NPFL147

Statistical Methods in Natural Language Processing

8. Hidden Markov Models

Pavel Pecina, Jindřich Helcl

9 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

Embedding Matrix

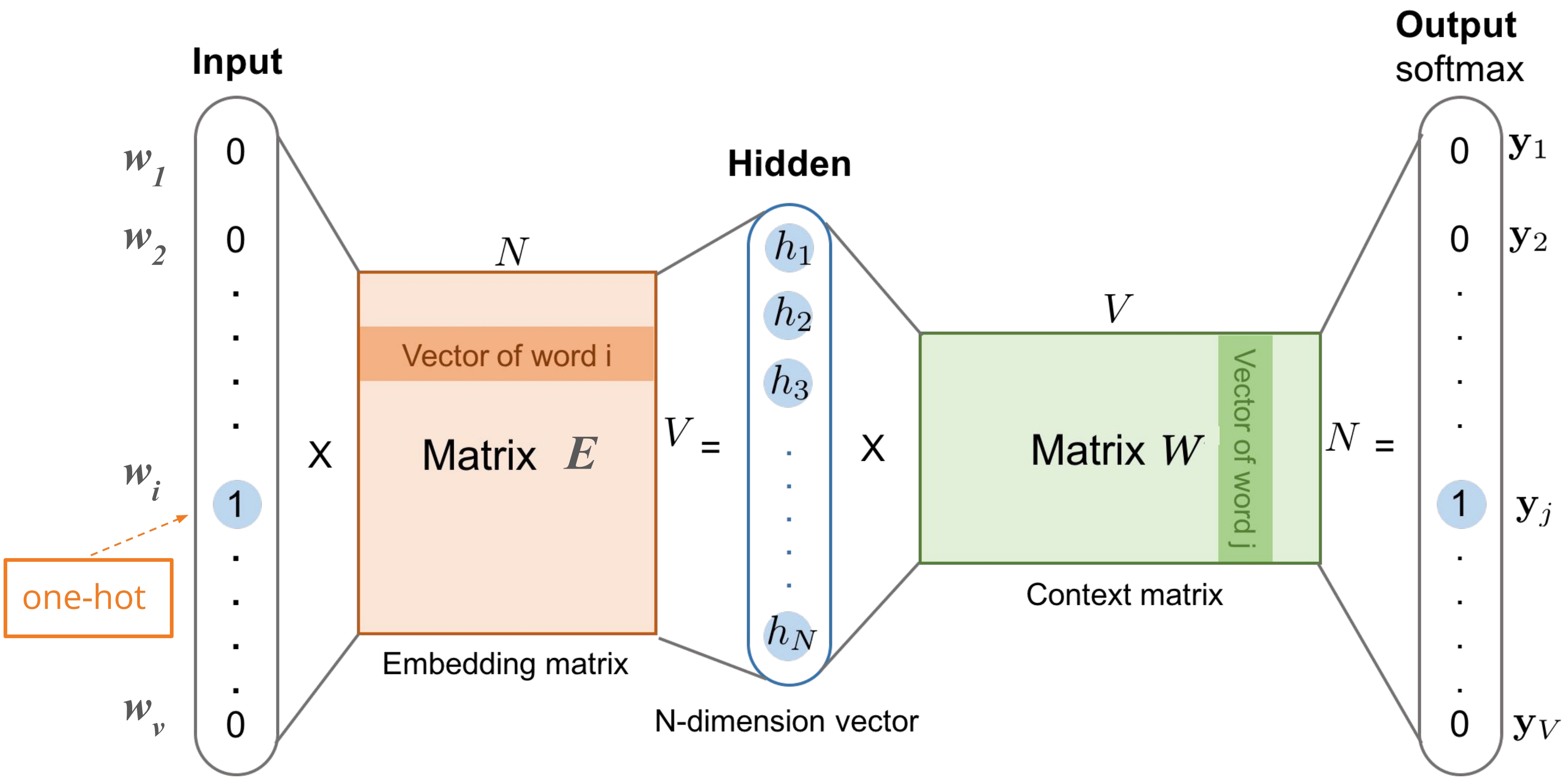
With pre-defined vocabulary V, let $E \in \mathbb{R}^{|V| \times d}$ be an **embedding matrix**

- Each row of E corresponds to a word from the vocabulary
- d is the **embedding size** (or dimension)
  - a hyper-parameter and needs to be decided before obtaining the embeddings

hello	-1.2	0.3	5.0	...
the	0.1	3.6	2.2	...
of	3.3	-1.0	3.1	...
dog	-1.0	7.5	7.1	...

- OOV words?
  - Usually collapsed into a special `<oov>` token that will get its embedding

Word2vec Model Architecture



Original image credit: https://lilianweng.github.io/posts/2017-10-15-word-embedding/

Word2vec Training

Training is done by sampling input and target word(s) from data.

- **CBOW** (continuous bag-of-words): For a given target word *w*, sum embeddings of the context words and predict *w* (one training example).
- **Skip-gram**: For a given input word *w*, predict words in its context (one training example per context word)

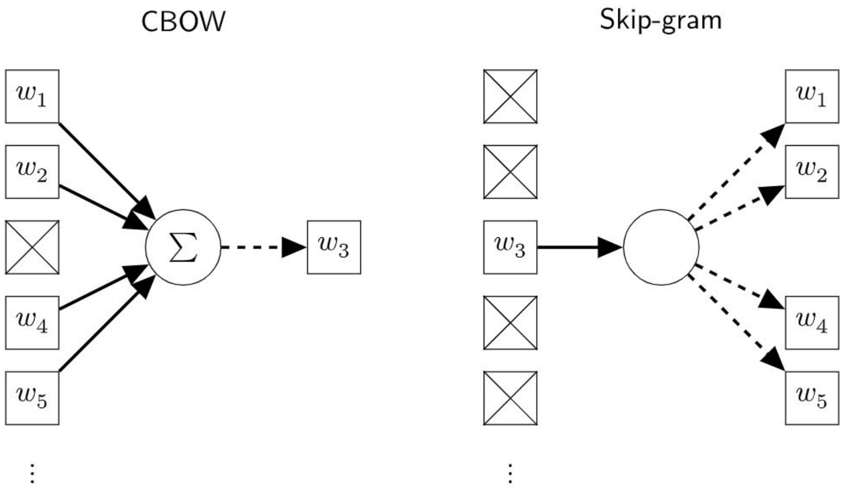


Image credit: Libovický et al., Slides for NPFL129 lecture 6, 2025

Word2vec Implementation, cont.

Using sigmoid instead of softmax gives us:

$$-\log \sigma(e^\top \cdot c)$$

- This is only for word pairs that do occur together (positive samples)
- For better training we need to also train on negative ones
  - a process
- Taking into *do not* occur together, we can complete th

we sample more negative examples than positive, usually around 2-5

loss from the negative example

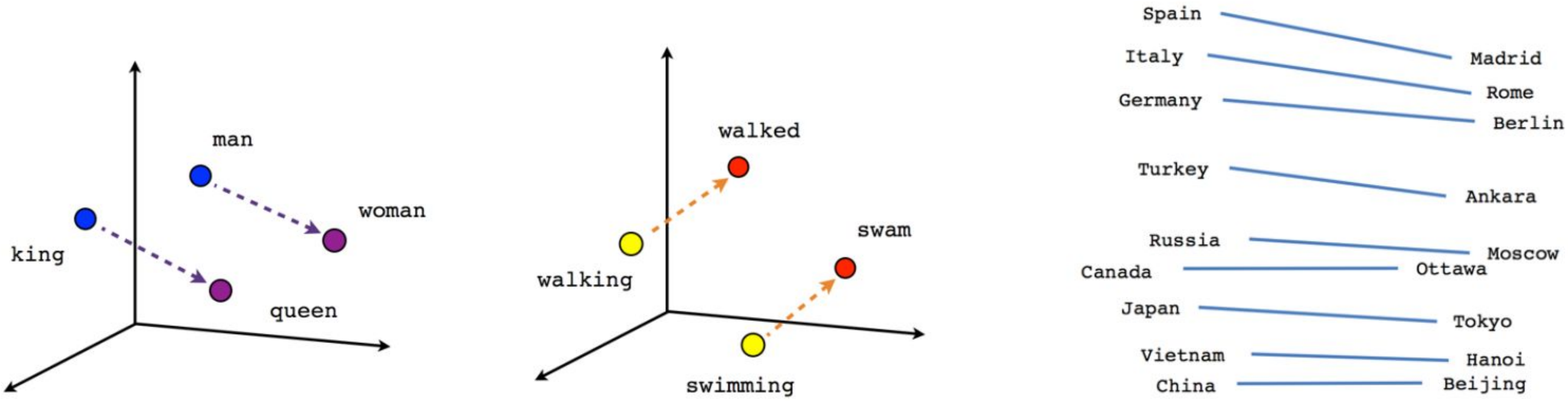
$$L = \boxed{-\log \sigma(e^\top \cdot c_p)} \boxed{-\sum_{c_n}} \boxed{\log(1-\sigma(e^\top \cdot c_n))}$$

loss from the positive example

note that the input word *e* stays the same

Word2vec Vector Properties I

Word2vec vectors seem to enable meaningful arithmetic operations



Male-Female

Verb tense

Country-Capital

Image from https://medium.com/data-science/mapping-word-embeddings-with-word2vec-99a799dc9695

Word2vec Vector Properties II

Word2vec vectors encode distributional semantics: similarity in meaning → proximity in vector space.

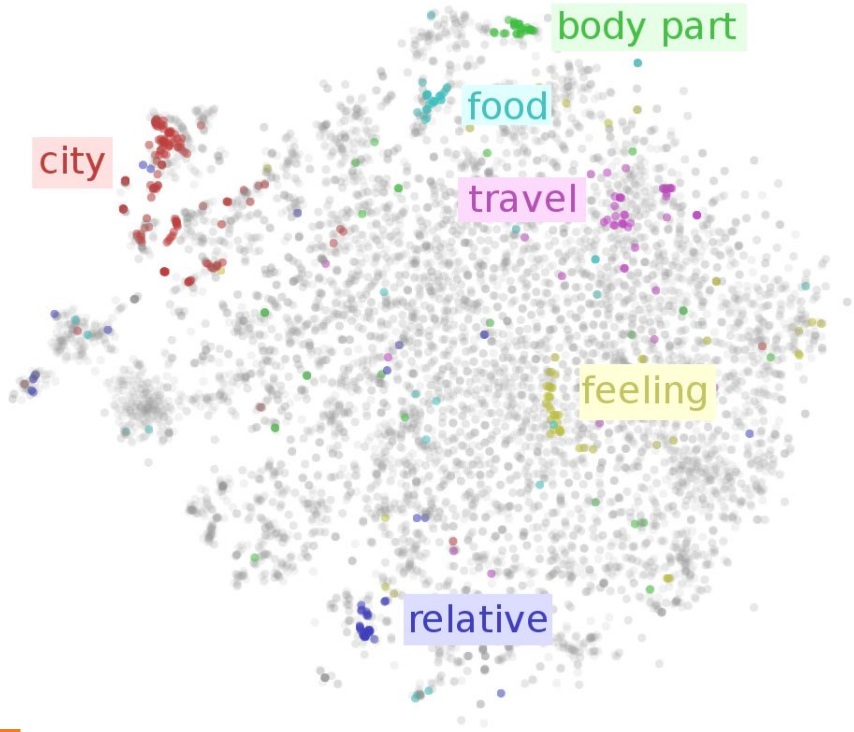


Image from https://www.ruder.io/word-embeddings-1/

Hidden Markov Models

Markov Process (Review)

- Bayes formula (chain rule):

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

Markov Process (Review)

- Bayes formula (chain rule):

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

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

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

Markov Process (Review)

- Bayes formula (chain rule):

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

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

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

Using just one 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 .

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

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: $S = \{s_0, s_1, s_2, \ldots, s_N\}$

Markov Properties

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

1. Limited history (context, horizon):

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

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

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:

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

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

Long History Possible

- What if we want trigrams:

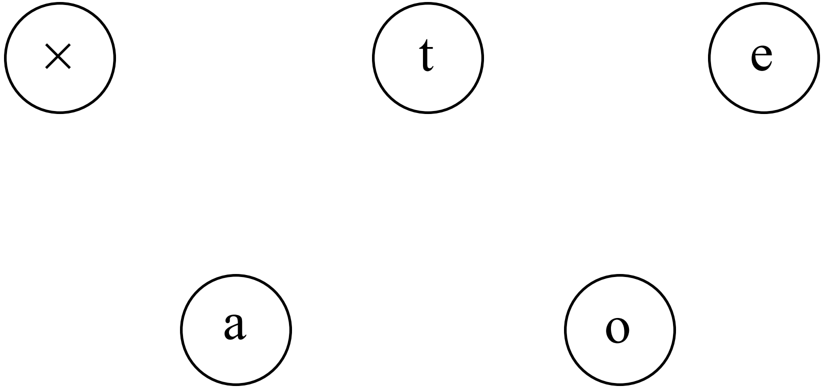
1 7 3 7 9 0 6 7 3 $\boxed{4\ \ 5}$ $\boxed{\ldots}$

- 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	…

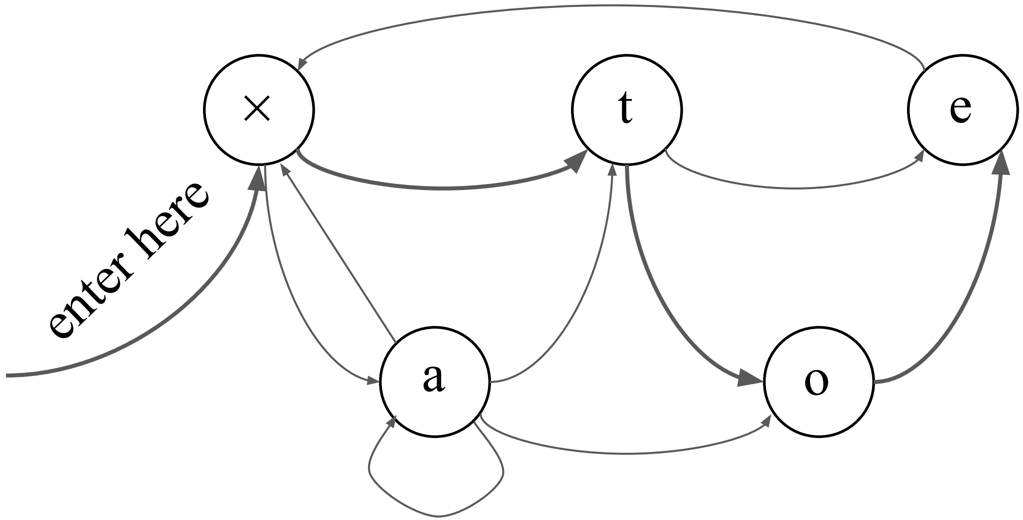
Graph Representation: State Diagram (Bigram case)

- Nodes: States, $S = \{s_0, s_1, s_2, \ldots, s_N\}$



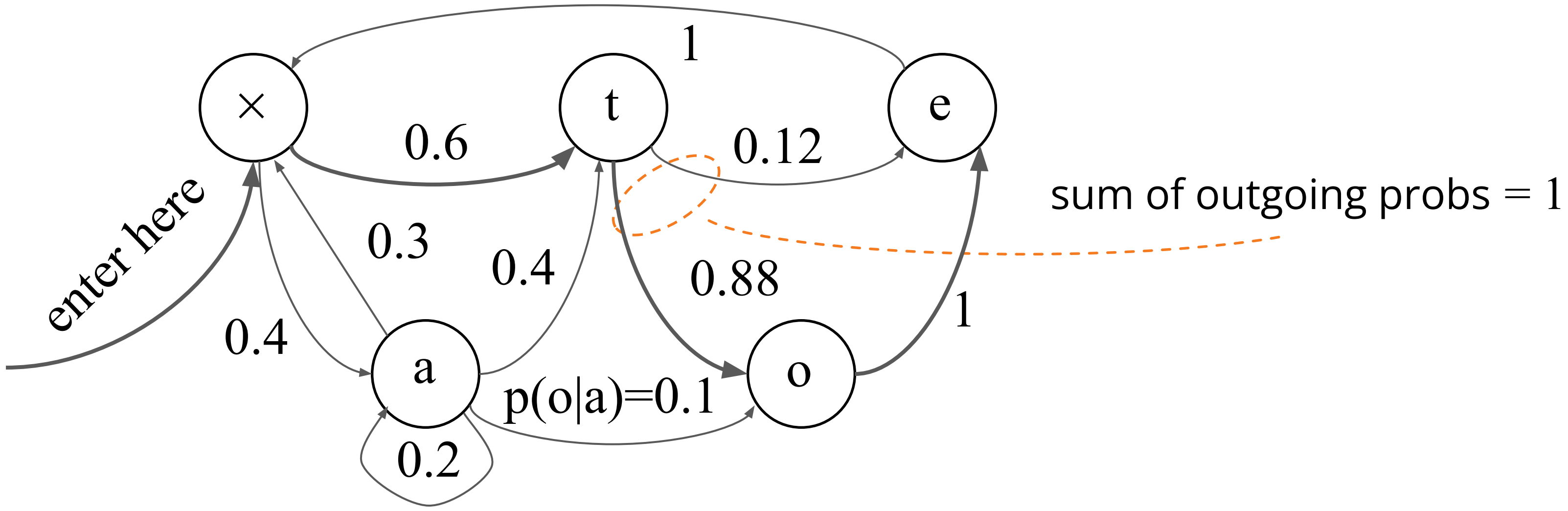
Graph Representation: State Diagram (Bigram case)

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



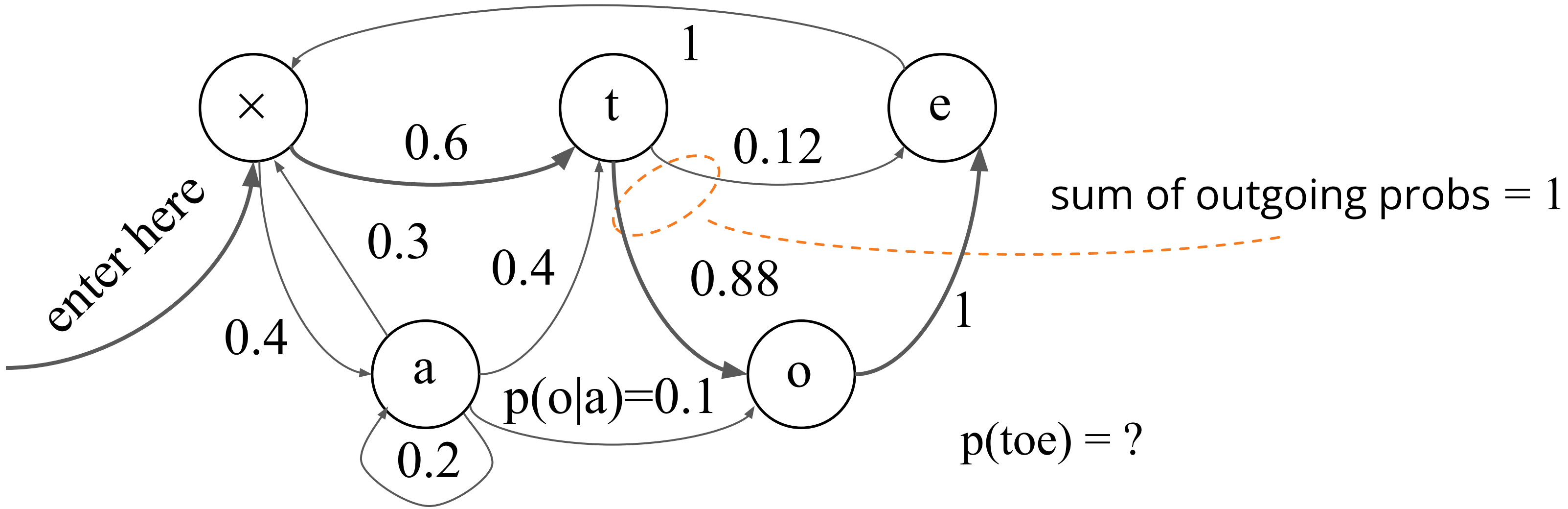
Graph Representation: State Diagram (Bigram case)

- 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



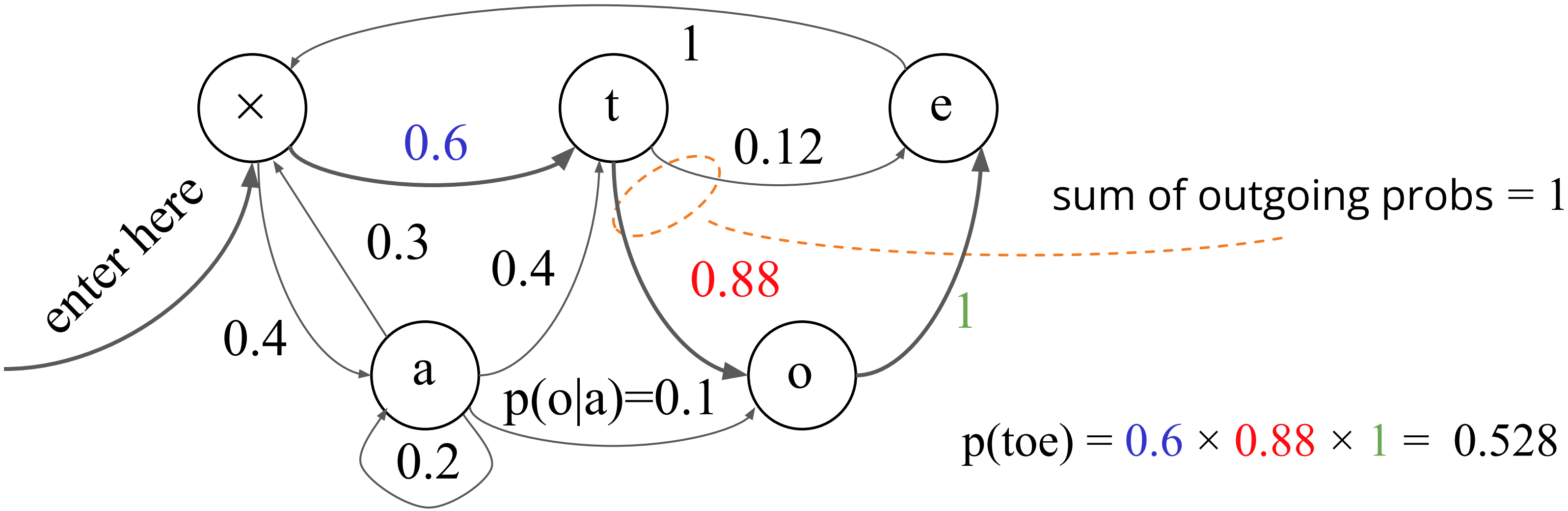
Graph Representation: State Diagram (Bigram case)

- 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



Graph Representation: State Diagram (Bigram case)

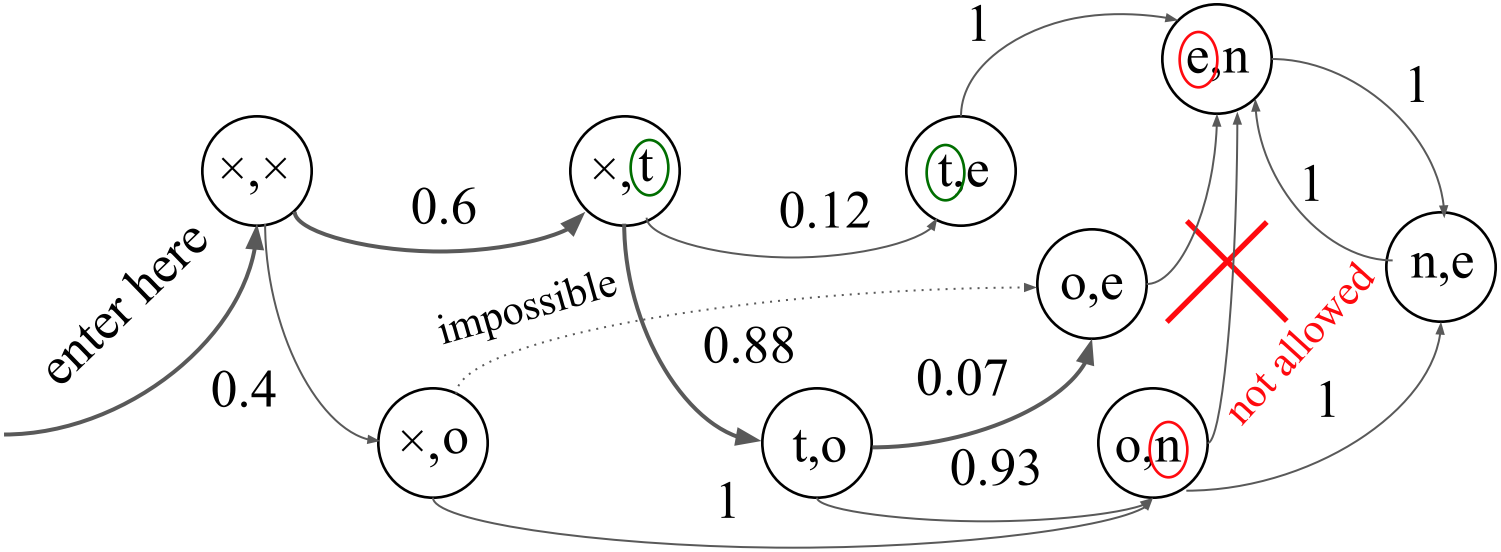
- Nodes: States, $S = \{s_0, s_1, s_2, \ldots, s_N\}$
- Arcs: Transitions with probabilities, $P(X_i|X_{i-1})$, X_i generates s_i



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

The Trigram Case

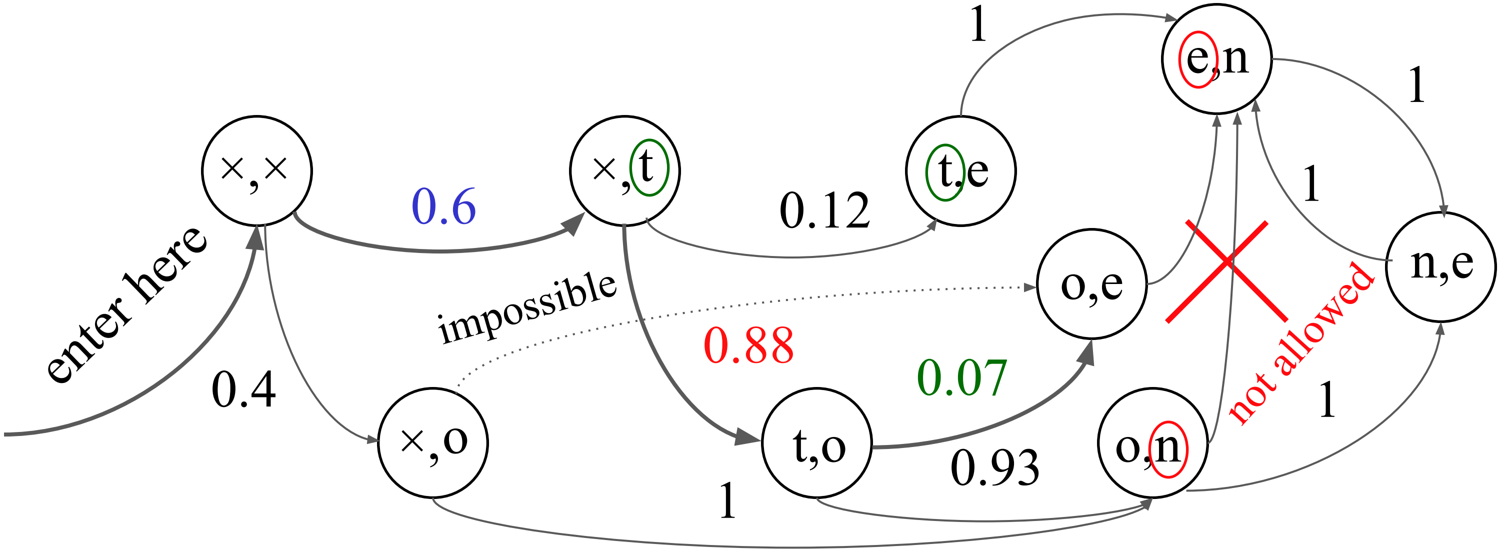
- <u>Nodes</u>: Pairs of states (s_k,s_l), $S = \{s_0,s_1,s_2,\ldots,s_N\}$
- <u>Arcs</u>: Transitions with probabilities, $P(X_i|X_{i-1})$, X_i generates (s_k,s_l)



p(toe) = ?

The 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)



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

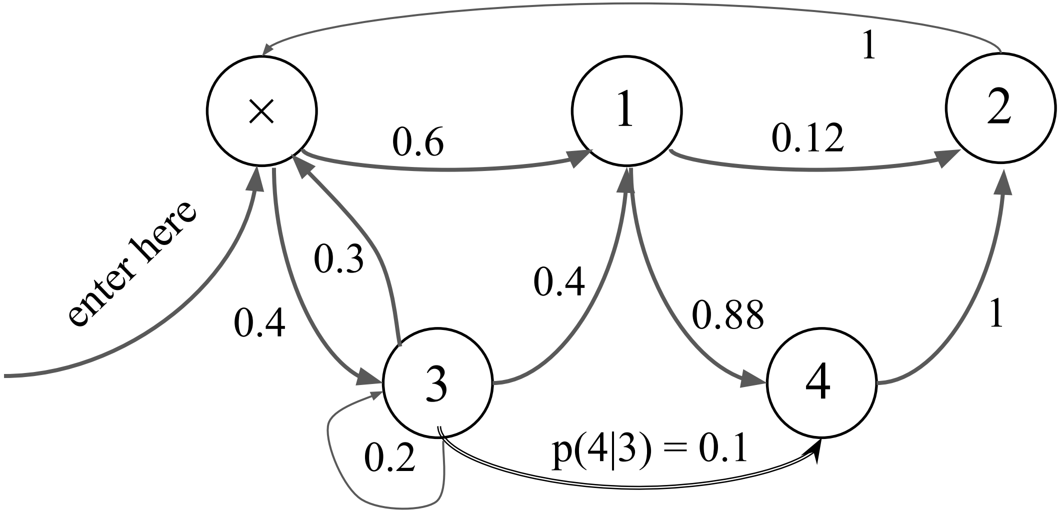
$p(one) = ?$

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

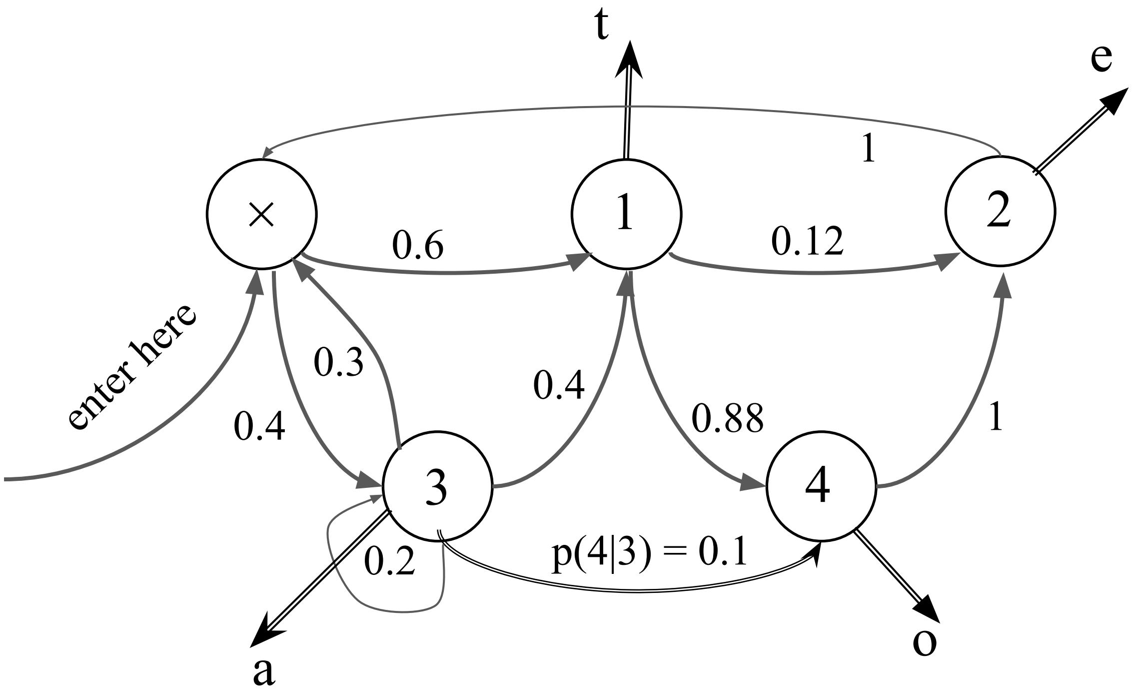
Hidden Markov Models

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



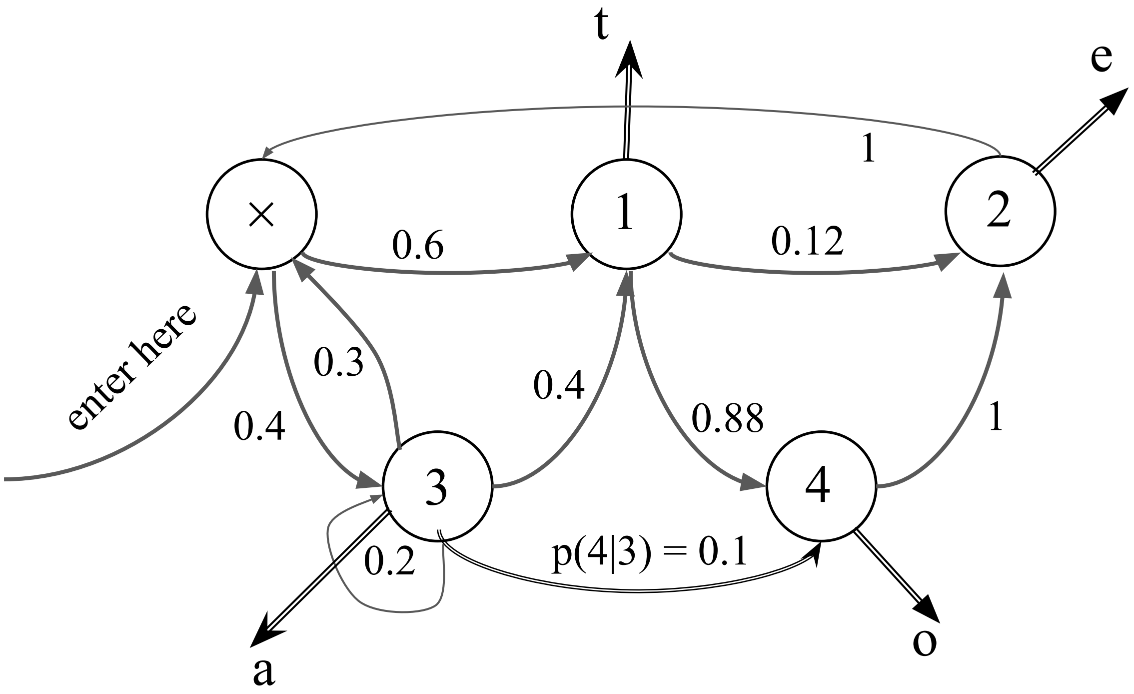
Hidden Markov Models

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



Hidden Markov Models

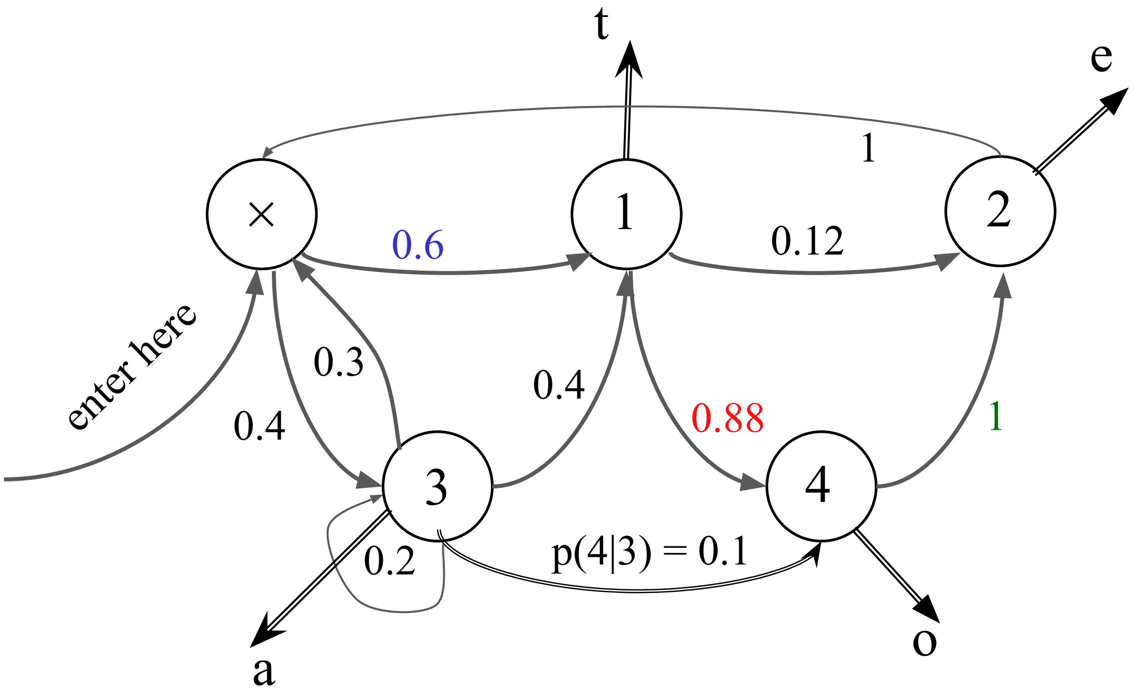
- The simplest HMM: states generate [*observable*] output (using the "data" alphabet) but remain "*invisible*":



p(toe) = ?

Hidden Markov Models

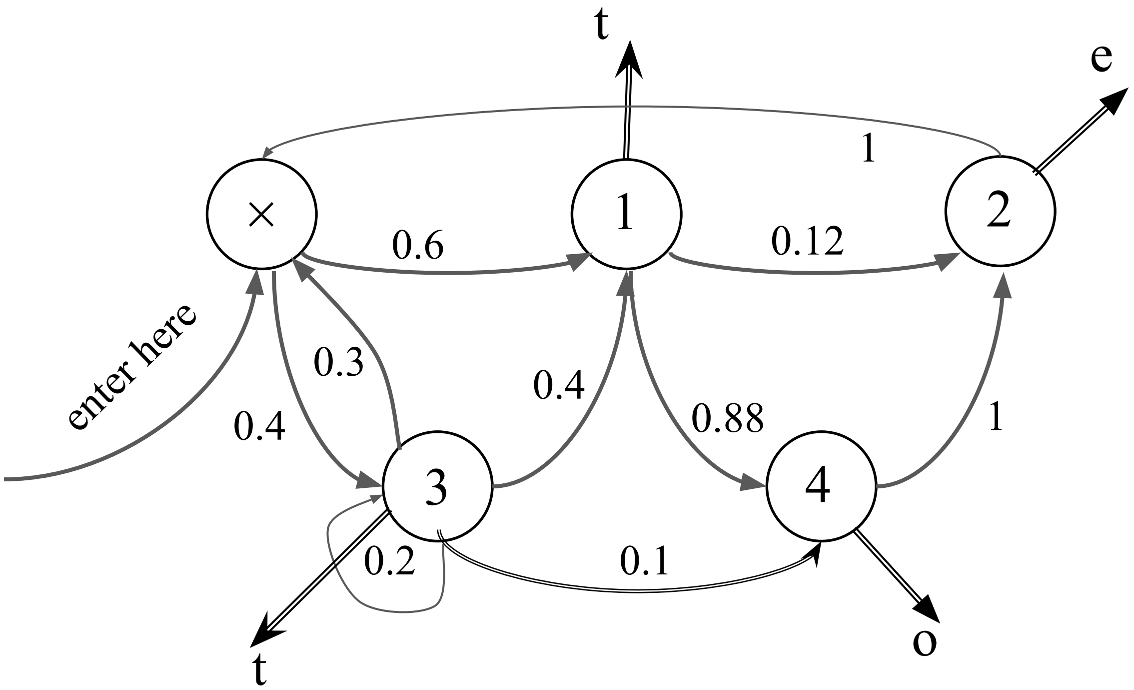
- The simplest HMM: states generate [observable] output (using the "data" alphabet) but remain "*invisible*":



p(toe) = 0.6×0.88×1= 0.528

Added Flexibility

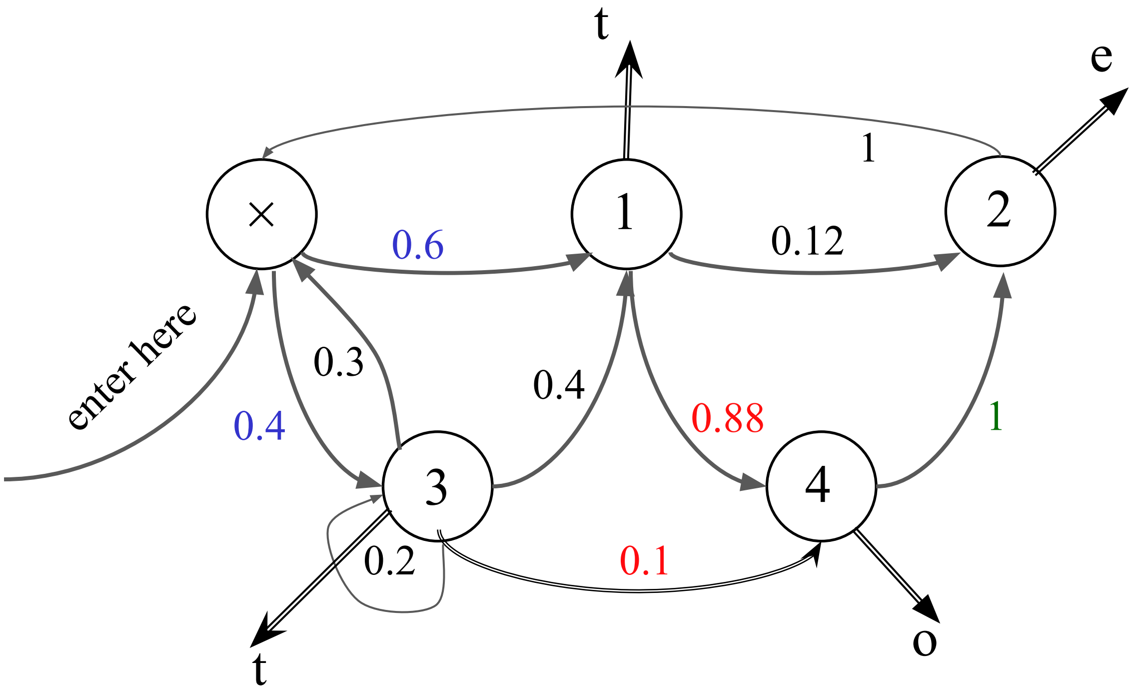
- So far, no change; but different states may generate the same output (why not?):



p(toe) = ?

Added Flexibility

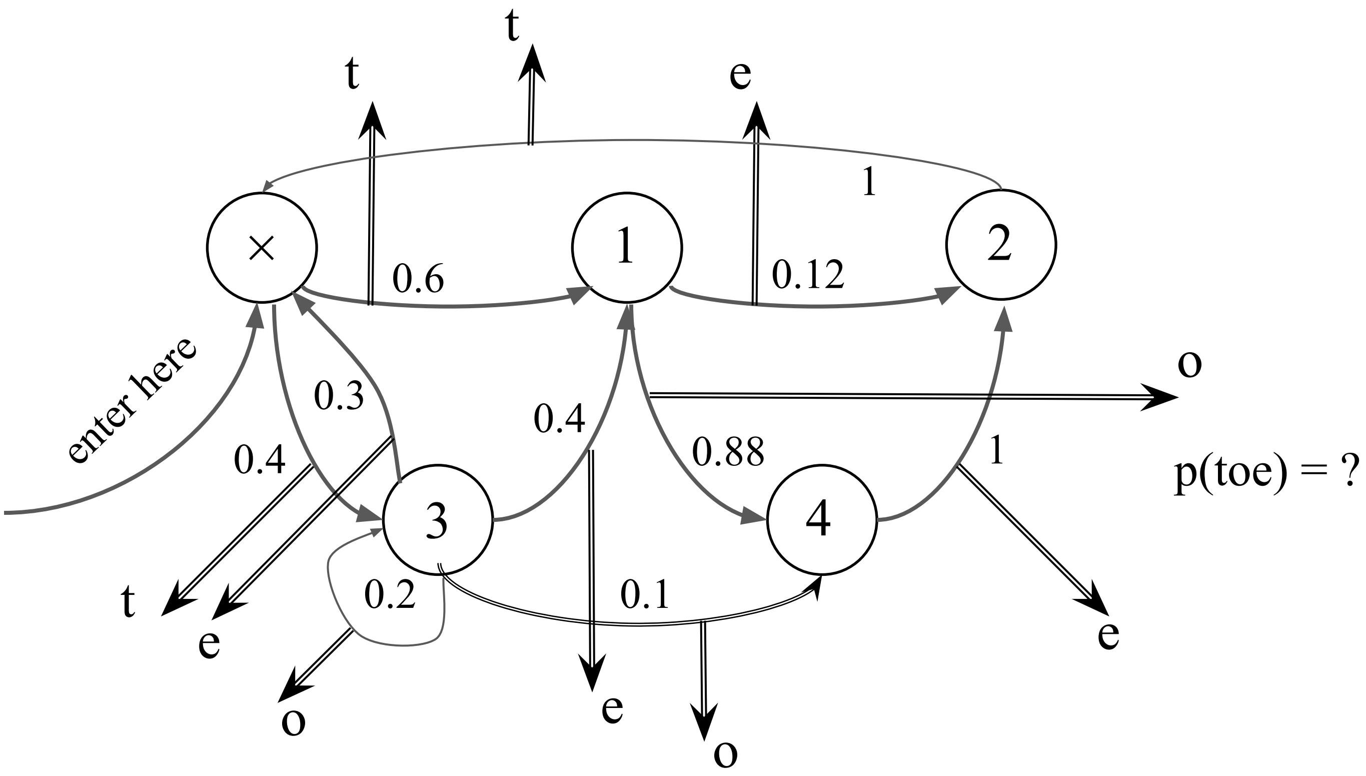
- So far, no change; but different states may generate the same output (why not?):



$$p(toe) = 0.6 \times 0.88 \times 1 + 0.4 \times 0.1 \times 1 \cong 0.568$$

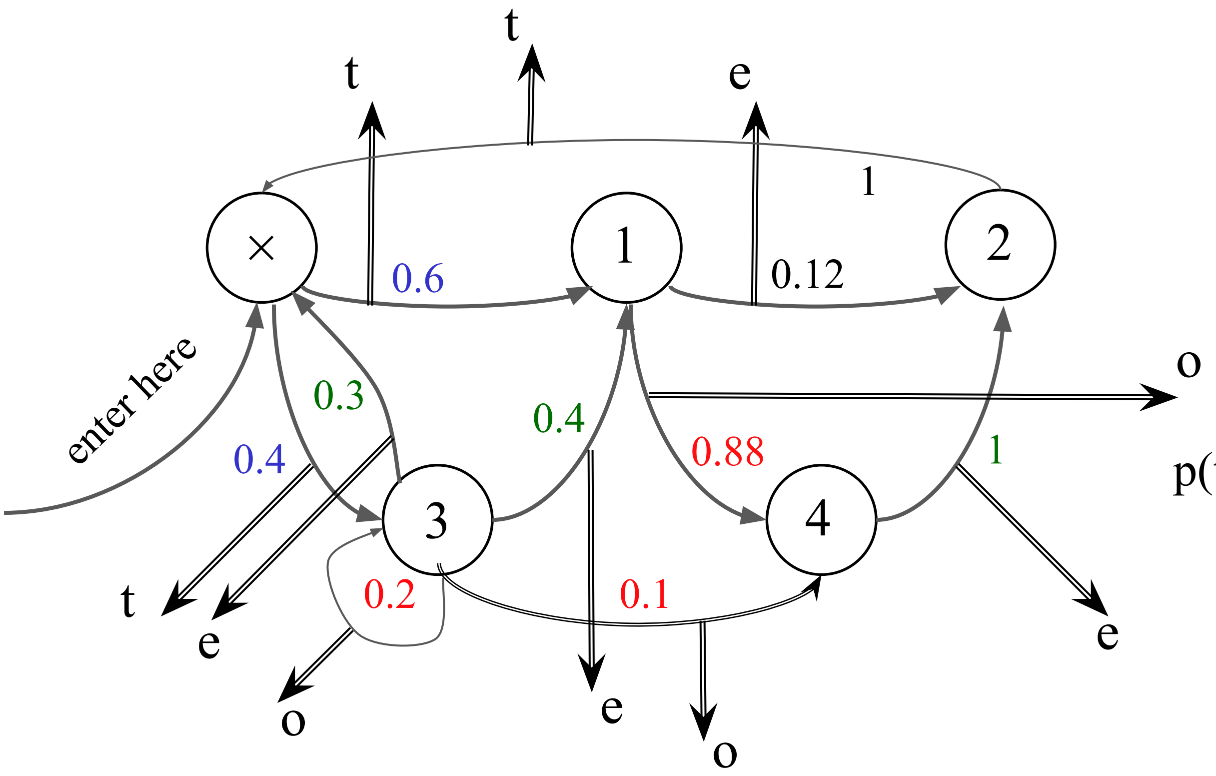
Output from Arcs...

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



Output from Arcs...

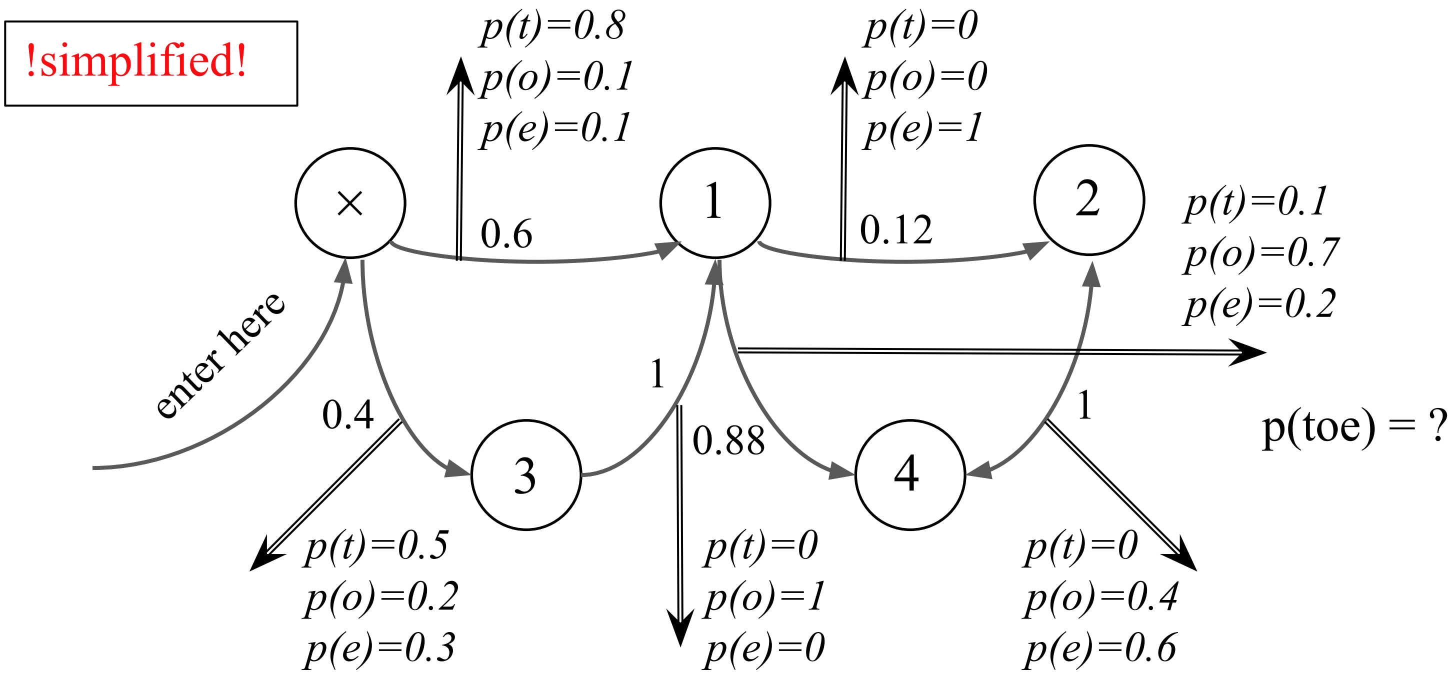
- Added flexibility: Generate output from arcs, not states:



$$
\begin{aligned}
p(toe) = {} & 0.6\times0.88\times1\ + \\
& 0.4\times0.1\times1\ + \\
& 0.4\times0.2\times0.3\ + \\
& 0.4\times0.2\times0.4 \cong 0.624
\end{aligned}
$$

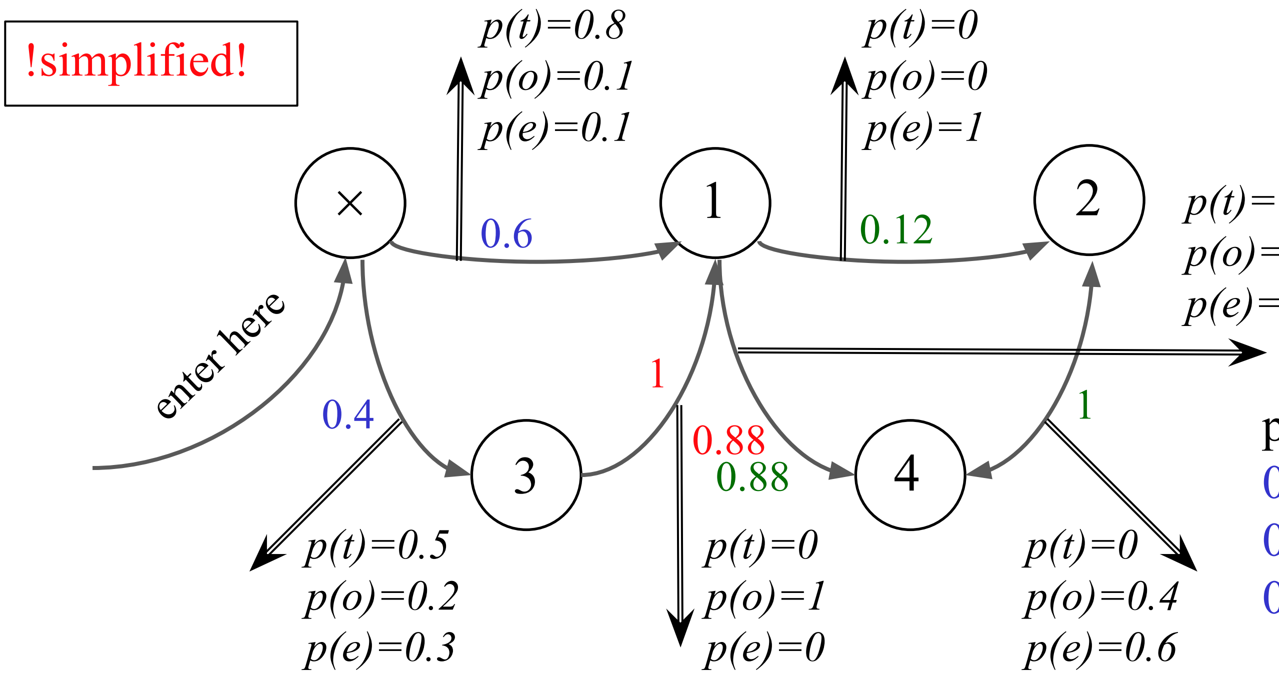
... and Finally, Add Output Probabilities

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



... and Finally, Add Output Probabilities

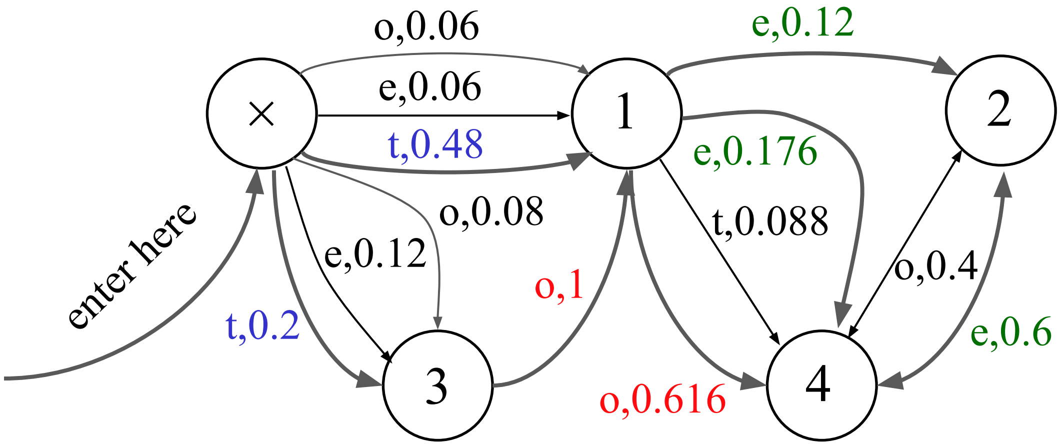
- Maximum flexibility: [Unigram] distribution (sample space: output alphabet) at each output arc:



p(toe) =
$0.6 \times 0.8 \times 0.88 \times 0.7 \times 1 \times 0.6 +$
$0.4 \times 0.5 \times 1 \times 1 \times 0.88 \times 0.2 +$
$0.4 \times 0.5 \times 1 \times 1 \times 0.12 \times 1 \cong 0.237$

Slightly Different View

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

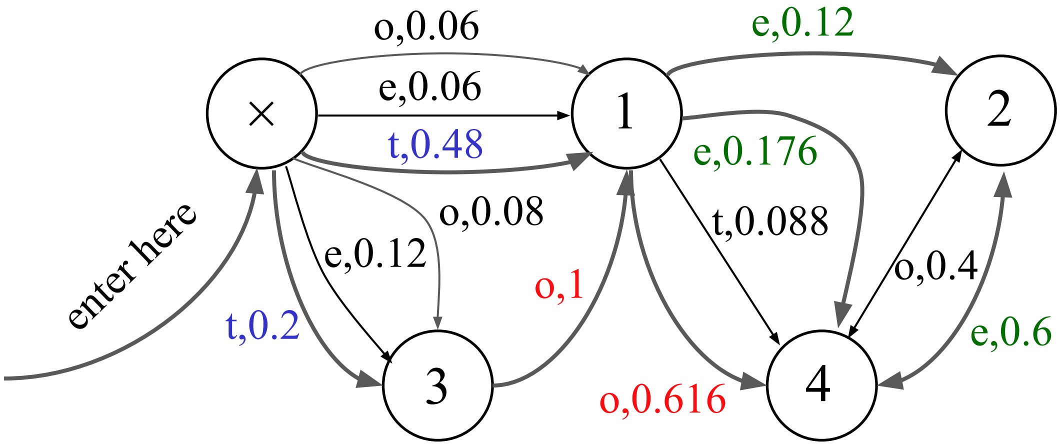


$p(toe) = ?$

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

Slightly Different View

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



$$p(\text{toe}) = 0.48\times0.616\times0.6+ 0.2\times1\times0.176 + 0.2\times1\times0.12 \cong 0.237$$

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

Formalization

- HMM (the general case): five-tuple $(\mathbf{S}, \mathbf{s_0}, \mathbf{Y}, \mathbf{P_S}, \mathbf{P_Y})$, where:
    - $\mathbf{S = \{s_1,s_2,...,s_T\}}$ is the set of states, $\mathbf{s_0}$ is the initial state,
    - $\mathbf{Y = \{y_1,y_2,...,y_V\}}$ 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: $|S|^2 \text{ x } |Y|$
- 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
×					0.7

e	×	1	2	3	4
×					0.2

$\Sigma = 1$

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:
  1. Given an output sequence $Y = \{y_1,y_2,\ldots,y_k\}$, compute its probability.
  2. 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

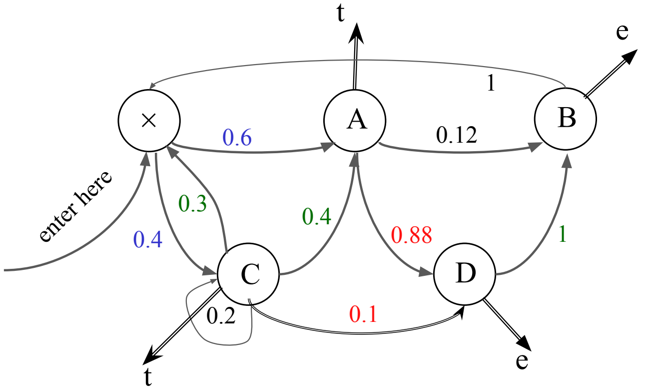
Trellis

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: HMM "roll-out"

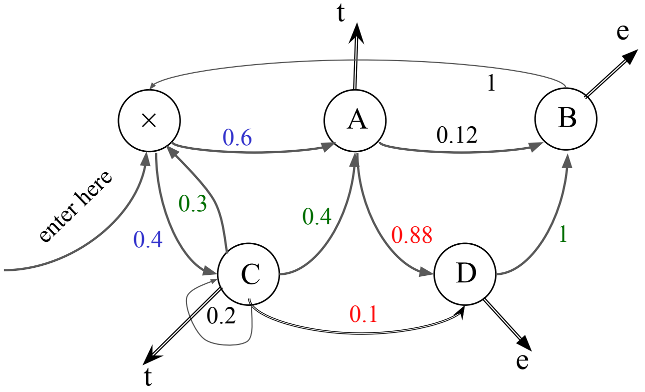
HMM:



p(toe) = 0.6×0.88×1 + 0.4×0.1×1 ≅ 0.568

Trellis: HMM “roll-out”

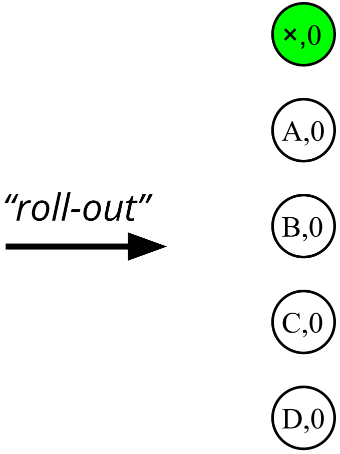
HMM:



p(toe) = 0.6×0.88×1 + 0.4×0.1×1 ≅ 0.568

Trellis:

time/position: 0

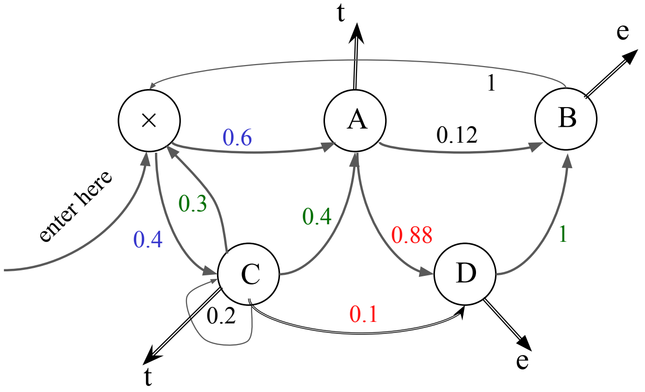


- Trellis state: (HMM state, position)

Y: t o e

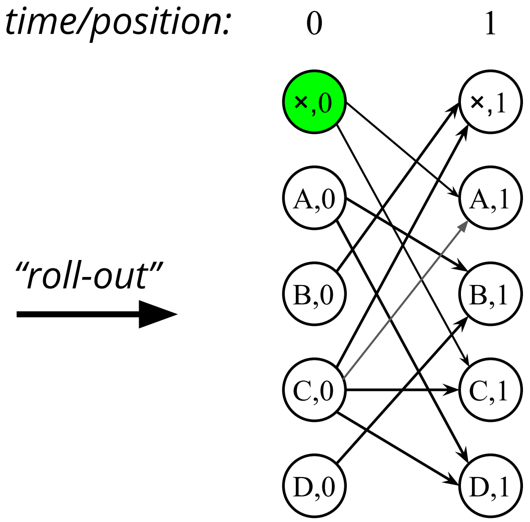
Trellis: HMM "roll-out"

HMM:



$$p(toe) = 0.6 \times 0.88 \times 1 + 0.4 \times 0.1 \times 1 \cong 0.568$$

Trellis:

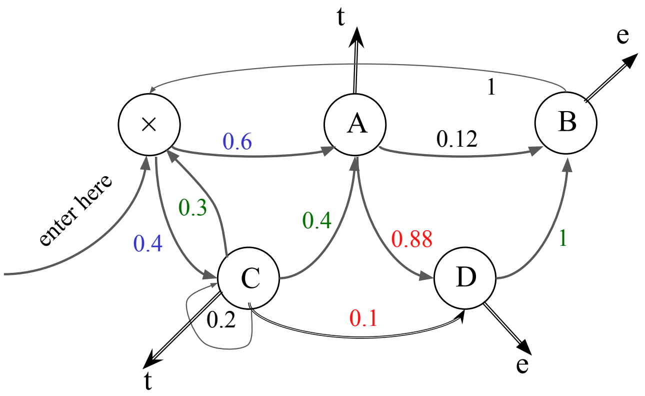


- Trellis state: (HMM state, position)

Y: t o e

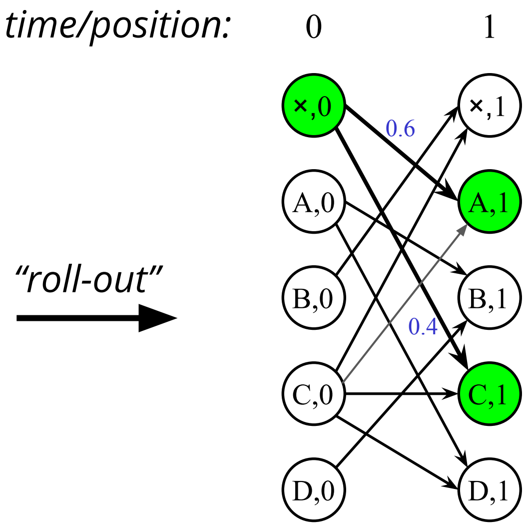
Trellis: HMM "roll-out"

HMM:



p(toe) = 0.6×0.88×1 + 0.4×0.1×1 ≅ 0.568

Trellis:

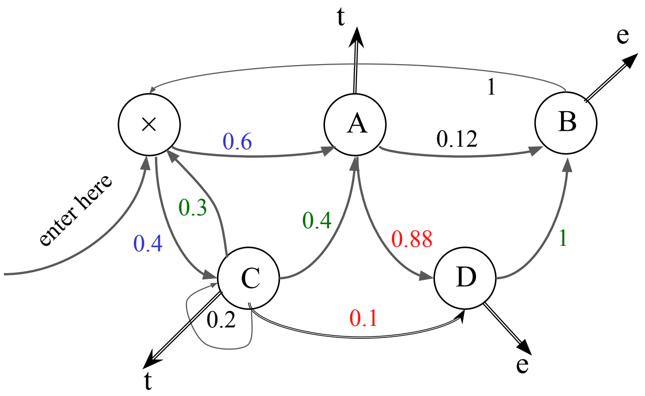


- Trellis state: (HMM state, position)

Y: t o e

Trellis: HMM “roll-out”

HMM:

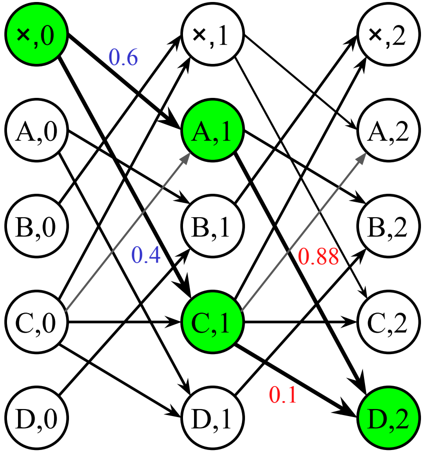


$p(toe) = 0.6 \times 0.88 \times 1 + 0.4 \times 0.1 \times 1 \cong 0.568$

Trellis:

time/position: 0 1 2

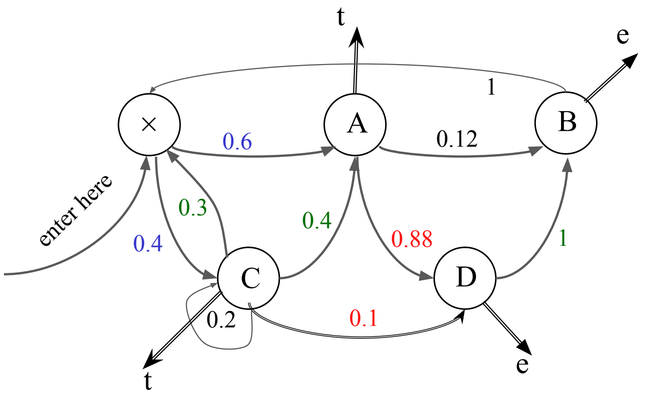
“roll-out”



- Trellis state: (HMM state, position)

Y: t o e

HMM:

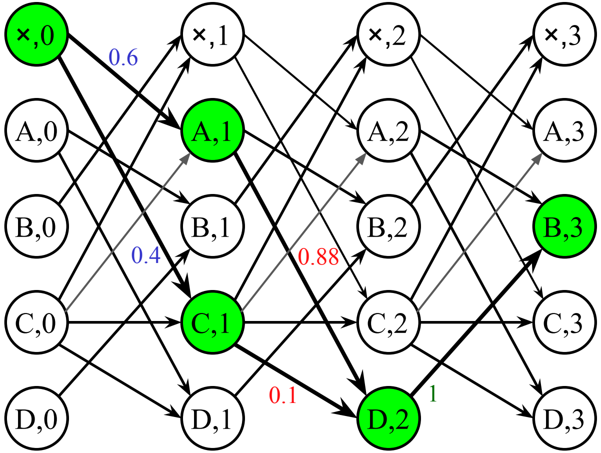


p(toe) = 0.6×0.88×1 + 0.4×0.1×1 ≅ 0.568

Trellis:

time/position: 0 1 2 3

"roll-out"

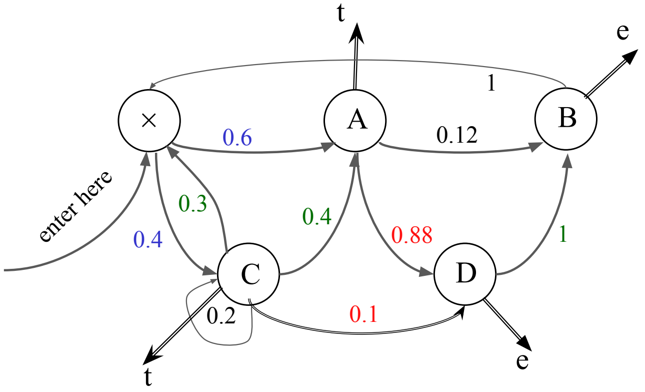


Y: t o e

- Trellis state: (HMM state, position)

Trellis: HMM “roll-out”

HMM:

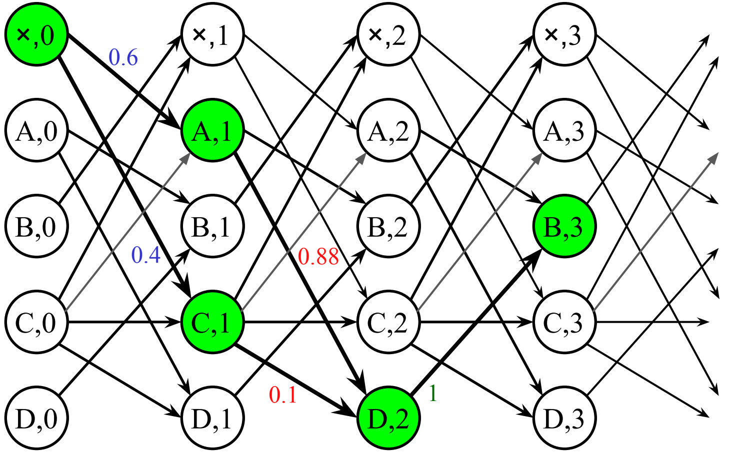


$p(\text{toe}) = 0.6 \times 0.88 \times 1 + 0.4 \times 0.1 \times 1 \cong 0.568$

“roll-out”

Trellis:

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

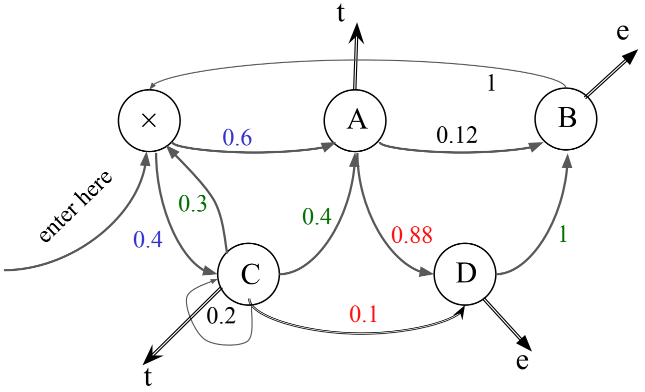


Y: t o e

- Trellis state: (HMM state, position)

Trellis: HMM “roll-out”

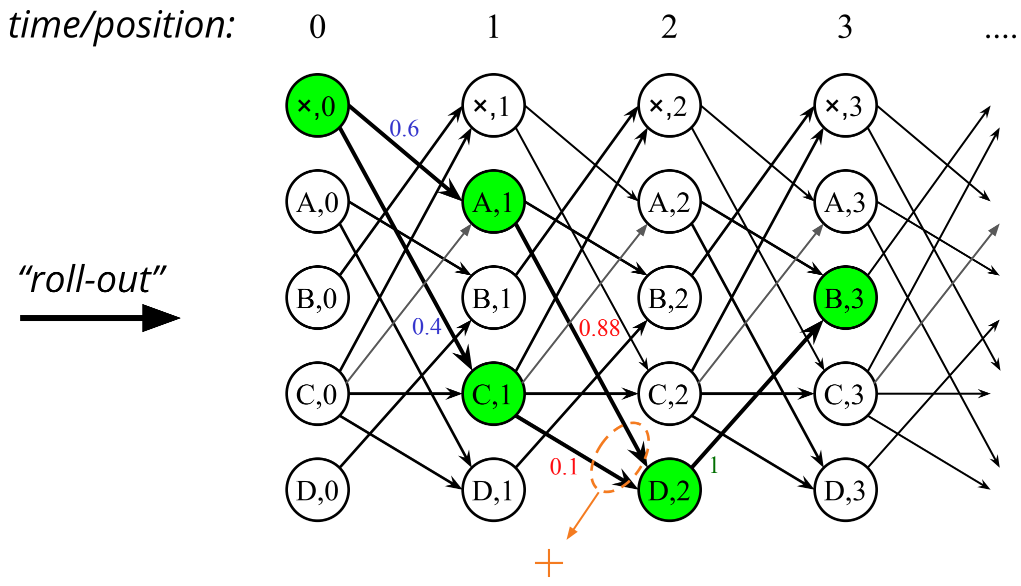
HMM:



p(toe) = 0.6×0.88×1 + 0.4×0.1×1 ≅ 0.568

Trellis:

“roll-out”



α(×,0) = 1 α(A,1) = 0.6 α(D,2) = 0.568 α(B,3) = 0.568

α(C,1) = 0.4

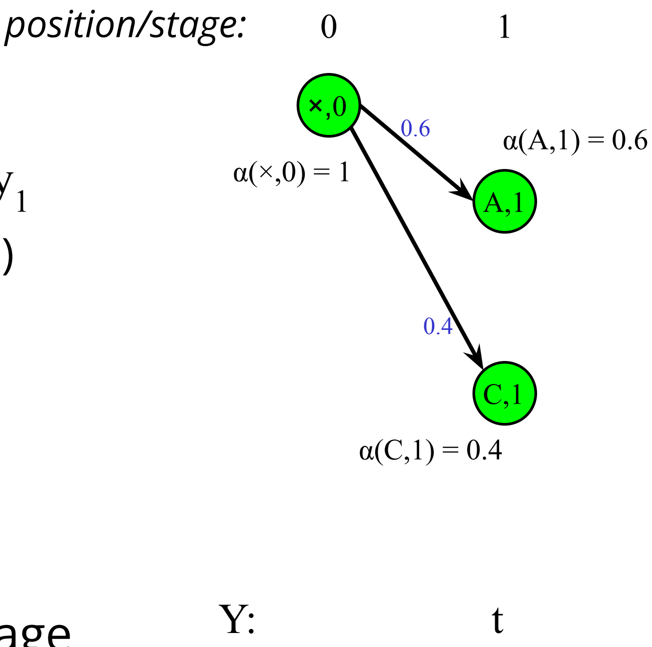
- Trellis state: (HMM state, position)
- each state: holds ***one*** number (prob): **α**
- probability of Y: Σα in the last state

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

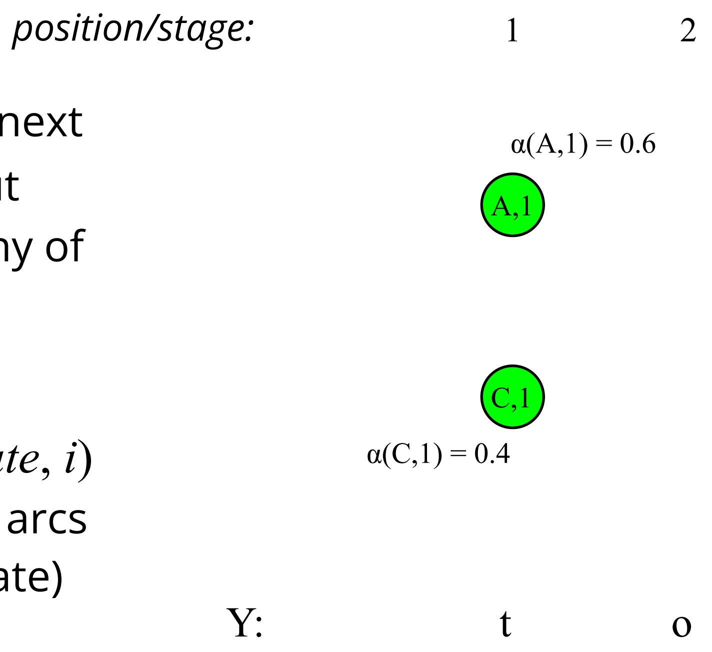
    which generate y_1
  - set their $\alpha(state, 1)$ to the

    $P_S(state|\times)\ \alpha(\times, 0)$
- and forget about the 0-th stage



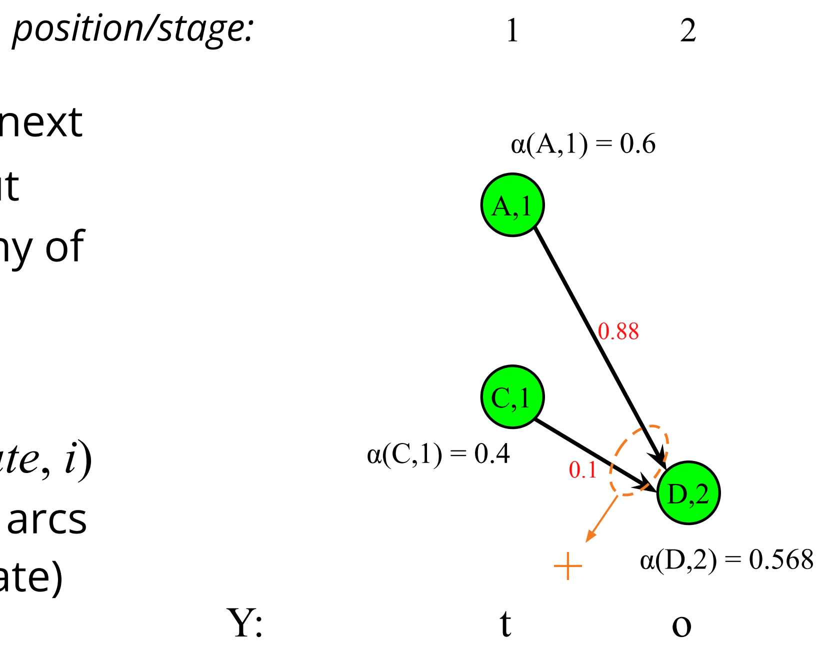
Trellis: The Next Step

- 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



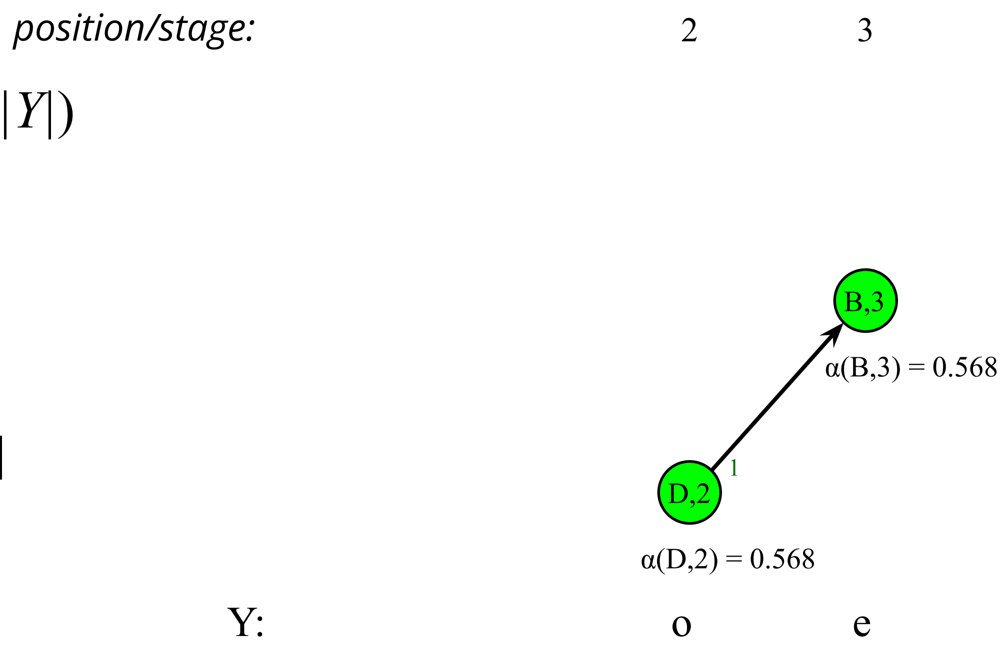
Trellis: The Next Step

- 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



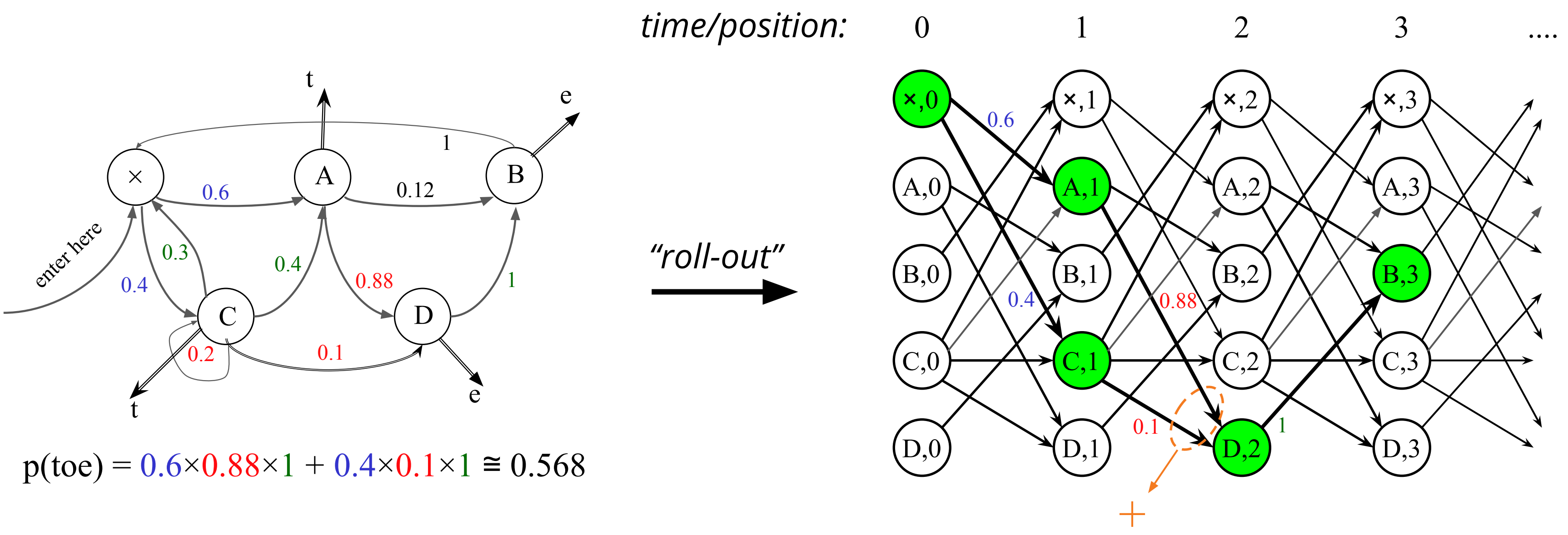
Trellis: The Last Step

- 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: Trellis:



p(toe) = 0.6×0.88×1 + 0.4×0.1×1 ≅ 0.568

α(×,0) = 1 α(A,1) = 0.6 α(D,2) = 0.568 α(B,3) = 0.568
α(C,1) = 0.4

- Trellis state: (HMM state, position)
- each state: holds ***one*** number (prob): **α**
- probability of Y: Σα 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:
  - $\mathbf{S = \{s_1,s_2,...,s_T\}}$ is the set of states, $\mathbf{S_0}$ is the initial state,
  - $\mathbf{Y = \{y_1,y_2,...,y_V\}}$ 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 $Y = \{y_1,y_2,...,y_k\}$:
  - (Task 1) compute the probability of Y;
  - (Task 2) compute the most likely sequence of states which has generated Y.

Moodle Quiz

Moodle Quiz

https://dl1.cuni.cz/course/view.php?id=18547

Trellis: The General Case (still, bigrams)

- B

General Trellis: The Next Step

- B

Trellis: The Complete Example

- B

The Case of Trigrams

- B

Trigrams with Classes

- B

Class Trigrams: the Trellis

- B

Overlapping Classes

- B

Overlapping Classes: Trellis Example

- B

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 Algorithm alias Baum-Welch)

- Implementation issues:
  - scaling/normalizing probabilities, to avoid too small numbers & addition problems with many transitions