INF4820: Algorithms for Artificial Intelligence and Natural Language Processing

Language Models & Hidden Markov Models

Stephan Oepen & Erik Velldal

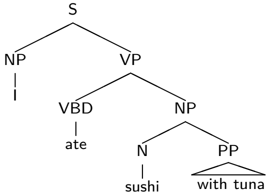
Language Technology Group (LTG)

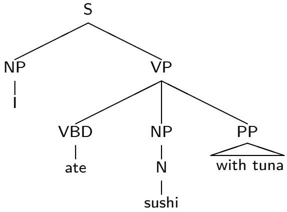
October 14, 2015

Recall: By the End of the Semester . . .

. . . you should be able to determine

- which string is most likely:
  - *How to recognize speech* vs. *How to wreck a nice beach*
- which category sequence is most likely for *flies like an arrow*:
  - **N V D N** vs. **V P D N**
- which syntactic analysis is most likely:





Language Models — N-Grams

A probabilistic (or stochastic) language model M assigns probabilities $P_M(x)$ to all strings x in language L.

We simplify using the Markov assumption (limited history):

> *the last $n-1$ elements approximate the effect of the full sequence.*

That is, instead of

- $P(w_i|w_1, \dots w_{i-1})$

selecting an n of 3, we use

- $P(w_i|w_{i-1}, w_{i-2})$

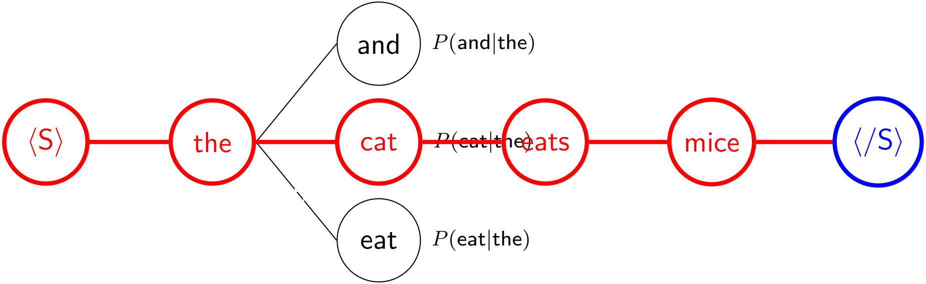
We call these short sequences of words n-grams:

- bigrams: *I want*, *want to*, *to go*, *go to*, *to the*, *the beach*
- trigrams: *I want to*, *want to go*, *to go to*, *go to the*
- 4-grams: *I want to go*, *want to go to*, *to go to the*

N-Gram Models

A generative model models a joint probability in terms of conditional probabilities.

We talk about the *generative story*:



$$P(S) = P(\text{the}|\langle S\rangle)\ P(\text{cat}|\text{the})\ P(\text{eats}|\text{cat})\ P(\text{mice}|\text{eats})\ P(\langle/S\rangle|\text{mice})$$

N-Gram Models

An n-gram language model records the n-gram conditional probabilities:

$$
\begin{array}{llll}
P(\textit{I}|\langle S\rangle) & = 0.0429 & P(\textit{to}|\textit{go}) & = 0.1540 \\
P(\textit{want}|\textit{I}) & = 0.0111 & P(\textit{the}|\textit{to}) & = 0.1219 \\
P(\textit{to}|\textit{want}) & = 0.4810 & P(\textit{beach}|\textit{the}) & = 0.0006 \\
P(\textit{go}|\textit{to}) & = 0.0131 & &
\end{array}
$$

We calculate the probability of a sentence as (assuming bi-grams):

$$
\begin{aligned}
P\left(w_1^n\right) & \approx \prod_{i=1}^{n} P\left(w_i|w_{i-1}\right) \\
& \approx P\left(\textit{I}|\langle S\rangle\right) \times P\left(\textit{want}|\textit{I}\right) \times P\left(\textit{to}|\textit{want}\right) \times P\left(\textit{go}|\textit{to}\right) \times P\left(\textit{to}|\textit{go}\right) \times \\
& \quad P\left(\textit{the}|\textit{to}\right) \times P\left(\textit{beach}|\textit{the}\right) \\
& \approx 0.0429 \times 0.0111 \times 0.4810 \times 0.0131 \times 0.1540 \times \\
& \quad 0.1219 \times 0.0006 = 3.38 \times 10^{-11}
\end{aligned}
$$

How to estimate the probabilities of n-grams?

By counting (e.g. for trigrams):

$$P\left(\text{bananas}|\text{i like}\right) = \frac{C\left(\text{i like bananas}\right)}{C\left(\text{i like}\right)}$$

The probabilities are estimated using the relative frequencies of observed outcomes. This process is called Maximum Likelihood Estimation (MLE).

Bigram MLE Example

"I want to go to the beach"

w_1	w_2	$C(w_1 w_2)$	$C(w_1)$	$P(w_2 \mid w_1)$
$\langle S \rangle$	I	1039	24243	0.0429
I	want	46	4131	0.0111
want	to	101	210	0.4810
to	go	128	9778	0.0131
go	to	59	383	0.1540
to	the	1192	9778	0.1219
the	beach	14	22244	0.0006

What's the probability of *Others want to go to the beach* ?

Problems with MLE of N-Grams

- Data sparseness: many perfectly acceptable n-grams will not be observed
- Zero counts will result in a estimated probability of 0

- Remedy—reassign some of the probability mass of frequent events to less frequent (or unseen) events.
- Known as smoothing or discounting
- The simplest approach is Laplace ('add-one') smoothing:

$$P_{\mathrm{L}}\left(w_n|w_{n-1}\right) = \frac{C\left(w_{n-1}w_n\right)+1}{C\left(w_{n-1}\right)+V}$$

Bigram MLE Example with Laplace Smoothing

"Others want to go to the beach"

w_1	w_2	$C(w_1 w_2)$	$C(w_1)$	$P(w_2 \mid w_1)$	$P_L(w_2 \mid w_1)$
$\langle S \rangle$	I	1039	24243	0.0429	0.01934
$\langle S \rangle$	Others	17	24243	0.0007	0.00033
I	want	46	4131	0.0111	0.00140
Others	want	0	4131	0	0.00003
want	to	101	210	0.4810	0.00343
to	go	128	9778	0.0131	0.00328
go	to	59	383	0.1540	0.00201
to	the	1192	9778	0.1219	0.03035
the	beach	14	22244	0.0006	0.00029

$$P_{\mathrm{L}}(w_n \mid w_{n-1}) = \frac{C(w_{n-1} w_n) + 1}{C(w_{n-1}) + 29534}$$

N-Gram Summary

- The likelihood of the next word depends on its context.
- We can calculate this using the chain rule:

$$P\left(w_1^N\right) = \prod_{i=1}^{N} P\left(w_i | w_1^{i-1}\right)$$

- In an n-gram model, we approximate this with a Markov chain:

$$P\left(w_1^N\right) \approx \prod_{i=1}^{N} P\left(w_i | w_{i-n+1}^{i-1}\right)$$

- We use Maximum Likelihood Estimation to estimate the conditional probabilities.
- Smoothing techniques are used to avoid zero probabilities.

Parts of Speech

- Known by a variety of names: part-of-speech, POS, lexical categories, word classes, morpho-syntactic classes, ...
- 'Traditionally' defined semantically (e.g. "nouns are naming words"), but (arguably) more accurately by their distributional properties.

- Open-classes
  - New words created/updated/deleted all the time
- Closed-classes
  - Smaller classes, relatively static membership
  - Usually function words

Open Class Words

- Nouns: dog, Oslo, scissors, snow, people, truth, cups
  - proper or common; countable or uncountable; plural or singular; masculine, feminine, or neuter; ...
- Verbs: fly, rained, having, ate, seen
  - transitive, intransitive, ditransitive; past, present, passive; stative or dynamic; plural or singular; ...
- Adjectives: good, smaller, unique, fastest, best, unhappy
  - comparative or superlative; predicative or attributive; intersective, subsective, or scopal; ...
- Adverbs: again, somewhat, slowly, yesterday, aloud
  - intersective; scopal; discourse; degree; temporal; directional; comparative or superlative; ...

Closed Class Words

- Prepositions: <u>on</u>, <u>under</u>, <u>from</u>, <u>at</u>, <u>near</u>, <u>over</u>, ...
- Determiners: <u>a</u>, <u>an</u>, <u>the</u>, <u>that</u>, ...
- Pronouns: <u>she</u>, <u>who</u>, <u>I</u>, <u>others</u>, ...
- Conjunctions: <u>and</u>, <u>but</u>, <u>or</u>, <u>when</u>, ...
- Auxiliary verbs: <u>can</u>, <u>may</u>, <u>should</u>, <u>must</u>, ...
- Interjections, particles, numerals, negatives, politeness markers, greetings, existential there ...

(Examples from Jurafsky & Martin, 2008)

POS Tagging

The (automatic) assignment of POS tags to word sequences

- non-trivial where words are ambiguous: *fly* (v) vs. *fly* (n)
- choice of the correct tag is *context-dependent*
- useful in pre-processing for parsing, etc; but also directly for text-to-speech synthesis: **con**tent (n) vs. con**tent** (adj)
- difficulty and usefulness can depend on the *tagset*
  - English
    - Penn Treebank (PTB)—45 tags: NNS, NN, NNP, JJ, JJR, JJS

    `http://bulba.sdsu.edu/jeanette/thesis/PennTags.html`
  - Norwegian
    - Oslo-Bergen Tagset—multi-part: ⟨subst appell fem be ent⟩

    `http://tekstlab.uio.no/obt-ny/english/tags.html`

Labeled Sequences

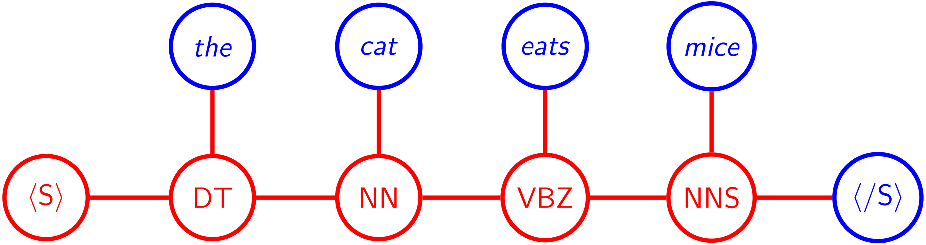
- We are interested in the probability of sequences like:

flies	like	the	wind	or	flies	like	the	wind
NNS	VB	DT	NN		VBZ	P	DT	NN

- In normal text, we see the words, but not the tags.
- Consider the POS tags to be underlying skeleton of the sentence, unseen but influencing the sentence shape.
- A structure like this, consisting of a hidden state sequence, and a related observation sequence can be modelled as a *Hidden Markov Model*.

Hidden Markov Models

The generative story:



$$P(S,O) = P(\text{DT}|\langle S\rangle)\ P(\text{the}|\text{DT})\ P(\text{NN}|\text{DT})\ P(\text{cat}|\text{NN})$$
$$P(\text{VBZ}|\text{NN})\ P(\text{eats}|\text{VBZ})\ P(\text{NNS}|\text{VBZ})\ P(\text{mice}|\text{NNS})$$
$$P(\langle /S\rangle|\text{NNS})$$

Hidden Markov Models

For a bi-gram HMM, with observations O_1^N:

$$P(S,O)=\prod_{i=1}^{N+1} P(s_i|s_{i-1})P(o_i|s_i) \quad \text{where} \quad s_0=\langle S\rangle,\ s_{N+1}=\langle /S\rangle$$

- The transition probabilities model the probabilities of moving from state to state.
- The emission probabilities model the probability that a state *emits* a particular observation.

Using HMMs

The HMM models the process of generating the labeled sequence. We can use this model for a number of tasks:

- $P(S, O)$ given S and O
- $P(O)$ given O
- S that maximizes $P(S|O)$ given O
- $P(s_x|O)$ given O

- We can learn the model parameters, given labeled observations.

Our observations will be words (w_i), and our states PoS tags (t_i).

Estimation

As so often in NLP, we learn an HMM from labeled data:

Transition Probabilities

Based on a training corpus of previously tagged text, with tags as our states, the MLE can be computed from the counts of observed tags:

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

Emission Probabilities

Computed from relative frequencies in the same way, with the words as observations:

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

Implementation Considerations

$$
\begin{aligned}
P(S, O) &= P(s_1|\langle S\rangle)P(o_1|s_1)P(s_2|s_1)P(o_2|s_2)P(s_3|s_2)P(o_3|s_3)\ldots \\
&= 0.0429 \times 0.0031 \times 0.0044 \times 0.0001 \times 0.0072 \times \ldots
\end{aligned}
$$

- Multiplying many small probabilities $\rightarrow$ risk of numeric underflow
- Solution: work in log(arithmic) space:
  - $\log(AB) = \log(A) + \log(B)$
  - hence $P(A)P(B) = \exp(\log(A) + \log(B))$
  - $\log(P(S, O)) = -1.368 + -2.509 + -2.357 + -4 + -2.143 + \ldots$

Still, the issues related to MLE that we discussed for n-gram models also apply here ...

Ice Cream and Global Warming

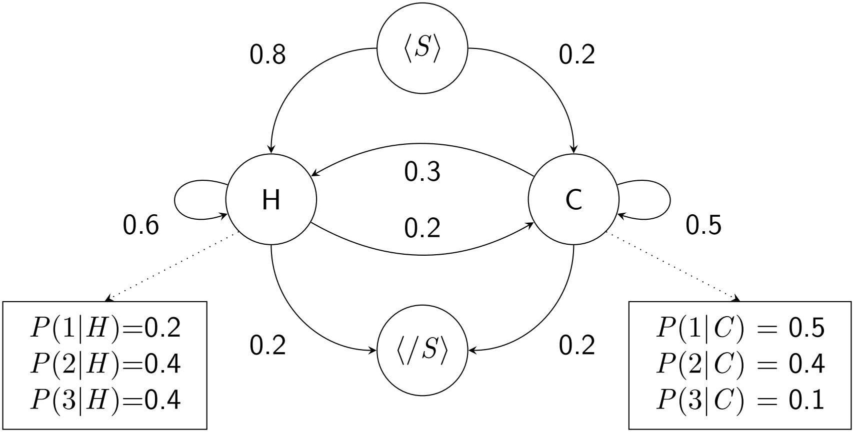
Missing records of weather in Baltimore for Summer 2007

- Jason likes to eat ice cream.
- He records his daily ice cream consumption in his diary.
- The number of ice creams he ate was influenced, but not entirely determined by the weather.
- Today's weather is partially predictable from yesterday's.

A Hidden Markov Model

with:

- Hidden states: $\{H, C\}$ (plus pseudo-states $\langle S \rangle$ and $\langle /S \rangle$)
- Observations: $\{1, 2, 3\}$



⟨S⟩
0.8
0.2
H
C
0.3
0.2
0.6
0.5
⟨/S⟩
0.2
0.2
P(1|H)=0.2
P(2|H)=0.4
P(3|H)=0.4
P(1|C) = 0.5
P(2|C) = 0.4
P(3|C) = 0.1

Using HMMs

The HMM models the process of generating the labeled sequence. We can use this model for a number of tasks:

- $P(S, O)$ given S and O
- $P(O)$ given O
- S that maximizes $P(S|O)$ given O
- $P(s_x|O)$ given O
- We can also learn the model parameters, given a set of observations.

Part-of-Speech Tagging

We want to find the tag sequence, given a word sequence. With tags as our states and words as our observations, we know:

$$P(S, O) = \prod_{i=1}^{N+1} P(s_i|s_{i-1})P(o_i|s_i)$$

We want: $P(S|O) = \dfrac{P(S, O)}{P(O)}$

Actually, we want the state sequence $\widehat{S}$ that maximizes $P(S|O)$:

$$\widehat{S} = \arg\max_{S} \frac{P(S, O)}{P(O)}$$

Since $P(O)$ always is the same, we can drop the denominator.

Decoding

Task

Find the most likely state sequence $\widehat{S}$, given an observation sequence O.

HMM

$P(H\|\langle S\rangle) = 0.8$	$P(C\|\langle S\rangle) = 0.2$
$P(H\|H) = 0.6$	$P(C\|H) = 0.2$
$P(H\|C) = 0.3$	$P(C\|C) = 0.5$
$P(\langle /S\rangle\|H) = 0.2$	$P(\langle /S\rangle\|C) = 0.2$
$P(1\|H) = 0.2$	$P(1\|C) = 0.5$
$P(2\|H) = 0.4$	$P(2\|C) = 0.4$
$P(3\|H) = 0.4$	$P(3\|C) = 0.1$

if $O = 3\ 1\ 3$

$\langle S\rangle$	H	H	H	$\langle /S\rangle$	0.0018432
$\langle S\rangle$	H	H	C	$\langle /S\rangle$	0.0001536
$\langle S\rangle$	H	C	H	$\langle /S\rangle$	0.0007680
$\langle S\rangle$	H	C	C	$\langle /S\rangle$	0.0003200
$\langle S\rangle$	C	H	H	$\langle /S\rangle$	0.0000576
$\langle S\rangle$	C	H	C	$\langle /S\rangle$	0.0000048
$\langle S\rangle$	C	C	H	$\langle /S\rangle$	0.0001200
$\langle S\rangle$	C	C	C	$\langle /S\rangle$	0.0000500

Dynamic Programming

For (only) two states and a (short) observation sequence of length three, comparing all possible sequences may be workable, but . . .

- for N observations and L states, there are L^N sequences;
- we end up doing the same partial calculations over and over again.

Dynamic Programming:

- records sub-problem solutions for further re-use
- useful when a complex problem can be described recursively
- examples: Dijkstra's shortest path, minimum edit distance, longest common subsequence, Viterbi algorithm

Viterbi Algorithm

Recall our problem:

$$\text{maximize } P(s_1 \ldots s_n | o_1 \ldots o_n) = P(s_1|s_0)P(o_1|s_1)P(s_2|s_1)P(o_2|s_2)\ldots$$

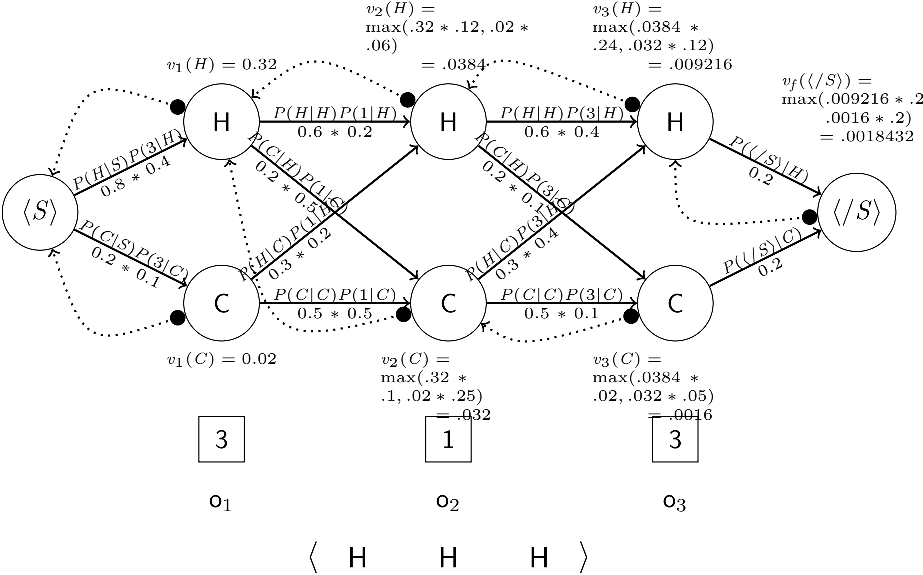
Our recursive sub-problem:

$$v_i(x) = \max_{k=1}^{L} \left[v_{i-1}(k) \cdot P(x|k) \cdot P(o_i|x) \right]$$

The variable $v_i(x)$ represents the maximum probability that the i-th state is x, given that we have seen O_1^i.

At each step, we record backpointers showing which previous state led to the maximum probability.

An Example of the Viterbi Algorithmn



Pseudocode for the Viterbi Algorithm

Input: *observations* of length N, *state set* of size L
Output: *best-path*
create a path probability matrix $viterbi[N, L+1]$
create a path backpointer matrix $backpointer[N, L+1]$
foreach state s from 1 to L **do**
    $viterbi[1, s] \leftarrow trans(\langle S \rangle, s) \times emit(o_1, s)$
    $backpointer[1, s] \leftarrow 0$
end
foreach time step i from 2 to N **do**
    foreach state s from 1 to L **do**
        $viterbi[i, s] \leftarrow \max_{s'=1}^{L} viterbi[i-1, s'] \times trans(s', s) \times emit(o_i, s)$
        $backpointer[i, s] \leftarrow \arg\max_{s'=1}^{L} viterbi[i-1, s'] \times trans(s', s)$
    end
end
$viterbi[N, L+1] \leftarrow \max_{s=1}^{L} viterbi[N, s] \times trans(s, \langle /S \rangle)$
$backpointer[N, L+1] \leftarrow \arg\max_{s=1}^{L} viterbi[N, s] \times trans(s, \langle /S \rangle)$
return the path by following backpointers from $backpointer[N, L+1]$

Diversion: Complexity and O(N)

Big-O notation describes the complexity of an algorithm.

- it describes the worst-case *order of growth* in terms of the size of the input
- only the largest order term is represented
- constant factors are ignored
- determined by looking at loops in the code

Pseudocode for the Viterbi Algorithm

Input: *observations* of length N, *state set* of length L
Output: *best-path*
create a path probability matrix $viterbi[N, L+1]$
create a path backpointer matrix $backpointer[N, L+1]$
foreach state s from 1 to L **do** L
    $viterbi[1, s] \leftarrow trans(\langle S \rangle, s) \times emit(o_1, s)$
    $backpointer[1, s] \leftarrow 0$
end
foreach time step i from 2 to N **do** N
    foreach state s from 1 to L **do** L
        $viterbi[i, s] \leftarrow \max_{s'=1}^{L} viterbi[i-1, s'] \times trans(s', s) \times emit(o_i, s)$ L
        $backpointer[i, s] \leftarrow \arg\max_{s'=1}^{L} viterbi[i-1, s'] \times trans(s', s)$
    end
end
$viterbi[N, L+1] \leftarrow \max_{s=1}^{L} viterbi[N, 1] \times trans(s, \langle /S \rangle)$
$backpointer[N, L+1] \leftarrow \arg\max_{s=1}^{L} viterbi[N, 1] \times trans(s, \langle /S \rangle)$
return the path by following backpointers from $backpointer[N, L+1]$ N

$$O(L^2 N)$$