V. Petridis and Ath. Kehagias.
"Modular Neural Networks for MAP Classification of Time Series and the Partition Algorithm".

This paper has appeared in the journal:
IEEE Trans. on Neural Networks, Vol.7, pp.73-86, 1996.

Modular Neural Networks for MAP Classification of Time Series and the Partition Algorithm

Vas. Petridis

Div. of Electronics and Comp. Eng., Dept. of Electrical Eng.,Aristotle University of Thessaloniki

Thessaloniki 540 06, Greece

and

Ath. Kehagias

Div. of Electronics and Comp. Eng., Dept. of Electrical Eng.,Aristotle University of Thessaloniki

Thessaloniki 540 06, Greece

and

Dept. of Mathematics, American College of Higher Studies

Pylea 540 06, Greece

e-mail: kehagias@egnatia.ee.auth.gr

Abstract

We apply the Partition Algorithm to the problem of time series classification. We assume that the source that generates the time series belongs to a finite set of candidate sources. Classification is based on the computation of posterior probabilities. Prediction error is used to adaptively update the posterior probability of each source. The algorithm is implemented by a hierarchical, modular, recurrent network. The bottom (partition) level of the network consists of neural modules, each one trained to predict the output of one candidate source. The top (decision) level consists of a decision module, which computes posterior probabilities and classifies the time series to the source of maximum posterior probability. The classifier network is formed from the composition of the partition and decision levels. This method applies to deterministic as well as probabilistic time series. Source switching can also be accommodated. We give some examples of application to problems of signal detection, phoneme and enzyme classification. In conclusion, the algorithm presented here gives a systematic method for the design of modular classification networks. The method can be extended by various choices of the partition and decision components.

This paper appeared in IEEE Trans. on Neural Networks, Vol.7, No.1, pp.73-86, 1996.

Copyright © 1996 Institute of Electrical and Electronics Engineers.

Reprinted, with permission, from IEEE Trans. on Neural Networks.

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by sending a blank email message to info.pub.permission@ieee.org. By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

1 Introduction

We investigate the following time series classification problem. A time series X_t, $t = 1, 2, ...$ is produced by a source $S(\theta_k)$, where θ_k is a parameter taking values in a *finite* set $\Theta = \{\theta_1, ..., \theta_K\}$. We want to identify the source that produces the time series; in other words to find the "true" or "best" value of θ_k. For example, the time series X_1, X_2, ... may be a speech signal, and the parameter θ_k a phoneme label taking values in the set Θ= { [ah], [oo], ... }. When the speech signal, for instance, corresponds to the phoneme [ah], X_1, X_2, ... is produced by S([ah]). This type of problem arises in phoneme recognition [28, 29], radar signal classification [2], seismic signals processing [14], EEG/ECG analysis [31, 32] etc.

In this paper we want to combine two approaches to classification, which we find particularly promising. The first is modular design methods (for instance, cf. [6, 7, 20, 21, 22]); the second is Bayesian statistics (for instance, cf.[2, 1, 11, 12, 27]) and in particular the *Maximum A Posteriori* (MAP) classification rule (see [25]). To this end we use the Partition Algorithm (henceforth PA) [4, 5, 13, 26, 15]. This is a recursive algorithm, which can be implemented by a modular, hierarchical neural network with *partition* and *decision* components. The first component partitions the source space to subsets which can be modelled more efficiently than the whole space at once. The second component combines the models of subsets into a modular structure using the notion of posterior probability. In this sense, PA provides a framework for the systematic design of modular recursive classifiers. The operation of the partition modules can be parallelized. The algorithm consists of the following steps.

1. For every source build a predictor module, and train it on time series data from that source. This is the *off-line learning* phase.
2. For t= 1,2, ... run (in parallel) all modules and compare their predictions with the actual observation. In this way compute the prediction error of every module.
3. Use the prediction error of each predictor module to update the posterior probability of the respective source generating the time series. This is the *recursive, on-line learning* phase.
4. Classify (for current time t) the time series to the source of highest posterior probability. Classification may change at future time steps, as more observations become available.

The algorithm has a natural two-level hierarchical network implementation. The bottom level is the partition component which consists of the predictor modules implemented as neural networks. These can be linear or nonlinear (with sigmoid, Gaussian, polynomial etc. neurons). They can also be feedforward or recurrent. In fact any type of parametric predictor will do. The top level is the decision module (component), which assigns credit to each predictor module. The hierarchy of (a) the partitioned predictors and (b) the decision module, constitutes the modular, recurrent classifier network. For obvious reasons, we call this a *partition* network, as opposed to typical, *nonpartitioned* classifier networks.

We assume a probabilistic mechanism of time series generation, but it will soon become evident that the PA applies to deterministic time series as well. Also, if one wishes, the entire probabilistic framework can be discarded and the algorithm can be considered as a method of dynamically assigning credit to a set of models. In fact, there is a number of alternative credit assignment methods, that can be used. Some of these are mentioned in section 5. Credit assignment affects the decision level of the modular network, but not the predictor (partition) level. Experiments indicate that the choice of a particular type or order of predictor is not crucial to the performance of PA. Modules are combined in a general, standard manner.

The PA has a theoretical foundation in statistics. The basic foundation is presented in Section 2; see also [4, 13]. Theoretical results, regarding the convergence properties of the posteriors, will be presented elsewhere. The basic result is that, if certain mild identifiability conditions are satisfied, we can prove convergence to the "true" or "best" model with probability one.

Our work has much in common with, for example, [6, 7, 9, 10, 21]. In [21] an hierarchical architecture is presented which is very similar to our Partition Network, with a gating network (similar to our decision module), combining the outcomes of local expert networks (corresponding to our predictor modules). The same idea is used by Jordan et al. in [6, 7, 9]. In addition, these papers emphasize the competition between experts, which, in the PA is manifested in the computation of posterior probabilities using Bayes' rule. However, there are important differences between our work and the previously cited papers. Firstly, we consider dynamic, time evolving, rather than static problems. Such problems require continuous, online computation of the posterior probabilities, and final classification depends in an essential way on the dynamic behavior of both the data and the posterior probabilities. This is particularly obvious in classification tasks with source switching, as will be explained in later sections. Secondly, PA learns by explicit application of Bayes' rule, whereas in [6, 7, 9, 10, 21] learning is formulated as a Maximum Likelihood problem which requires the use of some approximate optimization algorithm. Thirdly, we perform classification using the criterion of predictive power. A fourth difference is that, in [6, 7, 9, 10, 21] the emphasis is on actually learning the partition of the source space, and classification is a final step of this learning process. In our case, on the other hand, it is assumed that the partition is already known, and the hard part of the problem is online classification (because of the presence of noise in the data, source switching and so on).

Also, the classification approach presented in [16], focusses in classification of dynamic (time evolving) patterns. However, in [16] learning requires the use of approximate optimization algorithms in the training phase. Moreover the classifier proposed is not modular. Finally, classification is just one aspect of the problem examined in [16], besides segmentation.

The rest of the paper is organized as follows. In Section 2 we formulate time series classification as a MAP problem and develop the PA. In Section 3 simulation results are presented concerning classification of two sinusoids of different frequencies, detection of a logistic time series from white noise, classification of four sequential logic circuits driven by stochastic input, and phoneme and enzyme recognition. The results of the PA and other classifiers are compared. In Section 4 we discuss the general picture of PA that emerges from our experiments. Finally, in Section 5 we give an overview of our results, consider advantages and disadvantages of the PA and discuss possible future work.

2 The Partition Algorithm

2.1 MAP classification of time series

A time series X_t, $t = 1, 2, ...$ is produced by a source $S(\theta_k)$, where θ_k is a parameter taking values in a *finite* set $\Theta = \{\theta_1, ..., \theta_K\}$. We want to identify the source that produces the time series; in other words to find the "true" or "best" value of θ_k. Introduce a random variable Z which takes values in Θ $= \{\theta_1, ..., \theta_K\}$. Θ will be henceforth called the *source set*. The time series X_1, X_2, ... is produced by source $S(Z)$. For instance, if $Z = \theta_1$, then X_1, X_2, ... is produced by $S(\theta_1)$. [1]

At every time t a decision rule produces an estimate of Z. Call the estimate $\hat{Z}_t$; it takes values in Θ. For instance, if *at time* t we guess that the X_1, ... X_t have been produced by θ_1, then $\hat{Z}_t = \theta_1$. Clearly, this estimate may change with time, as more observations become available. The decision rule for selecting $\hat{Z}_t$ for every t, makes use of the *conditional posterior probability* $p_t^k(X_1, ..., X_t)$, or simply p_t^k. This is defined as

$$p_t^k(X_1, ..., X_t) \doteq Prob(Z = \theta_k \mid X_1, ..., X_t) \qquad k = 1, 2, ..., K, \;\; t = 1, 2, ... \, . \tag{1}$$

Also, our prior knowledge of Z is described by a *prior probability* p_0^k. This is defined as

$$p_0^k \doteq Prob(Z = \theta_k \mid at \;\; t = 0) \qquad k = 1, 2, ..., K. \tag{2}$$

In the absence of any prior information we can just assume all models to be equiprobable: $p_0^k = 1/K$ for $k = 1, 2, ..., K$.

The value p_t^k reflects our belief (at time t) that the time series is produced by $S(\theta_k)$. It is natural then to choose $\hat{Z}_t$ as follows

$$\hat{Z}_t \doteq \arg \max_{\theta_k \in \Theta} p_t^k.$$

In other words, at time t we claim that $X_1, ..., X_t$ was produced by source $S(\hat{Z}_t)$, where $\hat{Z}_t$ maximizes the posterior probability. This is called the *Maximum A Posteriori* (MAP) estimate.

The classification problem has now been reduced to computing p_t^k, $t = 1, 2, ...$, $k = 1, 2, ..., K$. This is performed recursively by the Partition Algorithm, described in the next subsection.

2.2 The Partition Algorithm

We present a recursive algorithm for the computation of p_{t+1}^k, given p_t^k, for $k = 1, 2, ..., K$ and $t = 0, 1, 2, ...$. This algorithm was first developed by Hilborn and Lainiotis [4] in a general form; it was applied to control and estimation problems in [26, 13, 24], see also [15]. Here we present a version within the context of neural networks. Start with

$$p_{t+1}^k = Prob(Z = \theta_k \mid X_1, ..., X_{t+1}) =$$

[1] We may also have *source switching*. For instance source $S(\theta_1)$ produces X_1, ... , X_{10}, and source $S(\theta_2)$ produces X_{11}, X_{12}, ... This case is considered in subsection 2.5.

$$\frac{Prob(X_{t+1}, Z = \theta_k \mid X_1, ..., X_t)}{Prob(X_{t+1} \mid X_1, ..., X_t)} =$$
$$\frac{Prob(X_{t+1}, Z = \theta_k \mid X_1, ..., X_t)}{\sum_{j=1}^{K} Prob(X_{t+1}, Z = \theta_j \mid X_1, ..., X_t)}. \tag{3}$$

Also note that

$$Prob(X_{t+1}, Z = \theta_k \mid X_1, ..., X_t) =$$
$$Prob(X_{t+1} \mid X_1, ..., X_t, Z = \theta_k) \cdot Prob(Z = \theta_k \mid X_1, ..., X_t) =$$
$$Prob(X_{t+1} \mid X_1 ... X_t, Z = \theta_k) \cdot p_t^k. \tag{4}$$

Now (3), (4) imply the recursion:

$$p_{t+1}^k = \frac{Prob(X_{t+1} \mid X_1, ..., X_t, Z = \theta_k) \cdot p_t^k}{\sum_{j=1}^{K} Prob(X_{t+1} \mid X_1, ..., X_t, Z = \theta_j) \cdot p_t^j}. \qquad k = 1, 2, ..., K, \quad t = 0, 1, 2, ... \tag{5}$$

To complete the recursion for p_t^k, we need to compute the quantity

$$Prob(X_{t+1} \mid X_1, ..., X_t; Z = \theta_k) \tag{6}$$

For every value $\theta_k \in \Theta$ we build a predictor

$$\hat{X}_{t+1}^k = f(X_1, ..., X_t; Z = \theta_k). \tag{7}$$

This predictor approximates X_{t+1} *when the time series is produced by* $S(\theta_k)$; hence the dependence on θ_k. We write $f(X_1, ..., X_t; Z = \theta_k)$ for simplicity; in most cases the predictor makes use only of the finite past, e.g. X_{t-M}, ... X_{t-1}, X_t, for some finite M.

Now, for $k = 1, 2, ..., K$ define the prediction error

$$e_{t+1}^k \doteq X_{t+1} - \hat{X}_{t+1}^k. \tag{8}$$

We assume that (for $k = 1, 2, .., K$, $t = 0, 1, 2, ...$) e_{t+1}^k has a conditional probability of the form

$$Prob(e_{t+1}^k \mid X_1, ..., X_t; \theta_k) = C(\sigma_k) \cdot exp(- \mid \frac{e_{t+1}^k}{\sigma_k} \mid^n). \tag{9}$$

Substituting $e_{t+1}^k = X_{t+1} - \hat{X}_{t+1}^k$ in (9) we get

$$Prob(X_{t+1} - \hat{X}_{t+1}^k \mid X_1, ..., X_t; \theta_k) = C(\sigma_k) \cdot exp(- \mid \frac{X_{t+1} - \hat{X}_{t+1}^k}{\sigma_k} \mid^n). \tag{10}$$

But, from (7) it follows that $\hat{X}_{t+1}^k$ is a function of $X_1, ..., X_t, \theta_k$, which are given in the conditioning part of (10). Hence we have [2]

$$Prob(X_{t+1} - \hat{X}^k_{t+1} \mid X_1, ..., X_t; \theta_k) = Prob(X_{t+1} \mid X_1, ..., X_t; \theta_k). \tag{11}$$

It then follows immediately from (10) and (11) that

$$Prob(X_{t+1} \mid X_1, ..., X_t; \theta_k) = C(\sigma_k) \cdot exp(- \mid \frac{X_{t+1} - \hat{X}^k_{t+1}}{\sigma_k} \mid^n). \tag{12}$$

The probability assumption of (9) is entirely arbitrary, but works well in practice. When the exponent n is taken to be 2, we have a Gaussian probability; this is ad hoc, but reasonable. The parameter σ_k is the variance and $C(\sigma_k)$ is a normalizing constant. Both σ_k and $C(\sigma_k)$ are chosen in a standard, non-arbitrary way: σ_k is set equal to the RMS error of predictor k, as computed in the training phase, and $C(\sigma_k)$ is chosen so that the integral of the probability in (12) (over all possible values of X_{t+1}) equals one. For example, for the case $n = 2$, $C(\sigma_k) = 1/\sqrt{2\pi} \cdot \sigma_k$. Extensions for vector valued X_t and e^k_t are obvious.

Using the above equations we can compute the posterior probability of every source $S(\theta_k)$, $k = 1, 2, ..., K$ and for time $t = 1, 2, ...$ At time t we classify the time series to the source that maximizes posterior probability:

$$\hat{Z}_t \doteq \arg\max_{\theta_k \in \Theta} p^k_t. \tag{13}$$

Combining (2), (5), (7), (12) and (13) we get the desired recursion for p^k_t. The complete description of the PA can be summarized in the following equations.

*** PARTITION ALGORITHM ***

For $k = 1, 2, ..., K$

$$p^k_0 \doteq Prob(Z = \theta_k \mid at \ \ t = 0)$$

Then, for $t = 0, 1, 2, ...$ and $k = 1, 2, ..., K$

$$p^k_{t+1} = \frac{Prob(X_{t+1} \mid X_1, ..., X_t, Z = \theta_k) \cdot p^k_t}{\sum_{j=1}^K Prob(X_{t+1} \mid X_1, ..., X_t, Z = \theta_j) \cdot p^j_t}.$$

$$\hat{X}^k_{t+1} = f(X_1, ..., X_t; Z = \theta_k).$$

$$Prob(X_{t+1} \mid X_1, ..., X_t; \theta_k) = C(\sigma_k) \cdot exp(- \mid \frac{X_{t+1} - \hat{X}^k_{t+1}}{\sigma_k} \mid^n).$$

$$\hat{Z}_t \doteq \arg\max_{\theta_k \in \Theta} p^k_t.$$

The previous equations underscore the recursive nature of the algorithm. While training of the modules

[2] More precisely we have $Prob(X_{t+1} - \hat{X}^k_{t+1} = x \mid X_1, ..., X_t; \theta_k) = Prob(X_{t+1} = x + \hat{X}^k_{t+1} \mid X_1, ..., X_t; \theta_k)$.

(by eq.(7)) is an off-line process, the actual learning of the time series (by eq.(5)) is a recursive, on-line process. For correct classification we must have $\lim_{t\to\infty} p_t^k = 1$ when $S(\theta_k)$ is the true source of the time series. Then, of course, we will also have $\lim_{t\to\infty} p_t^j = 0$ for all $j \neq k$. As long as certain identifiability conditions are satisfied we can prove that convergence with probability one is guaranteed; because of lack of space, this will be published elsewhere.

2.3 Neural network implementation

The name "Partition Algorithm" has been established in the control literature [13], but it is somewhat misleading. The algorithm has two equally important components: partition and decision. "Partition" refers to separating the source space to a finite number of sources and developing models for each one. "Decision" refers to the assignment of credit to the partitioned models and the selection of the most credible model. The partition / decision structure corresponds to a hierarchical and modular network implementation illustrated in Fig.1.

1. The bottom level of the network corresponds to partition, which is implemented on the basis of prediction. One predictor corresponds to each source in the source set. The predictor modules can be implemented as neural networks. Training of the bottom level is performed off-line.

2. The top level of the network corresponds to recursive decision making. The decision element is a network which performs online the computations of posterior probabilities according to eqs. (3)-(13). This adaptation process is a form of learning.

The hierarchy of partition and decision modules constitutes the modular, recurrent classifier network. As already mentioned, we call this a *partition* network, as opposed to typical, *nonpartitioned* classifier networks.

2.4 Modification for deterministic time series

Suppose we have a deterministic time series, in which the past $X_1, \ldots, X_t$ fully determines X_{t+1}. Further, assume that we can compute exactly the dependence of X_{t+1} on $X_1, \ldots, X_t$. This can be easily accommodated by the PA. Suppose for some k $\sigma_k \to 0$. In the limit (7) reduces to

$$X_{t+1} = f(X_1, ..., X_t; Z = \theta_k) \quad \text{with prob. 1.} \tag{14}$$

However, in practice there will always be an approximation error, that is, the predictors will not reproduce (14) exactly. The approximation error is still given by (8). Since we know nothing about the error, we simply assume that it has probability given by (9), where σ_k is the standard deviation of the error. In case the probabilistic interpretation is not valid or desirable the quantities p_t^k can be considered as "goodness of fit" scores, that behave like probabilities (positive, add up to one etc.).

2.5 Modification for source switching

The derivation of the PA was based on the assumption that the time series is produced by a single source. In many interesting cases this assumption is violated. For instance, consider a speech time series that consists of several distinct phonemes. Each phoneme is produced by a different source. We refer to this phenomenon as *source switching.* Here we propose an ad hoc solution that works well in practice.

Suppose that the time series is produced from a fixed source, say $S(\theta_1)$, and at time t_s there is a source switch to $S(\theta_2)$. The *desired* behavior close to the source - switching time point is the following. For a while p_t^1 is close to one and the other p_t^k's close to zero. As we pass the switch time t_s, p_t^1 starts decreasing and p_t^2 starts increasing, until finally we have p_t^2 very close to one and the other p_t^k's close to zero, something like Fig.2 (this is a plot of the posterior probability evolution computed in an actual classification experiment described in Section 3.2).

The problem with obtaining this behavior is that, before time t_s, p_t^1 is very close to 1 and p_t^k, $k = 2, 3, .., K$ very close to 0. After t_s, p_t^2 starts increasing; theoretically, if the probabilities update for a long enough time, p_t^2 will become 1. But, since *before* t_s p_t^2 is very close to 0, after t_s it starts from a very unfavorable initial condition. Therefore it is most likely that a new source switch will take place before p_t^2 becomes sufficiently large. In such a case, classification to $S(\theta_2)$ fails. In the extreme case, because of numerical computer underflow p_t^2 is set to zero before t_s; referring to (5) we observe that p_t^2 will remain 0 for all subsequent time steps. To resolve this problem, whenever p_t^k falls below a specified threshold h, it is reset to h. Then the usual normalization of the p_t^k's is performed; this ensures that the thresholded p_t^k's remain approximately within the [h,1] range and add to 1. In essence, this thresholding is equivalent to introducing a forgetting factor. A rigorous proof is omitted, for lack of space. However, an informal argument goes as follows. Suppose that several samples of the time series are observed, which have not been produced from source $S(\theta_k)$. For each such sample, predictor k produces a large error and, from (12), p_t^k is multiplied by a number close to zero. If this process continued for several time steps, p_t^k would become zero soon, as explained above. If we never let p_t^k go below h, we essentially stop penalizing predictor k for further bad predictions; these are, in effect, "forgotten". If h is small, then p_t^k will also be small and will not essentially alter the classification results. On the other hand, when $S(\theta_k)$ becomes active, p_t^k can recover quickly. In the experiments we present in Section 3, we always chose h=0.01; this choice is arbitrary but consistent and gives good results.

3 Classification Experiments

In this section we apply the PA to five time series classification tasks. The first three tasks use computer synthesized data: *sinusoid, logistic* and *sequential logic gates* time series. The final two tasks use real world data: *speech* and *enzyme inhibition* time series. We also experiment with various source search sets and noise levels. Some of the classification problems considered here, can be solved efficiently using alternative algorithms; in the sequel we present some of them and compare their performance with that of the PA. However, it turns out that the PA performs at least comparably and usually better than any of the alternatives.

3.1 Classification algorithms

Our task is to build a classifier network such that, when its input X_t is generated by source $S(\theta_k)$, the output is $Y_t = k$ (that is, $\hat{Z}_t = \theta_k$). Here we present several methods we used in order to build such a network and explain how we chose the network (algorithm) parameters.

Partition Algorithm: It is characterized by the following.

1. The type (sigmoid, linear etc.) as well as the order M of the predictor modules. We have experimented with both linear and sigmoid feedforward predictors; the order was chosen in such a way that a satisfactory prediction error was obtained during the training phase, while overfitting was avoided. As it will be seen, predictor type and order are not crucial for the performance of the algorithm.
2. The priors p_0^k, $k = 1, ..., K$. In all experiments presented here, we took $p_0^k = 1/K$, $k = 1, ..., K$, which expresses the lack of prior information.
3. The predictor variance σ_k, k=1,2,...,K. It was always computed in a standard way: we took it equal to the RMS error of predictor k, which was computed in the training phase.
4. The probability threshold h. In all experiments reported here, h was set to 0.01. This choice was arbitrary but consistent in all experiments.
5. Finally the power n in the error probability distribution (eq. (12)) was always taken equal to 2; this corresponds to the case of Gaussian error and appeared a natural choice.

One can use a sigmoid, feedforward, one-hidden-layer network. We have $M + 1$ inputs (which at t are the last $M + 1$ values X_t, X_{t-1}, ... , X_{t-M}) and N hidden neurons, with activation X_n^h. The input/output behavior of the classifier is given by

$$X_n^h = sigm(w_{n0} \cdot X_t + w_{n1} \cdot X_{t-1} + ... + w_{nM} \cdot X_{t-M} + u_n). \qquad n = 1, 2, .., N$$

$$Y_t = sigm(v_1 \cdot X_1^h + ... + v_N \cdot X_N^h + u) \qquad u, u_n\text{:thresholds;}$$

the desired output is $Y_t = k$ when the input X_t is generated by source $S(\theta_k)$. In practice, when the output is in a range, say [$k-$ 0.5, $k+$ 0.5], we classify to source $S(\theta_k)$. This method has been used for pattern classification; see [3], pp.130-135 and especially pp.177-178. The network is least-squares-trained by the standard back propagation algorithm. The parameters are M and N and in each experiment we tried several combinations to find their best values.

Classification by the Fast Fourier Transform (FFT) coefficients is a standard method [8]. For each source, the training phase consists of the following procedure. We choose a *window width* N and a *window overlap* L. At times t= N, $2 \cdot N - L$, $3 \cdot N - 2 \cdot L$,... we take an FFT of $X_{t-N+1}, ..., X_t$; we average these coefficients at the end of the training phase to get the source template. (Actually we use the moduli of the complex-valued FFT coefficients.). In the classification phase, we repeat the procedure to compute the FFT coefficients of the (unknown source) time series $X_{t+1}, ..., X_{t+N}$ and compare it to the template of each source; X_t is classified to the source template with least

Euclidean distance to the FFT coefficients. The appropriate choice of parameters N and L can be crucial to the performance of this method.

Classification by the Linear Predictive Coding (LPC) coefficients is also a standard method, popular for speech recognition tasks [17]. For each source, the training phase consists of the following procedure. We choose a *window width* N, a *window overlap* L and a *predictor order* M. At times $t= N$, $2 \cdot N - L$, $3 \cdot N - 2 \cdot L$,... we least-squares-fit a linear M-th order predictor of $X_{t-N+1}, ..., X_t$; we average the predictor coefficients at the end of the training phase to get the source template. In the classification phase, we repeat the procedure to compute the LPC coefficients of the (unknown source) time series $X_{t+1}, ..., X_{t+N}$ and compare it to the template of each source; X_t is classified to the source template with least Euclidean distance to the LPC coefficients. The appropriate choice of parameters N, L and M can be crucial to the performance of this method.

3.2 Sinusoid Time Series

This task consists of classifying a number of sinusoid time series of the form

$$X_t = sin\left(\frac{2\pi t}{T_j}\right) \qquad t = 1, 2, ... \qquad j = 1, 2, ..., 6$$

where the period of the sinusoid is T_j and plays the role of source parameter θ_j discussed in Section 2. For training we used five sinusoid sources, with periods T_1=6, T_2=8, T_3=10, T_4=12, T_5=14. (A sixth sinusoid source was used in one of the following experiments, with T_6=16.) We present four classification methods.

1. In the case of the PA, we used noise-free data to train both a linear and a sigmoid predictor for each source. The former was used for PA classification with linear predictor modules and the latter for PA classification with sigmoid predictor modules. In the training phase we *did not* assume any prior knowledge of the nature of the time series; our only objective was minimization of RMS error and avoidance of overfitting. In the case of linear predictors these objectives were achieved with M=2. [3] Predictors of the five sources had slightly different RMS prediction errors, but all were in the range 0.01 to 0.02. The RMS error of predictor k was used as an estimate of σ_k, k=1,2, ..., K. For the sigmoid predictors, we used 2-5-1 feedforward networks and trained them with back propagation until the RMS prediction error was in the range 0.01 to 0.03; σ_k was taken equal to the RMS error. The choice of a 2-5-1 network was arbitrary and satisfied the criteria of low prediction error and no overfitting. The rest of the PA parameters were set as explained in Section 3.1.

2. In the case of the nonpartitioned FF sigmoid classifier, we experimented with various values of M and N; the best results were obtained for M=10, N=10.

[3]Choice of M did *not* require prior knowledge; we started with arbitrary M and used a matrix inversion routine for least squares training. This routine gave a "matrix ill-conditioned" diagnostic whenever $M > 2$, which reflects the fact that a sinusoid time series is the output of a second order AR process.

3. In the case of the FFT classifier, after experimentation, we chose N=32, L=16.

4. In the case of the LPC classifier, after experimentation, we chose N=10, L=5, M=2.

In the test phase we performed two groups of experiments. In all cases we used a 1800 steps time series, where source switching took place at the 951-st step. In each experiment group we experimented both with a two-member source set and a five-member source set. For each of these we used PA classification with linear predictors, PA classification with sigmoid predictors, nonpartitioned FF sigmoid classification and LPC and FFT classification. We also experimented with noisy versions of the time series, adding to each original time series uniformly distributed white noise. The noise level was expressed as the ratio of noise variance to signal variance. Hence N/S=0.0 means a noise-free time series, N/S=0.1 means that the noise variance is one tenth of the signal variance and so on. Finally, classification performance was measured by the success rate, which is the ratio of the correctly classified samples X_t over the total number of samples. For example, a 0.9 sucess rate means that 0.9·1800=1620 samples out of 1800 were classified to the source that actually produced them. Obviously, classification with a five-member source set is harder than with a two-members source set, since there are more possibilities for misclassification.

1. Success rates for the first experiment group are presented in **Table 1**. X_1, ..., X_{950} was a sinusoid of period 10 and X_{951}, ... ,X_{1800} was a sinusoid of period 6. We used two source sets: $\Theta_1 = \{S(6), S(10)\}$ and $\Theta_2 = \{S(6), S(8), S(10), S(12), S(14)\}$. Some typical classification results are illustrated in Figs. 2 (N/S=0.0) and 3 (N/S=0.5).

2. Success rates for the second experiment group are presented in **Table 2**. X_1,...,X_{950} was a sinusoid of period 10 and X_{951},...,X_{1800} was a sinusoid of period 16. We used two source sets: $\Theta_1 = \{S(10), S(14)\}$ and $\Theta_2 = \{S(6), S(8), S(10), S(12), S(14)\}$. In this case, the source that produced X_{951},...,X_{1800} is actually outside the source set; we considered classification correct if X_{951},...,X_{1800} were classified to the "closest" source in the source set, namely to $S(14)$.

It can be seen that PA has a high (close to 1) success rate. This is true both when linear and sigmoid predictors are used, except that, for the sigmoid predictors, classification performance deteriorates in the case of very high noise (N/S=0.5). The feedforward nonpartitioned classifier performs completely inadequately. LPC classification performs well, but generally not as well as PA with linear predictors. The only method that outperforms PA is FFT classification. This was expected for the sinusoid time series, since it is completely characterized by its frequency, exactly what FFT measures.

Predictor type, prediction error and noise level (within certain bounds) do not have a crucial effect on the PA performance. It appears that what is important is not *absolute* but *relative* predictive performance of the predictors. Finally, training time of PA was much shorter than that of nonpartitioned classifiers, and execution time of PA was much shorter than that of both LPC and FFT classification.

3.3 Logistic Time Series

The next problem is to distinguish between logistic time series and white noise. The logistic time series is given by the following equation

$$X_t = \alpha_j \cdot X_{t-1} \cdot (1 - X_{t-1}) \qquad t = 1, 2, ... \qquad j = 1, 2, ..., 5.$$

This is a nonlinear equation which, for values of α greater than 3.6, is chaotic; statistically it looks very similar to white noise. Here α_j is the source parameter θ_j discussed in Section 2. We used five different values of α (α_1=3.9, α_2=3.8, α_3=3.6, α_4=3.3, α_5=3.1) to produce corresponding logistic time series. Hence we had sources $S(3.1)$, $S(3.3)$ $S(3.6)$, $S(3.8)$ and $S(3.9)$. We also used a sixth source, S_0, which produced a white noise time series, with X_t (t=1,2,...) uniformly distributed in [0,1]. Training was similar to the sinusoid time series case. The PA parameters were determined in the same way as in the case of sinusoid experiments.

1. In the case of the PA, we used noise-free data to train both a linear and a sigmoid predictor for each source. The former was used for PA classification with linear predictor modules and the latter for PA classification with sigmoid predictor modules. In the training phase we *did not* assume any prior knowledge of the nature of the time series; our only objective was minimization of RMS error and avoidance of overfitting. In the case of linear predictors we took M=25. Predictors of the six sources had RMS prediction errors 0.29, 0.25, 0.21, 0.06, 0.02, 0.02 (for S_0, $S(3.9)$, $S(3.8)$, $S(3.6)$, $S(3.3)$ and $S(3.1)$ respectively); σ_k was taken equal to the RMS error of predictor k, k=1,2, ... , K. For the sigmoid predictors, we used 2-10-1 feedforward networks and trained them with back propagation until the RMS prediction error was 0.28, 0.16, 0.12, 0.07, 0.06, 0.03 (for S_0, $S(3.9)$, $S(3.8)$, $S(3.6)$, $S(3.3)$ and $S(3.1)$ respectively); σ_k was taken equal to the RMS error. The rest of the PA parameters were set, as explained in Section 3.1.

2. In the case of the nonpartitioned FF sigmoid classifier, we experimented with various values of M and N; the best results were obtained for M=10, N=10.

3. In the case of the FFT classifier, after experimentation, we chose N=32, L=16.

4. In the case of the LPC classifier, after experimentation, we chose N=100, L=50, M=10.

In the test phase we performed three groups of experiments. In all cases we used a 1900 steps time series, where source switching took place at the 951-st step. In each experiment group we experimented with a two-member source set and a five-members source set. For each of these we used PA classification with linear predictors, PA classification with sigmoid predictors, nonpartitioned sigmoid classification and LPC and FFT classification. We also experimented with noisy versions of the time series, adding to each original time series uniformly distributed white noise. Noise-to-signal ratio and classification performance were measured just as in the case of sinusoid time series.

1. Success rates for the first experiment group are presented in **Table 3**. X_1,...,X_{950} was a logistic with α = 3.9 and X_{951},..., X_{1900} was a white noise time series. We used two source sets: Θ_1 =

$\{S(3.9), S_0\}$ and $\Theta_2 = \{S(3.1), S(3.6), S(3.8), S(3.9), S_0\}$. Note that the logistic and white noise time series appear quite similar, both visually (see Fig.4) and statistically (they have the same mean, variance etc.). This similarity became even stronger in the noisy experiments, where we mixed the original time series with additional noise. Typical classification results are illustrated in Figs. 5 and 6.

2. Success rates for the second experiment group are presented in **Table 4**. X_1,...,X_{950} was a logistic with $\alpha_1 = 3.9$ and X_{951},...,X_{1900} was a logistic with $\alpha_2 = 3.8$. We used two source sets: $\Theta_1 = \{S(3.8), S(3.9)\}$ and $\Theta_2 = \{S(3.1), S(3.6), S(3.8), S(3.9), S_0\}$. In this group of experiments the difference in α between the two logistics was small, which made discrimination rather hard.

3. Success rates for the final experiment group are presented in **Table 5**. X_1, ..., X_{950} was a logistic with $\alpha_1 = 3.9$ and X_{951}, ..., X_{1900} was white noise. We used two source sets: $\Theta_1 = \{S(3.8), S_0\}$ and $\Theta_2 = \{S(3.1), S(3.3),\ S(3.6), S(3.8), S_0\}$. In this case the source that produced X_1,...,X_{950} was actually outside the source set; we considered classification correct if X_1, ..., X_{950} were classified to the "closest" source in the source set, namely the one with $\alpha_2 = 3.8$.

The logistic experiments strengthen the conclusions drawn from the sinusoid experiments. In this case PA outperforms all the alternative methods. Sigmoid models perform very well in low noise cases, while linear ones perform almost as well in low noise and show great robustness in the presence of noise.

3.4 Sequential Logic Gates

This experiment group is different from the previous two in several respects. First, we only used the PA, having concluded from the previous two experiments that it performs better than the alternative classification methods. Second, the time series X_t, $t = 1, 2, ...$ took discrete (0-1) rather than continuous values. Also, we introduced an additional *input* time series U_t, $t = 1, 2, ...$ The source set consisted of four sources, $S(\theta_1)$, ... , $S(\theta_4)$, described by the following equations, respectively.

$$X_t = XOR(X_{t-1}, U_t). \tag{15}$$

$$X_t = NOT(U_t). \tag{16}$$

$$X_t = NOR(X_{t-1}, U_t). \tag{17}$$

$$X_t = NAND(X_{t-1}, U_t). \tag{18}$$

X_t and U_t are logical variables. XOR, NOT, NOR and $NAND$ are the usual logic gates. X_0, U_1, U_2, ... are 0 or 1 with probability 0.5; $\{U_t\}_{t=1}^{\infty}$ is a white noise sequence. Finally at every time step a decision was made as to which source was active (which of eqs. (15-18) generates the next X_t) for instance in Fig.7 we have a sequence of $S(\theta_1) \rightarrow S(\theta_2) \rightarrow S(\theta_3) \rightarrow S(\theta_4)$ source switchings.

The introduction of an input time series presents no problems for the PA. For each value $k = 1, 2, 3, 4$

we have one sigmoid 2-3-1 feedforward predictor. The input-output relationship is given by

$$X^{hk}_{m,t} = sigm(v^k_1 \cdot X_{t-1} + v^k_2 \cdot U_t + u^k_m) \qquad m = 1, 2, 3.$$

$$\hat{X}^k_t = sigm(w^k_1 \cdot X^{hk}_{1,t} + w^k_2 \cdot X^{hk}_{2,t} + w^k_3 \cdot X^{hk}_{3,t} + u^k).$$

Here u^k, u^k_m are thresholds. For each value $k = 1, 2, 3, 4$ we trained the respective predictor with data from a pure $S(\theta_k)$ time series. Training was performed by back propagation.

Given the predictors of (15-18), and the values X_{t-1}, U_t we computed $\hat{X}^k_t$, $k = 1, 2, 3, 4$. From these and X_t we computed the posterior probabilities as usual for two sequences of source switchings; the classification results are presented in Figs. 7 and 8. Once again, in most cases the results are very good. In a few cases the PA was a little slow in distinguishing between two logic circuits. This was because, for the time periods and input sequences in question, both logic circuits produced identical output; however, sooner or later a sufficiently differentiating input sequence appeared and the true circuit was discovered.

3.5 Phoneme Classification

In this experiment, we used real speech data and, once again, we only tested the PA; no comparisons were made to alternative classification methods. Of course there is a number of very good speech recognition systems which perform better than our algorithm, on much harder tasks. Here we tried a much smaller (but still fairly tough) problem, with *no* data preprocessing and *no* fine tuning of the algorithm.

We used speech because we wanted to test the PA on some real world data. We emphasize that we do not claim to have a competitive speech recognition method. However, it is possible that the PA can be incorporated in a speech recognition system and give quite good results; but at the moment we do not make any claims (positive or negative) about it.

We used the utterance "one" as test data. It was pronounced once and sampled at 10 KHz. It contained three phonemes: [oo], [ah] and [n]. At the plot of Fig.9 the three phonemes can be distinguished by the different look of various parts of the time series. The three phonemes correspond to three sources; hence we built three predictor modules. They were linear predictors of the form

$$\hat{X}^k_t = w^k_1 \cdot X_{t-1} + ... + w^k_M \cdot X_{t-M} \qquad k = 1, 2, 3.$$

For each phoneme we used a 250 - point - segment (about 4 periods) to train a linear perceptron as predictor. M was taken equal to 18 and the network was least - squares - trained. Then we combined the linear predictors in a modular network. The result of the classification experiment is presented in Fig.9. It can be seen that PA classified correctly all three phonemes.

3.6 Enzyme Classification

The final experiment we performed involved classification of a set of enzymes, called β-lactamases. The data and problem are described in detail in [23]; here we give a brief description. β-lactamases are a class of enzymes that determine the resistance of many bacteria to β-lactam antibiotics. Biomedical researchers have expended considerable effort in developing methods for classifying β-lactamases; unfortunately classification by conventional methods is difficult and use of amino-acid sequencing is time consuming (see [23]).

A new method that overcomes some of these limitations is presented in [23]; it is based on an "inhibition" experiment where the hydrolysis rate of a chemical called nitrocefin is measured, in the presence of a β-lactamase / β-lactam pair. The β-lactamase enzyme causes hydrolysis of nitrocefin, and the β-lactam inhibits the action of the enzyme and slows hydrolysis down. Henceforth we will use the terms enzyme (in place of β-lactamase) and inhibitor (in place of β-lactam). The nitrocefin concentration was measured optically and recorded over a 40-minute interval. In this way, for every enzyme/inhibitor pair there is a characteristic "inhibition profile", such as those of Fig.10. The idea is that, for a given inhibitor, the inhibition profile will characterize the enzyme. This works, and the method has a high classification success. However, there is a problem: since the enzymes and inhibitors are organic compounds, their properties may depend strongly on the conditions under which they were prepared. This may result in different inhibition profiles for two different preparations of the same enzyme/inhibitor pair. However, the dynamic properties of the profile appear to remain invariant. For example, it is reported in [23] that enzyme classification required use not only of the final concentration of nitrocefin, but also of the slope of the inhibition profile at various times during the 40 min duration of the experiment. This information was used by a human operator who combined various characteristics of the inhibition profile to obtain the final enzyme classification.

Here we automated the enzyme classification process, treating the inhibition profiles as input time series to the PA. Eight enzymes were classified using inhibition profiles for a given inhibitor. We separated the data set of inhibition profiles to a test set and a training set [4]. For each enzyme we trained a fifth order linear predictor using the corresponding inhibition profile (40 min time series) from the training set. In the test phase, we chose an inhibition profile from the test set; the task was to classify it, that is to determine the enzyme it corresponds to. The PA classified correctly all eight inhibition profiles in the test set; in Fig.11 and 12 we present the posterior probability evolution for two inhibition profiles. Alternative classification methods which we used for comparison purposes, such as nearest neighbor and LPC classification, classified correctly only 6 and 4 enzymes, respectively. The experiment was repeated for a different choice of inhibitor and PA classification was again 100% correct.

In conclusion, we have succesfully used the PA to automate the enzyme classification procedure, and obtained 100% correct classification. While the "human-operated" method also achieves 100% correct classification, it is not automated.

4 Discussion

The experiments of the previous section reveal certain important characteristics of the PA. First, it clearly outperforms nonpartitioned classifiers; we believe this is because the several prediction modules

[4] We want to thank G.A. Papanicolaou for kindly permitting us to use the inhibition profile data.

of PA have to learn only small parts of the source space. This is the main motivation behind modular methods, see also [9, 10]. The structure of the partition network results in reduced training time and better utilization of neurons. It is easier for a network to learn *only* one time series rather than several. Hence fewer neurons are required and they are used more efficiently. The use of fewer neurons also reduces the chance of overfitting. The local networks are combined by the decision component, which requires no training: it is designed in such a way as to exactly implement the MAP criterion.

Local learning is also a feature of the LPC and FFT classifiers, which can be considered partitioned classifiers. However, they lack another important feature of the PA, namely the recursive computation of "goodness of fit" scores, such as p_t^k. This computation is important because it implicitly takes into account the complete past performance of each predictor module. Such "historical" information can be very important for correct classification, as it can be seen in, for example, Fig.6. We can see that at several times, most notably between t=1300 and t=1500 the true p_t^k decreases, due to an increase in the respective prediction error. Presumably, this happens because the white noise time series enters an outlier phase, which in fact resembles a logistic. However, the decrease of p_t^k is tempered by the generally good performance of the white noise predictor in previous time periods; hence no misclassification occurs. Contrast this to the behavior of both the FFT and LPC classifier, both of which misclassify several portions of the 1300 to 1500 time interval.

One could distinguish two types of classification errors: "transient" misclassification and "steady-state" misclassification. Looking at Fig.6 again, we see that PA, LPC and FFT all misclassify for a few steps around the source switching time; this is what we call "transient" misclassification. However, past the source switch time, the PA settles into steady state, and after that point it never misclassifies; this is clearly not true of LPC and FFT, which misclassify several portions of the steady state part of the time series. This is true not only of Fig.6 but of most of our experiments: generally PA takes much better account of past classification, which reduces steady state misclassification.

This is related to another point, namely that PA works even for large prediction errors. This can be seen in the high noise experiments, such as Figs. 3 and 6. High noise results in increased prediction error for *all* predictors. In fact, classification does not depend on whether a predictor is *good* in an absolute sense but on whether it performs *better* than other predictors. This results from the normalization of p_t^k in eq.(5).

The experiments presented here indicate that, except for the high noise cases, PA performance does not crucially depend on predictor type. As explained in Section 3, a standard procedure has been followed in determining the PA parameters. The experiments indicate that this procedure results in good performance for a wide range of classification tasks.

5 Conclusion

We have presented the Partition Algorithm for classification of time series. The PA involves both off-line and on-line learning. Off-line learning consists in training predictors to reproduce local behavior of the time series source space. On-line learning consists in assigining credit to each predictor by recursive application of the MAP rule. A recursive, modular, hierarchical neural network implements the algorithm. The bottom level, i.e. the partition component is implemented by a collection of prediction modules. These modules can operate in parallel. The top level, i.e. the decision component, processes the prediction errors of the bottom level and recursively updates the posterior probabilities. In short, the algorithm is recursive, modular, hierarchical and parallelizable. We have applied the PA to number of diverse time series classification tasks; the PA compares very favorably with several alternative methods of time series classification. In summary the PA has the following advantages.

1. *Modularity.* It allows easy, piecewise training of the prediction modules. No training is necessary for the decision module, which implements the MAP decision rule exactly.

2. *Parallelism.* Parallel implementation of the PA will significantly reduce execution time.

3. *Robustness* to predictor type, prediction error and noisy data. It appears that PA is robust due to the competitive nature of the MAP rule: classification does not depend on whether a predictor is *good* in an absolute sense but on whether it performs *better* than other predictors in the source set.

4. *Recursiveness.* The recursive operation of the PA presents us with a method of taking into account the complete past performance of each predictor. This temporal dependence must be taken into account in a time series, dynamic classification task.

Also, alternative algorithms can be obtained by varying either the prediction or decision component; in this way a class of partitioning, predictive, hierarchical, recursive algorithms for time series classification is obtained. Experiments with different predictor types have been presented in Section 3; let us now turn to the choice of decision rules. Consider the repeated application of the probability update rule (5); the following expression results for p^k_t

$$p^k_t \sim e^{-J^k_t} \qquad k = 1, 2, ..., K \;\; t = 1, 2, ... \tag{19}$$

where $J^k_t \doteq \sum_{\tau=1}^{t} \mid e^k_\tau \mid^n$. Obviously the "best" model is the one that has least cumulative error J^k_t. In fact, one could use (19) directly, rather than use the posterior probability. This method is used in [16]. But this is just one extreme in the range of rules that can be used for classification. At the other extreme we can select the "best" model by looking only at the last error: $J^k_t \doteq \mid e^k_t \mid^n$. This has been used often, for instance in [21]. In fact this is the principle used in the LPC and FFT classifiers of Section 3. However, a more general J^k_t, which subsumes the two previous choices as special cases, is

$$J^k_t \doteq \sum_{\tau=t-N}^{t} \mid e^k_\tau \mid^n . \qquad k = 1, 2, ..., K, \;\; t = N+1, N+2, ... \tag{20}$$

By varying N and n in the equation above we get a whole range of classification criteria. Thresholding can also be introduced; instead of a lower bound to posterior probabilities, h, we can use an *upper* bound on total error. Other decision rules that can be used, include moving average of prediction error (here the "threshold" parameter is the discount factor), number of times a predictor has minimum error and so on. Note that changing the decision rule affects only the top level of the network, not the partitioned bottom level. We have experimented with several such decision rules and found that the

MAP rule has a higher success rate than other decision rules. In addition, application of the MAP rule generates the posterior probabilities, which can be used for prediction.

Finally we list several questions requiring further research.

1. The PA can be used for prediction in at least two ways. One could, for every time step, after classification is performed, use the prediction of the MAP source. This is a winner-take-all strategy. Or else, one can use the prediction of every module and the posterior probabilities, which have already been computed, to get a weighted average prediction.

2. If there is a source switching mechanism the posterior probability equations must be modified. This has been investigated for Markov Chain switchings in [4]; it would be interesting to extend it to Hidden Markov Models and other more complex mechanisms.

3. The source space can be countably or uncountably infinite. In this case we can use the Partition Algorithm, in combination with a quantization scheme and a search method, to successively refine our search of the source space, until we obtain the MAP classification. There has been some work on this problem, in a Control Theory context, but there is space for improvement.

4. Given an infinite source space, a finite but growing number of data, say T, and predictors with an increasing number of parameters, say M, how should M and T be related to guarantee correct classification? This is the problem of *consistency.* To avoid overfitting, we must find a rate of increasing M, call it $M(T)$, as more data become available.

Acknowledgements: The authors thank the anonymous referees for reading the paper and offering many helpful comments.

References

[1] J.B. Hampshire II and B.A. Pearlmutter, "Equivalence proofs for multi-layer perceptron classifiers and the Bayesian discriminant function" , in *Proceedings of the 1990 Connectionist Models Summer School*, ed. D. Touretzky et al. Morgan Kauffman, 1990.

[2] S. Haykin and D. Cong, "Classification of radar clutter using neural networks", *IEEE Trans. on Neural Networks", NN-2* , November 1991, pp. 589-600.

[3] J. Hertz, A. Krogh and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Redwood City, 1991.

[4] C.G. Hilborn and D.G. Lainiotis "Optimal estimation in the presence of unknown parameters", *IEEE Trans. on Systems Science and Cybernetics, SSC-5*, January 1969, pp. 38-43.

[5] C.G. Hilborn and D.G. Lainiotis, "Optimal adaptive filter realizations for stochastic processes with an unknown parameter", *IEEE Trans. on Automatic Control, AC-14*, December 1969, pp. 767-770.

[6] R.A. Jacobs et al., "Task decomposition through competition in a modular connectionist architecture: the what and where vision tasks", *Cognitive Science*, to appear.

[7] R.A. Jacobs, M.I. Jordan, S.J. Nowlan and G.E. Hinton, "Adaptive mixtures of local experts", *Neural Computation*, Vol.3, 1991, pp. 79-87.

[8] F. Jelinek et al., "A Maximum Likelihood Approach to Continuous Speech Recognition", *IEEE Trans. on Pattern Analysis, PAMI-5*, pp. 179-190.

[9] M.I. Jordan and R.A. Jacobs, "Hierarchical Mixtures of Experts and the EM Algorithm", *Neural Computation*, Vol.6, 1994, pp. 181-214.

[10] M.I. Jordan and R.A. Jacobs, Hierarchies of Adaptive Experts, in *Neural Information Processing Systems 4*, 1992, J. Moody, S. Hanson and R. Lippman (eds.), San Mateo, CA, Morgan Kaufmann.

[11] F. Kanaya and S. Miyake, "Bayes statistical behavior and valid generalization of pattern classifying neural networks", *IEEE Trans. on Neural Networks, NN-2*, July 1991, pp. 471-475.

[12] F. Kanaya and S. Miyake, "A neural network approach to a Bayesian statistical decision problem", *IEEE Trans. on Neural Nets, NN-2*, September 1991, pp. 538-540.

[13] D.G. Lainiotis, "Optimal adaptive estimation: structure and parameter adaptation" *IEEE Trans. on Automatic Control, AC-16*, April 1971, pp. 160-170.

[14] D.G. Lainiotis, S.K Katsikas and S.D. Likothanasis, "Adaptive deconvolution of seismic signals: performance, computational analysis, parallelism" , *IEEE Trans. Acoustics, Speech and Signal Processing, ASSP-36*, 1988, pp. 1715-1734.

[15] D.G. Lainiotis and K.N. Plataniotis, "Adaptive Dynamic Neural Network Estimation", in *Proc. of IJCNN 1994*, **6**, pp. 4736-4745.

[16] E. Levin, "Hidden control neural architecture modelling of nonlinear time - varying systems and its applications" , *IEEE Trans. on Neural Networks, NN-4*, January 1993, pp. 109-116.

[17] S.E. Levinson et al., "An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Chain", *The Bell Sys. Tech. J.*, April 1983, Vol. 62, No.4.

[18] R.M. Neal, "Learning stochastic feedforward networks" , Technical Report CRG-TR-90-7, Dept. of Computer Science, Un. of Toronto, November 1990.

[19] R.M. Neal, "Bayesian mixture modeling by Monte Carlo simulation" , Technical Report CRG-TR-91-2, Dept. of Computer Science, Un. of Toronto, November 1990.

[20] S.J. Nowlan, "Maximum likelihood competition in RBF networks" , Technical Report CRG-TR-90-2, Dept. of Computer Science, Un. of Toronto, February 1990.

[21] S.J. Nowlan, "Competing experts: an experimental investigation of associative mixture models", Technical Report CRG-TR-90-5, Dept. of Computer Science, Un. of Toronto, September 1990.

[22] S.J. Nowlan, "Maximum likelihood competitive learning" , in *Advances in Neural Information Processing Systems 2*, ed. D. Touretzky. Morgan Kauffman, 1990.

[23] G.A. Papanicolaou and A.A. Medeiros, "Discrimination of Extended-Spectrum β-Lactamases by a Novel Nitrocefin Competition Assay", *Antimicrobial Agents and Chemotherapy*, Vol.34, No.11, Nov. 1990, pp. 2184-2192.

[24] V. Petridis, "A Method for Bearings-Only Velocity and Position Estimation", *IEEE Trans. on Automatic Control, AC-26*, April 1981, pp. 488-493.

[25] A.P.Sage and J.L.Melsa, *Estimation Theory with Application to Communication and Control*, McGraw-Hill, New York, 1971.

[26] F.L. Sims, D.G. Lainiotis and D.T. Magill, "Recursive algorithm for the calculation of the adaptive Kalman filter coefficients" , *IEEE Trans. on Automatic Control, AC-14*, April 1969, pp. 215-218.

[27] D.F. Specht, "A general regression neural network", *IEEE Trans. on Neural Networks, NN-2*, November 1991, pp. 568-576.

[28] A. Waibel et al. "Phoneme recognition using time-delay neural networks" , *IEEE Trans. on Acoustics Speech and Signal Processing, ASSP-37*, March 1989, 328-339.

[29] A. Waibel et al., "Modularity and scaling in large phonemic neural networks", *IEEE Trans. on Acoustics Speech and Signal Processing, ASSP-37*, December 1989, pp. 1888-1898.

[30] E.A. Wan, "Neural network classification: a Bayesian interpretation", *IEEE Trans. on Neural Networks, NN-1*, December 1990, pp. 303-304.

[31] L.H. Zetterberg et al., "Computer analysis of EEG signals with parametric models", *Proc. of the IEEE*, vol.69, April 1981, pp. 451-461.

[32] K. Zhu et al., "Training neural networks for ECG feature Recognition" , in *Proceedings of the IJCNN*, ed. M. Caudill. Washington, 1989.

TABLES

[5]

Nr. of sources	Method	N/S 0.0	N/S 0.1	N/S 0.2	N/S 0.3	N/S 0.4	N/S 0.5	Train. Time
2	Partition, Lin.Models	0.999	0.999	0.999	0.998	0.996	0.995	2×0.11
2	Partition, Sig.Models	1.000	0.999	0.999	0.997	0.990	0.988	2×100
2	FFT coeffs	0.996	0.996	0.996	0.996	0.996	0.996	2×1
2	LPC coeffs	0.997	0.997	0.997	0.984	0.889	0.717	2×0.1
2	Sig.Nonpart. Classifier	0.236	0.562	0.457	0.501	0.645	0.598	1200
5	Partition, Lin.Models	0.999	0.998	0.997	0.996	0.995	0.993	5×0.1
5	Partition, Sig.Models	0.999	0.998	0.995	0.992	0.752	0.762	5×100
5	FFT coeffs	0.996	0.996	0.996	0.996	0.996	0.996	5×1
5	LPC coeffs	0.997	0.997	0.927	0.712	0.611	0.543	5×0.1
5	Sig.Nonpart. Classifier	0.218	0.193	0.505	0.412	0.302	0.197	3000

Table 1: Classification of two sinusoids; T_1=10, T_2=6

Nr. of sources	Method	N/S 0.0	N/S 0.1	N/S 0.2	N/S 0.3	N/S 0.4	N/S 0.5	Train. Time
2	Partition, Lin.Models	0.998	0.997	0.991	0.980	0.964	0.887	2×0.1
2	Partition, Sig.Models	0.999	0.996	0.984	0.971	0.917	0.952	2×100
2	FFT coeffs	0.996	0.996	0.996	0.996	0.996	0.996	2×1
2	LPC coeffs	1.000	0.997	0.739	0.587	0.526	0.516	2×0.1
2	Sig.Nonpart. Classifier	0.478	0.443	0.512	0.605	0.587	0.483	1200
5	Partition, Lin.Models	0.997	0.996	0.990	0.980	0.954	0.747	5×0.1
5	Partition, Sig.Models	0.999	0.996	0.988	0.958	0.845	0.255	5×100
5	FFT coeffs	0.996	0.996	0.996	0.996	0.996	0.996	5×1
5	LPC coeffs	0.997	0.966	0.582	0.229	0.095	0.042	5×0.1
5	Sig.Nonpart. Classifier	0.203	0.294	0.187	0.215	0.226	0.189	3000

Table 2: Classification of two sinusoids; T_1=10, T_2=16

[5] Training time for all experiments is measured in seconds of CPU time for an Apollo 720 workstation. The figures are approximate ($\pm$5%).

Nr. of sources	Method	N/S 0.0	N/S 0.1	N/S 0.2	N/S 0.3	N/S 0.4	N/S 0.5	Train. Time
2	Partition, Lin.Models	0.989	0.991	0.981	0.987	0.979	0.988	2×2
2	Partition, Sig.Models	0.970	0.971	0.980	0.983	0.955	0.763	2×180
2	FFT coeffs	0.987	0.964	0.939	0.864	0.847	0.805	2×1
2	LPC coeffs	0.973	0.973	0.973	0.946	0.973	0.946	2×1
2	Sig.Nonpart. Classifier	0.530	0.453	0.483	0.582	0.621	0.578	1200
5	Partition, Lin.Models	0.976	0.979	0.988	0.976	0.971	0.865	5×2
5	Partition, Sig.Models	0.969	0.965	0.942	0.945	0.780	0.296	5×180
5	FFT coeffs	0.987	0.962	0.880	0.889	0.747	0.763	5×1
5	LPC coeffs	0.973	0.973	0.946	0.973	0.892	0.811	5×1
5	Sig.Nonpart. Classifier	0.215	0.301	0.275	0.317	0.209	0.186	3000

Table 3: Classification of logistic and noise; α_1=3.9

Nr. of sources	Method	N/S 0.0	N/S 0.1	N/S 0.2	N/S 0.3	N/S 0.4	N/S 0.5	Train. Time
2	Partition, Lin.Models	0.911	0.928	0.901	0.810	0.915	0.821	2×2
2	Partition, Sig.Models	0.988	0.984	0.951	0.797	0.487	0.487	2×180
2	FFT coeffs	0.921	0.887	0.812	0.754	0.720	0.703	2×1
2	LPC coeffs	0.785	0.406	0.432	0.460	0.459	0.406	2×1
2	Sig.Nonpart. Classifier	0.478	0.517	0.521	0.465	0.519	0.483	1200
5	Partition, Lin.Models	0.898	0.856	0.900	0.675	0.782	0.657	5×2
5	Partition, Sig.Models	0.980	0.978	0.530	0.497	0.382	0.098	5×180
5	FFT coeffs	0.437	0.420	0.337	0.320	0.238	0.211	5×1
5	LPC coeffs	0.785	0.433	0.379	0.486	0.325	0.298	5×1
5	Sig.Nonpart. Classifier	0.189	0.178	0.220	0.213	0.256	0.203	3000

Table 4: Classification of two logistics; α_1=3.9, α_2=3.8

Nr. of sources	Method	N/S 0.0	N/S 0.1	N/S 0.2	N/S 0.3	N/S 0.4	N/S 0.5	Train. Time
2	Partition, Lin.Models	0.978	0.990	0.990	0.986	0.991	0.879	2×2
2	Partition, Sig.Models	0.978	0.980	0.990	0.986	0.933	0.683	2×180
2	FFT coeffs	0.971	0.971	0.971	0.912	0.847	0.805	2×1
2	LPC coeffs	0.973	0.973	0.973	0.919	0.838	0.702	2×1
2	Sig.Nonpart. Classifier	0.523	0.503	0.486	0.508	0.501	0.495	1200
5	Partition, Lin.Models	0.959	0.969	0.951	0.985	0.985	0.957	5×2
5	Partition, Sig.Models	0.976	0.972	0.986	0.978	0.845	0.185	5×180
5	FFT coeffs	0.971	0.962	0.954	0.904	0.805	0.845	5×1
5	LPC coeffs	0.946	0.973	0.892	0.756	0.865	0.675	5×1
5	Sig.Nonpart. Classifier	0.238	0.294	0.184	0.203	0.193	0.201	3000

Table 5: Classification of logistic and noise; α_1=3.9

Figure 1: Sketch of partition network architecture

Figure 2: Classification of two sinusoids. T_1=10, T_2=6, N/S=0.0. (A) Evolution of posterior probabilities:"—" is probability of $S(T_1)$,"- -" is probability of $S(T_2)$. (B) Classification output: "—" is desired classification, "- -" is PA classification, "..." is FFT classification, "-.-" is LPC classification. (FFT and LPC classification outputs are not clearly discernible in the figure because they follow ideal classification very closely.)

Figure 3: Classification of two sinusoids. T_1=10, T_2=6, N/S=0.5. (A) Evolution of posterior probabilities:"—" is probability of $S(T_1)$, "- -" is probability of $S(T_2)$. (B) Classification output: "—" is desired classification, "- -" is PA classification, "..." is FFT classification, "-.-" is LPC classification. (FFT and LPC classification outputs are not clearly discernible in the figure because they follow ideal classification very closely.)

Figure 4: Time series consist of logistic (α_1=3.9) and white noise.

Figure 5: Classification of logistic and white noise, N/S=0.1. (A) Evolution of posterior probabilities: ”—” is probability of $S(\alpha_1)$, ”- -” is probability of S_0. (B) Classification output: ”—” is desired classification, ”- -” is PA classification. (C) Classification output: ”—” is desired classification, ”- -” is FFT classification. (D) Classification output: ”—” is desired classification, ”- -” is LPC classification.

Figure 6: Classification of logistic and white noise, N/S=0.4. (A) Evolution of posterior probabilities: "—" is probability of $S(\alpha_1)$, "- -" is probability of S_0. (B) Classification output: "—" is desired classification, "- -" is PA classification. (C) Classification output: "—" is desired classification, "- -" is FFT classification. (D) Classification output: "—" is desired classification, "- -" is LPC classification.

Figure 7: Sequential logic circuits classification. (A) Source sequence.(B) Probability evolution: "—" is probability of source 1, "- -" is probability of source 2, "..." is probability of source 3, "-.-" is probability of source 4.

Figure 8: Sequential logic circuits classification. (A) Source sequence.(B) Probability evolution: "—" is probability of source 1, "- -" is probability of source 2, "..." is probability of source 3, "-.-" is probability of source 4.

Figure 9: Results of experiment in phoneme classification. The solid line indicates the phoneme time series. Probability evolution: "- -" is probability of [oo], "..." is probability of [ah], "-.-" is probability of [n].

Figure 10: Enzyme inhibition profiles.

Figure 11: Posterior probability evolution for enzyme classification.

Figure 12: Posterior probability evolution for enzyme classification.

CAPTIONS

Figure 1: Sketch of partition network architecture

Figure 2: Classification of two sinusoids. T_1=10, T_2=6, N/S=0.0. (A) Evolution of posterior probabilities:"—" is probability of $S(T_1)$,"- -" is probability of $S(T_2)$. (B) Classification output: "—" is desired classification, "- -" is PA classification, "..." is FFT classification, "-.-" is LPC classification. (FFT and LPC classification outputs are not clearly discernible in the figure because they follow ideal classification very closely.)

Figure 3: Classification of two sinusoids. T_1=10, T_2=6, N/S=0.5. (A) Evolution of posterior probabilities:"—" is probability of $S(T_1)$, "- -" is probability of $S(T_2)$. (B) Classification output: "—" is desired classification, "- -" is PA classification, "..." is FFT classification, "-.-" is LPC classification. (FFT and LPC classification outputs are not clearly discernible in the figure because they follow ideal classification very closely.)

Figure 4: Time series consist of logistic (α_1=3.9) and white noise.

Figure 5: Classification of logistic and white noise, N/S=0.1. (A) Evolution of posterior probabilities: "—" is probability of $S(\alpha_1)$, "- -" is probability of S_0. (B) Classification output: "—" is desired classification, "- -" is PA classification. (C) Classification output: "—" is desired classification, "- -" is FFT classification. (D) Classification output: "—" is desired classification, "- -" is LPC classification.

Figure 6: Classification of logistic and white noise, N/S=0.4. (A) Evolution of posterior probabilities: "—" is probability of $S(\alpha_1)$, "- -" is probability of S_0. (B) Classification output: "—" is desired classification, "- -" is PA classification. (C) Classification output: "—" is desired classification, "- -" is FFT classification. (D) Classification output: "—" is desired classification, "- -" is LPC classification.

Figure 7: Sequential logic circuits classification. (A) Source sequence.(B) Probability evolution: "—" is probability of source 1, "- -" is probability of source 2, "..." is probability of source 3, "-.-" is probability of source 4.

Figure 8: Sequential logic circuits classification. (A) Source sequence.(B) Probability evolution: "—" is probability of source 1, "- -" is probability of source 2, "..." is probability of source 3, "-.-" is probability of source 4.

Figure 9: Results of experiment in phoneme classification. The solid line indicates the phoneme time series. Probability evolution: "- -" is probability of [oo], "..." is probability of [ah], "-.-" is probability of [n].

Figure 10: Enzyme inhibition profiles.

Figure 11: Posterior probability evolution for enzyme classification.

Figure 12: Posterior probability evolution for enzyme classification.

BIOGRAPHIES

Vassilios Petridis received the Diploma in Electrical Engineering from the National Technical University in Athens, Greece, in 1969. He obtained the M.Sc. and Ph.D. degrees in electonics and systems from King's College, University of London, in 1970 and 1974 respectively. He has been consultant of the Naval Research Centre in Greece, Director of the Division of Electronics and Computer Engineering and Vice-Chairman of the Department of Electrical and Computer Engineering in the Aristotle University of Thessaloniki, Greece. He has also been project leader and technical manager in a number of research projects funded by the industry, the Greek government and the European Union. He is currently Professor in the Department of Electrical and Computer Engineering in the Aristotle University of Thessaloniki. He is author of three books on control and measurement systems and of over seventy research papers. His research interests include control systems, intelligent and autonomous systems, artificial neural networks, genetic algorithms, robotics and industrial automation.

Athanasios Kehagias received the Diploma in Electrical Engineering from the Aristotle University of Thessaloniki in 1984, the M.Sc. in Applied Mathematics from Lehigh University in 1985 and the Ph.D. in Applied Mathematics from Brown University in 1991. Since 1993 he has been an Assistant Professor of Mathematics at the American College of Higher Studies and a Research Associate at the Department of Electrical and Computer Engineering of Aristotle University; both institutions are located in Thessaloniki, Greece. His research interests include stochastic processes, time series, neural networks, learning and game theory.

Figures for the paper “Modular Neural Networks for MAP Classification of Time Series and the Partition Algorithm”, IEEE Trans. on Neural Networks, Vol.7, No.1, pp.73-86, 1996.

(by V.Petridis and Ath.Kehagias)

Figure 2A

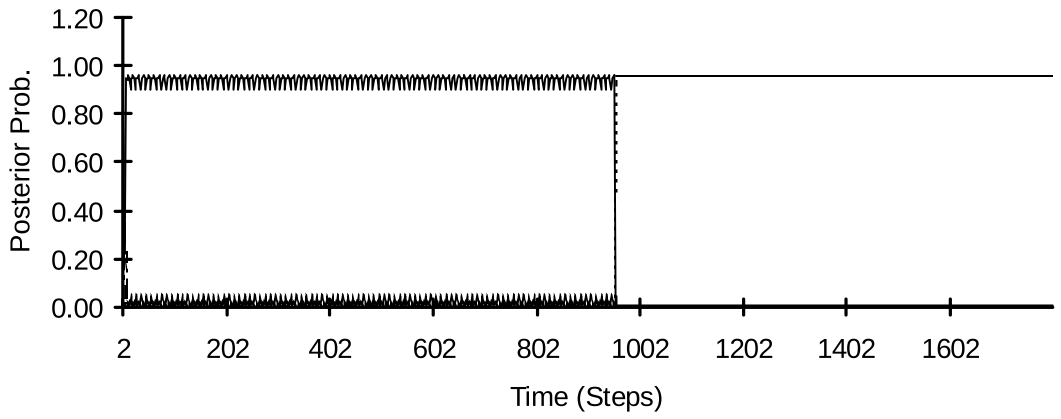


Figure 2B

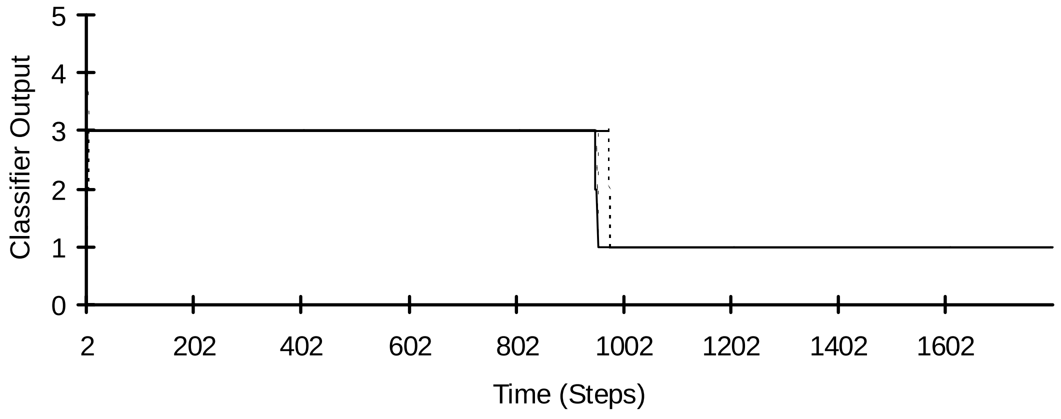


Figure 3A

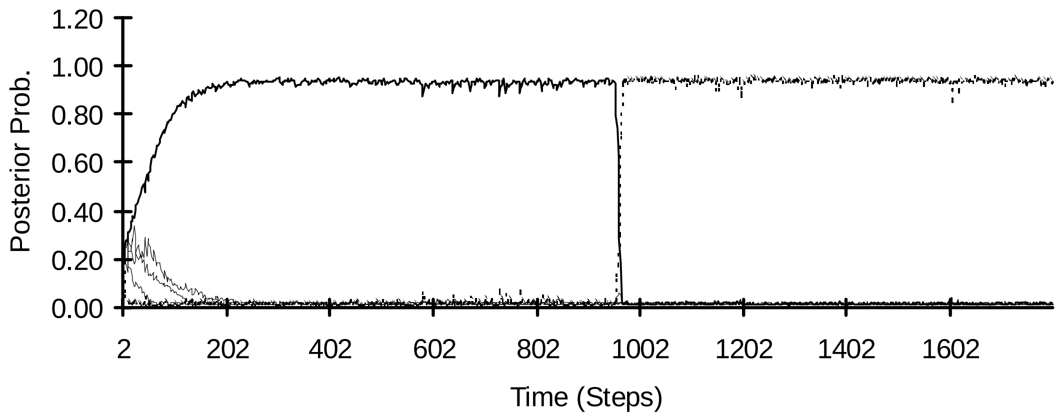


Figure 3B

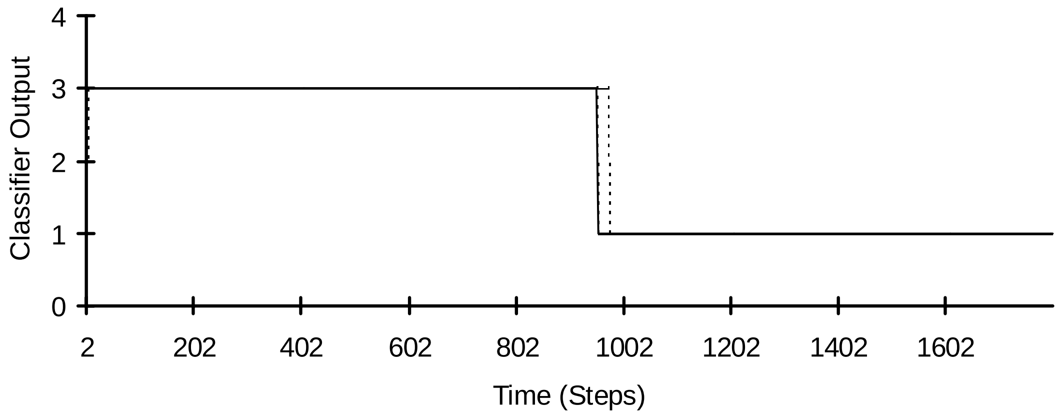


Figure 4

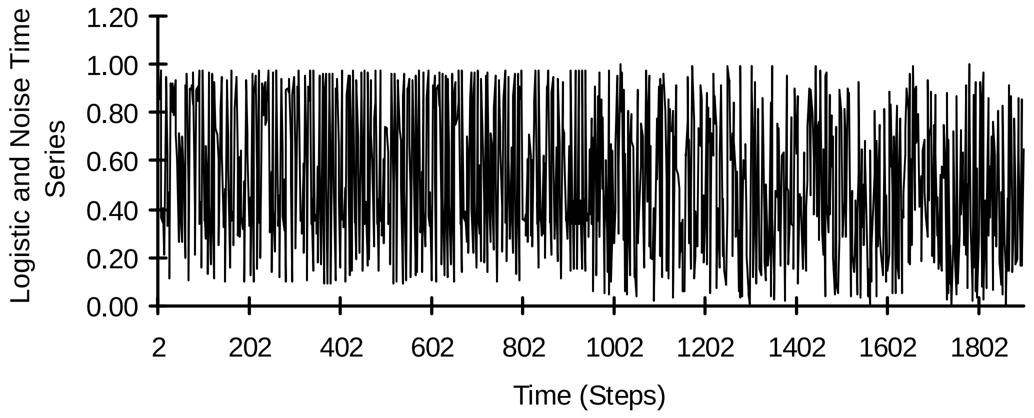


Figure 5A

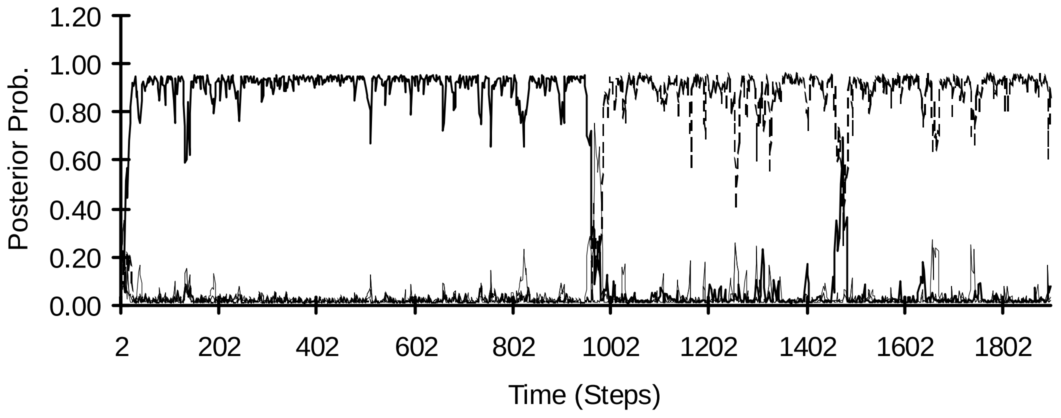


Figure 5B

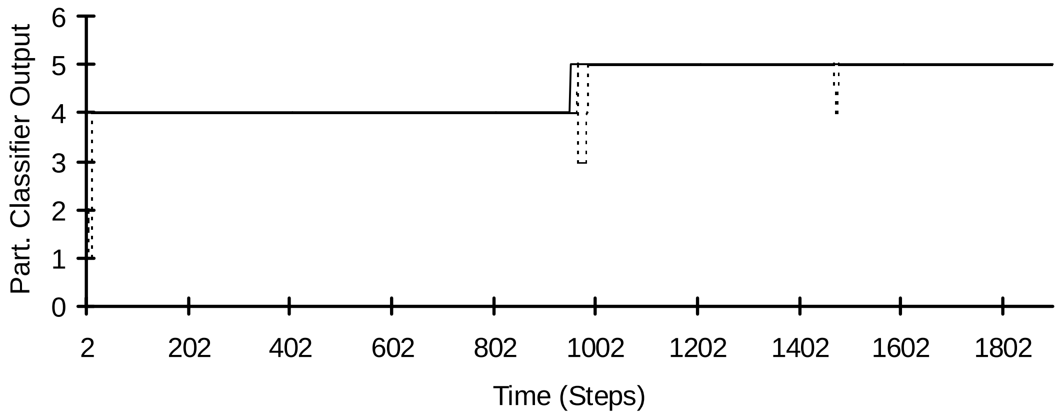


Figure 5C

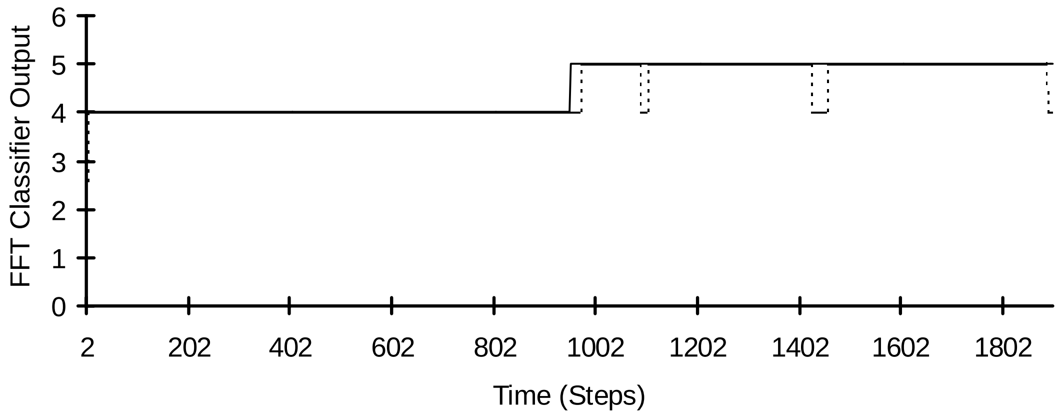


Figure 5D

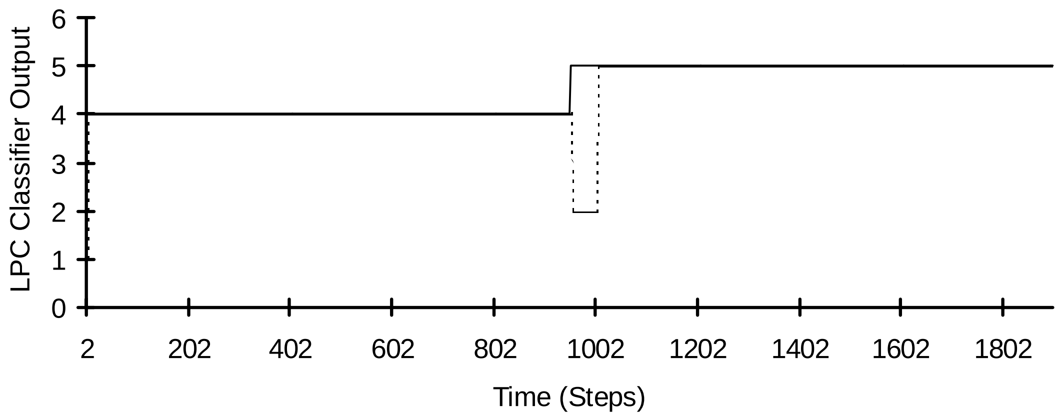


Figure 6A

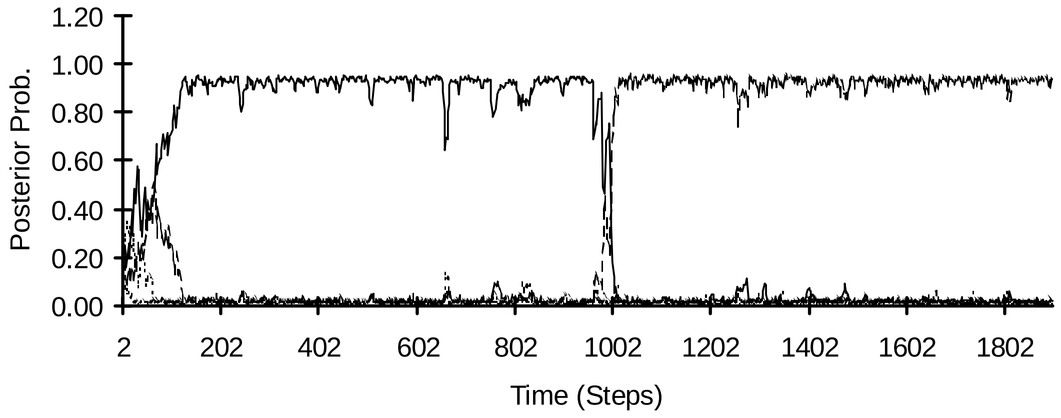


Figure 6B

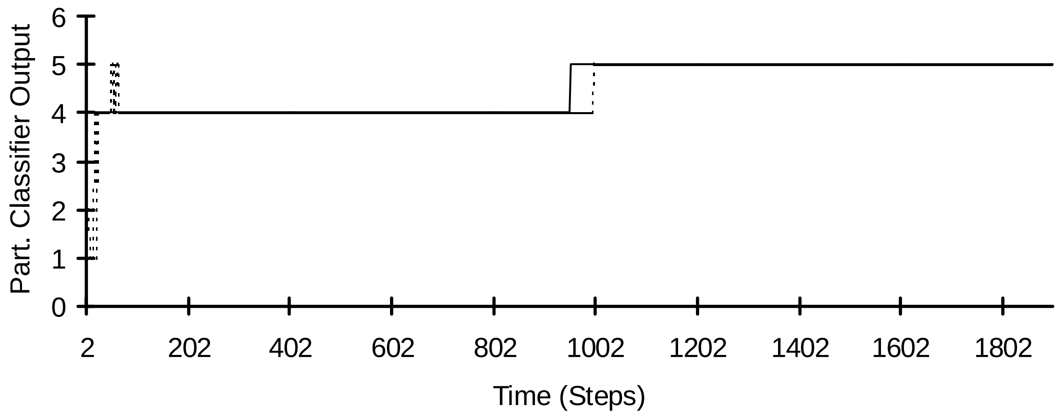


Figure 6C

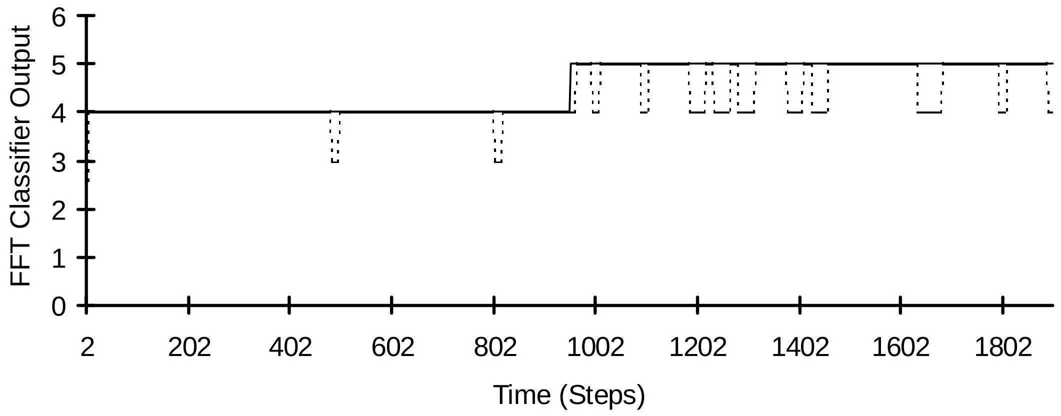


Figure 6D

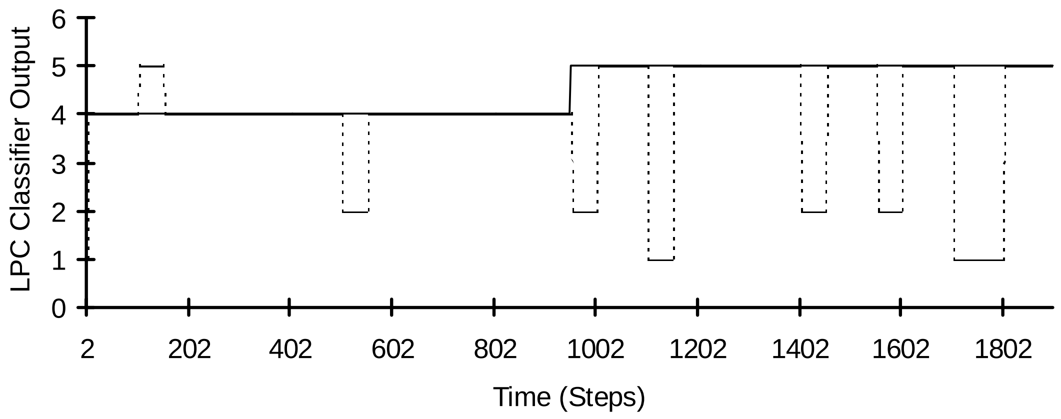


Figure 7B

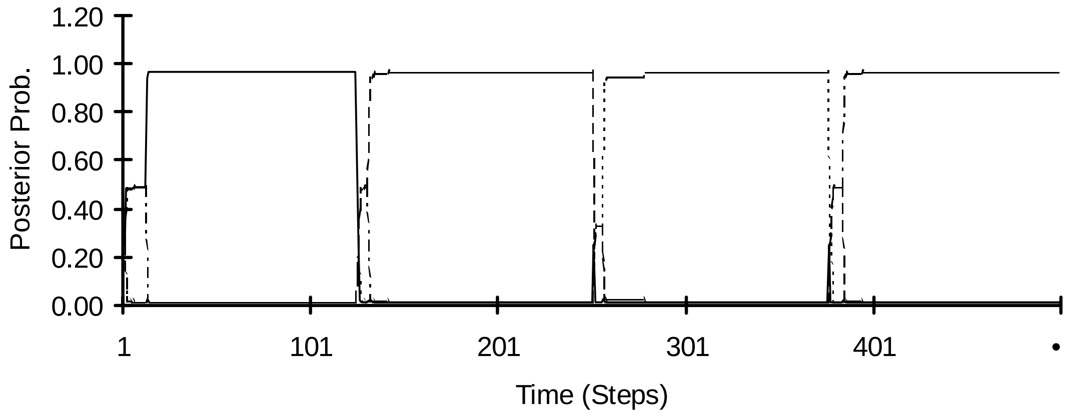


Figure 8A

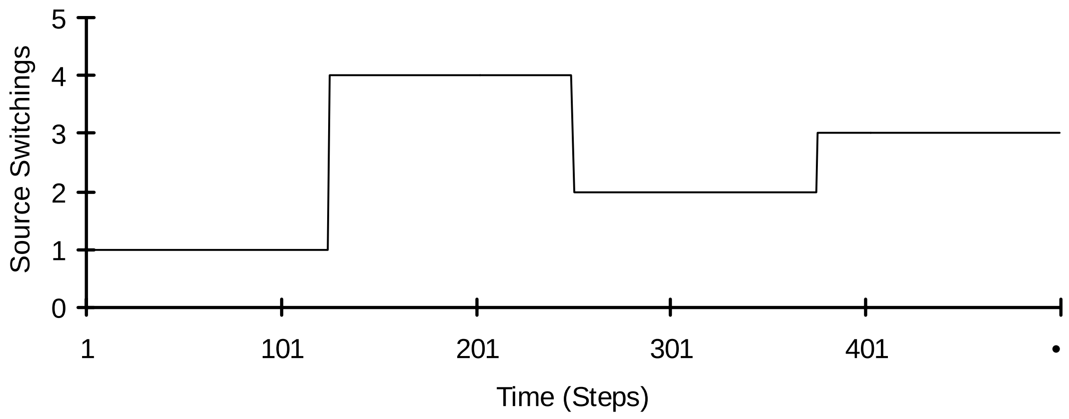


Figure 8B

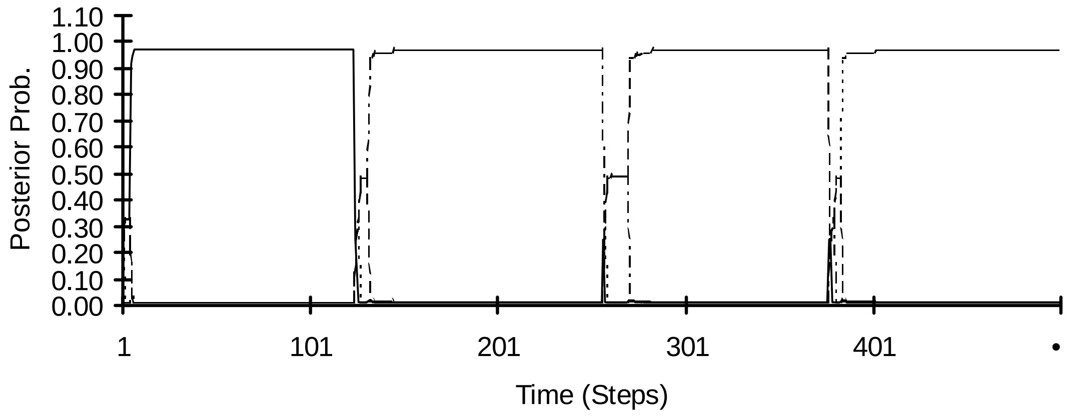


Figure 9

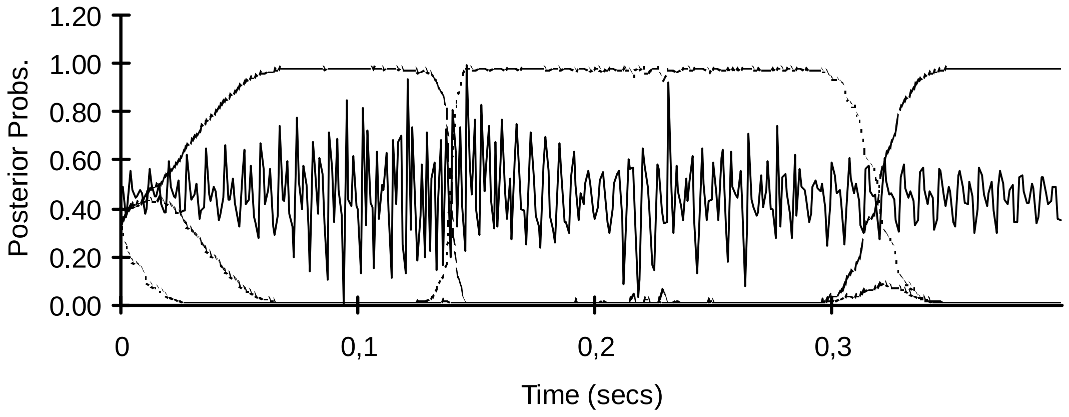


Figure 10

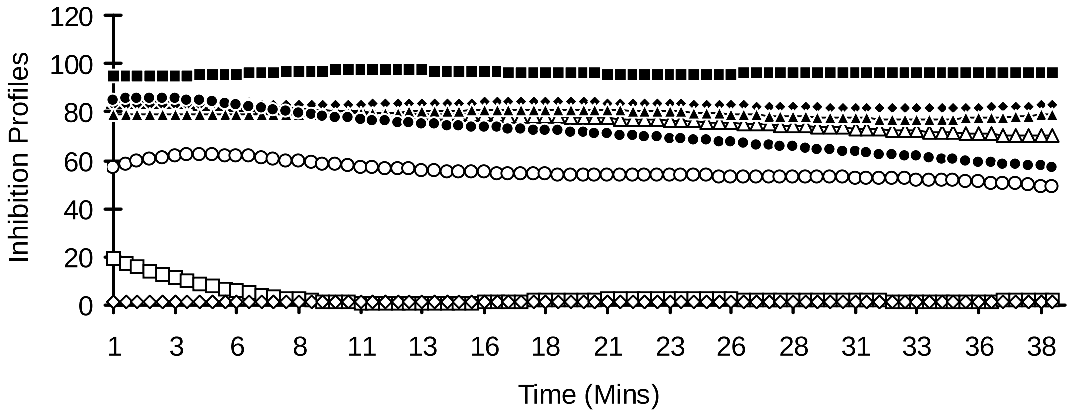


Figure 11

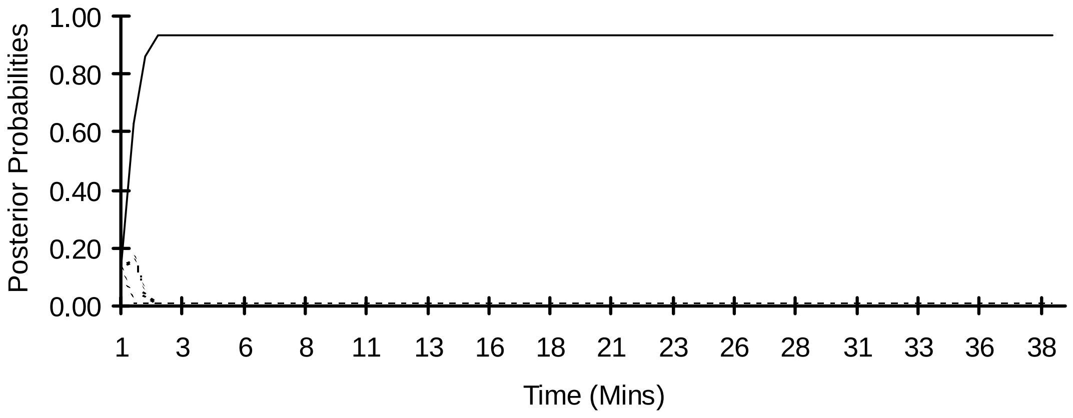
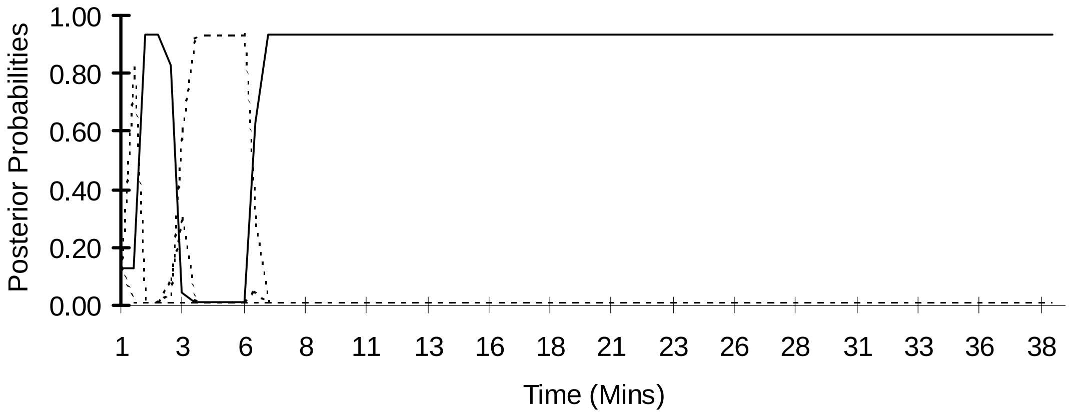


Figure 12


Posterior Probabilities
Time (Mins)