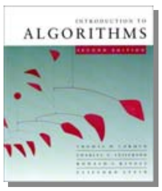
Introduction to Algorithms

6.046J/18.401J



LECTURE 17

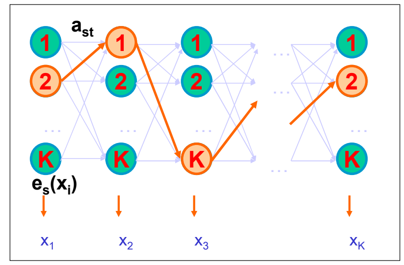
Hidden Markov Models II

- Review:
  - Path $P(x,\pi) = a_{0\pi_1} \prod_i e_{\pi_i}(x_i)\, a_{\pi_i \pi_{i+1}}$.
  - Viterbi $\pi^* = \text{argmax}_\pi\, P(x,\pi)$.
  - Forward $P(x) = \text{sum}_\pi\, P(x,\pi)$.
- Posterior decoding $P(\pi_i = k \mid x)$.
- Supervised learning $\max_\Lambda P(x,\pi|\Lambda)$.
- Unsupervised learning $\max_\Lambda P(x|\Lambda)$.

Prof. Manolis Kellis

April 10, 2008

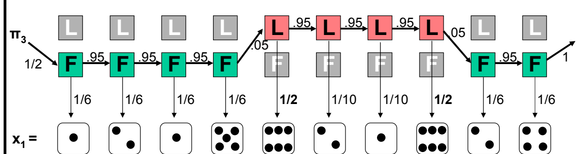
Notation: path π, emissions *x*, state *k*, time *i*.



Probability of observing emissions x with known path π.

- $P(x,\pi) = \underbrace{a_{0\pi_1}}_{\text{start}} * \prod_i \underbrace{e_{\pi_i}(x_i)}_{\text{emission}} \times \underbrace{a_{\pi_i \pi_{i+1}}}_{\text{transition}}$

Example: the dishonest casino



$P(x_1,\pi_3)$ = P(transition) * P(emission) * P(transition) * P(emission) * …

= ½ × P(1 | Fair) P(Fair$_{i+1}$ | Fair$_i$) P(2 | Fair) P(Fair | Fair) … P(4 | Fair)

= ½ × $(1/10)^2 \times (1/2)^2 \times (1/6)^6 \times (0.95)^7 \times (0.05)^2$

= 4.6×10^{-11} (least likely)

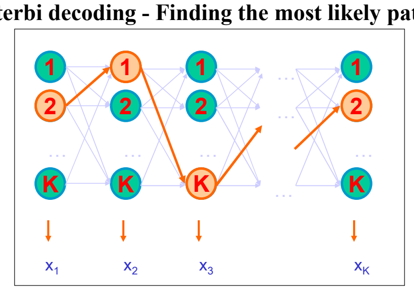
$P(x_1$,all-loaded$) = 7.9 \times 10^{-10}$ (more likely)

$P(x_1$,all-fair$) = 0.5 \times 10^{-9}$ (most likely)

	One path	All paths
Scoring	1. Scoring x, one path $P(x,\pi)$ Prob of a path, emissions	2. Scoring x, all paths $P(x) = \Sigma_\pi P(x,\pi)$ Prob of emissions, over all paths
Decoding	3. Viterbi decoding $\pi^* = \text{argmax}_\pi P(x,\pi)$ Most likely path	4. Posterior decoding $\pi^\wedge = \{\pi_i \mid \pi_i = \text{argmax}_k \Sigma_\pi P(\pi_i = k \mid x)\}$ Path containing the most likely state at any time point.
Learning	5. Supervised learning, given π $\Lambda^* = \text{argmax}_\Lambda P(x,\pi\|\Lambda)$ 6. Unsupervised learning. $\Lambda^* = \text{argmax}_\Lambda \max_\pi P(x,\pi\|\Lambda)$ Viterbi training, best path	6. Unsupervised learning $\Lambda^* = \text{argmax}_\Lambda \Sigma_\pi P(x,\pi\|\Lambda)$ Baum-Welch training, over all paths

---

Viterbi decoding - Finding the most likely path

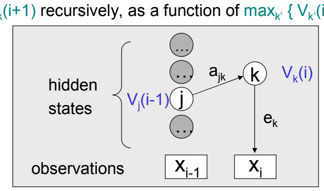


- Find path π^* that maximizes total joint probability $P(x, \pi)$
- $\pi^* = \text{argmax}\ P(x,\pi) = \underbrace{a_{0\pi_1}}_{\text{start}} * \prod_i \underbrace{e_{\pi_i}(x_i)}_{\text{emission}} \times \underbrace{a_{\pi_i \pi_{i+1}}}_{\text{transition}}$

Viterbi algorithm: Calculate $\max P(x,\pi)$ recursively

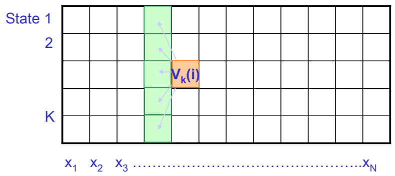
Define $V_k(i)$ = Probability of the most likely path through state $\pi_i = k$.

Compute $V_k(i+1)$ recursively, as a function of $\max_{k'} \{ V_{k'}(i) \}$.



- Assume we know all V_j values for previous time step i-1.
- Calculate $\underbrace{V_k(i)}_{\text{current max}} = \underbrace{e_k(x_i)}_{\text{this emission}} * \underbrace{\max_j \big(\underbrace{V_j(i-1)}_{\text{max ending in state j at step i}} \times \underbrace{a_{jk}}_{\text{Transition from state j}} \big)}_{\text{all possible previous states j}}$

The Viterbi Algorithm



Input: x = x1……xN

Initialization:
$V_0(0)=1$, $V_k(0) = 0$, for all k > 0

Iteration:
$V_k(i) = e_K(x_i) \times \max_j a_{jk} V_j(i-1)$

Termination:
$P(x, \pi^*) = \max_k V_k(N)$

Traceback:
Follow max pointers back

In practice:
Use log scores for computation

Running time and space:
Time: $O(K^2N)$
Space: $O(KN)$

Derivation of Viterbi algorithm

Let $V_k(i) = \max_{\{\pi 1,\ldots,i-1\}} P[x_1 \ldots x_{i-1}, \pi_1 \ldots \pi_{i-1}, x_i, \pi_i = k]$
= Probability of most likely sequence of states ending at state $\pi_i = k$

Calculate $V_k(i+1)$ recursively as a function of $V_j(i)$?

$V_l(i+1) = \max_{\{\pi 1,\ldots,i\}} P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i, x_{i+1}, \pi_{i+1} = l]$
from definition

$= \max_{\{\pi 1,\ldots,i\}} P(x_{i+1}, \pi_{i+1} = l \mid x_1 \ldots x_i, \pi_1, \ldots, \pi_i)\, P[x_1 \ldots x_i, \pi_1, \ldots, \pi_i]$
from Bayes's Law

$= \max_{\{\pi 1,\ldots,i\}} P(x_{i+1}, \pi_{i+1} = l \mid \pi_i)\, P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i]$
from Markov property (no memory)

$= \max_k P(x_{i+1}, \pi_{i+1} = l \mid \pi_i = k) \max_{\{\pi 1,\ldots,i-1\}} P[x_1 \ldots x_{i-1}, \pi_1, \ldots, \pi_{i-1}, x_i, \pi_i = k]$
from commutativity of multiplication, max

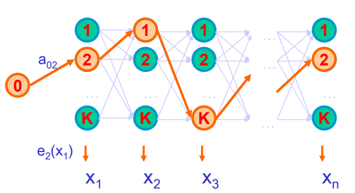
$= e_l(x_{i+1}) \max_k a_{kl} V_k(i)$
from recursive definition of V_k variable

	One path	All paths
Scoring	1. Scoring x, one path $P(x,\pi)$ Prob of a path, emissions	2. Scoring x, all paths $P(x) = \Sigma_\pi P(x,\pi)$ Prob of emissions, over all paths
Decoding	3. Viterbi decoding $\pi^* = \text{argmax}_\pi P(x,\pi)$ Most likely path	4. Posterior decoding $\pi\hat{} = \{\pi_i \mid \pi_i = \text{argmax}_k \Sigma_\pi P(\pi_i=k \mid x)\}$ Path containing the most likely state at any time point.
Learning	5. Supervised learning, given π $\Lambda^* = \text{argmax}_\Lambda P(x,\pi \mid \Lambda)$ 6. Unsupervised learning. $\Lambda^* = \text{argmax}_\Lambda \max_\pi P(x,\pi \mid \Lambda)$ Viterbi training, best path	6. Unsupervised learning $\Lambda^* = \text{argmax}_\Lambda \Sigma_\pi P(x,\pi \mid \Lambda)$ Baum-Welch training, over all paths

2. Model evaluation:
Total P(x|M), summed over all paths

Forward algorithm

Simple: Given the model, generate some sequence x

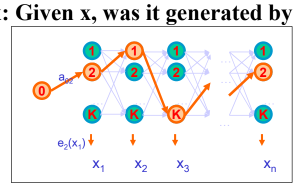


Given a HMM, we can generate a sequence of length n as follows:

1. Start at state π_1 according to prob $a_{0\pi 1}$
2. Emit letter x_1 according to prob $e_{\pi 1}(x_1)$
3. Go to state π_2 according to prob $a_{\pi 1 \pi 2}$
4. … until emitting x_n

We have some sequence x that can be emitted by p. Can calculate its likelihood.
However, in general, many different paths may emit this same sequence x.
How do we find the total probability of generating a given x, over any path?

Complex: Given x, was it generated by the model?

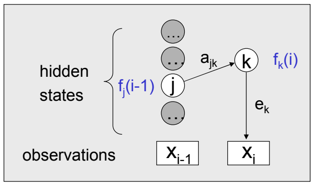


Given a sequence x,
What is the probability that x was generated by the model (using any path)?

– $P(x) = \Sigma_\pi P(x,\pi)$

- Challenge: exponential number of paths
- (cheap) alternative:
  – Calculate probability over maximum (Viterbi) path π^*
- (real) solution
  – Calculate sum iteratively using principles of dynamic programming

Calculate total probability $\Sigma_\pi P(x,\pi)$ recursively



- Assume we know f_j for the previous time step (i-1)
- Calculate $f_k(i)$ = $e_k(x_i)$ * $\underbrace{\text{sum}_j\,(\ f_j(i-1) \times a_{jk}\ )}_{\text{every possible previous state j}}$

current max · this emission · sum ending in state j at step i · transition from state j

every possible previous state j

The Forward Algorithm – derivation

Define the forward probability:

$f_l(i) = P(x_1\ldots x_i, \pi_i = l)$

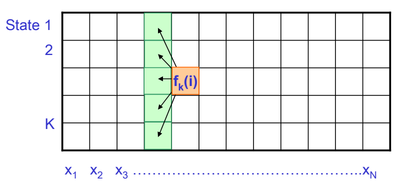
$= \boldsymbol{\Sigma}_{\pi 1\ldots\pi i-1} P(x_1\ldots x_{i-1}, \boldsymbol{\pi_1\ldots \pi_{i-2}, \pi_{i-1}}, \pi_i = l)\ \mathbf{e_l(x_i)}$

$= \boldsymbol{\Sigma_k}\ \Sigma_{\pi 1\ldots\pi i-2} \boxed{P(x_1\ldots x_{i-1}, \pi_1,\ldots, \pi_{i-2}, \boldsymbol{\pi_{i-1}=k})}\ \ \mathbf{a_{kl}}\ e_l(x_i)$

$= \Sigma_k \boxed{\mathbf{f_k(i-1)}}\ a_{kl}\ e_l(x_i)$

$= e_l(x_i)\ \Sigma_k\ f_k(i-1)\ a_{kl}$

The Forward Algorithm



Input: x = x1……xN

Initialization:
$f_0(0)=1$, $f_k(0) = 0$, for all k > 0

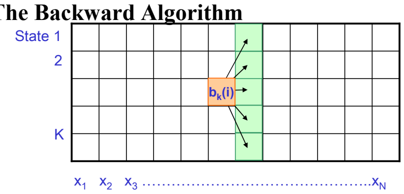
Iteration:
$f_k(i) = e_K(x_i) \times \text{sum}_j\ a_{jk}\ f_j(i-1)$

Termination:
$P(x, \pi^*) = \text{sum}_k\ f_k(N)$

In practice:
Sum of log scores is difficult
→ approximate exp(1+p+q)
→ scaling of probabilities

Running time and space:
Time: $O(K^2N)$
Space: $O(KN)$

The Backward Algorithm



Input: x = x1……xN

Initialization:
$b_k(N) = a_{k0}$, for all k

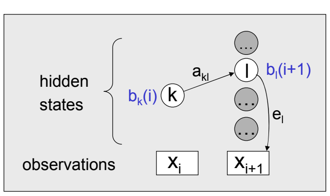
Iteration:
$b_k(i) = \Sigma_l\ e_l(x_{i+1})\ a_{kl}\ b_l(i+1)$

Termination:
$P(x) = \Sigma_l\ a_{0l}\ e_l(x_1)\ b_l(1)$

In practice:
Sum of log scores is difficult
→ approximate exp(1+p+q)
→ scaling of probabilities

Running time and space:
Time: $O(K^2N)$
Space: $O(KN)$

Calculate total end probability recursively



- Assume we know b_l for the next time step (i+1)
- Calculate $b_k(i)$ = $\underbrace{\text{sum}_l\,(\ e_l(x_{i+1}) \times a_{kl} \times b_l(i+1)\ )}_{\text{sum over all possible next states}}$

current max · next emission · transition to next state · prob sum from state l to end

sum over all possible next states

The Backward Algorithm – derivation

Define the backward probability:

$b_k(i) = P(x_{i+1}\ldots x_N \mid \pi_i = k)$

$= \Sigma_{\pi i+1\ldots\pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k)$

$= \Sigma_l\ \Sigma_{\pi i+1\ldots\pi N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1} = l, \pi_{i+2}, \ldots, \pi_N \mid \pi_i = k)$

$= \Sigma_l\ e_l(x_{i+1})\ a_{kl} \boxed{\Sigma_{\pi i+1\ldots\pi N} P(x_{i+2}, \ldots, x_N, \pi_{i+2}, \ldots, \pi_N \mid \pi_{i+1} = l)}$

$= \Sigma_l\ e_l(x_{i+1})\ a_{kl} \boxed{b_l(i+1)}$

4. Decoding, all paths

Find the likelihood an emission x_i is generated by a state

Calculate P(π_7= CpG+ | x_7=G)

- With no knowledge (no characters)
  - P(π_i=k) = most likely state (**prior**)
  - Time spent in markov chain states
- With very little knowledge (just that character)
  - P(π_i=k | x_i=G) = (prior) * (most likely emission)
  - Emission probabilities adjusted for time spent
- With knowledge of entire sequence (all characters)
  - P(π_i=k | x=AGCGCG…GATTATCGTCGTA)
  - Sum over all paths that emit 'G' at position 7
  - ➔ **Posterior** decoding

Combining the Forward and Backward algorithms

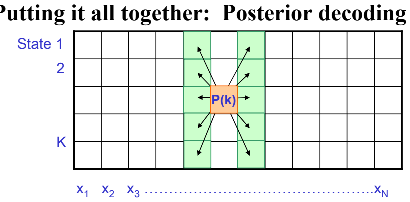
We want to compute:

$P(\pi_i = k \mid x)$, the probability distribution on the i^{th} position, given x

We can expand it into its forward and backward components:

$$\begin{aligned} P(\pi_i = k, x) &= P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N) \\ &= P(x_1 \ldots x_i, \pi_i = k)\, P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k) \\ &= \underbrace{P(x_1 \ldots x_i, \pi_i = k)}_{\text{Forward, } f_k(i)}\, \underbrace{P(x_{i+1} \ldots x_N \mid \pi_i = k)}_{\text{Backward, } b_k(i)} \end{aligned}$$

Putting it all together: Posterior decoding



- P(k) = P(π_i=k | x) = $f_k(i)$*$b_k(i)$ / P(x)
  - Probability that i^{th} state is k, given all emissions x
- Posterior decoding
  - Define most likely state for every of sequence x
  - $\hat{\pi}_i = \text{argmax}_k P(\pi_i = k \mid x)$
- Posterior decoding 'path' $\hat{\pi}_i$
  - For classification, more informative than Viterbi path π^*
    - More refined measure of "which hidden states" generated x
  - However, it may give an invalid sequence of states
    - Not all j➔k transitions may be possible

Summary this far

- Generative model. Hidden states, observed emissions.
  - Generate a random sequence
    - Choose random transition, choose random emission (#0)
- Scoring the likelihood of a sequence
  - Calculate likelihood of annotated path and sequence
    - Multiply emission and transition probabilities (#1)
  - Without specifying a path, total probability of generating x
    - Sum probabilities over all paths
    - Forward algorithm (#3)
- Decoding: Finding the most likely path, given a sequence
  - What is the most likely path generating entire sequence?
    - Viterbi algorithm (#2)
  - What is the most probable state at each time step?
    - Forward + backward algorithms, posterior decoding (#4)
- Next: Learning (#5 and #6)

	One path	**All paths**
Scoring	1. Scoring x, one path P(x,π) Prob of a path, emissions	2. Scoring x, all paths P(x) = Σ_π P(x,π) Prob of emissions, over all paths
Decoding	3. Viterbi decoding π^* = argmax$_\pi$ P(x,π) Most likely path	4. Posterior decoding $\hat{\pi} = \{\pi_i \mid \pi_i = \text{argmax}_k \Sigma_\pi P(\pi_i = k \mid x)\}$ Path containing the most likely state at any time point.
Learning	5. Supervised learning, given π Λ^* = argmax$_\Lambda$ P(x,π|Λ) 6. Unsupervised learning. Λ^* = argmax$_\Lambda$ max$_\pi$P(x,π|Λ) Viterbi training, best path	6. Unsupervised learning Λ^* = argmax$_\Lambda$ Σ_πP(x,π|Λ) Baum-Welch training, over all paths

Two types of learning: Supervised / Unsupervised

5. Supervised learning

infer model parameters given **labeled** training data

- GIVEN:
  - a HMM M, with unspecified transition/emission probs.
  - labeled sequence x,
- FIND:
  - parameters θ = (Ei, Aij) that maximize P[x,π | θ]

➔ Simply count frequency of each emission and transition, as observed in the training data

6. Unsupervised learning

infer model parameters given **unlabelled** training data

- GIVEN:
  - a HMM M, with unspecified transition/emission probs.
  - unlabeled sequence x,
- FIND:
  - parameters θ = (Ei, Aij) that maximize P[x | θ]

➔ Viterbi training:
guess parameters, find optimal Viterbi path (#2), update parameters (#5), iterate

➔ Baum-Welch training:
guess parameters, sum over all paths (#4), update parameters (#5), iterate

5: Supervised learning

Estimate model parameters
based on **labeled** training data

Two learning scenarios

Case 1. Estimation when the "right answer" is known

Examples:

GIVEN: a genomic region $x = x_1 \ldots x_{1,000,000}$ where we have good (experimental) annotations of the CpG islands

GIVEN: the casino player allows us to observe him one evening, as he changes dice and produces 10,000 rolls

Case 2. Estimation when the "right answer" is unknown

Examples:

GIVEN: the porcupine genome; we don't know how frequent are the CpG islands there, neither do we know their composition

GIVEN: 10,000 rolls of the casino player, but we don't see when he changes dice

QUESTION: Update the parameters θ of the model to maximize P(x|θ)

Case 1. When the right answer is known

Given $x = x_1 \ldots x_N$
for which the true $\pi = \pi_1 \ldots \pi_N$ is known,

Define:

A_{kl} = # times k→l transition occurs in π
$E_k(b)$ = # times state k in π emits b in x

We can show that the maximum likelihood parameters θ are:

$$a_{kl} = \frac{A_{kl}}{\Sigma_i \, A_{ki}} \qquad e_k(b) = \frac{E_k(b)}{\Sigma_c \, E_k(c)}$$

Case 1. When the right answer is known

Intuition: When we know the underlying states,
Best estimate is the average frequency of transitions & emissions that occur in the training data

Drawback:
Given little data, there may be **overfitting**:
P(x|θ) is maximized, but θ is unreasonable
0 probabilities – VERY BAD

Example:
Given 10 casino rolls, we observe

```
x = 2, 1, 5, 6, 1, 2, 3, 6, 2, 3
π = F, F, F, F, F, F, F, F, F, F
```

Then:
$a_{FF} = 1$; $a_{FL} = 0$
$e_F(1) = e_F(3) = .2$;
$e_F(2) = .3$; $e_F(4) = 0$; $e_F(5) = e_F(6) = .1$

Pseudocounts

Solution for small training sets:

Add pseudocounts

A_{kl} = # times k→l transition occurs in π + r_{kl}
$E_k(b)$ = # times state k in π emits b in x + $r_k(b)$

r_{kl}, $r_k(b)$ are pseudocounts representing our prior belief

Larger pseudocounts ⇒ Strong priof belief

Small pseudocounts ($\varepsilon < 1$): just to avoid 0 probabilities

Pseudocounts

Example: dishonest casino

We will observe player for one day, 500 rolls

Reasonable pseudocounts:

$r_{0F} = r_{0L} = r_{F0} = r_{L0} = 1$;
$r_{FL} = r_{LF} = r_{FF} = r_{LL} = 1$;
$r_F(1) = r_F(2) = \ldots = r_F(6) = 20$ (strong belief fair is fair)
$r_F(1) = r_F(2) = \ldots = r_F(6) = 5$ (wait and see for loaded)

Above #s pretty arbitrary – assigning priors is an art

6: Unsupervised learning

Estimate model parameters
based on **unlabeled** training data

Learning case 2. When the right answer is unknown

We don't know the true A_{kl}, $E_k(b)$

Idea:

- We estimate our "best guess" on what A_{kl}, $E_k(b)$ are
- We update the parameters of the model, based on our guess
- We repeat

Case 2. When the right answer is unknown

Starting with our best guess of a model M, parameters θ:

Given $x = x_1 \ldots x_N$
for which the true $\pi = \pi_1 \ldots \pi_N$ is unknown,

We can get to a provably more likely parameter set θ

Principle: EXPECTATION MAXIMIZATION

1. Estimate A_{kl}, $E_k(b)$ in the training data
2. Update θ according to A_{kl}, $E_k(b)$
3. Repeat 1 & 2, until convergence

Estimating new parameters

To estimate $\mathbf{A_{kl}}$:

At each position i of sequence x,

Find probability transition $k \rightarrow l$ is used:

$P(\pi_i = k, \pi_{i+1} = l \mid x) = [1/P(x)] \times P(\pi_i = k, \pi_{i+1} = l, x_1 \ldots x_N) = Q/P(x)$

where $Q = P(x_1 \ldots x_i, \pi_i = k, \pi_{i+1} = l, x_{i+1} \ldots x_N) =$
$= P(\pi_{i+1} = l, x_{i+1} \ldots x_N \mid \pi_i = k)\, P(x_1 \ldots x_i, \pi_i = k) =$
$= P(\pi_{i+1} = l, x_{i+1} x_{i+2} \ldots x_N \mid \pi_i = k)\, f_k(i) =$
$= P(x_{i+2} \ldots x_N \mid \pi_{i+1} = l)\, P(x_{i+1} \mid \pi_{i+1} = l)\, P(\pi_{i+1} = l \mid \pi_i = k)\, f_k(i) =$
$= b_l(i+1)\, e_l(x_{i+1})\, a_{kl}\, f_k(i)$

So: $$\mathbf{P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \frac{f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1)}{P(x \mid \theta)}}$$

(For one such transition, at time step $i \rightarrow i+1$)

Estimating new parameters

(Sum over all $k \rightarrow l$ transitions, at any time step i)

So,

$$A_{kl} = \sum_i P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \sum_i \frac{f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1)}{P(x \mid \theta)}$$

Similarly,

$$E_k(b) = [1/P(x)] \sum_{\{i \mid xi = b\}} f_k(i)\, b_k(i)$$

Estimating new parameters

(Sum over all training seqs, all k→l transitions, all time steps i)

If we have several training sequences, $x^1, \ldots, x^M$, each of length N,

$$A_{kl} = \Sigma_x \, \Sigma_i \, P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = \Sigma_x \, \Sigma_i \frac{f_k(i) \, a_{kl} \, e_l(x_{i+1}) \, b_l(i+1)}{P(x \mid \theta)}$$

Similarly,

$$E_k(b) = \Sigma_x \, (1/P(x)) \Sigma_{\{i \mid x_i = b\}} \, f_k(i) \, b_k(i)$$

The Baum-Welch Algorithm

Initialization:

Pick the best-guess for model parameters (or arbitrary)

Iteration:

1. Forward
2. Backward
3. Calculate A_{kl}, $E_k(b)$
4. Calculate new model parameters a_{kl}, $e_k(b)$
5. Calculate new log-likelihood $P(x \mid \theta)$

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until $P(x \mid \theta)$ does not change much

The Baum-Welch Algorithm – comments

Time Complexity:

\# iterations × $O(K^2N)$

- Guaranteed to increase the log likelihood of the model

  $P(\theta \mid x) = P(x, \theta) / P(x) = P(x \mid \theta) / (P(x) \, P(\theta))$

- Not guaranteed to find globally best parameters

  Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overtraining

Alternative: Viterbi Training

Initialization: Same

Iteration:

1. Perform Viterbi, to find π^*
2. Calculate A_{kl}, $E_k(b)$ according to π^* + pseudocounts
3. Calculate the new parameters a_{kl}, $e_k(b)$

Until convergence

Notes:

– Convergence is guaranteed – Why?
– Does not maximize $P(x \mid \theta)$
– In general, worse performance than Baum-Welch

	One path	**All paths**
Scoring	1. Scoring x, one path $P(x,\pi)$ Prob of a path, emissions	2. Scoring x, all paths $P(x) = \Sigma_\pi P(x,\pi)$ Prob of emissions, over all paths
Decoding	3. Viterbi decoding $\pi^* = \text{argmax}_\pi P(x,\pi)$ Most likely path	4. Posterior decoding $\pi\^{} = \{\pi_i \mid \pi_i = \text{argmax}_k \Sigma_\pi P(\pi_i = k \mid x)\}$ Path containing the most likely state at any time point.
Learning	5. Supervised learning, given π $\Lambda^* = \text{argmax}_\Lambda P(x,\pi \mid \Lambda)$ 6. Unsupervised learning. $\Lambda^* = \text{argmax}_\Lambda \max_\pi P(x,\pi \mid \Lambda)$ Viterbi training, best path	6. Unsupervised learning $\Lambda^* = \text{argmax}_\Lambda \Sigma_\pi P(x,\pi \mid \Lambda)$ Baum-Welch training, over all paths

What have we learned ?

- Generative model. Hidden states, observed emissions.
  - Generate a random sequence
    - Choose random transition, choose random emission (#0)
- Scoring: Finding the likelihood of a given sequence
  - Calculate likelihood of annotated path and sequence
    - Multiply emission and transition probabilities (#1)
  - Without specifying a path, total probability of generating x
    - Sum probabilities over all paths
    - Forward algorithm (#3)
- Decoding: Finding the most likely path, given a sequence
  - What is the most likely path generating entire sequence?
    - Viterbi algorithm (#2)
  - What is the most probable state at each time step?
    - Forward + backward algorithms, posterior decoding (#4)
- Learning: Estimating HMM parameters from training data
  - When state sequence is known
    - Simply compute maximum likelihood A and E (#5)
  - When state sequence is not known
    - Baum-Welch: Iterative estimation of all paths / frequencies (#6a)
    - Viterbi training: Iterative estimation of best path / frequencies (#6b)

The main questions on HMMs

SCORING

1. Scoring x, one path = Joint probability of a sequence and a path, given the model
- GIVEN a HMM M, a path π, and a sequence x,
- FIND Prob[x, π | M]
- ➔ "Running the model", simply multiply emission and transition probabilities
- ➔ Application: "all fair" vs. "all loaded" comparisons

2. Scoring x, all paths = total probability of a sequence, summed across all paths
- GIVEN a HMM M, a sequence x
- FIND the total probability P[x | M] summed across all paths
- ➔ Forward algorithm, sum score over all paths (same result as backward)

PARSING

3. Viterbi decoding = parsing a sequence into the optimal series of hidden states
- GIVEN a HMM M, and a sequence x,
- FIND the sequence π^* of states that maximizes P[x, π | M]
- ➔ Viterbi algorithm, dynamic programming, max score over all paths, trace pointers find path

4. Posterior decoding = total prob that emission x_i came from state k, across all paths
- GIVEN a HMM M, a sequence x
- FIND the total probability P[π_i = k | x, M)
- ➔ Posterior decoding: run forward & backward algorithms to & from state π_i =k

LEARNING

5. Supervised learning = optimize parameters of a model given training data
- GIVEN a HMM M, with unspecified transition/emission probs., labeled sequence x,
- FIND parameters $\theta = (e_i, a_{ij})$ that maximize P[x | θ]
- ➔ Simply count frequency of each emission and transition observed in the training data

6. Unsupervised learning = optimize parameters of a model given training data
- GIVEN a HMM M, with unspecified transition/emission probs., unlabeled sequence x,
- FIND parameters $\theta = (e_i, a_{ij})$ that maximize P[x | θ]
- ➔ Viterbi training: guess parameters, find optimal Viterbi path (#2), update parameters (#5), iterate
- ➔ Baum-Welch training: guess, sum over all emissions/transitions (#4), update (#5), iterate

	One path	**All paths**
Scoring	1. Scoring x, one path $P(x,\pi)$ Prob of a path, emissions	2. Scoring x, all paths $P(x) = \Sigma_\pi P(x,\pi)$ Prob of emissions, over all paths
Decoding	3. Viterbi decoding $\pi^* = \text{argmax}_\pi P(x,\pi)$ Most likely path	4. Posterior decoding $\pi\hat{} = \{\pi_i \mid \pi_i = \text{argmax}_k \Sigma_\pi P(\pi_i=k \mid x)\}$ Path containing the most likely state at any time point.