Segmenting Time Series with a Hybrid Neural Networks - Hidden Markov Model

Abstract

This paper describes work on a hybrid HMM/ANN system for finding patterns in a time series, where a pattern is a function realizable by a recurrent neural network embedded in the state of a hidden Markov model. The most likely (Viterbi) path of the hidden Markov model is used both for re-training the HMM/ANN model and for segmenting the time series into pattern occurrences. The number of patterns is determined from the data by increasing the number of networks as long as the likelihood of the segmentation increases. Preliminary results show that potentially useful patterns can be induced this way.

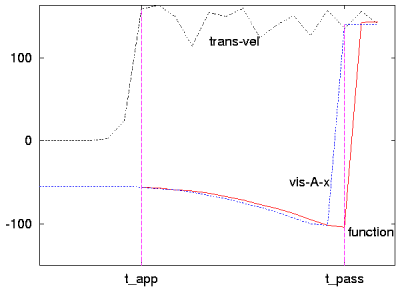

Figure 1. Sensor prediction during a "pass-right" experience; the function in this case is $\textit{vis-A-x}(t+1) = \textit{vis-A-x}(t)\left(1 - \frac{1}{1-c*\textit{trans-vel}(t)}\right)$.

1 Motivation

An agent embodied in a robot receives information about its interactions with the environment as time series of sensor vectors. As a first step in learning to represent the world, we want the agent to learn to identify segments of the time series that correspond to distinct parts of interactions, and create low-level concepts associated with these segments. For example, figure 1 shows that while approaching an object, the evolution of the *vis-A-x* sensor, the object's x-coordinate in the robot's visual field, is well predicted by a function $\textit{vis-A-x}(t+1) = f(\textit{trans-vel}(t), \textit{vis-A-x}(t))$, where *trans-vel* is the robot's translational velocity. When the robot passes the object, f's prediction is no longer accurate. We would like the agent to induce concepts like approaching or passing an object and recognize them in its future experiences. The example shows that a concept can be represented by a continuous function that predicts future sensor values. We will call such a concept a function pattern. The goal of this work is to induce this kind of patterns. Given a time series of sensor data, the learning task is to find an appropriate set of function patterns and to identify the times at which each pattern occurs, thus producing a segmentation of the time series.

Because they can approximate arbitrarily well continuous functions, we choose artificial neural networks (ANN) to represent the function patterns. There are certain regularities in the sequence of patterns – for example, during an object passing experience, a "no object" pattern usually follows an "approach" pattern, and we chose Markov chains to model these regularities. The resulting model is a hidden Markov model (HMM) whose states are neural networks: 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 goal of the induction algorithm is to find the model that produces the maximum likelihood segmentation.

2 The learning framework and algorithms

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 $x(1) \ldots x(t) \ldots x(T)$ with $x(t) \in R^n$, we assume that it was generated by a set of K processes – or function patterns, each process k being described by a continuous function $f_k = R^n \rightarrow R^d$:

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

with $e_k(t)$ a random normal variable representing noise. Variables x_{i_j}, whose probability densities are controlled by

functions f_k, are said to be the output, or predicted, 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. 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 $O = o(1)\ldots o(t)\ldots o(T)$. The likelihood of observing $o(t)$ within O, given function f_k, is the value of the multivariate normal density $N(e_k(t), 0, \Sigma^k)$:

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

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

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. This is a strong assumption that we intend to relax in future work, but for now it provides a computationally tractable model. Each function pattern k is a state of the Markov process and has a set $a_k = \{a_{k,l}, 1 \le l \le K\}$ of state transition probabilities. An additional parameter $a_{0,k}$ is the probability that k is the initial state. The resulting structure $\lambda = \{s_k = \langle f_k, \Sigma_k, a_{0,k}, a_k\rangle, 1 \le k \le K\}$ is a hidden Markov model ([3]) with K states $\{s_k\}$. Given a model λ, a state path $S = s(1)\ldots s(T)$ 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 because it is computed by the Viterbi algorithm [3].

We assume that the functions f_k can be computed by recurrent neural networks. Because neural networks can approximate arbitrarily well any continuous function, and even some discontinuous functions ([1]), 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.

2.1 Neural network 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 2 were chosen. The input and output units have identity

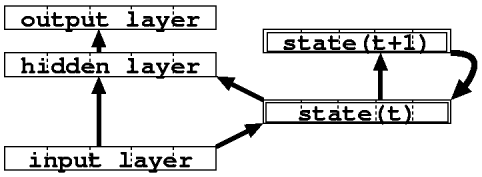

Figure 2. Recurrent network architecture: $s(t+1) = func(s(t), x(t))$ is the network's internal state, encoded in the recurrent units.

transfer functions, the hidden and state units have cosine activation functions. The cosine units can readily construct an approximating function – since they form a function's Fourier representation (see Barron [1]), and unlike the more popular sigmoidal units, they do not saturate during gradient descent. State units can also have identity transfer functions. The network architecture (size, topology, 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 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. This particular error function is the one that must be minimized when maximizing the likelihood $L_\lambda(O, V)$.

2.2 HMM induction

Given a segmentation V, and K functions $\{f^k\}$ implemented by neural networks, the model parameters $\{\Sigma_k, a_k\}$ can be estimated by maximizing the likelihood $L_\lambda(O, V)$. The maximum can be found by setting the partial derivatives of the likelihood to 0. 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 solutions are the average network errors:

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

where I_k is the set of (indices of) points allocated to network k in segmentation V. The solutions for the probability transitions are:

$$a_{k,l} = \frac{\#s_k \rightarrow s_l}{\#s_k^-} \quad (4)$$

where $\#s_k \rightarrow s_l$ is the number of transitions from state s_k to state s_l in path V, and $\#s_k^-$ is the number of occurrences of s_k in V (not counting V's last element).

2.3 Putting it all together: training both the networks and the HMM

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.

Algorithm 1 *Estimation of λ for maximizing $L_\lambda(O, V_\lambda)$:*

- *initialization: start with an arbitrary λ with m states*
- *repeat:*
  - *$\lambda \leftarrow$ the best model with m states (algorithm 2)*
  - *calculate V_λ*
  - *poorly predicted points along V_λ $\Rightarrow$ create p new networks and randomly assign these points to them (change segmentation), $m \leftarrow m + p$*

  until the likelihood stops increasing

A point is considered poorly predicted if its error is greater than a dynamically computed threshold.
Algorithm 2, which finds a (local) 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.

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

$L_\lambda(O, V_\lambda)$ increases at every iteration of algorithm 2:

- At step 1, $L_\lambda(O, V_\lambda) \leq L_{\lambda^*}(O, V_\lambda)$: training the networks reduces their errors along V_λ; the maximum likelihood estimates of the variances and transition probabilities are calculated 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: $L_\lambda(O, V_\lambda) \leq L_{\lambda^*}(O, V_\lambda) \leq L_{\lambda^*}(O, V_{\lambda^*})$.

3 Preliminary results

To understand what our induction algorithm can do, we first applied it to artificial data. A time series with 206 data points was generated by a process switching between two function patterns, f and g. For both patterns the self-transition probability is .9. Both functions are linear, f depends only on $x(t)$, g depends on both $x(t)$ and $x(t-1)$:

$$x(t+1) = f(\cdot) = -.95x(t) + .5$$
$$x(t+1) = g(\cdot) = 1.05x(t) - .95x(t-1) - .5$$

Normally distributed noise with variance .01 was added to the output. The networks are trained on the $\langle x(t), x(t+1)\rangle$ pairs. Because the domains and codomains of the two functions overlap, the task of the induction algorithm is not trivial. Nevertheless, the induced segmentation, shown in figure 3, identifies almost perfectly the two patterns. The first induced network state gets all the 125 points generated by f plus 2 points generated by g, and the other three induced states get the rest of g's points. All networks have two hidden units with cosine activation function and one state unit with identity activation function. Because f can be immediately computed from $x(t)$, a network approximating f was easily found by the algorithm. For g a network needs to save information about the previous value $x(t-1)$. Although each network in this experiment can approximate g well by storing $x(t-1)$ in its state unit, in several runs the algorithm found a three-network solution.
The algorithm was also applied to time series collected during experiences involving a robot approaching or passing an object. There were 14 kinds of experiences – "pass right a red object", "pass right a red object, then push a blue object", totaling 1082 time steps. 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; "vis-A-x" and "vis-A-y" are the coordinates of a red object in the robot's visual field, and "trans-vel" is the translational velocity. Small networks with four hidden and two state cosine units were used. The segmentations of four "pass right a red object" experiences, two in the training set (top plots) and two in the test set

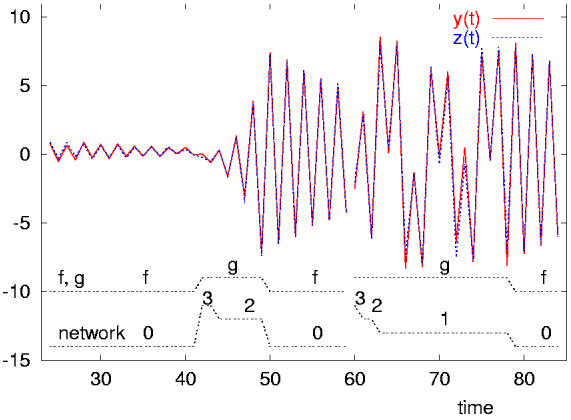

Figure 3. Artificial data: $y(t) = x(t+1)$; $z(t)$ is the network prediction; "f,g" is the function active at t, "network" is the Viterbi path.

(bottom plots), are shown in figure 4, where "vAx" is "vis-A-x", "vAy" is "vis-A-y", "vAxP" and "vAyP" are the network outputs, and "network" is the Viterbi path. The results are quite good, both for the training set and for the test set: experiences of the same kind are segmented in the same way, i.e. they are represented by approximatively the same sequence of networks. This indicates that the induced patterns can be used as building blocks in higher level representations of the robot's environment.

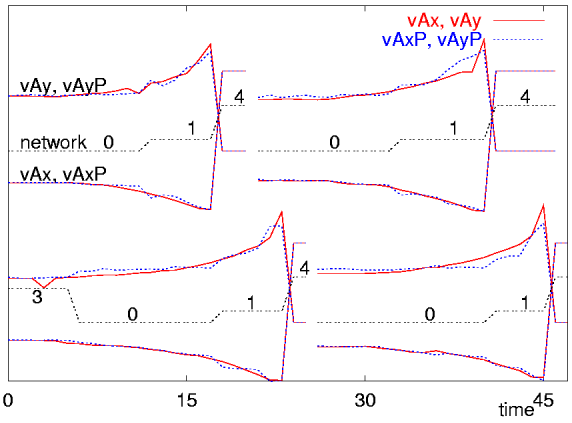

Figure 4. Sensor traces and induced segmentations of "pass-right" experiences.

4 Discussion and future work

Many different hybrid HMM/ANN systems have been developed, with the networks computing different probability parameters of the HMM states. Our architecture is novel in that the number of HMM states is determined from the data. The HMM/ANN models most similar with ours are those of Liehr, Kohlmorgen et. al. [2], and Tani and Nolfi [4]. In both cases an ensemble of networks, called experts, is trained to predict the next observation, and the time series is segmented by competition among experts. While their learning framework is similar with ours, there are some differences, the most important one being that their models have a fixed number of experts. Another difference is that they have a non-stationary, more complex model of the expert switching process: a neural network dynamically computes the expert activation probabilities, and the ensemble's prediction is a sum of the expert predictions weighted by these activation probabilities. While the non-stationary mixture of network outputs compensates for the fixed number of experts, and the mixture can be considered a more parsimonious representation of complex concepts like drift between regimes, the fixed maximum number of low level concepts can be a drawback.

The results we got with both artificial and robot data encourage us to pursue this work further. One issue we intend to address is the estimation of the network architecture from the data. Network size is important because it controls the segmentation: a large network may get all the points, with no distinct patterns identified in the time series; small networks may yield spurious segmentations because they cannot represent the underlying functions. Meaningless segmentations also result when the number of training examples is too small for a network's size. Regularization methods exist for adjusting the network complexity based on the size of the training set, but we need to understand how they can be applied in the case of variable training sets. Another direction we want to follow is the design of a model for the pattern switching process that not only relaxes the stationarity assumption, but also controls the induction of function patterns.

References

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

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

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

[4] J. Tani and S. Nolfi. Learning to perceive the world as articulated: an approach for hierarchical learning in sensory-motor systems. *Neural Networks*, 12(7-8):1131–1141, 1999.