Modeling Protein Families and Human Genes: Hidden Markov Models and a Little Beyond.

Pierre Baldi*
Division of Biology
California Institute of Technology
Pasadena, CA 91125
pfbaldi@juliet.caltech.edu
(818) 354-9038
(818) 393-5013 FAX

Yves Chauvin
Net-ID, Inc.
San Francisco, CA 94107
yves@netid.com

Abstract

We will first give a brief overview of Hidden Markov Models (HMMs) and their use in Computational Molecular Biology. In particular, we will describe a detailed application of HMMs to the G-Protein-Coupled-Receptor Superfamily. We will also derive a number of analytical results on HMMs that can be used in discrimination tests, and data base mining. We will then discuss the limitations of HMMs and some new directions of research. In particular, we will present a new class of hybrid HMM/Neural Network architectures. If time permits, we will conclude with some recent results on the application of HMMs to human gene modeling and parsing.

Keywords: Hidden Markov Models, Neural Networks, Language Modeling, Protein Modeling,Multiple Alignments, Data Base Searches, Gene Parsing, Exons, Introns

* and Jet Propulsion Laboratory, Caltech. To whom all correspondence should be addressed.

1 Introduction

Hidden Markov Models (HMMs) are a class of probabilistic and adaptive models that has been extensively used in speech recognition ([15]), but also in a number of other applications, such as single channel kinetic modeling ([5]) and optical character recognition ([13]). HMMs and the related EM (Expectation-Maximization) algorithm ([9]) have also been applied to several problems in computational biology including the modeling of coding/non-coding regions in DNA ([8]), of protein binding sites in DNA ([12]). Recently, the HMM approach has been extensively applied to model, align and recognize entire protein families using primary sequence information only ([4], [2] and [10]). Examples of families modeled with this approach include: globins, kinases, immunoglobulins, aspartic proteases, HIV membrane proteins, EF-hand and G-protein-coupled receptors. In related work, HMMs have also been applied to the problems of finding and parsing genes ([1], [11])

A first order discrete HMM can be viewed as a stochastic production system defined by a set of states $\mathbf{S}$, an alphabet $\mathcal{A}$ of M symbols, a probability transition matrix $\mathbf{T} = (t_{ij})$, and a probability emission matrix $\mathbf{E} = (e_{iX})$. The system randomly evolves from state to state, while randomly emitting symbols from the alphabet. In the case of protein modeling, the alphabet has $M = 20$ symbols, one for each amino acid ($M = 4$ for DNA or RNA models). When the system is in a given state i, it has a probability t_{ij} of moving to state j and a probability e_{iX} of emitting symbol X. Biological primary sequence modeling requires the introduction of particular states for possible insertions and deletions. The model is called hidden because only the output string of symbols is observable. One of the goals is precisely to gather information about the possible underlying random walks. One of the main properties of HMMs is that they are adaptable: given a set of training sequences, there exist algorithms to adjust the transition and emission parameters so as to optimize the fit of the model to the data, for instance by maximum likelihood methods. It is this property, combined with the rapidly increasing amount of available training data, that is essential for the application of HMMs to protein families and other problems in computational molecular biology.

In our oral presentation, we plan:

1. To give a brief review of the application of HMMs to immunoglobulins, followed by an in-depth study of the G-protein-coupled receptor superfamily.

2. To derive several analytical results (asymptotic behavior, standard deviation and normality of HMM scores for random sources) that are useful in data base searches and other tasks.

3. To discuss some of the limitations of HMMs and what can be done to overcome them. In particular, we will present a broad new class of hybrid HMM/Neural Network architectures.

4. To give, if time permits, an overview of the results on the application of HMMs to the problem of human gene parsing.

Background material for 1,2 and 4 can be found in the references quoted above. Here, in the next sections, we review the HMM technique for protein families, discuss some of its limitations, and then briefly present the HMM/NN hybrid architectures.

2 HMMs and Learning for Protein Families

As in the application of HMMs to speech recognition, a family of proteins can be seen as a set of different utterances of the same word generated by a common underlying HMM with a left-right architecture. In addition to the start and end state, there are three classes of states: the main states, the delete states and the insert states with

$$\mathbf{S} = \{start, m_1, ..., m_N, i_1, ..., i_{N+1}, d_1, ..., d_{N+1}, end\}$$

N is the length of the model, typically equal to the average length of the sequences in the family. The main and insert states always emit an amino-acid symbol, whereas the delete states are mute. The linear sequence of state transitions $start \rightarrow m_1 \rightarrow m_2 .... \rightarrow m_N \rightarrow end$ is the backbone of the model. Corresponding insert and delete states are needed for each main state to model insertions and deletions with respect to the backbone. There are no self-loops on the delete states nor on the main states. There is, however, a self-loop on the insert states to allow for multiple insertions at a given site. Naturally, a number of different possible architectures can be envisioned.

Given a set of training sequences, the parameters of the model can be iteratively modified to optimize the fit of the model to the data according to some measure, usually the likelihood of the data according to the model. Since the sequences can be considered as independent, this likelihood is equal to the product of the likelihoods of the single sequences. Different algorithms are available for HMM training, including the classical Baum-Welch algorithm ([6]). We have introduced a smooth algorithm in ([3]), which is particularly simple and can be used on-line, i.e. after the presentation of each example. The mathematical properties of this algorithm, its relation to other approaches and its advantages are discussed in ([3]). The basic idea behind the algorithm is simple: for each training sequence, first compute the corresponding most likely path through the model. This can be done efficiently in $O(N^2)$ steps using a dynamic programming scheme known as the Viterbi algorithm. Transition and emission probabilities along the path should then be increased, so as to increase also the likelihood of the corresponding sequence. This requires that other parameters be decreased accordingly, in order to preserve normalisation constraints on probability distributions.

More precisely, we first reparametrize the model using normalized exponentials and a new set of variables w_{ij} and v_{iX}

$$t_{ij} = \frac{e^{w_{ij}}}{\sum_k e^{w_{ik}}} \quad \text{and} \quad e_{iX} = \frac{e^{v_{iX}}}{\sum_Y e^{v_{iY}}} \tag{2.1}$$

This reparametrization has two advantages: (1) modification of the w's and v's automatically preserves the normalisation constraints on the original emission and

transition probability distributions; (2) transition and emission probabilities can never reach the absorbing value 0. We then iteratively cycle through the set of training sequences and, for each training sequence, through its Viterbi path. At each step along a Viterbi path, being in a state i, we update the parameters of the model according to

$$\Delta w_{ij} = \eta(T_{ij} - t_{ij}) \quad \text{and} \quad \Delta v_{iX} = \eta(E_{iX} - e_{iX}) \tag{2.2}$$

where η is the learning rate. At each step of the Viterbi path, and for any state i on the path, $T_{ij} = 1$ (resp. $E_{iX} = 1$) if the $i \rightarrow j$ transition (resp. emission of X from i) is used and 0 otherwise. In the case of a loop, as for the insert states, (2.2) must be repeated every time the loop is traversed. The new parameters are therefore updated incrementally using the discrepancy between the frequencies induced by the training data and the probability parameters of the model. These update rules must be repeated for each example until no significant variations for any of the parameters occur. In ([3]) it is shown that this algorithm is an approximation to a gradient descent procedure on the negative log-likelihood of the sequences given the model. As such, it can be expected to converge to a (possibly local) maximum likelihood estimator.

Once a HMM has been successfully trained on a family of primary sequences, it constitutes a model of the entire family and can be used in a number of tasks. First, for any given sequence, we can compute its likelihood according to the model or the likelihood of its most probable path using the Viterbi algorithm. A multiple alignment results immediately from aligning all the Viterbi paths of the sequences in the family. By looking at the transition and emission probabilities throughout the model, characteristic motifs and conserved regions can be identified and sometimes structural properties inferred. The model can also be used for discrimination i.e. to decide whether a given sequence belongs to the family based on its likelihood. This in turn can be applied to data base searches, classification, and fragment analysis.

3 Limitations of HMMs

There are two basic problems with HMMs, as a model class, in computational molecular biology: the number of parameters and the long range interactions.

3.1 Number of Parameters

The HMMs we have described have a large number of parameters that grows linearly with the length of the model, like $N[2M + 3F]$ (where F is the typical fan-out of the states). For large proteins, this can easily yield models with over 10,000 parameters. In the current early stages of genome sequencing projects, this can be a problem whenever only a few sequences are known in a given family. This limitation can be overcome by introducing prior knowledge in the models, when available. This knowledge can originate from several different sources: structural (crystallographic studies), functional (membrane protein), statistical (PAM and other substitution matrices) and so on. A natural way for incorporating certain forms of additional

knowledge into HMMs is by the use of Dirichlet priors (for instance [10]). Another possibility, aimed also in part at restricting the number of degrees of freedom in HMMs, is discussed in the next section on hybrid architectures.

3.2 Long Range Interactions

Because proteins have complex 3 dimensional shapes and long range interactions between their residues, it may seem surprising that good models can be derived using simple first order Markov processes. One must however be very careful in defining what kind of long range correlations cannot be captured by HMMs. Long range correlations related, for instance to folding properties, are not sufficient per se. Indeed, HMMs can capture those effects of long range interactions that manifest themselves in a more or less *constant* fashion, across a family of sequences. For instance, suppose that, as a result of a particular folding structure, two distant regions of a protein have a predominantly hydrophobic composition. Then this pattern will be present in all the members of the family and will be learnable by a HMM, with the proper local statistical variations. On the other hand, a *variable* long range interaction such as: "a residue X at position i implies a residue f(X) at position j" cannot be captured by a first order HMM, as soon as f is sufficiently complex. [Note that a HMM is still capable of capturing certain variable long range interactions. For instance, assume that the sequences in the family have either a fixed residue X at position i, and a corresponding fixed residue Y at position j, or a fixed residue X' a position i with a corresponding fixed Y' at position j. Then these 2 sub-classes of sequences in the family could be associated with 2 types of paths in the HMM where, for instance, $X-Y$ are emitted from main states and $X'-Y'$ are emitted from insert states.] But clearly, better models must be created to deal with long range variable interactions and, ultimately, with the folding problem. Higher-order Markov models, with fixed memory length, are too expensive from a computational standpoint and unlikley to be successfull. Variable length memory models (as in [16]) may be of some help in these matters.

3.3 Caveats

It has been our experience, however, that even in situations where the number of sequences available for training is relatively small, HMMs seem to perform rather well and certainly much better than what one would expect from a simple count of the number of parameters versus the number of training examples ([4]). We suspect this is yet another example of a more general emerging phenomena in the machine learning literature: at least for certain tasks, there exist certain model classes (with a lot of parameters), that are are *well-conditioned*, precisely in the sense that a good fit to the data can be obtained even when the training set is small. We even suspect, and this could easily be checked, that a HMM trained with only 2 sequences, using a form of Viterbi gradient learning, tends to produce an alignment close to the optimal pairwise alignment. Of course, we do not mean to imply that HMM should be used for pairwise alignments. HMM are an efficient tool for families and for producing multiple alignments.

Likewise, although long range correlations are important, their effects do not seem to have hampered the HMM approach so far. One reason described above is that only subtle variable long range correlations of the type $X \rightarrow Y/X' \rightarrow Y'$ between distant positions are likely to be missed by HMMs. But how likely is it, say in a data base search, that a sequence exist in the data base that has most of its major properties identical to those of the sequences in the family being modeled, but with a $X \rightarrow Y'$ correspondence?

For all these reasons, it seems to us that HMMs are in fact very close to being optimal for tasks such as multiple alignments and data base searches, and that there is little room for improvements.

In this context, and in connection with the next section, it is also useful to notice that there exists a simple hierarchy among statistical models for protein families. If a protein family is to be described on the basis of its multiple alignment, then first order models are entirely specified by the probability emission vectors at each position. The most trivial and weakest model would be one where these emissions are constant and set uniformly, followed by the model where they are constant and equal to the average composition of the sequences in the family. HMMs on the other hand allow for position dependent emission vectors. From this point of view, HMMs are in fact roughly equivalent to a multiple alignment since the HMM parameters can easily be inferred from a multiple alignment, and vice versa a properly trained HMM yields a multiple alignment. Variable long range correlations cannot be described in the formalism of a single HMM model, since they imply emission vectors that are not constant at a given position, but rather depend on the sequence being considered. Either the statistical model class must be changed or, if one is to stay with HMMs, several different HMMs are needed to represent a family, and the variability of its long range correlations. One must find ways, in some sort, of modulating the HMM as a function of different members of the family. Emission vectors must be variable not only as a function of position, but also to some extent as a function of sequence. Not surprisingly, this is also intimately related to the problem of finding sub-classes within a family.

4 HMM/NN Hybrid Architectures

To overcome some of the previous limitations, we are going to introduce a class of HMM/NN hybrid architectures and learning algorithms. Although these architectures are presented in the context of protein family modeling, they are readily applicable to other language tasks, and in fact to all problems where HMM techniques are applicable. It is of course not the first time HMMs and NNs are combined. Hybrid architectures have been used both in speech and cursive handwriting recognition ([7]). In many of these applications however, neural networks are used as front end processors to extract features, such as characters. HMMs are then used in the higher stages of the algorithms for word and language modeling. The HMMs and NNs components are often trained separately. Here, on the other hand, we assume that all the features have already been extracted and that we are given the exact

final sequences in digital format [1]. Thus the NN component of our architecture follows the HMM component. Or, more precisely, the two are intimately blended. This yields unified training algorithms where the HMM and the NN component are trained simultaneously.

4.1 Basic Idea

We are going first to derive one of the most simple HMM/NN hybrid architectures, and subsequently show how the basic ideas can be extended in many directions. For this first architecture, the basic idea is to combine the temporal-sequential structure of HMMs with the representational power of NNs, by having a NN on top of the HMM architecture for the computation of the HMM parameters. For simplicity, we shall begin with the emission parameters of the main states only.

If we look at (2.1), we can consider that each main state has its own independent little NN attached to it, consisting of 1 on/off input unit fully interconnected to $M = 20$ weighted exponential output units. The next natural step then is to link these little networks to take advantage of dependencies and to make them more complex by introducing hidden units Notice that for now, the underlying probabilistic model remains the HMM. Long range correlations and parameter compression is achieved through the NNs.

More formally, let us consider a NN for the emission of the mains states that consists of:

- **Input layer:** N input units, one for each main state $I = (i_1, ..., i_N)$. To calculate the emission vector of main state i: $i_k = 1$ for $k = i$ and 0 otherwise. Such an input will also be denoted just by i.
- **Hidden layer:** H hidden units indexed by h, each with transfer function f_h (logistic by default) with bias b_h ($H < M$). M is the size of the alphabet. Here $M = 20$. For parameter compression purposes, the number of hidden units must not exceed the size of the alphabet. The case of 0 hidden units corresponds to what we had in section 2.
- **Output layer:** M softmax units or weighted exponentials, indexed by X, the letters in the alphabet. It is possible to introduce a bias b_X parameter for the output units, as well as a gain factor, if desired.
- **Connections:** $\alpha = (\alpha_{hi})$ connects the i-th input position to the h hidden unit and $\beta = (\beta_{Xh})$ connects the h hidden to the X output unit.

Notice that with the proper choice of initial biases on the output units and with iniital weights close to 0, the emissions can easily be initialized uniformly across the model to the average composition of the family.

[1] It may be possible to use NN to interpret the analog output of various sequencing machines, but this is definetely not our focus here.

For input i, the activity in the hidden layer is given by:

$$f_h(\alpha_{hi} + b_h) = f_h(\alpha_{hi} + b_h) \tag{3.1}$$

The corresponding activity in the output layer is

$$e_{iX} = \frac{e^{-[\sum_h \beta_{Xh} f_h(\alpha_{hi}+b_h)+b_X]}}{\sum_Y e^{-\sum_h \beta_{Yh} f_h(\alpha_{hi}+b_h)+b_Y]}} \tag{3.2}$$

5 Training Algorithms

This simple architecture can be trained by combining any of the HMM training algorithms (EM or gradient descent, likelihood or Viterbi likelihood or MAP) with any of the NN training algorithms (backpropagation). Indeed, let us assume that there are K sequences in the training set $O_1, ..., O_K$. Two possible classical target functions to be optimised are the likelihood

$$Q = -\sum_{k=1}^{K} Q_k = -\sum_{k=1}^{K} \ln P(O_k) \tag{4.1}$$

and the Viterbi likelihood

$$Q = -\sum_{k=1}^{K} Q_k = -\sum_{k=1}^{K} \ln P(\pi(O_k)) \tag{4.2}$$

where $\pi(O)$ denotes the Viterbi path of sequence O. Learning can be on-line or off-line. Here we give the on-line equations (batch equations can be derived similarly). So, for a sequence O, we need to compute the partial derivatives of $\ln P(O)$ or $\ln P(\pi(O))$ with respect to the parameters α, β and b of the network.

5.1 Gradient Learning on Full Likelihood

Let $Q(O) = \ln P(O)$. If m_{iX} is the count for the emission of X from i for O derived using the forward-backward algorithm, then we have ([3])

$$\frac{\partial P(O)}{e_{iX}} = \frac{m_{iX}(O)}{e_{iX}} \tag{4.3}$$

so that

$$\frac{\partial Q}{\partial e_{iX}} = \sum_{k=1}^{K} \frac{1}{P(O)} \frac{m_{iX(O)}}{e_{iX}} \tag{4.4}$$

The partial derivatives with respect to the network parameters α, β and b can be obtained by the chain rule, that is by back-propagating through the network for each i. A simple calculation gives, for each sequence O and each main state i:

$$\frac{\partial Q(O)}{\partial \beta_{Xh}}(i) = \frac{1}{P(O)}[m_{iX}(O) - e_{iX} m_{i.}] f_h(\alpha_{hi} + b_h) \tag{4.5}$$

where, as usual, $m_{i.} = \sum_Y m_{iY}$. A similar equation holds for the biases b_X, by replacing $f_h(\alpha_{hi} + b_h)$ with 1.

$$\frac{\partial Q(O)}{\partial \alpha_{hj}}(i) = \begin{cases} \frac{1}{P(O)} f_h'(\alpha_{hi} + b_h)[\sum_Y (m_{iY}(O) - e_{iY} m_{i.})\beta_{Yh}] & \text{if } i = j \\ 0 & \text{if } j \neq i \end{cases} \tag{4.6}$$

This equation can also be rewritten in more compact form:

$$\frac{\partial Q(O)}{\partial \alpha_{hj}}(i) = \delta_{ij} f_h'(\alpha_{hi} + b_h)[\sum_Y (m_{iY}(O) - e_{iY} m_{i.})\beta_{Yh}] \tag{4.7}$$

$$\frac{\partial Q(O)}{\partial b_h}(i) = \frac{1}{P(O)} f_h'(\alpha_{hi} + b_h)[\sum_Y (m_{iY}(O) - e_{iY} m_{i.})\beta_{Yh}] \tag{4.8}$$

The full gradient is obtained by summing over all sequences O_k and all main states i. Thus, for instance,

$$\frac{\partial Q}{\partial \alpha} = \sum_O \sum_{i=1}^{N} \frac{\partial Q(O)}{\partial \alpha}(i) \tag{4.9}$$

and similarly for β and the bias b. So, the on-line learning equations, are (for each i and O):

$$\begin{cases} \Delta\beta_{Xh} = \eta \frac{1}{P(O)}[m_{iX}(O) - e_{iX} m_{i.}] f_h(\alpha_{hi} + b_h) \\ \Delta b_X = \eta \frac{1}{P(O)}[m_{iX}(O) - e_{iX} m_{i.}] \\ \Delta\alpha_{hi} = \eta \frac{1}{P(O)} f_h'(\alpha_{hi} + b_h)[\sum_Y (m_{iY}(O) - e_{iY} m_{i.})\beta_{Yh}] \\ \Delta\alpha_{hj} = 0 \qquad \text{for } j \neq i \\ \Delta b_h = \eta \frac{1}{P(O)} f_h'(\alpha_{hi} + b_h)[\sum_Y (m_{iY}(O) - e_{iY} m_{i.})\beta_{Yh}] \end{cases} \tag{4.10}$$

It is worth noticing, as in ([3]), that these learning equations are slightly different from those that would result by back-propagating on the local cross-entropy error measure between the emission distribution e_{iX} and the target distribution $m_{iX}/m_{i.}$ derived from the forward-backward algorithm.

5.2 Viterbi Learning

At least in the case of protein models, the use of Viterbi paths is best viewed as an algorithm in its own right rather than an approximation to the likelihood case. Here, let $Q(O) = \ln P(\pi(O))$. The component of this term that depends on emission from main states, and thus on α, β and b, along the Viterbi path $\pi = \pi(O)$ is given by

$$-\sum_{(i,X)\in\pi} \ln e_{iX} = -\sum_{(i,X)\in\pi} T_{iX} \ln \frac{e_{iX}}{T_{iX}} = -\sum_{i\in\pi}\sum_Y T_{iY} \ln \frac{e_{iY}}{T_{iY}} \tag{4.11}$$

where T_{iX} is the target, namely $T_{iX} = 1$ if X is emitted from main state i in $\pi(O)$, and 0 otherwise. Thus to compute the gradient of $Q(O) = -\ln P(\pi(O))$ with respect to α, β and b is equivalent to computing the gradient with respect to the cross entropy

$$H(T, E) = -\sum_{i\in\pi} H(T_i, e_i) = -\sum_{i\in\pi}\sum_Y T_{iY} \ln \frac{e_{iY}}{T_{iY}} \tag{4.12}$$

between the target output and the output of the network over all i in π. This cross entropy error function combined with the softmax ouput unit is the standard neural network framework for multinomial classification (see, for instance, [17]).

In summary, the relevant derivatives can be calculated on-line both with respect to the sequences $O_1, ..., O_K$ and, for each sequence, with respect to the Viterbi path. For each sequence O, and for each main state i on the Viterbi path $\pi = \pi(O)$, the corresponding contribution to the derivative can be obtained by standard back-propagation on the cross-entropy error function $H(T_i, e_i)$. For $(i, X) \in \pi$, a simple calculation gives:

$$\frac{\partial H(T_i, e_i)}{\partial \beta_{Yh}} = \begin{cases} (1 - e_{iX}) f_h(\alpha_{hi} + b_h) & \text{if } Y = X \text{ and } (i, X) \in \pi \\ -e_{iY} f_h(\alpha_{hi} + b_h) & \text{otherwise} \end{cases} \tag{4.13}$$

This equation can also be rewritten in more compact form:

$$\frac{\partial H(T_i, e_i)}{\partial \beta_{Yh}} = (T_{iY} - e_{iY}) f_h(\alpha_{hi} + b_h) \tag{4.14}$$

and similarly for the biases b_X, by replacing $f_h(\alpha_{hi} + b_h)$ with 1. Next,

$$\frac{\partial H(T_i, e_i)}{\partial \alpha_{hj}} = \begin{cases} f_h'(\alpha_{hi} + b_h)[\beta_{Xh}(1 - e_{iX}) - \sum_{Y \neq X} \beta_{Yh} e_{iY}] & \text{if } j = i \\ 0 & \text{if } j \neq i \end{cases} \tag{4.15}$$

This equation can also be rewritten in more compact form:

$$\frac{\partial H(T_i, e_i)}{\partial \alpha_{hj}} = \delta_{ij} f_h'(\alpha_{hi} + b_h)[\beta_{Xh}(1 - e_{iX}) - \sum_{Y \neq X} \beta_{Yh} e_{iY}] \tag{4.16}$$

For the biases in the hidden layer

$$\frac{\partial H(T_i, e_i)}{\partial b_h} = f_h'(\alpha_{hi} + b_h)[\beta_{Xh}(1 - e_{iX}) - \sum_{Y \neq X} \beta_{Yh} e_{iY}] \tag{4.17}$$

The full gradient is obtained by summing over all sequences O_k and all main states i present in the corresponding Viterbi paths $\pi(O_k)$. Thus, for instance,

$$\frac{\partial Q}{\partial \alpha} = \sum_O \sum_{i \in \pi(O)} \frac{\partial H(T_i, e_i)}{\partial \alpha} \tag{4.18}$$

and similarly for β and the bias b. So, the learning equations are:

$$\begin{cases} \Delta \beta_{Xh} = \eta(1 - e_{iX}) f_h(\alpha_{hi} + b_h) & \text{if } (i, X) \in \pi \\ \Delta \beta_{Yh} = -\eta e_{iY} f_h(\alpha_{hi} + b_h) & \text{if } Y \neq X \\ \Delta b_X = \eta(1 - e_{iX}) & \text{if } (i, X) \in \pi \\ \Delta b_X = -\eta e_{iY} & \text{if } Y \neq X \\ \Delta \alpha_{hi} = \eta f_h'(\alpha_{hi} + b_h)[\beta_{Xh}(1 - e_{iX}) - \sum_{Y \neq X} \beta_{Yh} e_{iY}] & \text{if } (i, X) \in \pi \\ \Delta \alpha_{hj} = 0 & \text{for } j \neq i \\ \Delta b_h = \eta f_h'(\alpha_{hi} + b_h)[\beta_{Xh}(1 - e_{iX}) - \sum_{Y \neq X} \beta_{Yh} e_{iY}] \end{cases} \tag{4.19}$$

In summary, unified learning algorithms can be derived for HMM/NN hybrid architectures. The previous derivations can easily be adapted to many other variations such as MAP target functions, or more complex NNs with, for instance, multiple hidden layers. Simulation results will be described in the oral presentation.

6 Extensions

The simple hybrid architectures we have just described can be extended in many directions. To mention just a few:

- Introduction of a NN for the insert state emissions as well as for the transition parameters. These NN can be independent or linked. In the case of protein models, the fan out of the states is relatively small so that litle compression is to be gained by using a NN for the transition parameters. These may be useful for other HMM applications. Notice also that having less hidden units in the NN that control insert state emission is an elegant way of ensuring predominance of main state over insert states without the need of specific regularizers (the same holds for transition parameters).
- Use of more complex NN architectures, with multiple hidden layers and all the usual connectionist tricks, such as weight sharing, weight pruning, weight decay, mixture of experts architectures, and so on.
- Introduction of various priors (Gaussian, Dirichlet, forcing strong sparse connections,...).
- Automatic adjustment of models to rapidly evolving genome data bases by incremental addition of hidden units to a given HMM/NN model as more sequences, in a given family, become available.
- Application to other domains.

To further develop these architectures, we must now return to some of our original motivations. One of the goals was to achieve parameter reduction and to capture long range correlation effects. It is quite obvious to see how the previous architectures provide a very flexible mean of reducing the number of parameters in a simple and quantifiable ways. As far as long range correlations, in spite of the introduction of NNs in the previous architectures, the final statistical model for the sequences remains so far a single HMM (where the NN only mediates the calculation of the HMM parameters). As discussed in the section on limitations, such a model cannot capture subtle variable long range correlations. It can however capture long range corrrelations which are expressed in a constant fashion and the NN parameters may make this more explicit. (For instance, in a simple HMM a disulphide bond is expressed by two distant emission vectors having a high probability for cysteine. This remains true in a HMM/NN architectures, but in addition there may be a high degree of similarity between the weights of the corresponding input units).

Within the HMM framework, sequence dependent long range effects can only be captured by using multiple HMMs or equivalently by modulating a single HMM as a function of sequences. In the HMM/NN hybrid architecture this requires the introduction of additional units at the input level and/or at the hidden level and/or at the output level to express multiple HMMs or to modulate a single HMM as a function of input sequence and context. For instance, the introduction of a relatively

small number of additional input units that modulate the HMM essentially contains, as a special case, the approach suggested in ([14]), with the added advantage of not requiring the availability of a good preexisting multiple alignment. These more general HMM/NN hybrid architecture will be discussed in the presentation.

7 Acknowledgement

The work of PB is in part supported by grants from the ONR, the AFOSR and a Lew Allen Award at JPL. The work of YC is supported by grant R43 LM05780 from the National Library of Medicine. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the National Library of Medicine."

References

[1] P. Baldi, S. Brunak, Y. Chauvin, and J. Engelbrecht. Hidden markov models of human genes. In G. Tesauro J. D. Cowan and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 761–768. Morgan Kaufmann, San Francisco, CA, 1994.

[2] P. Baldi and Y. Chauvin. Hidden markov models of the G-protein-coupled receptor family: a beginning. *Journal of Computational Biology*, 1994.

[3] P. Baldi and Y. Chauvin. Smooth on-line learning algorithms for hidden markov models. *Neural Computation*, 6(2):305–316, 1994.

[4] P. Baldi, Y. Chauvin, T. Hunkapillar, and M. McClure. Hidden markov models of biological primary sequence information. *PNAS USA*, 91(3):1059–1063, 1994.

[5] F. G. Ball and J. A. Rice. Stochastic models for ion channels: introduction and bibliography. *Mathematical Bioscience*, 112(2):189–206, 1992.

[6] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[7] H. Bourlard and N. Morgan. *Connectionist speech recognition: a hybrid approach*. Kluwer Academic Publishers, Boston, 1994.

[8] G. A. Churchill. Stochastic models for heterogeneous dna sequences. *Bulletin Mathematical Biology*, 51:79–94, 1989.

[9] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal Royal Statistical Society*, B39:1–22, 1977.

[10] A. Krogh, M. Brown, I. S. Mian, K. Sjolander, and D. Haussler. Hidden Markov models in computational biology: applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, 1994.

[11] A. Krogh, I. S. Mian, and D. Haussler. A hidden Markov model that finds genes in e. *coli* DNA. Technical Report UCSC-CRL-93-33, University of California at Santa Cruz, Computer Science, UC Santa Cruz, CA 95064, 1993. in preparation.

[12] C. E. Lawrence and A. A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7:41–51, 1990.

[13] E. Levin and R. Pieraccini. Planar hidden markov modeling: from speech to optical character recognition. In S. J. Hanson, J. D. Cowan, and C. Lee Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan Kaufmann, San Mateo, CA, 1993.

[14] D.J.C. MacKay. Bayesian neural networks and density networks, 1994. Proceedings of Workshop on Neutron Scattering Data Analysis and Proceedings of 1994 MaxEnt Conference, Cambridge (UK).

[15] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[16] D. Ron, Y. Singer, and N. Tishby. The power of amnesia. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann, San Francisco, CA, 1994.

[17] D.E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin. Backpropagation: the theory, 1994. In: Backpropagation: Theory, Architectures and Applications, Y. Chauvin and D.E. Rumelhart Editors, Lawrence Erlbaum Associates, New Jersey, in press.