Introduction to Natural Language Processing

a course taught as B4M36NLP at Open Informatics

by members of the Institute of Formal and Applied Linguistics

Today: **Week 3, lecture**
Today's topic: **Markov Models**
Today's teacher: **Jan Hajič**

E-mail: hajic@ufal.mff.cuni.cz
WWW: http://ufal.mff.cuni.cz/jan-hajic

Review: Markov Process

- Bayes formula (chain rule):

$$P(W) = P(w_1,w_2,\ldots,w_T) = \Pi_{i=1..T}\ p(w_i|w_1,\cancel{w_2,\ldots,}w_{i-n+1},\ldots,w_{i-1})$$

approximation

- n-gram language models:
  - Markov process (chain) of the order n-1:

$$P(W) = P(w_1,w_2,\ldots,w_T) = \Pi_{i=1..T}\ p(w_i|w_{i-n+1},w_{i-n+2},\ldots,w_{i-1})$$

Using just <u>one</u> distribution (Ex.: trigram model: $p(w_i|w_{i-2},w_{i-1})$):

Positions:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Words:	My	car	**broke**	**down**	**,**	and	within	hours	Bob	's	car	**broke**	**down**	**,**	too	.

$$p(\mathbf{,}|\mathbf{broke\ down}) = p(w_5|w_3,w_4)) = p(w_{14}|w_{12},w_{13})$$

Markov Properties

- Generalize to any process (not just words/LM):
  - Sequence of random variables: $X = (X_1, X_2, \ldots, X_T)$
  - Sample space S (*states*), size N: $S = \{s_0, s_1, s_2, \ldots, s_N\}$

1. Limited History (Context, Horizon):

$$\forall i \in 1..T;\ P(X_i|X_1,\ldots,X_{i-1}) = P(X_i|X_{i-1})$$

1 7 3 7 9 0 6 7 3 4 5...   1 7 3 7 9 0 6 7 7 4 5...

2. Time invariance (M.C. is stationary, homogeneous)

$$\forall i \in 1..T,\ \forall y,x \in S;\ P(X_i{=}y|X_{i-1}{=}x) = p(y|x)$$

1 7 3 7 9 0 6 7 3 4 5...

? ok...same ***distribution***

Long History Possible

- What if we want trigrams:

  1 7 3 7 9 0 6 7 3 4 5...

- Formally, use transformation:

  Define new variables Q_i, such that $X_i = \{Q_{i-1}, Q_i\}$:

  Then

  $$P(X_i|X_{i-1}) = P(Q_{i-1},Q_i|Q_{i-2},Q_{i-1}) = P(Q_i|Q_{i-2},Q_{i-1})$$

Predicting (X_i): 1 7 3 7 9 0 6 7 3 4 5...

. 1 7 3 .... 0 6 7 3 4

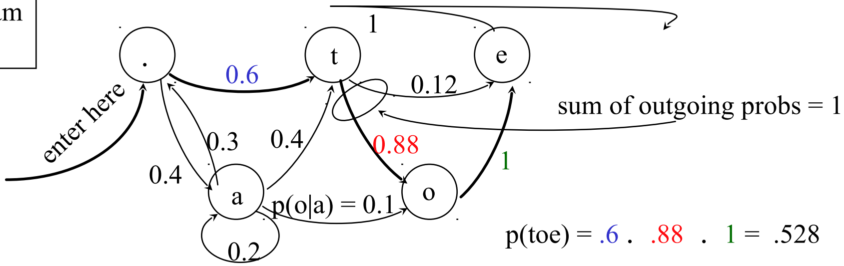
9

History ($X_{i-1} = \{Q_{i-2},Q_{i-1}\}$): . . 1 7 .... 9 0 6 7 3

Graph Representation: State Diagram

- $S = \{s_0, s_1, s_2, \ldots, s_N\}$: states
- Distribution $P(X_i|X_{i-1})$:
  - **transitions (as arcs) with probabilities attached to them:**

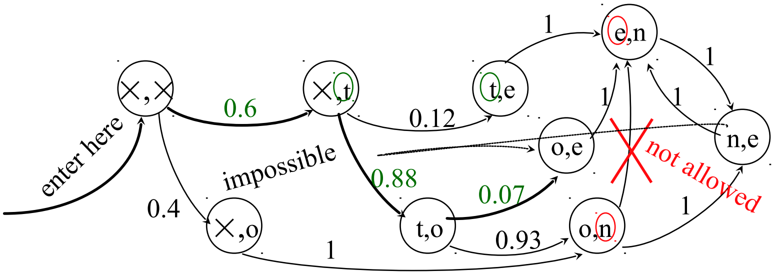
Bigram case:



p(toe) = .6 . .88 . 1 = .528

The Trigram Case

- $S = \{s_0,s_1,s_2,\ldots,s_N\}$: states: pairs $s_i = (x,y)$
- Distribution $P(X_i|X_{i-1})$: (r.v. X: generates pairs s_i)



p(toe) = .6 . .88 . .07 ≅ .037

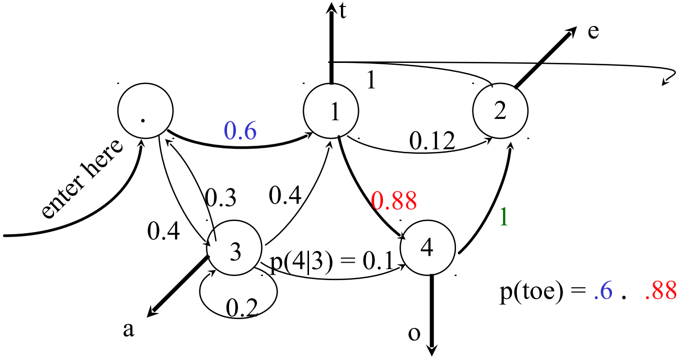
p(one) = ?

Finite State Automaton

- States ~ symbols of the [input/output] alphabet
  - pairs (or more): last element of the n-tuple
- Arcs ~ transitions (sequence of states)
- [Classical FSA: alphabet symbols on arcs:
  - transformation: arcs ↔ nodes]
- Possible thanks to the "limited history" M'ov Property
- So far: ***<u>Visible</u>*** Markov Models (VMM)

Hidden Markov Models

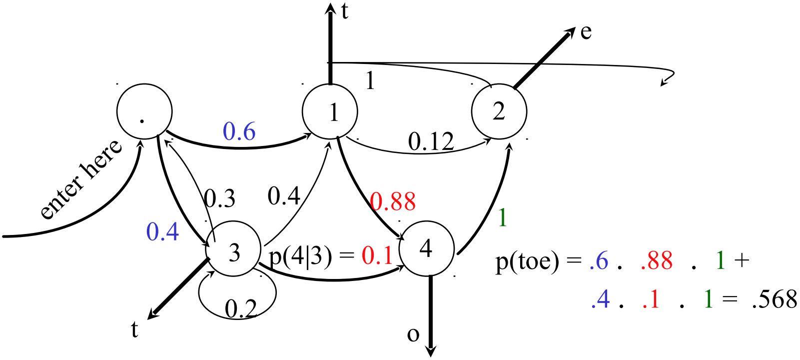
- The simplest HMM: states generate [observable] output (using the “data” alphabet) but remain “invisible”:



p(toe) = .6 . .88 . 1 = .528

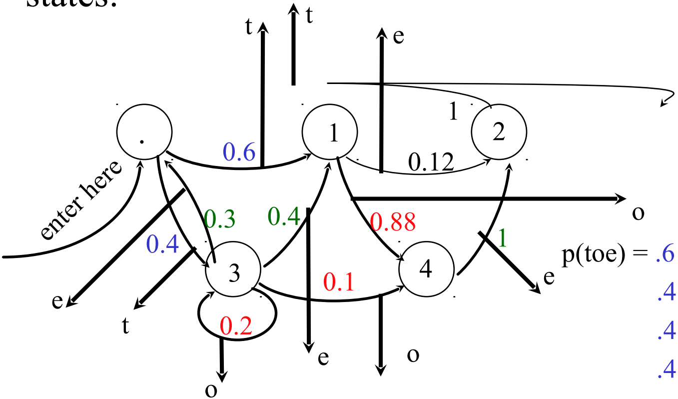
Added Flexibility

- So far, no change; but different states may generate the same output (why not?):



Output from Arcs...

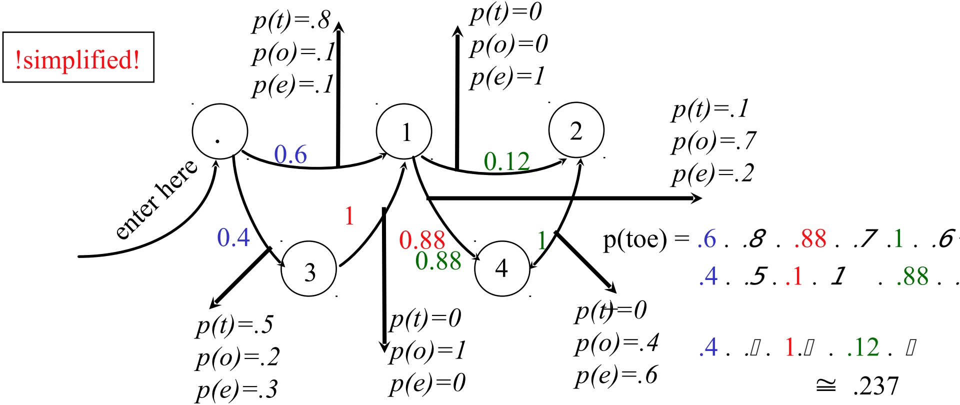
- Added flexibility: Generate output from arcs, not states:



$$p(toe) = .6 \cdot .88 \cdot 1 + .4 \cdot .1 \cdot 1 + .4 \cdot .2 \cdot .3 + .4 \cdot .2 \cdot .4 = .624$$

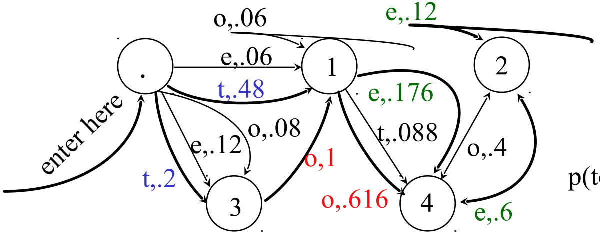
... and Finally, Add Output Probabilities

- Maximum flexibility: [Unigram] distribution (sample space: output alphabet) at each output arc:



Slightly Different View

- Allow for multiple arcs from $s_i \rightarrow s_j$, mark them by output symbols, get rid of output distributions:



p(toe) = .48 . .616 . .6+
.2 . 1 . .176 +
.2 . 1 . .12 ≅ .237

In the future, we will use the view more convenient for the problem at hand.

Formalization

- HMM (the most general case):
  - five-tuple (S, s_0, Y, P_S, P_Y), where:
    - **$S = \{s_0,s_1,s_2,\ldots,s_T\}$ is the set of states, s_0 is the initial state,**
    - **$Y = \{y_1,y_2,\ldots,y_V\}$ is the output alphabet,**
    - **$P_S(s_j|s_i)$ is the set of prob. distributions of transitions,**
      - size of P_S: $|S|^2$.
    - **$P_Y(y_k|s_i,s_j)$ is the set of output (emission) probability distributions.**
      - size of P_Y: $|S|^2 \times |Y|$
- Example:
  - $S = \{x, 1, 2, 3, 4\}$, $s_0 = x$
  - $Y = \{ t, o, e \}$

Formalization - Example

- Example:
  - $S = \{x, 1, 2, 3, 4\}$, $s_0 = x$
  - $Y = \{ e, o, t \}$
  - P_S:

	x	1	2	3	4
x	0	.6	0	.4	0
1	0	0	.12	0	.88
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	1	0	0

$\longrightarrow \Sigma = 1$

P_Y:

e	x	1	2	3	4
x					.2
1					
2					
3					
4					

o	x	1	2	3	4
x					.7
1					
2					
3					
4					

t	x	1	2	3	4
x		.8		.5	
1			0		.1
2					0
3		0			
4			0		

$\Sigma = 1$

Using the HMM

- The generation algorithm (of limited value :-)):
  1. Start in $s = s_0$.
  2. Move from s to s' with probability $P_S(s'|s)$.
  3. Output (emit) symbol y_k with probability $P_S(y_k|s,s')$.
  4. Repeat from step 2 (until somebody says enough).
- More interesting usage:
  - Given an output sequence $Y = \{y_1,y_2,\ldots,y_k\}$, compute its probability.
  - Given an output sequence $Y = \{y_1,y_2,\ldots,y_k\}$, compute the most likely sequence of states which has generated it.
  - ...plus variations: e.g., n best state sequences

HMM Algorithms: Trellis and Viterbi

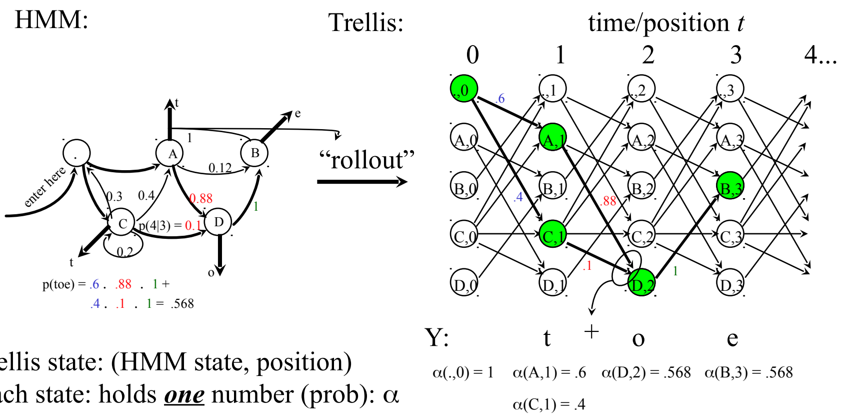
HMM: The Two Tasks

- HMM (the general case):
  - five-tuple (S, S_0, Y, P_S, P_Y), where:
    - **$S = \{s_1,s_2,\ldots,s_T\}$ is the set of states, S_0 is the initial state,**
    - **$Y = \{y_1,y_2,\ldots,y_V\}$ is the output alphabet,**
    - **$P_S(s_j|s_i)$ is the set of prob. distributions of transitions,**
    - **$P_Y(y_k|s_i,s_j)$ is the set of output (emission) probability distributions.**
- Given an HMM & an output sequence $Y = \{y_1,y_2,\ldots,y_k\}$:

  (Task 1) compute the probability of Y;

  (Task 2) compute the most likely sequence of states which has generated Y.

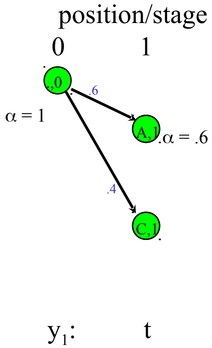
Trellis - Deterministic Output



- trellis state: (HMM state, position)
- each state: holds ***one*** number (prob): α
- probability or Y: Σα in the last state

Creating the Trellis: The Start

- Start in the start state (.),
  - set its $\alpha(.,0)$ to 1.
- Create the first stage:
  - get the first "output" symbol y_1
  - create the first stage (column)
  - but only those trellis states
    which generate y_1
  - set their $\alpha(state,1)$ to the $P_S(state|.)\ \underbrace{\alpha(.,0)}_{1}$
- ...and forget about the *0*-th stage

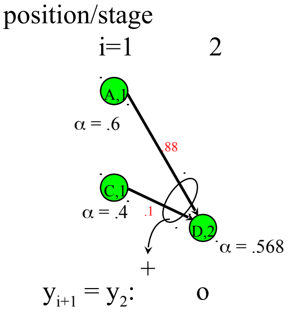


Trellis: The Next Step

- Suppose we are in stage *i*
- Creating the next stage:
  - create all trellis states in the next stage which generate y_{i+1}, but only those reachable from any of the stage-*i* states
  - set their $\alpha(state, i+1)$ to:

    $P_S(state|prev.state) \;.\; \alpha(prev.state, i)$

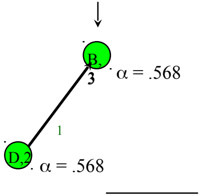
    (add up all such numbers on arcs going to a common trellis state)
  - ...and forget about stage *i*



Trellis: The Last Step

- Continue until "output" exhausted
  - $|Y| = 3$: until stage 3
- Add together all the $\alpha(state,|Y|)$
- That's the P(Y).
- Observation (pleasant):
  - memory usage max: $2|S|$
  - multiplications max: $|S|^2|Y|$

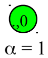
last position/stage

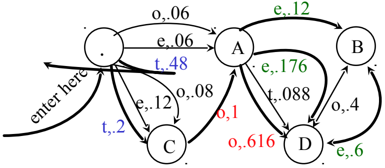


P(Y) = .568

Trellis: The General Case (still, bigrams)

- Start as usual:
  - start state (.), set its $\alpha(.,0)$ to 1.

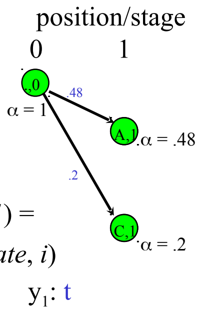


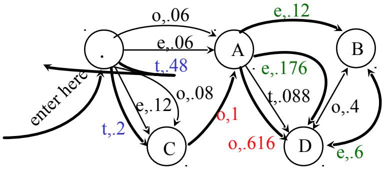


$p(\text{toe}) = .48 \cdot .616 \cdot .6 +$
$.2 \cdot 1 \cdot .176 +$
$.2 \cdot 1 \cdot .12 \cong .237$

General Trellis: The Next Step

- We are in stage i :
  - Generate the next stage $i+1$ as before (except now <u>arcs</u> generate output, thus use only those arcs marked by the output symbol y_{i+1})
  - For each generated *state*, compute $\alpha(state, i+1) =$
    $= \Sigma_{\text{incoming arcs}} P_Y(y_{i+1}|state, prev.state)\ .\ \alpha(prev.state, i)$

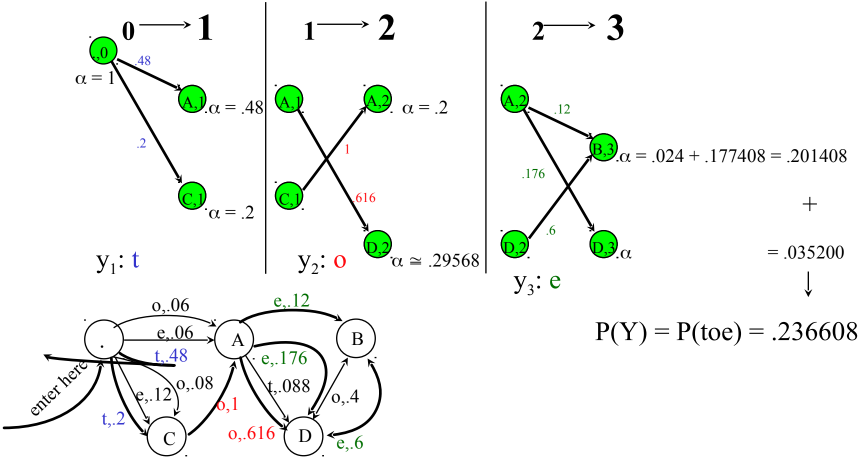




...and forget about stage i as usual.

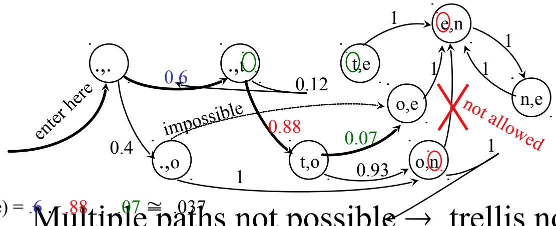
Trellis: The Complete Example

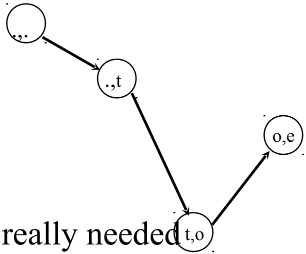
Stage:



The Case of Trigrams

- Like before, but:
  - states correspond to bigrams,
  - output function always emits the second output symbol of the pair (state) to which the arc goes:





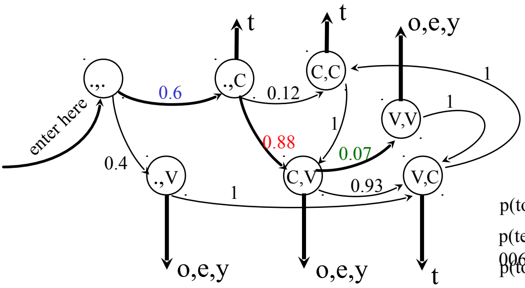
p(toe) = .6 · .88 · .07 ≅ .037

Multiple paths not possible → trellis not really needed

Trigrams with Classes

- More interesting:
  - n-gram class LM: $p(w_i|w_{i-2},w_{i-1}) = p(w_i|c_i)\, p(c_i|c_{i-2},c_{i-1})$

    → states are pairs of classes (c_{i-1},c_i), and emit “words”: (letters in our example)



$p(t|C) = 1$ usual,
$p(o|V) = .3$ non-
$p(e|V) = .6$ overlapping
$p(y|V) = .1$ classes

p(toe) = .6 . . .88 . . . .07 . . ≅ .00665

p(teo) = .6 . 1 . .88 . .6 . .07 . .3 ≅ .00665

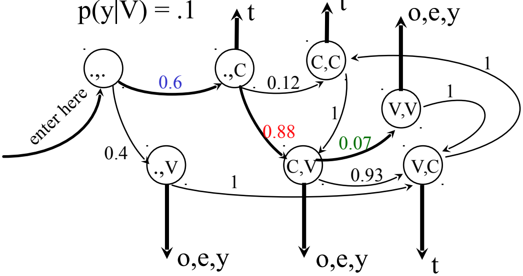
p(toy) = .6 . . .88 . . . .07 . . ≅ .00111

p(tty) = .6 . . .12 . . 1 . . ≅ .0072

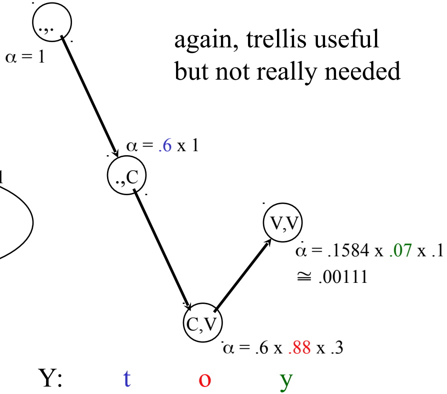
Class Trigrams: the Trellis

- Trellis generation (Y = “toy”):

p(t|C) = 1
p(o|V) = .3
p(e|V) = .6
p(y|V) = .1

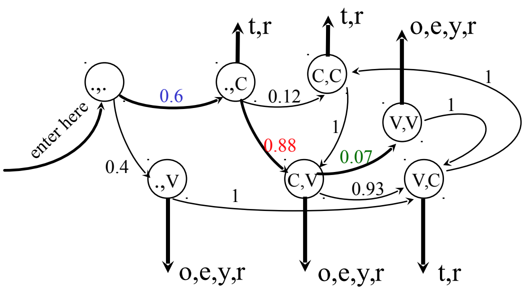


again, trellis useful
but not really needed



Overlapping Classes

- Imagine that classes may overlap
  - e.g. ‘r’ is sometimes vowel sometimes consonant, belongs to V as well as C:

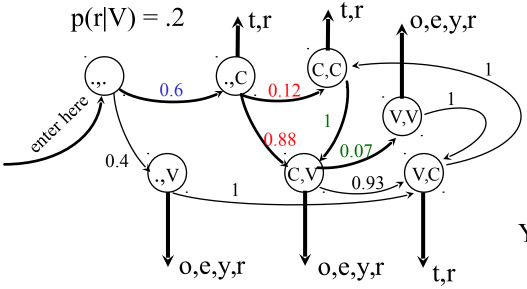


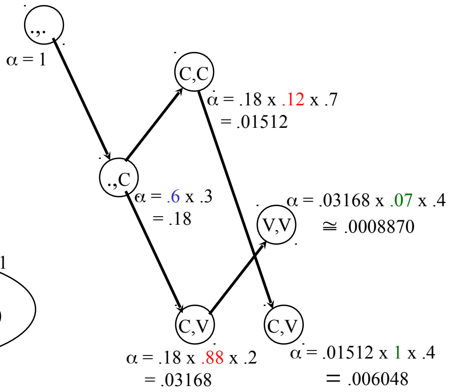
$p(t|C) = .3$
$p(r|C) = .7$
$p(o|V) = .1$
$p(e|V) = .3$
$p(y|V) = .4$
$p(r|V) = .2$

$p(try) = ?$

Overlapping Classes: Trellis Example

$p(t|C) = .3$
$p(r|C) = .7$
$p(o|V) = .1$
$p(e|V) = .3$
$p(y|V) = .4$
$p(r|V) = .2$





Y: t r y p(Y) = .006935

Trellis: Remarks

- So far, we went left to right (computing α)
- Same result: going right to left (computing β)
  - supposed we know where to start (finite data)
- In fact, we might start in the middle going left and right
- Important for parameter estimation

  (Forward-Backward Algortihm alias Baum-Welch)
- Implementation issues:
  - scaling/normalizing probabilities, to avoid too small numbers & addition problems with many transitions

The Viterbi Algorithm

- Solving the task of finding the most likely sequence of states which generated the observed data
- i.e., finding

$$S_{best} = \text{argmax}_S P(S|Y)$$

which is equal to (Y is constant and thus P(Y) is fixed):

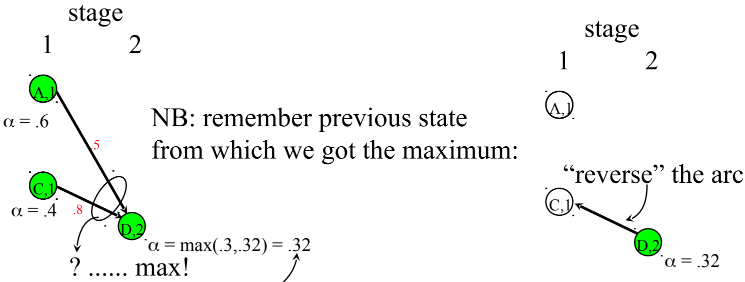
$$S_{best} = \text{argmax}_S P(S,Y) =$$

$$= \text{argmax}_S P(s_0,s_1,s_2,\ldots,s_k,y_1,y_2,\ldots,y_k) =$$

$$= \text{argmax}_S \Pi_{i=1..k}\ p(y_i|s_i,s_{i-1})p(s_i|s_{i-1})$$

The Crucial Observation

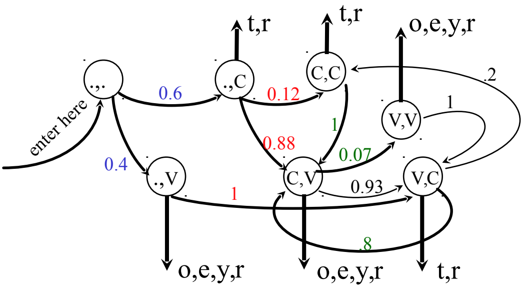
- Imagine the trellis build as before (but do not compute the αs yet; assume they are o.k.); stage *i*:



this is certainly the "backwards" maximum to (D,2)... but
it <u>cannot change even whenever we go forward</u> (M. property: limited history)

Viterbi Example

- ‘r’ classification (C or V?, sequence?):



p(t|C) = .3
p(r|C) = .7
p(o|V) = .1
p(e|V) = .3
p(y|V) = .4
p(r|V) = .2

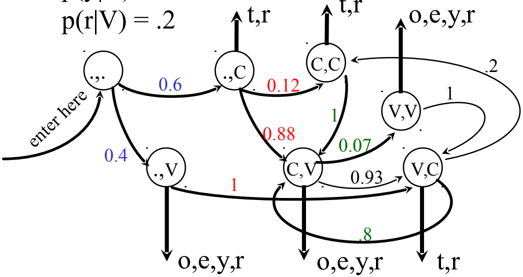
$\text{argmax}_{XYZ}\ p(rry|XYZ) = ?$

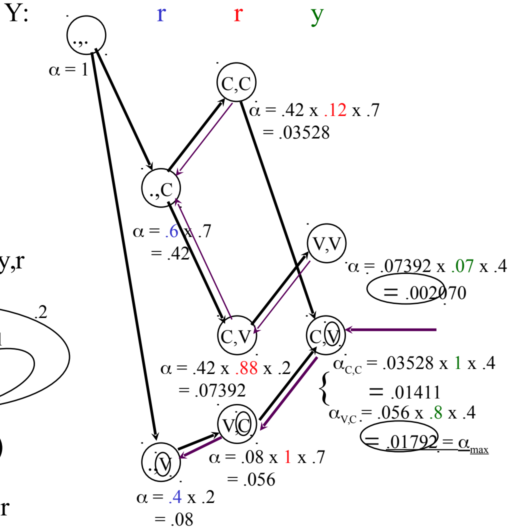
Possible state seq.: (.,v)(v,c)(c,v)[VCV], (.,c)(c,c)(c,v)[CCV], (.,c)(c,v)(v,v) [CVV]

Viterbi Computation

α in trellis state: best prob from start to here

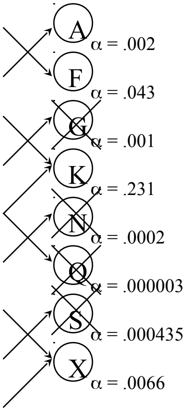
$p(t|C) = .3$
$p(r|C) = .7$
$p(o|V) = .1$
$p(e|V) = .3$
$p(y|V) = .4$
$p(r|V) = .2$





Pruning

- Sometimes, too many trellis states in a stage:



criteria: (a) $\alpha <$ threshold
(b) $\Sigma\pi <$ threshold
(c) # of states > threshold
(get rid of smallest α)

HMM: Parameter Estimation The Baum-Welch Algorithm

- HMM (the general case):
  - five-tuple (S, S_0, Y, P_S, P_Y), where:
    - **$S = \{s_1,s_2,\ldots,s_T\}$ is the set of states, S_0 is the initial state,**
    - **$Y = \{y_1,y_2,\ldots,y_V\}$ is the output alphabet,**
    - **$P_S(s_j|s_i)$ is the set of prob. distributions of transitions,**
    - **$P_Y(y_k|s_i,s_j)$ is the set of output (emission) probability distributions.**
- Given an HMM & an output sequence $Y = \{y_1,y_2,\ldots,y_k\}$:

✓(Task 1) compute the probability of Y;

✓(Task 2) compute the most likely sequence of states which has generated Y.

(Task 3) Estimating the parameters (transition/output distributions)

unsupervised (supervised: trivial, use relative frequencies)

A Variant of EM

- Idea (~ EM, for another variant see LM smoothing):
  - Start with (possibly random) estimates of P_S and P_Y.
  - Compute (fractional) "counts" of state transitions/emissions taken, from P_S and P_Y, given data Y.
  - Adjust the estimates of P_S and P_Y from these "counts" (using the MLE, i.e. relative frequency as the estimate).
- Remarks:
  - many more parameters than the simple four-way smoothing
  - no proofs here; see Jelinek, Chapter 9

Setting

- HMM (without P_S, P_Y) (S, S_0, Y), and data T = $\{y^i \in Y\}_{i=1..|T|}$
    - **will use T ~ |T|**
  - HMM structure is given: (S, S_0)
  - P_S:Typically, one wants to allow "fully connected" graph
    - **(i.e. no transitions forbidden ~ no transitions set to hard 0)**
    - **why? → we better leave it on the learning phase, based on the data!**
    - **sometimes possible to remove some transitions ahead of time**
  - P_Y: should be restricted (if not, we will not get anywhere!)
    - **restricted ~ hard 0 probabilities of p(y|s,s')**
    - **"Dictionary": states ↔ words, "m:n" mapping on S × Y (in general)s**

Initialization

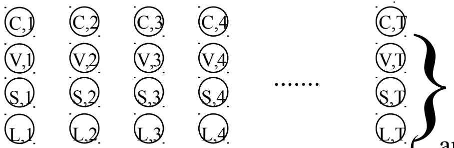
- For computing the initial expected "counts"
- Important part
  - EM guaranteed to find a *local* maximum only (albeit a good one in most cases)
- P_Y initialization more important
  - fortunately, often easy to determine
    - **together with dictionary ↔ vocabulary mapping, get counts, then MLE**
- P_S initialization less important
  - e.g. uniform distribution for each p(.|s)

Data Structures

- Will need storage for:
  - The predetermined structure of the HMM

    (unless fully connected → need not to keep it!)
  - The parameters to be estimated (P_S, P_Y)
  - The expected counts (same size as P_S, P_Y)
  - The training data $T = \{y^i \in Y\}_{i=1..T}$
  - The trellis (if f.c.): ↑ T Size: T . S (Precisely, |T|.|S|)

Each trellis state:
two [float] numbers
(forward/backward)



(...and then some)

The Algorithm Part I

1. Initialize P_S, P_Y
2. Compute "forward" probabilities:
   - **follow the procedure for trellis (summing), compute $\alpha(s,i)$**
   - **use the current values of P_S, P_Y ($p(s'|s)$, $p(y|s,s')$):**

     $$\alpha(s',i) = \sum_{s \to s'} \alpha(s,i-1) \,.\, p(s'|s) \,.\, p(y_i|s,s')$$
   - **NB: do not throw away the previous stage!**
3. Compute "backward" probabilities
   - **start at all nodes of the last stage, proceed backwards, $\beta(s,i)$**
   - **i.e., probability of the "tail" of data from stage *i* to the end of data**

     $$\beta(s',i) = \sum_{s \leftarrow s'} \beta(s,i+1) \,.\, p(s|s') \,.\, p(y_{i+1}|s',s)$$
   - **also, keep the $\beta(s,i)$ at all trellis states**

The Algorithm Part II

4. Collect counts:
   - for each output/transition pair compute

     $c(y,s,s') = \Sigma_{i=0..k-1,y=y_{i+1}} \alpha(s,i)\ p(s'|s)\ p(y_{i+1}|s,s')\ \beta(s',i+1)$

     one pass through data, only stop at (output) y

     prefix prob.

     this transition prob · output prob

     tail prob

     $c(s,s') = \Sigma_{y\in Y}\ c(y,s,s')$ (assuming all observed y_i in Y)

     $c(s) = \Sigma_{s'\in S}\ c(s,s')$

5. Reestimate: $p'(s'|s) = c(s,s')/c(s)$ $\quad p'(y|s,s') = c(y,s,s')/c(s,s')$

6. Repeat 2-5 until desired convergence limit is reached.

Baum-Welch: Tips & Tricks

- Normalization badly needed
  - long training data $\rightarrow$ extremely small probabilities
- Normalize α,β using the same norm. factor:

$$N(i) = \Sigma_{s \in S}\ \alpha(s,i)$$

as follows:

- **compute $\alpha(s,i)$ as usual (Step 2 of the algorithm), computing the sum N(i) at the given stage *i* as you go.**
- **at the end of each stage, recompute all αs (for each state s):**

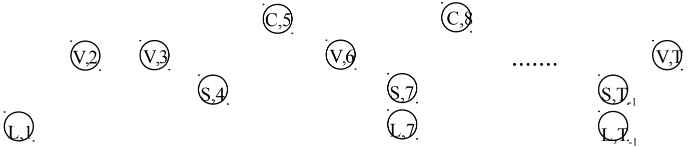
  $\alpha^*(s,i) = \alpha(s,i) / N(i)$
- **use the same N(i) for βs at the end of each backward (Step 3) stage:**

  $\beta^*(s,i) = \beta(s,i) / N(i)$

Example

- Task: pronunciation of "the"
- Solution: build HMM, fully connected, 4 states:
  - **S - short article, L - long article, C,V - starting w/consonant, vowel**
  - **thus, only "the" is ambiguous (a, an, the - not members of C,V)**
- Output from states only (p(w|s,s') = p(w|s'))
- Data Y: an egg and a piece of the big .... the end

Trellis:



Example: Initialization

- Output probabilities:

  $p_{init}(w|c) = c(c,w) / c(c)$; where $c(S,the) = c(L,the) = c(the)/2$

  (other than that, everything is deterministic)
- Transition probabilities:
  - $p_{init}(c'|c) = 1/4$ (uniform)
- Don't forget:
  - about the space needed
  - initialize $\alpha(X,0) = 1$ (X : the never-occurring front buffer st.)
  - initialize $\beta(s,T) = 1$ for all s (except for $s = X$)

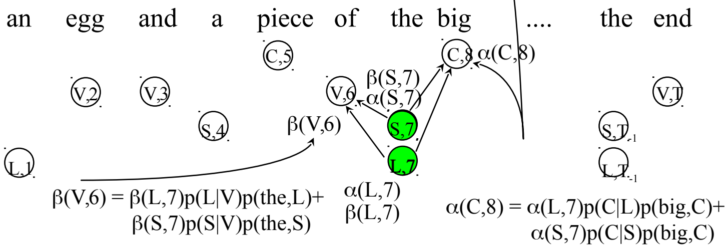
Fill in alpha, beta

- Left to right, alpha:

$$\alpha(s',i) = \sum_{s \to s'} \alpha(s,i\text{-}1)\ .\ p(s'|s)\ .\ p(w_i|s')$$

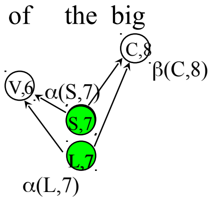
output from states

- Remember normalization (N(i)).
- Similarly, beta (on the way back from the end).



Counts & Reestimation

- One pass through data
- At each position *i*, go through all pairs (s_i, s_{i+1})
- Increment appropriate counters by frac. counts (Step 4):
  - **$\mathrm{inc}(y_{i+1}, s_i, s_{i+1}) = a(s_i, i)\ p(s_{i+1}|s_i)\ p(y_{i+1}|s_{i+1})\ b(s_{i+1}, i+1)$**
  - **$c(y, s_i, s_{i+1})$ += inc (for y at pos i+1)**
  - **$c(s_i, s_{i+1})$ += inc (always)**
  - **$c(s_i)$ += inc (always)**



- Reestimate $p(s'|s)$, $p(y|s)$
  - **and hope for increase in p(C|S) and p(V|L)...!!**

inc(big,L,C) $= \alpha(L,7)p(C|L)p(big,C)\beta(C,8)$

inc(big,S,C) $= \alpha(S,7)p(C|S)p(big,C)\beta(C,8)$

HMM: Final Remarks

- Parameter "tying":
  - keep certain parameters same (~ just one "counter" for all of them)
  - any combination in principle possible
  - ex.: smoothing (just one set of lambdas)
- Real Numbers Output
  - Y of infinite size (R, R^n):
    - **parametric (typically: few) distribution needed (e.g., "Gaussian")**
- "Empty" transitions: do not generate output
    - **~ vertical arcs in trellis; do not use in "counting"**