Statistical Methods in Natural Language Processing

9. Hidden Markov Models and Baum Welch.

Pavel Pecina, Jindřich Helcl

16 December, 2025

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Course Segments

1. Introduction, probability, essential information theory
2. Statistical language modelling (n-gram)
3. Statistical properties of words
4. Word representations
5. Hidden Markov models, Tagging

Recap from Last Week

Markov Properties

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

1. Limited history (context, horizon):

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

$\boxed{1\ 7\ 3\ 7\ 9\ 0\ 6\ 7}\ \boxed{3}\ 4\ 5\ldots$ 1 7 3 7 9 0 6 $\boxed{7}\ \boxed{3}\ 4\ 5\ldots$

2. Time invariance (Markov Chain is <u>stationary</u>, <u>homogeneous</u>)

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

$1\ \boxed{7}\ \boxed{3}\ \boxed{7}\ \boxed{9}\ 0\ 6\ \boxed{7}\ \boxed{3}\ 4\ 5\ldots$

ok ... same <u>distribution</u>

Long History Possible

- What if we want trigrams:

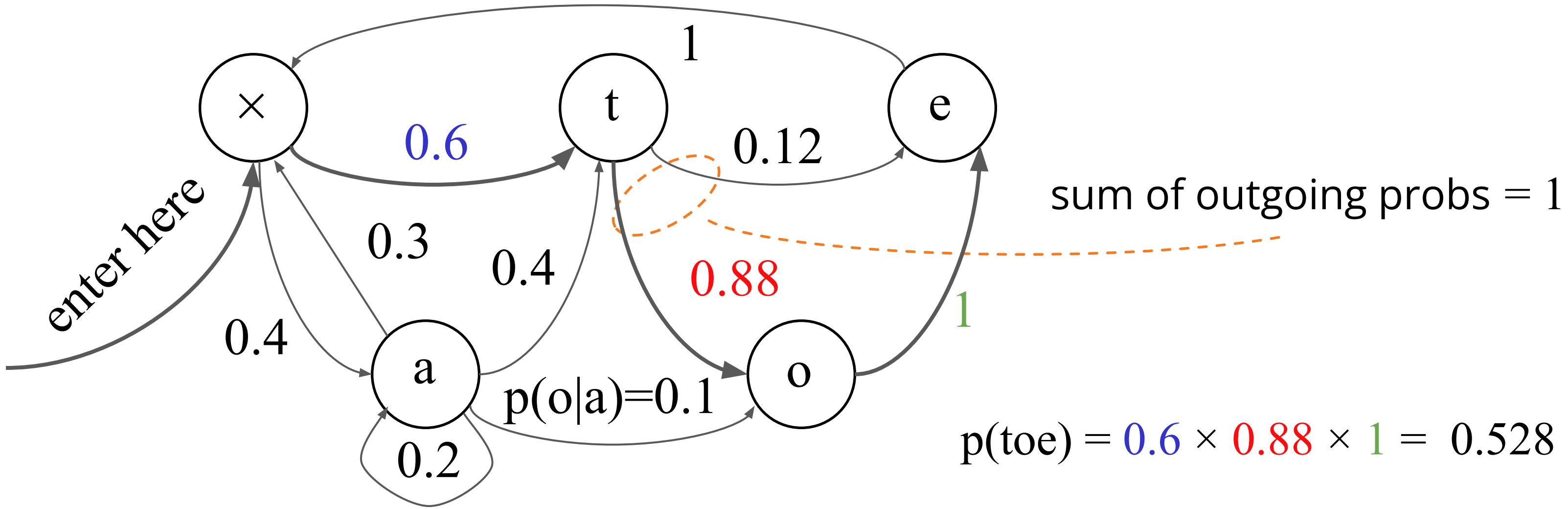
  1 7 3 7 9 0 6 7 3 $\boxed{4\ 5}$ $\boxed{\ldots}$

- Formally, use transformation:
  - Define new variables Q_i, such that $X_i = \{Q_{i-1},Q_i\}$
  - And 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	$\boxed{9\ 0}$	6	7	3	4	5	…
History ($X_{i-1} = \{Q_{i-2},Q_{i-1}\}$):	×	1	7	3	…	0	6	7	3	4	…
	×	×	1	7	…	9	0	6	7	3	…

Markov Models: Bigram Case

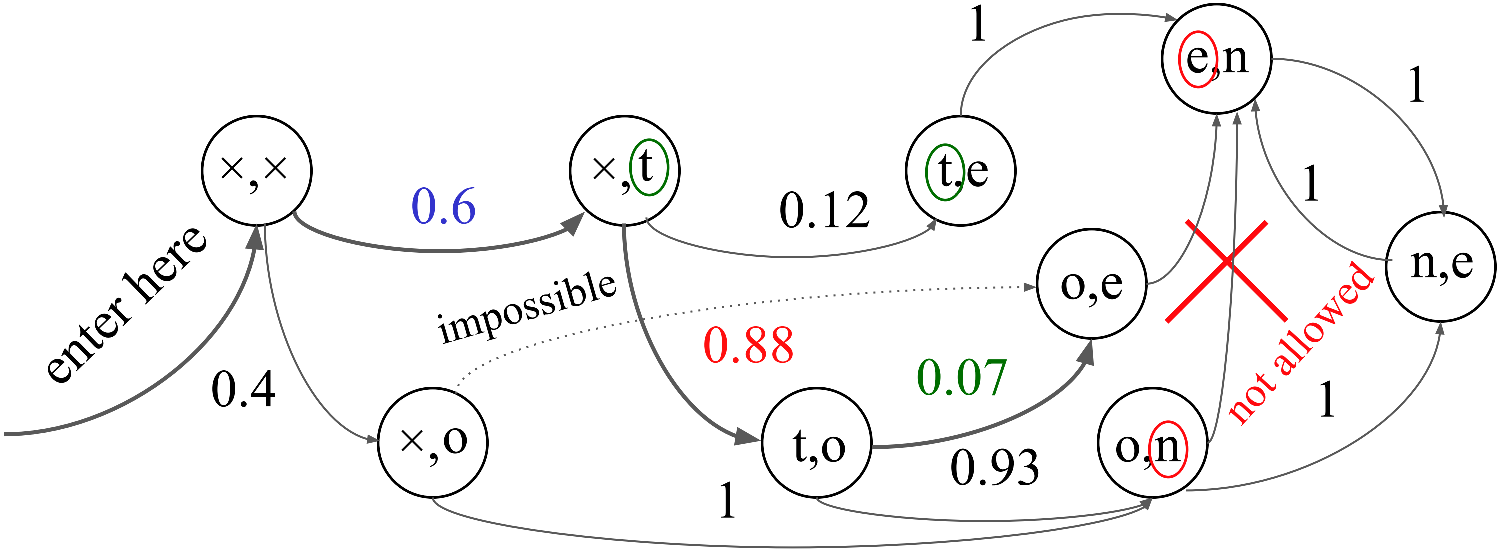
- Nodes: States, $S = \{s_0, s_1, s_2, \ldots, s_N\}$
- Arcs: Transitions with probabilities, $P(X_i|X_{i-1})$, X_i generates s_i



$p(toe) = 0.6 \times 0.88 \times 1 = 0.528$

Markov Models: Trigram Case

- Nodes: Pairs of states (s_k,s_l), $S = \{s_0,s_1,s_2,\ldots,s_N\}$
- Arcs: Transitions with probabilities, $P(X_i|X_{i-1})$, X_i generates (s_k,s_l)



$p(\text{toe}) = 0.6 \times 0.88 \times 0.07 = 0.037$

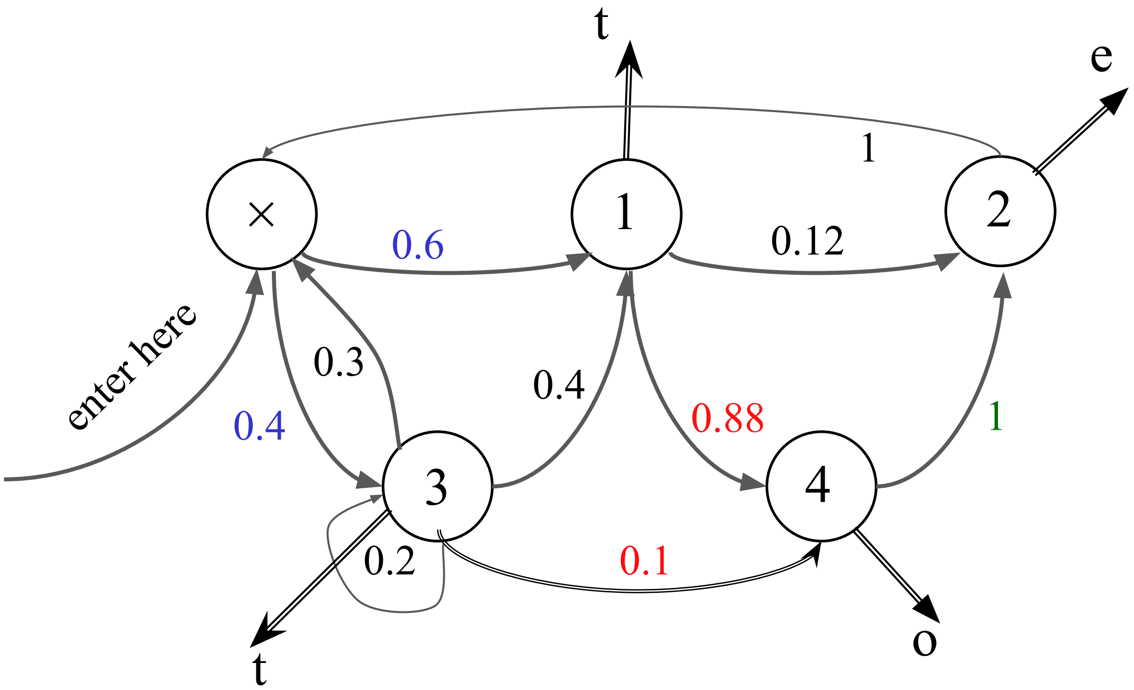
$p(\text{one}) = ?$

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:
  - possible transformation: arcs ↔ nodes
- Possible thanks to the “limited history” Markov Property
- So far: ***Visible*** Markov Models (VMM)

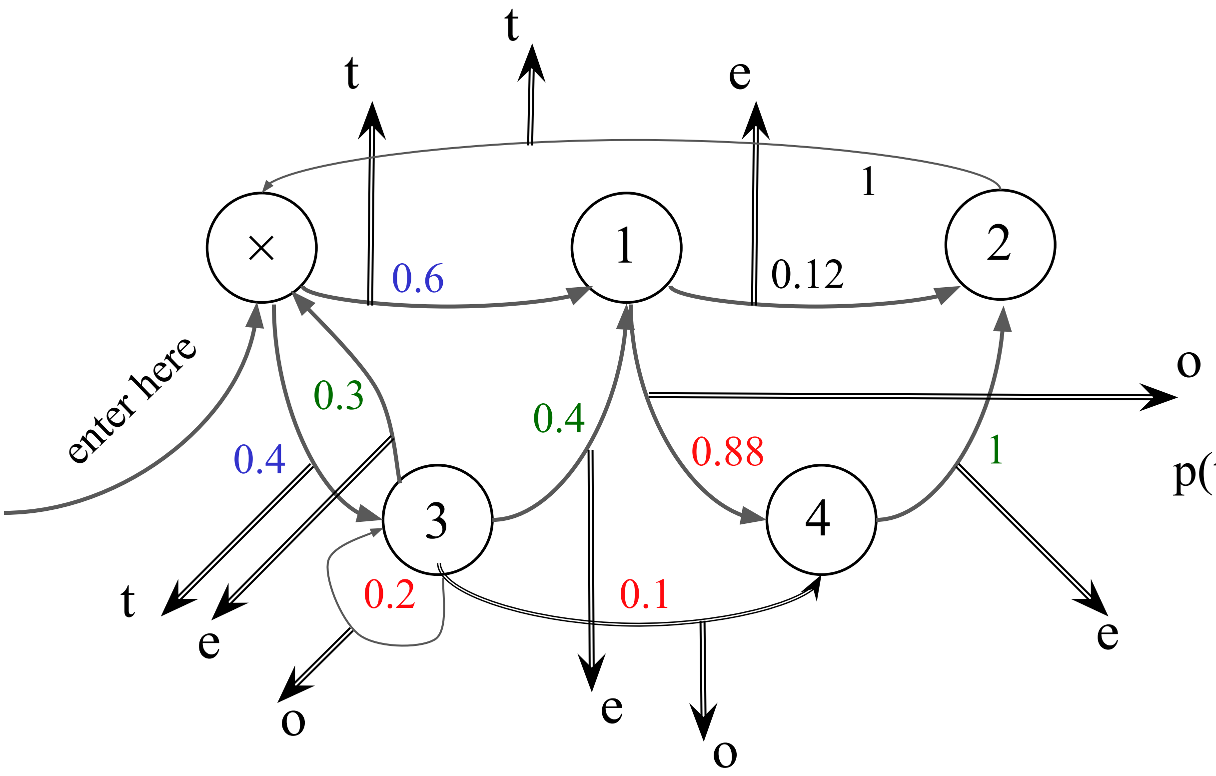
Hidden Markov Models

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



$$p(\text{toe}) = 0.6\times0.88\times1 + 0.4\times0.1\times1 \cong 0.568$$

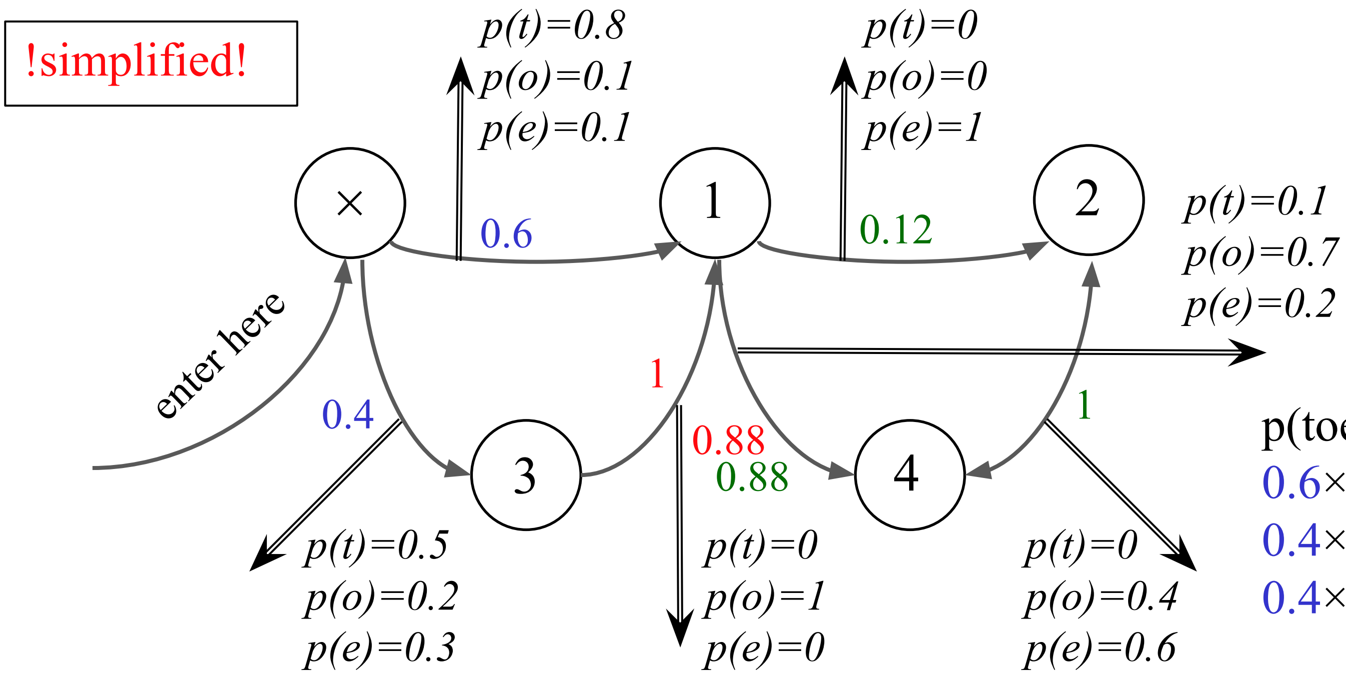
- Added flexibility: Generate output from arcs, not states:



$$
\begin{aligned}
p(toe) = {} & 0.6 \times 0.88 \times 1 \; + \\
& 0.4 \times 0.1 \times 1 \; + \\
& 0.4 \times 0.2 \times 0.3 \; + \\
& 0.4 \times 0.2 \times 0.4 \cong 0.624
\end{aligned}
$$

… and Finally, Add Output Probabilities

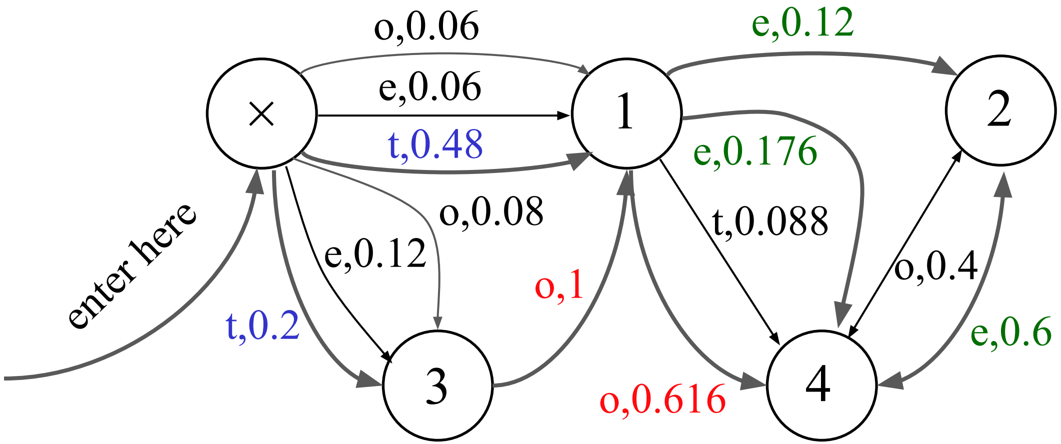
- Maximum flexibility: [Unigram] distribution (sample space: output alphabet) at each output arc:



$$
\begin{aligned}
p(\text{toe}) = {} & 0.6\times0.8\times0.88\times0.7\times1\times0.6+\\
& 0.4\times0.5\times1\times1\times0.88\times0.2 +\\
& 0.4\times0.5\times1\times1\times0.12\times1 \cong 0.237
\end{aligned}
$$

Slightly Different View

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



$$p(toe) = 0.48 \times 0.616 \times 0.6 + 0.2 \times 1 \times 0.176 + 0.2 \times 1 \times 0.12 \cong 0.237$$

Formalization

- HMM (the general case): five-tuple $\mathbf{(S, s_0, Y, P_S, P_Y)}$, where:
  - $\mathbf{S = \{s_1, s_2, \ldots, s_T\}}$ is the set of states, $\mathbf{s_0}$ is the initial state,
  - $\mathbf{Y = \{y_1, y_2, \ldots, y_V\}}$ is the output alphabet,
  - $\mathbf{P_S(s_j | s_i)}$ is the set of prob. distributions of transitions,
    - size of P_S: $|S|^2$
  - $\mathbf{P_Y(y_k | s_i, s_j)}$ 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 (for graph on slide 36):
  - $S = \{x, 1, 2, 3, 4\}$, $s_0 = x$
  - $Y = \{ t, o, e \}$
  - P_S:

	×	**1**	**2**	**3**	**4**
×	0	0.6	0	0.4	0
1	0	0	0.12	0	0.88
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	1	0	0

$\Sigma = 1$

P_Y:

t	×	**1**	**2**	**3**	**4**
×		0.8		0.5	
1					0.1
2					0
3		0			
4			0		

o	×	**1**	**2**	**3**	**4**
×					
1					0.7

e	×	**1**	**2**	**3**	**4**
×					
1					0.2

$\Sigma = 1$

HMM: The Two Tasks

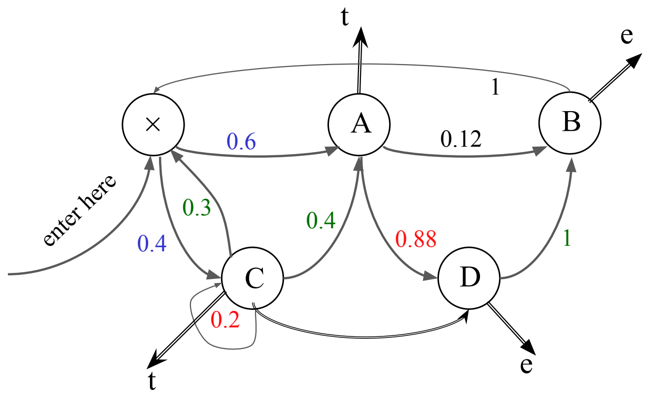
- HMM (the general case): five-tuple $(\mathbf{S}, \mathbf{S_0}, \mathbf{Y}, \mathbf{P_S}, \mathbf{P_Y})$, where:
  - $\mathbf{S} = \{\mathbf{s_1,s_2,\ldots,s_T}\}$ is the set of states, $\mathbf{S_0}$ is the initial state,
  - $\mathbf{Y} = \{\mathbf{y_1,y_2,\ldots,y_V}\}$ is the output alphabet,
 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

Trellis: HMM “roll-out”

HMM:

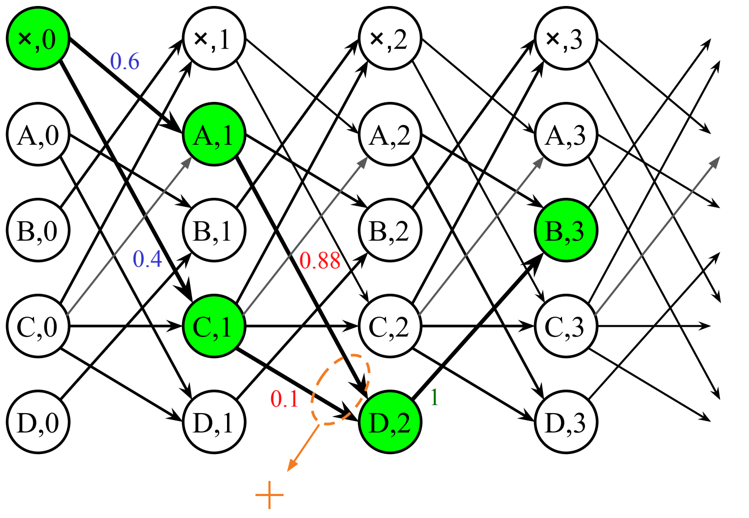


p(toe) = 0.6×0.88×1 + 0.4×1×1 ≅ 0.568

Trellis:

“roll-out”

time/position: 0 1 2 3 ....



Y: t o e

$\alpha(\times,0) = 1$ $\alpha(A,1) = 0.6$ $\alpha(D,2) = 0.568$ $\alpha(B,3) = 0.568$

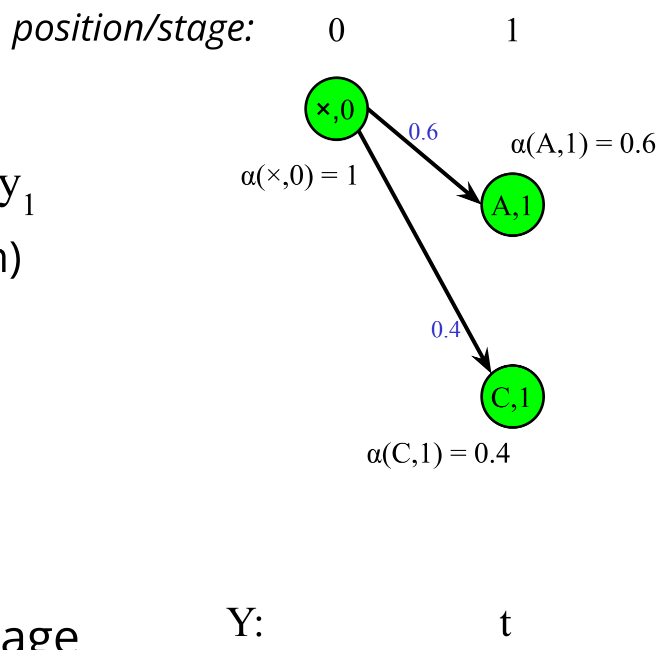
$\alpha(C,1) = 0.4$

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

Creating the Trellis: The Start

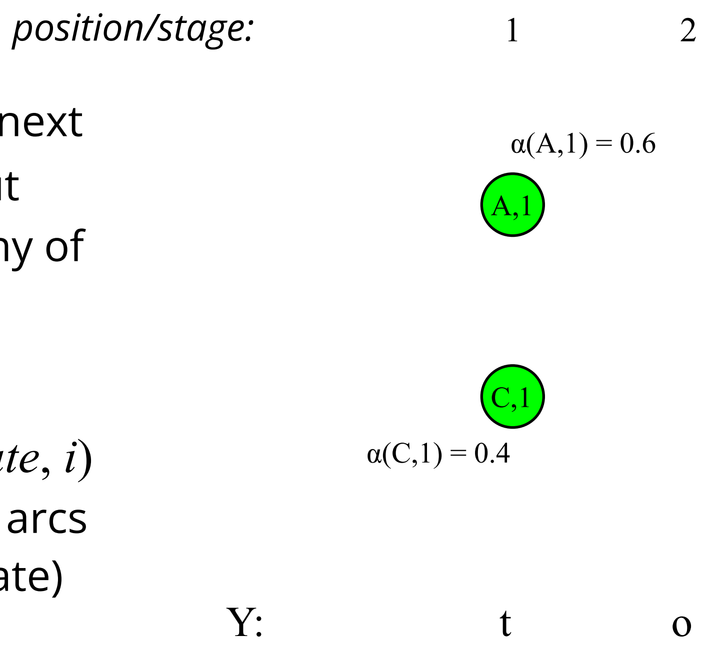
- Start in the start state (×),
  - set its $\alpha(\times,0)$ to 1
- Create the first stage:
  - get the first "output" symbol y_1
  - create the first stage (column)
  - but only those Trellis states

    which generate y_1
  - set their $\alpha(state,1)$ to the $P_S(state|\times)\ \alpha(\times,0)$
- and forget about the *0*-th stage



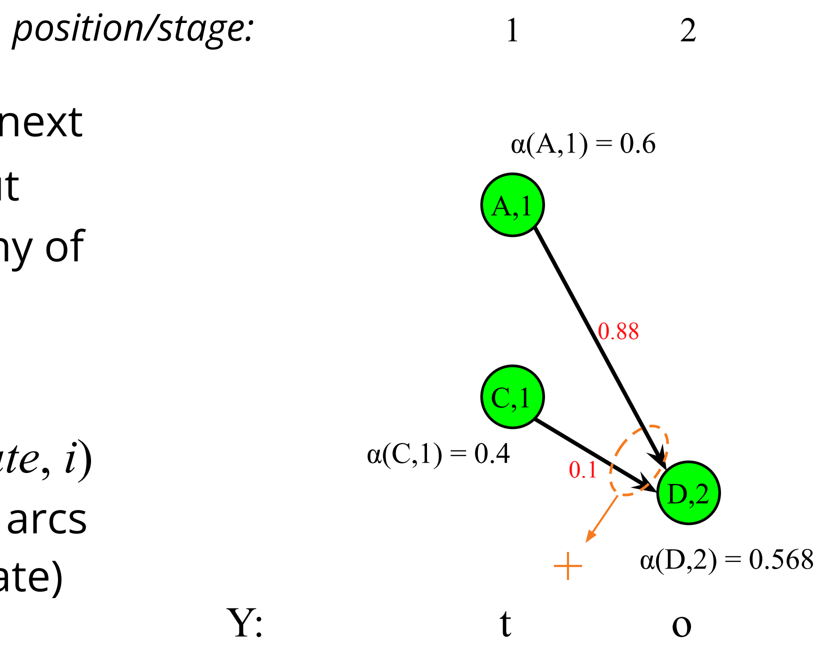
Trellis: The Next Step

- 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:
    $P_S(state|prev.state) \times \alpha(prev.state, i)$
    (add up all such numbers on arcs going to a common Trellis state)
    ...and forget about stage i



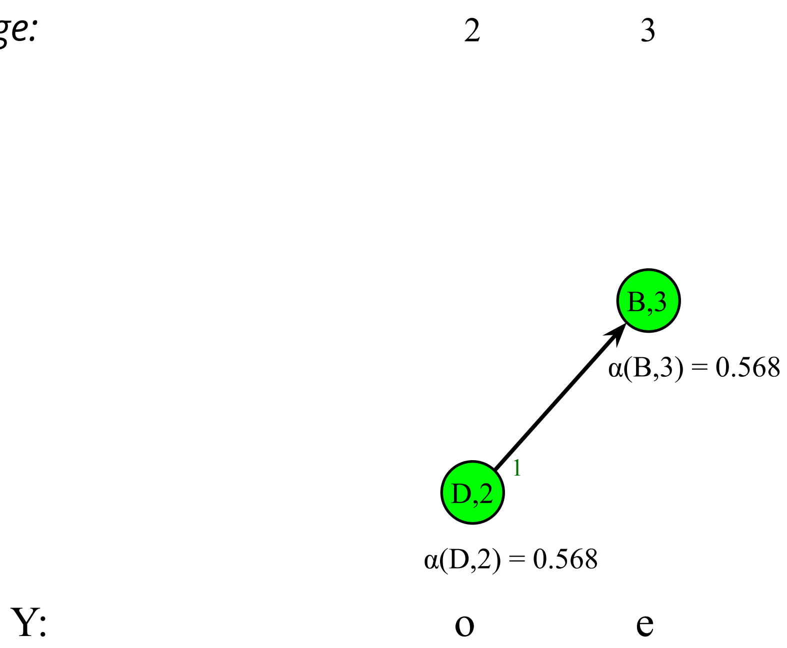
Trellis: The Next Step

- 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: $P_S(state|prev.state) \times \alpha(prev.state, i)$ (add up all such numbers on arcs going to a common Trellis state) ...and forget about stage i



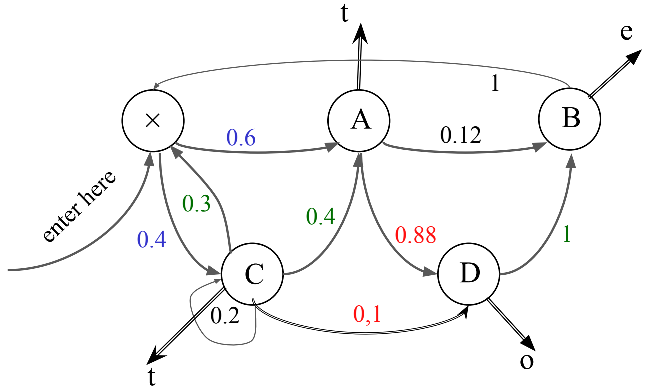
Trellis: The Last Step

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



Trellis (again)

HMM:

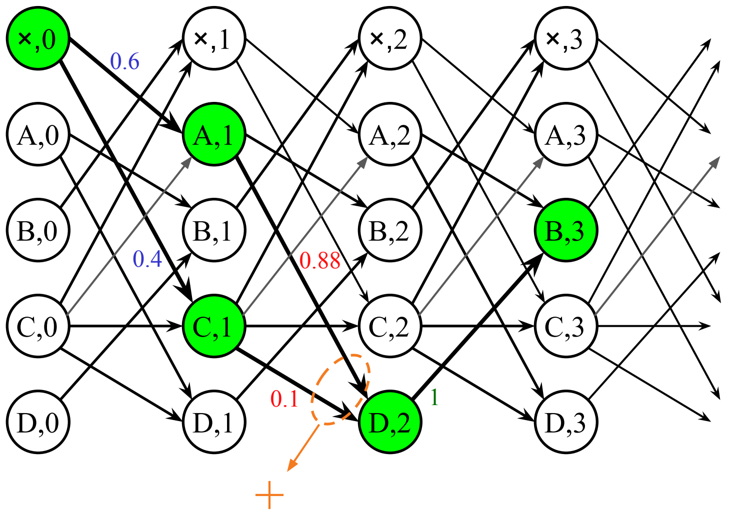


$p(toe) = 0.6 \times 0.88 \times 1 + 0.4 \times 1 \times 1 \cong 0.568$

“roll-out”

Trellis:

time/position: 0 1 2 3 ....



Y: t o e

$\alpha(\times,0) = 1$ $\alpha(A,1) = 0.6$ $\alpha(D,2) = 0.568$ $\alpha(B,3) = 0.568$

$\alpha(C,1) = 0.4$

- Trellis state: (HMM state, position)
- each state: holds ***one*** number (prob): $\boldsymbol{\alpha}$
- probability of Y: $\Sigma\alpha$ in the last state

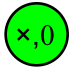
HMM: The Two Tasks

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

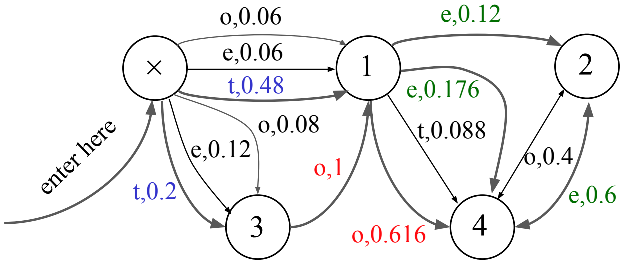
- Given an HMM & an output sequence $\mathrm{Y} = \{\mathrm{y_1,y_2,\ldots,y_k}\}$:
  - (Task 1) compute the probability of Y;
  - (Task 2) compute the most likely sequence of states which has generated Y.

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

position/stage: 0



$\alpha(\times,0) = 1$



$$p(toe) = 0.48 \times 0.616 \times 0.6 + 0.2 \times 1 \times 0.176 + 0.2 \times 1 \times 0.12 \cong 0.237$$

Y:

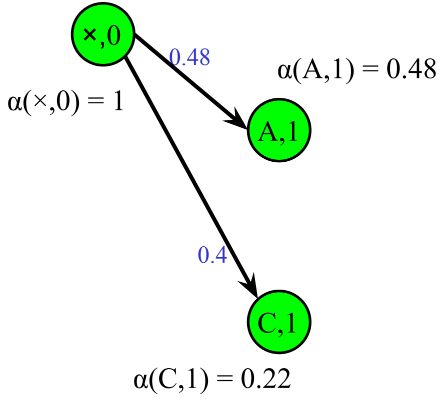
General Trellis: The Next Step

We are in stage i :

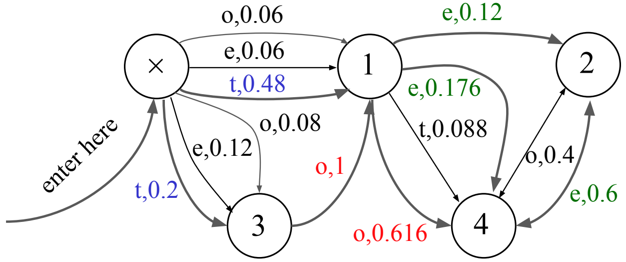
- Generate the next stage i+1 as before (except now <u>arcs</u> generate output, thus use only those arcs marked by the output symbol y_{i+1})

- For each generated *state*, compute $\alpha(state,i+1)$ $= \Sigma_{\text{inc. arcs}} P_Y(y_{i+1}|state, prev.state) \times \alpha(prev.state, i)$

position/stage: 0 1

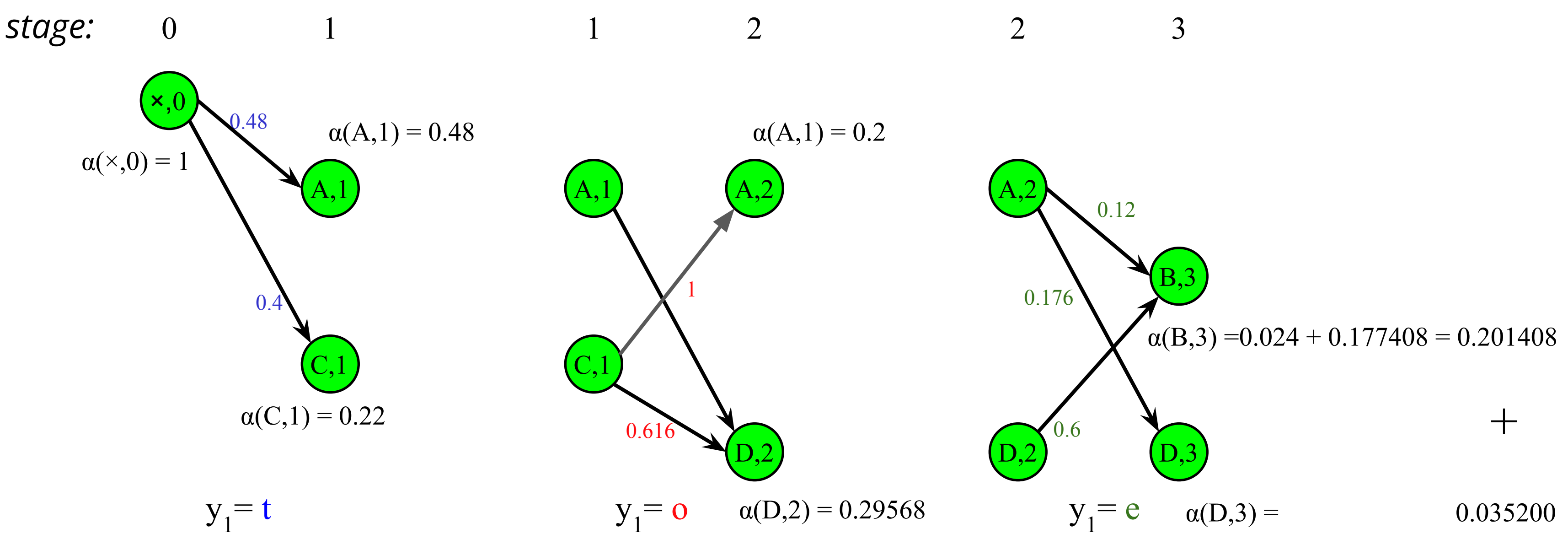


Y: t

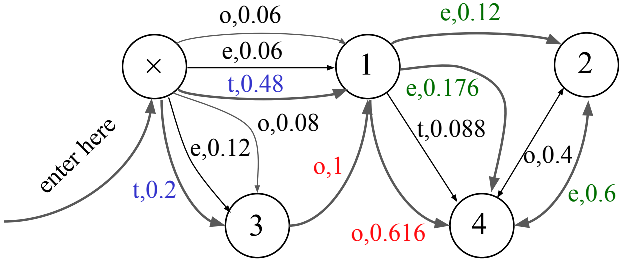


And forget about the previous state ...

Trellis: The Complete Example



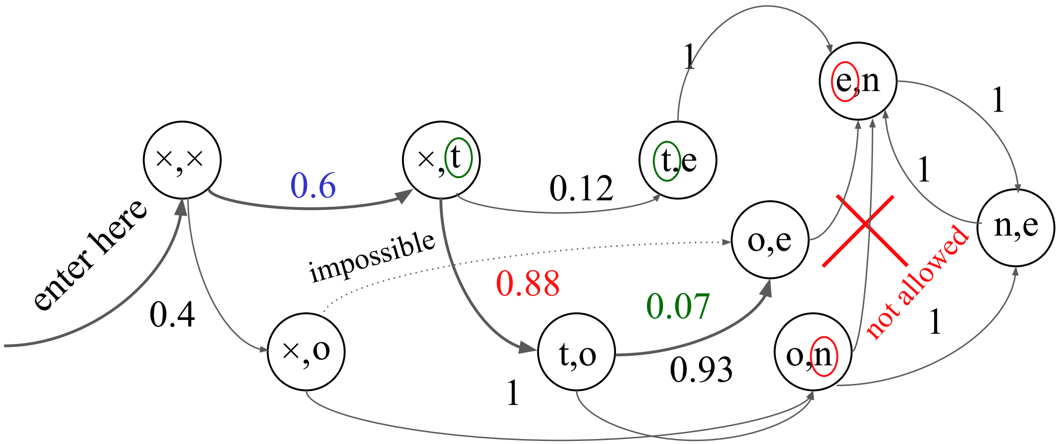
P(Y) = P(toe) = 0.236608

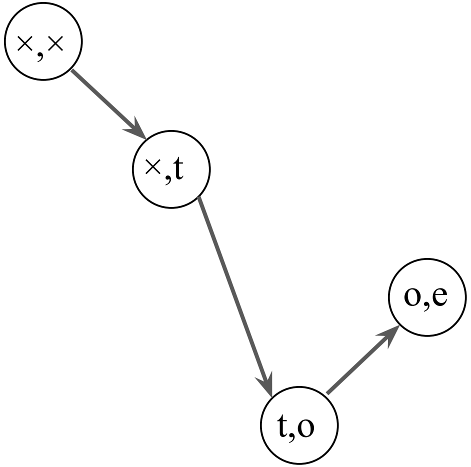


p(toe) = 0.48×0.616×0.6+
0.2×1×0.176 +
0.2×1×0.12 ≅ 0.237

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:





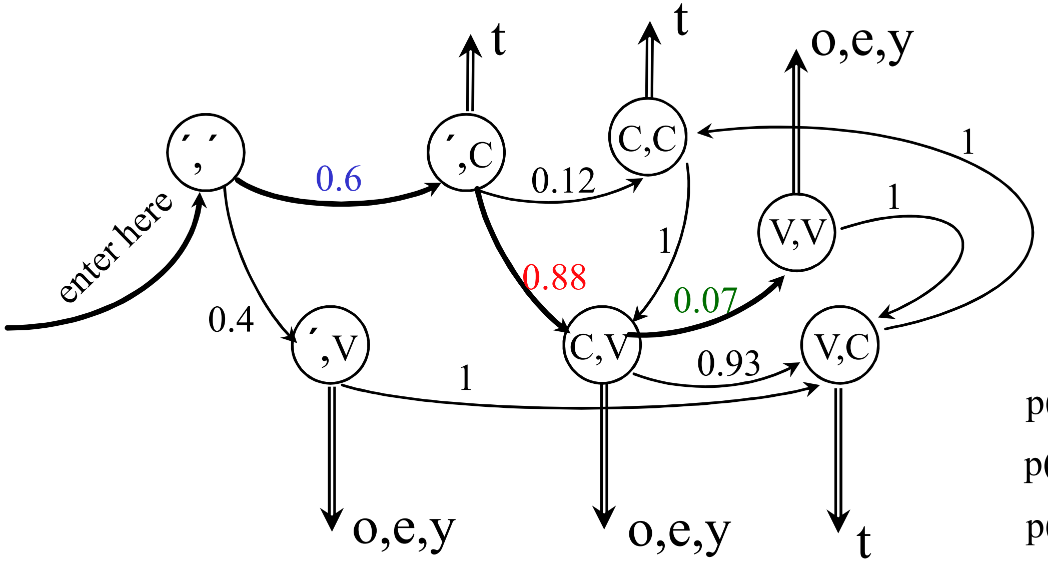
- Multiple paths not possible → trellis not really needed

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)



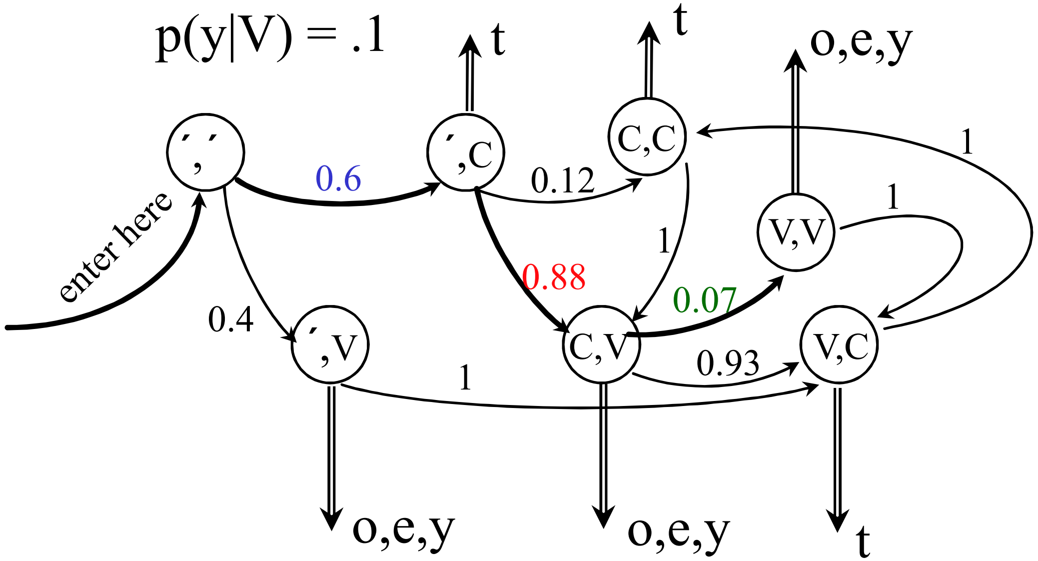
p(t|C) = 1 usual,
p(o|V) = .3 non-
p(e|V) = .6 overlapping
p(y|V) = .1 classes

p(toe) = .6 ´ 1 ´ .88 ´ .3 ´ .07 ´ .6 ≅ .00665
p(teo) = .6 ´ 1 ´ .88 ´ .6 ´ .07 ´ .3 ≅ .00665
p(toy) = .6 ´ 1 ´ .88 ´ .3 ´ .07 ´ .1 ≅ .00111
p(tty) = .6 ´ 1 ´ .12 ´ 1 ´ 1 ´ .1 ≅ .0072

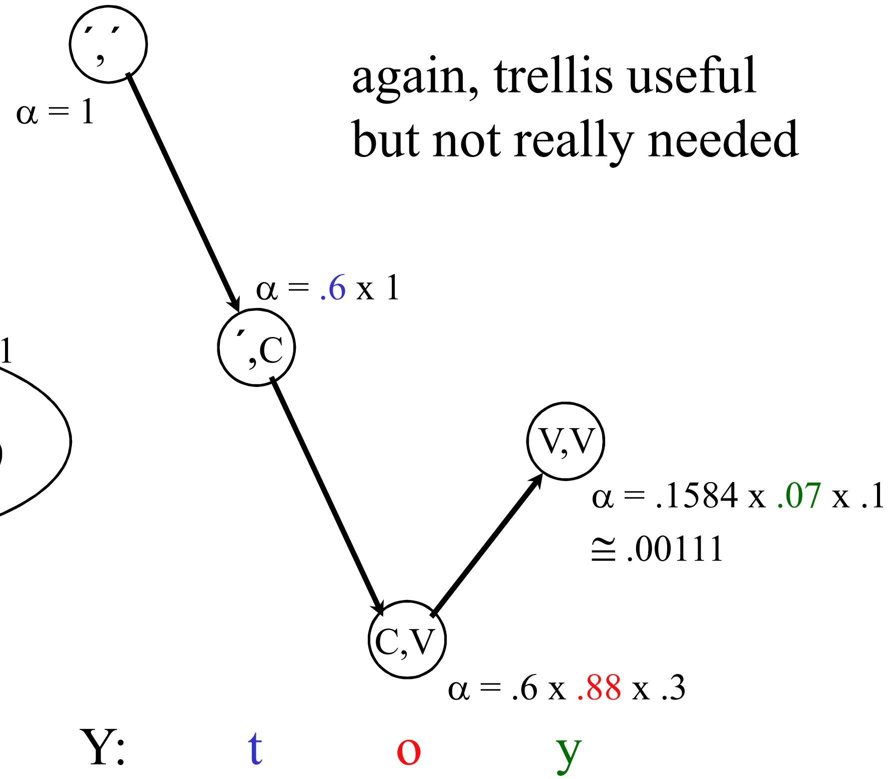
Class Trigrams: the Trellis

- Trellis generation (Y = “toy”):

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

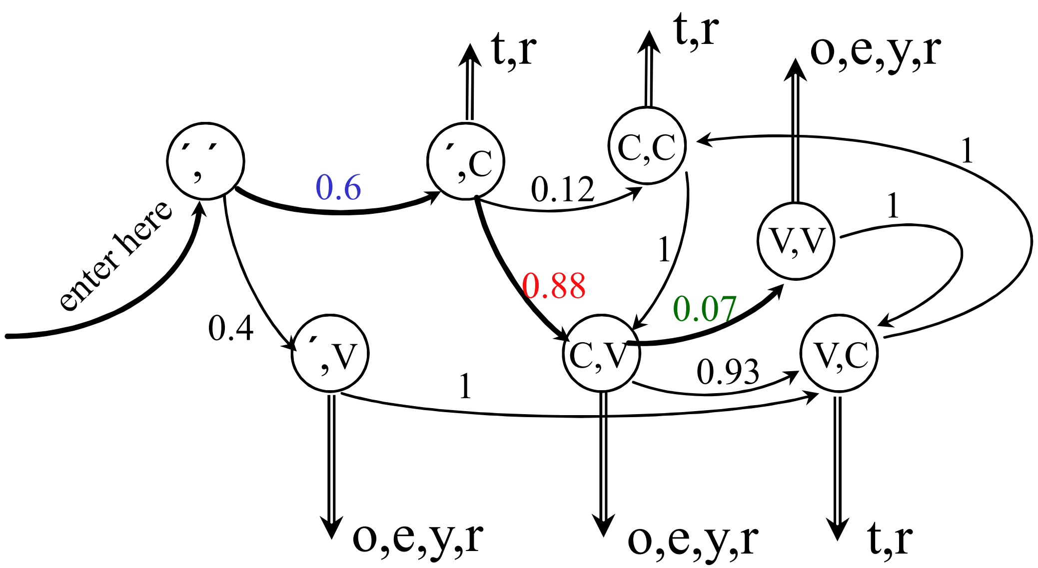


again, trellis useful
but not really needed



Overlapping Classes

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

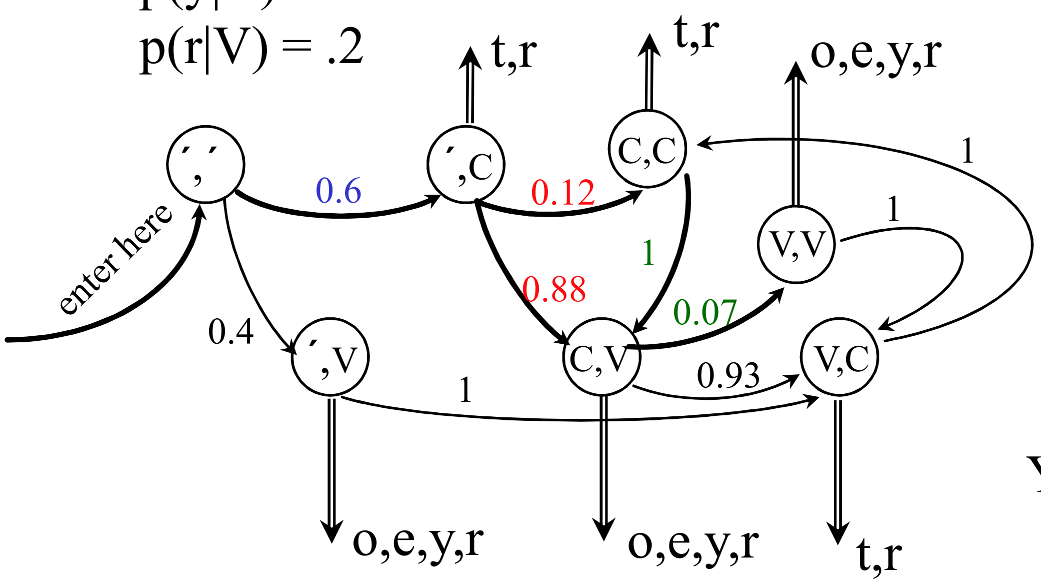


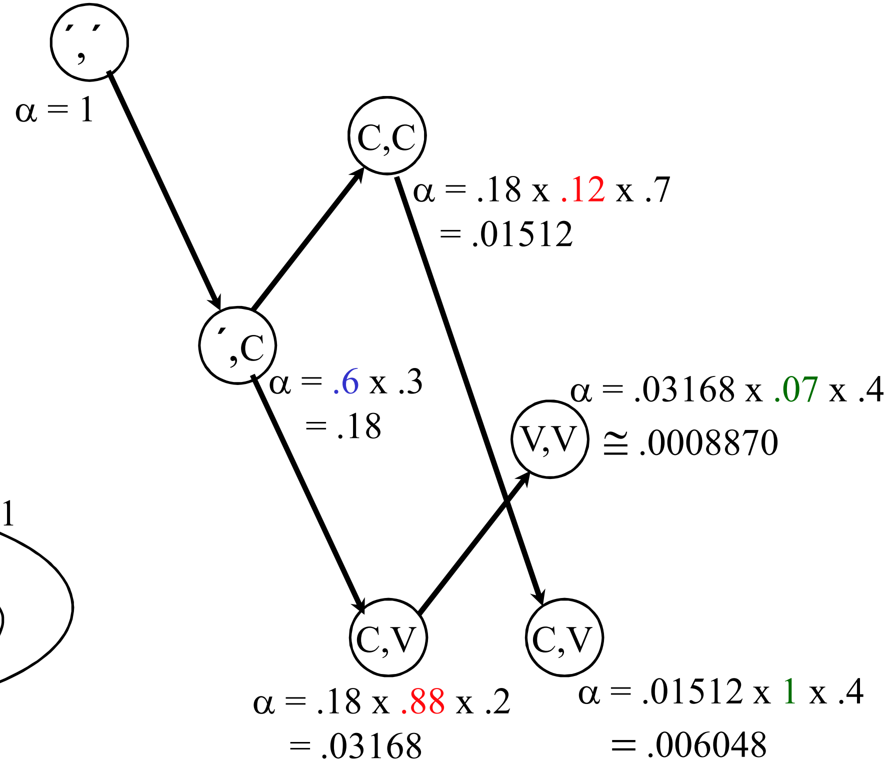
$p(t|C) = .3$
$p(r|C) = .7$
$p(o|V) = .1$
$p(e|V) = .3$
$p(y|V) = .4$
$p(r|V) = .2$

$p(try) = ?$

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 $p(Y) = \underline{.006935}$

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 <u>and</u> 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):

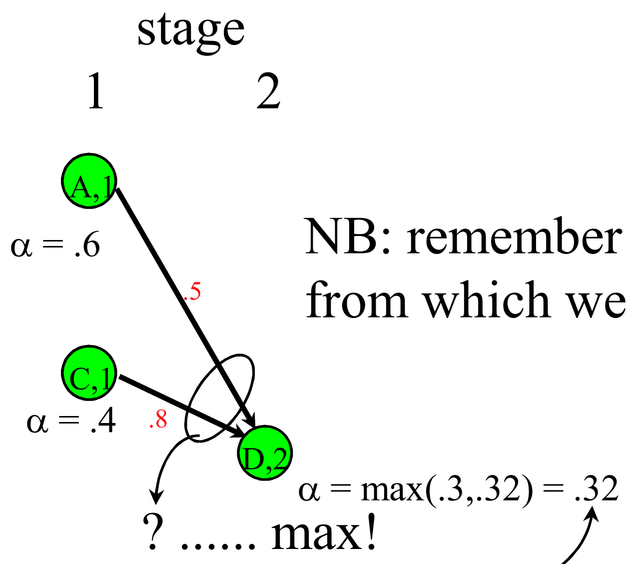
$$S_{best} = \text{argmax}_S P(S,Y) =$$

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

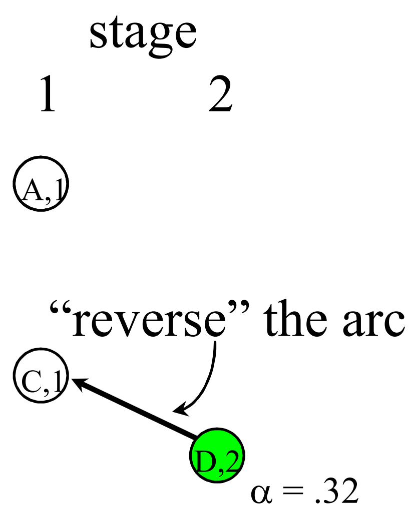
$$= \text{argmax}_S \Pi_{i=1..k}\, p(y_i|s_i,s_{i-1})p(s_i|s_{i-1})$$

The Crucial Observation

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



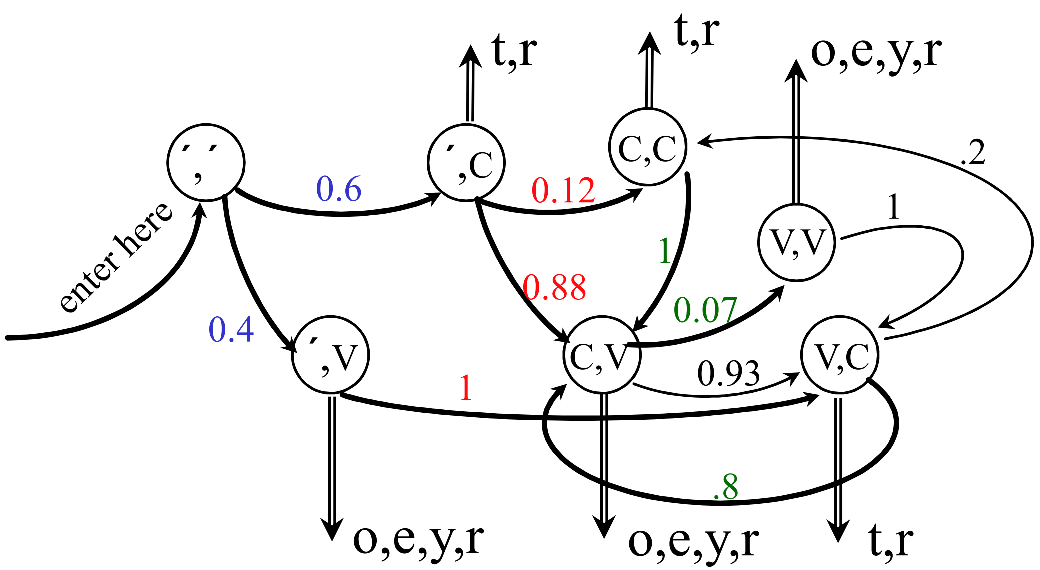
NB: remember previous state from which we got the maximum:



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?):



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

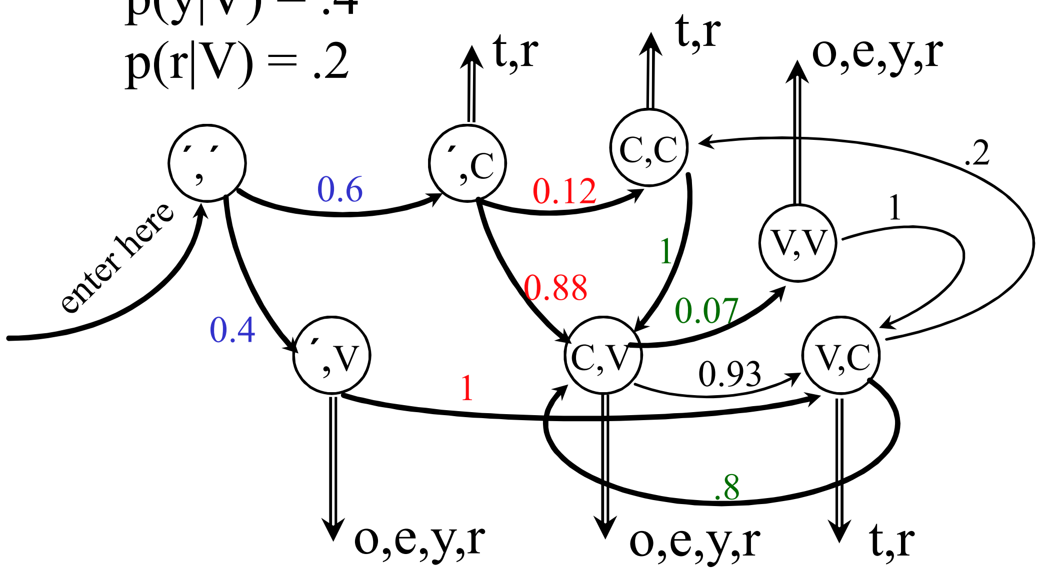
$\text{argmax}_{XYZ}\ p(rry|XYZ) = ?$

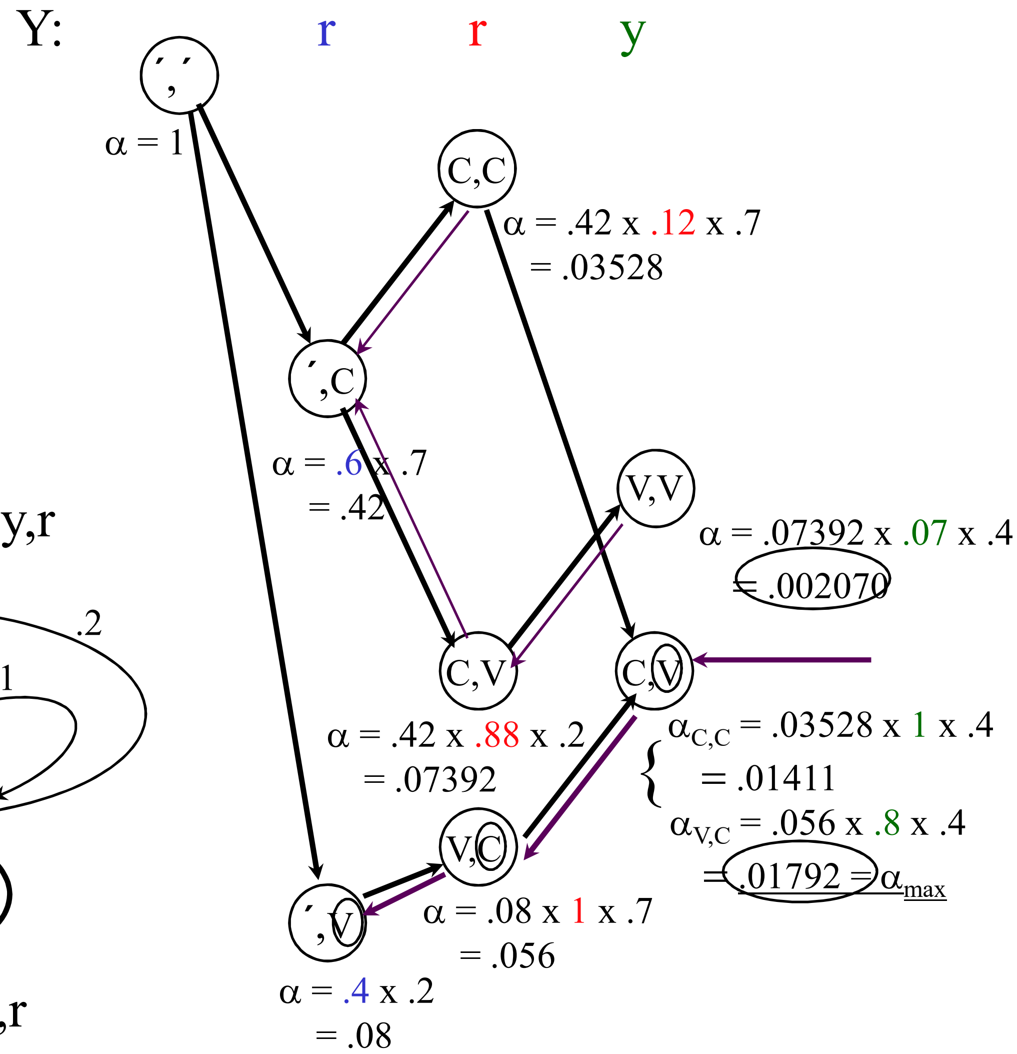
Possible state seq.: (´,v)(v,c)(c,v)[VCV], (´,c)(c,c)(c,v)[CCV], (´,c)(c,v)(v,v) [CVV]

Viterbi Computation

α in trellis state: best prob from start to here

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





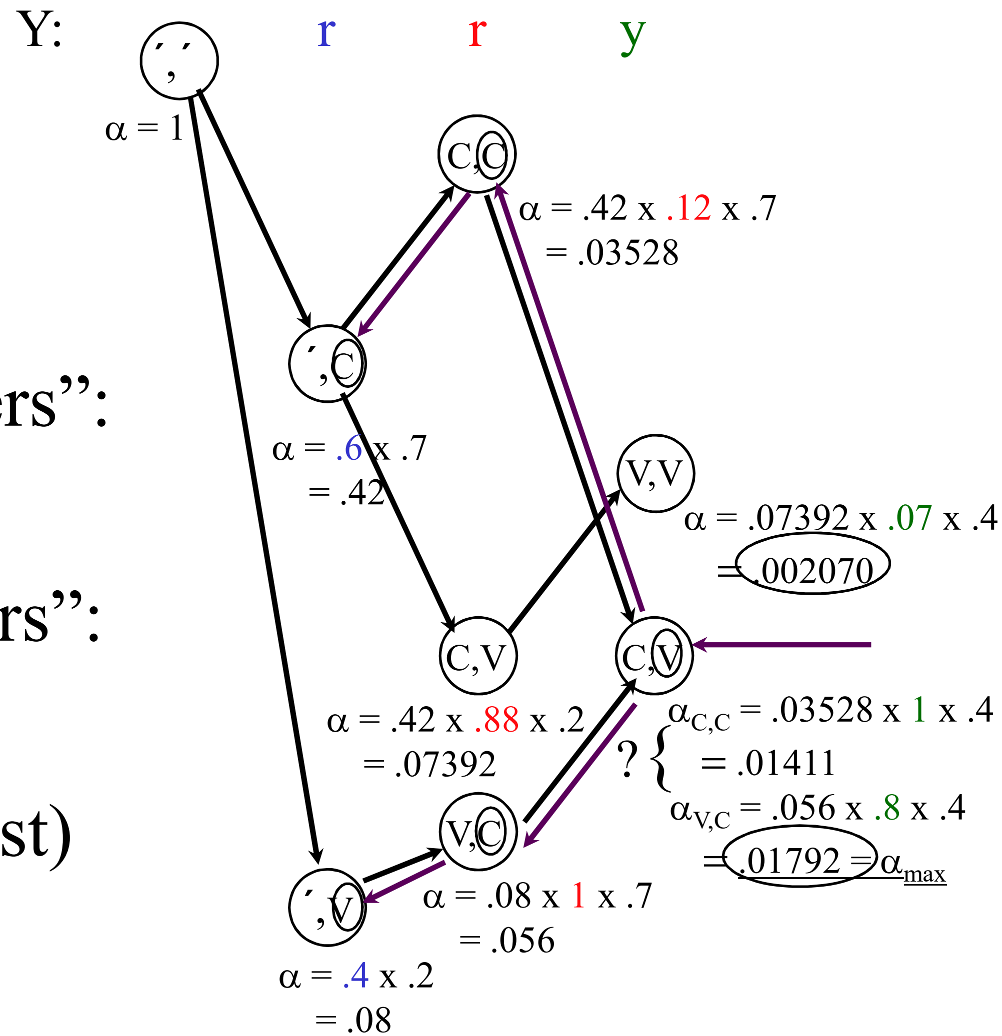
n-best State Sequences

- Keep track of n best "back pointers":
- Ex.: n= 2:

  Two "winners":

  VCV (best)

  CCV (2^{nd} best)

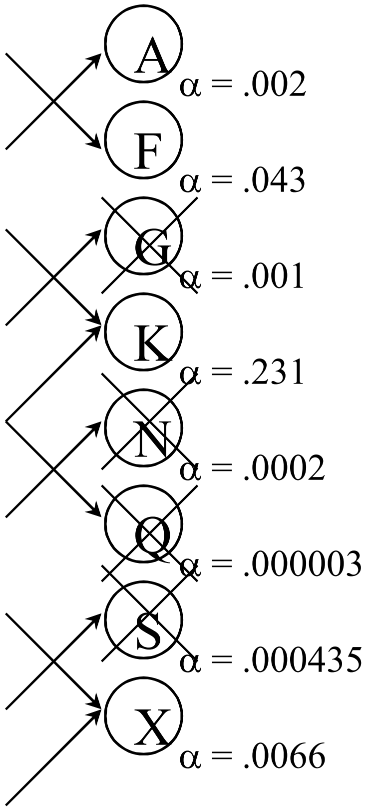


Tracking Back the n-best paths

- Backtracking-style algorithm:
  - **Start at the end, in the best of the n states (s_{best})**
  - **Put the other n-1 best nodes/back pointer pairs on stack, except those leading from s_{best} to the same best-back state.**
- Follow the back "beam" towards the start of the data, spitting out nodes on the way (backwards of course) using always only the <u>best</u> back pointer.
- At every beam split, push the diverging node/back pointer pairs onto the stack (node/beam width is sufficient!).
- When you reach the start of data, close the path, and pop the top-most node/back pointer(width) pair from the stack.
- Repeat until the stack is empty; expand the result tree if necessary.

Pruning

- Sometimes, too many trellis states in a stage:



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

HMM Parameter Estimation: the Baum-Welch Algorithm

HMM: The 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.

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

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)**

Initialization

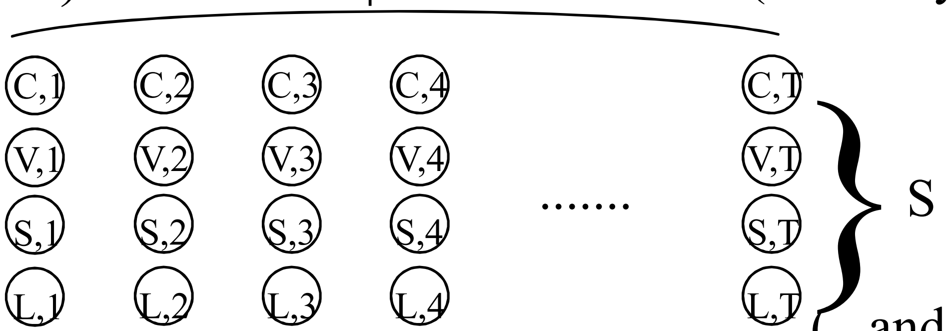
- For computing the initial expected "counts"
- Important part
  - EM guaranteed to find a *local* maximum only (albeit a good one in most cases)
- P_Y initialization more important
  - fortunately, often easy to determine
    - **together with dictionary ↔ vocabulary mapping, get counts, then MLE**
- 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 ($p(s'|s)$, $p(y|s,s')$):**

     $$\alpha(s',i) = \sum_{s \rightarrow s'} \alpha(s,i-1) \times p(s'|s) \times p(y_i|s,s')$$

   - **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**

     $$\beta(s',i) = \sum_{s \leftarrow s'} \beta(s,i+1) \times p(s|s') \times p(y_{i+1}|s',s)$$

   - **also, keep the $\beta(s,i)$ at all trellis states**

The Algorithm Part II

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

$$c(y,s,s') = \Sigma_{i=0..k-1,y=y_{i+1}} \alpha(s,i)\ \underbrace{p(s'|s)\ p(y_{i+1}|s,s')}_{\text{this transition prob ´ output prob}}\ \beta(s',i+1)$$

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

prefix prob.

tail prob

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

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

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):**

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

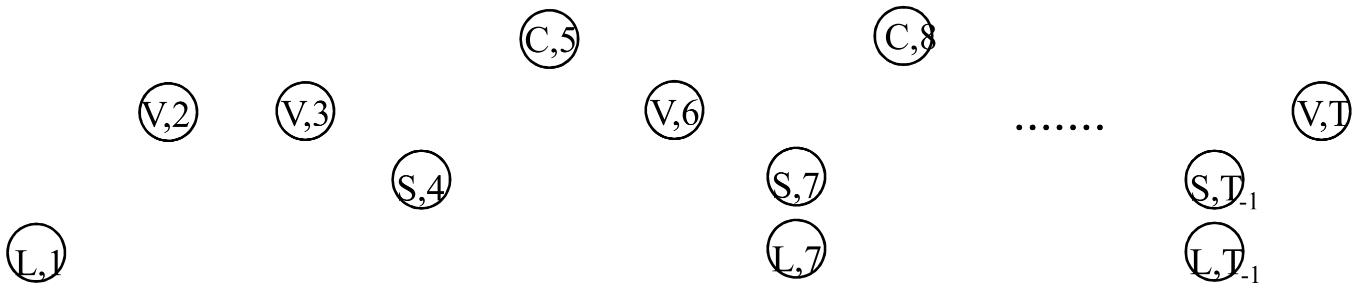
- **use the same N(i) for βs at the end of each backward (Step 3) stage:**

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

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)

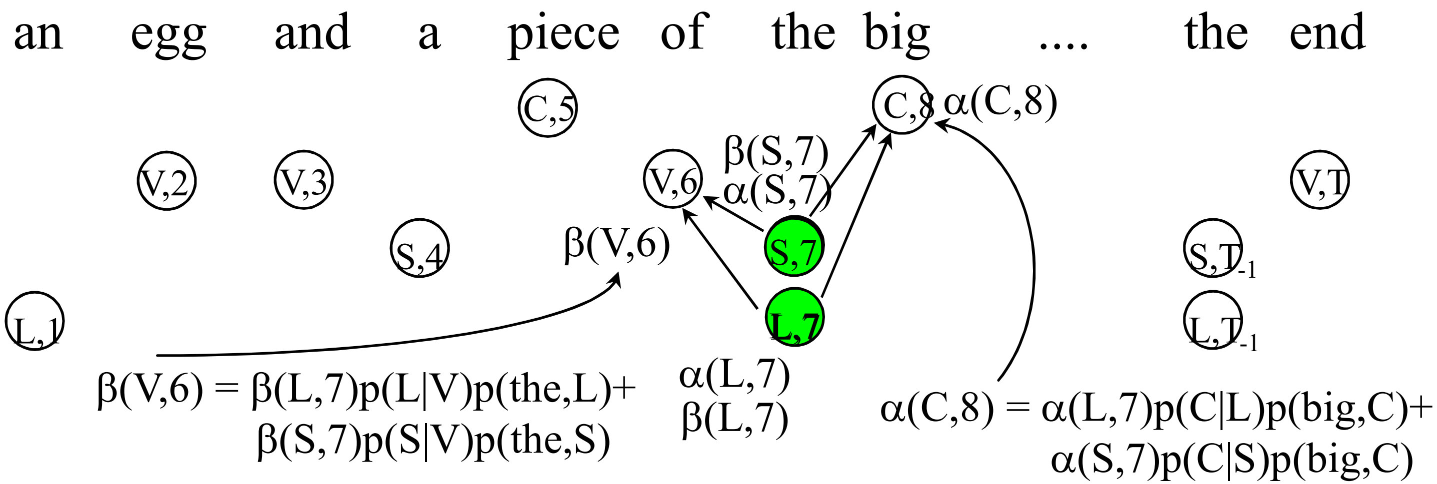
Fill in alpha, beta

- Left to right, alpha:

$$\boldsymbol{\alpha(s',i) = \Sigma_{s \to s'} \alpha(s,i\text{-}1) \times p(s'|s) \times p(w_i|s')}$$

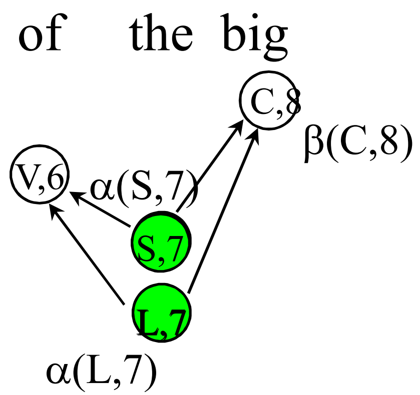
output from states

- 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):
  - **$inc(y_{i+1}, s_i, s_{i+1}) = a(s_i, i)\ p(s_{i+1}|s_i)\ p(y_{i+1}|s_{i+1})\ b(s_{i+1}, i+1)$**
  - **$c(y, s_i, s_{i+1})$ += inc (for y at pos i+1)**
  - **$c(s_i, s_{i+1})$ += inc (always)**
  - **$c(s_i)$ += inc (always)**



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

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

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

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”**