CSE 527 Lecture Notes
Autumn 2005 Lecture 14
Written 2003 by Martha Mercaldi and Tin Louie
Updated 12/17/2005 by Aditya Prasad - akprasad@gmail.com

Baum-Welch Training

The Baum-Welch algorithm is also known as the forward-backward algorithm. It is used to find the unknown parameters of an HMM, through an EM procedure.

The following quote is from [1]:

"What's imporant about the Baum-Welch Training Method is that we can feed in observation sequences, and as long as we present enough data, we will get an HMM that is pretty close to optimal for the given training sequences. Thus, if the training sequences are characteristic of the actual distribution of observation sequences in the system we're modelling, our HMM will be able to tell us what state the system is in with a high degree of certainty. It must be noted, however, that as with any gradient descent algorithm, it is not guaranteed that the HMM will end up in a state of globally minimal error. Instead, it settles into a local minimum, which hopefully is not too far from the global minimum."

The general consensus is that this works better than Viterbi Training, but because of its slower speed, people don't always use it.

$P(\pi_i = k, \pi_{i+1} = l|x, \Theta) = \frac{f_k(i) \cdot e_l(x_{i+1}) \cdot b_l(i+1)}{P(x|\Theta)}$ Number of $k \to l$ transitions $= \Sigma_{trainingdata} \frac{1}{P(x_j)} \cdot \Sigma_i P(\pi = k, \pi_{i+1} = l|x, \Theta)$ The number of emissions is a similar formula.

On the training sequences we run the forwards/backwards algorithm and use the formulas (see lecture slide) to arrive at some Θ. We then use that Θ to rerun the algorithm and generate a new set of values for Θ. Repeat this loop until the values for Θ converge. Convergence in this algorithm (as opposed to Viterbi Training) is a little trickier because you are optimizing a continuous sequence. Other disadvantages include the risk of overfitting the data or identifying local maxima. It also has high training data requirements.

Class discussion

In class, we discussed how to tell if a generated model is any good. The suggestion came up that we should be able to use some of our training points to train the model, and then check it against remaining training data. Prof. Ruzzo mentioned "Leave-one-out cross validation". The idea is to train the model on all but one data point, make a prediction for that point, and compute the average error across all such points. Details can be found here[2].

Note on Figures 5.5 and 5.6 from lecture slides: Odds score is the ratio of the sequence score with respect to the foreground model versus the score with respect to the background model. The log-odds score is just the log of this value. The followinq quote[3] describes how to determine statistical significance for these values.

"The threshold for statistical significance of an odds score depends upon the expected number of family members in the database. A sequence can be safely deemed a family member if its odds score is greater than the ratio of the total number of sequences in the database over the expected number of family members. For example, if you are searching a database of 36000 sequences for a family containing approximately 100 sequences, then any sequence that receives an odds score of (36000 / 100) = 360 or higher likely belongs to the family."

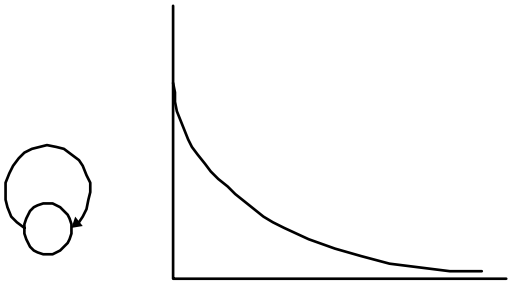
Summary

Viterbi: best single path
Forward: estimate probability by summing over all paths
Backwards: similar to forward, but on the back end of the sequence
Baum-Welch: training based on EM and forward/backward algorithms

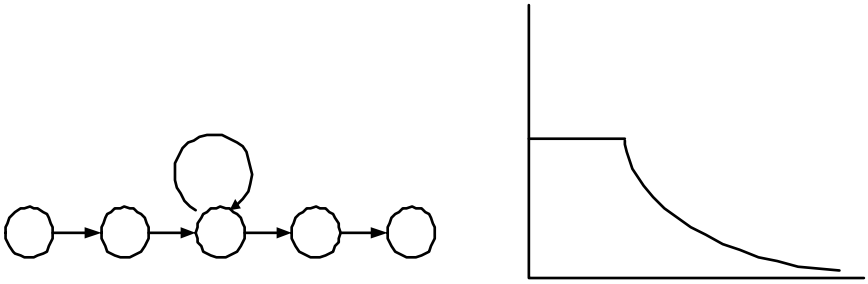
Model Structure

We want to find the correct number of states and topology to link them together. This tricky because there are many degrees of freedom in this problem and models that appear different can actually be equivalent. It is important in practice to constrain the allowable structures as much as possible. In particular, allowing all n^2 transitions between states in an n state model and "letting the data pick the model" doesn't work well in practice.

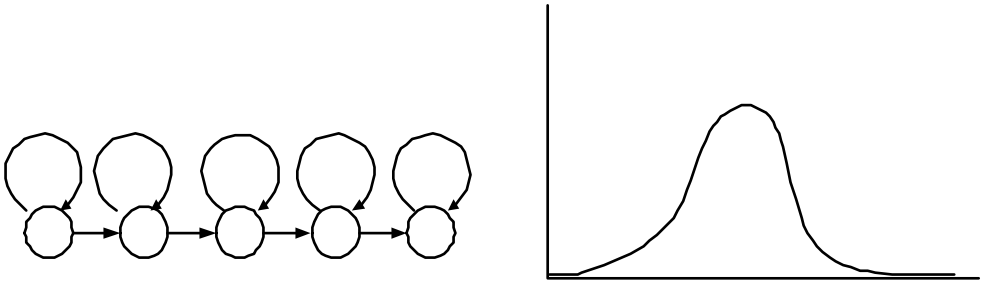
As another example of a model topology issue, consider the length distributions of features predicted by HMM's. For example, the lengths of C_pG islands predicted by our C_pG HMM is basically geometrically distributed $(p^n(1-p)$, where p is the probability of returning to a + state and $1-p$ is the probability of transitioning from a + to a − state), but in reality the C_PG islands don't follow this distribution (which is perhaps why the "postprocessing" step was needed to reduce false positives).



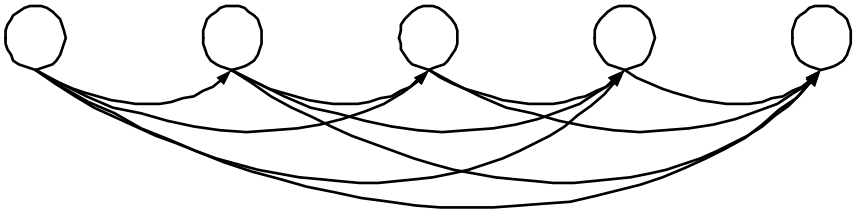
You could play with the model a little and require a minimum number of states thereby changing the distribution:



Alternatively, if you chain together several looping states, the "law of large numbers" will kick in and push you towards a bell shape

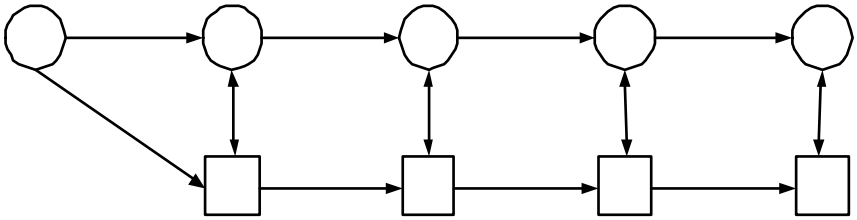


The downside of all of this is that the number of states quickly grows, making the model slow and requiring many parameters. For example, look at "profile HMMs" (next lecture) for recognizing similar proteins. Because it is possible to delete arbitrary runs of amino acids the model has $O(n^2)$ transitions.



However, we can get around this by implementing "silent" states corresponding to the normal states but, which don't emit any amino acid. We allow the model to switch arbitrarily between emitting and silent states. This gets

us back to $O(n)$ transition probablities to learn from training data, at the expense of less flexibility is specifying length distributions of consecutive deletions—a good compromise for modeling protein families, but certainly not universally appropriate.



If length distributions really differ from the above geometric distribution variants, it is also possible to explicity model residence time in a certain state, although this complicates the training and inference algorithms. We'll see an example of this in a week or so when we talk about gene finding.

Pfam: HMMs in action

Pfam uses HMMs to classify proteins into families. Pfam-A is curated, and Pfam-B is automated. As for genes, note that even though our genome databases contain over 100 billion bases, some 30-40% of newly-discovered genes are not recognizable.

"Profile hidden Markov models can be used to do sensitive database searching using statistical descriptions of a sequence family's consensus" (http://pfam.wustl.edu/hmmsearch.shtml)

No structural similarity is taken into account; this is NOT a method which includes scoring terms for nearby amino acids in a 3D structure.

Note that multiple alignment alone doesn't cut it: (the following is a quote from http://hmmer.wustl.edu/)

"You have carefully constructed a multiple sequence alignment [on a protein sequence family]. Your family, like most protein families, has a number of strongly (but not absolutely) conserved key residues, separated by characteristic spacing. You wonder if there are more members of your family in the sequence databases, but the family is so evolutionarily diverse, a BLAST search with any individual sequence doesn't even find the rest of the sequences you already know about. You're sure there are some distantly related sequences in the noise. You spend many pleasant evenings scanning weak BLAST alignments by eye to find ones with the right key residues are in the right places, but you wish there was a computer program that did this a little better."

Note that weight matrices don't handle variable-length gaps. In figure 5.2 from the lecture slides, if we ignore all the insert and delete states, what we have left (the M states) is essentially a weight matrix model (WMM).).

The model: (see the lecture slides)

- match states (have emission)
- insert states (have emission)
  - match portions of query sequences that do not match conserved regions
  - follows a geometric distribution
- delete states (no emissions)

The delete (silent) states allow you to skip arbitrarily many states. The choice of a silent state model (slide 43) has two implications: using more parameters, we get more accuracy, but more complexity. As an example, consider the last diagram above. If we want to skip from some state 17 to state 20, for example, we will have to go through a silent state that is shared with the skip $18 \rightarrow 20$. This forces us to accept a correlation that may be inaccurate. On the other hand, it requires many fewer parameters (n skip transitions as opposed to n^2). Transition & emission probabilities are based on counts from the training set.

How do we decide between a match or an insert state?

- Simple approach: if the multiple alignment shows that more than half of the sequences have a gap in this position, then decide this is an insert state

- Better approach: maximum a posteriori assignment with forward-like algorithm

The slides mentions two types of scoring (log-likelihood or log-odds). Either scoring method can be converted to Z scores (the following is paraphrasing from "Hidden Markov Models in Computational Biology", Anders Krogh, Michael Brown, et al., J Mol Biol (1994))

" — quantify the difference between log-likelihood scores for proteins containing the domain & scores for proteins not containing the domain — using a local windowing technique, we first calculate a smooth average curve for the roughly linear band of the log-likelihood score vs length plot. The standard deviation around this average curve is also calculated. Using this, we calculate the difference between the scores of a sequence and the average score of typical sequences of that same length, measured in standard deviations. This number is called the Z-score for the sequence. We then choose a Z-score cut-off, either a priori or by looking at the histogram of Z scores, and use it to decide if a given sequence fits the model or not."

Definitions for scoring used on those slides:

- log-odds: log of the ratio of the likelihoods of this model vs background/"random" model)
- z-score: difference from the mean, over standard deviation

Refinements:

- for small training sets, use pseudocounts to avoid a count of zero and zero emission probability
- a constant (called A in the formula on the lecture slides) and background rate
- weighted mixtures of pseudocounts, based on probabilities of being in certain regions
- downweight highly similar sequences due to sampling bias (the training set really should contain "independent" sequences, but often has many samples from phylogenetically close species, for example).

References

[1] http://occs.cs.oberlin.edu/~btaitelb/projects/honors/honorsnode11.html

[2] http://www.cs.cmu.edu/~schneide/tut5/node42.html

[3] http://metameme.sdsc.edu/mhmm-overview.html