Language Technology
Fall 2014

Unit 3: Natural Language Learning

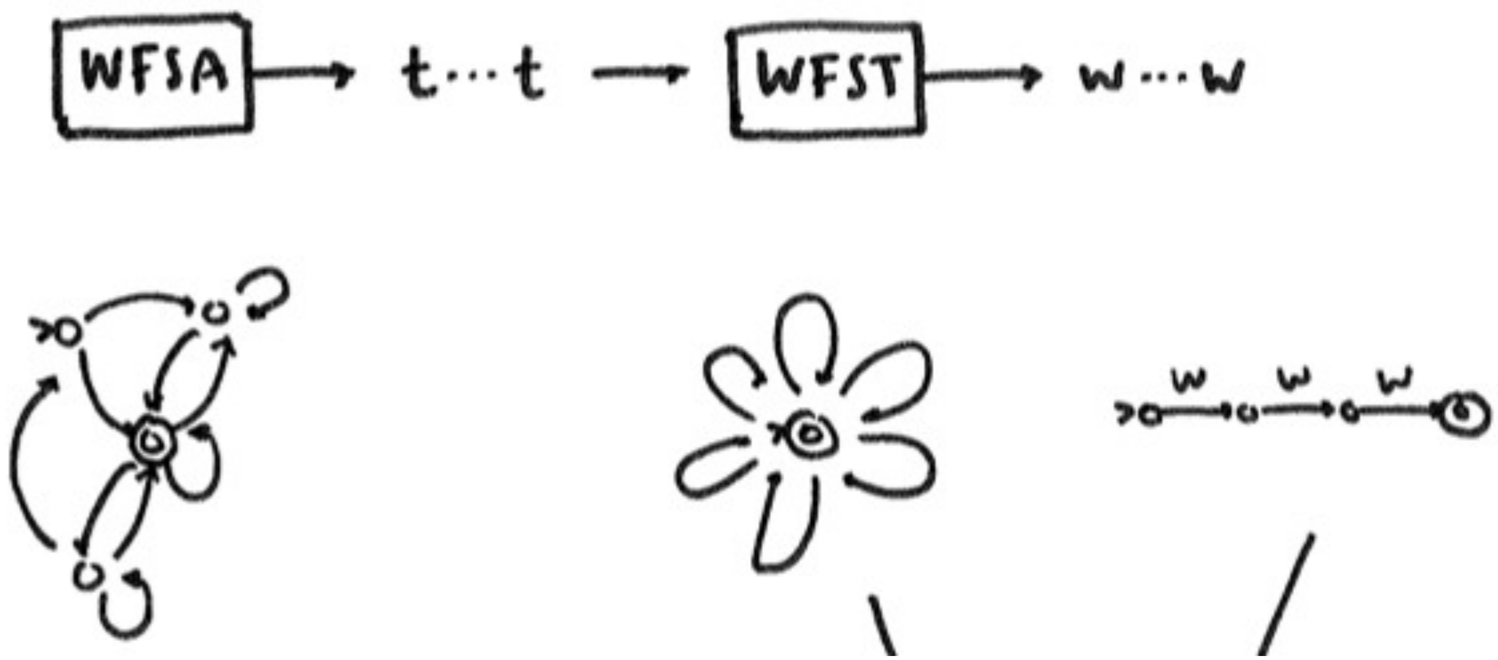
Unsupervised Learning

(EM, forward-backward, inside-outside)

Liang Huang

liang.huang.sh@gmail.com

Review of Noisy-Channel Model



Application	Input	Output	p(*i*)	p(*o*|*i*)
Machine Translation	L_1 word sequences	L_2 word sequences	p(L_1) in a language model	translation model
Optical Character Recognition (OCR)	actual text	text with mistakes	prob of language text	model of OCR errors
Part Of Speech (POS) tagging	POS tag sequences	English words	prob of POS sequences	p(*w*|*t*)
Speech recognition	word sequences	speech signal	prob of word sequences	acoustic model

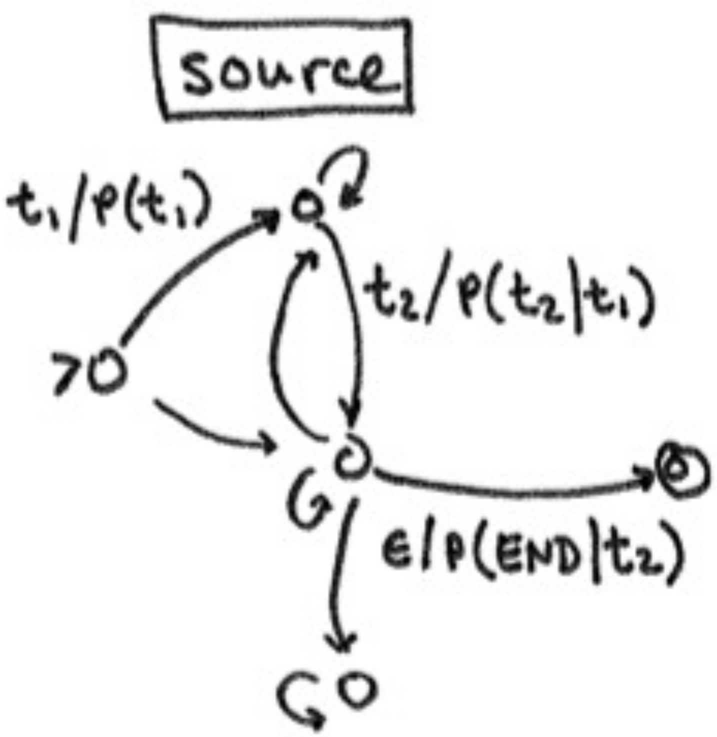
Example 1: Part-of-Speech Tagging

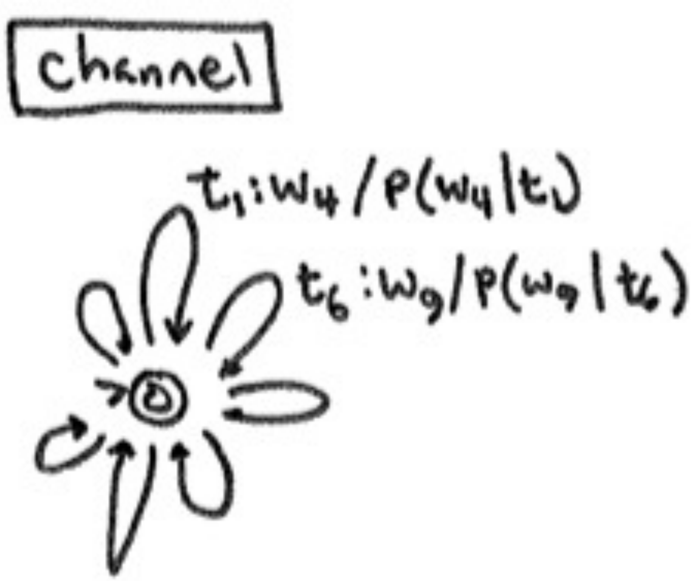
$$P(t \cdots t \mid w \cdots w)$$

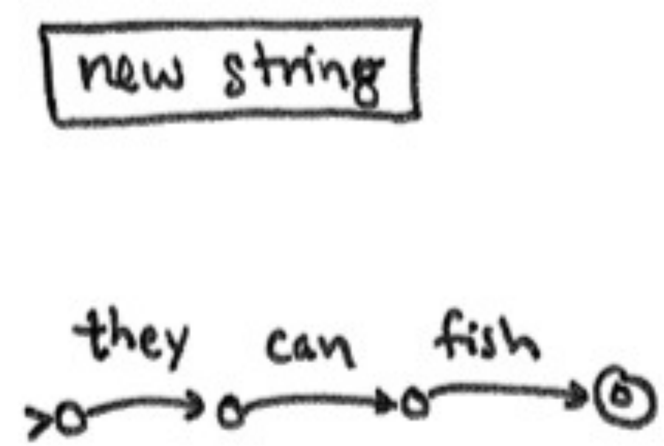
$$\sim P(t \cdots t) \cdot P(w \cdots w \mid t \cdots t)$$

$$\sim \underbrace{P(t_1) \cdot P(t_2 \mid t_1) \cdots P(t_n \mid t_{n-1})}_{\text{local grammar preference}} \cdot \underbrace{P(w_1 \mid t_1) \cdots P(w_n \mid t_n)}_{\text{lexical preference}}$$

- use tag bigram as a language model
- channel model is context-indep.







Ideal vs. Available Data

WFSA → t···t → WFST → w···w

CRYPTOGRAPHY

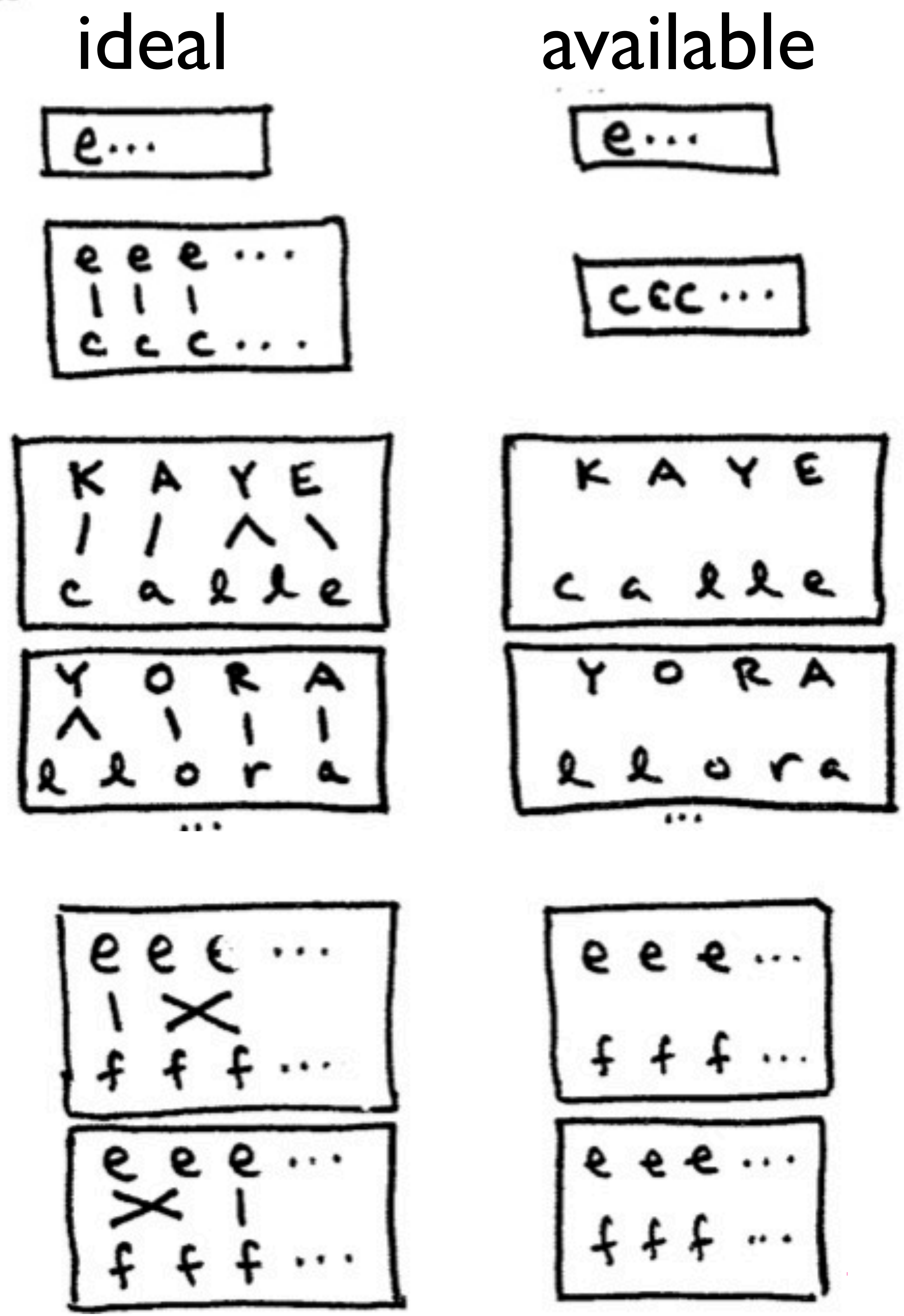
1. generate $e_1 \cdots e_n$ by $P(e_k | e_{k-1})$
2. for $i = 1$ to n
   output c_i by $P(c_i | e_i)$

SPELLING-TO-SOUND

1. generate $pho_1 \cdots pho_n$
2. transform into $c_1 \cdots c_m$ by WFST

MT

1. generate $e_1 \cdots e_n$ by $P(e_k | e_{k-1})$
2. for $i = 1$ to n
   generate f_i by $P(f_i | e_i)$
3. permute all f_i by $1/n!$



Ideal vs. Available Data

HW2: ideal

```
EY B AH L
A  B E  R U
1  2 3  4 4

AH B AW T
A  B A  U T O
1  2 3  3 4 4

AH L ER T
A  R A  A T O
1  2 3  3 4 4

EY S
E  E S U
1  1 2 2
```

HW4: realistic

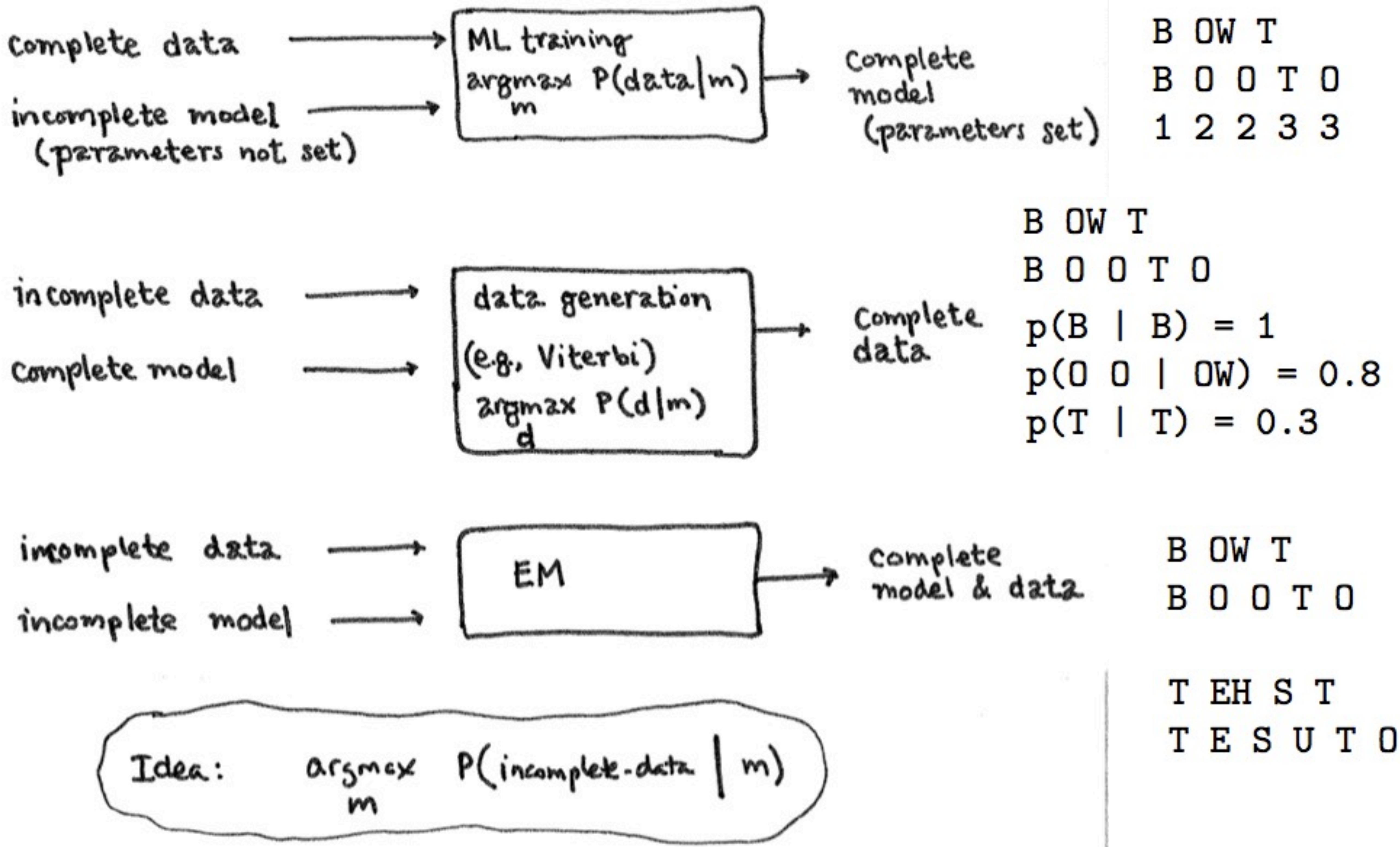
```
EY B AH L
A  B E  R U

AH B AW T
A  B A  U T O

AH L ER T
A  R A  A T O

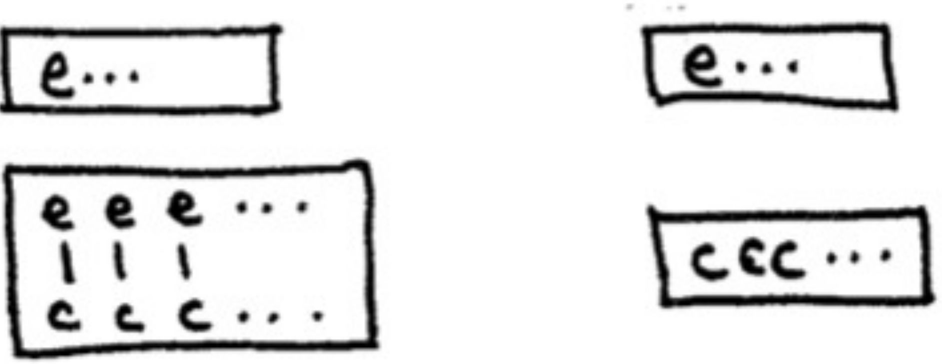
EY S
E  E S U
```

Incomplete Data / Model



EM: Expectation-Maximization

Example: Cryptography. $\underset{m}{\text{argmax}}\ P(c_1 \cdots c_n \mid m)$



$$\underset{m}{\text{argmax}} \sum_{e_1 \cdots e_n} P(e_1 \cdots e_n) \cdot P(c_1 \cdots c_n \mid e_1 \cdots e_n, m)$$

$$\underset{m}{\text{argmax}} \sum_{e_1 \cdots e_n} P(e_1 \cdots e_n) \cdot P(c_1 \mid e_1, m) \cdots P(c_n \mid e_n, m)$$

each choice of m yields a specific number!
some m are better than others!

which is best?

EM:

start with m such that $P(c_i \mid e_j, m) = 1/27$.

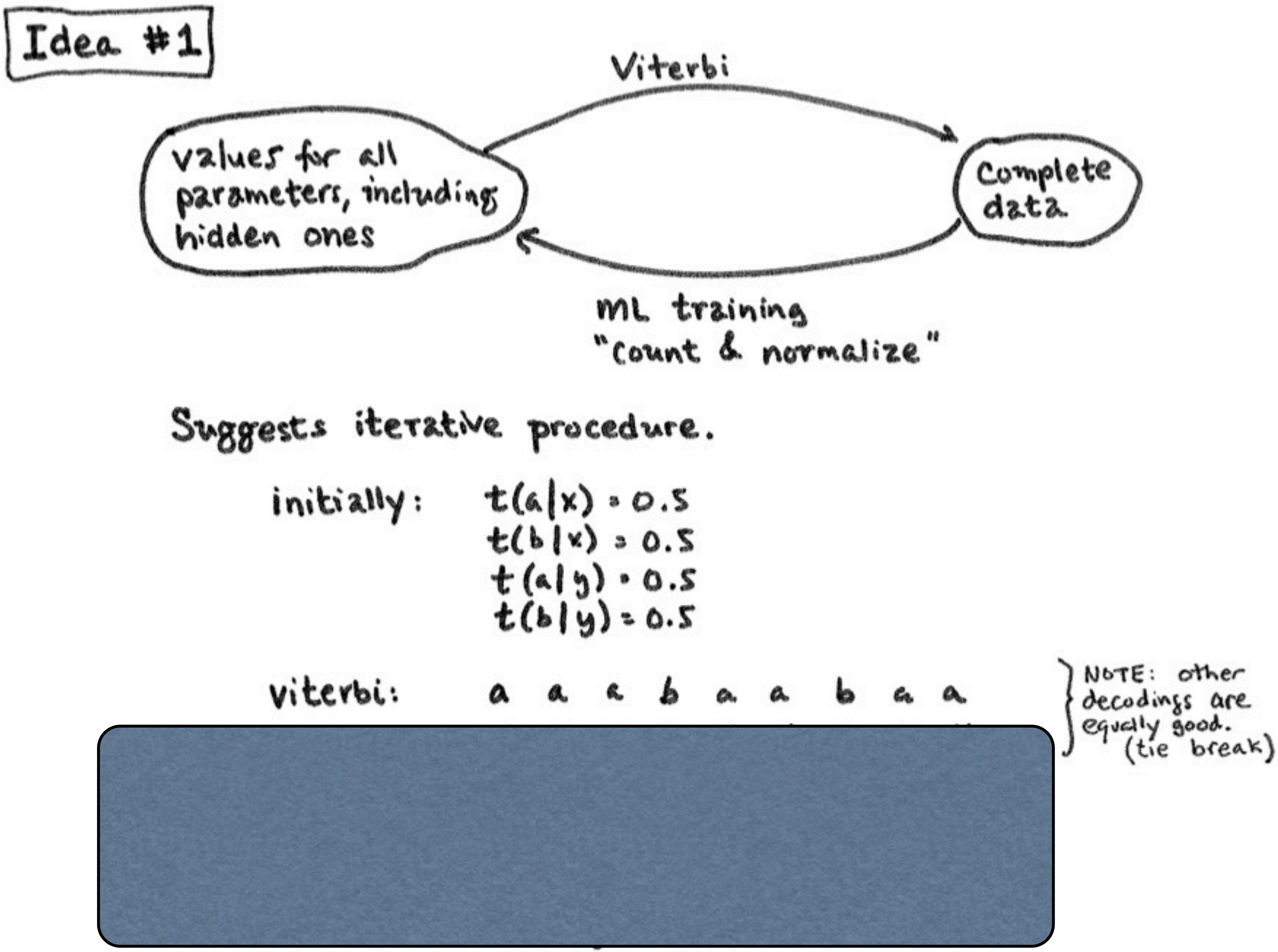
that gives a certain $P(c_1 \cdots c_n \mid m)$.

now, change m to m' such that

$$P(c_1 \cdots c_n \mid m') \geq P(c_1 \cdots c_n \mid m)$$

(& repeat)

How to Change *m*? 1) Hard



How to Change *m*? 1) Hard

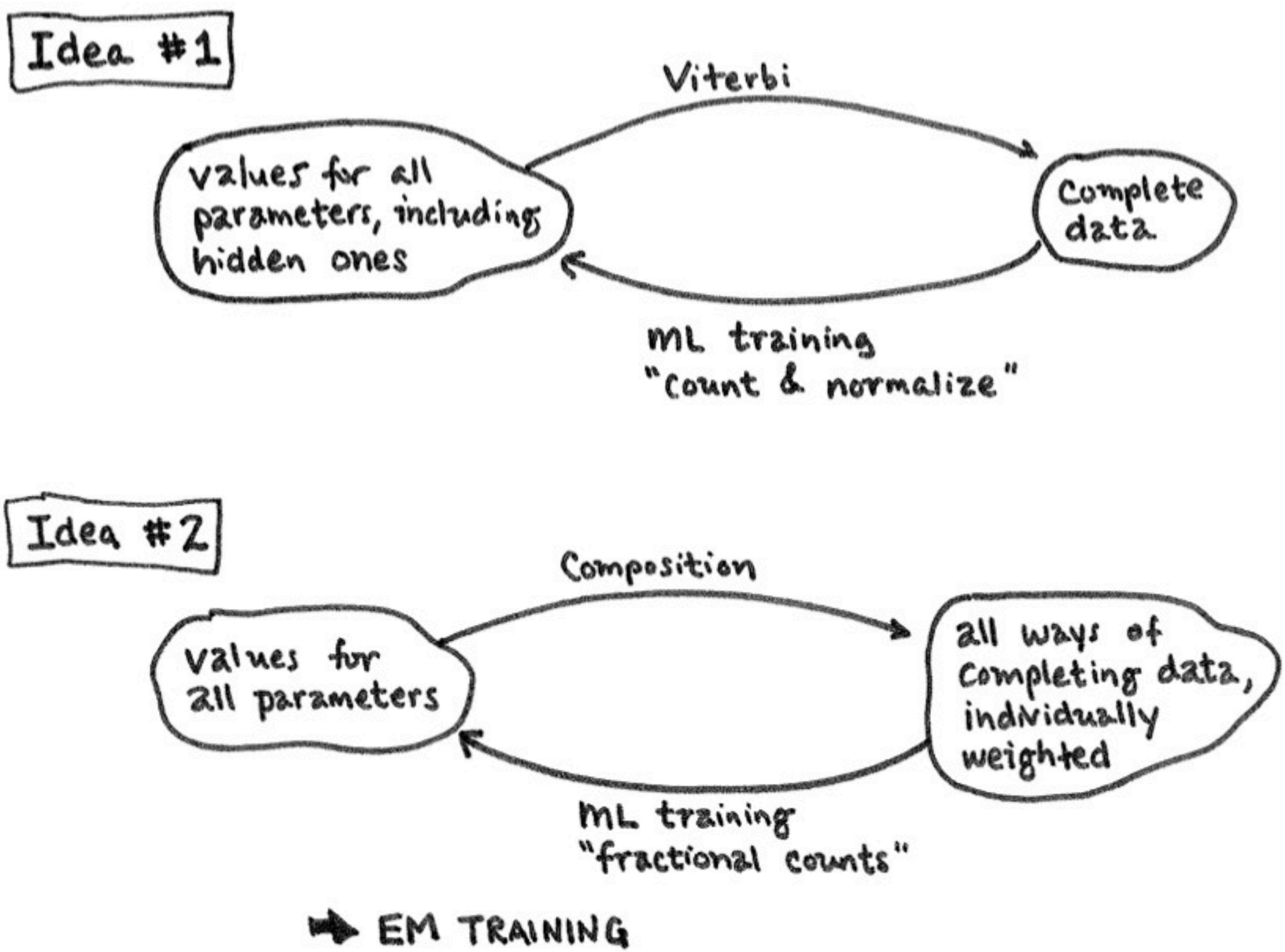
viterbi: a a a b a a b a a
y x x x x x y x x

NOTE: other decodings are equally good. (tie break)

revised: $t(a|x) = 6/7$
$t(b|x) = 1/7$
$t(a|y) = 1/2$
$t(b|y) = 1/2$

revised viterbi: a a a b a a b a a

How to Change *m*? 2) Soft



Fractional Counts

- distribution over all possible hallucinated hidden variables
- ```
  W AI N         W AI N       W AI N       W AI N
                 | |  / \     |  |\  \     |\  \  \
  W A I N        W A I N      W A I N      W A I N
  ```

hard-EM counts   1   0   0

fractional counts   0.333   0.333   0.333

```
AY|->   A: 0.333      A I: 0.333     I: 0.333
W|->    W: 0.667      W A: 0.333
N|->    N: 0.667      I N: 0.333
```

fractional counts   0.25   0.5   0.25

```
AY|->   A I: 0.500    A: 0.250       I: 0.250
W|->    W: 0.750      W A: 0.250
N|->    N: 0.750      I N: 0.250
```

eventually   ... 0   ... 1   ... 0


```

Fractional Counts

- how about

```
W EH T

W E T O

B IY         B IY
|  |\        |\  \
B I I        B I I
```

- so EM can possibly: (1) learn something correct (2) learn something wrong (3) doesn't learn anything
- but with lots of data => likely to learn something good

EM: slow version (non-DP)

- initialize the conditional prob. table to uniform
- repeat until converged:

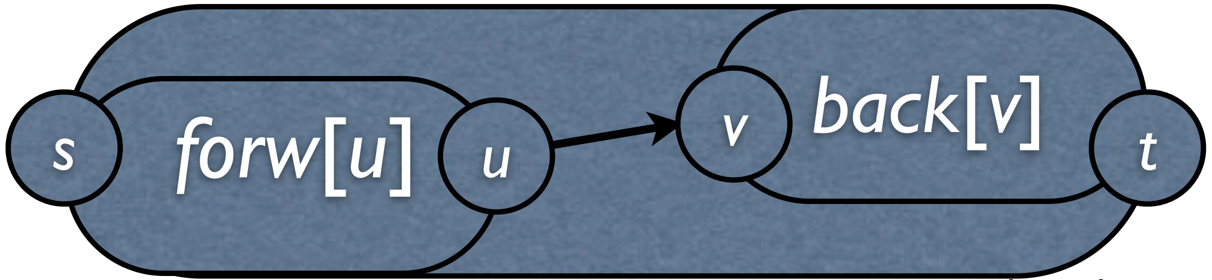
```
W AI N    W AI N    W AI N
| |  /\   | |\  \   |\  \  \
W A I N   W A I N   W A I N
```

z $(z_1\ z_2\ z_3)$ $\quad z' \quad z''$

  - E-step:
    - for each training example x (here: (e...e, j...j) pair):
      - for each hidden z: compute $p(x, z)$ from the current model
      - $p(x) = \text{sum}_z\, p(x, z)$; [debug: corpus prob $p(\text{data})$ *= $p(x)$]
      - for each hidden $z = (z_1\ z_2 \ldots z_n)$: for each i:
        - fraccount(z_i) += $p(x, z)$ / $p(x)$
  - M-step: count-n-divide on fraccounts => new model

EM: fast version (DP)

- initialize the conditional prob. table to uniform
- repeat until converged:



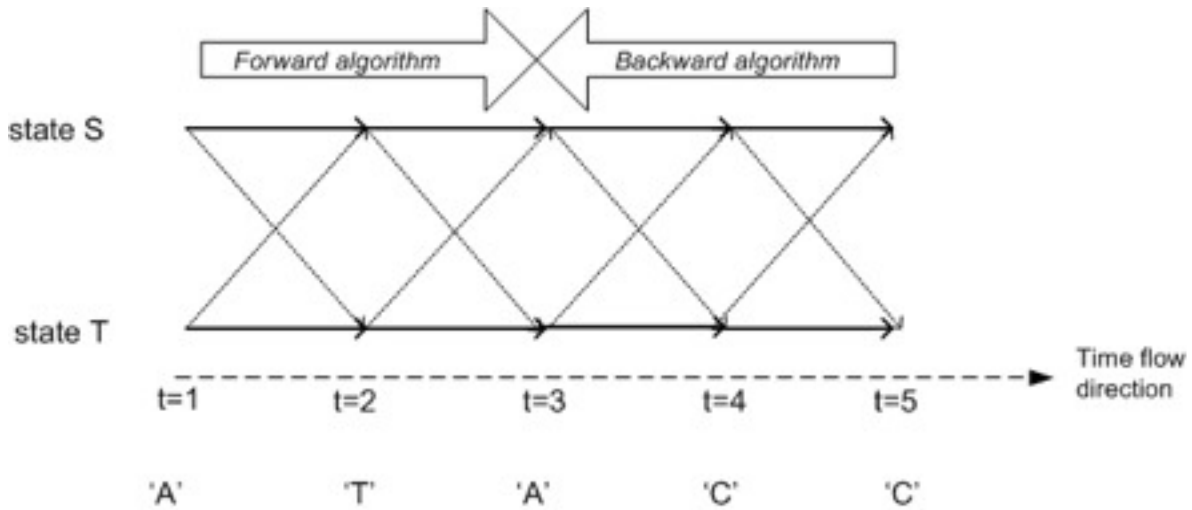
forw[t] = back[s] = p(x) = sum$_z$ p(x, z)

  - E-step:
    - for each training example *x* (here: (e...e, j...j) pair):
      - forward from *s* to *t*; note: *forw*[*t*] = *p*(*x*) = sum$_z$ *p*(*x*, *z*)
      - backward from *t* to *s*; note: *back*[*t*]=1; *back*[*s*] = *forw*[*t*]
      - for each edge (*u*, v) in the DP graph with *label*(*u*, v) = z_i
        - fraccount(z_i) += *forw*[*u*] * *back*[v] * *prob*(*u*, v) / *p*(*x*)
  - M-step: count-n-divide on fraccounts => new model

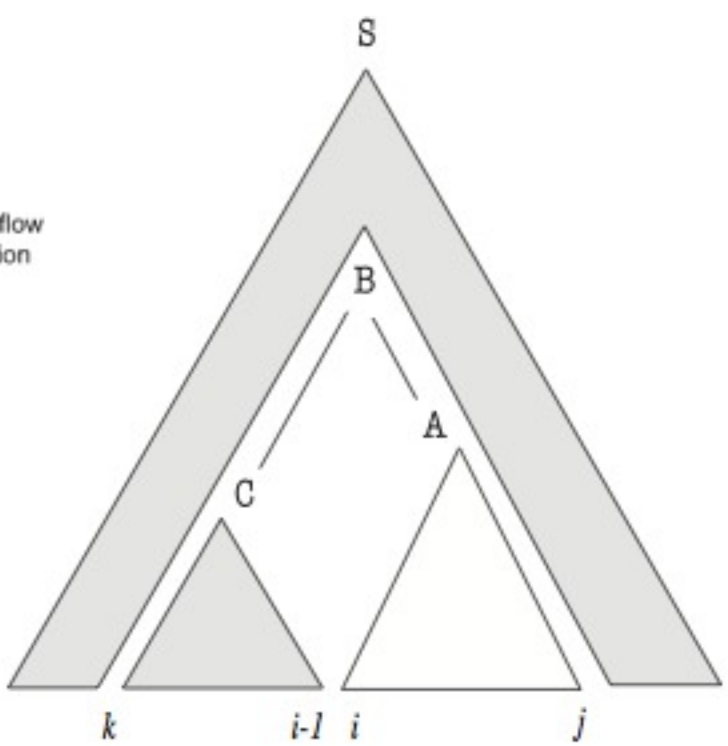
sum$_{z: (u, v) \text{ in } z}$ *p*(*x*, *z*)

How to avoid enumeration?

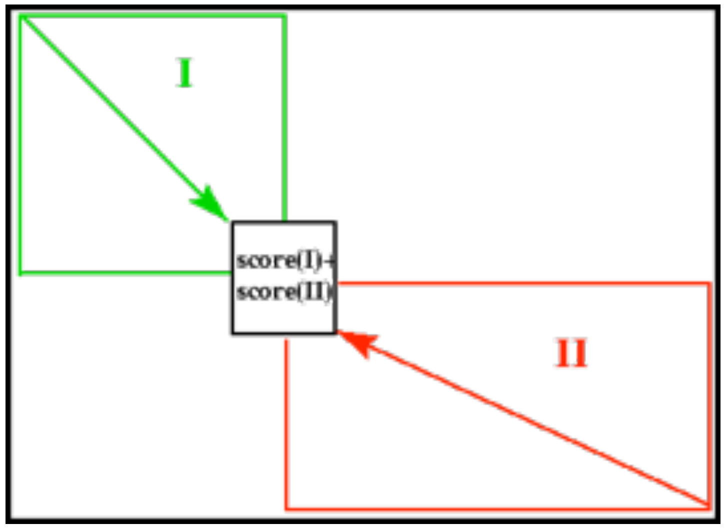
- dynamic programming: the forward-backward algorithm
- forward is just like Viterbi, replacing max by sum
- backward is like reverse Viterbi (also with sum)



POS tagging, crypto, ...



inside-outside: PCFG, SCFG, ...



alignment, edit-distance, ...

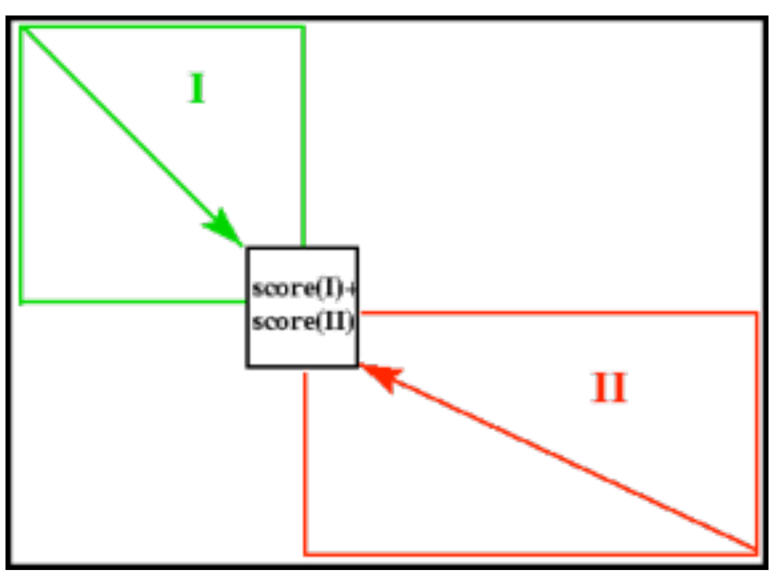
Example Forward Code

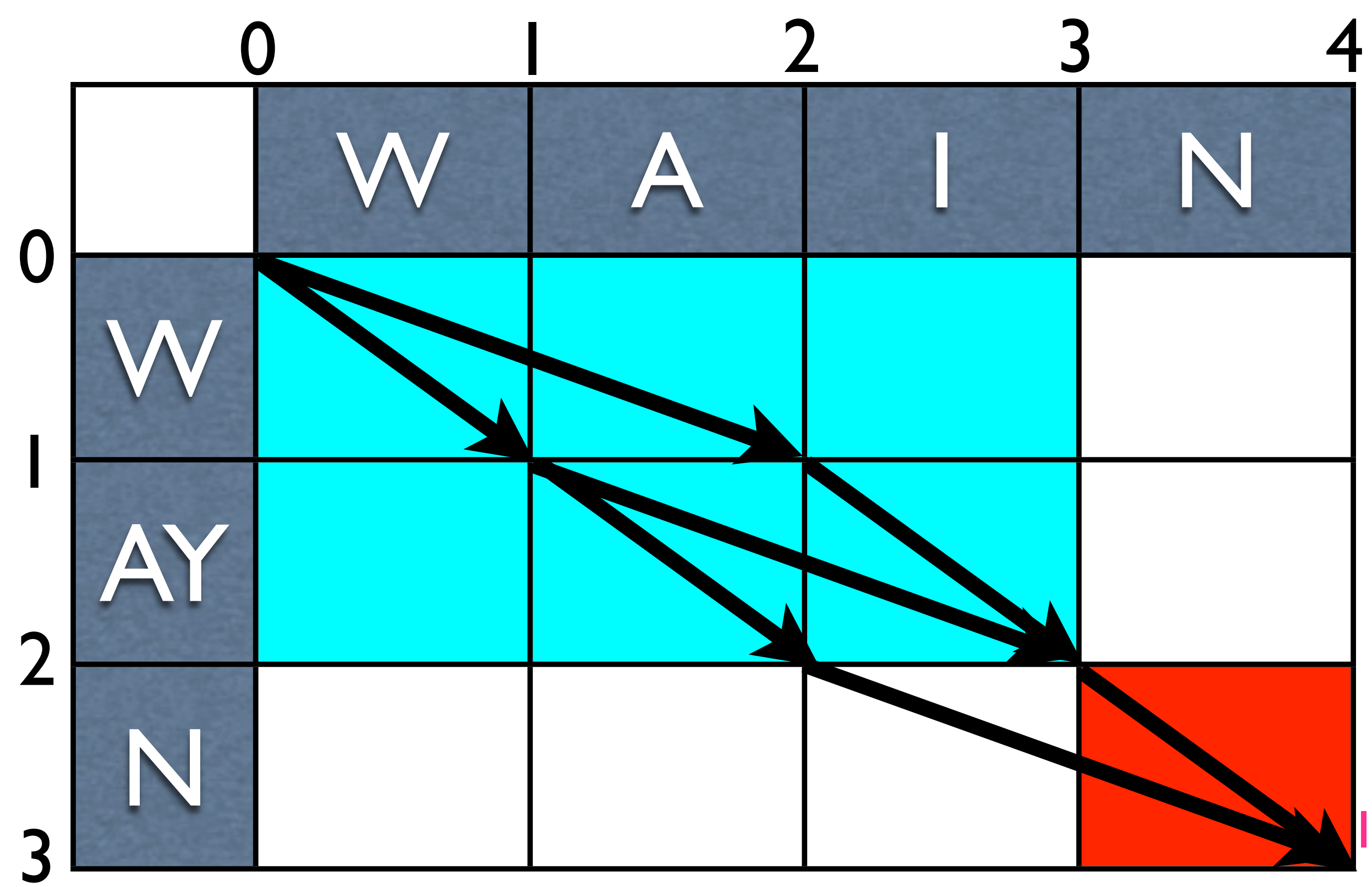
- for HW5. this example shows forward only.

```
n, m = len(eprons), len(jprons)
forward[0][0] = 1

for i in xrange(0, n):
    epron = eprons[i]
    for j in forward[i]:
        for k in range(1, min(m-j, 3)+1):
            jseg = tuple(jprons[j:j+k])
            score = forward[i][j] * table[epron][jseg]
            forward[i+1][j+k] += score

totalprob *= forward[n][m]
```





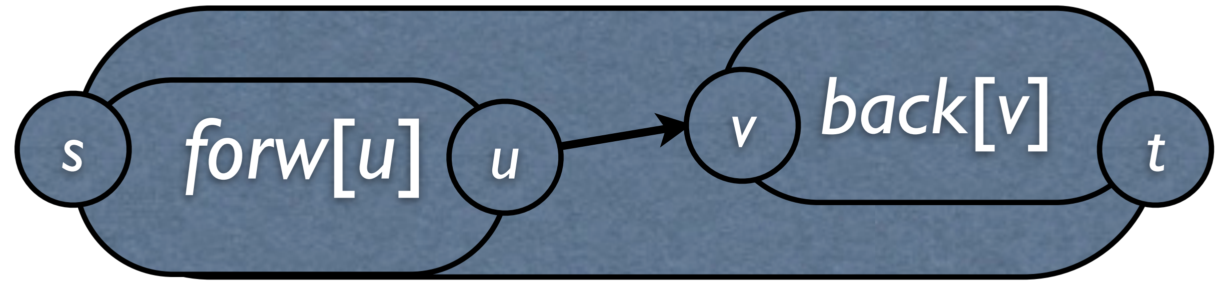
Example Forward Code

- for HW5. this example shows forward only.

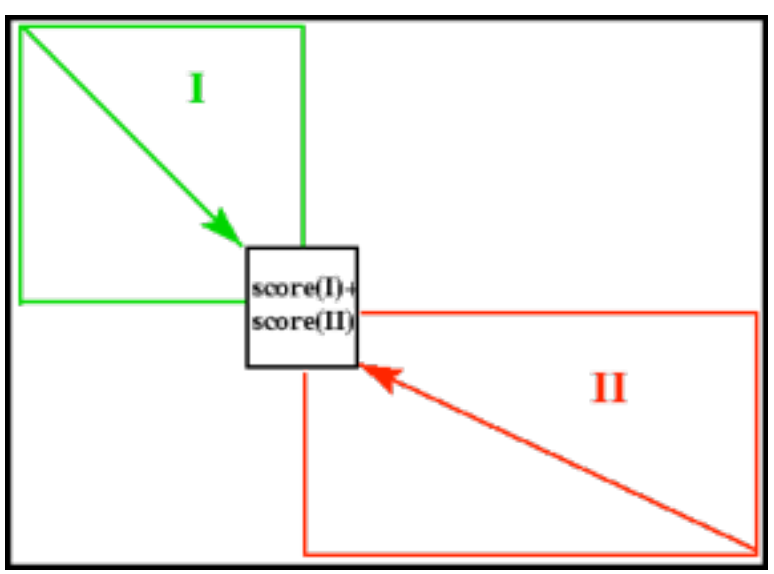
```
n, m = len(eprons), len(jprons)
forward[0][0] = 1

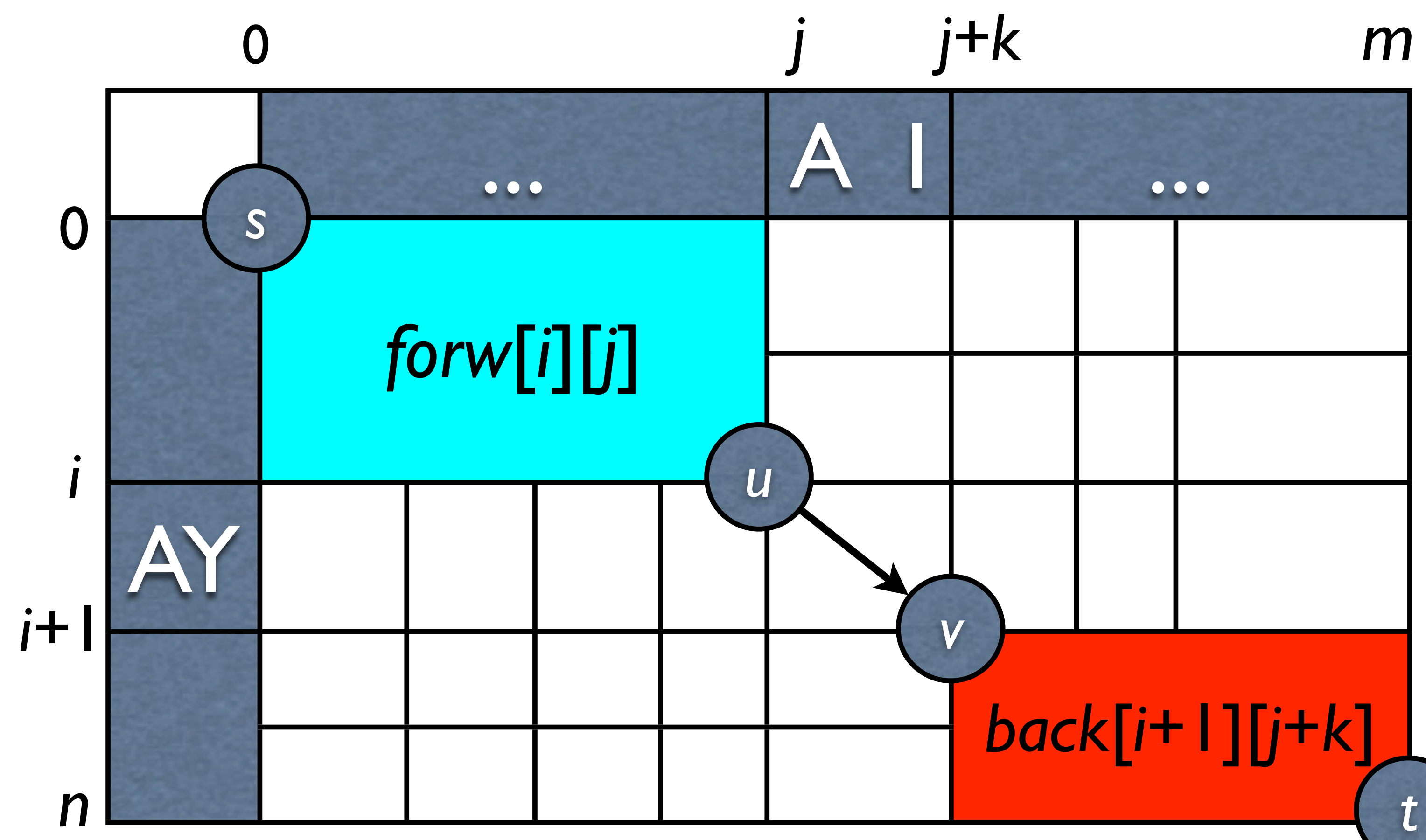
for i in xrange(0, n):
    epron = eprons[i]
    for j in forward[i]:
        for k in range(1, min(m-j, 3)+1):
            jseg = tuple(jprons[j:j+k])
            score = forward[i][j] * table[epron][jseg]
            forward[i+1][j+k] += score

totalprob *= forward[n][m]
```



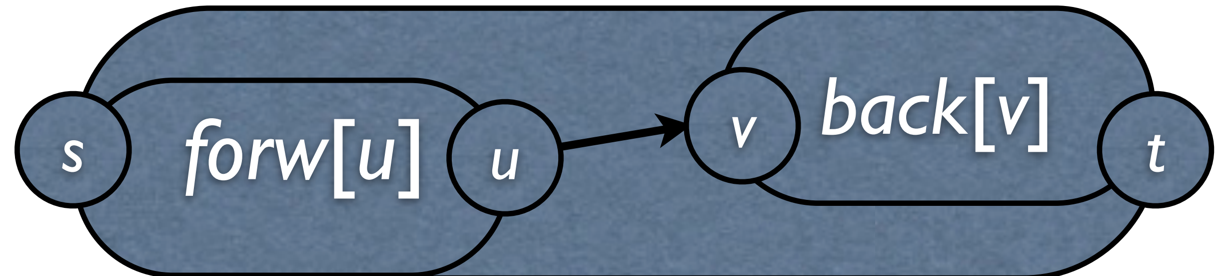
forw[*s*] = *back*[*t*] = 1.0
forw[*t*] = *back*[*s*] = *p*(*x*)





EM: fast version (DP)

- initialize the conditional prob. table to uniform
- repeat until converged:



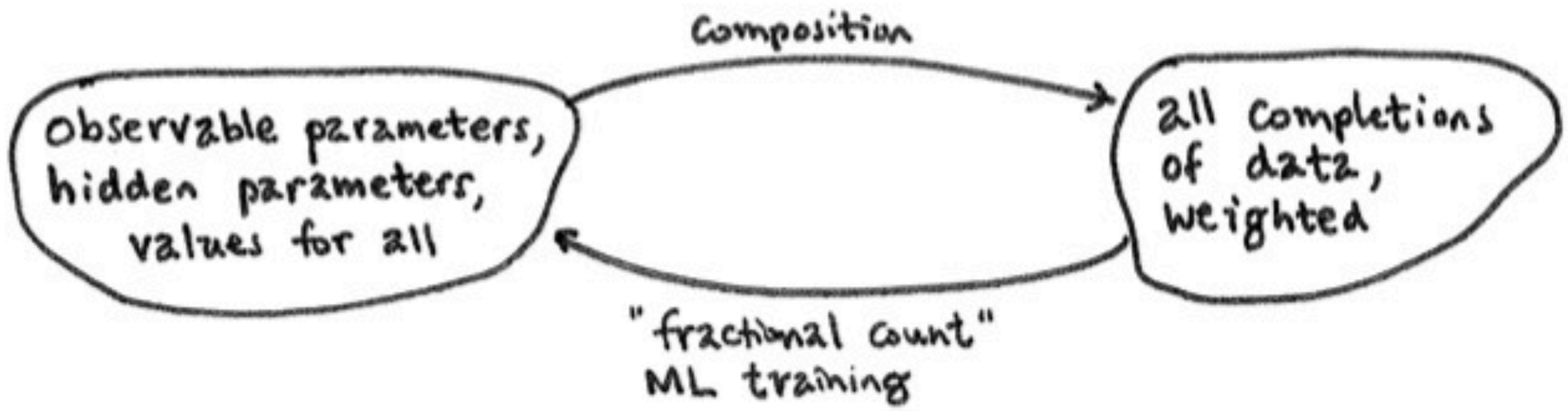
forw[*t*] = *back*[*s*] = $p(x)$ = $\text{sum}_z\, p(x, z)$

  - E-step:
    - for each training example x (here: (e...e, j...j) pair):
      - forward from *s* to *t*; note: *forw*[*t*] = $p(x) = \text{sum}_z\, p(x, z)$
      - backward from *t* to *s*; note: *back*[*t*]=1; *back*[*s*] = *forw*[*t*]
      - for each edge (u, v) in the DP graph with *label*(u, v) = z_i
        - fraccount(z_i) += *forw*[*u*] * *back*[*v*] * *prob*(*u*, *v*) / $p(x)$
  - M-step: count-n-divide on fraccounts => new model

$\text{sum}_{z:\,(u,v)\text{ in } z}\; p(x, z)$

EM

EM



example: cryptanalysis

$x_1 \cdots x_n$	observed ciphertext	
$z_1 \cdots z_n$	hidden plaintext	
$b(z_j \mid z_k)$	source bigram probabilities	OBSERVABLE, FIXED
$t(x_j \mid z_k)$	channel substitution ("encoding") probs	HIDDEN
$P(x_1 \cdots x_n, z_1 \cdots z_n) = \prod_{i=1}^{n} b(z_i \mid z_{i-1}) \cdot t(x_i \mid z_i)$		GENERATIVE STORY
$P(x_1 \cdots x_n) = \sum_{z_1 \cdots z_n} \prod_{i=1}^{n} b(z_i \mid z_{i-1}) \cdot t(x_i \mid z_i)$		FORWARD PROCEDURE
$P(z_1 \cdots z_n \mid x_1 \cdots x_n) = \frac{P(x_1 \cdots x_n, z_1 \cdots z_n)}{P(x_1 \cdots x_n)}$		COND. PROB.

Why EM increases p(data) iteratively?

$$D = \log p(x;\theta) = \log \sum_z p(x,z;\theta)$$

Note that $\sum_z p(z|x;\theta_t) = 1$ and $p(z|x;\theta_t) \geq 0$ for all z. Therefore D is the logarithm of a weighted sum, so we can apply Jensen's inequality, which says $\log \sum_j w_j v_j \geq \sum_j w_j \log v_j$, given $\sum_j w_j = 1$ and each $w_j \geq 0$. Here, we let the sum range over the values z of Z, with the weight w_j being $p(z|x;\theta_t)$. We get

$$D \geq E = \sum_z p(z|x;\theta_t) \log \frac{p(x,z;\theta)}{p(z|x;\theta_t)}.$$

Separating the fraction inside the logarithm to obtain two sums gives

$$E = \Big(\sum_z p(z|x;\theta_t) \log p(x,z;\theta)\Big) - \Big(\sum_z p(z|x;\theta_t) \log p(z|x;\theta_t)\Big).$$

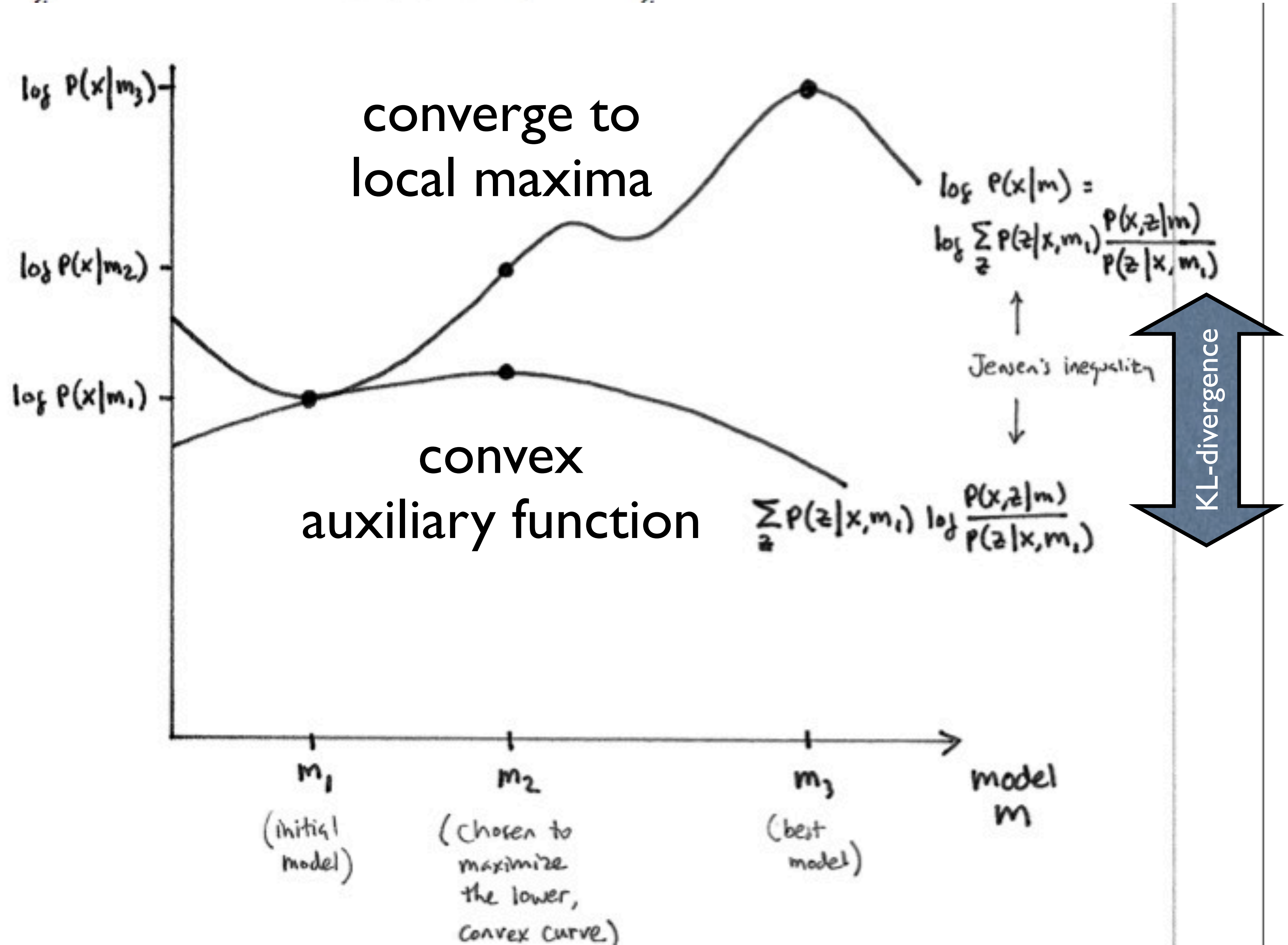
Since $E \leq D$ and we want to maximize D, consider maximizing E. The weights $p(z|x;\theta_t)$ do not depend on θ, so we only need to maximize the first sum, which is

$$\sum_z p(z|x;\theta_t) \log p(x,z;\theta).$$

Why EM increases p(data) iteratively?

How do we know that maximizing E actually leads to an improvement in the likelihood? With $\theta = \theta_t$,

$$E = \sum_z p(z|x;\theta_t) \log \frac{p(x,z;\theta_t)}{p(z|x;\theta_t)} = \sum_z p(z|x;\theta_t) \log p(x;\theta_t) = \log p(x;\theta_t)$$



How to maximize the auxiliary?

$$\sum_z p(z|x;\theta_t) \log p(x,z;\theta).$$

In general, the E-step of an EM algorithm is to compute $p(z|x;\theta_t)$ for all z. The M-step is then to find θ to maximize $\sum_z p(z|x;\theta_t) \log p(x,z;\theta)$.

```
W  AI  N        W  AI  N        W  AI  N
|  |  /\        |  |\  \        |\  \  \
W  A  I  N      W  A  I  N      W  A  I  N
```

p(z|x)=0.5 *p(z'|x)=0.3* *p(z''|x)=0.2*

just count-n-divide on the fractional data!

(as if MLE on complete data)

```
W  AI  N        W  AI  N        W  AI  N
|  |  /\        |  |\  \        |\  \  \
W  A  I  N      W  A  I  N      W  A  I  N
```

5x *3x* *2x*