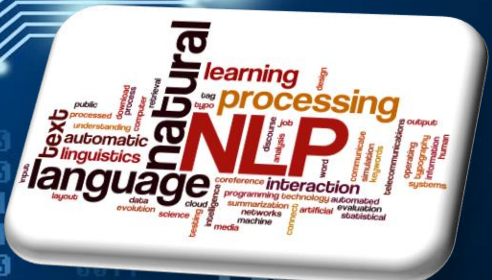
Cairo University

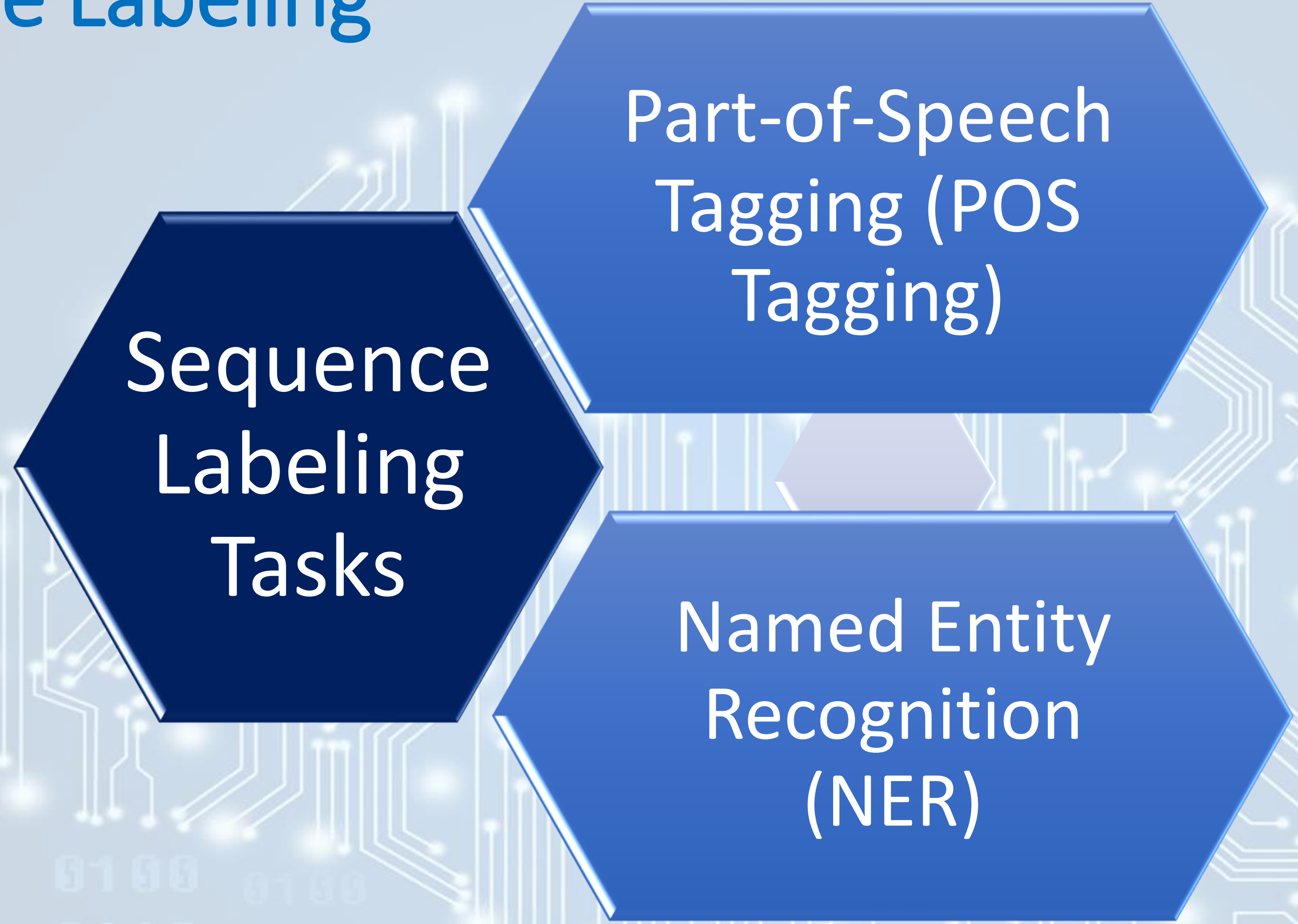
Cairo University
Faculty of Engineering
Computer Engineering Department

Natural Language Processing

Dr. Sandra Wahid



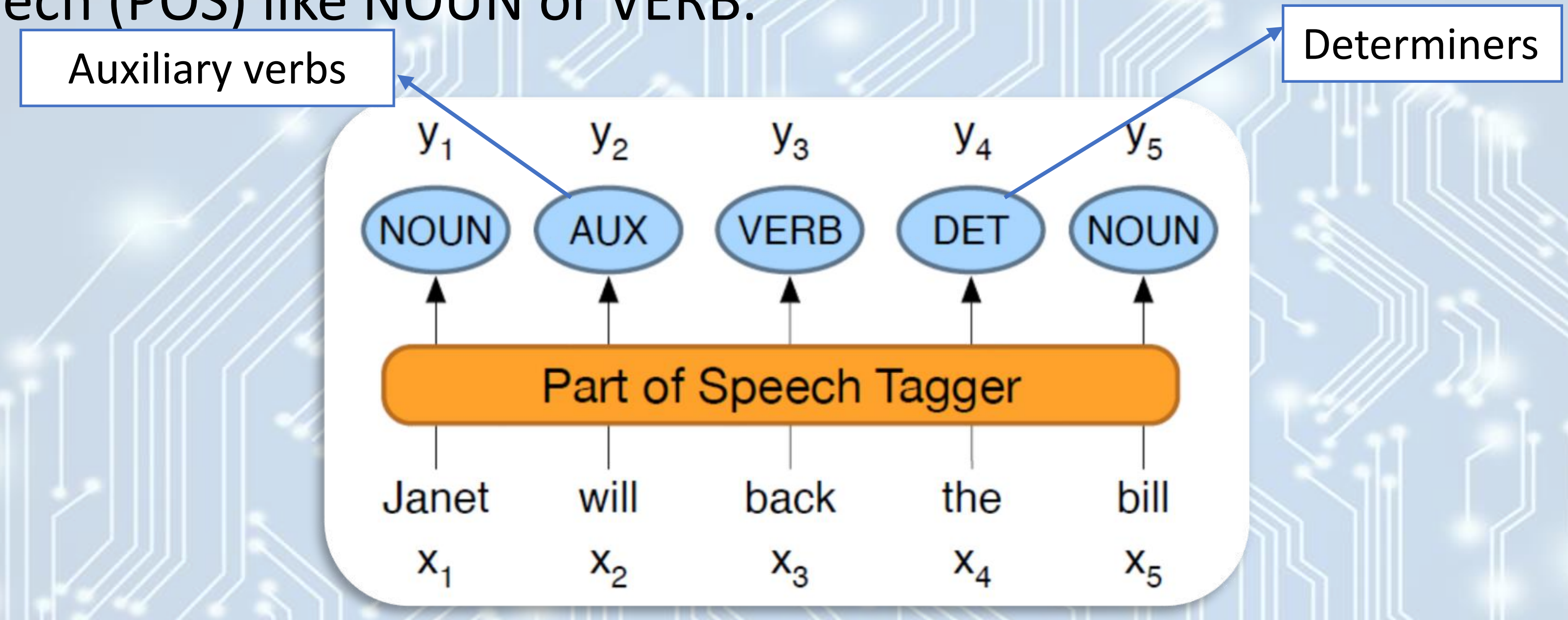
Sequence Labeling



- Tasks in which we assign, to each word x_i in an input word sequence, a label y_i, so that the output sequence Y has the same length as the input sequence X are called **sequence labeling tasks.**

Part-of-Speech Tagging (POS Tagging)

- Part-of-speech tagging: is the task of taking a sequence of words and assigning each word a part of speech (POS) like NOUN or VERB.



The task of part-of-speech tagging: mapping from input words x1,x2,…,xn to output POS tags y1,y2,…,yn

- Tagging is a **disambiguation task:** words are ambiguous—have more than one possible part-of-speech—the goal of POS-tagging is to resolve these ambiguities, choosing the proper tag for the context.
  - For example, book can be a **verb** (book that flight) or a **noun** (hand me that book).
- POS tagging is a useful first step in lots of natural language processing tasks: Named Entity Recognition (NER), sentiment analysis, machine translation and word sense disambiguation.

Part-of-Speech Tagging (POS Tagging)

- Parts of speech fall into two broad categories:
  - closed class
  - open class
- **Open classes:** (such as nouns, verbs and adjectives) acquire new members constantly. New nouns and verbs like *iPhone* or *to fax* are continually being created or borrowed.
- **Closed classes:** (such as pronouns, prepositions and conjunctions) acquire new members infrequently, if at all. These are generally function words like *of, it, and,* or *you*, which tend to be very short, occur frequently, and often have structuring uses in grammar.

- The most-frequent-tag baseline has an accuracy of about **92%**. The baseline thus differs from the state-of-the-art and human ceiling (97%) by only 5%.

- **Penn Treebank** is a famous English-specific part-of-speech tagset that has been used to label many syntactically annotated corpora like the Penn Treebank corpora.

Named Entity Recognition (NER)

- NER: is the task of assigning words or phrases tags like PERSON, LOCATION, or ORGANIZATION.

> 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**].

- NER is a useful first step in lots of natural language processing tasks: sentiment analysis (want to know a consumer's sentiment toward a particular entity), question answering and information extraction.

- Unlike part-of-speech tagging, where there is no **segmentation** problem

  → each word gets one tag
- NER is to find and label spans of text, it is partly difficult due to:
  - **Segmentation ambiguity**: need to decide what's an entity and what isn't, and where the **boundaries** are. Indeed, most words in a text will not be named entities.
  - **Type ambiguity:** for example *Kentucky* is a restaurant, person or location.

Named Entity Recognition (NER)

- The standard approach to sequence labeling for a span-recognition problem like NER is **"BIO tagging"** and its variants: **"IO tagging"** and **"BIOES tagging".**
  - BIO tagging: we label any token that begins a span of interest with the label B, tokens that occur inside a span are tagged with an I, and any tokens outside of any span of interest are labeled O.
  - IO tagging: loses some information by eliminating the B tag.
  - BIOES tagging: adds an end tag E for the end of a span, and a span tag S for a span consisting of only one word.

[$_{PER}$ **Jane Villanueva**] of [$_{ORG}$ **United**] , a unit of [$_{ORG}$ **United Airlines Holding**] , said the fare applies to the [$_{LOC}$ **Chicago**] route.

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

- This way the NER is a sequence labeling task same as part-of-speech tagging assigning a single label yi to each input word xi: a sequence labeler is trained to label each token in a text with tags that indicate the presence (or absence) of particular kinds of named entities.

Sequence Labeling Approaches

- Hidden Markov Model: HMM
- Conditional Random Field: CRF
- Recurrent Neural Networks: RNN
- Transformers
- Others

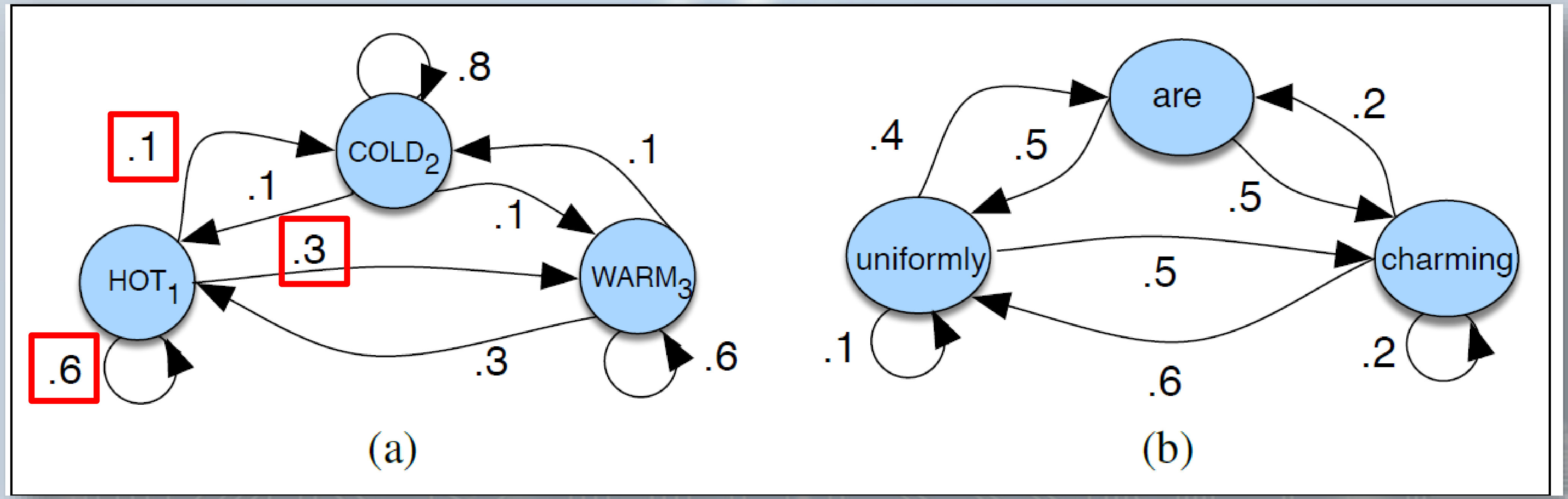
Hidden Markov Model (HMM)

- HMM is a **generative** approach.
- An HMM is a **probabilistic** sequence model: given a sequence of units (words, letters, morphemes, sentences, …), it computes a probability distribution over possible sequences of labels and chooses the best label sequence.

- HMM is based on augmenting the Markov chain.
- A **Markov chain** is a model that tells us something about the probabilities of sequences of random variables/states, each of which can take on values from some set.
  - These sets can be words, or tags, or symbols representing anything, for example the weather.
- A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the **current state**.
  - To predict tomorrow's weather, you could examine today's weather but you weren't allowed to look at yesterday's weather.

$$\textbf{Markov Assumption:} \quad P(q_i = a | q_1 \ldots q_{i-1}) = P(q_i = a | q_{i-1})$$

Markov Chains



(a) A Markov chain for weather

(b) A Markov chain for words

- The graph consists of nodes and edges:
  - **Nodes→ states**
  - **Edges→ the transitions, with their probabilities**
- The values of arcs leaving a given state must **sum to 1**.
- (a) shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of *HOT*, *COLD*, and *WARM*.
- (b) shows a Markov chain for assigning a probability to a sequence of words w1...wt .
  - This Markov chain should be familiar: it represents a **bigram language model**, with each edge expressing the probability p(wi|wj)

Markov Chains

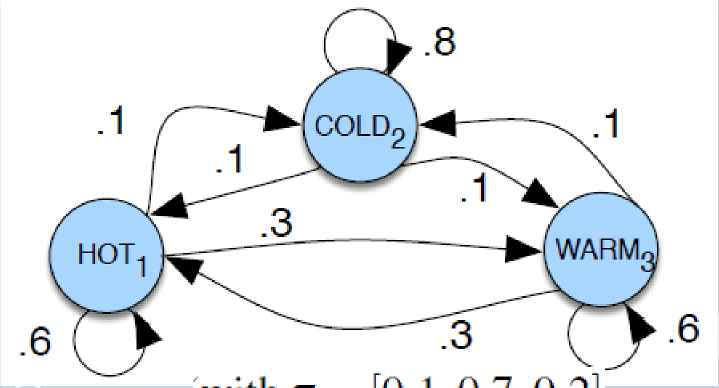
- Formally, a Markov chain is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

- Example: compute the probability of each of the following sequences:

Can you prove these calculations mathematically??

  - hot hot hot hot:
    P(hot)*P(hot|hot)*P(hot|hot)*P(hot|hot)
    =0.1*0.6*0.6*0.6= 0.0216
  - cold hot cold hot:
    P(cold)*P(hot|cold)*(cold|hot)*P(hot|cold)
    =0.7*0.1*0.1*0.1= 0.0007



with $\pi = [0.1, 0.7, 0.2]$

Hidden Markov Model

- A **Markov chain** is useful when we need to compute a probability for a sequence of **observable events**.

- In many cases, however, the events we are interested in are **hidden**: we don't observe them directly.

- For example, we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence.
  - We call the tags hidden because they are not observed.

- A hidden Markov model (**HMM**) allows us to talk about both **observed events** (like **words** that we see in the input) and **hidden events** (like part-of-speech **tags**).

Hidden Markov Model

- An HMM is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state q_i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

- A first-order hidden Markov model instantiates two simplifying assumptions:

1. probability of a particular state depends only on the previous state:

$$\textbf{Markov Assumption:} \quad P(q_i|q_1,...,q_{i-1}) = P(q_i|q_{i-1})$$

2. probability of an output observation oi depends only on the state that produced the observation qi and not on any other states or any other observations:

$$\textbf{Output Independence:} \quad P(o_i|q_1,\ldots q_i,\ldots,q_T,o_1,\ldots,o_i,\ldots,o_T) = P(o_i|q_i)$$

HMM Tagger

An HMM has two components, the A and B probabilities:

- The **A** matrix contains the tag transition probabilities **P(ti|ti-1)** which represent the probability of a tag occurring given the previous tag.
  - For example, modal verbs (MD) like *will* are very likely to be followed by a verb in the base form (VB) like *learn* →we expect this probability to be high.
  - We compute the **maximum likelihood estimate** of this transition probability by **counting**: out of the times we see the first tag in a labeled corpus, how often the first tag is followed by the second:

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

$$P(VB|MD) = \frac{C(MD,VB)}{C(MD)} = \frac{10471}{13124} = .80$$

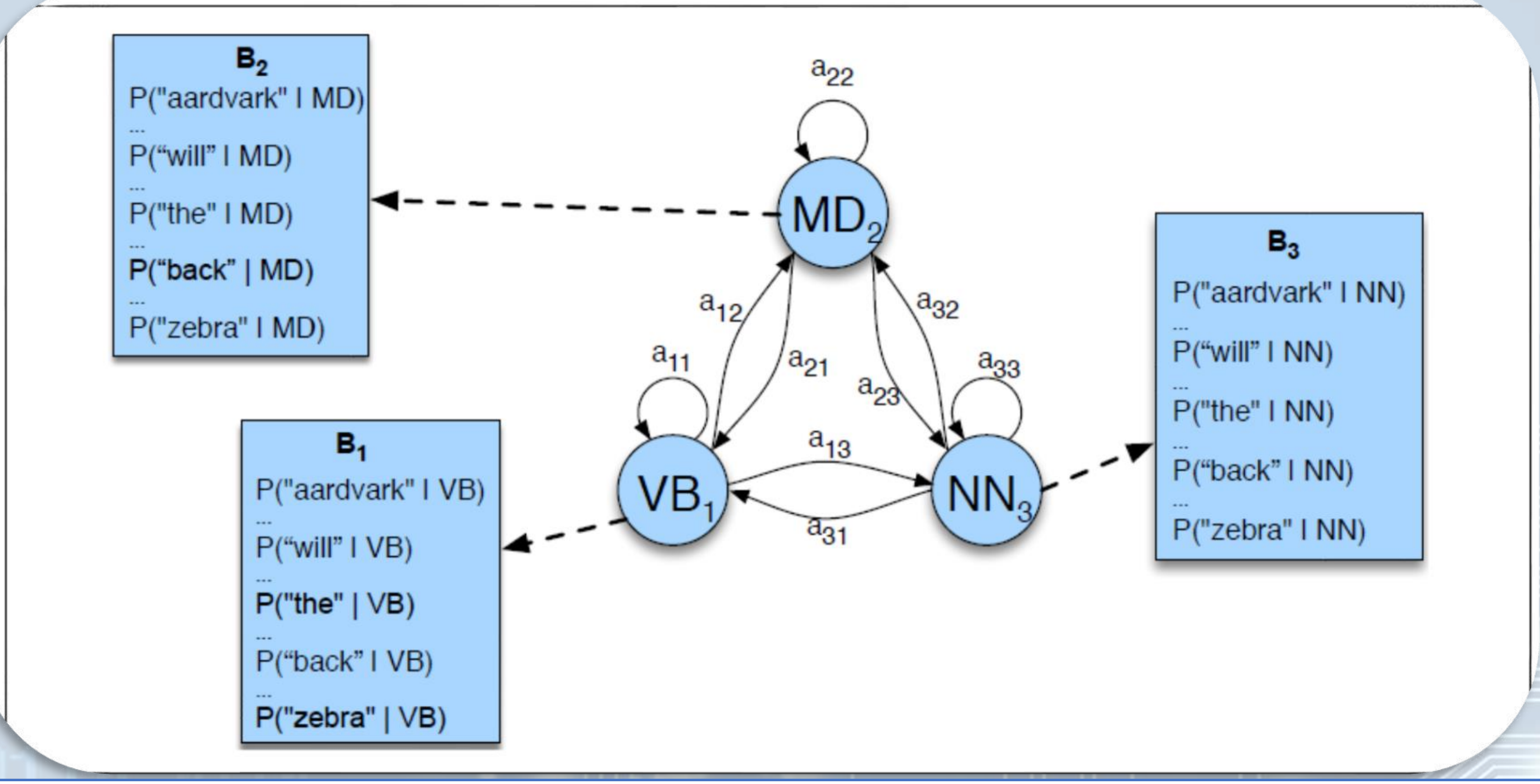
- The **B** emission probabilities **P(wi|ti)** represent the probability, given a tag (say *MD*), that it will be associated with a given word (say *will*).
  - The **MLE** of the emission probability is: $\boxed{P(w_i|t_i) = \frac{C(t_i,w_i)}{C(t_i)}}$

$$P(will|MD) = \frac{C(MD,will)}{C(MD)} = \frac{4046}{13124} = .31$$

This likelihood term is NOT asking:
“which is the most likely tag for the word will?”
→ the posterior P(MD|will).
Instead, P(will|MD) answers the question:
“If we were going to generate a MD, how likely is it that this modal would be will?”

HMM Tagger

- A three states HMM part-of-speech tagger (the full tagger would have one state for each tag):



An illustration of the two parts of an HMM representation:
- the A transition probabilities used to compute the prior probability
- the B observation likelihoods that are associated with each state, one likelihood for each possible observation word.

HMM Tagging as Decoding

- **Decoding:** is the task of determining the hidden variables sequence corresponding to the sequence of observations.

> **Decoding**: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

- For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence *t1…tn* that is most probable given the observation sequence of *n* words *w1…wn*:

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(t_1 \ldots t_n | w_1 \ldots w_n)$$

- Using Bayes' rule: $\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} \dfrac{P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)}{P(w_1 \ldots w_n)}$
- Dropping the denominator: $\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)$
- HMM taggers make two further simplifying assumptions:

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

probability of a word appearing depends only on its own tag and is independent of neighboring words and tags.

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

the bigram assumption, is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence.

$$\hat{t}_{1:n} = \underset{t_1 \ldots t_n}{\operatorname{argmax}} P(t_1 \ldots t_n | w_1 \ldots w_n) \approx \underset{t_1 \ldots t_n}{\operatorname{argmax}} \prod_{i=1}^{n} \overbrace{P(w_i | t_i)}^{\text{emission}} \overbrace{P(t_i | t_{i-1})}^{\text{transition}}$$

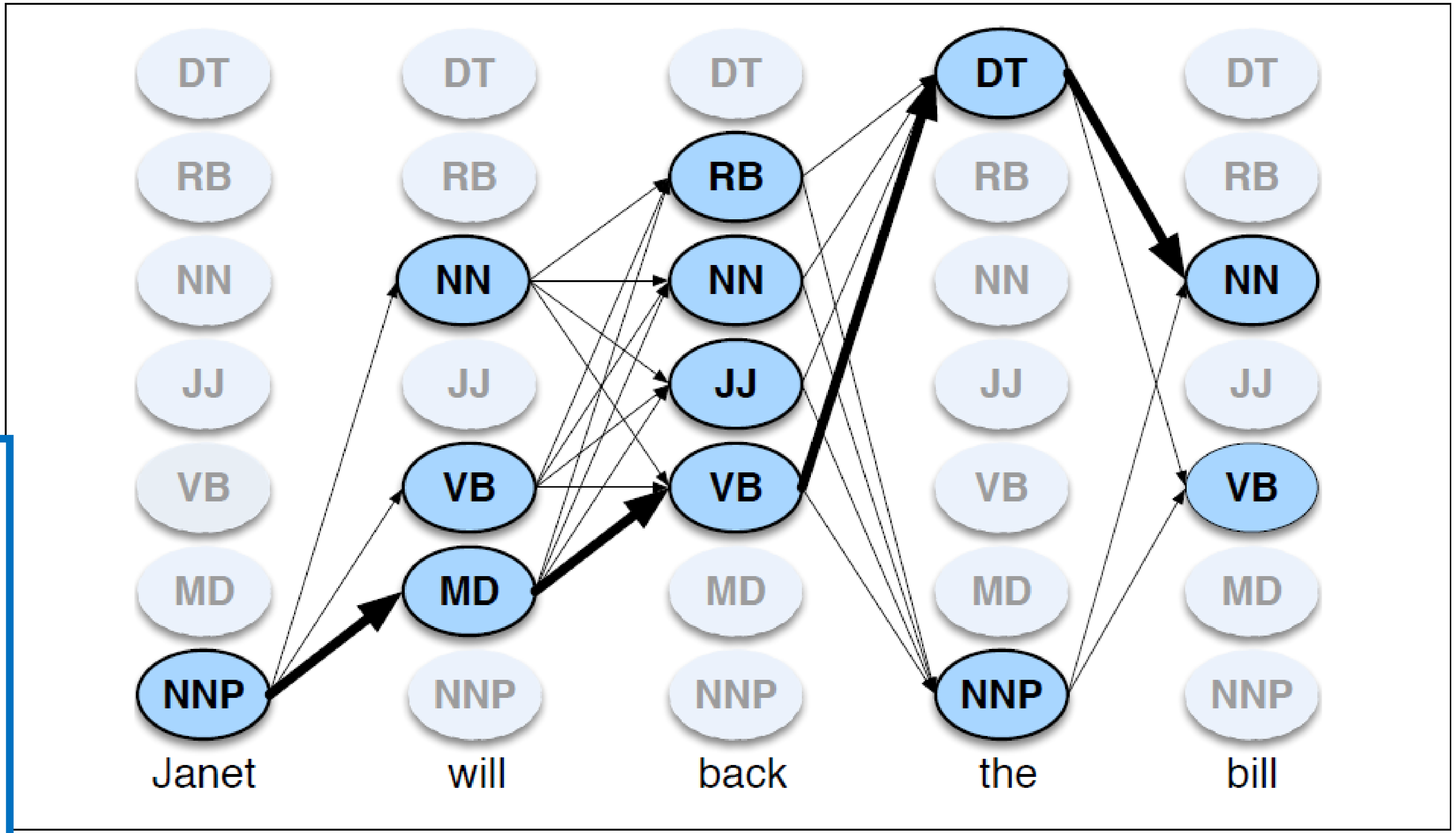
The Viterbi Algorithm

- The decoding algorithm for HMMs is the Viterbi algorithm, it is an instance of dynamic programming.
- The algorithm first sets up a **probability matrix or lattice**, with one column for each observation *ot* and one row for each state *qi* in the state graph.

- DT: determiner
- RB: adverb
- NN: singular or mass noun
- JJ: adjective
- VB: verb base
- MD: modal
- NNP: proper noun, singular

A sketch of the lattice for *Janet will back the bill:*

- The possible tags (qi) for each word.
- The path corresponding to the correct tag sequence is highlighted.
- States (parts of speech) which have a zero probability of generating a particular word according to the B matrix such as P(Janet|DT) are greyed out.



The Viterbi Algorithm

- Each cell of the lattice, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $q_1, \ldots, q_{t-1}$, given the HMM λ.
- The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell:

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

$v_{t-1}(i)$	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

The Viterbi Algorithm Example

- Let's tag the sentence *Janet will back the bill*
- The gold answer: *Janet/NNP will/MD back/VB the/DT bill/NN*
- The **A transition probabilities P(ti|ti-1)** computed from the WSJ corpus without smoothing *(ONLY part is shown).* Rows are labeled with the conditioning event:
  - E.g.: P(VB|MD)=0.7968, P(NNP|<s>)=$\pi_{NNP} = 0.2767$

	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

The Viterbi Algorithm Example

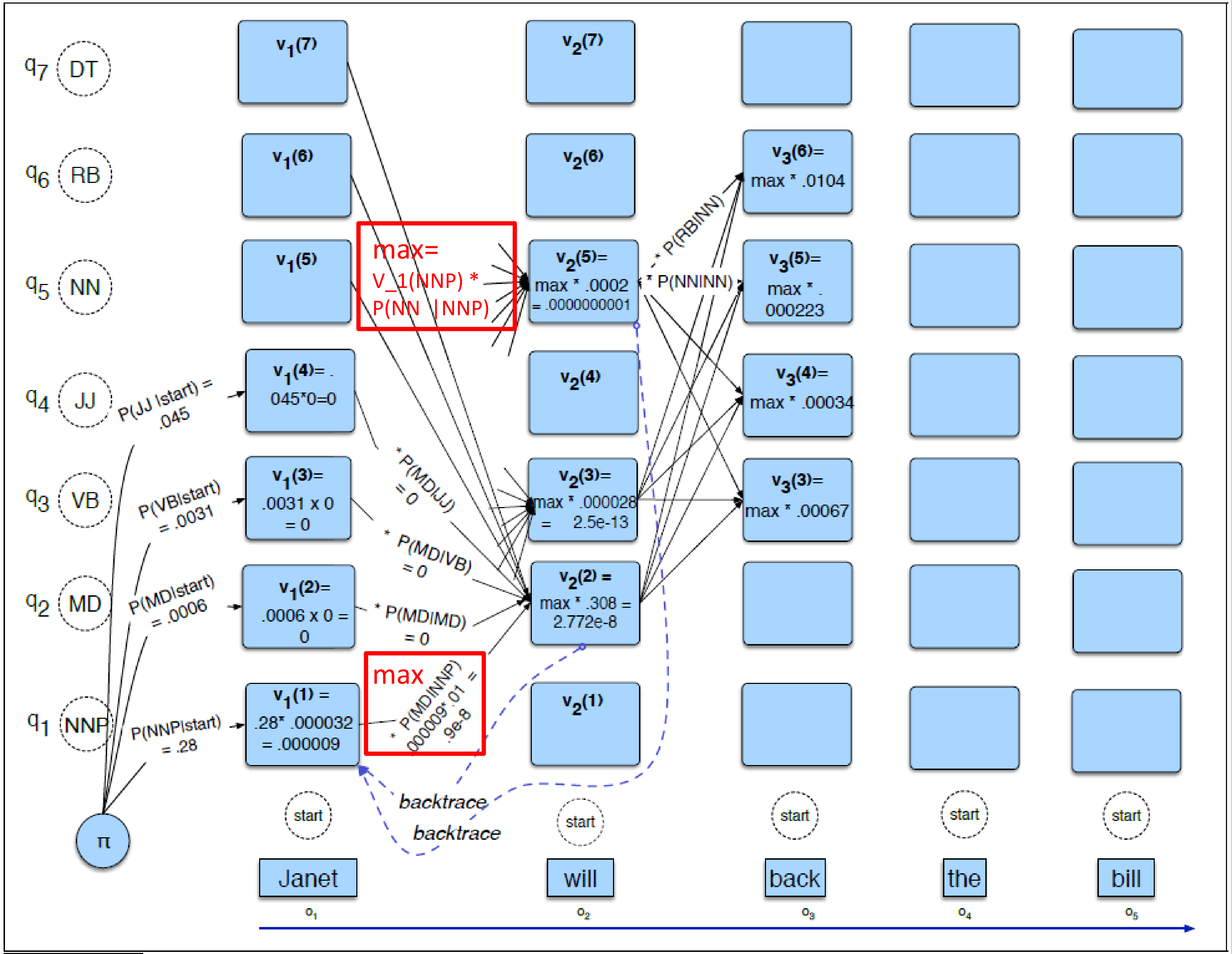
- The **observation likelihoods B (Emission probability matrix)** computed from the WSJ corpus without smoothing (*simplified slightly*).
  - E.g.: P(back|JJ)=0.000340
  - The word *Janet* only appears as an *NNP*, *back* has *4* possible parts of speech, and the word *the* can appear as a *determiner* or as an *NNP*.

Remember the greyed out nodes in the graph →Prob=0

	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

The Viterbi Algorithm Example

- We begin in column 1 (for the word *Janet*) by setting the Viterbi value in each cell to the product of the π transition probability and the observation likelihood of the word *Janet* given the tag for that cell.
- Next, each cell in the *will* column gets updated. For each state, we compute the value viterbi[s,t] by taking the maximum over the extensions of all the paths from the previous column that lead to the current cell.
- Each cell keeps the probability of the best path so far and a pointer to the previous cell along that path.
- Termination: take the max value from the last column of the Viterbi matrix selecting its tag and then use the pointer to go back (selecting tags) until reach the 1rst word.



The 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]*
for each state s **from** 1 **to** N **do** ; initialization step
$\quad viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
$\quad backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**

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

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

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

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

bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*


Thank You