embeddings) of the input
- A non-linear classifier: more powerful than logistic regression.
- Also trained using gradient-based methods, but vulnerable to local optima, so can be more difficult to train.
- Like other NNet architectures, really just a complex function computed by multiplying weights by inputs and passing through non-linearities. Not magic.

# Outline

- Rule-Based Tagger: ENGTWOL (ENGlish TWO Level analysis)
- Stochastic Tagger
  - HMM
  - MEMM
- **PoS tagging conclusion**

# POS Tagging Accuracies

- Rough accuracies:
  - Baseline: most freq tag: ~90%
  - Trigram HMM: (senza smoothing) ~95%
  - Maxent P(t|w): (senza memoria) (so non ho bisogno di vicini) 93.7%
  - MEMM tagger: 96.9%
  - Bidirectional MEMM: 97.2%
  - Upper bound: ~98%

    (human agreement)

↑↑↑

dipendono in maniera
una delle come si modella
lo smoothing delle parole
sconosciute

# Tagging Unknown Words

- New words added to (newspaper) language 20+ per month
- Plus many proper names …
- **Increases error rates by 1-2%**

Possible approaches for UW:

- *Method 1*: assume they are nouns
- *Method 2*: assume the unknown words have a probability distribution similar to words only occurring once in the training set.
- *Method 3*: Use morphological information, e.g., words ending with –ed tend to be tagged VBN
- *Method 4*: *Dictionary* (morph-it)
- *Method 5*: *Assume uniform distribution over PoS*

# Evaluation

- The result is compared with a manually coded "Gold Standard"
- Corpus tagged by-hand = Gold standard
- Usually:
  - Training data: 80%
  - Development data: 10%
  - Test data: 10%

il corpus è diviso in 3 insiemi

percentuale di parole che ricadono in questo insieme

# Evaluation

- Typically accuracy reaches 96-97% on WSJ
  - Train: 02-21  Dev: 00/24  Test: 23
  - **Important**: 100% is impossible even for human annotators.
- Factors that affects the performance
  - The amount of training data available
  - The tag set
  - The difference between training corpus and test corpus
  - Dictionary
  - Unknown words
  - **Domain!** -> tweet -> EVALITA 2016 -> Neural Features!!!