UW CSE Technical Report 03-06-02
The Trajectory Mixture Model for Learning Collections of Nonlinear Functions

Aaron P. Shon Chris L. Baker David B. Grimes Rajesh P. N. Rao
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{*aaron,clbaker,grimes,rao*}@*cs.washington.edu*

July 8, 2003

Abstract

Learning statistical models of nonlinear dynamical systems has long been an important problem in machine learning. The problem becomes especially hard when the dynamical system is composed of a mixture of nonlinear models, not just a single nonlinear model. To address this general case of nonlinear time-series modeling, we propose a new hierarchical architecture: the Trajectory Mixture Model (TMM). The TMM learns collections of different nonlinear "trajectories" through state space. The model uses an expectation maximization (EM) algorithm to train a collection of nonlinear function approximators based on Gaussian radial basis function units. State densities are represented using samples estimated by particle filtering and smoothing. A sample-based representation provides an effective means of representing non-parametric state densities that change arbitrarily over time. We use entropy-based model selection to ensure that the individual function approximators, as well as the higher-level mixture model, do not overfit the data. Our results suggest that TMMs can learn complex nonlinear state space models directly from observations and may offer greater flexibility in modeling time-series data than existing methods such as extended Kalman filters and particle filters.

1 Introduction

Nonlinear dynamical systems are ubiquitous in nature. Such systems generate nonlinear "trajectories" over time through a state space. For example, kinematics of human and robotic limb movements, changing weather conditions, and the yearly acceptance rates at a conference can all be described using trajectories in a state space defined by a set of relevant variables. Given an initial location in state space, the states of such systems change over time according to some set of nonlinear differential equations. A particularly important class of systems is one in which a high-level model selects from one of several low-level models, which then generates a specific trajectory followed over time. A good example of such a system is speech, in which a high-level model may select a particular word and a lower-level model may follow the trajectory corresponding to that word. Such systems cannot be modeled using existing nonlinear state space techniques such as extended Kalman filters (EKFs) or particle filters [8, 5].

In this paper, we propose the Trajectory Mixture Model (TMM) to address the problem of learning dynamical systems composed of a mixture of nonlinear models. We introduce an EM algorithm [1, 6] for model learning and selection. Our algorithm offers several advantages over previous approaches:

1. **Flexibility:** The lowest level of the TMM consists of nonlinear function approximators that allow arbitrary state trajectories to be learned. Apart from using a sum squared error (SSE) loss function and being able to compute the entropy of the individual function approximators, the rest of the model is agnostic to the type of approximators used. Following Ghahramani and Roweis [4], we use radial basis function (RBF) networks as our function approximators.

2. **Improved efficiency in high-dimensional state spaces:** The method described in [4] did not adapt the basis function centers and covariances but relied on gridding the space with fixed hidden unit representations. In contrast, we combine an exact linear weight update step with a conjugate gradient-based algorithm for adapting our hidden unit centers and covariances in order to learn trajectories through high-dimensional state spaces.

3. **Computational parsimony:** The expressiveness and flexibility of the TMM necessitate model selection to avoid overfitting. This is done using concepts drawn from Rissanen's work on the minimum description length (MDL) principle [14]. At the lower level of the model, the algorithm adds Gaussian hidden units to each RBF network until the entropy of adding another hidden unit outweighs the information gained about the target trajectory by adding that new hidden unit. At the higher level, the algorithm adds new mixture components until the information cost required to add another component outweighs the benefit in accuracy derived from adding a new component.

2 The TMM Model

2.1 Nonlinear State Space Models

Throughout this paper, we write vectors as lower-case boldface letters, matrices as upper-case boldface letters, and scalars in italics. The state and observation equations for nonlinear state space models are given by:

$$\begin{aligned} \mathbf{x}(t+1) &= f(\mathbf{x}(t), \mathbf{u}(t)) + \mathbf{v}(t) \qquad (1) \\ \mathbf{z}(t) &= g(\mathbf{x}(t)) + \mathbf{w}(t) \qquad (2) \end{aligned}$$

where $\mathbf{x}(t)$ is an m-dimensional state vector, $\mathbf{z}(t)$ is an n-dimensional observation vector, and $\mathbf{v}(t)$ and $\mathbf{w}(t)$ are Gaussian-distributed white noise processes. The state dynamics and observation processes are governed by (potentially) nonlinear functions f and g, and $\mathbf{u}$ is an optional control input that may be a function of the current state (imposed by a higher-level control policy):

$$\mathbf{u}(t) = h(\mathbf{x}(t)) \qquad (3)$$

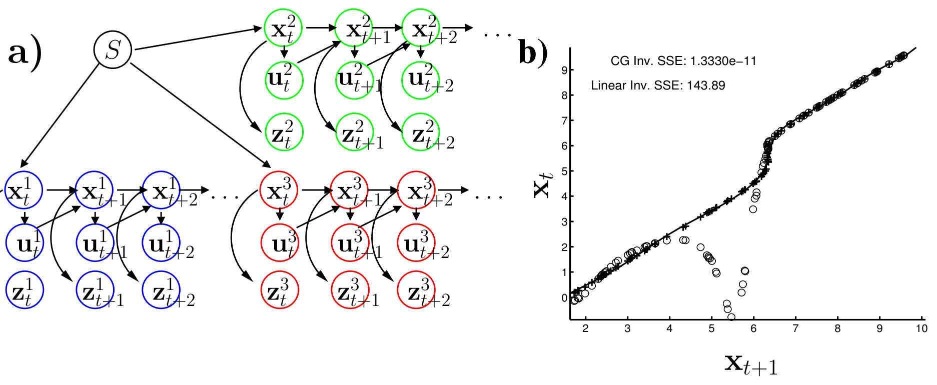

Figure 1: **Trajectory mixture model and inversion method:** (a) Graphical model underlying the TMM. The higher level node in the model (S) selects from one of the C different classes to determine the initial observation state. Each function approximator (depicted here in red, green, and blue) is responsible for learning a different trajectory class. (b) Using conjugate gradient (crosses) to invert RBF network outputs on nonlinear functions gives much more accurate smoothing estimates than linearization (circles).

2.2 Graphical Model for the TMM

Fig. 1 illustrates the graphical model underlying the TMM. The TMM assumes the existence of several independent nonstationary processes, or "trajectory classes," $S^1, S^2, \dots S^C$ through the state space. At the topmost level, a random variable S chooses one of these trajectory classes for each time series observed by the model. Each class $i \in 1 \dots C$ is learned by a separate function approximator f^i. Each of the different f^i generates a different trajectory $\mathbf{x}_2^i, \dots \mathbf{x}_T^i$, given an initial state $\mathbf{x}_1^i$ and a noise process $\mathbf{v}^i$. Additionally, at each time step t, the model may receive some input vector $\mathbf{u}_t^i$, which influences the state at the next time step. The results presented in this paper assume that no control inputs are provided. We therefore learn dynamical systems consisting of a mixture of nonlinear models, each given by:

$$\begin{aligned} \mathbf{x}(t+1) &= f^i(\mathbf{x}(t)) + \mathbf{v}^i(t) && (4) \\ \mathbf{z}(t) &= g^i(\mathbf{x}(t)) + \mathbf{w}^i(t) && (5) \end{aligned}$$

3 Model learning and selection

The TMM uses an EM algorithm to learn multimodal state dynamics for particle filters. This is a two-level process. The higher level involves estimating an occupancy matrix specifying conditional class probabilities that each trajectory class generated an observed sequence. The lower level involves fitting an individual function approximator for each trajectory class.

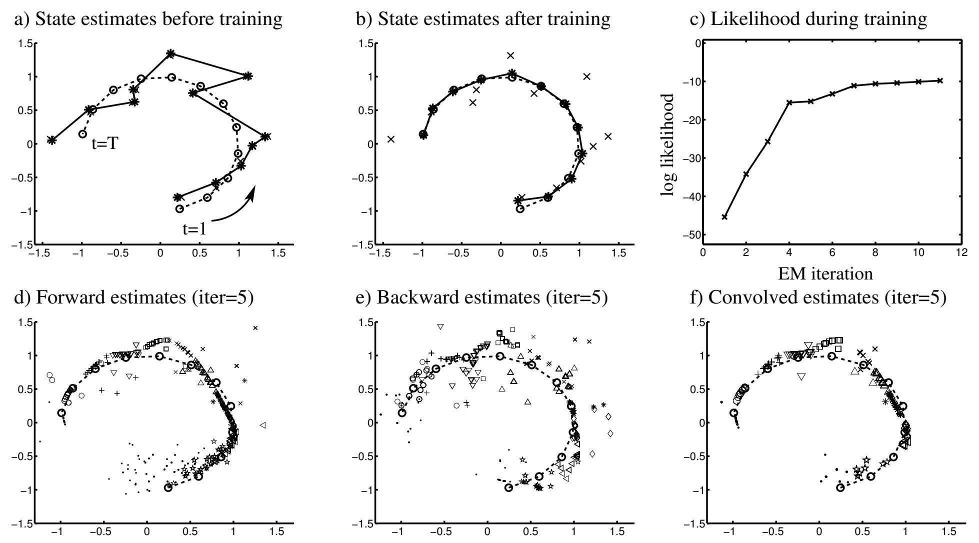

Figure 2: **Learning nonlinear particle trajectories:** (a) Plot of a 2D nonlinear trajectory in phase space. Before training, state estimates "*" lie along the noisy observations denoted by "x." Actual states are shown by bold circles, and the state trajectory (dashed curve) proceeds counterclockwise as shown by the arrow. The function approximator's estimate of the trajectory is shown by the bold curve. Despite a piecewise linear appearance, function approximators learn smooth functions. (b) After training a single function approximator on several noisy instances of circles, state estimates converge to the true state. (c) Log likelihood for a single RBF network function approximator over several iterations of EM. (d) Forward particle estimates for each state on EM iteration 5. Different symbols correspond to each different actual state. (e) Corresponding backward (smoothing) particle estimates on EM iteration 5. (f) Particle estimates derived from convolving sample sets shown in (d) and (e).

3.1 EM algorithm for the TMM

The lower level of the TMM consists of one or more Gaussian RBF networks, which compete to encode the trajectory classes. The basic learning algorithm for each RBF network is shown in Table 1. Our EM algorithm employs approximate E and M steps.

For the low-level RBF networks, the E step computes the conditional state density under that model given an observation sequence. The E step uses a sample-based forward-backward algorithm [13, 15]. The forward step (particle filtering) involves generating sample sets by sampling from an initial distribution π_f, then feeding the samples through the RBF network and adding Gaussian noise. For each sample point $\hat{\mathbf{x}}_f(t-1)$, this gives a corresponding predicted sample $\overline{\mathbf{x}}_f(t)$. The resulting sample sets are weighted by the observations and resampled to generate the forward state sample sets for each time step. To generate backward sample sets, the algorithm must produce an estimate of the RBF input $\hat{\mathbf{x}}_t$ for each corresponding output sample $\hat{\mathbf{x}}_{t+1}$ beginning with an initial sample set with distribution π_b. The traditional approach for finding inverses of nonlinear functions inverts a first-order Taylor expansion about a given backward sample point $\hat{\mathbf{x}}_b(t+1)$. To obtain a more accurate estimate of $\hat{\mathbf{x}}_b(t)$, the TMM iteratively finds better solutions using conjugate gradient on the error surface $\frac{\partial \epsilon}{\partial \hat{\mathbf{x}}_b(t)}$, where $\epsilon = (f(\hat{\mathbf{x}}_b(t)) - \hat{\mathbf{x}}_b(t+1))^T(f(\hat{\mathbf{x}}_b(t)) - \hat{\mathbf{x}}_b(t+1))$. The initial point for the gradient descent is given by $\mathbf{x}_b(t+1)$. The time-consuming nature of this step is mitigated somewhat by the fact that it must only be done during learning; on-line inference only requires the forward step (particle filtering).

These steps produce sample sets representing forward and backward state estimates. We use the

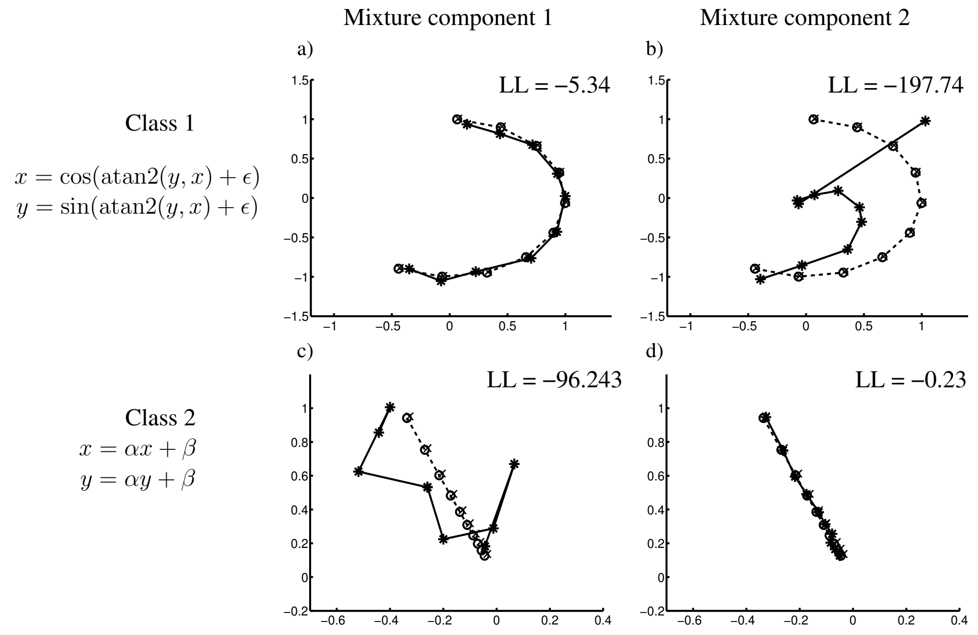

Figure 3: **Learning multiple nonlinear particle trajectories:** The model learns to assign trajectory classes to specific function approximators. (a),(d) Show inferred trajectories matched to their corresponding learned mixture components. (b),(c) Show how mixture components fail to infer trajectories from classes matched to other components during learning. See text for additional details.

method described by Thrun et al. [15] to convolve the forward and backward sample set estimates. The method uses density trees to convert from discrete samples to continuous probability densities. This convolution step generates the final set of samples representing the conditional state density for each time step given an observation sequence for each lower level RBF model.

The M step uses conjugate gradient to move the RBF hidden unit centers and covariances. For efficiency our implementation uses diagonal hidden unit covariance matrices. After each gradient step, the algorithm applies an exact weight update step (see equation (13) for derivations). The input-output training point pairs provided to the RBF are the most likely sample points computed in the E step for time steps t and $t + 1$, with a weighting factor on the error surface given by the class conditional probability (over RBF models) that this RBF model generated the input sequence.

3.2 Model selection

Our model uses the MDL principle of Rissanen [14], which states that the error obtained by a model learning a function should be balanced against the length of the code required to represent the model. For our RBF function approximators, we begin by providing each function approximator some base number of hidden units k. We train the model, then assess its SSE performance on the training data. We add a new Gaussian hidden unit $\mathcal{G}_{k+1}$ to the RBF approximator for class $i \in 1, \ldots, C$ if the SSE on the model trained with $k + 1$ hidden units plus the entropy H of the

$k+1$ unit is less than or equal to the SSE of the model trained with k hidden units:

$$\sum_{t=1}^{T_i} \left(\mathbf{e}_t^{i\mathbf{T}} \mathbf{e}_t^i\right) \geq \sum_{t=1}^{T_i} \left(\mathbf{e}_t'^{i\mathbf{T}} \mathbf{e}_t'^i\right) + H\left(\mathcal{G}_{k+1}\right) \tag{6}$$

$$H(\mathcal{G}_{k+1}) = \frac{1}{2}\left[\ln|\mathbf{\Sigma}_{k+1}| \cdot m \cdot (1+\ln(2\pi))\right] + n \cdot h_w \tag{7}$$

where $|\mathbf{\Sigma}|$ is the determinant of the $m \times m$ covariance matrix $\mathbf{\Sigma}$ and h_w is the entropy per weight connection. Again, the internal structure of our function approximators is irrelevant for the purpose of model selection; only the SSE and entropy of the function approximators must be known.

```
Input: A set of input sequences $\mathbf{Z}^1,\ldots,\mathbf{Z}^R$. Each sequence $\mathbf{Z}^a$ comprises an ordered set of observations $\{\mathbf{z}_1^a,\mathbf{z}_2^a,\ldots,\mathbf{z}_T^a\}$.
Output: A set of minimum-entropy RBF networks $f_{best}^i$ (and optionally, $g_{best}^i$), $i=1,\ldots,C$.

for $i=1,\ldots,C$                                   /* Initialize parameters */
  $H_{prev}^i \leftarrow \infty$;  $H_{curr}^i \leftarrow \infty$;  $k^i \leftarrow 0$;  $f^i \leftarrow$ init_RBF_network;
E step:
for $i=1,\ldots,C$                                   /* For each lower-level trajectory model: */
1. for $r \in \{1\ldots R\}$                         /* Loop over all input sequences */
   1) $\hat{x}_f^r(0) \leftarrow$ resample($\pi_f$)  /* Generate forward samples from prior distribution $\pi_f$ */
   2) for $t \in \{1\ldots$ length($\mathbf{Z}^r$)$\}$  /* Loop over all time steps in the $r$th sequence */
        $\overline{x}_f^r(t) \leftarrow f^i(\hat{x}_f^r(t-1)) + v_t$   /* Use RBF network to generate next sample set */
        $\tilde{x}_f^r(t) \leftarrow$ weight($\overline{x}_f^r(t), \mathbf{z}_t^r$)   /* Weight new samples according to observation */
        $\hat{x}_f^r(t) \leftarrow$ resample($\tilde{x}_f^r(t)$)   /* Importance resampling for forward estimate */
   3) $\hat{x}_b^r$(length($\mathbf{Z}^r$)$+1$) $\leftarrow$ resample($\pi_b$)   /* Generate initial backward sample set */
   4) for $t \in \{$length($\mathbf{Z}^r$)$\ldots 1\}$
        Use conjugate gradient to compute samples $\overline{x}_b^r(t)$ such that $f^i(\overline{x}_b^r(t)) \approx \hat{x}_b^r(t+1)$)
        $\tilde{x}_b^r(t) \leftarrow$ weight($\overline{x}_b^r(t), \mathbf{z}_t^r$))   /* Weight backward samples according to observation */
        $\hat{x}_b^r(t) \leftarrow$ resample($\tilde{x}_b^r(t)$))   /* Importance resampling for backward estimate */
   5) for $t \in \{1\ldots$ length($\mathbf{Z}^r$)$\}$   /* Convolve forward and backward sample sets using density trees to get final state distribution */
        $\hat{x}^{r,i}(t) \leftarrow$ convolve_samples($\hat{x}_f^r(t), \hat{x}_b^r(t)$)

M step:                                              /* Update RBF parameters based on outputs of E step */
for $i=1,\ldots,C$                                   /* For each lower-level trajectory model: */
1. for $r \in \{1\ldots R\}$                         /* Loop over all input sequences: */
   1) Compute mixing proportion $p_i^r = P(i|\mathbf{Z}^r)$
   2) $f^i \leftarrow$ weighted_conjugate_gradient($p_i^r, \{\hat{x}^{r,i}(1),\ldots,\hat{x}^{r,i}(T)\}$)
2. $f_{best}^i \leftarrow f^i$
3. $H_{curr}^i \leftarrow$ RBF_network_entropy($f^i$)
4. if $H_{curr}^i \leq H_{prev}^i$                   /* If entropy of the model (plus error) decreases */
     $\langle \mu_k^f, \Sigma_k^f \rangle \leftarrow$ init_RBF_unit($f, k$)   /* Add another RBF unit with mean $\mu_k^f$, covariance $\Sigma_k^f$ */
     $k \leftarrow k+1$
5. $H_{prev}^i \leftarrow H_{curr}^i$
```

Table 1: **The TMM Algorithm**

Higher-level model selection is performed by selecting a mixture model with lowest entropy. The entropy of the mixture model is computed by summing the entropy over all function approximators, plus the entropy of the occupancy matrix of class conditional probabilities. As with model selection in the low-level function approximators, we continue adding a new function approximator to the mixture until the entropy of the new function approximator exceeds the reduction in sum squared error resulting from addition of the approximator.

4 Results

Fig. 1(b) demonstrates how our conjugate gradient method for RBF inversion outperforms a simple linearization. The solid curve denotes a function in the $\langle \mathbf{x}_t, \mathbf{x}_{t+1} \rangle$ space learned by a single function approximator. Crosses plotted along the curve depict smoothing estimates on the function computed using conjugate gradient. Circles depict smoothing estimates for the same curve using linearization, evaluated about the same points as the conjugate gradient inverse. Note the lower SSE reconstruction for the conjugate gradient method.

The results shown in the current paper assume no external control inputs $\mathbf{u}_t$. We also assume for simplicity that observations are related to states by a multiple of the identity matrix, although it would be straightforward to learn a linear transformation mapping states into observations (or another nonlinear function approximator if a nonlinear transformation is known a priori to be likely). We trained the model on a handful of instances generated by a number of different linear and nonlinear functions. Fig. 2 shows how our function approximators learn nonlinear trajectories using EM. Fig. 2(a) shows a nonlinear trajectory in 2D phase space (dashed line), and the model's estimate of the trajectory before learning (solid line). Despite the piecewise linear appearance, the model learns a smooth nonlinear function due to the interpolation properties of the RBF network. Fig. 2(b) shows the estimated trajectory after training, (c) shows log likelihood increasing over several EM training iterations, (d) shows filtering estimates for each state after 5 EM iterations, (e) shows smoothing estimates for each state after 5 iterations, and (f) shows the sample set resulting from the convolution of the sample sets in (d) and (e) using density trees.

Fig. 3 shows an example of the model learning multiple trajectory classes. Mixture component 1 selects for circular trajectories, while mixture component 2 selects linear trajectories. The figure shows model performance after learning: (a) shows how the trajectory inferred by mixture component 1 follows the true trajectory (dashed line), while in (b) the mixture component that learned linear trajectories fails to infer the circular trajectory. Fig. 3(c,d) show similar results on a linear trajectory instance for the two mixture components. The figures contain log likelihoods (LL) of each mixture component generating the data.

5 Related Work

While several models have been suggested for learning nonlinear state space models, we are not aware of methods aimed specifically at learning continuous state space models composed of mixtures of nonlinear functions. For example, the model presented in [7] assumes switching state space dynamics but only learns a single nonlinear trajectory function. Systems such as the mixture of factor analyzers [2] learn mixtures of linear functions while recent work by Ghahramani and

Roweis uses extended Kalman smoothing and an EM algorithm to learn a single dynamical nonlinear function [4]. Hierarchical mixtures of experts (HMEs) [9] learn static nonlinear functions in a supervised manner using hierarchical local linear approximations.

Although the TMM is similar in spirit to the switching state space model (SSM) proposed by Ghahramani and Hinton [3], our model differs from the SSM in several ways. First, the TMM uses nonlinear function approximators rather than linear approximations. The TMM uses particle filters and does not assume a Gaussian state distribution. It also performs model selection to prevent overfitting. Finally, unlike [3], the TMM assumes no switching between trajectories over a single time course of data, i.e. the higher-level "switching" state stays fixed throughout all observations of a given trajectory until a "goal" state is reached. The usefulness of such a model for natural applications such as speech has already been mentioned.

The Monte Carlo HMM (MCHMM) [15] offers a mechanism for learning a single dynamic nonlinear function using samples. It relies on sampling to approximate all distributions of interest, including state transition functions. Particularly in high-dimensional spaces, over-reliance on sampling can limit performance. The function approximators in our model do not suffer from the potentially exponential blowup in complexity seen in purely sample-based approaches. Using a sample-based representation for transition functions is equivalent to using a look-up table to compute state transition probabilities, and therefore also lacks the interpolation properties inherent in function approximator models. Although the MCHMM includes model selection (in the form of shrinkage trees [11, 12]), the criterion used does not give an intuitive understanding of the tradeoff between generalization and accuracy. Our model uses an information-theoretic model selection criterion that more intuitively captures this tradeoff.

6 Conclusion

This paper proposes the Trajectory Mixture Model as a hierarchical framework for learning mixtures of nonlinear dynamical systems. To our knowledge, the proposed approach is the first to allow mixtures of particle filters to be learned. The model includes entropy-based model selection to avoid overfitting. We also employ conjugate gradient to achieve a better smoothing result than afforded by linearization. Although the model as currently stated uses particles to represent state densities, it is relatively straightforward to extend it to other state estimation frameworks, e.g. unscented Kalman filters [10].

We presented preliminary results illustrating the performance of the TMM in learning synthetic mixtures of nonlinear dynamical systems. We are currently investigating the applications of the TMM in speech processing, EEG-based brain-computer interfaces, and motor control in an active vision stereo head.

Appendix A: RBF learning algorithm

Following Ghahramani and Roweis [4], we use radial basis function (RBF) networks to perform nonlinear function approximation in the TMM. For greater efficiency, we also use a linear regression matrix $\mathbf{D}$ in our function approximation.

The RBF network model consists of an input layer, a hidden layer and an output layer. The input layer at time t is given by the data point $\mathbf{x}_t$. The hidden layer is a set of k Gaussian basis functions, where the basis function a is parameterized by a mean $\mathbf{c}^a$ and covariance matrix $\mathbf{S}^a$. The set of k means and covariance matrices in an RBF network are denoted $\{\mathbf{c}\}$ and $\{\mathbf{S}\}$ respectively. The output layer is parameterized by a linear weight matrix $\mathbf{W}$ of dimension $m \times k$, and a linear regression matrix $\mathbf{D}$. We denote column a of $\mathbf{W}$ as $\mathbf{w}_a$. The output of the network for data point $\mathbf{x}_t$ is denoted $\mathbf{o}_t$, and is a linear product of $\mathbf{W}$ with the output of the hidden layer, plus the linear product of $\mathbf{D}$ with $\mathbf{x}_t$.

$$\begin{aligned} h_t^a &= \exp(-(\mathbf{x}_t - \mathbf{c}^a)^T \mathbf{S}^a (\mathbf{x}_t - \mathbf{c}^a)) && (8) \\ \mathbf{o}_t &= \sum_{a=1}^{k} h_t^a \mathbf{w}_a + \mathbf{D}\,\mathbf{x}_t && (9) \\ &= \mathbf{W}\,\mathbf{h}_t + \mathbf{D}\,\mathbf{x}_t && (10) \\ \mathbf{e}_t &= \mathbf{y}_t - \mathbf{o}_t && (11) \\ \mathcal{E} &= \sum_{t=1}^{T} \mathbf{e}_t^T \mathbf{e}_t && (12) \end{aligned}$$

The weight matrix is set to minimize the sum squared error $\mathcal{E}$ with respect to $\mathbf{W}$. The exact least squares solution for the weight matrix is

$$\mathbf{W} = \left(\left(\sum_{t=1}^{T} \lambda_t \mathbf{h}_t \mathbf{h}_t^T \right)^{-1} \left(\sum_{t=1}^{T} \lambda_t \mathbf{h}_t \mathbf{o}_t^T \right) \right)^T \quad (13)$$

We minimize the sum squared error $\mathcal{E}$ with respect to $\{\mathbf{S}\}$ and $\{\mathbf{c}\}$ using conjugate gradient. The conjugate gradient algorithm requires computing jacobians and hessians of $\mathcal{E}$ with respect to all of the free parameters. To derive the conjugate gradient algorithm, we compute the quantities $\frac{\partial \mathcal{E}}{\partial \mathbf{c}^a}$ and $\frac{\partial^2 \mathcal{E}}{\partial \mathbf{c}^a \partial \mathbf{c}^b}$ (first and second partial derivatives of the error with respect to the mean(s) of units a and b), $\frac{\partial \mathcal{E}}{\partial \mathbf{S}^a}$ and $\frac{\partial \mathcal{E}}{\partial \mathbf{S}^a \mathbf{S}^b}$ (first and second partial derivatives of the error with respect to the covariance(s) of units a and b), and $\frac{\partial^2 \mathcal{E}}{\partial \mathbf{S}^a \mathbf{c}^b}$ (second partial derivative of the error with respect to the covariance of unit a and mean of unit b). Note that $\frac{\partial^2 \mathcal{E}}{\partial \mathbf{S}^a \mathbf{c}^b} = \frac{\partial^2 \mathcal{E}}{\partial \mathbf{c}^b \mathbf{S}^a}$.

$$\frac{\partial \mathcal{E}}{\partial \mathbf{c}^a} = \sum_t h_t^a \mathbf{S}_a(\mathbf{x}_t - \mathbf{c}_a)\mathbf{w}_a^T \mathbf{e}_t \tag{14}$$

$$\frac{\partial^2 \mathcal{E}}{\partial \mathbf{c}^a \partial \mathbf{c}^b} = \begin{cases} \sum_t (h_t^a \mathbf{S}_a(\mathbf{x}_t - \mathbf{c}_a)\mathbf{w}_a^T)(h_t^b \mathbf{S}_b(\mathbf{x}_t - \mathbf{c}_b)\mathbf{w}_b^T)^T \\ \sum_t (h_t^a \mathbf{S}_a(\mathbf{x}_t - \mathbf{c}_a)\mathbf{w}_a^T)(h_t^b \mathbf{S}_b(\mathbf{x}_t - \mathbf{c}_b)\mathbf{w}_b^T)^T \end{cases} \tag{15}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{S}a} = \sum_t h_t^a (\mathbf{x}_t - \mathbf{c}_a)(\mathbf{x}_t - \mathbf{c}_a)^T(\mathbf{w}_a^T)\mathbf{e}_t \tag{16}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{S}^a \mathbf{S}^b} = \begin{cases} \sum_t h_t^a h_t^b (\mathbf{x}_t - \mathbf{c}_a)(\mathbf{x}_t - \mathbf{c}_a)^T(\mathbf{w}_a^T)(\mathbf{x}_t - \mathbf{c}_b)(\mathbf{x}_t - \mathbf{c}_b)^T(\mathbf{w}_b^T) \\ \sum_t h_t^a h_t^b (\mathbf{x}_t - \mathbf{c}_a)(\mathbf{x}_t - \mathbf{c}_a)^T(\mathbf{w}_a^T)(\mathbf{x}_t - \mathbf{c}_b)(\mathbf{x}_t - \mathbf{c}_b)^T(\mathbf{w}_b^T) \end{cases} \tag{17}$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{S}^a \mathbf{c}^b} = \begin{cases} \sum_t h_t^a h_t^b (\mathbf{x}_t - \mathbf{c}_a)(\mathbf{x}_t - \mathbf{c}_a)^T(\mathbf{w}_a^T)\mathbf{S}_b(\mathbf{x}_t - \mathbf{c}_b)\mathbf{w}_b^T \\ \sum_t h_t^a h_t^b (\mathbf{x}_t - \mathbf{c}_a)(\mathbf{x}_t - \mathbf{c}_a)^T(\mathbf{w}_a^T)\mathbf{S}_b(\mathbf{x}_t - \mathbf{c}_b)\mathbf{w}_b^T \end{cases} \tag{18}$$

Appendix B: RBF inversion

To obtain estimates for the RBF input $\hat{\mathbf{x}}_t$ for each output sample $\hat{\mathbf{x}}_{t+1}$, we use iterative conjugate gradient steps to minimize $\mathbf{e}_t^T \mathbf{e}_t$ with respect to $\hat{\mathbf{x}}_\mathbf{t}$. The first and second derivatives are

$$\frac{\partial}{\partial \mathbf{x}_t}(\mathbf{e}_t^T \mathbf{e}_t) = -2 \sum_a h_t^a \mathbf{S}^a(\mathbf{x}_t - \mathbf{c}^a)\mathbf{w}_a^T \mathbf{e}_t \tag{19}$$

$$\frac{\partial^2}{\partial \mathbf{x}_t^2}(\mathbf{e}_t^T \mathbf{e}_t) = h_t^a \left[\mathbf{S}_a(\mathbf{x}_t - \mathbf{c}^a)(\mathbf{S}_a(\mathbf{x}_t - \mathbf{c}^a))^T + \mathbf{S}_a \mathbf{S}_a(\mathbf{x}_t - \mathbf{c}^a)\right] \mathbf{w}_t^a \mathbf{e}_t \tag{20}$$

$$- 2h_t^a \mathbf{S}_a(\mathbf{x}_t - \mathbf{c}^a)\mathbf{w}_a^t \left(\textstyle\sum_a h_t^a \mathbf{S}^a(\mathbf{x}_t - \mathbf{c}^a)\mathbf{w}_a^T \mathbf{e}_t\right) \tag{21}$$

Finding the inverse of an RBF requires computing one possible solution from a potentially infinite set of solutions. Sometimes, a possible inversion isn't found using conjugate gradient. In this case, we don't include the backward estimate we obtain among the set of backward estimates.

Appendix C: High Level Mixture Model

The higher level EM algorithm in its most general form maximizes the likelihood Q of the data given the model parameters Θ and the latent data H. This is equivalent to maximizing the log likelihood

$$Q = P(D|H, \Theta) \tag{22}$$

$$= \prod_{t=1}^{T} P(\mathbf{x}_t|H, \Theta)P(\mathbf{z}_t|H, \Theta) \tag{23}$$

$$\log Q = \sum_{t=1}^{T} P(\mathbf{x}_t|H, \Theta) \tag{24}$$

Acknowledgements: This work was supported by NSF grant no. 130705 and an NSF Career award to RPNR. APS was supported by an NDSEG fellowship. We thank Seth Bridges, David Hsu, Jonathan Ko, and Cody Kwok for their comments.

References

[1] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Soc., Series B*, 39(1):1–38, 1977.

[2] Z. Ghahramani and G. E. Hinton. The EM algorithm for mixtures of factor analyzers. Technical Report CRG-TR-96-1, University of Toronto Dept. of Computer Science, 1996.

[3] Z. Ghahramani and G. E. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):831–864, 2000.

[4] Z. Ghahramani and S. Roweis. Learning nonlinear dynamical systems using an EM algorithm. In *Advances in Neural Information Processing Systems*, volume 11, pages 431–437. MIT Press, 1999.

[5] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *IEE Proceedings on Radar and Signal Processing*, 140:107–113, 1993.

[6] H. Hartley. Maximum likelihood estimation from incomplete data. *Biometrics*, 14:174–194, 1958.

[7] A. Honkela. Nonlinear switching state space models. Master's thesis, Helsinki University of Technology, 2001.

[8] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[9] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[10] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, 1997.

[11] A. McCallum, R. Rosenfeld, T. Mitchell, and A. Y. Ng. Improving text classification by shrinkage in a hierarchy of classes. In J. W. Shavlik, editor, *Proc. 15th International Conf. on Machine Learning*, pages 359–367. Morgan Kaufmann, San Francisco, CA, 1998.

[12] S. M. Omohundro. Bumptrees for efficient function, constraint, and classification learning. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 693–699. 1991.

[13] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77, 1989.

[14] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[15] S. Thrun, J. C. Langford, and D. Fox. Monte Carlo hidden Markov models: Learning non-parametric models of partially observable stochastic processes. In *Proc. 16th International Conf. on Machine Learning*, pages 415–424. Morgan Kaufmann, San Francisco, CA, 1999.