Markov chains. While the first order Markov assumption may not hold in reality, it provides a simple, tractable model, which we hope is a good enough approximation at this low level of representation. The resulting model, illustrated in figure 2 and presented in more detail in the next section, is a hidden Markov model (HMM) whose states are neural networks. The learning task is defined as finding the model that produces the maximum likelihood segmentation.

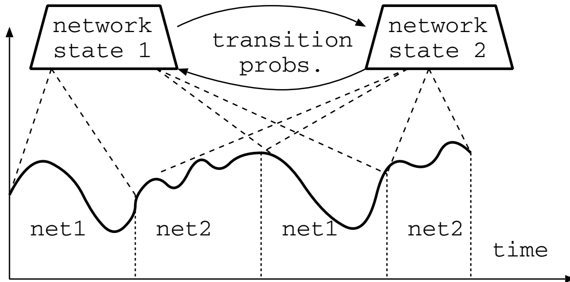

Figure 2: The hybrid HMM/ANN model: at each moment the data generating process is assumed to be in one HMM state, and the observed value is assumed to be generated by the neural network embedded in that state.

The model

While motivated by the goal of having a robot agent learn representations, the learning paradigm described here applies to time series generated by other processes as well, so it will be presented in general terms. Given a time series of observed vectors $x(1)\dots x(t)\dots x(T)$ of n variables, $x(t) = (x_1(t), \dots x_i(t)\dots x_n(t)) \in R^n$, we assume that it was generated by a set of K processes – or function patterns, each process k being described by a continuous function $f_k = R^n \rightarrow R^d$:

$$\begin{aligned} y(t) &= f_k(x(t), x(t-1), \dots) \\ x_{i_j}(t+1) &= y_j(t) + e_k(t)\ 1 \le j \le d \end{aligned}$$

with $e_k(t)$ a random normal variable representing noise. Variables $x_{i_j}(t+1)$, whose probability densities are controlled by functions f_k, are said to be the output variables; we want to identify the K processes by learning to predict the values of these variables. The remaining variables are considered input variables provided by the environment; their probability distribution will be ignored here. The partitioning into input and output variables is considered given – for example, *trans-vel*(t) and *vis-A-x*(t) sensor may be the input variables, and *vis-A-x*(t+1) the output variable. A data point at time t is the pair $\langle x(t), y(t)\rangle$, with $y \in R^d$ the vector of output variables. The pair $\langle x(t), y(t)\rangle$ will be denoted by $o(t) \in R^{n+d}$, and the resulting time series by the sequence $O = o(1)\dots o(t)\dots o(T)$. The likelihood of observing $o(t)$ within O, given function f_k, is the value of the multivariate normal density $N(e_k(t), 0, \Sigma^k)$:

$$\rho_k(o(t)) = \frac{1}{(2\pi|\Sigma_k|)^{d/2}} e^{-\dfrac{e_k(t)^t \Sigma_k^{-1} e_k(t)}{2}} \quad (1)$$

$$e_k(t) = y(t) - f_k(x(t), x(t-1), \dots) \quad (2)$$

where $e_k(t)^t$ is the transpose of vector $e_k(t)$. The parameter Σ^k is a diagonal covariance matrix (i.e. the noise variables are assumed uncorrelated) associated with process k. The process of switching from one function pattern to another is assumed to be described by a stationary first-order Markov chain – every function pattern is a state of this process. This is a strong assumption, but for now it provides a computationally tractable model. Each function pattern k has a set $\{a_{k,l}, 1 \le l \le K\}$ of transition probabilities – $a_{k,l}$ is the probability that the chain will be in state l at time $t+1$ if it is in state k at t. An additional parameter $a_{0,k}$ is the probability that k is the initial state of the Markov process. The initial and transition probabilities will be denoted by vector a_k. The resulting structure

$$\lambda = \{s_k = \langle f_k, \Sigma_k, a_k\rangle, 1 \le k \le K\}$$

is a hidden Markov model (see Rabiner's tutorial (Rabiner 1989) for a comprehensive description of the model and its estimation algorithms) with K states $\{s_k\}$. Given a model λ, a state path $S = s(1)\dots s(T)$ specifying the state of the process at each time, and an observed time series O, the likelihood of O being generated by λ along path S is:

$$L_\lambda(O, S \mid \lambda) = \prod_{t=1}^{T} a_{s(t-1),s(t)} * \rho_{s(t)}(o(t))$$

For time series O and model λ, the best segmentation of O is considered to be the one given by the most likely path:

$$V = arg\max_S L_\lambda(O, S \mid \lambda)$$

This path is called the Viterbi path and can be computed by the Viterbi algorithm (Rabiner 1989).

We assume that the functions f_k can be computed by recurrent neural networks. Because neural networks can approximate arbitrarily well any continuous function (see Pinkus's survey (Pinkus 1999)), and even some discontinuous functions (see (Barron 1993)), this is a weak (non-restrictive) assumption.

The learning task can now be described as finding a model $\lambda = \{\langle net_k, \Sigma_k, a_k\rangle, 1 \le k \le K\}$ that maximizes the likelihood $L_\lambda(O, V_\lambda)$ for a given time series O; net_k is the neural network of state k. The subscript in V_λ indicates that the segmentation depends on the model λ.

Induction algorithms

The goal is to find the model that maximizes the likelihood of the observed time series along its Viterbi path. Algorithm 1 searches heuristically for a local maximum by adding new network states as long as the likelihood keeps increasing, then reducing the number of states until a minimum description length criterion (Rissanen 1984) is satisfied.

Algorithm 1 *Main algorithm – Model induction*

1. *initialization:*
   - *available-points* ← *all points*
   - *create a non-content state from available-points*
2. *initial HMM/ANN induction*
   repeat while available-points $\neq \emptyset$ *and* $L_\lambda(O, V_\lambda)$ *increases*

(a) *create a new network state and train the network with the "reduced support" algorithm (alg. 3) on available-points; add the network state to the model*

(b) *find the best model λ with the current number of states with algorithm 2, and its segmentation V_λ*

(c) *available-points $\leftarrow$ non-content state's points $\cup$ points poorly predicted by the networks*

3. *final HMM/ANN – model reduction with algorithm 4*

The model is initialized with one non-content state. This state does not model the observed data with a neural network, but only with a multivariate normal density. Its goal is to collect the noisy or difficult to predict data points. Each network added to the model in step 2a has a minimal architecture and is initially trained with the "reduced support" training algorithm described later. We call the support of a network the set of points used to estimate its parameters. The algorithm 2 for finding the best model with given number of states also creates a segmentation of the observed time series, by allocating each data point to one state. The set of available points computed at step 2c contains all the points allocated to the non-content state in the previous step, and the points allocated to the content states that are poorly predicted by them. A point is poorly predicted by its network owner if the network error in that point is larger than a dynamically computed threshold.

Finding the best model with given number of states

Algorithm 2, which finds the best model with a fixed number of states, is related to the expectation maximization (EM) algorithm. It differs in that instead of trying to maximize the model's expected likelihood, it tries to maximize the model's maximum likelihood – the likelihood along the model's Viterbi path. Because it does not look at all possible paths, algorithm 2 is computationally less expensive than the Baum-Welch(Rabiner 1989) algorithm, which sums the likelihoods along all paths.

Algorithm 2 *Best model with fixed number of states:*

- *start with λ and V_λ*
- *repeat:*
  1. *estimate a new model λ^* from V_λ: train the networks, calculate the variances and the transition probabilities*
  2. *calculate V_{λ^*}; $\lambda \leftarrow \lambda^*$, $V_\lambda \leftarrow V_{\lambda^*}$*

  until the segmentation no longer changes

At step 1 each network is trained with all the points assigned to it by segmentation V. After the networks are trained to minimize the error along V, the rest of the model parameters $\{\Sigma_k, a_k\}$ can be estimated from the data by maximizing the likelihood $L_\lambda(O, V)$. The maximum can be found in this case simply by setting the partial derivatives of the likelihood to 0, and solving the resulting system of equations. The covariance matrix Σ_k is assumed diagonal, so we need to estimate only the variances σ_j^k of the y_j variables in state k. The unique solutions are the average network errors:

$$\sigma_j^k = \frac{1}{size(T_k)} \sum_{t \in T_k} (y(t) - f_k(x(t), t))^2 \tag{3}$$

where T_k is the set of (indices of) points allocated to network k in segmentation V. The solutions for the probability transitions of each state are obtained by imposing the constraint that they must sum up to 1. These solutions are:

$$a_{k,l} = \frac{\#s_k \to s_l}{\#s_k^-} \tag{4}$$

where $\#s_k \to s_l$ is the number of observed transitions from state s_k to state s_l in path V, and $\#s_k^-$ is the number of occurrences of state s_k in V (not counting the last element of sequence V). These solutions are unique, too.

At step 2 a new segmentation is computed for the new model λ^* by a dynamic programming algorithm: first the likelihood of every possible state run (contiguous repetition of the same state) $(s, t_i, t_f), 1 \leq t_i \leq t_f \leq T$ is computed, then Dijkstra's shortest path algorithm finds the best sequence of state runs. We need to consider state runs instead of one state at a time as in the Viterbi algorithm, because the networks embedded in the HMM states are recurrent and their internal states change along state runs – the state runs, and not the state occurrences have the Markov property.

It can be easily noticed that the likelihood $L_\lambda(O, V_\lambda)$ increases at every iteration of algorithm 2:

- at step 1, $L_\lambda(O, V_\lambda) \leq L_{\lambda^*}(O, V_\lambda)$ because training the networks reduces their errors along V_λ; also, the maximum likelihood estimators of the network variances and transition probabilities are easily calculated for V_λ with formulas 3 and 4;
- at step 2, $L_{\lambda^*}(O, V_\lambda) \leq L_{\lambda^*}(O, V_{\lambda^*})$ because V_{λ^*} is the most likely path for model λ

We have $L_\lambda(O, V_\lambda) \leq L_{\lambda^*}(O, V_\lambda) \leq L_{\lambda^*}(O, V_{\lambda^*})$, meaning that the likelihood either increases or stays the same with every iteration of alg 2.

Neural networks: architecture and training

Expression 2 shows that in the general case the output value of network net_k (which computes function f_k) depends on the entire past sequence of observed variables. In order to implement this variable length memory, recurrent neural networks with the architecture depicted in figure 3 were chosen. The input and output units have identity transfer

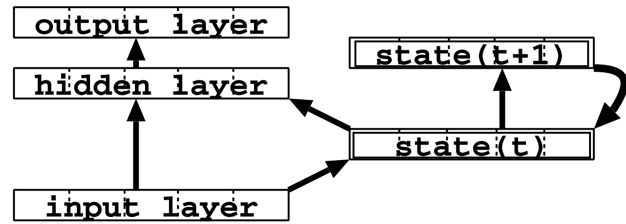

Figure 3: Recurrent network architecture: $s(t+1) = func(s(t), x(t))$ is the network's internal state, encoded in the recurrent units.

functions, the hidden and state units have cosine activation functions. The cosine units can readily construct an approximating function – since they form the function's Fourier representation (see Barron (Barron 1993)), and unlike the more

popular logistic units, they do not saturate during gradient descent. State units can also have identity transfer functions, in which case they are used as memory rather than computational units. For now, the network architecture (size, topology and transfer functions) is predetermined, not adapted during training.

Assuming an initial segmentation of the time series, each network is trained to estimate the y values within its assigned segments. The networks are trained with gradient descent on the error surface, the learning rate being adjusted automatically to follow the error surface faithfully until a local minimum is found. The error function is the normalized mean square error:

$$E = \frac{1}{2l} \sum_{t=1}^{l} \sum_{j=1}^{d} \frac{(y_j(t) - z(t))^2}{\sigma_j}$$

where $z(t) = f(x(t), t)$ is the network output, l is the number of points allocated to the network, and the first sum is taken over these points. The argument t in the function $f(x(t), t)$ computed by the recurrent network indicates that the output depends on the past inputs. The individual error of each output variable j must be normalized by the variance σ_j associated with the network's state so that one variable is not overfitted at the expense of the others. This particular error function is the one that must be minimized when maximizing the likelihood $L_\lambda(O, V)$ due to the $e^t \Sigma^{-1} e$ form in the multivariate normal density function.

Reduced support training When a new network is trained on the set "available-points" in algorithm 1, most often these points were generated by more than one function pattern. This means that a network trained to minimize the average error for the entire set is not likely to identify (approximate well on the entire domain) any of the generating functions. We try to solve this problem by allowing the network to choose its support (the set of points used to estimate its parameters) from the given training set. Algorithm 3 finds the support by iteratively excluding from the training set the points p whose errors are larger than the average network error but not smaller than a given threshold.

Algorithm 3 *Reduced support*

- *support* $\leftarrow$ *available-points*
- *repeat while the support changes*
  - *train the network on the current support with gradient descent until a local minimum is reached*
  - *exclude from support the point p with:*
    error(p) > average-error, error(p) > acceptable-error

Because we do not know how many computational units are needed to approximate well any of the generating functions, and because large networks are prone to overfitting, all networks are created with a predefined minimal architecture. The price we pay for this is that several small networks may be needed to approximate well any of the generating functions, with each network covering a subdomain of the function. This means that we may have a one-to-many rather than a one-to-one correspondence between the function patterns and the network states.

Model reduction – the second HMM, the final HMM/ANN

Since a function pattern may be represented by several network states, a better – cleaner, simpler – segmentation may be obtained by partitioning the set of network states into subsets, and associate each subset with a function pattern. These subsets are not necessarily disjoint: two distinct functions can be very close on some common subdomain, so one network may approximate well both of them on that subdomain. We find these subsets by inducing a discrete hidden Markov model (Rabiner 1989), θ^*, from V_λ, the sequence of network states. The network state identifiers in V_λ are the symbols observed by the discrete HMM θ^*. We call this model's states θ^*-states, to distinguish them from the network states. Each θ^*-state is considered to represent a function pattern. Because we do not know the number of function patterns, we must estimate from V_λ the number of states in θ^*. We can do this by finding the model θ^* with m parameters that satisfies Rissanen's (Rissanen 1984) minimum description length(MDL) criterion:

$$\theta^* = arg \min_{\theta, m} \{-\log P_\theta(V_\lambda) + \frac{1}{2} m \log T\} \qquad (5)$$

$P_\theta(V_\lambda)$ is the probability of the network state sequence V_λ under model θ, and T is the length of V_λ, the observed sequence for θ. Model θ^* is found by inducing for every k from 1 to n, the number of network states, an HMM θ with k states, and then selecting the model that minimizes the right side of expression 5. The number of parameters in a model θ with k states is $m = k * (1 + k + n)$, and $P_\theta(V_\lambda)$ is computed with the Baum-Welch algorithm. Let K be the number of states in θ^*. The segmentation of the time series is obtained by computing the Viterbi path, V_{θ^*}, of model θ^* for the sequence of network states V_λ induced by the initial HMM/ANN model . The final HMM/ANN model with K states is obtained by assuming that each θ^*-state corresponds to a function pattern – although we cannot expect a perfect correspondence –, and then applying algorithm 2 to the segmentation V_{θ^*}. The final networks are chosen larger than the initial networks, with their architecture again preselected. The pseudo-code of the model reduction algorithm is very simple:

Algorithm 4 *Model reduction*

- *induce* θ^* *from* V_λ *– determine its number of states* K *with the MDL criterion*
- *compute segmentation* V_{θ^*}
- $\lambda \leftarrow$ *arbitrary model with* K *states and given architecture;* $V_\lambda \leftarrow V_{\theta^*}$
- *estimate* λ*'s parameters with algorithm 2*

It must be noticed that we can now address the problem of estimating the network size: assuming that each θ^*-state corresponds to a function pattern, a network can be trained with all the points allocated to a θ^*-state in V_{θ^*}, including a regularization term in the cost to be minimized. This was not possible until now (step 3 in the main algorithm), because the network's support was not considered known, and the regularization term depends on the size of the training set.

It must also be noticed that the likelihood of the observed data O under the final HMM/ANN may be smaller than the likelihood under the initial HMM/ANN. This is because the number of network states was determined with a minimum description length, not a maximum likelihood criterion, and also because there is no guarantee that the final networks have smaller approximation errors than the initial networks. For now, we prefer to eventually give up in the final step a higher likelihood model, for a simpler one that produces a less complex segmentation.

Experimental results

To understand what our induction algorithm can do, we first applied it to artificial data. We present here two such experiments. For the first experiment, a time series with 257 data points was generated by a process switching between two function patterns, f and g:

$$x(t+1) = f(\cdot) = -1.05x(t) + .05$$
$$x(t+1) = g(\cdot) = \frac{-2x+5}{.5x^2(t)+1}$$

For both patterns the self-transition probability is .9. Normally distributed noise with variance .01 was added to the output. The initial networks have two hidden and one state cosine units. The initial HMM/ANN model, λ, found at the second step of algorithm 1, has seven network states: one network gets almost all of f's points and a couple from g, five networks get most of g's points plus a few from f, and one non-content state gets several points from both f and g. Because g is a more complex function, several simple networks are needed to estimate it. The final HMM/ANN model, θ, has exactly two network states, and the resulting segmentation, as it can be seen in figure 4 where the segmentation plots show the state indices along the two best paths, identifies almost perfectly the two generating functions. The two networks of model θ were given different architectures: the one that approximates f has the same simple architecture as the initial networks; the one approximating g has nine hidden and no state units. In several runs of the model reduction algorithms, this configuration yielded the smallest approximation errors.

For the second experiment with artificial data, four functions generated a time series with 582 points:

lin1: $x(t+1) = .9 * x(t) + .1$
lin2: $x(t+1) = .9 * x(t) + .5$
quad: $x(t+1) = -1.05 * x^2(t) - 1.75 * x(t) + .25$
frac: $x(t+1) = \frac{-1.75*x^2(t)-2.5*x(t)+.5}{1.5*x^2(t)+.5*x(t)+.17}$

Again, the self-transition probability was .9, and the noise variance was .01. Table 1 shows the allocation of the four functions points to the states of the initial model λ, and final model θ. Because the linear functions "lin1" and "lin2" are similar (same slope), in each model, one network – s1 in λ, and s2 in θ – approximates most of their points. In θ, network s1 gets almost all the points generated by the quadratic function "quad", and s3 of the fractional function "frac".

The algorithm was also applied to a time series collected during experiences involving a robot approaching or passing an object. There were 14 non-disjoint kinds of experiences –

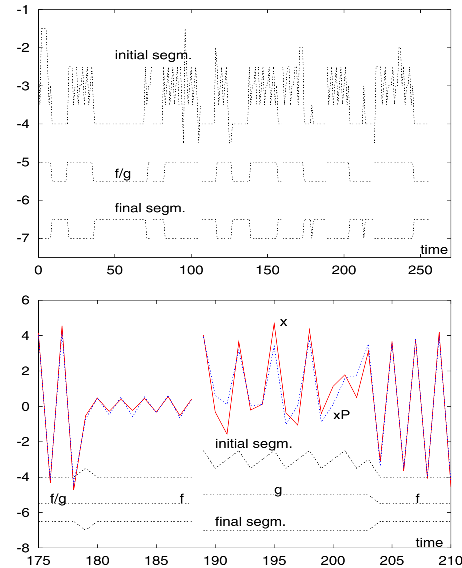

Figure 4: Top: "f/g" is the sequence of the generating functions f and g, "initial segm." is λ's best bath, "final segm." is θ's best path. Bottom: part of the generated time series, its approximation(prediction) by model θ, and the segmentations produced by the two models; "x" plots the observed data, and "xP" the predicted data.

λ	s0	s1	s2	s3	s4	s5	s6	s7	s8
lin1	0	71	0	0	0	0	2	20	0
lin2	0	133	0	79	0	0	0	0	0
quad	0	0	157	0	1	1	1	0	0
frac	16	0	2	0	24	21	5	1	48

θ	s0	s1	s2	s3	s4
lin1	5	0	88	0	0
lin2	0	0	133	0	79
quad	0	157	1	2	0
frac	8	1	0	108	0

Table 1: The allocation of the pattern points to the HMM states; s0 is the non-content state for both models.

"pass right a red object", "pass right a red object, then push a blue object", with several of each kind, totaling 1082 time steps and 42 experiences. Two thirds of the experiences were selected for the training set, and the remaining ones formed the test set. A model was induced from the training set for the task of predicting the next "vis-A-x" and "vis-A-y" sensor values from the current "trans-vel", "vis-A-x" and "vis-A-y" values. Sensors "vis-A-x" and "vis-A-y" return the coordinates of the center of an object in the robot's visual field, and "trans-vel" is the robot's translational velocity. Small networks with four hidden and two state cosine units were used. Only the initial HMM/ANN was induced (no clean-up of the resulting segmentation), with good results. In both the training and the test set the experiences of the same kind are segmented in the same way, i.e. they are represented by approximatively the same sequence of network states. During

the "pass-right" experiences, one network state, network 1, is active when the robot gets close to the object and passes it, and another, network 4, is active immediately after the robot passed the object. This indicates that these two patterns can be used to describe a concept like "approaching and passing an object", and as such are potentially useful building blocks in higher level representations of the robot's environment. The segmentations of four "pass right a red object" experiences, two in the training set and two in the test set, are shown in figure 5.

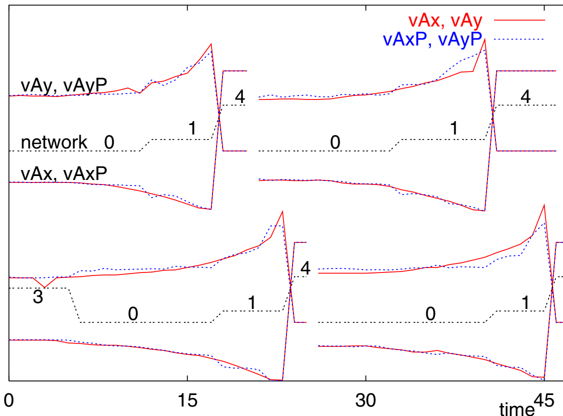

Figure 5: Induced segmentation of four "pass-right" experiences: top – from the training set, bottom – from the test set. The plots are: "vAx" is "vis-A-x", "vAy" is "vis-A-y"; "vAxP" and "vAyP" are the network outputs; "network" is the Viterbi path.

Related work

Many different hybrid HMM/ANN architectures have been developed, with the networks computing state transition probabilities or observation probability density parameters. These systems were successfully used in applications like speech recognition , or time series prediction. Among the latter, we are most interested in the work of Liehr, Kohlmorgen et. al. (Liehr *et al.* 1999) and Tani and Nolfi (Tani & Nolfi 1999). In both cases an ensemble of networks, called experts, is trained to predict the next observation, and the time series is segmented by soft competition among the experts. While their learning framework is similar with ours, there are some differences. The most important one is that their systems have a fixed number of experts, while in ours the number of networks is determined from the data. Another difference is that their systems employ a non-stationary, more complex model of the expert switching process. Liehr et. al. have a HMM whose transition probabilities are computed dynamically by a neural network, Tani et. al. have a neural network compute at every time step the expert activation probabilities. The prediction of their ensembles is a weighted sum of the individual expert predictions, with the weights depending on the dynamically calculated probabilities. It can be noticed that the non-stationary mixture of network outputs compensates for the fixed number of experts. While the mixture can be considered a more parsimonious representation, eventually identifying more complex concepts like drifts between regimes, the fixed maximum number of low level concepts can be a drawback.

Conclusions and future work

A new hybrid HMM/ANN system for segmenting time series was presented. The main difference from other HMM/ANN approaches is that the number of networks is not fixed, but induced from the data. In experiments with artificial data the algorithm identified almost perfectly the generating functions. Preliminary results with robot data suggest that the induced patterns can be associated with low-level concepts, and are thus potentially useful representation elements. These results are preliminary not only because we need more experiments, but also because the induction algorithms and the resulting concepts are not yet part of an architecture for predicting and controlling an agent's interactions with its environment. In future work we intend to develop a hierarchical architecture for prediction and control, with the model and algorithms described in this work forming the lowest level. We also intend to employ regularization methods to estimate from the data the network architectures. This is an important problem because a network's estimation and generalizations capabilities, and thus the quality of the induced concepts, depend on its architecture.

Acknowledgements

This research is supported by DARPA under contract DARPA/USASMDCDASG60-99-C-0074. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements either expressed or implied, of the DARPA or the U.S. Government.

References

Barron, A. R. 1993. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory* 39(3):930–945.

Liehr, S.; Pawelzik, K.; Kohlmorgen, J.; and Muller, K. R. 1999. Hidden markov mixtures of experts with an application to EEG recordings from sleep. *Theory in Biosciences* 118(3-4):246–260.

Pinkus, A. 1999. Approximation theory of the MLP model in neural networks. *Acta Numerica* 143–195.

Rabiner, L. R. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE* 77(2):257–285.

Rissanen, J. 1984. Universal Coding, Information, Prediction, and Estimation. *IEEE Transactions on Information Theory* 30(4):629–636.

Tani, J., and Nolfi, S. 1999. Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks* 12(7-8).