S8.2

COMBINING HIDDEN MARKOV MODEL AND NEURAL NETWORK CLASSIFIERS

Les T. Niles and Harvey F. Silverman

LEMS, Division of Engineering, Brown University, Providence, RI 02912

ABSTRACT

An architecture for a neural network that implements a hidden Markov model (HMM) is presented. This HMM Net suggests integrating signal preprocessing (such as VQ) with the classifier. A minimum mean squared error training criterion for the HMM/neural net is presented, and compared to maximum likelihood and maximum mutual information criteria. The HMM forward-backward algorithm is shown to be the same as the neural net back propagation algorithm. The implications of probability constraints on the HMM parameters are discussed. Relaxing these constraints allows negative "probabilities," equivalent to inhibitory connections. A probabilistic interpretation is given for a network with negative, and even complex-valued, parameters.

1. INTRODUCTION

Hidden Markov modeling is a popular, and arguably the most successful, technique used for speech recognition. Recently, there has been much interest in using neural network classifiers for speech recognition [1]. Neural nets are commonly presented as static pattern classifiers, poorly suited to analyzing time-sequence information such as speech. Thus it's sometimes suggested that neural nets be used for pre-processing, one-frame-at-a-time, and/or for post-processing, to refine or integrate information at an utterance level, leaving temporal processing to other techniques, such as hidden Markov modeling.

Neural net architectures for temporal processing have been developed, such as the TDNN of Waibel, *et al.* [2] and the recurrent Temporal Flow model of Watrous [3]. Kehagias [4] has pointed out that recurrent neural network can implement a hidden Markov model (HMM).

In this paper, an architecture for such a neural network is described. The implication is that HMMs are just special cases of neural nets. Extensions to the HMM classifier, suggested by the neural net formulation, are discussed. Ways of enforcing the probability constraints on the HMM parameters, and the implications of not doing so, are also discussed. And it is shown that a probabilistic interpretation, in the form of a "quantum mechanical model," can be provided even with unconstrained parameters.

2. HMM AS NEURAL NETWORK

2.1. Hidden Markov Model

The following HMM formulation will be used. A Markov chain has N states $q_1, \ldots, q_N$ and transition probabilities $\Pr\{q_i \to q_j\} = a_{ij}$. Let $s(t)$ denote the state at time t. At each $t = 1, \ldots T$, one of M output symbols, or observations, $v_1, \ldots, v_M$, is generated with probability $\Pr\{v_k | s(t) = q_i\} = b_{ik}$. A hidden Markov model $\mathcal{M}$ is specified by the $N \times N$ matrix $A = [a_{ij}]$, $N \times M$ matrix $B = [b_{ik}]$, and initial distribution $\pi_i = \Pr\{s(0) = q_i\}$.

Given a model $\mathcal{M}$, the probability of a sequence of observations $v_{y_1}, \ldots, v_{y_T}$ (each $y_t \in \{1, \ldots, M\}$) can be calculated by the forward-backward algorithm [5]: For $1 \le t \le T$, let $\alpha_i(t)$ be the "forward probability" $\Pr_{\mathcal{M}}\{y_1^t, s(t) = q_i\}$, and let $\beta_i(t)$ be the "backward probability" $\Pr_{\mathcal{M}}\{y_{t+1}^T | s(t) = q_i\}$, where $y_{t_1}^{t_2}$ stands for $\{y_{t_1}, \ldots, y_{t_2}\}$ and $\Pr_{\mathcal{M}}$ means probability given $\mathcal{M}$. Then $\alpha_i(t)$ and $\beta_i(t)$ can be calculated recursively,

$$\begin{aligned} \alpha_i(t) &= \sum_{j=1}^{N} \alpha_j(t-1) a_{ji} b_{iy_t}, \quad \alpha_i(0) = \pi_i, \\ \beta_i(t) &= \sum_{j=1}^{N} a_{ij} b_{jy_{t+1}} \beta_j(t+1), \quad \beta_i(T) = 1, \end{aligned} \tag{2.1}$$

and the probability of the entire observation sequence is

$$\lambda_{\mathcal{M}}(y_1^T) \equiv \Pr_{\mathcal{M}}\{y_1^T\} = \sum_i \alpha_i(t)\beta_i(t) \tag{2.2}$$

for any t between 1 and T.

The straightforward implementation of (2.1) and (2.2) suffers from numerical underflow of $\alpha_i(t)$ and $\beta_i(t)$ [5]. A standard solution is to compute normalized variables $\tilde{\alpha}_i(t) = \alpha_i(t) / \sum_i \alpha_i(t) = \Pr_{\mathcal{M}}\{s(t) = q_i | y_1^t\}$ and $\tilde{\beta}_i(t) = \beta_i(t) / \sum_i \alpha_i(t)$. Then $\Pr_{\mathcal{M}}\{y_1^T\}$ is computed recursively, as a logarithm $\Lambda_{\mathcal{M}}(y_1^T)$ to avoid underflow. After some manipulation, (2.1) and (2.2) become

$$\begin{aligned} \tilde{\alpha}_i(t) &= \frac{\sum_j \tilde{\alpha}_j(t-1) a_{ji} b_{iy_t}}{\sum_{i,j} \tilde{\alpha}_j(t-1) a_{ji} b_{iy_t}}, \quad \alpha_i(0) = \pi_i, \\ \tilde{\beta}_i(t) &= \frac{\sum_j a_{ij} b_{jy_{t+1}} \tilde{\beta}_j(t+1)}{\sum_{i,j} \tilde{\alpha}_j(t-1) a_{ji} b_{iy_t}}, \quad \beta_i(T) = 1, \end{aligned} \tag{2.3}$$

and

$$\begin{aligned} \Lambda_{\mathcal{M}}(y_1^t) &\equiv \log \Pr_{\mathcal{M}}\{y_1^t\} \\ &= \Lambda_{\mathcal{M}}(y_1^{t-1}) + \log \sum_{i,j} \tilde{\alpha}_j(t-1) a_{ji} b_{iy_t}. \end{aligned} \tag{2.4}$$

CH2847-2/90/0000-0417 $1.00 © 1990 IEEE

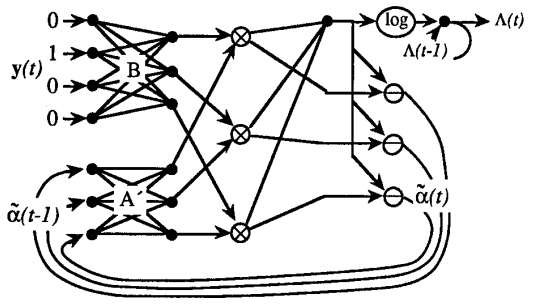

Figure 1. The HMM Net, a neural net implementation of a hidden Markov model. Filled circles indicate summing units; "⊗" units multiply two inputs together; and "⊖" units divide one input by the other. The A and B matrices represent connection weights; all other connections have weight= 1. The network is initialized by $\tilde{\alpha}_i(0) = \pi_i$.

2.2. Neural Network

An architecture for a recurrent neural network that implements (2.3) and (2.4) to compute $\Lambda(y_1^t)$, is shown in Fig. 1. The input vector uses a distributed coding: $\mathbf{y}(t)$ is an M-dimensional vector with a 1 for element y_t and 0 for all other elements. The product units used in this network have been used before for neural networks, and it can be argued that they are biologically plausible [6].

2.3. Classification

A neural network that implements Bayes' decision rule (assuming equal prior class probabilities) to discriminate amongst K classes, using K models, is shown in Fig. 2. The outputs of K HMM Nets, taken after $t = T$ when the entire sequence has been input, are combined by an additional layer. This layer acts as a maximum selector, giving outputs X_k between 0 and 1. Ideally, one of the X_k would be 1 and all the others would be 0, indicating the classification of the sequence y_1^T. In practice this won't happen, so the decision rule is to pick the largest X_k.

2.4. Implications

The above, of course, is merely a casting of an HMM into neural net terms, and does not change the classifier. However, it does demonstrate that a simple recursive neural net can perform the same operations as an HMM, and thus is at least as powerful a classifier. Given a particular HMM, there's a corresponding neural net that calculates the same output function (probability) of the input. And for a neural net of the HMM Net architecture, whose parameters satisfy probability constraints, there's an equivalent HMM and hence a probabilistic interpretation.

Viewing an HMM as a neural network suggests integrating signal preprocessing with the model. The HMM described above assumes discrete symbols as input, such as codebook entries from a vector quantizer (VQ). The VQ could also be implemented as a neural net, and the preprocessing plus model can be viewed not as two disparate parts but as one large network. This would allow training the entire classifier, HMM together with VQ, for minimum error rate.

The signal preprocessing network could allow the elements of $\mathbf{y}(t)$ to take on values intermediate between 0 and 1. Then a type of continuous mixture density model of the observations' distributions can be obtained [7], with the B matrix playing the role of mixture coefficients.

3. TRAINING

For either an HMM or a neural net, estimation of the parameters from a set of training data $\{Y_1, \ldots, Y_{N_{\text{Train}}}\}$ is of paramount importance. (Each Y_n is a sequence $y_1^{T_n}$, from class C_n.) Most training algorithms can be cast as maximization of a criterion function G with respect to the classifier parameters. Criteria commonly used for HMMs are maximum likelihood (ML) and maximum mutual information (MMI) [8].

The neural net formulation suggests a third training criterion, namely minimization of the mean-squared-error (MMSE) of the network outputs from ideal targets $X_{C_n}(n) = 1$ and $X_{k \neq C_n}(n) = 0$, where $X_k(n)$ is output k of the classifier network in Fig. 2, for input sequence Y_n.

Let $\lambda_k(n)$ and $\Lambda_k(n)$ be the probability and log probability, respectively, of training sequence Y_n given the k^{th} class model. Then the criterion functions can be written

$$\begin{aligned} G^{\text{ML}} &\equiv \log(\Pr\{Y_1, \ldots, Y_{N_{\text{Train}}} | \mathcal{M}_1, \ldots, \mathcal{M}_K\}) \\ &= \sum_n \log \Pr\{Y_n | \mathcal{M}_{C_n}\} \;=\; \sum_n \Lambda_{C_n}(n), \end{aligned}$$

$$\begin{aligned} G^{\text{MMI}} &= \sum_n \Big\{ \log \lambda_{C_n}(n) - \log\Big(\frac{1}{K} \sum_{k=1}^{K} \lambda_k(n)\Big) \Big\} \\ &= \sum_n \Big\{ \log X_{C_n}(n) + \log K \Big\}, \end{aligned}$$

and, with $\delta_{a,b}$ the Kronecker delta,

$$G^{\text{MMSE}} = -\frac{1}{2} \sum_n \sum_k \big(X_k(n) - \delta_{k,C_n}\big)^2.$$

3.1. Derivatives

Training consists of maximizing one of the objective functions G^{ML}, G^{MMI}, or G^{MMSE}. Gradient ascent is usually used for the latter two, and may be used for the first. Taking a_{ij} as an example of one of the parameters (which could just as well be a b_{ik} or π_i), the derivative can be expanded in terms of the individual class log likelihoods,

$$\frac{\partial G}{\partial a_{ij}} = \sum_{n=1}^{N_{\text{Train}}} \sum_{k=1}^{K} \phi_{n,k} \frac{\partial \Lambda_k(n)}{\partial a_{ij}},$$

where $\phi_{n,k} \equiv \frac{\partial G}{\partial \Lambda_k(n)}$:

$$\begin{aligned} \phi_{n,k}^{\text{ML}} &= \delta_{k,C_n} \\ \phi_{n,k}^{\text{MMI}} &= \delta_{k,C_n} - X_k(n) \\ \text{and} \quad \phi_{n,k}^{\text{MMSE}} &= X_{C_n}(n)\big(\delta_{k,C_n} - X_k(n)\big) \\ &\quad - \sum_{k'} X_{k'}^2(n)\big(\delta_{k,k'} - X_k(n)\big). \end{aligned}$$

The above shows how the different criteria make use of the training data. ML estimation weights all tokens equally, since for each n, $\phi_{n,k}^{\mathrm{ML}}$ is significant for exactly one k. MMI estimation ignores tokens that are clearly correctly classified, since if Y_n is correctly classified and not close to the decision boundary then $X_k(n) \approx \delta_{k,C_n} \Rightarrow \phi_{n,k} \approx 0$. In MMSE training, the first term of $\phi_{n,k}^{\mathrm{MMSE}}$ gives weight only to tokens near a decision boundary between the correct class and an incorrect class. The second term (which doesn't occur if absolute error, rather than squared error, is used) gives weight to any token that is near the decision boundary between some pair of classes k, k', and acts to move that boundary towards the token.

3.2. Forward and Backward Propagation

Gradient ascent in the above training algorithms requires taking derivatives of the HMM log-likelihood (output of the HMM Net). In a multilayer neural network the derivatives are computed recursively, using the back propagation algorithm [9], by expanding as

$$\frac{\partial \Lambda_{\mathcal{M}}}{\partial a_{ij}} = \sum_{t=1}^{T} \sum_{k} \frac{\partial \Lambda_{\mathcal{M}}}{\partial \tilde{\alpha}_k(t)} \frac{\partial \tilde{\alpha}_k(t)}{\partial a_{ij}}.$$

The $\partial \Lambda_{\mathcal{M}} / \partial \tilde{\alpha}_k(t)$ are the "back-propagated" variables, and can be computed recursively. It can be shown [7] that

$$\frac{\partial \Lambda_{\mathcal{M}}}{\partial \tilde{\alpha}_i(t)} = \tilde{\beta}_i(t).$$

Equivalently, $\partial \lambda_{\mathcal{M}} / \partial a_{ij}$ could be expanded in terms of the $\alpha_k(t)$, and $\lambda_{\mathcal{M}} = \sum_i \alpha_i(t)\beta_i(t) \Rightarrow \frac{\partial \lambda_{\mathcal{M}}}{\partial \alpha_i(t)} = \beta_i(t)$. In either case, the backward probabilities are exactly the backward-propagated variables that would be computed in applying back propagation to the HMM Net, just as the forward probabilities are the forward-propagated variables.

4. PROBABILITY CONSTRAINTS

The parameters of an HMM are probabilities, which means they must satisfy non-negativity ($a_{ij}, b_{ik}, \pi_i \geq 0$) and sum-to-one ($\sum_j a_{ij} = \sum_k b_{ik} = \sum_j \pi_j = 1$) constraints. The Baum-Welch algorithm for ML estimation ensures that these constraints will be satisfied, if they are satisfied initially [5]. But such an algorithm isn't known for MMI or MMSE, and casting the HMM as a neural net suggests using gradient ascent even for ML training. Gradient ascent does not automatically maintain the constraints.

4.1. Parameter Transformation

In order to retain the probabilistic interpretation of the models, the constraints on the parameters must be enforced. This can be done by performing gradient ascent on a set of auxiliary parameters and making the actual parameters functions of these in such a way that the constraints are satisfied. An example of such a transformation is $a_{ij} = c_{ij}^2 / \sum_k c_{ik}^2$.

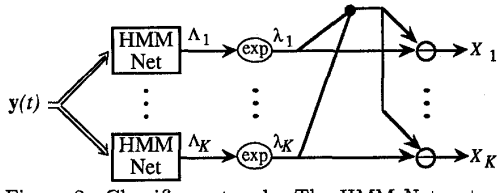

Figure 2. Classifier network. The HMM Net outputs are combined by an imperfect maximum selector. Units as in Fig. 1; all connections have weight= 1.

4.2. Constraint Relaxation

If the classifier is viewed as a neural net it's natural to just ignore the probability constraints. The interpretation as probabilities of the model outputs (and variables such as $\alpha_i(t)$) is lost, but the result may be a better classifier.

Suppose only the sum-to-one constraint is dropped. By properly scaling the parameters, it can be assumed without loss of generality that the a_{ij} and π_i still sum to one and that $\sum_k b_{ik} \leq 1$. The scaling multiplies $\lambda(y_1^T)$ by a factor of the form $c_1(c_2)^T$, effectively adding a prior distribution on the model and the observation sequence length. Having $\sum_k b_{ik} < 1$ is equivalent to allowing state q_i to generate, with probability $1 - \sum_k b_{ik}$, a null observation, that is, some symbol that is never actually seen. Thus relaxing the sum-to-one constraint does not change the fundamental nature of the resulting classifier; it can still be interpreted as an HMM. (But maximum "likelihood" training will be useless without the sum-to-one constraint.)

Relaxing the non-negativity constraint is potentially much more interesting. Negative parameters just can't be interpreted as transition or output probabilities in an HMM in the usual way. But if the classifier is thought of as a neural network, negative parameters have a simple interpretation: They correspond to inhibitory connections. This means that the presence of certain inputs or activations can act to inhibit other activations. With strictly non-negative parameters, the best that could be done would be for the presence of those inputs or activations to not contribute to increasing the other activations.

4.3. Quantum Mechanical Model

Although negative parameters cannot be interpreted as probabilities, it is still possible to provide a probabilistic interpretation of the model, by analogy with quantum mechanics. In quantum mechanics, the parameters and variables can even be complex-valued. (Complex values are not unheard of in neural networks. They arise quite naturally in Durbin and Rumelhart's product unit [6], when exponents, which play the role of connection weights, take on non-integral values.) Probabilities for observations are found by taking absolute-values-squared. Thus a probability distribution on sequences of observations can be defined by taking the absolute value before the log in accumulating the log probability in (2.4). To be a proper distribution, the probabilities should sum to one over all observation sequences. This normalization is accomplished by defining

$$\Pr_{\mathcal{M}}\{y_t|y_1^{t-1}\} = \frac{|\sum_{i,j}\tilde{\alpha}_j(t-1)a_{ji}b_{iy_t}|^2}{\sum_{k=1}^{M}|\sum_{i,j}\tilde{\alpha}_j(t-1)a_{ji}b_{ik}|^2}, \quad (4.1)$$

and (2.4) becomes

$$\Lambda_{\mathcal{M}}(y_1^T) = \Lambda_{\mathcal{M}}(y_1^{t-1}) + \log \Pr_{\mathcal{M}}\{y_t|y_1^{t-1}\}.$$

The result is a quantum mechanical model (QMM), having the following interpretation [7]: The system is described by a state vector in $\mathcal{C}^M \times \mathcal{C}^N$ ($\mathcal{C}$ is the complex numbers). The time-evolution operator H combines the HMM transition and observation matrices, $H_{(ik),(jl)} = a_{ji}b_{ik}$. The observation operator O has M eigenvalues (each N-fold degenerate), corresponding to the M symbols v_k. Making an observation means applying O, which projects the state onto the eigensubspace corresponding to the particular v_k observed. The probability of observing each v_k is given by (4.1), according to the usual rules of quantum mechanics.

In quantum mechanics, the use of complex-valued probability amplitudes is a necessity [10]. It leads to much of the structure of the physical world; in particular, the phenomenon of interference would not occur at all if wave functions were real and non-negative. Interference in quantum mechanics is analogous to inhibition in neural networks. In either case, the resulting models are fundamentally richer in structure than those, such as HMMs, that can be achieved with strictly non-negative numbers.

5. CONCLUSION

The HMM Net is an architecture for a neural network that implements an HMM. It demonstrates that an HMM is "just" a certain type of neural net, not a disparate type of classifier. Discrete HMM observations, such as VQ codebook indices, were assumed, but the HMM Net could easily be extended to continuous densities by replacing the B matrix with networks that calculate the probabilities of an observation.

By viewing the HMM as a neural network, some extensions are suggested:

1. Integration of signal preprocessing with the classifier. This integration can include a form of continuous mixture density modeling of the HMM observation probabilities. The entire classifier, preprocessing together with HMM, could be trained for minimum error rate.
2. The HMM forward-backward algorithm is essentially the same as the neural net back propagation algorithm. Therefore many techniques for improving neural net training should be applicable to the HMM Net.
3. Error-corrective training. By training the classifier outputs to a set of target values, using the MMSE criterion that is common for neural nets, an error-correcting training algorithm is provided.

The analysis of §3.1 compared ML, MMI, and MMSE training. ML uses training tokens without regard for whether or not they are correctly classified; MMI gives weight to those tokens that are incorrectly classified, or nearly so; and MMSE gives weight only to tokens that are near the decision boundary, close to both correct and incorrect classes.

Relaxing the sum-to-one constraint on the parameters of the HMM Net does not change the nature of the model in any great way. But relaxing the non-negativity constraint can lead to significant changes in the classifier. While allowing negative values renders the parameters meaningless as probabilities, it corresponds to allowing inhibition in the neural network. An analogy can be drawn between inhibition in neural networks and interference in quantum mechanical systems. A probabilistic interpretation can be given for a variant of the unconstrained-parameter HMM Net, which we refer to as a quantum mechanical model (QMM). By allowing inhibition/interference, both the HMM Net without constraints and the QMM constitute much richer classes of models than the original HMM, yet the HMM Net has the same number of parameters to learn, and the QMM has only twice as many parameters.

ACKNOWLEDGEMENTS

The authors thank Thanasis Kehagias and Don Kimber for a number of useful discussions. John Bridle pointed out that a neural network layer like the output layer of the classifier network in Fig. 2, can be viewed as a maximum selector. This work was supported by NSF grant MIP-8809742.

REFERENCES

[1] R. P. Lippmann, "Review of Neural Networks for Speech Recognition," *Neural Computation* **1**, 1-38, 1989.
[2] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme Recognition Using Time-Delay Neural Networks," *IEEE Trans. ASSP* **37**, 328-339, March 1989.
[3] R. L. Watrous, "Connectionist Speech Recognition Using the Temporal Flow Model," presented at IEEE Speech Recognition Workshop, Harriman, NY, May 1988.
[4] A. Kehagias, "Optimal Control for Training: The Missing Link Between Hidden Markov Models and Connectionist Networks," *Division of Applied Math preprint*, Brown University, 1989.
[5] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi, "An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition," *Bell System Tech. J.* **62**, 1035-1074, April 1983.
[6] R. Durbin and D. E. Rumelhart, "Product Units: A Computationally Powerful and Biologically Plausible Extension to Backpropagation Networks," *Neural Computation* **1**, 133-142, 1989.
[7] L. T. Niles, *Ph.D. thesis*, Brown University, in preparation.
[8] P. F. Brown, "The Acoustic Modeling Problem in Automatic Speech Recognition," *Ph.D. thesis*, Carnegie-Mellon University, May 1987.
[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature* **323**, 533-536, October 1986.
[10] R. P. Feynman, R. B. Leighton, and M. Sands, *The Feynman Lectures on Physics, Volume III: Quantum Mechanics*, Reading, Mass.: Addison-Wesley, 1965.