Morphology and Syntax, I

Sequence Tagging

Aaron Mueller
CAS CS 505: Natural Language Processing
Boston University
Spring 2026

Admin

- **HW1** grades have been released. The average was again very high—many people got the extra credit.
- **HW2**'s deadline has been extended by one day. It is now due tomorrow, **Mar. 6** at 11:59pm.
- **HW3** has been released. It is due **March 20**.
    - There is *no coding* for this assignment.
- Remember to post to the "Search for Teammates" thread on Piazza before noon on Friday (3/6) if you'd like us to assign you to a final project team.
    - If you've posted but already found a team, please make this clear on Piazza so we can avoid awkward reassignments!
- I will release a practice exam in the coming week.
- (Also, some students pointed out that they couldn't see their HW0 coding grades. Hopefully this is fixed now. Note that the maximum points was 25 for the HW0 coding; if you got all the autograder points, ignore Q2, Q3, Q4.)

HW1 Review: Q9(c)

- The most commonly missed points were on Q9(c).
- Most people proposed vectors that gave high probabilities for positive pairs, but many people missed some negative pairs. Example:

$\mathbf{w}_{\text{Rome}}$: [-1, 3] $\mathbf{w}_{\text{in}}$: [1, 3] $p(\mathbf{w}_{\text{in}} | \mathbf{c}_{\text{Rome}}) = 0.05$ ✅

$\mathbf{c}_{\text{Rome}}$: [-6, 1] $\mathbf{c}_{\text{in}}$: [1, 3] $p(\mathbf{w}_{\text{Rome}} | \mathbf{c}_{\text{in}}) > 0.99$ ❌

- Here's an example of a working vector set (we'll assume $\mathbf{w}_i = \mathbf{c}_i$ for all i):

France: [2, 10] Italy: [2, -10] in: [10, 0]

Paris: [-5, 10] Rome: [-5, -10]

HW2 Bugfix

```
# Load your pre-trained Pile model
model_pile_shakespeare = MultiHeadTransformer(
    vocab_size=vocab_size,
    hidden_dim=hidden_dim,
    context_len=context_len,
    num_heads=4
)
model_pile_shakespeare.load_state_dict(torch.load('TRANSFORMER_MH_pile.pt', map_location=device))
model_pile_shakespeare.to(device)

print("Fine-tuning PILE model on Shakespeare...")

# Fine-tune on Shakespeare (use lower learning rates and num epochs for fine-tuning)
train(
    model_pile_shakespeare,
    train_data_shakes,
    dev_data_shakes,
    epochs=5,
    lr=5e-4,
    device=device,
    save_path='TRANSFORMER_MH_pile_shakespeare.pt'
)

# Evaluate
shakespeare_pile_perp = evaluate_perplexity(model_pile_shakespeare, dev_data_shakes, device)
print(f"\nShakespeare-adapted model perplexity: {shakespeare_perp:.4f}")
```

```
# TODO: Re-evaluate your fine-tuned model's perplexity
# on the dev split of the Pile.
# STUDENT START ----------------------------
# STUDENT END ------------------------------
```

This cell was loading the Pile model into model_shakespeare, rather than model_pile_shakespeare.

It was fine-tuning and evaluating model_shakespeare, so its perplexity was correct.

Please double-check which model you evaluated in this following cell. If you got huge perplexities, please make sure you are evaluating the model that was fine-tuned.

Why study non-LLM-based NLP methods?

- A lot of companies use NLP pipelines that are far more diverse and sophisticated than just "throw an LLM at it."

- **Disadvantages of LLMs:**
  - Expensive to host and query
  - Not the best-performing approaches in low-data settings
  - LLMs often confidently give false answers
- **Advantages of classical methods:**
  - Usually more computationally efficient
  - Perform better for more specialized tasks
  - Usually more transparent and understandable

Overview of Concepts

Parts of speech (POS) include categories like noun, verb, adjectives, among others.

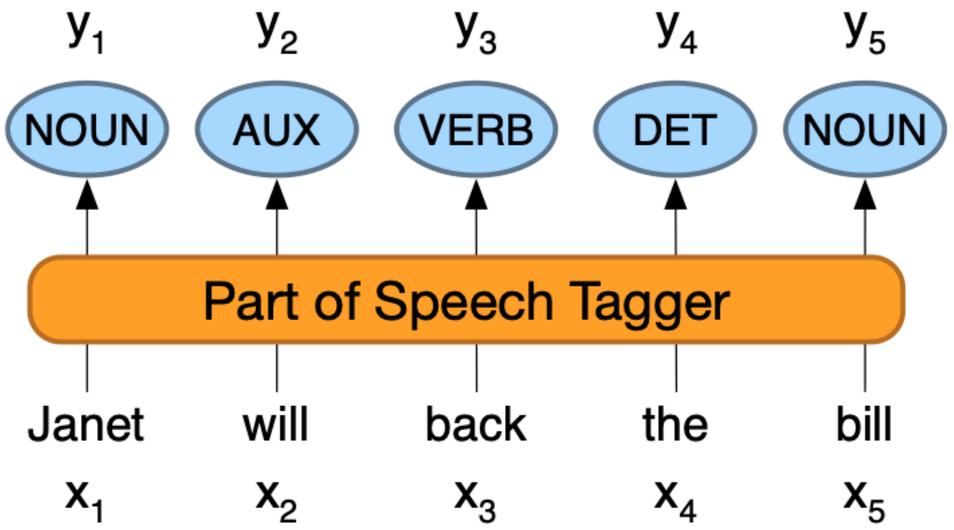
Sequence labeling tasks involve assigning labels to every word in a sequence.

POS tagging is the task of assigning a POS tag to each word in a sequence.

Named entity recognition (NER) is the task of recognizing specific entities and their types.

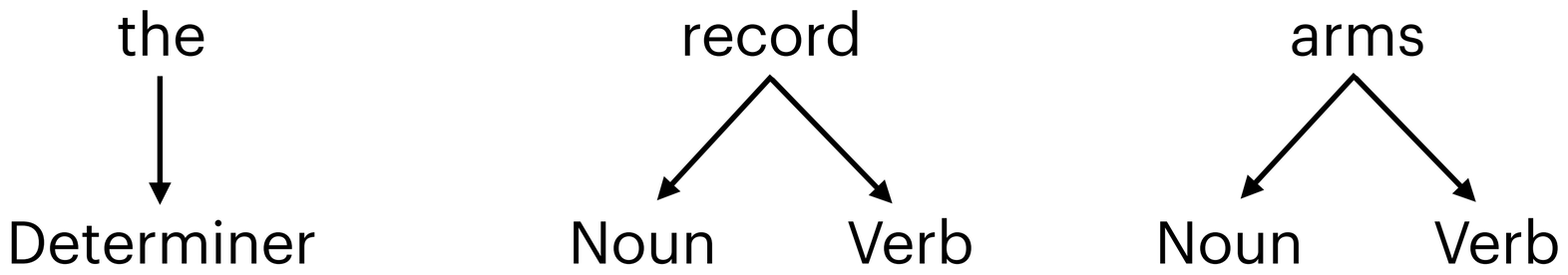
Hidden Markov Models (HMMs) are common sequence labeling methods.

The **Viterbi algorithm** allows us to efficiently decode from HMMs.



Parts of Speech

- Words can be classified into grammatical categories **[Yaska & Panini, 5th c. BC; Aristotle, 4th c. BC]**
- 8 parts of speech attributed to **Dionysius Thrax [1st c. BC]:**
  - nouns, verbs, pronouns, prepositions, adverbs, conjunctions, participles, adverbs



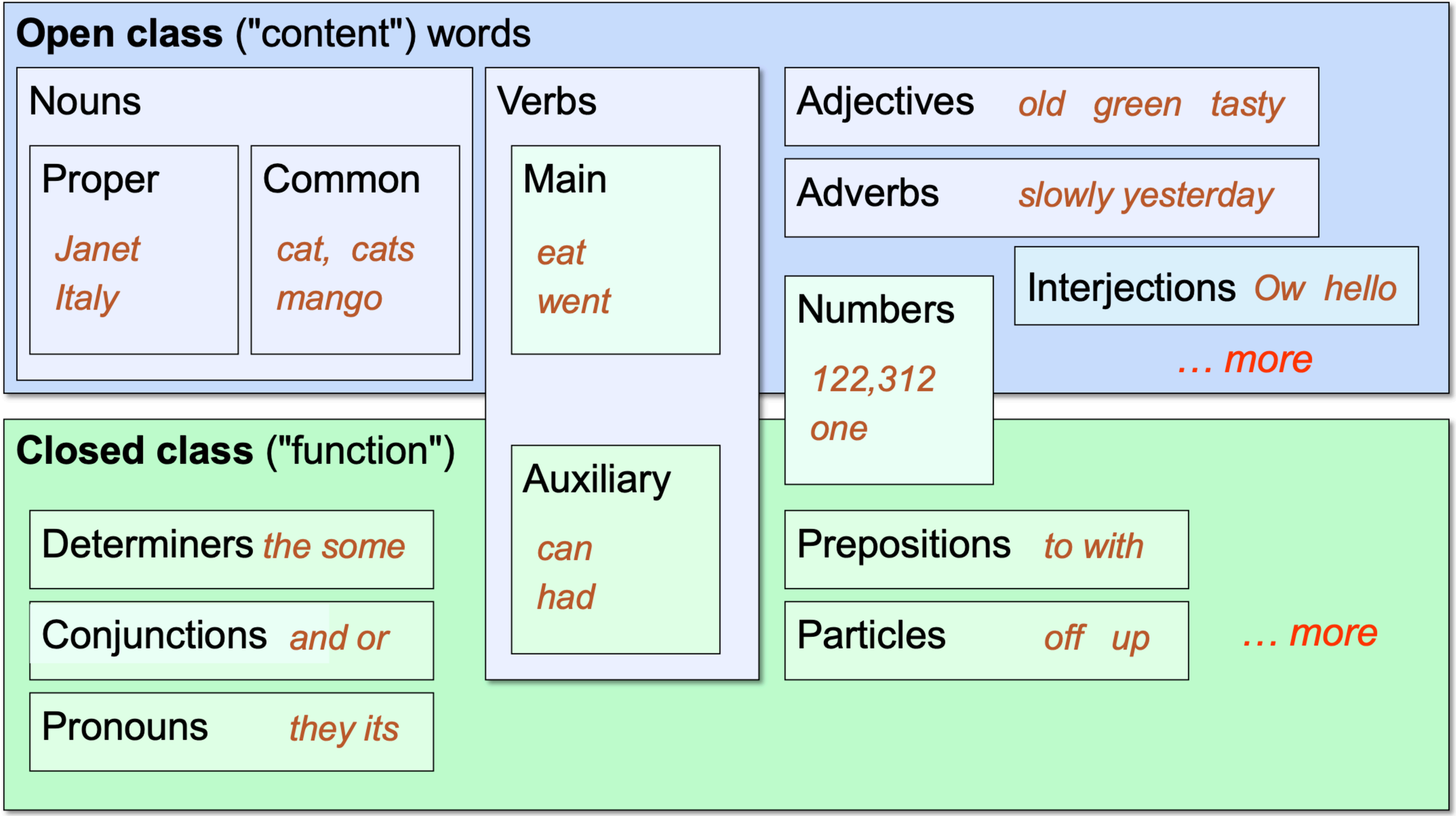
Open vs. Closed Classes

Closed class words:

- Fixed membership
- Usually **function words**: short and frequent terms with a specific grammatical function
    - Determiners: the, a, an
    - Pronouns: they, I, we
    - Prepositions: on, under, over, near, by

Open class words:

- Usually **content words**: nouns, verbs, adjectives, adverbs
    - Plus interjections; oh, um, yep, hello
- New items can be added to these classes.
    - Recent examples: to fax, iPhone


Open class ("content") words
Nouns
Proper
Janet
Italy
Common
cat, cats
mango
Verbs
Main
eat
went
Auxiliary
can
had
Adjectives old green tasty
Adverbs slowly yesterday
Numbers
122,312
one
Interjections Ow hello
… more
Closed class ("function")
Determiners the some
Conjunctions and or
Pronouns they its
Prepositions to with
Particles off up
… more

Word Subclasses

Nouns

- Parts of speech go far deeper than nouns, verbs, etc.
- Nouns describe entities and concepts. They are diverse:
  - **Common nouns**: the default. Examples: “cat”, “mango”
  - **Count nouns**: have a plural and need an article in the singular. Examples: “dog”, “banana”
  - **Mass nouns**: cannot be counted. Examples: “water”, “communism”
  - **Proper nouns:** names of specific people or entities. Examples: “IBM”, “Boston”

Penn Treebank tags:

NN: singular or mass common noun

NNS: plural common noun

NNP: singular proper noun

NNPS: plural proper noun

Word Subclasses

Verbs

- Verbs are also a diverse class (in English):
  - **Modal verbs:** often mark likelihood or mood. Examples: “can”, “might”
  - **Auxiliary verbs:** marks grammatical features. Example: “has left”, “will leave”
  - **Copula**: “to be” with a predicate. Examples: “she is hungry”, “they are interesting”
  - **(Full) Verbs**: most verbs. Examples: “eat”, “write”, “sleep”
- Verbs have inflections: infinitive (to eat), present tense (I eat), 3rd person singular (he/she eats), past tense (ate), present participle (eating), past participle (eaten)

Penn Treebank tags:

VB: infinitive or base form

VBD: past tense

VBN: past participle

MD: modal verbs

VBG: present participle

VBP: present tense (non 3rd person singular)

VBZ: present tense (3rd person singular)

Part-of-speech Tags

We'll use standard labels from the **Penn Treebank** — 36 tags

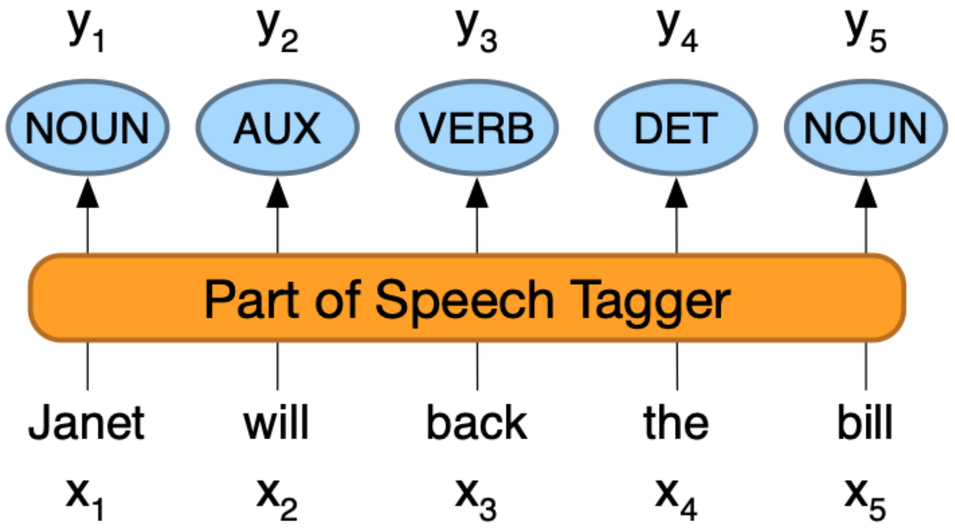
Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coord. conj.	*and, but, or*	NNP	proper noun, sing.	*IBM*	TO	infinitive to	*to*
CD	cardinal number	*one, two*	NNPS	proper noun, plu.	*Carolinas*	UH	interjection	*ah, oops*
DT	determiner	*a, the*	NNS	noun, plural	*llamas*	VB	verb base	*eat*
EX	existential 'there'	*there*	PDT	predeterminer	*all, both*	VBD	verb past tense	*ate*
FW	foreign word	*mea culpa*	POS	possessive ending	*'s*	VBG	verb gerund	*eating*
IN	preposition/ subordin-conj	*of, in, by*	PRP	personal pronoun	*I, you, he*	VBN	verb past participle	*eaten*
JJ	adjective	*yellow*	PRP$	possess. pronoun	*your*	VBP	verb non-3sg-pr	*eat*
JJR	comparative adj	*bigger*	RB	adverb	*quickly*	VBZ	verb 3sg pres	*eats*
JJS	superlative adj	*wildest*	RBR	comparative adv	*faster*	WDT	wh-determ.	*which, that*
LS	list item marker	*1, 2, One*	RBS	superlatv. adv	*fastest*	WP	wh-pronoun	*what, who*
MD	modal	*can, should*	RP	particle	*up, off*	WP$	wh-possess.	*whose*
NN	sing or mass noun	*llama*	SYM	symbol	+, %, &	WRB	wh-adverb	*how, where*

Part-of-speech Tagging

There	are	100	students	in	class	!
EX	**VBP**	**CD**	**NNS**	**IN**	**NN**	**.**

POS tagging involves assigning a part of speech y_i to every word x_i in a sequence:

$$(x_1, \ldots, x_n) \rightarrow (y_1, \ldots, y_n)$$



Why do we care about part of speech tagging?

- Tags can be useful for NLP tasks—especially with statistical systems.
  - Parsing
  - Machine translation: reordering adjectives and nouns (e.g., French and English have opposite orderings)
  - Sentiment classification: may want to treat adjectives or other POS differently
  - Text-to-speech: how should we pronounce “lead” or “record”?

- Computational linguistics tasks:
  - Need to control for POS when analyzing new words or meaning shifts
  - Need to control for POS when measuring meaning (dis)similarities

Parts of speech are usually not obvious!

The *majority* of words in most datasets could be assigned to more than one part of speech! (In context, there is usually only one possibility.)

		WSJ	Brown
Types:			
Unambiguous	(1 tag)	44,432 (**86%**)	45,799 (**85%**)
Ambiguous	(2+ tags)	7,025 (**14%**)	8,050 (**15%**)
Tokens:			
Unambiguous	(1 tag)	577,421 (**45%**)	384,349 (**33%**)
Ambiguous	(2+ tags)	711,780 (**55%**)	786,646 (**67%**)

Book me a flight.

VBP

I'm reading an interesting book.

NN

earnings growth took a **back/JJ** seat
a small building in the **back/NN**
a clear majority of senators **back/VBP** the bill
Dave began to **back/VB** toward the door
enable the country to buy **back/RP** debt
I was twenty-one **back/RB** then

Some POS Tagging Baselines

- Baseline: just assign every word to its most frequent POS. Assign unknown words to the most frequent class (nouns).
    - 90% accuracy on Penn Treebank
    - “Most frequent class” baseline is important for many tasks
    - Partly easy because many words are unambiguous.

- Better approach: use a **Hidden Markov model** (HMM)
    - 95% accuracy
    - Hasn’t changed much in the past decade
    - Human accuracy is about the same

- Is POS tagging solved? Kind of, but not for languages with more complex morphology, nor for non-standard English domains.

Sources of Information in POS Tagging

I will read an interesting book.

AUX/NN/VB? NN/VB?

- Prior probability of word/tag pair
  - “will” is usually an AUX
- Identity of neighboring words
  - “the” means that the next word is probably not a verb
- Morphology:
  - Prefixes
  - Suffixes
  - Capitalization

unable:	**un-**	→ JJ	(adjective)
important**ly**:	**-ly**	→ RB	(adverb)
Janet:	**CAP**	→ NNP	(proper noun)

A Part-of-speech Classifier

- Let's start by implementing a simple classifier based on ideas we've already used.
- Given all possible POS tags Y, input sequence $\mathbf{x}$, we can implement classifier f:

$$f : p(y_i = t \,|\, x_i)$$

Fed raises interest rates

x_2

- We can also take positional info into account:

index of token (e.g., 2)

$$f_2 : p(y_i = t \,|\, x_i, i)$$

Problems with Our Classifier

- The classifier would probably predict $(y_2, y_3) =$ (VBS, VBP)
  - *Invalid combo!*
- So we don't want independent token classifiers. We want a **sequence model:**

$$p(\mathbf{y} \mid \mathbf{x})$$

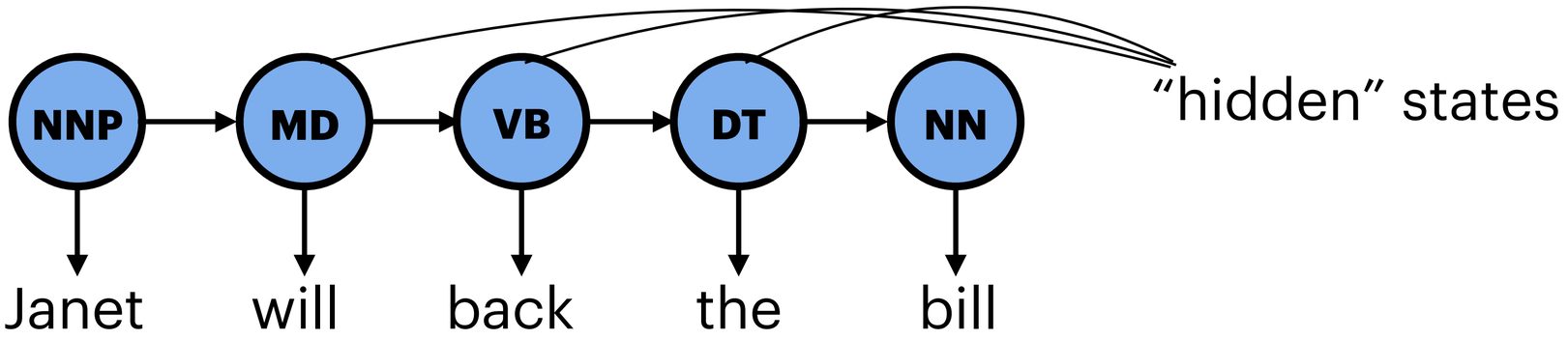
models the probability of the entire label sequence

Hidden Markov Models (HMMs)

- HMMs are kind of like bigram sequence models:

$$p(y_i \mid y_{i-1})$$

- Remember the **Markov assumption** from a while back: we just ignore most of the prior context and only condition on the current state
- Captures **transition probabilities**: how likely is this label given the previous one?



Sequence Modeling with HMMs

$$p(\mathbf{y} \mid \mathbf{x}) = p(y_1)p(x_1 \mid y_1)p(y_2 \mid y_1)p(x_2 \mid y_2)\ldots p(\text{STOP} \mid y_n)$$

Initial probability

Emission probabilities

Transition probabilities

- HMMs require the following to be specified:
  - **States**: $Q = q_1, q_2, \ldots, q_N$. E.g., N possible POS tags
  - **Transition probabilities**: For all pairs of states (q_i, q_j), what is $p(q_j \mid q_i)$?
  - **Emission probabilities**: What is the probability of an observation o_i given a state q_i; i.e., $p(o_i \mid q_i)$?
  - **Initial probabilities**: For all i, π_i is the probability that the sequence will start with state q_i

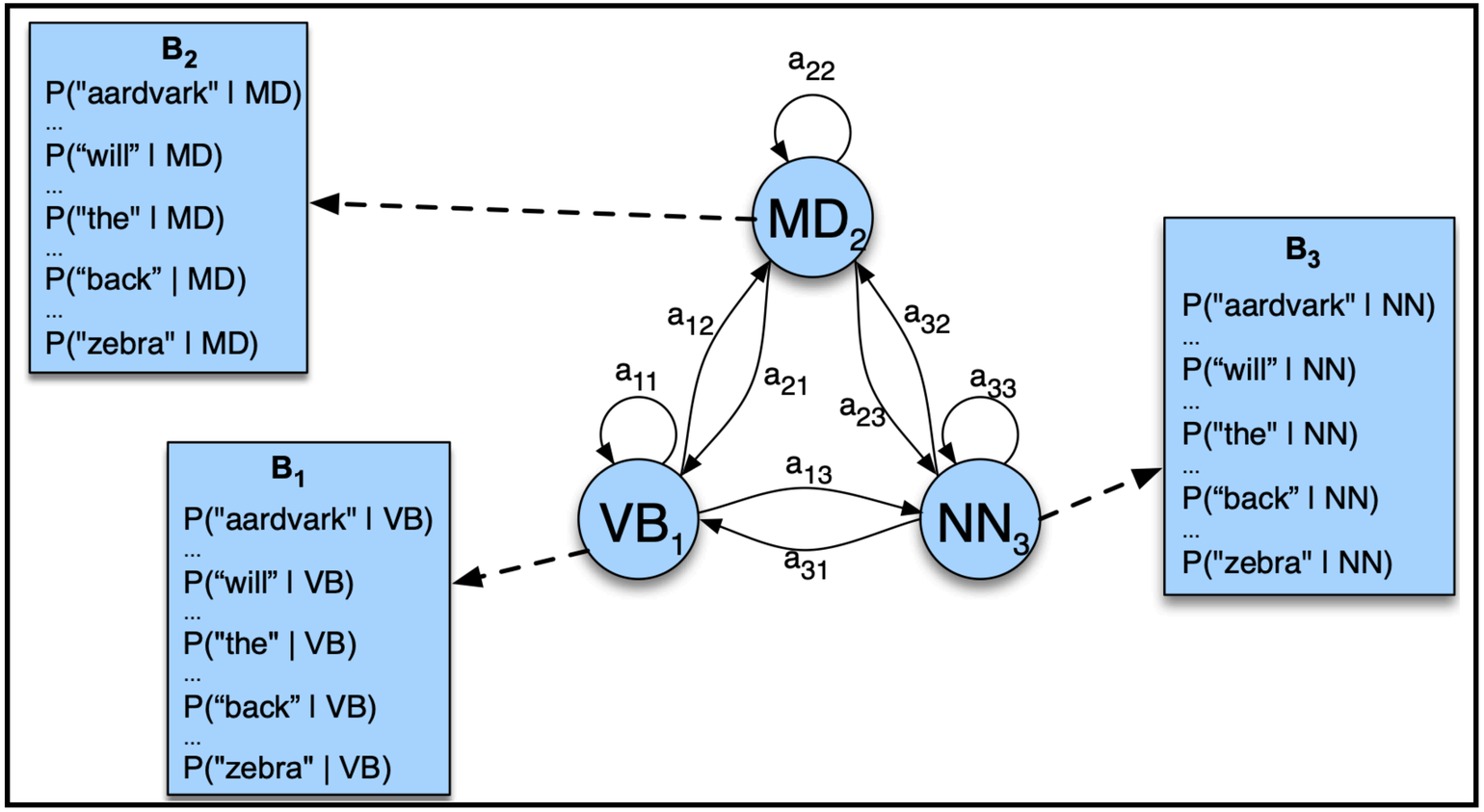
Estimating HMM Parameters

- We must learn two components: the A matrix and B matrix.

- A: transition probabilities

- B: emission probabilities

- In reality, this figure should have one state for each possible tag.



Example HMM

Transition Matrix A

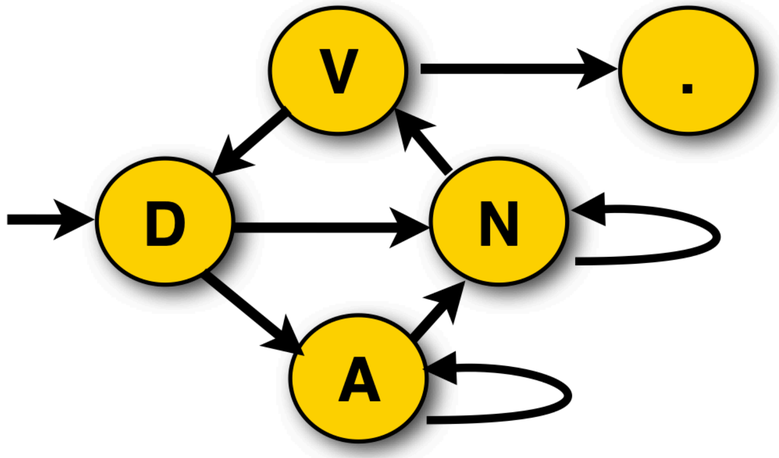
	D	N	V	A	.
D		0.8		0.2	
N		0.7	0.3		
V	0.6				0.4
A		0.8		0.2	
.					

Emission Matrix B

	the	*man*	*ball*	*throws*	*sees*	*red*	*blue*	.
D	1							
N		0.7	0.3					
V				0.6	0.4			
A						0.8	0.2	
.								1

Initial state vector π

	D	N	V	A	.
π	1				



Building an HMM

To build an HMM POS tagger, we need to:

1. **Train** the model, i.e. estimate its parameters (transition and emission probabilities)
   - Easy when we have a dataset labeled with POS tags (supervised learning)
   - Hard when we have raw text without tags (unsupervised learning)

2. Define a **decoding** algorithm that can generate tag sequences given a sequence of words.

Estimating HMM Parameters

A matrix

```
Pierre_NNP Vinken_NNP ,_, 61_CD years_NNS
old_JJ ,_, will_MD join_VB the_DT board_NN
as_IN a_DT nonexecutive_JJ director_NN Nov._NNP
29_CD ._.
```

- To learn transition probabilities, we'll use the same technique that we used for bigram language modeling:

$$p(t_i \mid t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

Tag transition probability

Count of tag pair

Count of previous tag

- For example, modal verbs MD occur 13124 times in the Wall Street Journal corpus. 10471 of these times, it is followed by a VB. So:

$$p(\text{VB} \mid \text{MD}) = \frac{10471}{13124} = 0.80$$

Estimating HMM Parameters

Initial Probabilities

- To get the initial probabilities π_i, just count the number of times each tag starts a sequence:

$$p(t_i \mid \text{START}) = \frac{C(\text{START}, t_i)}{C(\text{START})}$$

Initial probability π_i

Count of tag at starts of sequences

Number of documents in training set

Estimating HMM Parameters

B matrix

- To learn emission probabilities, we will compute:

$$p(w_i \mid t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

Emission probability

Count of tag/word pair

Count of tag

- For example, of the 13124 modal verbs MD in the Wall Street Journal corpus, 4046 of these are the word *will*:

$$p(\text{will} \mid \text{MD}) = \frac{4046}{13124} = 0.31$$

HMMs are generative models.

⚠️ Caution: this equation might be backwards from what you were expecting!

$$p(w_i \mid t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

- This term is asking "If we know we're going to generate tag t_i, how likely is it that the word is going to be w_i?"
  - This term is *not* asking "what is the most likely tag for this word?"
- Why do we do this? It's because HMMs are **generative models**; i.e., they model $p(x, y)$.
- Our classifiers were **discriminative models**; i.e., they modeled $p(y \mid x)$.

Decoding from an HMM

Definitions

- **Decoding:** the process of determining the hidden variables corresponding to a given sequence of observations.
- In our case, given:

$$\lambda = (A, B)$$ — Our learned HMM

$$W = w_1, w_2, \ldots, w_T$$ — A sequence of observations (words)

- We want to find:

$$Q = t_1, t_2, \ldots, t_T$$ — A sequence of states (tags)

Decoding from an HMM

Method, Pt. I

Goal: compute

But we don't have $p(t_i \mid w_i)$!

$$\hat{t}_{1:n} = \arg\max_{t_1,\ldots,t_n} p(t_1, \ldots, t_n \mid w_1, \ldots, w_n)$$

In practice, we use Bayes' rule to instead compute:

$$\hat{t}_{1:n} = \arg\max_{t_1,\ldots,t_n} \frac{p(w_1, \ldots, w_n \mid t_1, \ldots, t_n)p(t_1, \ldots, t_n)}{p(w_1, \ldots, w_n)}$$

The denominator is constant for any given prediction, so we can drop it:

$$\hat{t}_{1:n} = \arg\max_{t_1,\ldots,t_n} p(w_1, \ldots, w_n \mid t_1, \ldots, t_n)p(t_1, \ldots, t_n)$$

Decoding from an HMM

Method, Pt. II

$$\hat{t}_{1:n} = \arg\max_{t_1,\ldots,t_n} p(w_1, \ldots, w_n \mid t_1, \ldots, t_n) p(t_1, \ldots, t_n)$$

We'll make a couple more simplifying assumptions. We'll assume that the probability of a word depends only on its own tag (i.e., it's independent of its neighbors):

$$p(w_1, \ldots, w_n \mid t_1, \ldots, t_n) \approx \prod_{i=1}^{n} p(w_i \mid t_i)$$

The second is our old friend, the **Markov assumption**:

$$p(t_1, \ldots, t_n) \approx \prod_{i=1}^{n} p(t_i \mid t_{i-1})$$

Decoding from an HMM

Method, Pt. III

$$\hat{t}_{1:n} = \arg\max_{t_1,\ldots,t_n} p(w_1, \ldots, w_n \mid t_1, \ldots, t_n) p(t_1, \ldots, t_n)$$

$$\approx \arg\max_{t_1,\ldots,t_n} \prod_{i=1}^{n} p(w_i \mid t_i) p(t_i \mid t_{i-1})$$

Emission probabilities

Transition probabilities

These two probabilities correspond to the A and B matrices we've learned!

Efficiently Decoding from an HMM

- Naively, decoding from an HMM is impossibly slow: it requires us to loop over all possible sequences of states.
  - $O(T^n)$ time (exponential)!

- We instead use an efficient dynamic programming algorithm called the **Viterbi algorithm.**
  - $O(nT^2)$ time
  - Basic idea: memoize most likely tags for each subsequence from 0 to t ending in state k. Solve larger problems by composing subsolutions.

Viterbi Algorithm

- Basic idea: memoize most likely tags for each subsequence from 0 to t ending in state k. Solve larger problems by composing subsolutions.
- Base case:

initial probability

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

- Recursion:

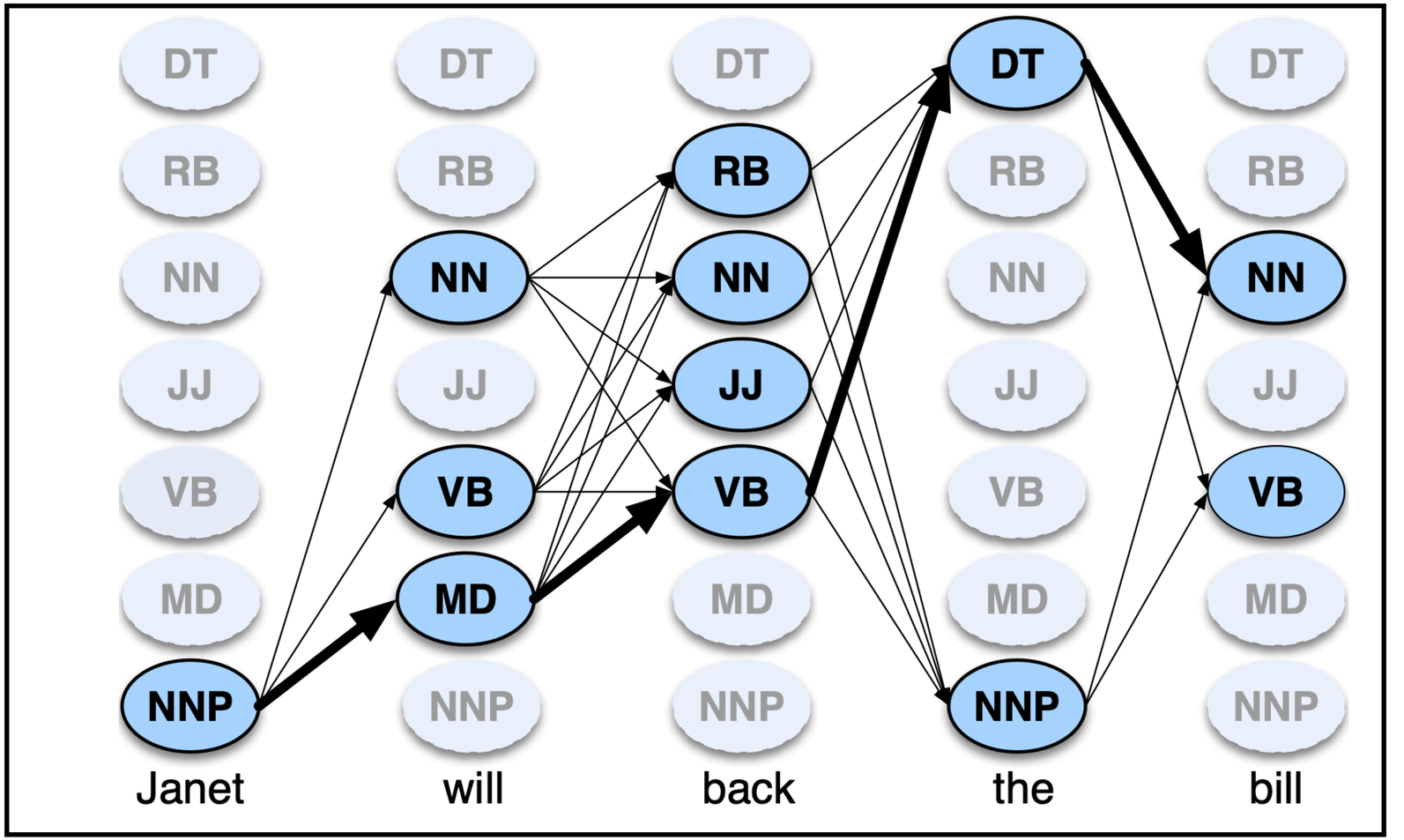
emission probability

$$\delta_n(k) = \max_j (\delta_{n-1}(j) \cdot p(q_k \mid q_j)) p(o_n \mid q_k)$$

backpointer to previous state

transition probability

Viterbi Algorithm



We'll keep a *lattice* of all possible tags for each word.

Example

We want to get the correct tag sequence for this word sequence:

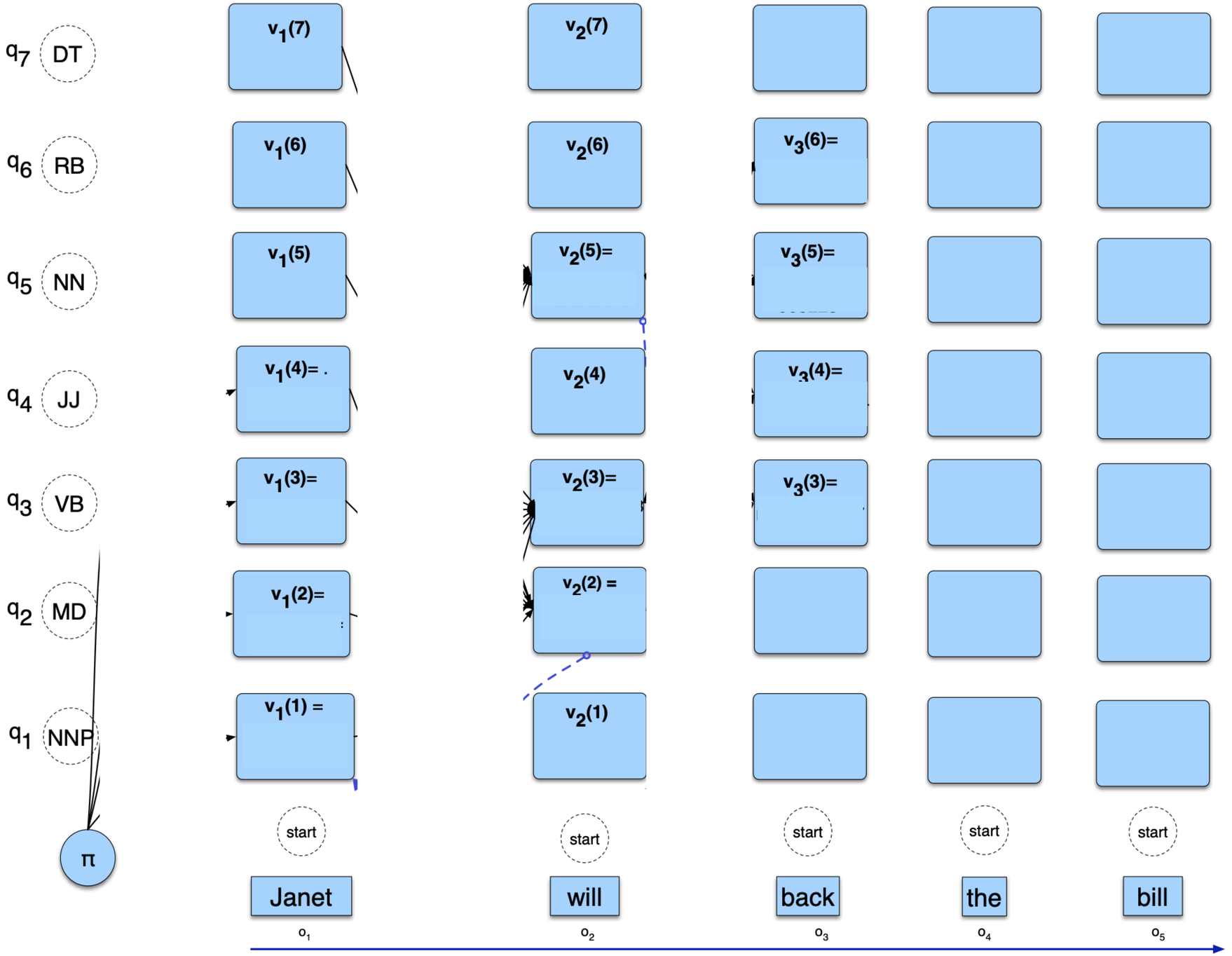
Janet	will	back	the	bill
NNP	**MD**	**VB**	**DT**	**NN**

Say we've precomputed these transition probs:

	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

And these emission probs:

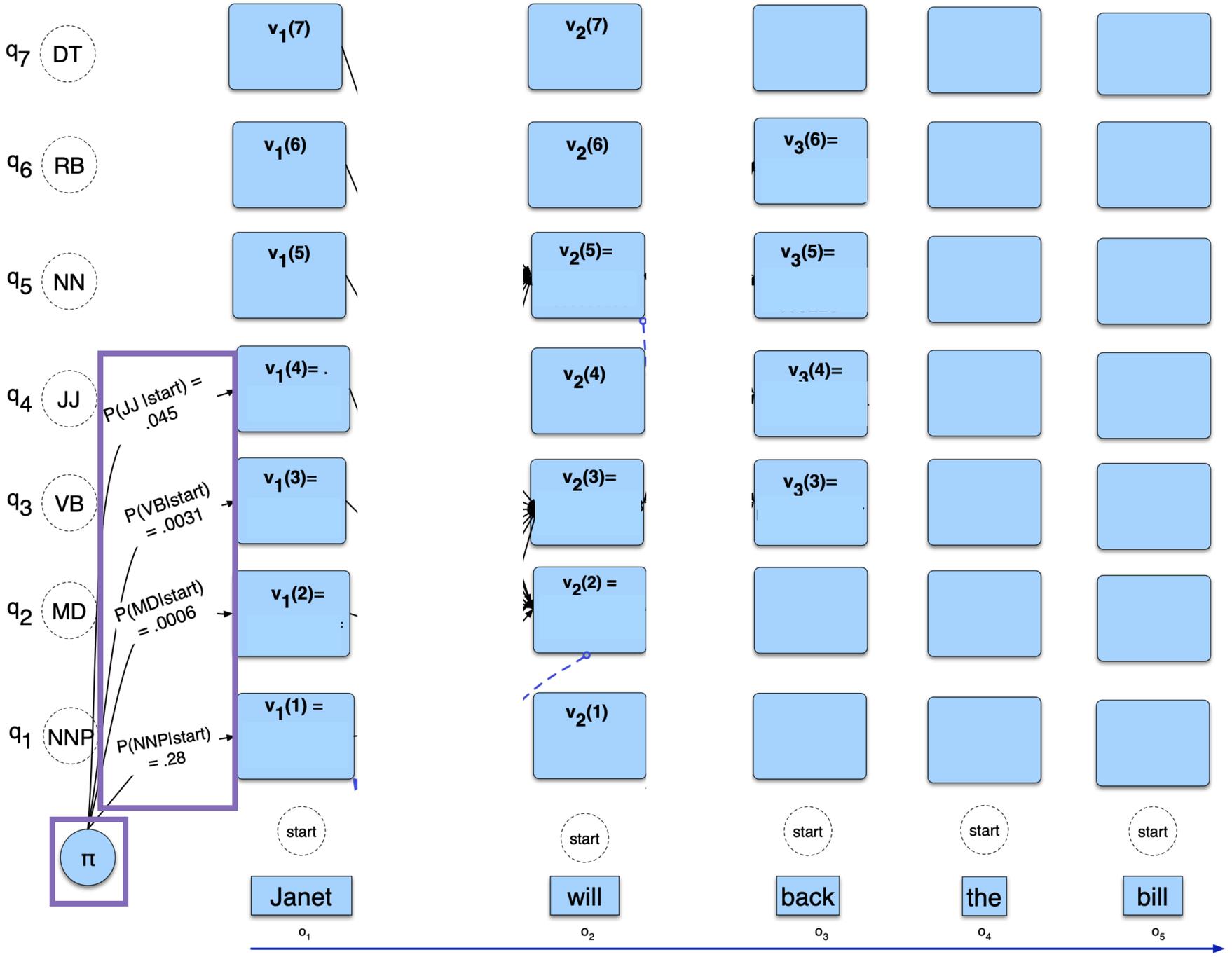
	Janet	**will**	**back**	**the**	**bill**
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0


q_7 DT
q_6 RB
q_5 NN
q_4 JJ
q_3 VB
q_2 MD
q_1 NNP
π
$v_1(7)$
$v_1(6)$
$v_1(5)$
$v_1(4)=$
$v_1(3)=$
$v_1(2)=$
$v_1(1) =$
$v_2(7)$
$v_2(6)$
$v_2(5)=$
$v_2(4)$
$v_2(3)=$
$v_2(2) =$
$v_2(1)$
$v_3(6)=$
$v_3(5)=$
$v_3(4)=$
$v_3(3)=$
start
start
start
start
start
Janet
will
back
the
bill
o_1
o_2
o_3
o_4
o_5

First, our base case:

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

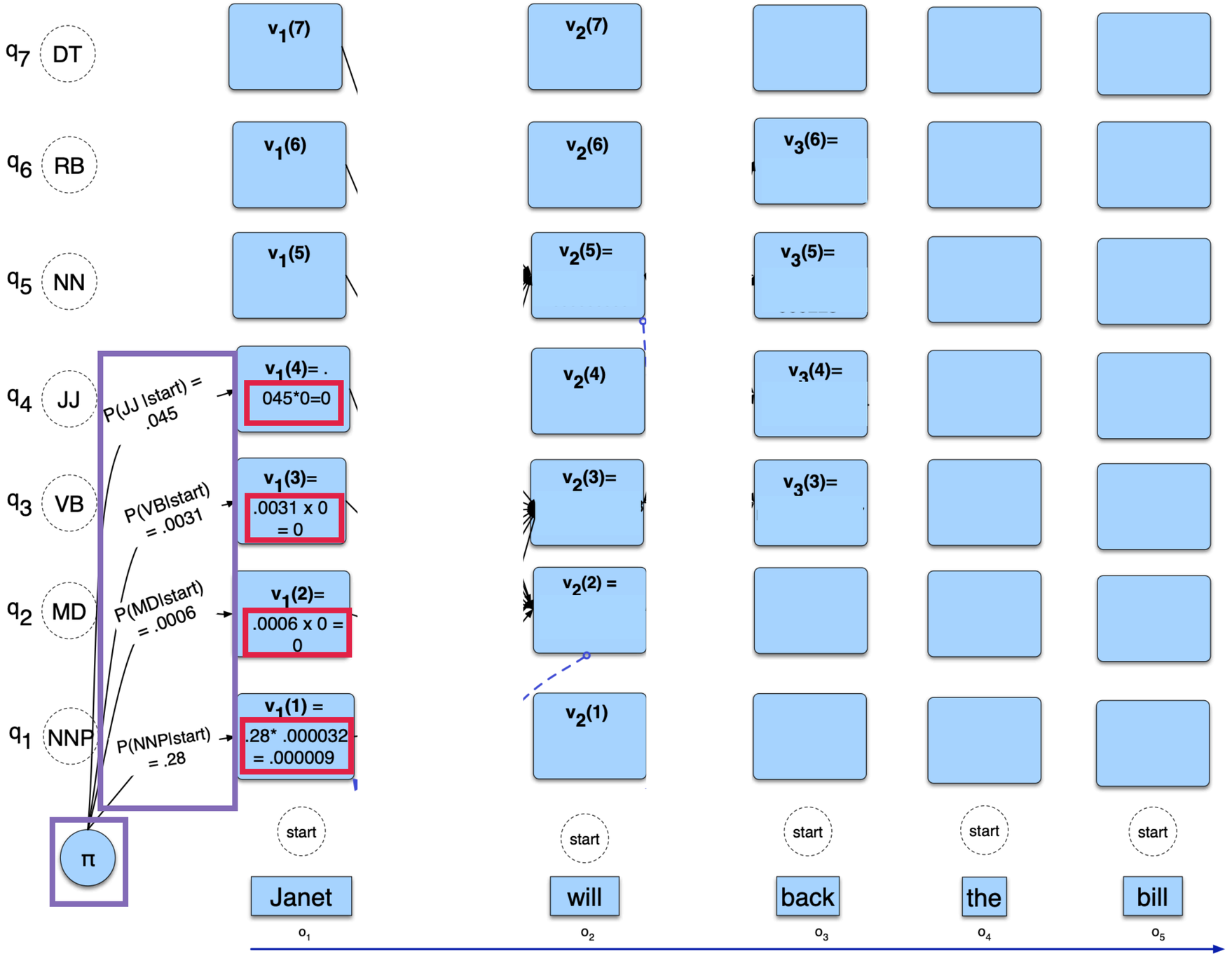


First, our base case:

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

Multiply each by our **emission probabilities**.

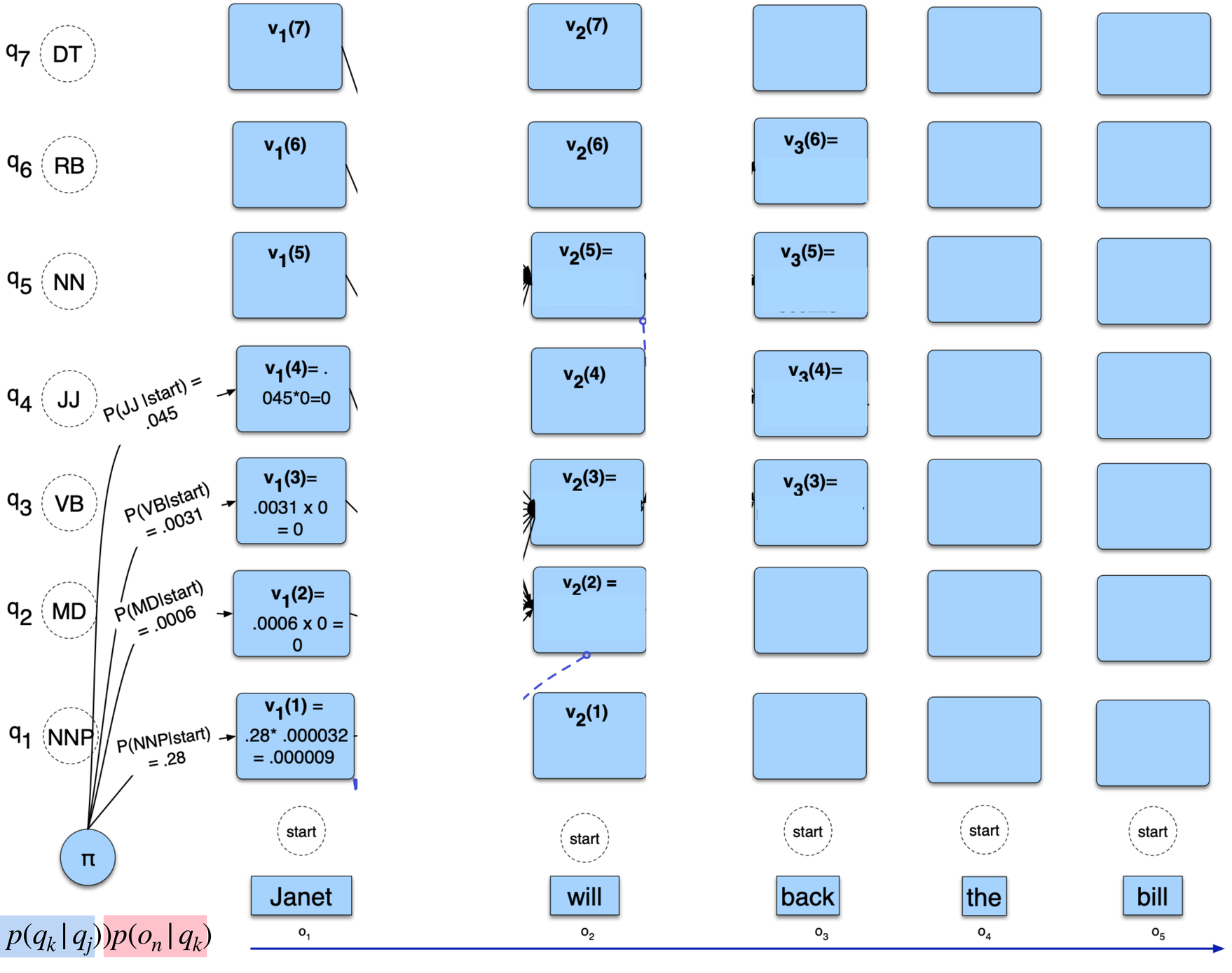


First, our base case:

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

Multiply each by our **emission probabilities**.



Now for our recursive step(s):

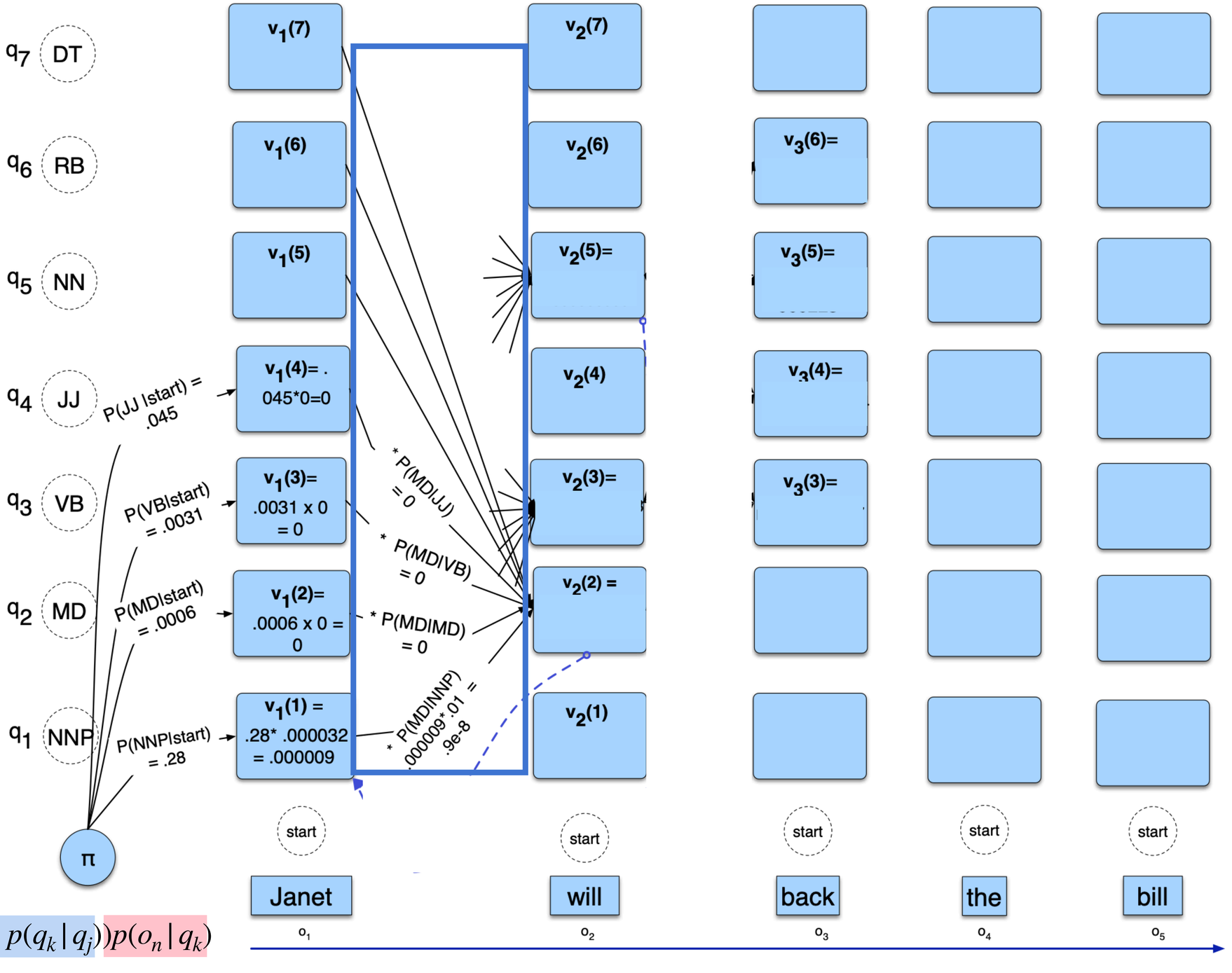
$$\delta_n(k) = \max_j(\delta_{n-1}(j) \cdot p(q_k \mid q_j))p(o_n \mid q_k)$$

First, our base case:

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

Multiply each by our **emission probabilities**.



Get our **transition probabilities**.

Now for our recursive step(s):

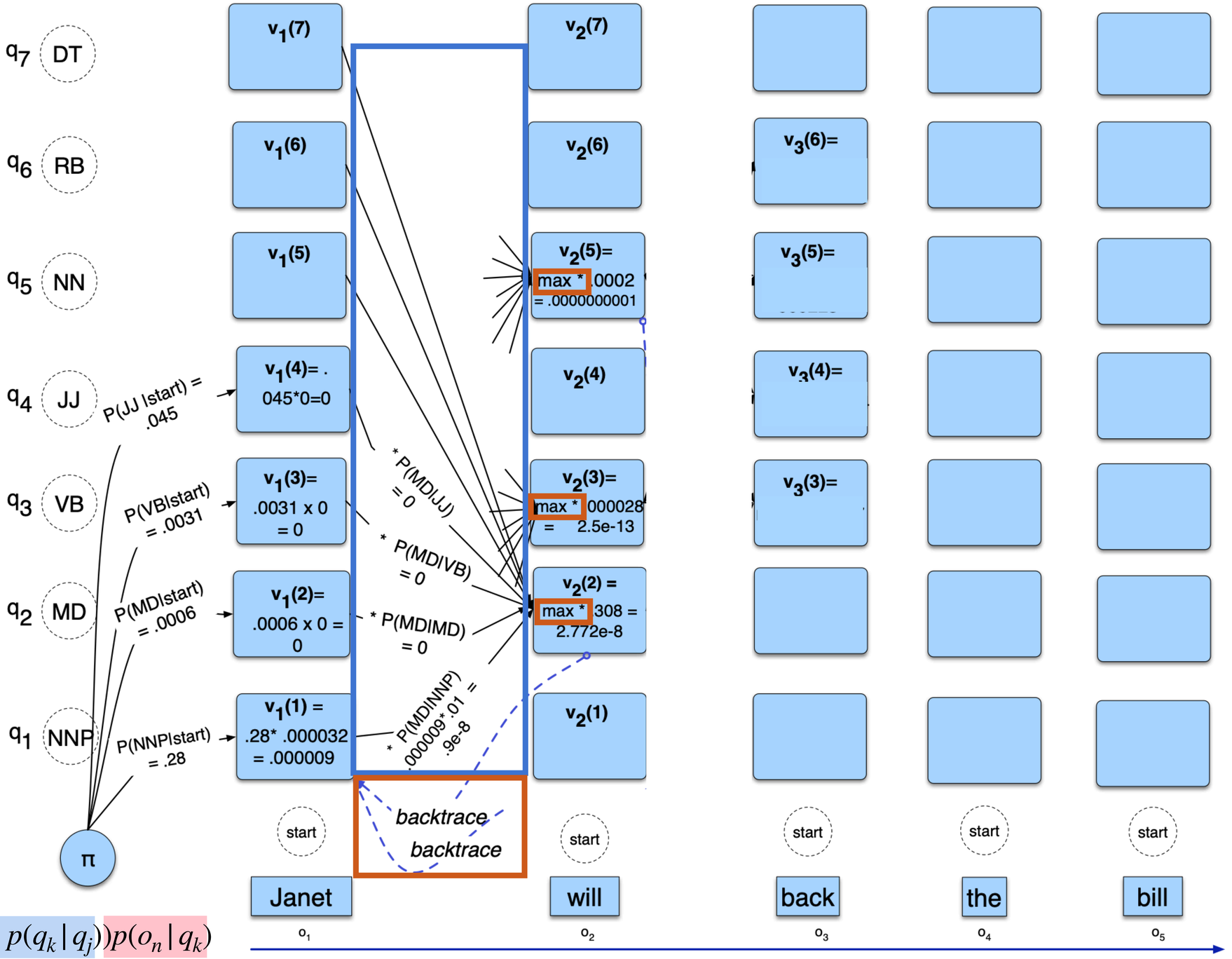
$$\delta_n(k) = \max_j(\delta_{n-1}(j) \cdot p(q_k \mid q_j))p(o_n \mid q_k)$$

First, our base case:

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

Multiply each by our **emission probabilities**.



Get our **transition probabilities**.

Backtrace to most likely previous state; multiply that probability with our new one.

Now for our recursive step(s):

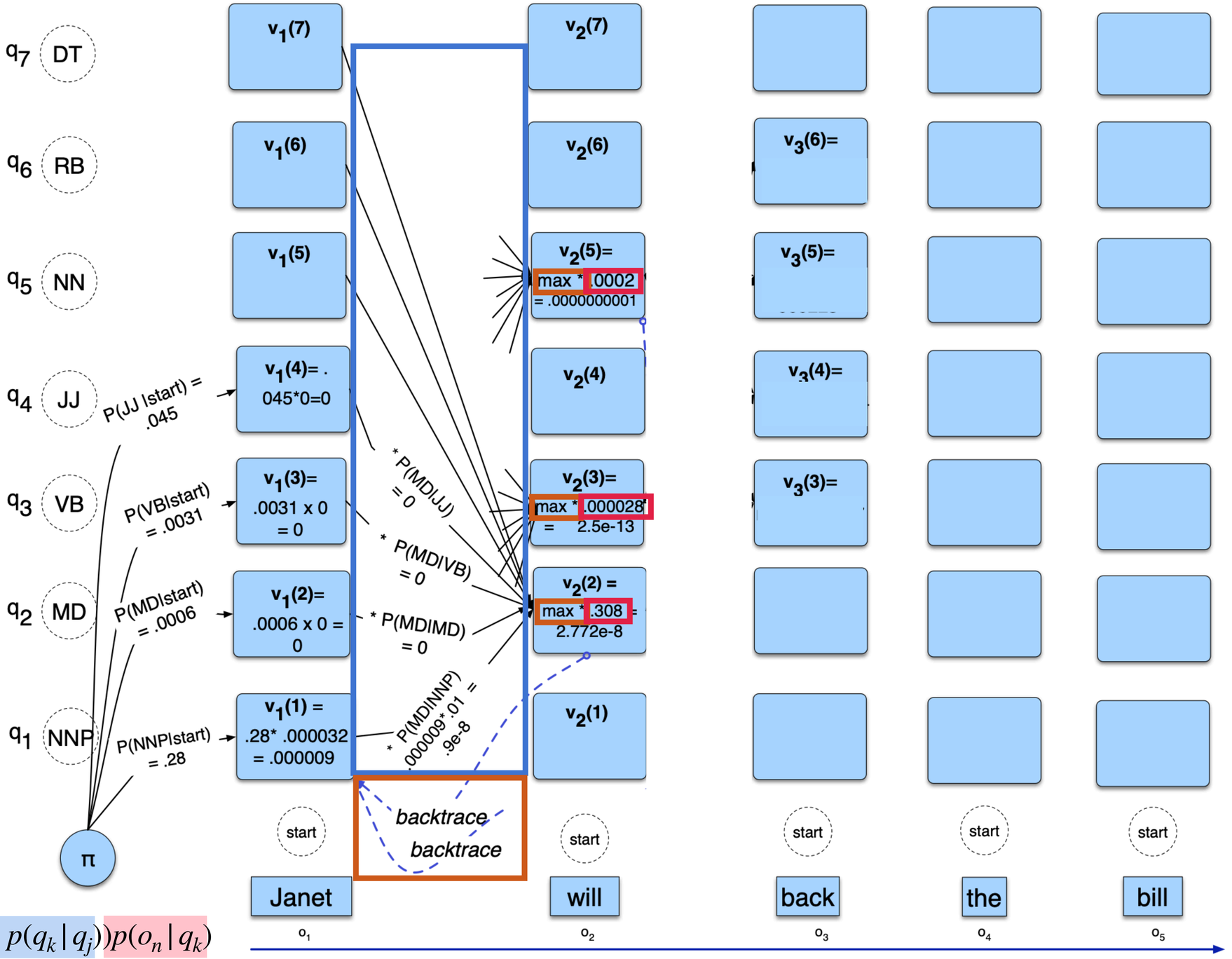
$$\delta_n(k) = \max_j(\delta_{n-1}(j) \cdot p(q_k \mid q_j))p(o_n \mid q_k)$$

First, our base case:

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

Multiply each by our **emission probabilities**.



Get our **transition probabilities**.

Backtrace to most likely previous state; multiply that probability with our new one.

Multiply each (backtrace_prob * transition_prob) term by the **emission probability** for the current word.

Now for our recursive step(s):

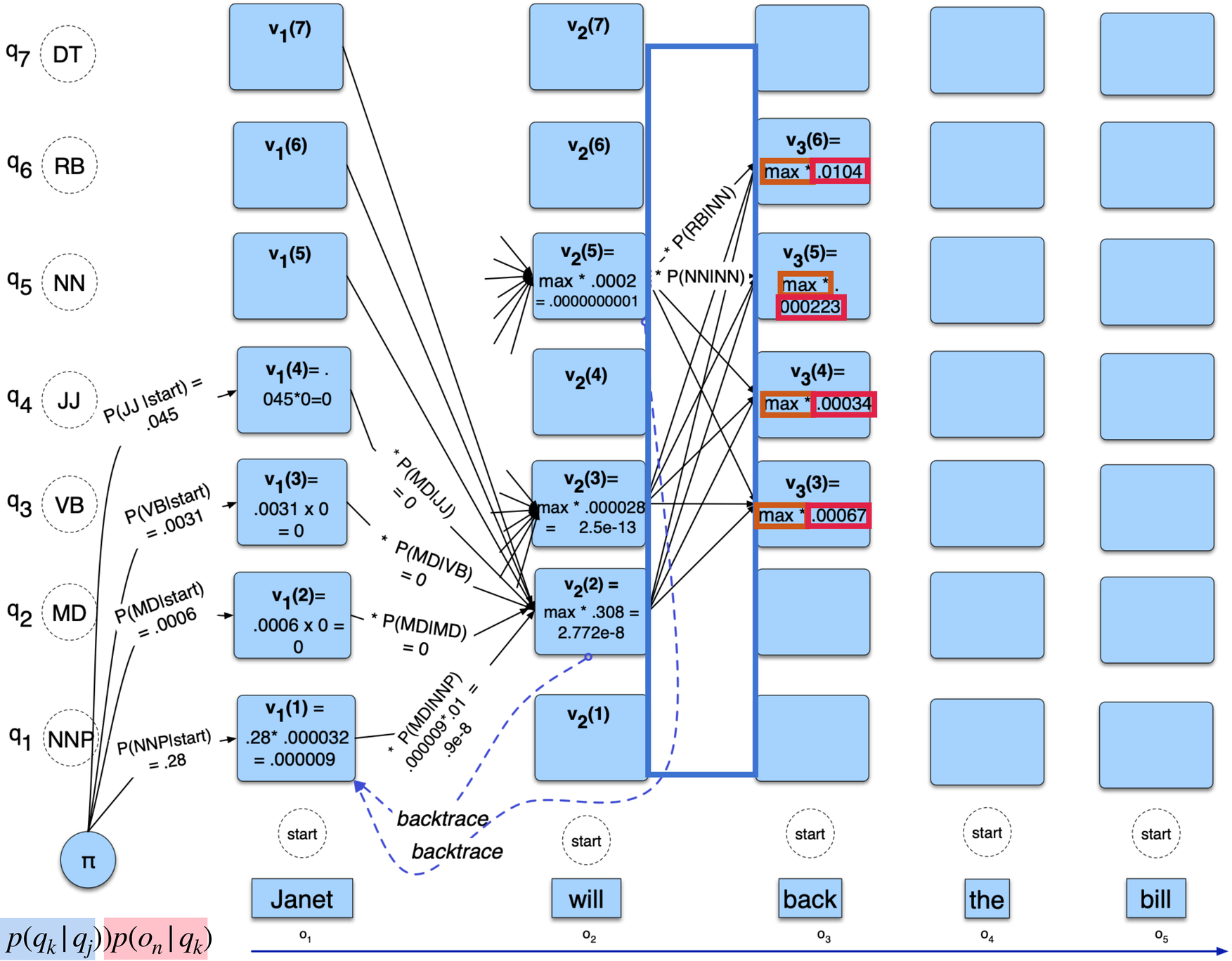
$$\delta_n(k) = \max_j(\delta_{n-1}(j) \cdot p(q_k \mid q_j))p(o_n \mid q_k)$$

First, our base case:

$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

Multiply each by our **emission probabilities**.



Get our **transition probabilities**.

Backtrace to most likely previous state; multiply that probability with our new one.

Multiply each (backtrace_prob * transition_prob) term by the **emission probability** for the current word.

Now for our recursive step(s):

$$\delta_n(k) = \max_j(\delta_{n-1}(j) \cdot p(q_k \mid q_j))p(o_n \mid q_k)$$

First, our base case:

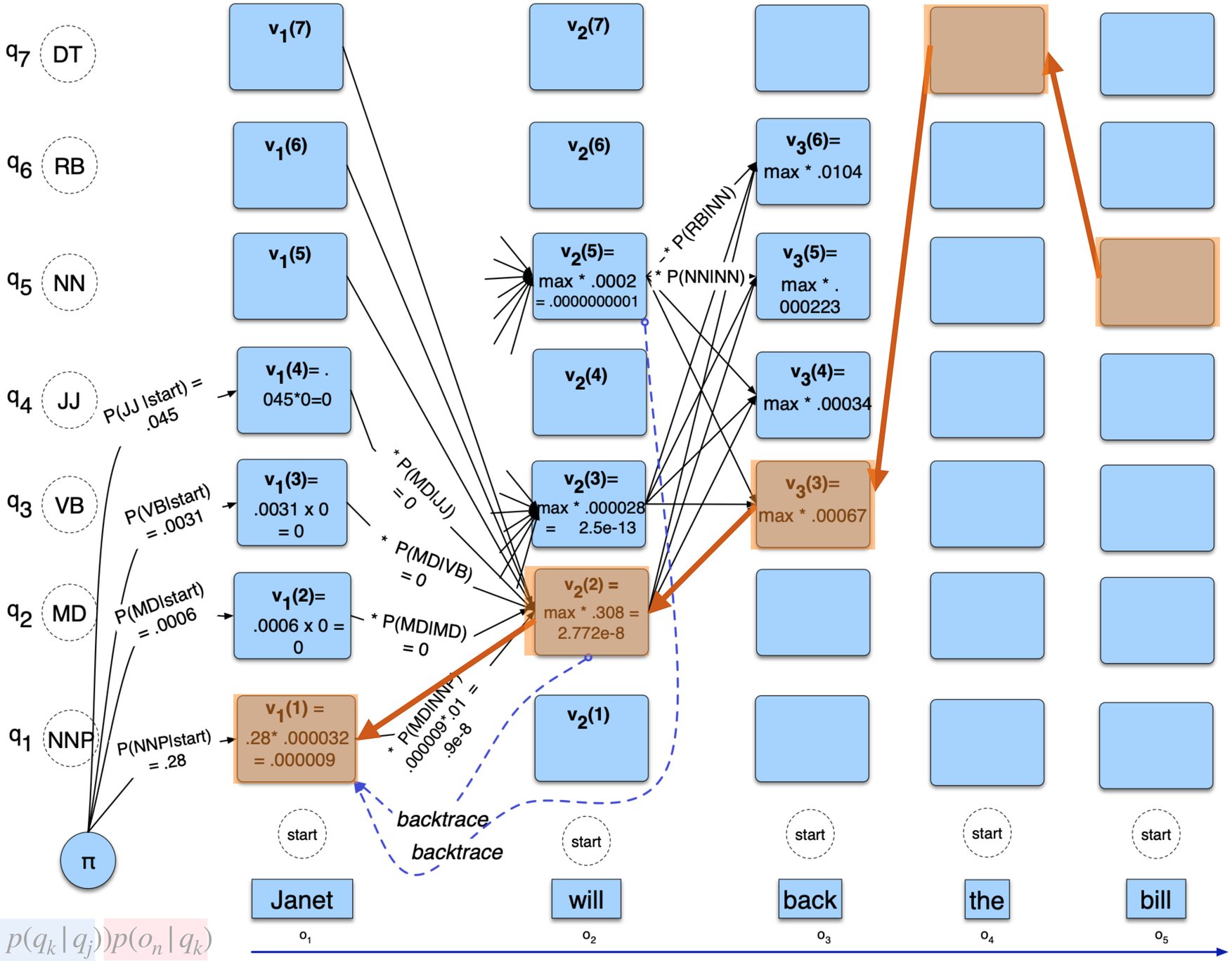
$$\delta_1(k) = \pi_k p(o_1 \mid q_k)$$

Get **initial probabilites**.

Multiply each by our **emission probabilities**.

Now for our recursive step(s):

$$\delta_n(k) = \max_j(\delta_{n-1}(j) \cdot p(q_k \mid q_j))p(o_n \mid q_k)$$



Get our **transition probabilities**.

Backtrace to most likely previous state; multiply that probability with our new one.

Multiply each (backtrace_prob * transition_prob) term by the **emission probability** for the current word.

At the end, **backtrace** through the most probable full path; this will yield the tags.

Viterbi Algorithm

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

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

set up lattice

for each state s **from** 1 **to** N **do**
    ; initialization step
    $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
    $backpointer[s,1] \leftarrow 0$

base case

for each time step t **from** 2 **to** T **do**
    ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$

recursive step(s)

backtrace to get best tag sequence (and its probability)

$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step

$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step

$bestpath \leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time

return *bestpath*, *bestpathprob*

Performance Comparison

- Data: English Penn Treebank (44 tags)
- Assign each word its most frequent tag: 90% accuracy
- Trigram HMM: 95% accuracy
    - ($O(nT^3)$ time)
- BERT: 97.5% accuracy

- If we were looking at a low-resource language like Old English or Xhosa, HMMs would probably outperform BERT.

Common Errors in POS Tagging

	JJ	NN	NNP	NNPS	RB	RP	IN	VB	VBD	VBN	VBP	Total
JJ	0	**177**	**56**	0	**61**	2	5	10	15	**108**	0	488
NN	**244**	0	**103**	0	12	1	1	29	5	6	19	525
NNP	**107**	**106**	0	**132**	5	0	7	5	1	2	0	427
NNPS	1	0	**110**	0	0	0	0	0	0	0	0	142
RB	**72**	21	7	0	0	16	**138**	1	0	0	0	295
RP	0	0	0	0	**39**	0	**65**	0	0	0	0	104
IN	11	0	1	0	**169**	**103**	0	1	0	0	0	323
VB	17	**64**	9	0	2	0	1	0	4	7	**85**	189
VBD	10	5	3	0	0	0	0	3	0	**143**	2	166
VBN	**101**	3	3	0	0	0	0	3	**108**	0	1	221
VBP	5	34	3	1	1	0	2	**49**	6	3	0	104
Total	626	536	348	144	317	122	279	102	140	269	108	3651

JJ/NN NN
official knowledge
art gallery

VBD RP/IN DET NN
made up the story

RB VBD/VBN NNS
recently sold shares

Remaining Challenges in POS Tagging

Manning [2011]

- What will it take to get from 97% to 100%
- Lexicon gap (word not seen with that tag in training): 4.5% of errors
- Unknown word: 4.5%
- Possible to get right: 16% (many of these could be fixed with parsing)
- Difficult linguistics: 20%

VBD/VBP? *unclear without further context*

They set up absurd situations

- Underspecified/unclear, inconsistent or wrong labels: **58%**

JJ/VBN?

a $ 10 million fourth-quarter charge against discontinued operations

Remaining Challenges in POS Tagging

- Morphologically rich languages are usually harder to tag:

1. Yerdeki **izin** temizlenmesi gerek. **The trace** on the floor should be cleaned.	iz + Noun+A3sg+Pnon+Gen
2. Üzerinde parmak **izin** kalmiş. **Your** finger **print** is left on (it).	iz + Noun+A3sg+P2sg+Nom
3. Içeri girmek için **izin** alman gerekiyor. You need **permission** to enter.	izin + Noun+A3sg+Pnon+Nom

- Should each tag *set* be considered as one tag? Is this better modeled as a multi-label problem?

Named Entity Recognition

- **Named entity recognition (NER)** is an important and very common task.

- Also harder than POS tagging! Need to track:
  - Boundaries of named entitites
  - Types of named entities

Citing high fuel prices, [ORG **United Airlines**] said [TIME **Friday**] it has increased fares by [MONEY **$6**] per round trip on flights to some cities also served by lower-cost carriers. [ORG **American Airlines**], a unit of [ORG **AMR Corp.**], immediately matched the move, spokesman [PER **Tim Wagner**] said. [ORG **United**], a unit of [ORG **UAL Corp.**], said the increase took effect [TIME **Thursday**] and applies to most routes where it competes against discount carriers, such as [LOC **Chicago**] to [LOC **Dallas**] and [LOC **Denver**] to [LOC **San Francisco**].

Types of Named Entities

Type	Tag	Sample Categories	Example sentences
People	PER	people, characters	**Turing** is a giant of computer science.
Organization	ORG	companies, sports teams	The **IPCC** warned about the cyclone.
Location	LOC	regions, mountains, seas	**Mt. Sanitas** is in **Sunshine Canyon**.
Geo-Political Entity	GPE	countries, states	**Palo Alto** is raising the fees for parking.

Why is NER hard?

- Segmentation:
  - In POS tagging, no segmentation problem; every word has 1 tag.
  - In NER, we have to figure out the boundaries of each entity.

- Type ambiguity

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.

BIO Tagging

B: token that **b**egins a span

I: token **i**nside a span

O: token **o**utside any span

So we have two tags per entity type (**B** and **I**), plus a generic **O** tag.

$2n + 1$ tags, where n is number of entity types.

Words	BIO Label
Jane	B-PER
Villanueva	I-PER
of	O
United	B-ORG
Airlines	I-ORG
Holding	I-ORG
discussed	O
the	O
Chicago	B-LOC
route	O
.	O

BIO Tagging

Words	IO Label	BIO Label	BIOES Label
Jane	I-PER	B-PER	B-PER
Villanueva	I-PER	I-PER	E-PER
of	O	O	O
United	I-ORG	B-ORG	B-ORG
Airlines	I-ORG	I-ORG	I-ORG
Holding	I-ORG	I-ORG	E-ORG
discussed	O	O	O
the	O	O	O
Chicago	I-LOC	B-LOC	S-LOC
route	O	O	O
.	O	O	O

O	**O**	**O**	**O**	**B-PER**	**I-PER**	**I-PER**	**O**	**O**
They	read	books	by	Jolly	B.	Goode	after	lunch.

Common NER Methods

- Supervised machine learning
  - Hidden Markov models
  - Conditional random fields
  - Neural sequence models (RNNs, Transformers)
  - Encoders (BERT), fine-tuned

Evaluating Sequence Taggers

- POS taggers are typically evaluated with **accuracy**: how many tags did we get right?
- NER taggers are typically evaluated with **precision**, **recall**, and **F1**.
  - **Precision:** the number of correctly labeled entities compared to the number of sequences labeled as named entites
  - **Recall:** the number of correctly labeled entities compared to how many entities there were
  - **F1:** the harmonic mean of precision and recall

Takeaways

- Sequence labeling is a fairly common task in NLP.
  - POS tagging
  - Named entity recognition
  - Common but didn’t discuss: typo/error detection

- Hidden Markov models (HMMs) allow us to relatively quickly set up probabilistic sequence taggers.

- The Viterbi algorithm allows us to efficiently label sequences.