L95: Natural Language Syntax and Parsing
1) HMMs and Viterbi

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge

Study Information

Useful Textbooks

Much of the lecture content can be found in the following text books:

- Jurafsky, D. and Martin, J. *Speech and Language Processing*
- Manning, C. and Schutze, H. *Foundations of Statistical Natural Language Processing*
- Ruslan M. *The Oxford Handbook of Computational Linguistics*
- Clark, A., Fox, C, and Lappin, S. *The Handbook of Computational Linguistics and Natural Language Processing*

A **formal language** is a set of **strings** over an **alphabet**

ALPHABET

An alphabet is specified by a **finite** set, Σ, whose elements are called symbols. Some examples are shown below:

- $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ the 10-element set of decimal digits.
- $\{a, b, c, ..., x, y, z\}$ the 26-element set of lower case characters of written English.
- $\{aardvark, ..., zebra\}$ the 250,000-element set of *words* in the Oxford English Dictionary.[1]

Note that e.g. the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, ...\}$ cannot be an alphabet because it is infinite.

[1]Note that the term *alphabet* is overloaded

A **formal language** is a set of **strings** over an **alphabet**

Strings

A string of length n over an alphabet Σ is an ordered n-tuple of elements of Σ.

Σ^* denotes the set of all strings over Σ of finite length.

- If $\Sigma = \{a, b\}$ then ϵ, ba, bab, aab are examples of strings over Σ.
- If $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, ...\}$
- If $\Sigma = \{cats, dogs, eat\}$ then $\Sigma^* = \{\epsilon, cats, cats\ \ eat, cats\ \ eat\ \ dogs, ...\}$[2]

Languages

Given an alphabet Σ any subset of Σ^* is a **formal language** over alphabet Σ.

Language models

Define a particular subset of strings $S \subseteq \Sigma^*$

[2]The spaces here are for readable delimitation of the symbols of the alphabet.

A **formal language** is a set of **strings** over an **alphabet**

Strings

A string of length n over an alphabet Σ is an ordered n-tuple of elements of Σ.

Σ^* denotes the set of all strings over Σ of finite length.

- If $\Sigma = \{a, b\}$ then ϵ, ba, bab, aab are examples of strings over Σ.
- If $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, ...\}$
- If $\Sigma = \{cats, dogs, eat\}$ then $\Sigma^* = \{\epsilon, cats, cats\ \ eat, cats\ \ eat\ \ dogs, ...\}$[2]

Languages

Given an alphabet Σ any subset of Σ^* is a **formal language** over alphabet Σ.

Language models

Define a particular subset of strings $S \subseteq \Sigma^*$

[2]The spaces here are for readable delimitation of the symbols of the alphabet.

A **formal language** is a set of **strings** over an **alphabet**

Strings

A string of length n over an alphabet Σ is an ordered n-tuple of elements of Σ.

Σ^* denotes the set of all strings over Σ of finite length.

- If $\Sigma = \{a, b\}$ then ϵ, ba, bab, aab are examples of strings over Σ.
- If $\Sigma = \{a\}$ then $\Sigma^* = \{\epsilon, a, aa, aaa, ...\}$
- If $\Sigma = \{cats, dogs, eat\}$ then $\Sigma^* = \{\epsilon, cats, cats\ \ eat, cats\ \ eat\ \ dogs, ...\}$[2]

Languages

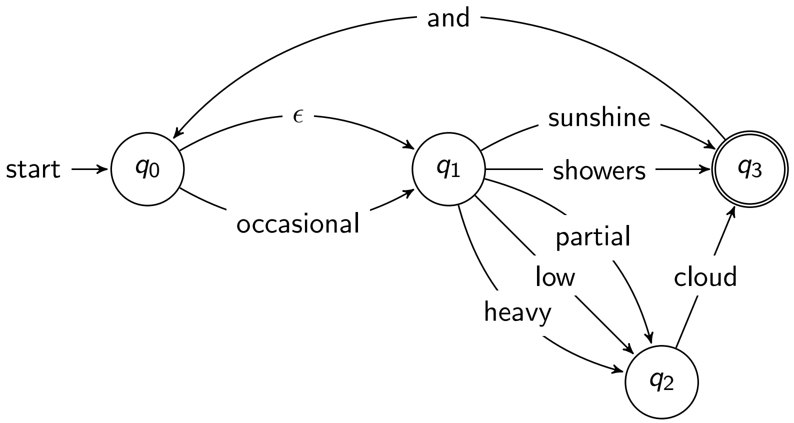
Given an alphabet Σ any subset of Σ^* is a **formal language** over alphabet Σ.

Language models

Define a particular subset of strings $S \subseteq \Sigma^*$

[2]The spaces here are for readable delimitation of the symbols of the alphabet.

Languages can be defined using **automata**



The language of (limited) weather prediction modelled by a **finite state automaton**

Languages can be defined using **automata**

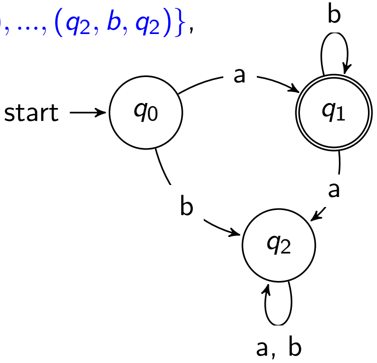
A formal language is **regular** if it is equal to the set of strings accepted by some deterministic finite-state automaton (DFA). A DFA is defined as $M = (\mathcal{Q}, \Sigma, \Delta, s, \mathcal{F})$ where:

- $\mathcal{Q} = \{q_0, q_1, q_2 ...\}$ is a finite set of states.
- Σ is the alphabet: a finite set of transition symbols.
- $\Delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a function $\mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ which we write as δ. Given $q \in \mathcal{Q}$ and $i \in \Sigma$ then $\delta(q, i)$ returns a new state $q' \in \mathcal{Q}$
- s is a starting state
- $\mathcal{F}$ is the set of all end states

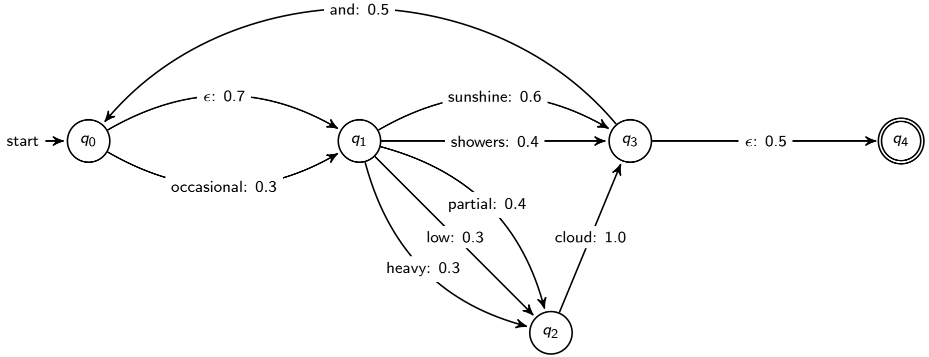
Regular languages are accepted by **DFAs**

For $\mathcal{L}(M) = \{a, ab, abb, ...\}$:

$$\begin{aligned} M = (\ & \mathcal{Q} = \{q_0, q_1, q_2\}, \\ & \Sigma = \{a, b\}, \\ & \Delta = \{(q_0, a, q_1), (q_0, b, q_2), ..., (q_2, b, q_2)\}, \\ & s = q_0, \\ & \mathcal{F} = \{q_1\}\ ) \end{aligned}$$

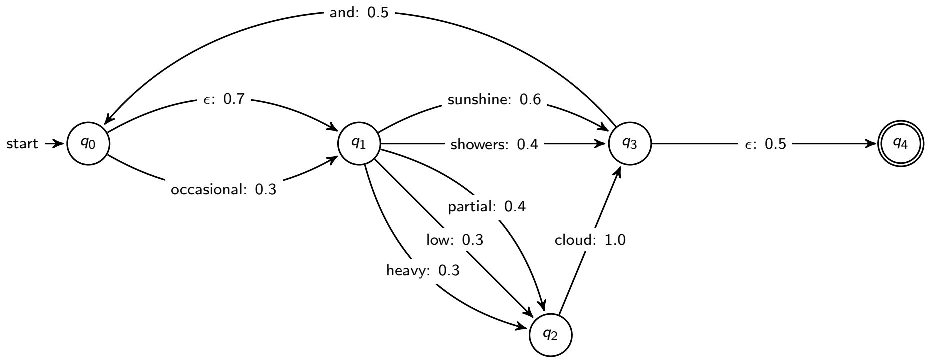


Edge **probabilities** allow us to calculate string probabilities



For the string **sunshine and occasional low cloud**:
$0.7 * 0.6 * 0.5 * 0.3 * 0.3 * 1.0 * 0.5 = 0.00945$
Markov assumption on the probabilities in the sequence: when predicting the future, the past doesn't matter, only the present.

Edge **probabilities** allow us to calculate string probabilities

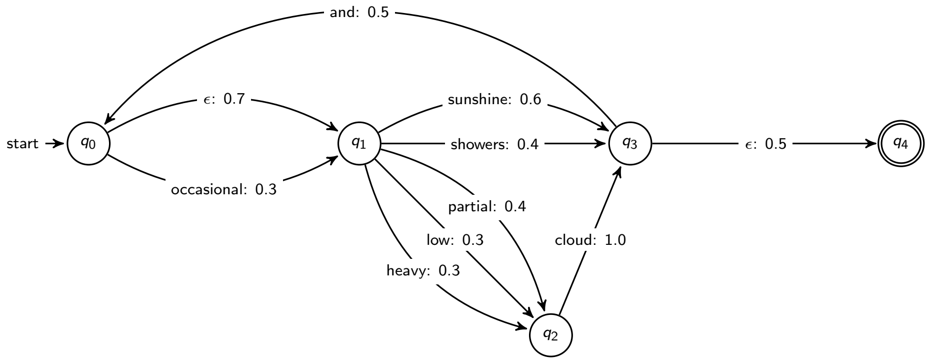


For the string **sunshine and occasional low cloud**:

$0.7 * 0.6 * 0.5 * 0.3 * 0.3 * 1.0 * 0.5 = 0.00945$

Markov assumption on the probabilities in the sequence: when predicting the future, the past doesn't matter, only the present.

Edge **probabilities** allow us to calculate string probabilities



For the string **sunshine and occasional low cloud**:
$0.7 * 0.6 * 0.5 * 0.3 * 0.3 * 1.0 * 0.5 = 0.00945$
Markov assumption on the probabilities in the sequence: when predicting the future, the past doesn't matter, only the present.

Markov models assume the Markov property

1st Order Markov Property

- For a sequence of state variables: $q_1, q_2, ...q_i$

  $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$

  i.e. the probability of the next state is dependant only on the current state

Clearly an oversimplification for modelling natural language but Markov models can be very efficient so we use them when we can.

n-**gram** language models are $(n-1)$-order Markov models

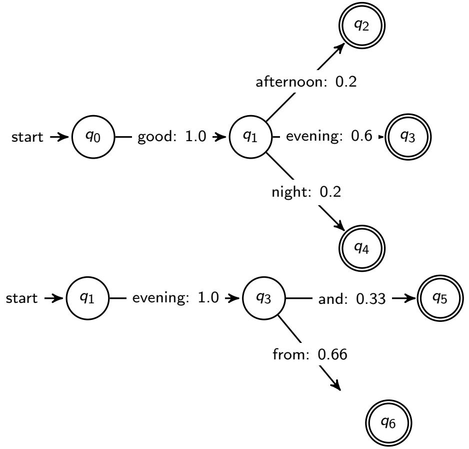
Consider a bi-gram model of the very short corpus:

good afternoon, good evening and good night.
it's good evening from me and good evening from him.

(MLE) bi-grams starting with **good** and **evening**:

good afternoon	0.2
good evening	0.6
good night	0.2
evening and	0.33
evening from	0.66

Markov chain models an observed random variable that changes through time.



Markov chains model sequences of random variables

Markov Chain

- $\mathcal{Q} = \{q_0, q_1, q_2 ...\}$ is a finite set of **states**.
- $A = (a_{11}, a_{12}, ...a_{n1}...a_{nn})$ is a **transition probability matrix** where a_{ij} represents the probability of moving from state i to state j s.t. for all i, $\sum_{j=1}^{n} a_{ij} = 1$
- $\pi = (\pi_1, \pi_2, ...\pi_n)$ is an **initial probability distribution** over the states where π_i is the probability that the chain starts in state i

Use **Hidden Markov Models** for unobserved sequences

- **Markov chains** are used for **observed** event sequences
- In some scenarios we are interested in **hidden event sequences** which are not directly observed but are causal factors
- e.g. We don't normally observe **part-of-speech** tags in a text. We observe words and infer the tags from the word sequences.
- A **hidden Markov model** (HMM) combines observed events (e.g. words) and hidden events (e.g. part-of-speech tags)

Use **Hidden Markov Models** for unobserved sequences

Hidden Markov Model

- $\mathcal{Q} = \{q_0, q_1, q_2 ...\}$ is a finite set of **states**.
- $A = (a_{11}, a_{12}, ... a_{n1} ... a_{nn})$ is a **transition probability matrix** where a_{ij} represents the probability of moving from state i to state j s.t. for all i, $\Sigma_{j=1}^{n} a_{ij} = 1$
- $O = (o_1, o_2, ... o_T)$ is a sequence of T **observations** drawn from a **alphabet**, $\Sigma = \{v_1 ... v_V\}$.
- $B = (b_{1v_1}, b_{1v_2}, ... b_{iv_1} ... b_{nv_V})$ is an **emission probability matrix** where b_{iv_t} expresses the probability of an observation v_t given the current state i
- $\pi = (\pi_1, \pi_2, ... \pi_n)$ is an **initial probability distribution** over the states where π_i is the probability that the chain starts in state i

Use **Hidden Markov Models** for unobserved sequences

$$\begin{tikzcd}
\boxed{\begin{array}{l} B_1 = (\\ p(v_1|q_1) \\ p(v_2|q_1) \\ \dots \\ p(v_V|q_1)) \end{array}} & q_1 \arrow[l, dashed] \arrow[loop above, "a_{11}"] \arrow[r, bend right, "a_{12}"'] & q_2 \arrow[l, bend right, "a_{21}"'] \arrow[loop above, "a_{22}"] \arrow[r, dashed] & \boxed{\begin{array}{l} B_2 = (\\ p(v_1|q_2) \\ p(v_2|q_2) \\ \dots \\ p(v_V|q_2)) \end{array}}
\end{tikzcd}$$

1st-order HMMs make two assumptions

- For a sequence of state variables: $q_1, q_2, ...q_i$

  $P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$

  i.e. the probability of the next state is dependant only on the current state

  $P(o_i|q_1...q_T, o_1...o_i...o_T) = P(o_i|q_i)$

  i.e. the observations depend only on the current state

HMMs may be used for POS-tagging

- The transition probability matrix A represents **tag transition probabilities** $P(t_i|t_{i-1})$
- Probabilities may be estimated from frequencies in annotated corpora (Maximum Likelihood Estimation)
  $P(t_i|t_{i-1}) = C(t_{i-1}, t_i)/C(t_{i-1})$
- The emission probability matrix B represents probabilities of words being associated with particular tags $P(w_i|t_i)$
- MLE may be calculated from annotated corpora
  $P(w_i|t_i) = C(t_i, w_i)/C(t_i)$

Decoding determines the hidden state sequence

Determining the hidden sequence corresponding to the sequence of observations is called **decoding**.

Decoding

- Given
  - HMM with transition probability matrix A
  - and emission probability matrix B
  - with observation sequence $O = o_1, o_2, ...o_T$
- Find
  - most probable state sequence $Q = q_1, q_2, ...q_T$

For POS-tagging: choose the tag sequence t_1^n that is most probable given the observation sequence of w_1^n

Decoding for part-of-speech tagging

For POS-tagging: choose the tag sequence t_1^n that is most probable given the observation sequence of w_1^n

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

Using Bayes:

$$P(t_1^n|w_1^n) = \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)} \rightarrow P(w_1^n|t_1^n)P(t_1^n)$$

If we assume independence $P(t_1^n)$ can be approximated as:

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

And $P(w_1^n|t_1^n)$ may be approximated as:

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

Decoding for part-of-speech tagging

For POS-tagging: choose the tag sequence t_1^n that is most probable given the observation sequence of w_1^n

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

(t_1^n hidden states, Q)
(w_1^n observations, O)

Using Bayes:

$$P(t_1^n | w_1^n) = \frac{P(w_1^n | t_1^n) P(t_1^n)}{P(w_1^n)} \rightarrow P(w_1^n | t_1^n) P(t_1^n)$$

If we assume independence $P(t_1^n)$ can be approximated as:

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

(transition probabilities, A)

And $P(w_1^n | t_1^n)$ may be approximated as:

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

(emission probabilities, B)

Can also think of POS-tagging a noisy channel model

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

$$\begin{tikzcd} \arrow[r, "tags,"] & \boxed{channel,} \arrow[r, "words,"] & \boxed{decoder,} \arrow[r, "tags',"] & {} \end{tikzcd}$$

Can also think of POS-tagging a noisy channel model

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

$$\begin{tikzcd} {} \arrow[r, "{tags, A}"] & \boxed{channel, B} \arrow[r, "{words, O}"] & \boxed{decoder, (A, B)} \arrow[r, "{tags', Q}"] & {} \end{tikzcd}$$

Viterbi algorithm finds the optimum sequence of tags

Viterbi algorithm is a dynamic programming algorithm

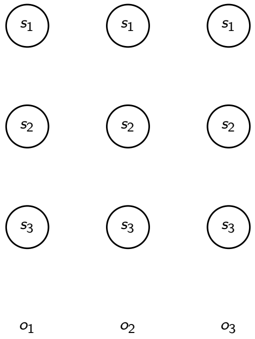
- For a HMM with N states, transitions A, emissions B and observations O_1^T:
- Builds a lattice with one column for each observation and one row for each possible hidden state, T by N lattice
- **After passing through the most probable state sequence**, $q_1 ... q_{t-1}$, $v_t(j)$ is the probability of being in state j after the first t observations

  $v_t(j) = max\ P(q_1 ... q_{t-1}, o_1 ... o_t, q_t = j)$
- Calculate $v_t(j)$ recursively

  $v_t(j) = max\ v_{t-1}(i) a_{ij} b_{jo_t}$

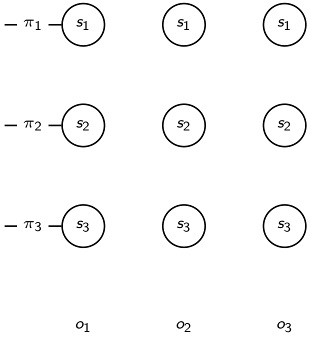
Considers max of N paths at every observation.

Viterbi algorithm finds the optimum sequence of tags



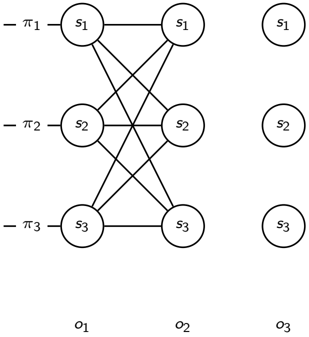
$$v_t(j) = max\ v_{t-1}(i) a_{ij} b_{jo_t}$$

Viterbi algorithm finds the optimum sequence of tags



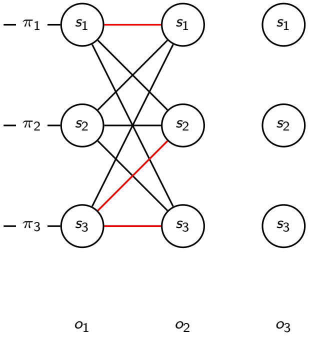
$$v_t(j) = max\ v_{t-1}(i) a_{ij} b_{jo_t}$$

Viterbi algorithm finds the optimum sequence of tags



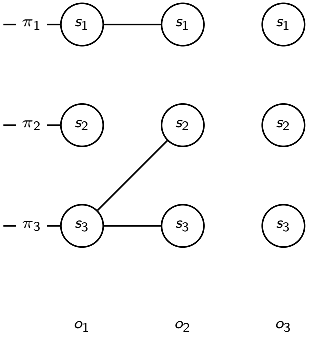
$$v_t(j) = max\ v_{t-1}(i) a_{ij} b_{jo_t}$$

Viterbi algorithm finds the optimum sequence of tags



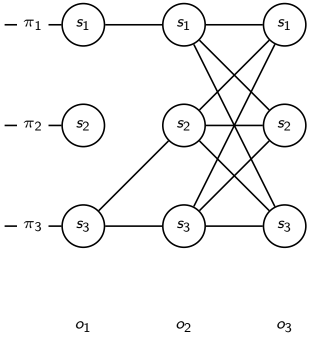
$$v_t(j) = max\ v_{t-1}(i) a_{ij} b_{jo_t}$$

Viterbi algorithm finds the optimum sequence of tags



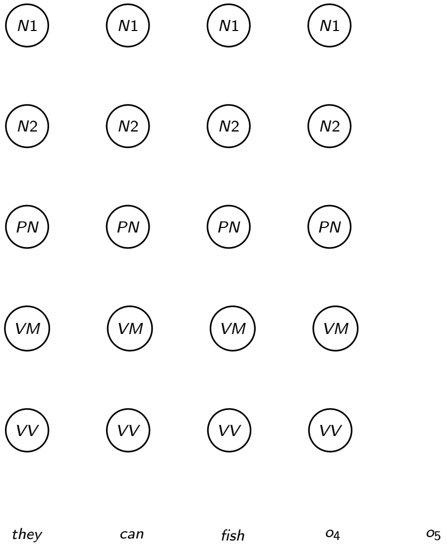
$$v_t(j) = max\ v_{t-1}(i) a_{ij} b_{jo_t}$$

Viterbi algorithm finds the optimum sequence of tags



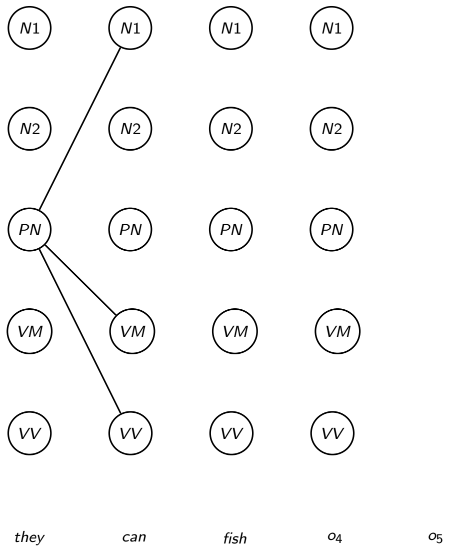
$$v_t(j) = max\ v_{t-1}(i) a_{ij} b_{jo_t}$$

Viterbi algorithm finds the optimum sequence of tags



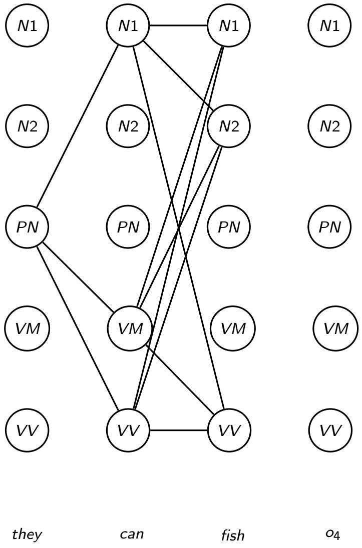
they	PN
can	N1, VM, VV
fish	N1, N2, VV

Viterbi algorithm finds the optimum sequence of tags



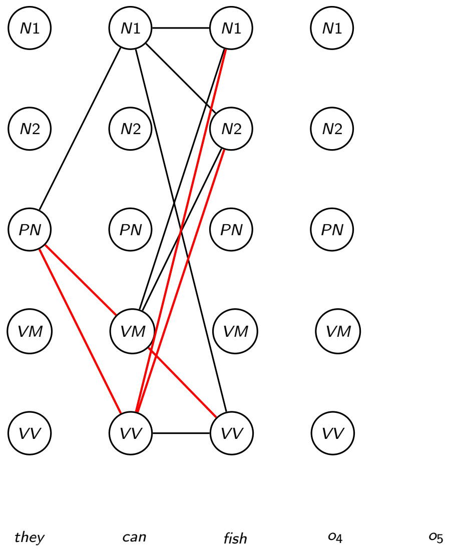
they	PN
can	N1, VM, VV
fish	N1, N2, VV

Viterbi algorithm finds the optimum sequence of tags



they	PN
can	N1, VM, VV
fish	N1, N2, VV

Viterbi algorithm finds the optimum sequence of tags



they	PN
can	N1, VM, VV
fish	N1, N2, VV

The basic Viterbi algorithm can be extended

- To extend to **trigrams** we consider N^2 states at very observation rather than N
- For a **beam search** instead of keeping all N state paths at each time point t, just keep the best few hypothesis
- Complexity of basic algorithm is $O(N^2 T)$. It's necessary to use beam search when N is large: consider neural network decoding

Other algorithms associated to HMMs:

- the **forward algorithm** the probability of a state given the history of observations
- the **Baum–Welch algorithm** estimate the parameters of a hidden Markov model given the observations