Training

- EM can be used to learn HMM parameters from just a corpus of observation sequences.
- but as in earlier IBM models case, brute-force EM is not computationally feasible.
- Baum-Welch is name of a clever efficient implementation for HMMs which makes it feasible.
- by now, a brute-force EM algorithm for HMMs should be clear: the hidden variable will be the state sequence s. Need to repeatedly (1) calculate \( P(s|o) \) and use as 'count' for the completion \( s,o \), deriving 'expected counts' of aspects of \( s,o \) eg. \( i \) then \( j \) (2) recalculate parameters \( \pi, A \) and \( B \) from these

\[
\text{until a fixed-point \{ enumerate all state seqs for all observations use current } (\pi_i, A_i, B_i) \text{ to generate expected counts from expected counts generate new } (\pi_{i+1}, A_{i+1}, B_{i+1}) \}
\]
Brute force EM for HMMs in words

- Suppose initial HMM \((\pi, A, B)\) and corpus of \(D\) obs. seqs. \(o^1 \ldots o^D\). For each observation sequence \(o^d\), there is \(P(s|o^d)\) (aka \(\gamma^d(s)\)) for every state sequence \(s\). Treat \(\gamma^d(s)\) as a ‘virtual count’ of the completion \((s, o^d)\).

- From this derive expected counts for various possibilities
- eg. expected count for state \(i\): \(E^d(i)\).
  this will be \(N \times \gamma^d(s)\), where \(N\) is freq of \(i\) in \(s\).
- Sim. there will be an expected count for the occurrence of \(ij\) in \(s\) \((E^d(ij))\), for occurrence of state \(i\) paired with obs \(k\) \((E^d(i; k))\), and for the occurrence of \(i\) as start state \((E^d(start = i))\).

- From these \(E^d\) numbers, probabilities can be re-estimated eg. the transition probability \(a_{ij}\):

  \[ a_{ij} = \sum_d E^d(ij) / \sum_d E^d(i) \]

  iterate over the corpus till fixed-point is reached.

- Can see Labs/em_for_hmm_worked_egs/brute_force_eg_slides.pdf to see one iteration

Avoiding the brute force cost (outline)

- can’t enumerate all possible seqs, because exponentially many: order \(N^T\)
- this problem is cleverly solved by splitting the overall expectations on one observation sequence \(o^d\) into \(T\) individual clock-tick versions \(^3\)
- instead of the responsibility \(\gamma^d(s) = p(s|o^d)\) work with ‘clock tick’ responsibilities like \(\gamma^d_t(i) = p(s_t = i|o^d)\), for prob of being in state \(i\) at clock tick \(t\) (given \(o^d\)).

- Summing over all clock ticks, this gives the necessary expected counts eg \(E^d(i)\).

- These clock-tick versions are determined by a very ingenious bit of dynamic programming, which factorises the calculations into two functions, \(\alpha\) and \(\beta\):
  \(\alpha\) has already been seen; its algorithm goes forward in time. \(\beta\) is a related function whose algorithm computes backwards in time.
  At each time tick, these two functions can be combined to give a joint probability of the observations and an event at that time tick.

Consider the just the calc of the \(E^d(i)\) – expected count of state \(i\) (if \(o\) is \(d^{th}\) obs sequence). Could be written like this being explicit about positions \(t\)

\[
\text{for each } o^d \text{ calculate } p(s|o^d) \\
\text{for each } t \in 1: \text{len}(o^d) \\
E^d(i) := p(s(o^d)) \text{ if } s_t = i
\]

- Consider a particular \(t\). As you go through all possible \(s\) for \(o^d\), whenever \(s\) has \(s_t = i\), you get an increment \(p(s|o^d)\) to \(E^d(i)\) due to that position.
  Can define position-specific expectations \(E^d_t(i)\) and sum these to get \(E^d(i)\)

  \[
  \sum_{s \text{ with } s_t = i} p(s|o^d)
  \] (3)

This EM re-estimation has the usual properties

- the observations \(o^1 \ldots o^D\) get likelier: if \(\theta^n\) and \(\theta^{n+1}\) are parameter settings at successive iterations:

  \[ P(o^1 \ldots o^D; \theta^n) \leq P(o^1 \ldots o^D; \theta^{n+1}) \]

  (this increasing quantity is the 'total observation probability', and is summing over all the state-sequences 'hiding' behind the observations i.e. the standard EM thing)

- there may be many local maxima

But biggest issue is

- prohibitively expensive to compute expectations via an enumeration of possible state sequences; Baum-Welch is clever optimised implementation of this brute-force EM algorithm, avoiding exponential cost.
This is just \( p(s_t = i | o^t) \)

for \( s \) position \( t \), \( s_t \) is fixed \( i \). For every other \( s \) position \( t' \), \( s_{t'} \) can be anything between 1 and \( N \), hence

\[
\sum_{s \mid s_t = i} p(s|o^t) = \sum_{s_t=1}^{N} \sum_{s_{t'-1}=1}^{N} \sum_{s_{t''}=1}^{N} \sum_{s_{t'''}=1}^{N} \left[ p(s_t \ldots s_{t'-1}, s_t = i, s_{t+1} \ldots s_T, o^t) \right] / p(o^t)
\]

we can put the summation all on top, and we get

\[
\frac{p(s_t = i, o^t)}{p(o^t)} = p(s_t = i | o^t)
\]

\[
= (\text{defn}) \gamma^t(i)
\]

So if can calculate \( \gamma^t(i) \) quickly, can find \( E^t(i) \) efficiently – its \( \sum_t(\gamma^t(i)) \). Note the denominator \( p(o^t) \) can be obtained by the forward algorithm. The numerator is similar but not the same as \( \alpha_t(i) = p(s_t = i, o_{i+1}^T) \) ... 

\( \alpha \), \( \beta \) and \( \alpha \beta \)

\[
\alpha_t(i) = \text{the joint probability of being in state } i \text{ at time } t \text{ and emitting the observation symbols } o_{1:t}
\]

\[
= P(o_{1:t}, s_t = i)
\]

\[
\beta_t(i) = \text{the conditional probability of emitting observation symbols } o_{t+1:T}, \text{ given being in state } i \text{ at time } t
\]

\[
= P(o_{t+1:T} | o_{1:t}, s_t = i)
\]

\[
\alpha_t(i)\beta_t(i) \quad \text{the joint probability of emitting observations symbols } o_{1:T} \text{ and being in state } i \text{ at time } t
\]

\[
= P(o_{1:T}, s_t = i, o_{t+1:T})
\]

now for the seriously clever bit ...

\( \alpha \) and \( \beta \) in pictures

[Diagram showing \( \alpha \) and \( \beta \) calculations]

\( \alpha_t(i) \) is the joint probability of being in state \( i \) at time \( t \) and emitting the observation symbols \( o_{1:t} \)

\[
= P(o_{1:t}, s_t = i)
\]

\( \beta_t(i) \) is the conditional probability of emitting observation symbols \( o_{t+1:T} \), given being in state \( i \) at time \( t \)

\[
= P(o_{t+1:T} | o_{1:t}, s_t = i)
\]

Now about that multiplication

Why does \( \alpha_t(i)\beta_t(i) \) give \( P(o_{1:T}, s_t = i, o_{t+1:T} \ldots o_T) \)?

\( \beta_t(i) \) is defined to be \( P(o_{t+1:T} | o_{1:t}, s_t = i) \)

But the independence properties of HMMs mean this is also equal to the seemingly more complex

\[
P(o_{t+1:T}, o_T | o_{1:t}, s_t = i)
\]

and applying the defn of cond. prob this means:

\[
\beta_t(i) = \frac{P(o_{t+1:T}, o_T | o_{1:t}, s_t = i)}{P(o_{1:t}, s_t = i)}
\]

The denominator of this is \( \alpha_t(i) = P(o_{1:t}, s_t = i) \).

so multiplying \( \alpha_t(i) \) and \( \beta_t(i) \) gives \( P(o_{1:T}, o_{t+1:T} \ldots o_T, s_t = i) \)
We will skip over the details of the incremental algorithm that calculates the \( \beta_t(.) \) quantities; it is an algorithm in a very similar style that for \( \alpha_t(.) \) except that it works backwards from \( t = T \) to \( t = 1 \).

The most crucial property is the fact that \( \alpha_t(.) \) and \( \beta_t(.) \) multiply together to give a joint probability involving a single state-at-a-time and all of \( o^d \). When divided by the total observation probability \( P(o^d) \) this gives the sought after \( p(s_t = i|o^d) \), from which then \( E^d(i) \) can be calculated.

We have shown some of the details how the state expectations \( E^d(i) \) can be efficiently calculated. Essentially for the other necessary expectations – \( E^d(ij), E^d(ik) \) etc, analogous manoeuvres are made, defining position-specific values which are obtainable easily once the \( \alpha_t(.) \) and \( \beta_t(.) \) quantities are known for all \( t \).

so the training requires the forward (\( \alpha \)) and backward (\( \beta \)) algorithms to be run for every training sequence \( o^d \). Each is linear wrt to len\( (o^d) \).

for the full details of the Baum Welch algorithm see slides/baum_welch_reference_slides.pdf and for a worked example see Labs/em_for_hmm_worked_egs/efficient_eg.pdf

---

High level Pseudo code for Baum Welch

```
input: initial parameters (\( \pi, A, B \))

create expectation accumulators
START[N], TRANS[N][N], OCC[N], OUT[N][M]
to hold start, transition, occupation, output expectations

for each obs seq \( o \) {
    make \( \alpha \) and \( \beta \) tables given \( o \)
    update START, TRANS, OCC, OUT using \( \alpha \), \( \beta \) tables
}

use accumulators to redefine parameters (\( \pi', A', B' \))

stop if fixed point in params (or overall corpus likelihood)
```