Smooth On-Line Learning Algorithms for Hidden Markov Models

Pierre Baldi *
Jet Propulsion Laboratory
California Institute of Technology

Yves Chauvin #
Net-ID, Inc.

A simple learning algorithm for Hidden Markov Models (HMMs) is presented together with a number of variations. Unlike other classical algorithms such as the Baum-Welch algorithm, the algorithms described are smooth and can be used on-line (after each example presentation) or in batch mode, with or without the usual Viterbi most likely path approximation. The algorithms have simple expressions which result from using a normalized-exponential representation for the HMM parameters. All the algorithms presented are proved to be exact or approximate gradient optimization algorithms with respect to likelihood, log-likelihood or cross-entropy functions, and as such are usually convergent. These algorithms can also be casted in the more general EM (Expectation-Maximization) framework where they can be viewed as exact or approximate GEM (Generalized Expectation-Maximization) algorithms. The mathematical properties of the algorithms are derived in the appendix.

* and Division of Biology, California Institute of Technology (to whom all correspondence should be addressed).
# and Department of Psychology, Stanford University.

Hidden Markov Models (HMMs) are a particular class of adaptive systems which have been extensively used in speech recognition problems (see Rabiner (1989) for a review), but also in other interesting applications such as single channel kinetic modeling (Ball and Rice (1992). More recently, HMMs and related more general statistical techniques such as the EM (Expectation-Maximization) algorithm (Dempster et al. (1977)) have been applied to the modeling and analysis of DNA and protein sequences in biology (Baldi et al. (1992) and (1993), Cardon and Stormo (1992), Haussler et al. (1992), Krogh et al. (1993), and references therein) and optical character recognition (Levin and Pieraccini (1993)).

A first order HMM M is characterized by a set of states, an alphabet of symbols, a probability transition matrix $T = (t_{ij})$ and a probability emission matrix $E = (e_{ij})$. The parameter t_{ij} (resp. e_{ij}) represents the probability of transition from state i to state j (resp. of emission of symbol j from state i). HMMs can be viewed as adaptive systems: given a training sequence of symbols O, the parameters of a HMM can be iteratively adjusted in order the optimize the fit between the model and the data, as measured by some criterion. Most commonly, the goal is to maximize the likelihood $L_O(M) = p(O|M)$ (or rather its logarithm) of the sequence according to the model. This likelihood can be computed exactly using a simple dynamic programming scheme, known as the forward procedure, which takes into account all possible paths through M capable of producing the data. In certain situations, it is preferable to approximate this likelihood with a lower bound corresponding to the likelihood of the most probable path only. The most probable path is also easily computed using a slightly different recursive procedure, known as the Viterbi algorithm. When several training sequences are available, they can usually be treated as independent and therefore the goal is to maximize the likelihood of the model $L(M) = \prod_O L_O(M)$, or its logarithm $\sum_O \log L_O(M)$. Different objective functions, such as the posterior distribution of the model given the data, are also possible (for instance, Stolcke and Omohundro (1993)).

Learning from examples in HMMs is typically accomplished using the Baum-Welch algorithm. At each iteration of the Baum-Welch algorithm, the expected number $n_{ij}(O)$ (resp. $m_{ij}(O)$) of $i \to j$ transitions (resp. emissions of letter j from state i) induced in a given model, with a fixed set of parameters, by each training sequence O, is calculated using the forward-backward procedure (see Rabiner (1989) and our appendix for more formal definitions). The transition and emission probabilities are then reset according to

$$t_{ij} = \frac{\sum_O [n_{ij}(O)/L_O(M)]}{\sum_O [n_i(O)/L_O(M)]} \quad \text{and} \quad e_{ij} = \frac{\sum_O [m_{ij}(O)/L_O(M)]}{\sum_O [m_i(O)/L_O(M)]} \tag{1}$$

where $n_i(O) = \sum_j n_{ij}(O)$ and $m_i(O) = \sum_j m_{ij}(O)$. Thus, in the case of a single training sequence, at each iteration the Baum-Welch algorithm resets a transition or emission parameter to its expected frequency, given the current model and the training data. In the case of multiple training sequences, the contribution of each sequence must be weighted by the inverse of its likelihood.

It is clear that the Baum-Welch algorithm can lead to abrupt jumps in parameter space and that the procedure is not suitable for on-line learning, i.e. after each training example. This is even more so if, in order to save some computations, the Viterbi approximation is used to estimate likelihoods and transition and emission statistics by computing only the most likely path associated with the production of a sequence (see also Juang and Rabiner (1990), Merhav and Ephraim (1991)), as opposed to the forward-backward procedure where all possible paths are examined. Along such a single path, the transition or emission counts are necesssarily 0 or 1 (provided there are no loops) and therefore cannot be reasonably used in an on-line version of (1). Another problem with the Baum-Welch algorithm is that 0 probabilities are absorbing: once a transition or emission probability is set to 0, it is never used again and therefore remains equals to 0. This is of course undesirable and usually is prevented by artificially enforcing that no parameter be less than a fixed small threshold.

A different algorithm for HMM learning which is smooth, overcomes the previous obstacles and can be used on-line or in batch mode, with or without the Viterbi approximation, can be defined as follows. First, a normalized-exponential representation for the parameters of the model is introduced. For each t_{ij} (resp. e_{ij}), a new parameter w_{ij} (resp. v_{ij}) is defined according to

$$t_{ij} = \frac{e^{\lambda w_{ij}}}{\sum_k e^{\lambda w_{ik}}} \quad \text{and} \quad e_{ij} = \frac{e^{\lambda v_{ij}}}{\sum_k e^{\lambda v_{ik}}} \tag{2}$$

where λ is a temperature parameter. Whichever changes are applied to the w's and v's by the learning algorithm, the normalisation constraints on the original parameters are automatically enforced by this reparametrization. One additional advantage of this representation is that none of the parameters can reach the absorbing value 0.

The representation (2) is general in the sense that any finite probability distribution can be written in this form, provided there are no 0 probabilities (or else one must allow for infinite negative exponents). This is a very good property for HMMs since, in general, 0 probabilities need to be avoided.

We shall now describe the on-line version of the learning algorithm, the batch version can be obtained immediately by summing over all training sequences. After estimating the statistics $n_{ij}(O)$ and $m_{ij}(O)$ using, for instance, the forward-backward procedure, the update equations of the new algorithm are particularly simple and given by

$$\Delta w_{ij} = \eta \frac{1}{L_O(M)}[n_{ij}(O) - n_i(O)t_{ij}] \quad \text{and} \quad \Delta v_{ij} = \eta \frac{1}{L_O(M)}[m_{ij}(O) - m_i(O)e_{ij}] \tag{3}$$

where η is the learning rate which incorporates all the temperature effects. Although, as we shall prove in the appendix, equation (3) represents nothing more than on-line gradient descent on the negative log-likelihood of the data, its remarkable simplicity seem to have escaped the attention of previous investigators, partly because of the monopolistic role played by the Baum-Welch algorithm. Notice that the on-line version of gradient descent on the negative likelihood itself is given by

$$\Delta w_{ij} = \eta \frac{L(M)}{L_O(M)}[n_{ij}(O) - n_i(O)t_{ij}] \quad \text{and} \quad \Delta v_{ij} = \eta \frac{L(M)}{L_O(M)}[m_{ij}(O) - m_i(O)e_{ij}] \tag{3'}$$

Thus, with suitable learning rates and excluding the probably rare possibility of convergence to saddle points, (3) and (3') will converge to a possibly local maximum of the likelihood $L(M)$. As for back-propagation or any other gradient algorithm, the convergence is exact in batch mode and stochastic in the on-line version. In general, for dynamic range reasons, (3) should be preferable to (3'). Furthermore, in the case of multiple training sequences, (3') requires that the global factor $L(M)$ be available. In certain situations, it may be possible to use a non-scaled version of (3) in the form

$$\Delta w_{ij} = \eta[n_{ij}(O) - n_i(O)t_{ij}] \quad \text{and} \quad \Delta v_{ij} = \eta[m_{ij}(O) - m_i(O)e_{ij}] \tag{3''}$$

This is particularly true when the bulk of the training sequences tend to have likelihoods which are roughly in the same range. If the distribution of the likelihoods of the training sequences is approximately gaussian, this requires that the standard deviation be relatively small. In many cases, it is reasonable to expect this standard deviation to decrease as learning progresses. When the likelihoods of the training sequences are in the same range, then (3") can be viewed as an approximate rescaling of (3) with the corresponding vectors being almost aligned and certainly in the same half space. So, with a proper choice of learning rate (in general different from the rate used in (3)), (3") should also lead to an increase of the likelihood.

The previous algorithms rely on the local discrepancy between the transitions and emission counts induced by the data and their predicted value from the parameters of the model. Variations on (3) and (3') can be constructed using the discrepancy between the corresponding frequencies in the form

$$\Delta w_{ij} = \eta \frac{1}{L_O(M)}[\frac{n_{ij}(O)}{n_i(O)} - t_{ij}] \quad \text{and} \quad \Delta v_{ij} = \eta \frac{1}{L_O(M)}[\frac{m_{ij}(O)}{m_i(O)} - e_{ij}] \tag{4}$$

$$\Delta w_{ij} = \eta \frac{L(M)}{L_O(M)}[\frac{n_{ij}(O)}{n_i(O)} - t_{ij}] \quad \text{and} \quad \Delta v_{ij} = \eta \frac{L(M)}{L_O(M)}[\frac{m_{ij}(O)}{m_i(O)} - e_{ij}] \tag{4'}$$

and

$$\Delta w_{ij} = \eta[\frac{n_{ij}(O)}{n_i(O)} - t_{ij}] \quad \text{and} \quad \Delta v_{ij} = \eta[\frac{m_{ij}(O)}{m_i(O)} - e_{ij}] \tag{4''}$$

In the appendix, it is shown that (4") can be reached through a different heuristic line of reasoning: by approximating gradient descent on an objective function constructed as the sum of locally defined cross-entropy terms. All the variations (4)-(4") are obtained by multiplying (3) or (3') by positive coefficients such as $n_i(O)$, which may depend both on the sequence O and the state i. In the case of a single training sequence, all the vectors associated with (3)-(4") are in the same half plane and all these rules will increase the likelihood, provided the learning rate is sufficiently small. In the case of multiple training sequences, it is reasonable to expect that on average, in many situations, (4)-(4") will still tend to increase the likelihood $L(M)$, or its logarithm, although not along the line of steepest ascent. Accordingly, their convergence can be expected to be slower.

In the case of an on-line Viterbi approximation, the optimal path $\pi^*(O)$ associated with the current training sequence is first computed, together with the associated likelihood $L^*_O(M)$. Any one of the previous algorithms can then be approximated by replacing the expected transition counts $n_{ij}(O)$ and $n_i(O)$ by the corresponding counts $n^*_{ij}(O)$ and $n^*_i(O)$ obtained along the optimal Viterbi path (and similarly for the emissions). From the definition of n_{ij} given in the appendix, it is easy to see that $n^*_{ij}(O) = c_{ij}(\pi^*)L^*_O(M)$, where $c_{ij}(\pi^*)$ is the number of times the $i \rightarrow j$ transition appears along the path π^*. For instance, for any i along π^*, the gradient descent equations of (3) can then be approximated by $\Delta w_{ij} = \eta[c_{ij}(\pi^*) - t_{ij}c_i(\pi^*)]$ and similarly for the emissions. In the case of an architecture without any loops, the Viterbi approximations are particularly simple since $c_{ij}(\pi^*) = c_i(\pi^*) = 0$ or 1. In a specific application (Baldi et al. (1992) and (1993)), good results have been obtained with the Viterbi approximation to (3), rewritten in the form

$$\Delta w_{ij} = \eta[t^t_{ij} - t_{ij}] \quad \text{and} \quad \Delta v_{ij} = \eta[t^e_{ij} - e_{ij}] \tag{5}$$

Here, for a fixed state i, t^t_{ij} and t^e_{ij} are the target transition and emission values: $t^t_{ij} = 1$ every time the $i \rightarrow j$ transition is part of the Viterbi path of the corresponding training sequence and 0 otherwise; and similarly for t^e_{ij}. In particular if, as a result of a loop, a Viterbi path visits the state i several times, then (5) must be repeated at each visit ((5) is "on-line" not only with respect to different paths associated with different training sequences but also within each path). As for (4"), it is shown in the appendix that (5) can also be derived by approximate gradient descent on a sum of local cross-entropy terms. One important difference, however, is that this time, this objective function can be discontinuous as a result of the evolution of the Viterbi paths themselves.

In many applications, the likelihoods $L_O(M)$ are very small and beyond machine precision. This problem can be dealt with by using a scaling approach (Rabiner (1989)). The scaling equations derived for the Baum-Welch algorithm can readily be extended to the algorithms presented here. Another possibility, often used for obvious reasons in conjunction with a Viterbi approximation, is to calculate only the logarithm of the likelihoods or the logarithm of the likelihood of the optimal paths. Accordingly, it may be possible in some situations, to replace in the previous algorithms the factors $1/L_O(M)$ by $-\log L_O(M)$.

The algorithms previously described can also be seen in the more general EM framework (Dempster et al. (1977)). In the general EM framework, one considers two dependent random variables X and Y, where Y is the observed random variable. For a given value of X, there is a unique value of Y but, in general, different values of X can lead to the same value of the observable Y. In addition, there is a parametrised family of densities $f(x|\theta)$ depending on a set of parameters θ. In HMM terminology, X corresponds to the paths, Y to the output sequences and θ to the transition and emission parameters. As usual, the problem is to try to find a maximum likelihood estimate for the set of parameters θ, from the observations y. The EM algorithm is a recursive procedure defined by:

$$\theta(t+1) = \arg\max_{\theta} F(\theta, \theta(t), y) = \arg\max_{\theta} E[\log f(x|\theta)|\theta(t), y] \tag{6}$$

It can be shown that, in general, (6) converges monotonically to a possibly local maximum likelihood estimator of θ. When applied in the HMM context, the EM algorithm becomes the Baum-Welch algorithm. An interesting and slightly different view of the EM algorithm, which leads also to incremental variants, can be found in Neal and Hinton (1993), where X can be interpreted as representing the states of a statistical mechanical system. If the energy of a state is measured by its negative log likelihood, then (6) can be seen as a double minimization step on the corresponding free energy function. Any algorithm which increases the function F in (6) (and, as a result, increases also the likelihood of the observation given the parameters), without necessarily maximizing it, is called a GEM algorithm. It can be shown in general (a proof is given in the appendix in the HMM context) that the gradient of F and the gradient of the log likelihood of the observations given the parameters are identical. A small gradient ascent step on the log likelihood must lead to an increase of F. Thus, with sufficiently small learning rates, gradient descent on the negative log-likelihood and the other related algorithms presented here can be seen as special cases of GEM algorithms or approximations to GEM algorithms.

For a specific application, it is natural to ask which of the previous learning rules should be used. We believe that the answer to this question depends both on the application considered, the architecture of the HMM and possibly other implementation constraints, such as the available precision. It is clear that (3) should play for HMMs the role back-propagation plays for neural networks. Indeed, it is well known that HMMs can be viewed as a particular kind of linear neural networks. Back-propagation applied to these equivalent networks together with the normalized-exponential parameter representation leads immediately to (3). On the other hand, it is easy to see on simple examples that, for instance, (3) and (5) can behave very differently. There are applications, such as those in the area of DNA or protein sequence modeling, where the Viterbi paths play a particular role and where a Viterbi learning approximation may be more desirable. In this context, extensive simulation results on one of these smooth algorithms (5), can be found in (Baldi et al. (1992) and (1993)). As in any gradient method, the choice of the learning rate is also crucial and may require some experimentation. It should also be obvious that, as in the case of neural networks and other modeling techniques, the present ideas can easily be extended to more complex objective functions including, for instance, regularizer terms reflecting prior knowledge on the HMM parameters. The same is true also for higher order HMMs.

In general, the simple algorithms introduced above should be useful in situations where smoothness and/or on-line learning are important. These could include:
(i) large models with many parameters and relatively scarce data which may be more prone to overfitting, local minima trapping and other pathological behaviors when trained with discontinuous or batch rules;
(ii) analog physical implementations where only continuous learning rules can be realized;
(iii) and all situations where the storage of examples for batch learning is not desirable.

In the following mathematical appendix, we first show that (3) and (3') correspond to gradient ascent on the log-likelihood and likelihood respectively. We also give a different derivation of (4") and (5) as gradient descent algorithms on a different objective function. We then examine the relation of the algorithms to the Baum-Welch algorithm and to the more general EM approach.

Mathematical Appendix

For brevity, only transition parameters will be considered here but the analysis for emission parameters is analogous. In what follows, we will need the partial derivatives

$$\frac{\partial t_{ij}}{\partial w_{ij}} = \lambda t_{ij}(1 - t_{ij}) \quad \text{and} \quad \frac{\partial t_{ij}}{\partial w_{ik}} = -\lambda t_{ij} t_{ik} \tag{7}$$

A. Derivation of (3) and (3') as Global Gradient Descent Algorithms

For any path π through the model M and any fixed sequence O, let

$$\alpha_{\pi,O} = P(\pi, O|M) \tag{8}$$

so that

$$L_O(M) = P(O|M) = \sum_{\pi} \alpha_{\pi,O} \tag{9}$$

the summation being over all possible paths through the architecture (for a finite sequence, only a finite number of paths needs to be considered). With several training sequences, the likelihood of the model is given by

$$L(M) = \prod_{O} L_O(M) \tag{10}$$

Since the probability $\alpha_{\pi,O}$ is the product of all the corresponding transition and emission probabilities along the path π, it is easy to see that

$$\frac{\partial \alpha_{\pi,O}}{\partial t_{ij}} = \frac{c_{ij}(\pi)}{t_{ij}} \alpha_{\pi,O} \tag{11}$$

where $c_{ij}(\pi)$ is the number of times the $i \to j$ transition is used along the path π consistent with the production of O. From (9) and (11),

$$\frac{\partial L_O(M)}{\partial t_{ij}} = \frac{n_{ij}(O)}{t_{ij}} \tag{12}$$

where $n_{ij}(O) = \sum_{\pi} c_{ij}(\pi)\alpha_{\pi,O}$ is the expected number of times the $i \to j$ transition is used to produce O in model M. Combining (10) and (12) leads to

$$\frac{\partial L(M)}{\partial t_{ij}} = \frac{L(M)}{t_{ij}} \sum_{O} \frac{n_{ij}(O)}{L_O(M)} \tag{13}$$

Finally, using (7), the chain rule and a few simplifications, we get

Proposition 1:

$$\frac{\partial L(M)}{\partial w_{ij}} = \sum_k \frac{\partial L(M)}{\partial t_{ik}} \frac{\partial t_{ik}}{\partial w_{ij}} = \lambda L(M) \sum_O \frac{1}{L_O(M)} \{ n_{ij}(O) - t_{ij} n_i(O) \} \tag{14}$$

If in (10) we use the log-likelihood, then we get

Proposition 2:

$$\frac{\partial \log L(M)}{\partial w_{ij}} = \frac{1}{L(M)} \frac{\partial L(M)}{\partial w_{ij}} = \lambda \sum_O \frac{1}{L_O(M)} \{ n_{ij}(O) - t_{ij} n_i(O) \} \tag{15}$$

Clearly, (3) is the on-line version of (15), (4) is the on-line version of (14), and the temperature parameter λ can be absorbed in the learning rate. The key conclusive point, is that (3) corresponds to gradient descent on the negative log-likelihood and therefore is the most sensible algorithm. Notice that the contribution of each sequence must be weighted by the *inverse* of its likelihood. This may seem a little paradoxical at first sight, but is easily understood: if the model assigns a small likelihood to a given sequence, the model is not performing properly on that sequence and therefore the corresponding verror signal should be high.

B. Heuristic Derivation of (4”) and (5) as Approximate Gradient Descent Algorithms on a Sum of Local Cross-Entropy Terms

Once the numbers $n_{ij}(O)$ have been calculated, the *local* distance between the current distribution t_{ij} of transitions in the model and the expected distribution induced by the data can be measured using the cross-entropy function

$$H_{iO}(n,t) = \sum_j \frac{n_{ij}(O)}{n_i(O)} \log \frac{n_{ij}(O)/n_i(O)}{t_{ij}} \tag{16}$$

Equivalently, one could consider the local likelihood associated with the distribution of the transitions out of state i. The logarithm of this likelihood then, yields a term similar to (16). The parameters can now be updated by gradient descent on $H(n,t) = \sum_O \sum_i H_{iO}(n,t)$ in order to reduce this distance

$$\frac{\partial H(n,t)}{\partial w_{ij}} = \sum_O \frac{\partial H_{iO}}{\partial w_{ij}} \approx \sum_O \sum_k \frac{\partial H_{iO}}{\partial t_{ik}} \frac{\partial t_{ik}}{\partial w_{ij}} = \lambda \sum_O \left(-\frac{n_{ij}(O)}{n_i(O)}(1-t_{ij}) + \sum_{k \neq j} \frac{n_{ik}(O)}{n_i(O)} t_{ij} \right) \tag{17}$$

The approximation in (17) computes only the *explicit* derivative of $H(n,t)$ with respect to w_{ij}. All other higher order contributions, asociated with the fact that a change in t_{ik} affects all the quantities $n_{jl}(O)$ ($\partial H_{jO}/\partial t_{ik}$ is usually non-zero), are neglected. After simplifications, we get

Proposition 3:

$$\frac{\partial H(n,t)}{\partial w_{ij}} \approx \lambda \sum_O [-\frac{n_{ij}(O)}{n_i(O)} + t_{ij}] \tag{18}$$

Clearly, (4") is the on-line descent version of (18). Thus (4") is the learning rule derived by approximating gradient descent on the sum of the local cross-entropy terms between the distribution of transitions in the model at state i and the expected value of this distribution once the data is taken into account. Similar more complex learning rules could also be derived by weighting the terms H_{iO} in the sum H (for instance by $1/L_O(M)$). Again, the temperature coefficient λ can be merged into the learning rate. The same reasoning using the Viterbi approximation yields (5) (notice that the instantaneous distribution of transitions out of any state along a Viterbi path is a multinomial distribution of the form $\prod_j (t_{ij})^{t^t_{ij}}$).

C. Relations to Baum-Welch, EM and GEM Algorithms

To examine the relation of the previous algorithms to the Baum-Welch algorithm, it is useful to examine the proof of convergence for the Baum-Welch algorithm (Baum (1972)). Consider, for a fixed architecture, two models M and M' with different transitions and emission parameters. M' can be thought as being the improvement over M that we are seeking. For any path π through the model, we can define the probabilities

$$\alpha_{\pi,O} = P(\pi, O|M) \quad \text{and} \quad \beta_{\pi,O} = P(\pi, O|M') \tag{19}$$

Assuming, as usual, independence between the sequences the likelihood of each model is then given by

$$L(M) = \prod L_O(M) \quad \text{and} \quad L(M') = \prod L_O(M') \tag{20}$$

with

$$L_O(M) = \sum_\pi \alpha_{\pi,O} \quad \text{and} \quad L_O(M') = \sum_\pi \beta_{\pi,O} \tag{21}$$

the summations being over all possible paths π through the architecture. For each sequence, we can then define two probability distributions $\alpha_{\pi,O} / \sum_\pi \alpha_{\pi,O}$ and $\beta_{\pi,O} / \sum_\pi \beta_{\pi,O}$ induced by the two models and the sequence O over all paths. We can again measure the distance between these two distributions using the cross-entropy:

$$H_O(\alpha, \beta) = \sum_\pi \frac{\alpha_{\pi,O}}{\sum_\pi \alpha_{\pi,O}} \log \left(\frac{\alpha_{\pi,O} / \sum_\pi \alpha_{\pi,O}}{\beta_{\pi,O} / \sum_\pi \beta_{\pi,O}} \right) \tag{22}$$

The cross entropy being always positive, (20) yields, after simple manipulations, the relation

$$\log \frac{L_O(M')}{L_O(M)} \geq \frac{1}{L_O(M)} \sum_\pi \alpha_{\pi,O} \log \beta_{\pi,O} - \sum_\pi \alpha_{\pi,O} \log \alpha_{\pi,O} \tag{23}$$

which gives

$$\log \frac{L(M')}{L(M)} \geq \sum_O [\frac{1}{L_O(M)} \sum_\pi \alpha_{\pi,O} \log \beta_{\pi,O} - \sum_\pi \alpha_{\pi,O} \log \alpha_{\pi,O}] = R \tag{24}$$

When $M = M'$, the right hand side R of (24) is equal to 0. Therefore, in order to find a model M' with a higher likelihood than the model M, we must increase the term $\sum_O \sum_\pi [\alpha_{\pi,O} \log \beta_{\pi,O}]/L_O(M)$ (in the general EM context, this term corresponds to the function F defined above). The Baum-Welch algorithm directly maximizes this term. On the other hand, the algorithm introduced here in (3) is just a gradient descent step towards the maximization of this term. Remarkably, therefore, both the log-likelihood $\log L(M)$ and the rigth hand side of (24) have the same gradient.

To see this, one must first observe that the probability $\beta_{\pi,O}$ is a product of probabilities corresponding to all the transitions associated with the path π and all the corresponding emissions associated with the sequence O. The contribution to the sum $\sum_O \sum_\pi [\alpha_{\pi,O} \log \beta_{\pi,O}]/L_O(M)$ from the transitions only (there is an additional similar term for the contributions resulting from the emissions) can therefore be rewritten as

$$Q = \sum_O \frac{1}{L_O(M)} \sum_{ij} \sum_{\pi} P(\pi, O|M) c_{ij}(\pi) \log t'_{ij} = \sum_O \frac{1}{L_O(M)} \sum_{ij} n_{ij} \log t'_{ij} \tag{25}$$

where $c_{ij}(\pi)$ is the number of $i \rightarrow j$ transitions contained in the path π and, as above, n_{ij} is the expected number of $i \rightarrow j$ transitions in model M, given the data. It is easy to check that the expression for the probability distribution that maximizes Q corresponds to the Baum-Welch algorithm with $t'_{ij} = \sum_O [n_{ij}/L_O(M)] / \sum_O [n_i/L_O(M)]$ and that, if we use a normalized-exponential representation for the parameters w', the gradient of R is given by

Proposition 4:

$$\begin{aligned} \frac{\partial R}{\partial w'_{ij}} = \frac{\partial Q}{\partial w'_{ij}} &= \sum_O \frac{1}{L_O(M)} [n_{ij}(1 - t'_{ij}) - \sum_{k \neq j} n_{ik} t'_{ij}] \\ &= \sum_O \frac{1}{L_O(M)} [n_{ij} - \sum_k n_{ik} t'_{ij}] = \sum_O \frac{1}{L_O(M)} (n_{ij} - t'_{ij} n_i) \end{aligned} \tag{26}$$

When used on-line, this immediately yields the algorithm in (3). Thus, the batch version of (3) performs gradient ascent on R or on $\log L(M)$ and leads, for suffiently small step sizes, to an increase of the likelihood unless the gradient is zero. It leads also to an increase of R and as such, can also be viewed as a GEM algorithm. The gradient of the cross-entropy term maximized by Baum-Welch is also the gradient of the log-likelihood function. Although the proof given here is based on HMMs, the same result can be proved similarly in the general EM framework by showing that the gradient of the likelihood and the gradient of the function being maximized during the M step of the EM algorithm are identical.

Acknowledgements

We would like to thank David Haussler, Anders Krogh and Esther Levin for useful discussions. The work of P. B. is supported by grants from the AFOSR and the ONR.

References

(1) Baldi, P., Chauvin, Y., Hunkapiller, T. and McClure, M. A. (1992) Adaptive Algorithms for Modeling and Analysis of Biological Primary Sequence Information. Net-ID Technical Report and submitted for publication.

(2) Baldi, P., Chauvin, Y., Hunkapiller, T. and McClure, M. A. (1993) Hidden Markov Models in Molecular Biology. New Algorithms and Applications. Advances in Neural Information Processing Systems **5**, S. J. Hanson, J. D. Cowan and C. Lee Giles Editors, Morgan Kaufmann, San Mateo, CA.

(3) Ball, F. G. and Rice, J. A. (1992) Stochastic Models for Ion Channels: Introduction and Bibliography. Mathematical Bioscience. In press.

(4) Baum, L. E. (1972) An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. Inequalities, **3**, 1-8.

(5) Blahut, R. E. (1987) Principles and Practice of Information Theory. Addison Wesley.

(6) Cardon, L. R. and Stormo, G. D. (1992) Expectation Maximization Algorithm for Identifying Protein-binding Sites with Variable Lengths from Unaligned DNA Fragments. Journal of Molecular Biology, **223**, 159-170.

(7) Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977) Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal Royal Statistical Society, **B 39**, 1-22.

(8) Haussler, D., Krogh, A., Mian, S. and Sjolander, K. (1992) Protein Modeling using Hidden Markov Models. Computer and Information Sciences Technical Report (UCSC-CRL-92-93), University of California, Santa Cruz.

(9) Juang, B. and Rabiner, L. R. (1990) The Segmental K-Means Algorithm for Estimating Parameters of Hidden Markov Models. IEEE Transactions on Acoustics, Speech and Signal Processing. **38**, 9, 1639-1641.

(10) Krogh, A., Brown, M., Mian, S., Sjolander, K. and Haussler, D. (1993) Hidden Markov Models in Computational Biology: Applications to Protein Modeling. Submitted for publication.

(11) Levin, E. and Pieraccini, R. (1993) Planar Hidden Markov Modeling: from Speech to Optical Character Recognition. Advances in Neural Information Processing Systems **5**, S. J. Hanson, J. D. Cowan and C. Lee Giles Editors, Morgan Kaufmann, San Mateo, CA.

(12) Merhav, N. and Ephraim, Y. (1991) Maximum Likelihood Hidden Markov Modeling Using a Dominant Sequence of States. IEEE Transactions on Signal Processing, **39**, 9, 2111-2115.

(13) Neal, R. M. and Hinton, G. E. (1993) A New View of the EM Algorithm that Justifies Incremental and Other Variants. Submitted to Biometrika.

(14) Rabiner, L. R. (1989) A Tutorial on Hidden Markor Models and Selected Applications in Speech Recognition. Proceedings of the IEEE, **77**, 2, 257-286.

(15) Stolcke, A. and Omohundro, S. (1993) Hidden Markov Model Induction by Bayesian Model Merging. Advances in Neural Information Processing Systems **5**, S. J. Hanson, J. D. Cowan and C. Lee Giles Editors, Morgan Kaufmann, San Mateo, CA.