Communicated by Terrence Sejnowski

Hybrid Modeling, HMM/NN Architectures, and Protein Applications

Pierre Baldi
Division of Biology, California Institute of Technology, Pasadena, CA 91125 USA

Yves Chauvin
Net-ID, Inc., San Francisco, CA 94107 USA

We describe a hybrid modeling approach where the parameters of a model are calculated and modulated by another model, typically a neural network (NN), to avoid both overfitting and underfitting. We develop the approach for the case of Hidden Markov Models (HMMs), by deriving a class of hybrid HMM/NN architectures. These architectures can be trained with unified algorithms that blend HMM dynamic programming with NN backpropagation. In the case of complex data, mixtures of HMMs or modulated HMMs must be used. NNs can then be applied both to the parameters of each single HMM, and to the switching or modulation of the models, as a function of input or context. Hybrid HMM/NN architectures provide a flexible NN parameterization for the control of model structure and complexity. At the same time, they can capture distributions that, in practice, are inaccessible to single HMMs. The HMM/NN hybrid approach is tested, in its simplest form, by constructing a model of the immunoglobulin protein family. A hybrid model is trained, and a multiple alignment derived, with less than a fourth of the number of parameters used with previous single HMMs.

1 Introduction: Hybrid Modeling

One fundamental step in scientific reasoning is the inference of parameterized probabilistic models to account for a given data set D. If we identify a model $M(\theta)$, in a given class, with its parameter vector θ, then the goal is to approximate the distribution $P(\theta|D)$, and often to find its mode $\max_\theta P(\theta|D)$. Problems, however, arise whenever there is a mismatch between the complexity of the model and the data. Too complex models result in overfitting; too simple models result in underfitting.

The hybrid modeling approach attempts to finess both problems. When the model is too complex, it is reparameterized as a function of

a simpler parameter vector $\vec{w}$, so that $\theta = f(\vec{w})$.[1] When the data are too complex, short of resorting to a different model class, the only solution is to model the data with several $M(\theta)$s, with θ varying discretely or continuously across different regions of data space. Thus the parameters must be modulated as a function of input, or context, in the form $\theta = f(I)$. In the general case, both may be desirable, so that $\theta = f(\vec{w}, I)$. This approach is hybrid, in the sense that the function f can belong to a different model class. Since neural networks (NN) have well-known universal approximation properties, a natural approach is to compute f with an NN, but other representations are possible. This approach is also hierarchical because model reparameterizations can easily be nested at several levels. Here, for simplicity, we confine ourselves to a single level of reparameterization.

For concreteness, we focus on a particular class of probabilistic models, namely Hidden Markov Models (HMMs), and their application in molecular biology. To overcome the limitations of simple HMMs, we propose to use hybrid HMM/NN architectures[2] that combine the expressive power of artificial NNs with the sequential time series aspect of HMMs. 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 (Bourlard and Morgan 1994; Cho and Kim 1995). In many of these applications, however, NNs are used as front end processors to extract features, such as strokes, characters, and phonemes. HMMs are then used in higher processing stages for word and language modeling.[3] The HMM and NN components are often trained separately, although there are some exceptions (Bengio *et al.* 1995). A different type of hybrid architecture is also described in Cho and Kim (1995), where the NN component is used to classify the pattern of likelihoods produced by several HMMs. Here, in contrast, the HMM and NN components are inseparable. This yields, among other things, unified training algorithms where the HMM dynamic programming and the NN backpropagation blend together.

In what follows, we first briefly review HMMs, how they can be used to model protein families, and their limitations. In Section 3, we develop HMM/NN hybrid architectures for single models, to address the problem of parameter complexity and control or overfitting. Simulation results are presented in Section 4 for a simple HMM/NN hybrid architecture used

[1] Classical Bayesian hierarchical modeling relies on the description of a parameterized prior $P_\alpha(\theta)$, where α are the hyperparameters. This is related to the present situation $\theta = f(\vec{w})$, provided a prior $P(\vec{w})$ is defined on the new parameters.

[2] HMM/NN architectures were first described at a NIPS94 workshop (Vail, CO) and at the International Symposium on Fifth Generation Computer Systems (Tokyo, Japan), in December 1994. Preliminary versions were published in the Proceedings of the Symposium, and in the Proceedings of the ISMB95 Conference.

[3] In the molecular biology applications to be considered, NNs could conceivably be used to interpret the analog output of various sequencing machines, but this is definitely not the focus here.

to model a particular protein family (immunoglobulins). In Section 5, we discuss HMM/NN hybrid architectures for multiple models, to address the problem of long-range dependencies or underfitting.

2 HMMs of Protein Families

Many problems in computational molecular biology can be cast in terms of statistical pattern recognition and formal languages (Searls 1992). The increasing abundance of sequence data creates a favorable situation for machine learning approaches, where grammars are learned from the data. In particular, HMMs are equivalent to stochastic regular grammars and have been extensively used to model protein families and DNA coding regions (Baldi *et al.* 1994a,b; Krogh *et al.* 1994a; Baldi and Chauvin 1994a; Krogh *et al.* 1994b).

Proteins consist of polymer chains of amino acids. There are 20 important amino acids, so that proteins can be viewed as strings of letters, over a 20-letter alphabet. Protein sequences with a common ancestor share functional and structural properties, and can be grouped into families. Aligning sequences in a family is important, for instance to detect highly conserved regions, or motifs, with particular significance. Multiple alignment of highly divergent families where, as a result of evolutionary insertions and deletions, pairs of sequences often share less than 20% amino acids, is a highly nontrivial task (Meyers 1994).

A first-order discrete HMM can be viewed as a stochastic generative model 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 emitting symbols from the alphabet. 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. 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. One of the standard HMM architectures for protein applications (Krogh *et al.* 1994a), is the left-right architecture depicted in Figure 1. The alphabet has $M = 20$ symbols, one for each amino acid ($M = 4$ for DNA or RNA models, one symbol per nucleotide). 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 $\mathbf{S} = \{start, m_1, \ldots, m_N, i_1, \ldots, i_{N+1}, d_1, \ldots, d_N, end\}$. 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 $start \rightarrow m_1 \rightarrow m_2 \cdots \rightarrow m_N \rightarrow end$ is the backbone of the model. For each main state, corresponding insert and delete states are needed to model insertions and deletions. The self-loop on the insert states allows for multiple insertions at a given site.

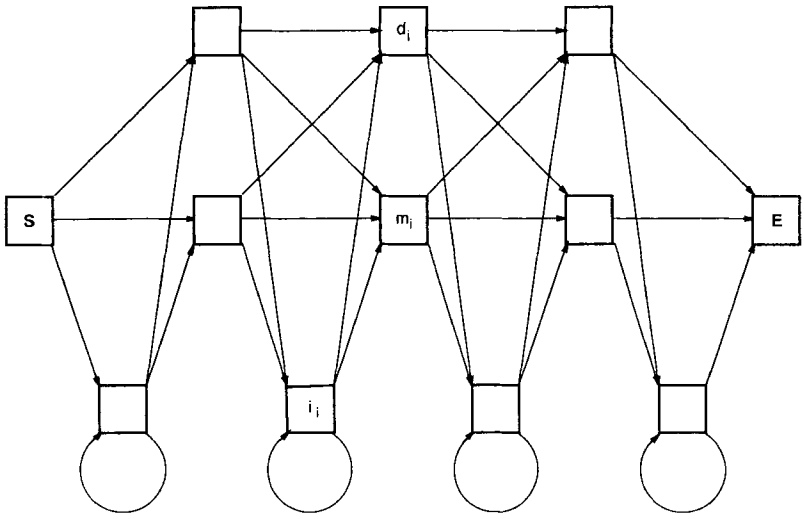

Figure 1: Example of HMM architecture used in protein modeling. S is the start state, E the end state. d_i, m_i, and i_i denote delete, main, and insert states, respectively.

2.1 Learning Algorithms. Given a sample of K training sequences $O_1, \ldots, O_K$, the parameters of an HMM can be iteratively modified, in an unsupervised way, to optimize the data fit according to some measure, usually based on the likelihood of the data. Since the sequences can be considered as independent, the overall likelihood is equal to the product of the individual likelihoods. Two target functions, commonly used for training, are the negative log-likelihood:

$$Q = -\sum_{k=1}^{K} Q_k = -\sum_{k=1}^{K} \ln P(O_k) \tag{2.1}$$

and the negative log-likelihood based on the optimal paths:

$$Q = -\sum_{k=1}^{K} Q_k = -\sum_{k=1}^{K} \ln P[\pi(O_k)] \tag{2.2}$$

where $\pi(O)$ is the most likely HMM production path for sequence O. $\pi(O)$ can be computed efficiently by dynamic programming (Viterbi algorithm). Depending on the situation, the Viterbi path approach can be considered as a fast approximation to the full maximum likelihood, or as an algorithm in its own right. This can be the case in protein modeling where, as described below, the optimal paths play an important role.

When priors on the parameters are included, one can also add regularizer terms to the objective functions for maximum a posteriori (MAP) estimation.

Different algorithms are available for HMM training, including the Baum-Welch or expectation–maximization (EM) algorithm, and different forms of gradient descent and other generalized EM (GEM) (Dempster *et al.* 1977; Rabiner 1989; Baldi and Chauvin 1994a) algorithms. In the Baum-Welch algorithm, the parameters are updated according to

$$t_{ij} = \frac{n_{ij}}{n_i} \qquad \text{and} \qquad e_{iX} = \frac{m_{iX}}{m_i} \tag{2.3}$$

where $m_i = \sum_Y m_{iY}$ (respectively $n_i = \sum_j n_{ij}$) and m_{iX} (respectively n_{ij}) are the normalized[4] expected emission (respectively transition) counts, induced by the data, that can be calculated using the forward–backward dynamic programming procedure (Rabiner 1989), or the Viterbi paths in Viterbi learning. As for gradient descent, and other GEM algorithms, a useful reparameterization (Baldi and Chauvin 1994b), in terms of normalized exponentials consists of

$$t_{ij} = \frac{e^{w_{ij}}}{\sum_k e^{w_{ik}}} \qquad \text{and} \qquad e_{iX} = \frac{e^{v_{iX}}}{\sum_Y e^{v_{iY}}} \tag{2.4}$$

with w_{ij} and v_{iX} as the new variables. This reparameterization has two advantages: (1) modification of the ws and vs automatically preserves normalization constraints on emission and transition distributions; and (2) transition and emission probabilities can never reach the absorbing value 0. The on-line gradient descent equations on the negative log-likelihood are then

$$\Delta w_{ij} = \eta(n_{ij} - n_i t_{ij}) \qquad \text{and} \qquad \Delta v_{iX} = \eta(m_{iX} - m_i e_{iX}) \tag{2.5}$$

where η is the learning rate. The variables n_{ij}, n_i, m_{iX}, m_i are again the expected counts derived by the forward–backward procedure, for each single sequence if the algorithm is to be used on-line. Similarly, in Viterbi learning, at each step along a Viterbi path, and for any state i on the path, the parameters of the model are updated according to

$$\Delta w_{ij} = \eta(T_{ij} - t_{ij}) \qquad \text{and} \qquad \Delta v_{iX} = \eta(E_{iX} - e_{iX}) \tag{2.6}$$

$T_{ij} = 1$ (respectively $E_{iX} = 1$) if the $i \to j$ transition (respectively emission of X from i) is used, and 0 otherwise. 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.

[4]Unlike in Baldi and Chauvin (1994b), throughout this paper we use the more classical notation of (Rabiner 1989) where the counts, for a given sequence, automatically incorporate a normalization by the probability $P(O)$ of the sequence itself.

Regardless of the training method, once an HMM has been successfully trained on a family of sequences, it can be used in a number of tasks. For instance, for any given sequence, one can compute its most likely path, as well as its likelihood. A multiple alignment results immediately from aligning all the optimal paths. The likelihoods can be used for discrimination tests and data base searches (Krogh *et al.* 1994a; Baldi and Chauvin 1994a). In the case of proteins, HMMs have been successfully applied to several families such as globins, immunoglobulins, kinases, and G-protein-coupled receptors. In most cases, HMMs have performed well on all tasks yielding, for instance, multiple alignments that are comparable to those derived by human experts.

2.2 Limitations of HMMs. In spite of their success in various applications, HMMs can suffer from two weaknesses. First, they often have a large number of unstructured parameters. In the case of protein models, the architecture of Figure 1 has a total of approximately $49N$ parameters ($40N$ emission parameters and $9N$ transition parameters). For a typical protein family, N is of the order of a few hundreds, resulting immediately in models with over 10,000 free parameters. This can lead to overfitting when only a few sequences are available,[5] not an uncommon situation in early stages of genome projects. Second, first-order HMMs are limited with respect to dependencies between hidden states, found in most interesting problems. Proteins, for instance, fold into complex 3D shapes, essential to their function. Subtle long-range correlations in their polypeptide chains may exist that are not accessible to a single HMM. For instance, assume that whenever X is found at position i, it is generally followed by Y at position j; and whenever X' is found at position i, it tends to be followed by Y' at j. A single HMM has typically two *fixed* emission vectors associated with the i and j positions. Therefore it cannot capture such correlations. Related problems are also the nonstationarity of complex time series, as well as the variability often encountered in "speaker-independent" recognition problems. Only a small fraction of distributions over the space of possible sequences, essentially the factorial distributions, can be represented by a reasonably constrained HMM.[6]

3 HMM/NN Hybrid Architectures: Single Model Case

3.1 Basic Idea. In a general HMM, an emission or transition vector θ is a function of the state i only: $\theta = f(i)$. The first basic idea is to have

[5] It should be noted, however, that a typical sequence provides on the order of $2N$ constraints, and 25 sequences or so provide a number of examples in the same range as the number of HMM parameters.

[6] Any distribution can be represented by a single *exponential size* HMM, with a start state connected to different sequences of deterministic states, one for each possible alphabet sequence, with a transition probability equal to the probability of the sequence itself.

a NN on top of the HMM, for the computation of the HMM parameters, that is for the computation of the function f. NNs are universal approximators, and, therefore, can represent any f. More importantly perhaps, NN representations enable the flexible introduction of many possible constraints. For simplicity, we discuss emission parameters only, but the approach extends immediately to transition parameters as well.

In the reparameterization of 2.4, we can consider that each one of the HMM emission vectors is calculated by a small NN, with one input set to one (bias), no hidden layers, and 20 softmax output units (Fig. 2a). The connections between the input and the outputs are the v_{iX}. This can be generalized immediately by having arbitrarily complex NNs, for the computation of the HMM parameters. The NNs associated with different states can also be linked with one or several common hidden layers, the overall architecture being dictated by the problem at hand. In the case of a discrete alphabet however, such as for proteins, the emission of each state is a multinomial distribution, and, therefore, the output of the corresponding network should consist of M softmax units.

As a simple example, consider the hybrid HMM/NN architecture of Figure 2b consisting of the following:

1. Input layer: one unit for each state i. At each time, all units are set to 0, except one which is set to 1. If unit i is set to 1, the network computes e_{iX}, the emission distribution of state i.
2. Hidden layer: H hidden units indexed by h, each with transfer function f_h (logistic by default) with bias b_h ($H < M$).
3. Output layer: M softmax units or weighted exponentials, indexed by X, with bias b_X.
4. Connections: $\alpha = (\alpha_{hi})$ connects input position i to hidden unit h. $\beta = (\beta_{Xh})$ connects hidden unit h to output unit X.

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

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

The corresponding activity in the output layer is

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

For hybrid HMM/NN architectures, a number of points are worth noticing:

- The HMM states can be partitioned into different groups, with different networks for different groups. In protein applications, for instance, one can use different NNs for insert states and for main states, or for different groups of states along the protein sequence corresponding, for instance, to different regions (hydrophobic, hydrophilic, alpha-helical, etc.).

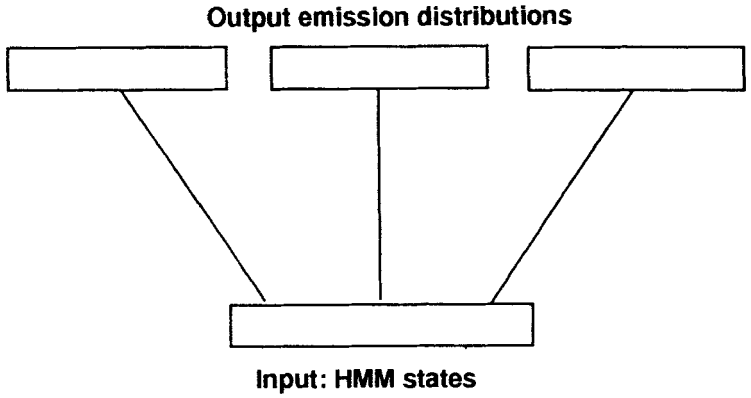

Fig. 2a

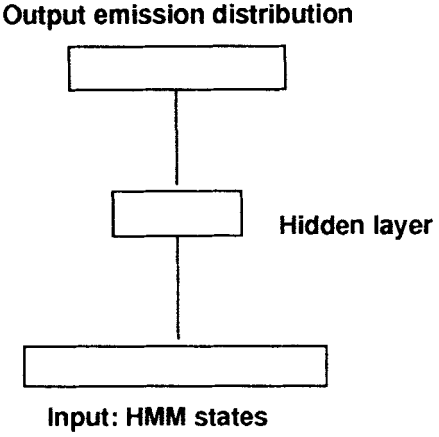

Fig. 2b

Figure 2: (a) Schematic representation of simple HMM/NN hybrid architecture used in Baldi *et al.* (1994b). Each HMM state has its own NN. Here, the NNs are extremely simple, with no hidden layer, and an output layer of softmax units computing the state emission, or transition, parameters. Only output emissions are represented for simplicity. (b) Schematic representation of an HMM/NN architecture where the NNs associated with different states (or different groups of states) are connected via one or several hidden layers.

- HMM parameter reduction can easily be achieved using small hidden layers with H hidden units, and H small compared to N or M. In the example of Figure 2b, with H hidden units and considering only main states, the number of parameters is $H(N + M)$ in the HMM/NN architecture, versus NM in the corresponding simple HMM. For protein models, this yields roughly HN parameters for the HMM/NN architecture, versus 20N for the simple HMM. $H = M$ is equivalent to 2.4.
- The number of parameters can be adaptively adjusted to variable training set sizes, merely by changing the number of hidden units. This is useful in environments with large variations in data base sizes, as in current molecular biology applications.
- The entire bag of connectionist tricks can be brought to bear on these architectures, such as radial basis functions, multiple hidden layers, sparse connectivity, weight sharing, gaussian priors, and hyperparameters. Several initializations and structures can be implemented in a flexible way. For instance, by allocating different numbers of hidden units to different subsets of emissions or transitions, it is easy to favor certain classes of paths in the models, when needed. In the HMM of Figure 1, for instance, one must introduce a bias favoring main states over insert states, prior to any learning. It is easy also to tie different regions of a protein that may have similar properties by weight sharing, and other types of long-range correlations. By setting the output bias to the proper values, the model can be initialized to the average composition of the training sequences, or any other useful distribution.
- Classical prior information in the form of substitution matrices, for instance, is easily incorporated. Substitution matrices (Altschul 1991) can be computed from data bases, and essentially produce a background probability matrix $P = (p_{XY})$, where p_{XY} is the probability that X be changed into Y over a certain evolutionary time. P can be implemented as a linear transformation in the emission NN.
- HMMs with continuous emission distributions are also easy to incorporate in the HMM/NN framework. The output emission distributions can be represented, for instance, in the form of samples, moments, and/or mixture coefficients. In the classical mixture of gaussians case, means, covariances, and mixture coefficients can be computed by the NN. Likewise, additional HMM parameters, such as exponential parameters to model the duration of stay in any given state, can be calculated by a NN.

With hybrid HMM/NN architectures, in general the M step of the EM algorithm, cannot be carried analytically. One can still use, however, some form of gradient descent using the chain rule, by computing the derivatives of the target likelihood functions 2.1 or 2.2 with respect to the HMM parameters, and then the derivatives of the HMM parameters

with respect to the NN parameters. For completeness, a derivation of the learning equations for the HMM/NN architecture described above is given in the Appendix. In the resulting learning equations (A.3 and A.7), the HMM dynamic programming and the NN backpropagation components are intimately fused. These algorithms can also be seen as GEM (generalized EM) (Dempster *et al.* 1977) algorithms. They can easily be modified to MAP optimization with inclusion of priors.

3.2 Representation in Simple HMM/NN Architectures. Consider the particular HMM/NN described above, where a subset of the HMM states are fully connected to H hidden units, and the hidden units are fully connected to M softmax output units. The hidden unit bias is not really necessary in the sense that for any HMM state i, any vector of biases b_{hi}, and any vector of connections α_{hi}, there exists a new vector of connections α'_{hi} that produces the same vector of hidden unit activations with 0 bias. This is not true in the general case, for instance, as soon as there are multiple hidden layers, or if the input units are not fully interconnected to the hidden layer. We have left the biases for the sake of generality, and also because even if the biases do not enlarge the space of possible representations, they may still facilitate the learning procedure. Similar remarks hold more generally for the transfer functions. With an input layer fully connected to a single hidden layer, the same hidden layer activation can be achieved with different activation functions, by modifying the weights.

A natural question to ask is what is the representation used in the hidden layer, and what is the space of emission distributions achievable in this fashion? Each HMM state in the network can be represented by a point in the $[-1,1]^H$ hypercube. The coordinates of a point are the activities of the H hidden units. By changing its connections to the H hidden units, an HMM state can occupy any position in the hypercube. So, the space of achievable emission distributions is entirely determined by the connections from the hidden to output layer. If these connections are held fixed, then each HMM state can select a corresponding optimal position in the hypercube, where its emission distribution, generated by the NN weights, is as close as possible to the truly optimal distribution, for instance in cross-entropy distance. During on-line learning, all parameters are learned at the same time so this may introduce additional effects.

To further understand the space of achievable distributions, consider the transformation from hidden to output units. For notational convenience, we introduce one additional hidden unit numbered 0, always set to 1, to express the output biases in the form: $b_X = \beta_{X0}$. If, in this extended hidden layer, we turn a single hidden unit to 1, one at a time, we obtain $H+1$ different emission distributions in the output layer $P^h = (p^h_X)$

$(0 \leq h \leq H)$ with

$$p_X^h = \frac{e^{-\beta_{Xh}}}{\sum_Y e^{-\beta_{Yh}}} \tag{3.3}$$

Consider now a general pattern of activity in the hidden layer of the form $(1, \mu_1, \ldots, \mu_H)$. Using 3.2 and 3.3, the emission distribution in the output layer is then

$$e_{iX} = \frac{e^{-\sum_{h=0}^{H} \beta_{Xh}\mu_h}}{\sum_Y e^{-\sum_{h=0}^{H} \beta_{Yh}\mu_h}} = \frac{\prod_{h=0}^{H} [p_X^h]^{\mu_h} [\sum_Y e^{-\beta_{Yh}}]^{\mu_h}}{\sum_Y \prod_{h=0}^{H} [p_Y^h]^{\mu_h} [\sum_Z e^{-\beta_{Zh}}]^{\mu_h}} \tag{3.4}$$

After simplifications, this yields

$$e_{iX} = \frac{\prod_{h=0}^{H} [p_X^h]^{\mu_h}}{\sum_Y \prod_{h=0}^{H} [p_Y^h]^{\mu_h}} \tag{3.5}$$

Therefore, all the achievable emission distributions by the NN have the form of 3.5, and can be viewed as "combinations" of $H + 1$ fundamental distributions P^h associated with each single hidden unit. In general, this combination is different from a convex linear combination of the P^hs. It consists of three operations: (1) raising each component of P^h to the power μ_h, the activity of the hth hidden unit, (2) multiplying all the corresponding vectors componentwise, and (3) normalizing. In this form, the hybrid HMM/NN approach is different from a mixture of Dirichlet distributions approach.

4 Simulation Results

Here we demonstrate a simple application of the principles behind HMM/NN hybrid architectures on the immunoglobulin protein family. Immunoglobulins, or antibodies, are proteins produced by B cells that bind with specificity to foreign antigens in order to neutralize them, or target their destruction by other effector cells. The various classes of immunoglobulins are defined by pairs of light and heavy chains that are held together principally by disulfide bonds[7] (Fig. 3). Each light and heavy chain molecule contains one variable (V) region, and one (light) or several (heavy) constant (C) regions. The V regions differ among immunoglobulins, and provide the specificity of the antigen recognition. About one-third of the amino acids of the V regions form the hypervariable sites, responsible for the diversity of the vertebrate immune response. Our data base is the same as the one used in Baldi *et al.* (1994b), and consists of human and mouse heavy chain immunoglobulin V region sequences, from the Protein Identification Resources (PIR) data base. It contains 224 sequences, with minimum length 90, average length $N = 117$, and maximum length 254.

[7]Disulfide bonds are covalent bonds between two sulfur atoms in different amino acids (typically cysteines) of a protein that are important in determining secondary and tertiary structure.

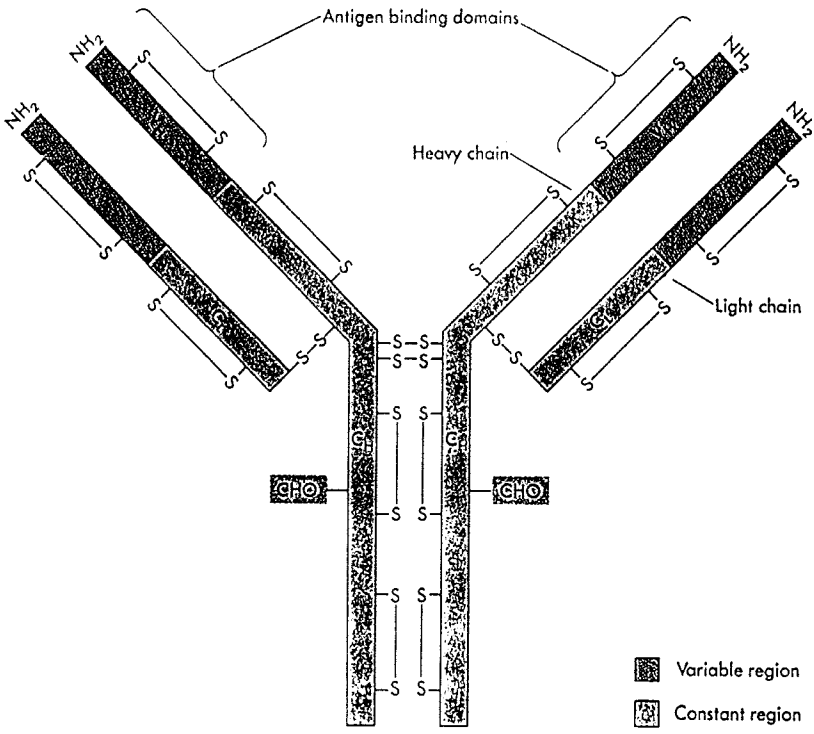

Figure 3: A model of the structure of a typical human antibody molecule, composed of two light and two heavy polypeptide chains. Interchain and intrachain disulfide bonds are indicated. Cysteine (C) residues are associated with the bonds. Two identical active sites for antigen binding, corresponding to the variable regions, are located in the arms of the molecule. (From *Molecular Biology of the Gene*. Vol. II. Fourth Edition, by Watson *et al.* Copyright © 1987 by James D. Watson. Published by The Benjamin/Cummings Publishing Company.)

For the immunoglobulin V regions, our results (Baldi *et al.* 1994b) were obtained by training a simple HMM, similar to the one in Figure 1, containing a total of $52N + 23 = 6107$ adjustable parameters. Here we train a hybrid HMM/NN architecture with the following characteristics. The basic model is an HMM with the architecture of Figure 1. All the main state emissions are calculated by a common NN, with 2 hidden units. Likewise, all the insert state emissions are calculated by a common NN, with one hidden unit only. Each state transition distri-

bution is calculated by a different softmax network, as in our previous work. With edge effects neglected, the total number of parameters of this HMM/NN architecture is 1507 ($117 \times 3 \times 3 = 1053$ for the transitions, $(117 \times 3 + 3 + 3 \times 20 + 40) = 454$ for the emissions, including biases). This architecture is not at all optimized: for instance, we suspect we could have significantly reduced the number of transition parameters. Our goal at this time is only to demonstrate the general HMM/NN principles, and test the learning algorithm.

The hybrid architecture is then trained on-line, using both gradient descent (A.3), and the Viterbi version (A.7). The training set consists of a random subset of 150 sequences, identical to the training set used previously. There, emission and transition parameters were initialized uniformly. Here, the input-to-hidden weights are initialized with independent gaussians, with mean 0 and standard deviation 1. The hidden-to-output weights are initialized to 1. This yields a uniform emission probability distribution on all the emitting states.[8] Notice also that if all the weights are initialized to 1, including those from input to hidden layer, then the hidden units cannot differentiate from each other. The transition probabilities out of insert or delete states are initialized uniformly to 1/3. We introduce, however, a small bias along the backbone that favors main to main transitions, in the form of a Dirichlet prior. This prior is equivalent to introducing a regularization term in the objective function, equal to the logarithm of the backbone transition path. The regularization constant is set to 0.01, and the learning rate to 0.1. Typically, 10 training cycles are more than sufficient to reach equilibrium.

In Figure 4, we display the multiple alignment of 20 immunoglobulin sequences, selected randomly from both the training and validation sets. The validation set consists of the remaining 74 sequences. This alignment is very stable between 5 and 10 epochs.[9] It corresponds to a model trained by A.7 for 10 epochs. While there is currently no universally accepted measure of the quality of an alignment, the present alignment is similar to the previous one, derived with a simple HMM with more than four times as many parameters. The algorithm has been able to detect most of the salient features of the family. Most importantly, the cysteine residues (C) toward the beginning and the end of the region (positions 24 and 100 in this alignment), which are responsible for the disulfide bonds that hold the chains, are perfectly aligned. The only exception is the last sequence (PH0097), which has a serine (S) residue in its terminal portion. This is a rare but recognized exception to the conservation of this position. Some of the sequences in the family came with a "header" (transport signal peptide). We did not remove the headers

[8] With Viterbi learning, this is probably better than a nonuniform initialization, such as the average composition. A nonuniform initialization may introduce distortions in the Viterbi paths.

[9] Differences with the alignment published in the ISMB95 Proceedings result from differences in regularization, and not in the number of training cycles.

prior to training. The model is capable of detecting and accommodating these headers, by treating them as initial repeated inserts, as can be seen from the alignment of three of the sequences (S09711, A36194, S11239). This multiple alignment contains also a few isolated problems, related in part to the overuse of gaps and insert sates. Interestingly, this is most evident in the hypervariable regions, for instance at positions 30–35 and 50–55. These problems should be eliminated with a more careful selection of hybrid architecture and/or regularization. Alignments did not improve using A.3 and/or a larger number of hidden units, up to 4.

In Figure 5, we display the activity of the two hidden units associated with each main state (see 3.2). For most states, at least one of the activities is saturated. The activities associated with the cysteine residues responsible for the disulfide bridges (main states 24 and 100) are all saturated, and in the same corner $(-1.+1)$. Points close to the center (0.0) correspond to emission distributions determined by the bias only. For the main states, the three emission distributions of equation 3.3, associated with the bias and the two hidden units, are given by

$$\begin{aligned} P^0 &= (0.442.0.000.0.005.0.000.0.001.0.000.0.004.0.002.0.133.\\ &\quad 0.000.0.000.0.000.0.000.0.113.0.195.0.000.0.104.0.001.\\ &\quad 0.000.0.000) \\ P^1 &= (0.000.0.000.0.000.0.036.0.000.0.900.0.000.0.000.0.000.\\ &\quad 0.000.0.000.0.000.0.037.0.000.0.000.0.000.0.000.\\ &\quad 0.000.0.027) \end{aligned}$$

and

$$\begin{aligned} P^3 &= (0.000.0.040.0.000.0.000.0.000.0.000.0.000.0.000.0.000.\\ &\quad 0.942.0.001.0.000.0.016.0.000.0.000.0.000.0.000.0.001.\\ &\quad 0.000.0.000) \end{aligned}$$

using alphabetical order on single-letter amino acid symbols.

5 Discussion: The Case of Multiple Models

The hybrid HMM/NN architectures described address the first limitation of HMMs: the control of model structure and complexity. No matter how complex the NN component, however, the final model so far remains a single HMM. Therefore the second limitation of HMMs, long-range dependencies and underfitting, remains. This obstacle cannot be overcome by simply resorting to higher-order HMMs. Most often these are computationally intractable.

A possible approach is to try to introduce a new state for each relevant context. This requires a systematic method for determining relevant contexts of variable lengths, directly from the data. Furthermore, one must

Figure 4: Multiple alignment of 20 immunoglobulin sequences, randomly extracted from the training and validation data sets. Validation sequences: F37262, G1HUDW, A36194, A31485, D33548, S11239, I27888, A33989, A30502. Alignment is obtained with a hybrid HMM/NN architecture trained for 10 cycles, with two hidden units for the main state emissions, and one hidden unit for the insert state emissions. Lower case letters correspond to emissions from insert states. Notice the initial header (transport signal peptide) on some of the sequences, captured as repeated transitions through the first insert state in the model. The cysteines (C), associated with the disulfide bridge, in columns 24 and 100, are perfectly aligned (PH0097 is a known biological exception).

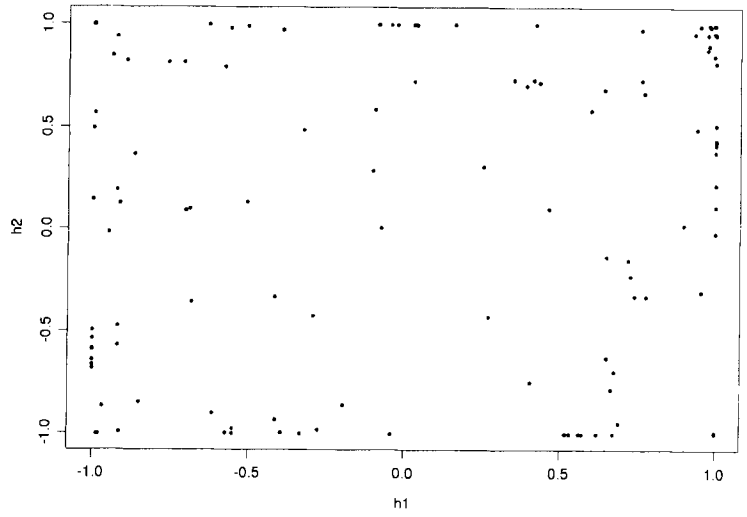

Figure 5: Activity of the two hidden units associated with the emission of the main states. The two activities associated with the cysteines (C) are in the upper left corner, almost overlapping, with coordinates $(-1, +1)$.

hope the number of relevant contexts remains small. An interesting approach along these lines can be found in Ron *et al.* (1994), where English is modeled as a Markov process with variable memory length of up to 10 letters or so.

To address the second limitation without resorting to a different model class, one must consider more general HMM/NN hybrid architectures, where the underlying statistical model is a *set* of HMMs. To see this, consider again the $X - Y/X' - Y'$ problem. To capture such dependencies requires *variable* emission vectors at the corresponding locations, together with a linking mechanism. In this simple case, four different emission vectors are needed: e_i, e_j, e'_i and e'_j. Each one of these vectors must assign a high probability to the letters X, Y, X', and Y', respectively. More importantly, there must be some kind of memory, so that e_i and e_j are used for sequence O, and e'_i and e'_j are used for sequence O'. The combination of e_i and e'_j (or e'_i and e_j) should be rare or not allowed, unless required by the data. Thus e_i and e_j must belong to a first HMM, and e'_i and e'_j to a second HMM, with the possibility of switching from one HMM to the other, as a function of input sequence. Alternatively,

there must be a single HMM, but with variable emission distributions, modulated again by some input.

In both cases then, we consider that the emission distribution of a given state depends not only on the state itself, but also on an additional stream of information I. That is now $\theta = f(i, I)$. In a multiple HMM/NN hybrid architecture, f can be computed again by a NN. Depending on the problem, the input I can assume different forms, and may be called "context" or "latent variable." When feasible, I may even be equal to the currently observed sequence O. Other inputs are, however, possible, over different alphabets. An obvious candidate in protein modeling tasks would be the secondary structure of the protein (α-helices, β-sheets, and coils). In general, I could also be any other array of numbers representing latent variables for the HMM modulation (MacKay 1994). We shall now describe, without any simulations, two simple but somewhat canonical architectures of this sort. Learning is briefly discussed in the Appendix.

5.1 Example 1: Mixtures of HMM Experts. A first possible approach is to put an HMM mixture distribution on the sequences. With n HMMs $M_1, \ldots, M_n$,

$$P(O) = \sum_{i=1}^{n} \lambda_i P_{M_i}(O) \tag{5.1}$$

where $\sum_i \lambda_i = 1$, and λs are the mixture coefficients. Similarly, the Viterbi likelihood is $\max_i \lambda_i P[\pi_{M_i}(O)]$. In generative mode, sequences are produced at random by each individual HMM, and M_i is selected with probability λ_i. Such a system can be viewed as a larger single HMM, with a starting state connected to each one of the HMMs M_i, with transition probability λ_i (Fig. 6). This type of model is used in Krogh *et al.* (1994a) for unsupervised classification of globin protein sequences. Notice that the parameters of each submodel can be computed by an NN to create an HMM/NN hybrid architecture. Since the HMM experts form a larger single HMM, the corresponding hybrid architecture is also identical to what we have seen in the section on single HMMs. The only peculiarity is that states have been replicated, or grouped, to form different submodels. One further step is to have variable mixture coefficients that depend on the input sequence, or some other relevant information. These mixture coefficients can be computed as softmax outputs of an NN, as in the mixture of experts architecture of Jacobs *et al.* (1991).

5.2 Example 2: Mixtures of Emission Experts. A different approach is to modulate a single HMM by considering that the emission parameters e_{iX} should also be function of the additional input I. So $e_{iX} = P(i, X, I)$.

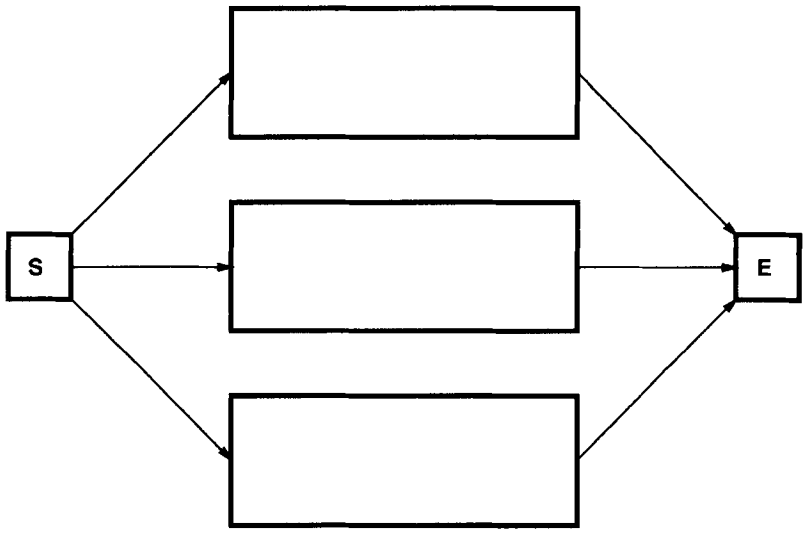

Figure 6: Schematic representation of the type of multiple HMM architecture used in Krogh *et al.* (1994a) for discovering subfamilies within a protein family. Each "box," between the start and end states, corresponds to an HMM with the architecture of Figure 1.

Without any loss of generality, we can assume that P is a mixture of n emission experts P_j:

$$P(i.X.I) = \sum_{j=1}^{n} \lambda_j(i.X.I)P_j(i.X.I) \tag{5.2}$$

In many interesting cases, λ_j is independent of X, resulting in the probability vector equation, over the alphabet:

$$P(i.I) = \sum_{j=1}^{n} \lambda_j(i.I)P_j(i.I) \tag{5.3}$$

If $n = 1$ and $P(i.I) = P(i)$, we are back to a single HMM. An important special case is derived by further assuming that λ_j does not depend on i, and $P_j(i.X.I)$ does not depend on I explicitly. Then

$$P(i.I) = \sum_{j=1}^{n} \lambda_j(I)P_j(i) \tag{5.4}$$

This provides a principled way for designing the top layers of general hybrid HMM/NN architectures, such as the one depicted in Figure 7.

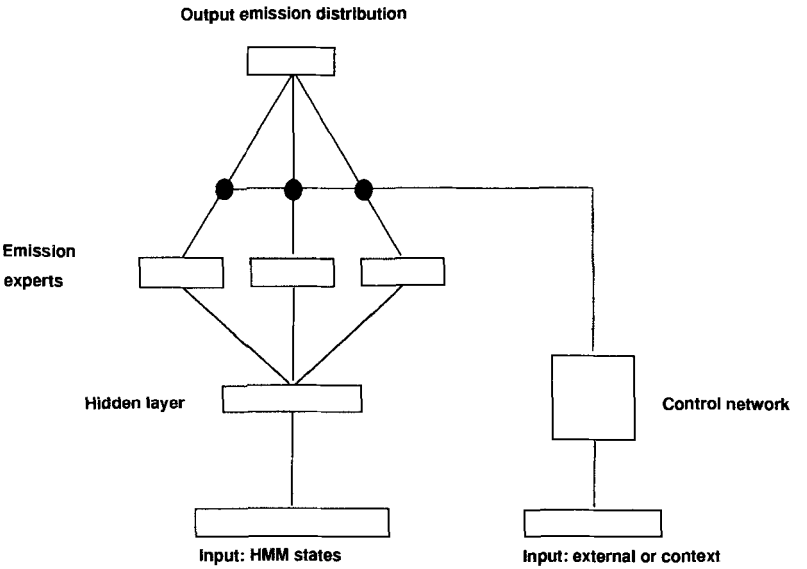

Figure 7: Schematic representation of a general HMM/NN architecture, where the HMM parameters are computed by an NN of arbitrary complexity, that operates on state information, but also on input or context. The input or context is used to modulate the HMM parameters, for instance, by switching or mixing different parameter experts. For simplicity, only emission parameters are represented, with three emission experts, and a single hidden layer. Connections from the HMM states to the control network, and from the input to the hidden layer, are also possible.

The components P_j are computed by a NN, and the mixture coefficients by another gating NN. Naturally, many variations are possible and, in the most general case, the switching network can depend on the state i, and the distributions P_j on the input I. In the case of protein modeling, for instance, if the switching depends on position i, the emission experts could correspond to different types of regions, such as hydrophobic and hydrophilic, rather than different subclasses within a protein family.

6 Conclusion

A large class of hybrid HMM/NN architectures has been described. These architectures improve on single HMMs in two complementary directions. First, the NN reparameterization provides a flexible tool for the

control of overfitting, the introduction of priors, and the construction of an input-dependent mechanism for the modulation of the final model. Second, modeling a data set with multiple HMMs allows for the coverage of a larger set of distributions, and the expression of non-stationarity and correlations inaccessible to single HMMs. We recently found out that related ideas have been proposed independently in Bengio and Frasconi (1995), but from a different viewpoint in terms of input/output HMMs. Not surprisingly, these ideas are also related to data compression, information complexity, factorial codes, autoencoding and generative models [for instance, Dayan *et al.* (1995), and references therein].

The concept of hybrid HMM/NN architecture has been demonstrated, in its simplest form, by providing a model of the immunoglobulin family. The HMM/NN approach is meant to complement rather than substitute many of the already existing techniques for incorporating prior information in sequence models. Additional work is required to develop optimal architectures and learning algorithms, and to test them on more challenging protein families and other domains.

Two important issues for the success of a hybrid HMM/NN architecture on a real problem are the design of the NN architecture, and the selection of the external input or context. These issues are problem dependent and cannot be dealt with generally. We have described some examples of architectures using mixture ideas for the design of the NN component. Different input choices are possible, such as contextual information or latent variables, sequences over a different alphabet (for instance strokes versus letters in handwriting recognition), or just real vectors, in the case of manifold parameterization (MacKay 1994).

As pointed out in the introduction, the ideas presented here are not limited to HMMs, or to protein or DNA modeling. They can be viewed in a more general framework, where a class of parameterized model is first constructed for the data, and then the parameters of the models are calculated, and possibly modulated, by one or several other NNs (or any other flexible reparameterization). In fact, several examples of simple hybrid architectures can be found scattered throughout the literature. A classical case consists of binomial (respectively multinomial) classification models, where membership probabilities are calculated by a NN with a sigmoidal (respectively normalized exponential) output (Rumelhart *et al.* 1995). Other examples are the master-slave approach of Lapedes and Farber (1986), and the sigmoidal belief networks in Neal (1992), where NNs are used to compute the weights of another NN, or the conditional distributions of a belief network. Although the principle of hybrid modeling is not new, by exploiting it systematically in the case of HMMs, we have generated new classes of models. There are other classes where the principle has not been applied systematically yet. As an example, it is well known that HMMs are equivalent to stochastic regular grammars. The next level in the Chomsky hierarchy is context-free grammars (SCFGs). One can consider hybrid SCFG/NN architectures, where a NN

is used to compute the parameters of a SCFG, and/or to modulate or mix different SCFGs. Such hybrid grammars might be useful, for instance, in extending the work of Sakakibara *et al.* (1994), on RNA modeling. Finding optimal architectures for molecular biology applications and other domains, and developing a better understanding of how probabilistic models should be NN-modulated as a function of input or context, are some of the main challenges for hybrid approaches.

7 Appendix

7.1 Learning for Simple HMM/NN Architectures. Here we give on-line equations (batch equations can be derived similarly). 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.

7.1.1 Gradient Learning on Full Likelihood. Let $Q(O) = \ln P(O)$. If $m_{iX}(O)$ is the normalized count for the emission of X from i for O, derived using the forward–backward algorithm (Baldi and Chauvin 1994b) then

$$\frac{\partial P(O)}{e_{iX}} = \frac{m_{iX}(O)}{e_{iX}} P(O) \tag{A.1}$$

so that

$$\frac{\partial Q}{\partial e_{iX}} = \sum_O \frac{m_{iX}(O)}{e_{iX}} \tag{A.2}$$

The partial derivatives with respect to the network parameters α, β, and b can be obtained by the chain rule, that is by backpropagating through the network for each i. For each O and i, the resulting on-line learning equations are

$$\begin{cases} \Delta\beta_{Xlh} &= \eta[m_{iX}(O) - e_{iX}m_i] f_h(\alpha_{lhi} + b_h) \\ \Delta b_X &= \eta[m_{iX}(O) - e_{iX}m_i] \\ \Delta\alpha_{lhj} &= \delta_{ij}\eta f_h'(\alpha_{lhi} + b_h)\{\sum_Y [m_{iY}(O) - e_{iY}m_i]\beta_{Ylh}\} \\ \Delta b_h &= \eta f_h'(\alpha_{lhi} + b_h)[\sum_Y (m_{iY}(O) - e_{iY}m_i)\beta_{Ylh}] \end{cases} \tag{A.3}$$

with $\delta_{ii} = 1$, and $\delta_{ij} = 0$ for $j \neq i$. The full gradient results by summing over all sequences, and all main states. For instance,

$$\frac{\partial Q}{\partial \alpha} = \sum_O \sum_{i=1}^{N} \frac{\partial Q(O)}{\partial \alpha}(i) \tag{A.4}$$

and similarly for β, and the biases. It is worth noticing that these equations are slightly different from those obtained by gradient descent on the local cross-entropy between the emission distribution e_{iX} and the target distribution m_{iX}/m_i.

7.1.2 Viterbi Learning. Here $Q(O) = \ln P[\pi(O)]$. 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

$$-\sum_{(i,X)\in\pi} \ln e_{iX} = -\sum_{(i,X)\in\pi} T_{iX} \ln \frac{e_{iX}}{T_{iX}} = -\sum_{i\in\pi}\sum_{Y} T_{iY} \ln \frac{e_{iY}}{T_{iY}} \tag{A.5}$$

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

$$H(T, E) = -\sum_{i\in\pi} H(T_i, e_i) = -\sum_{i\in\pi}\sum_{Y} T_{iY} \ln \frac{e_{iY}}{T_{iY}} \tag{A.6}$$

between the target output and the output of the network, over all i in π. This cross-entropy error function, combined with the softmax output unit, is the standard NN framework for multinomial classification (Rumelhart *et al.* 1995). In summary, the relevant derivatives can be calculated on-line both with respect to the sequences $O_1, \ldots, O_K$ 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 backpropagation on $H(T_i, e_i)$. The Viterbi on-line learning equations, similar to (A.3), are given by

$$\begin{cases} \Delta\beta_{Yh} &= \eta(T_{iY} - e_{iY}) f_h(\alpha_{hi} + b_h) \\ \Delta b_Y &= \eta(T_{iY} - e_{iY}) \\ \Delta\alpha_{hj} &= \delta_{ij}\eta f_h'(\alpha_{hi} + b_h)[\sum_Y \beta_{Yh}(T_{iY} - e_{iY})] \\ \Delta b_h &= \eta f_h'(\alpha_{hi} + b_h)[\beta_{Xh}(1 - e_{iX}) - \sum_{Y\neq X} \beta_{Yh} e_{iY}] \end{cases} \tag{A.7}$$

for $(i, X) \in \pi(O)$, with $T_{iX} = 1$, and $T_{iY} = 0$ for $Y \neq X$. The full gradient is obtained again by summing over all sequences, and all main states present in the corresponding Viterbi paths. For instance,

$$\frac{\partial Q}{\partial \alpha} = \sum_{O}\sum_{i\in\pi(O)} \frac{\partial H(T_i, e_i)}{\partial \alpha} \tag{A.8}$$

and similarly for β and the biases.

7.2 Learning for General HMM/NN Architectures. For a given setting of all the parameters, for a given observation sequence, and for a given input vector I, the general HMM/NN hybrid architectures reduce to a single HMM. The likelihood of a sequence, or some other measure of its fitness, with respect to such HMM, can be computed by dynamic programming. As long as it is differentiable in the model parameters, we can then backpropagate the gradient through the NN, including through the portion of the network depending on I, such as the gating network of Figure 4. With minor modifications, this leads to learning algorithms

similar to those described above. This form of learning encourages cooperation between the emission experts of Figure 7. As in the usual mixture of experts architecture of Jacobs *et al.* (1991), it may be useful to introduce some degree of competition between the experts, so that each one of them specializes, for instance, on a different subclass of sequences.

When the relevant input or hidden variable I is not known, it can be learned together with the model parameters using Bayesian inversion. Indeed, consider for instance the case where there is an input I associated with each observation sequence O, and a hybrid model with parameters w, so that we can compute $P(O \mid I, w)$. Let $P(I)$ and $P(w)$ denote our priors on I and w. Then

$$P(I \mid O, w) = \frac{P(O \mid I, w)P(I)}{P(O \mid w)} \tag{A.9}$$

with

$$P(O \mid w) = \int_I P(O \mid I, w)P(I)\, dI \tag{A.10}$$

The probability of the model parameters, given the data, can then be calculated, using Bayes theorem again:

$$P(w \mid D) = \frac{P(D \mid w)P(w)}{P(D)} = \frac{[\Pi_O P(O \mid w)]P(w)}{P(D)} \tag{A.11}$$

assuming the observations are independent. These parameters can be optimized by gradient descent on $-\log P(w \mid D)$. The main step is the evaluation of the likelihood $P(O \mid w)$ (A.10), and its derivatives with respect to w, which can be done by Monte Carlo sampling. The distribution on the latent variables I is calculated by A.9. The work of MacKay (1994) is an example of such a learning approach. The density network used for protein modeling can be viewed essentially as a special case of HMM/NN hybrid architecture, where each emission vector acts as a softmax transformation on a low-dimensional real "hidden" input I, with independent gaussian priors on I and w. The input I modulates the emission vectors, and therefore the underlying HMM, as a function of sequence.

7.3 Priors. There are many ways to introduce priors in HMMs. Additional work is required to compare them to the present methods. For instance, it is natural to use Dirichlet priors (Krogh *et al.* 1994a) on multinomial distributions, such as emission vectors over discrete alphabets. It is easy to check that if a multinomial distribution is calculated by a set of normalized exponential output units, a gaussian prior on the weights of these units is in general *not* equivalent to a Dirichlet prior on the outputs.

Acknowledgments

The work of P.B. is supported by a grant from the ONR. The work of Y.C. is supported in part by Grant R43 LM05780 from the National Library of Medicine. The contents of this publication are solely the responsibility of the authors and do not necessarily represent the official views of the National Library of Medicine.

References

Altschul, S. F. 1991. Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.* **219**, 1–11.

Baldi, P., and Chauvin, Y. 1994a. Hidden Markov models of the G-protein-coupled receptor family. *J. Comp. Biol.* **1**(4), 311–335.

Baldi, P., and Chauvin, Y., 1994b. Smooth on-line learning algorithms for hidden Markov models. *Neural Comp.* **6**(2), 305–316.

Baldi, P., Brunak, S., Chauvin, Y., and Engelbrecht, J. 1994a. Hidden Markov models of human genes. In *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, eds., Vol. 6, pp. 761–768. Morgan Kaufmann, San Mateo, CA.

Baldi, P., Chauvin, Y., Hunkapillar, T., and McClure, M. 1994b. Hidden Markov models of biological primary sequence information. *Proc. Natl. Acad. Sci. U.S.A.* **91**(3), 1059–1063.

Bengio, Y., and Frasconi, P. 1995. An input-output HMM architecture. In *Advances in Neural Information Processing Systems*, J. D. Cowan, G., Tesauro, and J. Alspector, eds., Vol. 7. Morgan Kaufmann, San Mateo, CA.

Bengio, Y., Le Cunn, Y., and Henderson, D. 1995. Globally trained handwritten word recognizer using spatial representation, convolutional neural networks and hidden Markov models. In *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, eds., Vol. 6. Morgan Kaufmann, San Mateo, CA.

Bourlard, H., and Morgan, N. 1994. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic, Boston.

Cho, S. B., and Kim, J. H. 1995. An HMM/MLP architecture for sequence recognition. *Neural Comp.* **7**, 358–369.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. 1995. The Helmholtz machine. *Neural Comp.* **7**(5), 889–904.

Dempster, A. P., Laird, N. M., and Rubin, D. B. 1977. Maximum likelihood from incomplete data via the em algorithm. *J. Roy. Stat. Soc.* **B39**, 1–22.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. 1991. Adaptive mixtures of local experts. *Neural Comp.* **3**, 79–87.

Krogh, A., Brown, M., Mian, I. S., Sjolander, K., and Haussler, D. 1994a. Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.* **235**, 1501–1531.

Krogh, A., Mian, I. S., and Haussler, D. 1994b. A hidden Markov model that finds genes in *E. coli* DNA. *Nucleic Acid Res.* **22**, 4768–4778.

Lapedes, A., and Farber, R. 1986. A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition. *Physica* **22D**, 247–259.

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

Myers, E. W. 1994. An overview of sequence comparison algorithms in molecular biology. *Protein Sci.* **3**(1), 139–146.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **77**(2), 257–286.

Neal, R. M. 1992. Connectionist learning of belief networks. *Artificial Intelligence* **56**, 71–113.

Ron, D., Singer, Y., and Tishby, N. 1994. The power of amnesia. In *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, eds., Vol. 6. Morgan Kaufmann, San Mateo, CA.

Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. 1995. Backpropagation: The basic theory. In *Backpropagation: Theory, Architectures and Applications*, pp. 1–34. Lawrence Erlbaum, Hillsdale, NJ.

Sakakibara, Y., Brown, M., Hughey, R., Saira Mian, I., Sjolander, K., Underwood, R. C., and Haussler, D. 1994. The application of stochastic context-free grammars to folding, aligning and modeling homologous RNA sequences. *UCSC Technical Report UCSC-CRL-94-14.*

Searls, D. B. 1992. The linguistics of DNA. *Am. Sci.* **80**, 579–591.

Received February 2, 1995; accepted February 8, 1996.