Robot environment learning with a mixed-linear probabilistic state-space model

William Robert Chesters

Thesis submitted for the degree of Doctor of Philosophy
University of Edinburgh, 1999

Abstract

This thesis proposes the use of a probabilistic state-space model with mixed-linear dynamics for learning to predict a robot's experiences. It is motivated by a desire to bridge the gap between traditional models with predefined objective semantics on the one hand, and the biologically-inspired "black box" behavioural paradigm on the other.

A novel *EM*-type training algorithm for the model is presented, which is less computationally demanding than the Monte Carlo techniques recently developed for use in (for example) visual tracking applications. The algorithm's E-step is slightly approximative, but an extension is described which would in principle make it asymptotically correct. Investigation using synthetically sampled data shows that the uncorrected E-step can in any case make correct inferences about quite complicated systems.

Results collected from two simulated mobile robot environments support the claim that mixed-linear models can capture both discontinuous and continuous structure in the world in an intuitively natural manner; while they proved to perform only slightly better than simpler autoregressive hidden Markov models on these simple tasks, it is possible to claim tentatively that they might scale more effectively to environments in which trends over time played a larger role. Bayesian confidence regions—easily supported by the mixed-linear model—proved to be an effective guard for preventing it from making over-confident predictions outside its area of competence.

A section on future extensions discusses how the model's easy invertibility could be harnessed to the ultimate aim of choosing actions, from a continuous space of possibilities, which maximise the robot's expected payoff over several steps into the future.

Acknowledgements

I owe a large debt of thanks to my supervisor Gillian Hayes, both for her advice, and for her patience and support. In addition the work has benefited greatly from discussion with Chris Williams, and the thesis itself from the thoughtful comments of its examiners Peter Ross and Ben Kröse.

Finally, I'm grateful to EPSRC for an Advanced Studentship grant which enabled me to follow the Ph.D. programme of which this thesis is the outcome.

Table of Contents

Chapter 1

Introduction

One of the great arguments in contemporary robotics, still continuing a decade after it was initiated by Brooks[1], concerns the nature and role of representation: whether a robot should strive to maintain an objective model of its environment, extracting from its sensorium a picture of how the world really is; or on the other hand shun any such notion as a delusion born of our human tendency to reification, relying instead on superficial and as far as possible memoryless rules to get by; or perhaps call its world a complex dynamical system, and deploy neural networks or evolutionary algorithms to develop a "black box" oracle, which is capable of predicting its experiences, or of recommending what action it should take, but whose internal semantics are deliberately left obscure.

Cogent arguments can be marshalled for and against each of these positions. It is certainly true that explicit, objective representations, in terms familiar from the way we talk about the world ourselves, are very difficult to construct, and in any case often turn out to be a fragile and inconvenient basis for supporting the generation of behaviour. More opportunistic methods, like those advocated by Brooks, are better able to take advantage of untidy and shallow—but useful and robust—regularities in the environment. On the other hand, it is also clear that the robot's controller will often need to be furnished with internal state in some form or another. And, if the talk is of harnessing the subjective and idiosyncratic phenomena of the robot's world, it makes plausible sense to go further and allow the robot to learn (or evolve) a means of exploiting them for itself, by any means

[1] Brooks, *Intelligence without representation*

necessary. Yet one must at some point confront the scientific question: how are we to understand what is then going on in the robot? If we abandon the attempt to make robots know the world in the way we do, how can we make sense of the idea that they know it at all?

The motivation behind the work reported here is to help narrow the gap between objective representations of the robot's world on the one hand, and black box dynamical systems models on the other. Within the framework of probabilistic modelling, it is possible to unify the two and understand how a robot can learn a representation and a model of its world in a way which is simultaneously adaptive, in that the form taken by the representation is driven by what is found useful in practice; open in principle to analysis and understanding; and, in an interesting and strong sense, rational. This is the line taken by, among others, the Brown University robotics group[2]. The present thesis describes a probabilistic model which aims to capture the complex, continuous environmental dynamics faced by real robots at the low level, making as few assumptions as possible about how the quantities in play (such as sensor readings) should be interpreted—but inferring in the process a representational scheme at once subjective and reasonably transparent. It can be seen as a kind of missing link between explicit models used for tasks such as map-building with "opaque" neural networks.

Chapter 2 reviews three important sub-fields of robot learning: mapping, reinforcement learning, and neural network system identification. Chapter 3 discusses the properties which an environment model must have if it is to achieve the ends set out above, and introduces the theory of Bayesian probabilistic reasoning on which (it is argued) it must be based, with particular reference to model confidence regions, and to dynamical systems models including the hidden Markov model and Kalman filter. Chapter 4 proposes that a mixed-linear probabilistic state-space model has many of the required attributes, and presents a novel algorithm called "Samovar" for learning and performing inference with it; by drawing on the representational schemes and learning algorithms of both the hidden Markov model and the Kalman filter, Samovar is (it is claimed) positioned to inherit the strengths, and some of the tractability, of both. Connections are drawn between Samovar and related techniques in the literature—including the Condensation algorithm, which has been used to learn a similar class of model. Chapter 5 analyses Samovar's behaviour using synthetically generated time series data, before evaluating its performance on the task of predicting the experiences of two simulated robots. Chapter 6 points out how the model and learning algorithm could be improved and extended, and suggests a way in which the invertibility property of one variant of the mixed-linear model could be used as the basis for a reasonably efficient (albeit theoretically suboptimal) algorithm for planning actions to achieve the robot's goals.

[2] *e.g.* Basye *et al.*, *Learning Dynamics*

A glossary of the terms and symbols used in the text, together with pointers to their first uses, is given in appendix A. Works cited are referenced via the formula "authors, *first few words of title*", which keys into the full list provided at the end of the thesis. Numbers in parentheses refer to equations, which are numbered sequentially without regard to the chapter in which they appear.

Chapter 2

Robot Learning

Chapter 1 outlined the goal of this thesis, namely the development of a probabilistic model for the low-level dynamics of a robot's environment. The general theory on which the model is based will be described in chapter 3; this chapter reviews the three main fields specific to robotics which intersect with or touch on the problem of probabilistic environment modelling.

First, techniques for autonomous *localisation and mapping* are surveyed: some of the most sophisticated "explicit" models of real robot environments have been developed for handling this important problem. Next, a summary is given of the application of *neural networks* to the task of low-level environment modelling. If both these problems are cast in terms of probabilistic learning and inference, their essential similarity becomes explicit and space for a "missing link" between them is opened up.

Finally, the area of robot decision-making is considered: the work presented in this thesis is concerned only with learning an environment model, and not directly with using it to guide a robot's behaviour, but the question of how this could be achieved will be addressed in section 6.2, for which *reinforcement learning* and the related Bayesian theory of acting under uncertainty are the essential background.

2.1. Localisation

One of the competences generally agreed to be most desirable for an autonomous robot is self-orientation: the ability to find its way around. Industrial robots have for many years been solving the problem of locating themselves on a predefined map by enlisting the help of external navigational aids such as radio, laser or barcode "beacons", smooth floors which make it feasible to move considerable distances by dead reckoning on the basis of odometry information, and wires laid in the floor for detection by inductive sensors. It's obviously interesting to look at ways of doing this in a less structured environment, and of learning the map so that it need not be known in advance; the latter is perhaps the single most intensively researched problem in robot environment modelling, so it is reviewed here as background for the techniques (not specific to mapping) to be introduced later.

2.1.1. Unstructured environments

Where the designer does not have tight control over her robot's environment, she is unable to engineer out two sources of uncertainty which render these brittle approaches ineffective: odometry readings becomes much less reliable, accumulating errors over time, and the unambiguously locatable beacons against which they could be corrected are replaced (if she is lucky) by hard-to-detect and mutually confusible landmarks. The challenge is to fuse the hints from all the robot's sensors into a robust estimate of its true position, using rough odometry data to disambiguate information about possible landmarks, and conversely using the conclusions thus reached to compensate for dead-reckoning drift.

For reasons which will become clear when Bayesian inference is discussed in section 3.2, the most successful approaches work by interpreting the inevitable uncertainties in a probabilistic framework, against the background of a stochastic model specifying roughly how the quantities in play are believed to relate to each other. It is then easy to write down the right thing to do, in principle, using Bayes' rule. Adopting the following notation[1]

H^t	random variable: the robot's position/orientation at timestep t
r^t	the robot's sensor readings at timestep t
A^t	the robot's motor commands at timestep t
$p(r^t \mid h^t)$	probability density: how likely readings are in each position
$p(h^{t+1} \mid h^t, a^t)$	prob. density: how likely new positions are after taking action in old one

[1] see also appendix A

the inference one should make about one's new position H^{t+1} on the basis of one's new sensor readings r^{t+1} and previous experiences $a^{[0,t]}, r^{[0,t]}$ consists in the probability distribution

$$\begin{aligned} p(h^{t+1} \mid r^{t+1}, a^{[0,t]}, r^{[0,t]}) &\propto p(r^{t+1} \mid h^{t+1}) p(h^{t+1} \mid a^{[0,t]}, r^{[0,t]}) \\ &= p(r^{t+1} \mid h^{t+1}) \int_{h^t} p(h^{t+1} \mid h^t, a^t) p(h^t \mid r^t, a^{[0,t-1]}, r^{[0,t-1]}) \quad (1) \end{aligned}$$

where first Bayes' rule (section 3.2.2.2) and then the sum (marginalisation) rule (3) have been invoked. Note that this rule can be applied inductively, since the term $p(h^t \mid r^t, a^{[0,t-1]}, r^{[0,t-1]})$ is just the analogous distribution obtained at the previous timestep for H^t.

To use this "Markov localisation" formula in practice, it is necessary to find a way of expressing h^t, the dynamical rule $p(h^{t+1} \mid h^t, a^t)$ and the output rule $p(r^t \mid h^t)$ which make sense with respect to the robot's exact aims and the sensorium at its disposal, together with a way of working with them which makes the integration over h^t feasible.

2.1.1.1. Kalman filtering and uncertain geometry

One immediately familiar scheme is to encode the robot's position in Cartesian space in a real vector h^t, and model its movements as a linear difference equation

$$H^{t+1} = \begin{pmatrix} I & \lambda \end{pmatrix} \begin{pmatrix} H^t \\ A^t \end{pmatrix} + N(0, \alpha)$$

where A^t is known to determine the distance and direction by which the robot moves between the two timesteps via the linear map λ, and $N(0, \alpha)$ is an anonymous zero-mean Gaussian random variable (noise). (To extend this scheme to cases where A^t instead influences the robot's speed and bearing, it would be necessary only to augment the state H^t to include the current velocity.) In the (unlikely) event that the robot's sensor readings were more or less directly related to its position (and possibly velocity or acceleration), according to another linear/Gaussian mapping κ

$$R^t = \kappa H^t + N(0, \beta)$$

—*e.g.* they were odometers and range sensors operating in a convex rectilinear arena—the solution to (1) would be a Gaussian and could be calculated straightforwardly using the well-known Kalman recursions (section 3.3.3.5).

In more general cases, it is possible to get this approach to work by using a technique called "scan matching". It is assumed that the most probable position suggested by the odometry information is close to the truth, and the map is used to compute a first-order local expansion of how the values expected from the robot's sensors (range sensors or vision-based object detectors) would vary as the robot's position deviated from that point. This linearisation plays the role of κ and the sensor noise model that of β.[2] The correction to the robot's position distribution in the light of the scan fit is calculated as above (the approximative framework going by the name "extended Kalman filter" or EKF), and could be propagated back into the past using the Rauch recursions, although the original work by Durrant-Whyte[3] and subsequent extensions by others[4] use slightly different formulations.

The major weakness of any method which relies on a unimodal (*e.g.* Gaussian) distribution for representing its beliefs about the robot's position is that this approximation will only ever be remotely true when the robot is able to pin its location down pretty closely. It suffices for correcting small errors in odometry before they build up disastrously, but is inappropriate for initial localisation when the prior distribution over H^0 is wide, or for recovering after a loss of registration. However, by modelling the different ways in which world features such as walls and corners are likely to follow on from one another, Cox and Leonard are able to construct a tree of hypotheses within each of which the robot's sensor readings *can* realistically be fused using a EKF.[5]

2.1.1.2. Markov grids

Another way of encoding the robot's position (and perhaps orientation) is to quantise it onto a grid; the Markovian rule will then be something like "the robot may move to one of the blocks near to the one it is currently in, depending on the action it takes". This makes the dynamics density $p(h^{t+1} \,|\, h^t, a^t)$ a known discrete distribution for each given a^t, so that the integral in (1) is just a summation. As long as the output distribution $p(r^t \,|\, h^t)$ is known, therefore, the solution of (1) is not in principle difficult; and the advantage of a grid-based method is that $p(r^t \,|\, h^t)$ can take any form whatsoever: it is just a record of what sensor readings are to be expected in each grid square, which (as noted above) need not necessarily be related in any systematic way with the robot's actual position. So this technique can handle arbitrary "landmarks", in the most general sense of static environmental features giving rise to sensor readings which allow the robot to reduce significantly the entropy of its belief distribution over its possible locations (but which may

[2] Gutmann *et al., An Experimental Comparison*

[3] Durrant-Whyte, *Consistent integration and propagation*

[4] Lu & Milios, *Globally consistent range scan alignment*; Gutmann & Konolige, *Incremental Mapping*; Fusiello & Caprile, *Synthesis of indoor maps*

[5] Cox & Leonard, *Modeling a dynamic environment*; Leonard *et al., Underwater Sonar Data Fusion*; this work is interestingly related to the recursive mixed-linear models discussed in section 4.1 and section 4.3

not be perfectly interdistinguishable).[6] As will be shown in section 2.1.2, this is the basis of many successful map-learning methods in which the observation densities are conditioned on the robot's position through such varied models such as multi-layer perceptrons[7], kernel-based PCA regression[8], and kernel-based regression controlled by multi-layer perceptrons[9], as well as models of the properties of various kinds of range sensor, and can be seen as a biologically plausible account of animal localisation[10].

Crucially, the grid representation also makes it possible to entertain a multimodal belief distribution over the robot's position: if landmark information is currently not sufficient to disambiguate two regions that look the same, both hypotheses will be kept alive until some observation is made which one predicts well and the other does not; as was noted above, the Kalman filter and similar formalisms must be augmented with relatively complicated mechanisms for explicitly representing divergent interpretations of the world if they are to avoid the brittleness arising from reliance on a unimodal distribution.

Of course, if the grid is made too fine, the summation (integral) in (1) will become too time-consuming, so there is a difficult tradeoff between the acuity with which it is desired to estimate the robot's position and the constraints of CPU time and memory, both of which rise exponentially with the resolution. On the other hand, Gutmann and co-workers have shown that if the robot is most of the time reasonably sure of its position, it is possible to prune the vast majority of the states out of the sum.[11]

2.1.1.3. Monte Carlo methods

Researchers at Bonn and Carnegie Mellon universities have recently introduced an interesting way[12] of sidestepping the tradeoff between resolution and tractability which afflicts grid-based representations of the robot's position: abandon the attempt to perform the integration in (1) in closed form, and instead approximate the expectation by a weighted sum over a suitably-generated *sample* of hypothetical grid positions h^t. This can be done

[6] *Cf.* the criterion developed in Vlassis *et al.*, *An information-theoretic localization criterion.* Defining landmarks any other way is difficult; see *e.g.* the discussion of "Local Distinguished Places" in Kuipers & Levitt, *Navigation and mapping*, and footnote 1 of Thrun, *Bayesian Landmark Learning*

[7] Thrun, *Bayesian Landmark Learning*

[8] Vlassis & Kröse, *Robot Environment Modeling*, Kröse *et al.*, *Appearance based robot localization*

[9] Oore *et al.*, *A mobile robot that learns its place*

[10] Hermann *et al.*, *Self-Localization*

[11] Gutmann *et al.*, *An Experimental Comparison*

[12] Dellaert *et al.*, *Using the Condensation Algorithm*

by starting with a large cloud of "particles" drawn from the prior distribution $p(h^0)$, and then, whenever the robot moves, passing each particle through the stochastic dynamics several times, and resampling from the resulting set of position predictions with probabilities proportional to those with which each accounts for the observed data (see also section 4.3.3). Over time, the particles in play will become concentrated around a small number of high density regions (*e.g.* just one) as the true distribution is cut down by the data. If enough particles are tracked, it can be shown[13] they can be used as good estimators of any desired statistic of $p(h^{t+1} \,|\, r^{t+1}, a^{[0,t]}, r^{[0,t]})$. Because the Monte Carlo filtering process (stochastically) concentrates its attention on a relatively tiny number of the most significant possibilities, it is much more computationally tractable than an exact algorithm running on an equivalent grid size; and it was shown to perform impressively on the task of localising and then tracking a mobile robot equipped only with unreliable odometry and an intensity map of its building's roof. The method has been extended to the problem of handling dynamic environments in which sensor readings are affected by third parties coming between the robot and the fixed environment[14], and most recently to the problem of learning about the environment discussed in section 2.1.2[15].

2.1.1.4. Topological techniques

If the available landmarks are scattered sparsely across the environment, then it may be better to connect them via a sparse, so-called topological[16] representation, in some kind of graph, rather than in a dense "metric" one: a bitmap or set of overlapping depth scans. Sparse methods are common in the literature on visual navigation based on high-level geometric recognition of fixed objects.

2.1.2. Learning the environment

Alongside inferring its position within a pre-described environment, the other desirable skill in an autonomous robot is that of mapping the world for itself. If the robot is able to perform a calibration run, during which it is granted accurate knowledge of its position, then the aim is simply to merge noisy and/or partial sensor readings covering the same areas; this is essentially the dual of the problem of localisation from a known map discussed in section 2.1.1. For instance, from sonar or other range sensors, it is relatively easy for the

[13] Isard & Blake, *Condensation*

[14] Fox *et al.*, *Markov Localization*

[15] Thrun *et al.*, *A real-time algorithm*

[16] note that this term is often used of schemes which do include metric information, against one of its common meanings

robot to construct a two-dimensional probabilistic representation of the distribution of open space as against walls and other large objects in its immediate vicinity[17], and then, following Moravec[18], use Bayes' rule to piece together a series of overlapping local maplets recorded in various places at different times into a global "occupancy grid"[19].

But in general this option is not available to a robot exploring a new environment with complete autonomy: if it cannot leverage perfect knowledge of its position to help it in the map-building process, the problem of localisation using unreliable sensor information familiar from section 2.1.1 recurs—only this time the map itself is also uncertain, and the resulting chicken and egg dilemma appears unbreakable.

2.1.2.1. Local matching

One way to work around this is to use previously measured sections of the global map as a template against which to fit corresponding new local ones. Combined (albeit in a not obviously Bayesian way) with prior information about the relative orientations of walls (*i.e.* parallel or perpendicular), this technique has been found to give good results (by, for example, Thrun[20]). However, since estimates once integrated into the map are never revised, this can only counteract a part of the buildup in positional error, and there is a danger that the map will end up not only warped but, if the environment contains a cycle, potentially inconsistent.[21]

2.1.2.2. Global landmark matching

It's better to update past estimates of the robot's pose, and hence of the mapping inferences based on them, in the light of subsequent observations. All the localisation techniques outlined in section 2.1.1 can be adapted to perform this reverse inference. Many workers have used either scan matching or landmark-based techniques to build up networks of spatial relationships between robot poses at different times, which have then been globally optimised in the maximum likelihood sense.[22]

[17] although the effect of noise, sensor calibration and specular effects means that this isn't quite a trivial task; see Thrun, *Learning Metric-Topological Maps*, section 2.1 for a brief review and a solution using multi-layer perceptrons

[18] Moravec, *Sensor fusion in certainty grids*

[19] Elfes, *Sonar Based Real World Mapping*

[20] Thrun, *Learning Metric-Topological Maps*

[21] Gutmann & Konolige, *Incremental Mapping*; Thrun *et al.*, *A real-time algorithm*

[22] Durrant-Whyte, *Consistent integration and propagation*; Lu & Milios, *Globally consistent range scan alignment*; Koenig & Simmons, *Passive Distance Learning*; Fusiello & Caprile, *Synthesis of indoor maps*; Gutmann & Konolige, *Incremental Mapping*; Thrun *et al.*, *A real-time algorithm*

2.1.2.3. EM *Mapping*

With the popularisation of the *EM* algorithm (section 3.3.1), it has become almost a reflex to consider it as the obvious best solution to chicken and egg problems; and indeed there are several examples in the literature of true *EM* learning applied to concurrent mapping and localisation. Here the map plays the role of the unknown model parameter and the robot's position that of the hidden data, and the algorithm involves alternately

- computing a distribution $p(h)$ for the robot's position at each point in time on the basis of the current map estimate and the known sensor readings (E-step), and
- reestimating the map assuming that the location density history is in fact optimal (M-step).

The reason why this works well is that the maximum likelihood criterion is good at identifying previously identified map features in a slightly unexpected place: the hypothesis that a known feature is responsible for the sensor readings gives them a higher likelihood than is obtained by postulating a previously unobserved nearby feature, even when the data are consistent with the latter view.

In applications which involve estimating the positions of discrete (but not mutually distinguishable) landmarks, *EM* has been shown by Thrun *et al.* to work well over a grid representation of the robot's (and the landmarks') positions[23], and by Shatkay *et al.* to be equally applicable to a graph-based[24] one[25]. In both cases the robot's location is represented discretely, so that the well-known forward-backward equations (section 3.3.3.3) can be used for the E-step.

Oore *et al.*[26] also use a discrete grid, and suggest that it is not necessarily critical to implement the backward as well as the forward pass of the standard E-step, as long as the M-step (in this case the training of a multi-layer perceptron) increases the overall likelihood[27]. Other authors have, however, found that failure to make inferences backwards in time can lead to problems in mapping cycles in the environment.[28]

[23] Thrun *et al.*, *A Probabilistic Approach*; this is arguably best seen as a variational algorithm rather than an *EM* one, in that the M-step computes a probability distribution over the landmark locations rather than a maximum likelihood estimate (binary occupancy grid)

[24] topological but including metric information

[25] Shatkay & Kaelbling, *Learning Topological Maps*; Shatkay & Kaelbling, *Heading in the Right Direction*, which interestingly uses the von Mises' or circular normal distribution for expressing angular uncertainty

[26] Oore *et al.*, *A mobile robot that learns its place*

[27] for the generalised view of the *EM* algorithm on which this assertion rests, see section 3.3.1.3

[28] *e.g.* Thrun *et al.*, *A real-time algorithm*

2.1.2.4. Bringing together dense and sparse representations

An algorithm based on applying naive *EM* directly to an occupancy grid would require a great deal of pragmatic pruning to achieve acceptable performance, and appears not to have been reported; but Burgard *et al.* obtain good results with a two-stage approach which first constructs a series of small local maplets, whose internal consistency is assured by the fact that odometry errors are bounded over short timescales, and then using *EM* to optimise their global positions.[29] This can be seen either as a good way to perform occupancy scan matching, or as a way of obtaining a set of landmarks (the maplets) to combine in a maximally consistent spatial relations graph. Conversely, Thrun *et al.* use a graph-type map, learned by *EM* optimisation over a landmark occupancy grid, as a framework on which to hang a dense map constructed by Kalman filter scan matching.[30]

Another point at which the graph-based and occupancy-based approaches meet is in a nice procedure for obtaining the former kind of map from an instance of the latter, applying geometric algorithms based on Voronoi diagrams to a learned occupancy grid in order to compute a topological representation of the connectivity of the environment[31] in terms of its choke points (*e.g.* doors). This furnishes the robot with a much more efficient data structure for the purposes of applying standard navigation planning algorithms.

2.1.2.5. Monte Carlo mapping

The powerful technique of Monte Carlo localisation opens up the possibility of estimating the robot's pose in a continuous space, and correcting it with the help of information from dense sensor scans without having to resort to brittle approximations such as the extended Kalman filter (section 2.1.1.1). It has been applied by Thrun *et al.* to the problem of map-building, with impressive results.[32] A reasonable-sized set of particles sampled using the motion and observation distributions according to the algorithm defined in section 2.1.1.3 can stand in for the true posterior distribution—however multimodal and non-Gaussian—when it comes to computing statistics such as the most probable pose of the robot at each step in time, and hence, given range scan data and the observation model describing what it measures, a probabilistic representation of the shape of the environment.

[29] Burgard *et al.*, *Sonar-based mapping*; it proves necessary to use a technique called deterministic annealing which can be considered as a means of flattening out the peaks in the likelihood landscape temporarily so that a hill-climbing algorithm can pass over them (although this is not how the authors appear to understand what they have done)

[30] Thrun *et al.*, *Integrating Topological and Metric Maps*

[31] Thrun, *Learning Metric-Topological Maps*

[32] Thrun *et al.*, *A real-time algorithm*

In fact, the maximum likelihood pose could simply be approximated by the most probable pose in the sample set. However, the probability concerned has to be conditioned on sensor readings succeeding the timestep as well as on those made before; and it is currently a recognised weakness of particle filter techniques that they are unable to implement the necessary backward recursion efficiently.[33] Apparently for this reason, the authors instead proceed by explicitly detecting the situations in which historical revision is necessary, *i.e.* when closing a loop in the environment and returning to a previously mapped area, but slightly out of registration with it. First, an estimate is made of the absolutely most probable pose for the robot before a sensor scan is taken at its new position, obtained by starting a hill-climbing optimiser off at every pose sample in the new set and recording the best answer it finds. (It is not made clear exactly on what basis the likelihood gradient is approximated using, presumably, the previous step's pose sample.) Next, a similar best estimate is made of the pose, but now conditioned also on the new scan. If the robot is re-observing a location in which it has found itself before, and its position estimate has drifted in the meantime, the scan will conflict directly with that previously recorded, and the second position estimate will be dragged significantly away from the first in order to bring them into registration. When this happens, the displacement is distributed evenly between all the moves the robot believes it has made since the last time it was here, and the roughly corrected estimates for each timestep in the loop are used as a seed for the same gradient ascent procedure. The eventual result is a good maximum probability estimate of the robot's position at each point in history. The authors acknowledge that a stricter *EM* algorithm might be even more robust but point out that their method is fast and offer as evidence for its efficacy an accurate 3D map of a large building interior.

2.1.2.6. Appearance-based methods

Some kinds of sensors, such as video cameras, provide readings which take the form of a vast array of numbers from which semantically useful information cannot be extracted in any straightforward way. Direct learning of $p(r^t \mid h^t)$ will then be impossible, and some kind of preprocessing will have to be applied to reduce the dimensionality. The upside of using a less restricted sensorium than the usual range sensors is that in realistic environments, which typically contain many highly distinctive landmarks if only one can recognise them, it can take a lot of strain off the fusion mechanism by making perceptual aliasing much less likely.

One interesting piece of work[34] attacks this problem by deploying a battery of multi-layer perceptrons to act as stochastic "feature detectors". When the robot wishes to make

[33] Isard & Blake, *A smoothing filter*, p. 8; North *et al.*, *Learning and classification*, p. 26; see also section 4.3.3.2

[34] Thrun, *Bayesian Landmark Learning*

an observation, it applies each network i to a vector of seven general properties computed from a camera image, and treats its output as the likelihood $p(r_i^t \,|\, h^t)$ of feature i being present in the image, the required likelihood $p(r^t \,|\, h^t)$ being assumed to be the product of these marginals. To train the networks to pick out "useful" landmarks, a calibration run is made during which images are collected and labelled with the robot's true position (on a one-dimensional grid representing places along a corridor); an iterative optimiser is deployed to adjust the neural networks' parameters so as to minimise the expected absolute error in the estimate a robot would make of its position if it were deposited at one of the sampled locations, shown the corresponding sensor reading, and asked to compute from it the probability that it was at each of them. Essentially the landmark-learning problem is reduced to classification, with a penalty for confusing two places proportional to the distance between them.

As an aside, this arguably solves slightly the wrong problem. The authors are correct when they claim that the procedure will (with enough data, and subject to the practical effectiveness of the neural net training) learn landmarks which are optimal for the problem of localising a robot which has a given fixed, prior belief distribution about where it is, and a single sensor snapshot taken at its true position. It will tend to give priority to extracting features which make it unlikely that two widely separated landmarks will be confused. But strictly, if one wishes to target absolute localisation error, a more appropriate measure with respect to the goal of "lifelong localisation" might be *e.g.* the expected absolute localisation error summed over several timesteps, or something similar. And then the most common case will (with any luck) be one in which the robot does have a reasonably good idea where it is, so that it is unlikely to have any trouble distinguishing between well-spaced locations; the emphasis will then switch to fine-tuning the distinctions between adjacent points which might genuinely both seem seem plausible at once.[35]

A better approach suggested by Vlassis and colleagues involves optimising the reduction in entropy with respect to the prior distribution over positions which will on average be obtained from each observation, assuming that the positions and readings obtained during the calibration run are representative.[36] The observation model first uses PCA to reduce (unconditionally) the dimensionality of the images obtained from a camera—a relatively efficient way of extracting features from complex data—and then fits a linear model based on

[35] It might not be difficult in principle to extend the algorithm to take this objection into account, using the existing machinery as the M-step of an EM-type algorithm. The theoretically optimal measure could only be calculated using a simulation or real-world trial involving the robot's actual behaviour policy (*c.f.* Kaelbling *et al.*, *Planning and Acting*).

[36] Vlassis *et al.*, *An information-theoretic localization criterion*

Gaussian kernels in pose space to model the appearance of the environment at each point.[37] By assessing separately the effect of each principal component on the expected localisation entropy, it is possible to select those which are most invariant with respect *e.g.* to changing light conditions.[38]

It's interesting to compare these schemes for learning an array of orthogonal, but cooperatively deployed, feature detectors with recent work on "products of experts".[39] One of the most interesting features of the latter framework is that the formulation implicitly gives rise to a tendency for the experts—actually not generative models in their own right but feature detectors—to diversify so that they specialise in different situations. This phenomenon is explained briefly in section 6.1.2.3; the rightmost term in equation 119 in which it is manifested has a similar function to the term $P(f)$ in equation 34 of Thrun, *Bayesian Landmark Learning.*

Appearance-based approaches have also been generalised to mitigate further the effects of perceptual aliasing by adopting techniques from active vision.[40]

2.1.2.7. Unsupervised neural networks for landmark learning

The problem of landmark learning has also been addressed through the use of neural networks of the "unsupervised competitive learning" type[41]. An appealing aspect of this school—the emphasis on speedy learning algorithms—is typified by Duckett & Nehmzow, *Performance Comparison*, which plots the performance of and computational resources required by several different methods in a realistic navigation experiment.

The simplest unsupervised nets are essentially prototype-based classifiers: for instance, the Reduced Coulomb Energy network employed in Kurz, *Constructing maps* implements a nearest-neighbour rule in sensor space to extract landmarks from the robot's experiences. Many others are variants on the adjustable-prototype clustering paradigm; Hertz *et al.* point out that what they call the "standard competitive learning rule" is equivalent to the k-means algorithm (and hence similar to the *EM* mixture classification algorithm described in section 3.3.2.1). The ART2 (Adaptive Resonance Theory) network, which has been proposed as an engine for finding landmarks as clusters in sensor space,[42] incorporates a

[37] Vlassis & Kröse, *Robot Environment Modeling*; also Pourraz & Crowley, *Continuity Properties* and deVerdière & Crowley, *Local Appearance Space*

[38] Kröse *et al.*, *Appearance based robot localization*

[39] Hinton, *Products of experts*

[40] Kröse & Bunschoten, *Probabilistic localization*, Fox & Burgard, *Active Markov Localization*

[41] *e.g.* Hertz *et al.*, *Introduction to the theory of neural computation*, chapter 9

[42] Racz & Dubrawski, *Artificial neural network*

mechanism for limiting the number of clusters between which the data are divided. However, the criterion is motivated by neurobiological rather than statistical considerations; and indeed it appears that ART is not "consistent" in the sense of converging, in the infinite limit, to a true representation of some underlying property in the data: the pattern of clusters it finds depends strongly on the order of presentation of the training data.[43] More interestingly, the Self-Organising Map or Kohonen network[44] implements a kind of clustering with topological constraints, which could be compared with the intertwined landmark learning and localisation algorithms discussed in section 2.1.2.2.

2.2. Neural networks for dynamics learning

Since connectionist models began to show some promise during the mid-1980s, a large literature has grown up around the subject of using neural nets for nonlinear system identification. Only a brief overview of this field will be given here, with a view to drawing connections with the work presented in this thesis.

2.2.1. Multi-layer perceptrons

By far the most widely used type of neural network (and not only in the process modelling field) is the multi-layer perceptron or MLP.[45]

2.2.1.1. Static MLPs

The inspiration behind MLPs was a (deliberately simplified) description of how nerve cells in animals propagate activity amongst themselves. Each unit (cell) receives signals telling it the activation levels of all the other units which are connected to it through a directed link, amplified or attenuated according to the links' "weights". The incoming signals are summed and passed through a softened version of a thresholding function, such as a logistic, and the result determines the unit's own activation level. (It's easy to see that this is closely related to the logistic regression model used by statisticians to describe the

[43] Sarle, *Why Statisticians Should Not FART*

[44] Hertz *et al.*, *Introduction to the theory of neural computation*, p. 236

[45] Bishop, *Neural Networks* is one of the current standard texts, placing MLPs and radial basis functions firmly in the framework of statistical inference

relationship between a dichotomous response variable and a set of explanatory variables.) In the simplest case, the units are arranged into a series of one or more layers (typically three), with an activation-propagating link from every unit in each layer to every unit in the successor layer. Inputs, such as explanatory variables or sensor readings, are presented to the network in the activiation levels of the first layer of units; the activation is allowed to propagate through to the last layer and read off as the network's outputs. It can be shown that given enough units in the middle, "hidden" layer, and the freedom to adjust the weights parameter, it is possible to get an MLP to approximate any piecewise continuous function to arbitrary accuracy over a given domain.

The process of training a neural network from example input and output vectors, adjusting its weights so that when presented with inputs in future, it produces an output which follows the same pattern, is best seen as an instance of maximum likelihood parameter estimation.[46] First, a measure is defined of the distance between each example output and the network's output when presented with the corresponding input—for instance, the sum of the squared differences between the components of each—and summed over the whole set examples to form an error function. If the network is considered as a stochastic mapping, which generates outputs by adding diagonal symmetrical Gaussian noise samples to the activation levels of its final layer, the error function is proportional to the negative log likelihood of the weights parameter given the example data. Then, a hill-climbing optimiser is deployed to find a minimum of the error function by adjusting the weights; the result will be a maximum likelihood parameter.

Because the activity vector of each layer depends on that of the previous one through a linear-logistic function, it turns out to be quite straightforward to propagate the derivatives of the error function backwards through the net in an efficient way, and they are on the whole quite well-behaved, so that iterative optimisers based on quadratic extrapolation can give good results quite quickly. (Because the parameter space of even a moderate-sized MLP is large, variants of the method of conjugate gradients are widely preferred for their modest memory requirements.) The most sophisticated training algorithms treat the evolution of the network's weights during training as a state-space model, and treat it in an extended Kalman filter formalism.[47]

Obviously, the log of any reasonable exponential-family distribution will give rise to a perfectly usable error function if the Gaussian noise assumption is inappriopriate. The scale parameters of the noise distribution can be reestimated from the residuals; if necessary,

[46] section 3.2.4 2

[47] Puskorius & Feldkamp, *Decoupled extended*; for the state of the art, see deFreitas *et al.*, *The EM Algorithm and Neural Networks*

the network then can be retrained in the light of the adjust noise model, and so on to convergence.

The main difficulty that arises with neural network learning is overfitting of the data. If the hidden layer is large enough to implement more or less poorly-behaved nonlinear mappings, and the weights are allowed to grow without limit, then it will be possible to find a parameter which can account perfectly for any moderately sized set of examples, but which fails to interpolate smoothly between the points pinned down by the training inputs. Various *ad hoc* solutions to this problem have been proposed; one of these, known as "weight decay", can be seen to be equivalent to placing a Gaussian Bayesian prior on the weight parameter.

2.2.1.2. Bayesian methods

Indeed, since full Bayesian methods avoid (in principle) the overfitting syndrome to which MLPs are especially prone entirely—by refraining from adopting a single best estimate of the parameter—a lot of effort has gone into finding ways to apply them. MacKay demonstrated a way of approximating a full Bayesian inference over a space of MLPs of fixed architecture, interleaving gradient-based parameter updates with reestimations of both the noise distribution and the parameter prior, and then marginalising over a Gaussian approximation to the parameter posterior.[48] Neal applied the hybrid Markov chain Monte Carlo numerical integration algorithm to the same problem, representing the posterior parameter space by a sample.[49] More recently, groups at (for example) Cambridge University have proposed reversible jump Monte Carlo methods for sampling from a posterior space covering nets with different numbers of hidden units;[50] they also propose a hybrid of EKF training and Monte Carlo sampling importance resampling[51].

2.2.2. MLPs for modelling dynamics

Traditionally, neural networks researchers have considered the problem of modelling dynamics systems in the context of "recurrent networks" of neuron-like units; but there is another, arguably more principled possibility available.

2.2.2.1. Recurrent MLPs

MLPs can be generalised to handle time series prediction and system identification by relaxing the restriction that activity-propagating links can only point forwards through

[48] MacKay, *A Practical Bayesian Framework*

[49] Neal, *Bayesian Learning*

[50] Andrieu, *Robust Full Bayesian Methods*

[51] deFreitas *et al.*, *Sequential Monte Carlo methods*

the net, from the input end towards the output end. The network then becomes itself a (discrete) nonlinear dynamical system, with feedback loops, but a maximum likelihood parameter can still be estimated by minimising an appropriate training error (negative log likelihood) over an example series of system outputs.[52] At the simplest, the recurrent network can be "unfolded" over time to form a non-recurrent MLP which embodies the same dynamics, albeit over a finite sliding time window; then the usual gradient backpropagation will work fine. For cases in which one is interested in trying to infer temporal relationships over unbounded periods, Williams and Zipser[53] showed how to compute the error gradients directly in their "Real time recurrent learning" algorithm; Narendra and Parasathy[54] introduced a scheme called "Dynamic back-propagation" which achieves faster performance by framing the derivatives themselves as a recursively evolving quantity.

2.2.2.2. Using MLPs in an EKF

The problem with using the dynamics of the net itself to stand in for those of the system being modelled is that there is no way of representing uncertainty about the current state; the only means which the net has of making uncertain predictions is via its output noise, which is of course constant, while the output uncertainty should ideally vary depending on how precisely the system state is known. In general, there are three problems to solve:

1) representing and making uncertain estimates of the system's hidden state at each timestep from examples of its behaviour
2) making uncertain predictions from that uncertain state estimate
3) learning a model parameter which can handle 1 when the model is being used

For linear/Gaussian state-space models, a precise and comprehensive solution is available in the shape of the Kalman filter (section 3.3.3.5); the so-called extended Kalman filter, in which a nonlinear model is simply linearised for the purposes of propagating Gaussian uncertainty, can of course be used with MLPs to achieve (2) and (3), with the usual caveats about the drastic approximation on which it relies. Nelson and Wan show how two EKFs can be run in parallel, one to estimate the state of the system during the training sequences and the other to optimise the network parameters by EKF training in the light of that state estimate;[55] this is easily seen to be (an approximation to) an *EM* algorithm, like the standard one for estimating Kalman filter parameters. It is possible that techniques for

[52] For a concise application-oriented survey, see Tutschku, *Recurrent Multilayer Perceptrons.*

[53] cited in Tutschku, *Recurrent Multilayer Perceptrons*

[54] cited in Tutschku, *Recurrent Multilayer Perceptrons*

[55] Nelson & Wan, *Neural Speech Enhancement*

avoiding the worst consequences of the EKF's assumption of model linearity and Gaussian state uncertainty[56], or even out-and-out particle filters (section 4.3.3), could be adapted for use with MLPs.

2.2.3. Robotics applications of neural networks

MLPs are quite widely used in robotics applications, for tasks such as inverse kinematics of robot arms or wheeled vehicles, sensor interpretation (*e.g.* the navigation examples mentioned in section 2.1.2), trajectory planning, *etc.*[57] It is possible to discuss recurrent neural nets and the environments they model or control in terms of coupled dynamical systems; for instance Tani has a robot learn to match its behaviour to a stable attractor in its task space.[58] The use of neural networks as "function approximators" in reinforcement learning is discussed in Sutton and Barto's standard textbook.[59]

2.3. The theory of acting

Robotics theoreticians have always been interested in the general problem of how a robot should choose its actions. Until relatively recently, the focus tended to be on techniques for searching large, but deterministic state spaces, much as a chess computer searches for a good move. In the last decade, the complementary problem of how to choose actions in simple, but uncertain situations has received a lot of attention, generally from the standpoint of Bayesian decision theory (discussed briefly in section 3.2.6.2).

The work presented in this thesis is not directly concerned with action selection, but rather with learning a more or less goal-neutral internal model of the robot's environment. So only a brief overview is given of the field, summarised from the standard text on reinforcement learning by Sutton and Barto[60]. The reader is referred to that text for further material and an extensive bibliography.

[56] for instance Julier & Uhlmann, *A General Method*

[57] Narendra, *Neural networks for control*

[58] Tani & Fukumura, *A Dynamical Systems Approach*

[59] Sutton & Barto, *Reinforcement Learning: An Introduction*

[60] Sutton & Barto, *Reinforcement Learning: An Introduction*

2.3.1. Stochastic worlds: reinforcement learning

Researchers into human and animal behaviour have long studied what natural agents actually do when faced with the task of achieving their ends in an unknown and unpredictable world, starting with the (now very large) literature on "conditioning". It was, however, from the operations research community that a framework began to emerge in the context of which it was possible to say what the *right* thing to do is.

2.3.1.1. Markov processes

At the heart of the modern conception of the problem is the idea of describing the world as a Markov decision process, *i.e.* a system which is, at each timestep, in one of a set of possible states, and moves to a new one according to an arbitrary probability distribution conditional only on its current one and on an input, or action, fed into it by the agent.[61] The agent's goals are encoded by assigning a relative benefit or cost to the system's being in each particular state, or to the performance of each action in each state; the aim is then to choose at each timestep an action which is optimal as measured by, most commonly, the total of future net gains it will on average yield, discounted exponentially according how far ahead they lie. (This yardstick has obvious applications in economics and business decision-making.)

In principle, no loss of generality is implied by this representation, and its simplicity makes it ideal as a test bed for theoretical analysis. It can be applied directly to nontrivial robot control problems provided that the state space has been rendered discrete and reasonably small by adding an abstraction layer to shield the decision-maker from the full complexity of the world as perceived through the robot's sensors. Large and even continuous spaces can be made somewhat manageable if some extra structure is introduced, as in the engineering literature on optimal control.

2.3.1.2. Optimal policies

Given a decision process of known character, the simplest algorithms for computing optimal policies, *i.e.* assignments of recommended actions to states, work by starting with an arbitrary policy and iteratively increasing its effectiveness as follows:

- "policy evaluation": work out the worth of being in each state (or taking each action in each state) in terms of its net discounted future reward, on the assumption that the policy is already optimal

[61] strictly, this is a first order Markov decision process

- "policy improvement": adopt a new policy which chooses the action achieving the best net discounted future reward from each state, according to those state value assessments

It can be shown that this bootstrapping process (reminiscent of the *EM* algorithm discussed in section 3.3.1) will converge to an optimal policy. The policy evaluation step can be carried out with an iterated dynamic programming technique using the dynamics of the decision process to bring about local consistency, or simply by the Monte Carlo method of recording the payoffs obtained in practice after visiting each state; it turns out that this step need not be carried to full convergence (*c.f.* *EM* with partial *E*-steps, section 3.3.1.3).

2.3.1.3. Unknown environments

Because the dynamics of the decision process are not used in Monte Carlo policy evaluation, the method can be applied when no model of the environment is available (although they still require that the agent be able to distinguish reliably between different world states). However, they suffer from disadvantages arising from the requirement that estimators of the state values under the current policy must be collected over an extended period before the policy is improved. A class of "temporal difference" algorithms avoids this by using the difference between the rewards predicted and those observed empirically over some window following a visit to a step to drive improvements to the state value table. The most sophisticated of these, called $TD(\lambda)$, uses "eligibility traces" to enable it to take into account reward differences extended indefinitely into the future, discounting by a factor λ.

2.3.1.4. Model-based learning

It is also possible to obtain an optimal policy for an unknown decision process by interleaving the learning of an explicit model with the optimisation of a state value table and policy conditioned on it. A policy-neutral model has the advantage that it can, if the agent's aims change, help estimate a policy which is optimal with respect to the new state-reinforcement assignments without the need for a further extended period of exploration.

2.3.2. Partially observable worlds

If the agent does not know directly what state the world is in, but can only observe evidence for it in the form of outputs produced according to a state-dependent probability distribution, the problem of choosing optimal actions becomes much more difficult. It is no longer sufficient to construct a policy from a state or state-action value table; projections must also be made of the effect of actions and likely observations on the agent's belief state: for instance, it is preferable to end up in a known, moderately good state than to be in either a very good or very bad one and not be sure which. Of course, this situation, known as "perceptual aliasing" in the reinforcement learning community, is endemic in robotics, since robot sensors are rarely all-seeing.

What to do about partially observable Markov decision processes has been studied in the context of robotics by Whitehead and Ballard[62], and more recently by a group at Brown university[63]; the latter have demonstrated exact algorithms for planning in the discrete case.[64] Essentially, the idea is to treat the agent's belief distribution about which state it is in (which is uniquely determined by the Bayesian laws of uncertain reasoning) as a continuous Markov variable.[65] A cheaper solution is to include a general penalty for actions which result in nonspecific (high-entropy) belief distributions; this is much easier to assess, and is related to the use of entropy as a guide in deciding which sensor readings to take or which features to extract from sensor data.[66] It's possible that a Monte Carlo approach could be adopted, at least for planning over relatively short timescales (section 6.2.2.2).

2.4. Summary

The aim of the work presented in this thesis is to develop a probabilistic model which can be used in a similar way to the neural networks described in section 2.2—for learning the low-level, continuous dynamics of a robot's environment in a semantically neutral manner—but which is as open to scientific understanding and as well-founded in its handling of uncertainty as the high-level, localisation-specific techniques described in section 2.1. The common link will turn out to be the *EM* framework for coming to an understanding simultaneously of the world's dynamics and of its hidden state, which, it has been suggested,[67] underlies the most satisfactory methods in both localisation and neural net "system identification". This important concept will be treated in detail in section 3.3, and will appear in section 4.2.2.1 at the heart of the Samovar model.

[62] Whitehead & Ballard, *Active Perception*; Whitehead & Ballard, *Learning to Perceive and Act*

[63] Basye *et al.*, *A Decision-Theoretic Approach*; Cassandra *et al.*, *Acting under Uncertainty*; see also Fusiello & Caprile, *Synthesis of indoor maps* and Kristensen, *Sensor planning*

[64] Kaelbling *et al.*, *Planning and Acting*

[65] see also Chrisman, *Reinforcement learning*

[66] Kröse & Bunschoten, *Probabilistic localization*

[67] section 2.2.2.2; section 2.1.2.3

Once the Samovar model has been demonstrated, section 6.2 will discuss how the theory of robotic decision-making surveyed in section 2.3 might be applied to using it as a basis for behaviour.

Chapter 3

Background

The character of the experiences of an autonomous robot places strenuous demands on the model which is asked to predict them. Partly this is a matter of the sheer complexity of any realistic environment. Another, related, problem is the environment's unpredictability, at least on the basis of the information available to the robot, which can (arguably) only be treated successfully within the framework of Bayesian inference. This chapter expands on the problems the model has to overcome and provides an introduction to the Bayesian theory of how to deal with them.

3.1. Requirements for the model

3.1.1. Why the robot's world is complex

Robot environments are complicated in at least three ways at once.

3.1.1.1. Arbitrariness

A robot starting with no specific knowledge about its world is going to discover many facts about it which must be treated as unconnected. For instance, the layout of the space in which a mobile robot operates will have a decisive influence on the robot's experience as it moves around. Although these effects will be deterministic, there is no way they can be modelled as other than brute facts; generalising from them will tend not to work. This requirement favours the choice of a model which explicitly makes room for arbitrary phenomena, rather than one (such as a multi-layer perceptron) which treats all learning as a problem in interpolation.

3.1.1.2. Heterogeneous regularity

Most environments will also exhibit strong quantitative relationships between some of the variables in play, from which it is appropriate to generalise by interpolation. The model should be able to capture these dependencies. But they will not remain the same in all situations; they may change or break down. For instance, the way the readings from a mobile robot's range sensor changes over time might often vary linearly with its speed, but the coefficient might change when the robot reaches a bend in the surface of a nearby object; or something entirely different might happen if it moves into a very cluttered area. So again, the model should not be predisposed to shoehorn all its experiences into a single continuously varying mapping.

3.1.1.3. Partial observability

The model will generally not be able to make good predictions based purely on information from its current sensor readings (and motor commands). Instead, it will have to maintain an estimate of some quantities representing that part of the state of the world which is causally efficient with respect to its future experiences and which it cannot observe directly, based on past data as well as present observations. In the case of a mobile robot this might involve recognising a feature marking an area it has previously explored, or inferring the relative orientation of a nearby surface from successive measurements of its distance.

3.1.2. Why the robot's world is unpredictable

Uncertainty is, if anything, an even more troublesome aspect of the environment's general capriciousness than complexity. There are several reasons why the information available to the model in a certain situation might not warrant its offering a precise prediction of what will happen next; intuitively, one feels that the prediction should carry some health warning or be vague, and that the robot should therefore choose a cautious action. The model has to draw together all the sources of uncertainty outlined below—incommensurable as they may seem—so that they have a quantitatively correct influence on its prediction and on the robot's behaviour.

3.1.2.1. Noise and nondeterminism

When unpredictability is mentioned, we most commonly think of randomness. A process is random if it is impossible in principle to make exact predictions about its evolution, based on the available evidence: no observer, however closely they approached the ideal of perfect rationality, could foresee precisely how it was going to behave. Clearly the model should be able to recognise randomness in the environment and adjust its predictions appropriately.

In fact, it should be able to distinguish between two kinds of randomness, which could be called "noise" and "nondeterminism". The former term denotes the unpredictable perturbation of some quantity away from its "true" value, larger disturbances being less likely than smaller ones—the effect, for example, which electrical noise might have on the readings from a robot's sensor. The latter refers to situations in which the system can develop in two or more qualitatively different ways, each yielding a distinct and numerically separate outcome, as might be the case if a robot's sensor were faulty and sometimes read zero instead of a meaningful number. Attempts to model nondeterminism as an instance of noise will necessarily result in misleading predictions: in the case of the broken sensor, the conclusions will be that a low value intermediate between zero and the "working" range is the most likely reading, even though it may in fact be impossible.

3.1.2.2. Lack of experience

The fact that the model is attempting to generalise from a finite amount of experience in the environment should also be grounds for a measure of scepticism as to the reliability of its predictions. If the robot finds itself in a situation whose consequences are to a greater or lesser extent random, and which it has only had a few opportunities to observe in the past, then it cannot be sure that it has seen all the possible outcomes. Of course, if it gets into some position which it has never previously experienced, its uncertainty will be even more acute. In the most extreme case the whole character of the world might change, so that every situation is effectively a new one: for instance, a sensor which has previously functioned correctly might go wrong. The model should be able to make allowances for the finite nature of its knowledge, incorporating the right degree of confidence or tentativeness into its predictions.

3.1.2.3. Ignorance of the state of the world

If the model's predictions are based on estimates of quantities in the world which are not directly observable (section 3.1.1.3), then it needs to make allowance for how errors in those estimates would affect the accuracy of its predictions.

3.1.2.4. Model failure

Even if the environment is entirely deterministic, and the model has had plenty of experience and knows exactly what is current state of the world is, its predictions may still be unreliable if the assumptions built into the model by its designers are untrue. In this case the model's performance should degrade gracefully, and it should perhaps also signal that a mistake has perhaps been made.

3.1.3. Pragmatic considerations

To the wish lists above, which lay out the expressive power desired of the model, must be added the further requirement that it should be implementable in a form which uses as few computational resources as possible. An autonomous robot is continually making decisions[1] about what to do next, so a system which takes a long time over learning, or—worse—one from which it takes a long time to extract a recommended action is going to be inconvenient.

3.2. Bayesian modelling

Of the requirements for the robot's environment model discussed in section 3.1, it is the ones relating to the handling of uncertainty which appear to be the most difficult to meet in principle. How can uncertainty be represented? How can the various kinds of uncertainty, apparently so essentially different from each other, be brought together in a unified framework? How can the effect which uncertainty should have on the robot's choice of action—intuitively, "cautiousness"—be quantified? In this section a brief introduction is given to a reasoning system which is increasingly popular across a range of disciplines and provides answers to all these questions: Bayesian inference[2].

3.2.1. The idea behind Bayesian inference

The long-established science of logic offers an account of what absolutely certain consequences a rational being could draw from a given set of absolutely certain facts.

[1] explicit or implicit

[2] Lee, *Bayesian Statistics*; Box & Tiao, *Bayesian Inference in Statistical Analysis*; Jaynes, *Probability Theory*, Bishop, *Neural Networks*

Bayesian inference is an extension of logic to more realistic cases in which the reasoner's beliefs, and hence the conclusions she can draw, are not necessarily certain. It works by replacing the binary truth value of Boolean logic, which, when predicated of a proposition, indicates whether a subject ought to assent to it or not, with a real value which denotes instead the *degree of certainty* with which she should credit it.

3.2.1.1. The laws of rationality

Obviously this move only makes sense if the semantics of the new continuous scale of truth values, and the axioms by which they are combined, can be given a rigorous and justifiable definition. Modern Bayes theory does this by demonstrating, from remarkably simple and uncontroversial desiderata, that there is only one possible set of axioms for the system, and that they determine the truth values up to isomorphism[3]. The axioms obtained are identical with the familiar basic rules of probability theory; and as one might expect, the extremal truth values can be fixed by convention so that unity means "definitely true", zero means "definitely untrue", and the axioms of logic drop out as special cases.

Since the foundation of the theory—the proof that the laws of probability are so clearly the only possible ones that they can reasonably be given normative status as *the* axioms of rationality—can sound counterintuitive, it's perhaps worth quoting the premises in full here:[4]

1) Degrees of plausibility are represented by real numbers.
2) If $p(A \mid D) > p(A \mid C)$, then $p(\neg A \mid D) < p(\neg A \mid C)$;
3) ... and if additionally $p(B \mid A, D) = p(B \mid A, C)$, then $p(A, B \mid D) \geq p(A, B \mid C)$.
4) If a conclusion can be reasoned out in more than one way, then every possible way must lead to the same result.
5) The robot always takes into account all of the evidence it has relevant to a question. It does not arbitrarily ignore some of the information, basing its conclusions only on what remains.
6) The robot always represents equivalent states of knowledge by equivalent plausibility assignments. That is, if in two problems the robot's state of knowledge is the same (except perhaps for the labelling of the propositions), then it must assign the same plausibilities in both.

[3] *i.e.* uniquely given the choice of 0 and 1 as extremal values

[4] adapted from Jaynes, *Probability Theory*, chapter 1, equations 1–17 and 1–20 to 1–23 (eliding in the interests of brevity the distinction between "probabilities" and "plausibilities")

The mathematics involved in getting from these apparently innocuous inequalities and principles to the product rule

$$p(A, B \mid C) = p(A \mid B, C)\, p(B \mid C) = p(B \mid A, C)\, p(A \mid C) \tag{2}$$

and sum rule

$$p(A \mid B) + p(\neg A \mid B) = 1 \tag{3}$$

is not trivial, but it is rigorous.[5] The rest of the apparatus of probability theory follows from those results, and its applicability to the standard population-sampling problems is established using the symmetry principle (item 6 above).[6]

3.2.1.2. Conditionality and subjectivity

The most important thing to understand about Bayesian theory is the emphasis it places on conditional probability. In fact, almost all (non-tautological) Bayesian probabilities are conditional, just as all (non-tautological) logical truths are conditional on a set of hypotheses. Unlike in some other theories which consider probabilities to be inherent properties of physical systems, there is no notion of an "objectively true" answer to the question "What is the probability/degree of certainty that the random variable X will take the value x?". Instead, Bayes offers an answer which is "uniquely rational", *given* one's state of knowledge. The talk is in principle always of $p(X = x \mid \cdots)$, never of $p(X = x)$. In this sense Bayesian inference is openly subjective. Probabilities are all in the mind, and two reasoners whose beliefs differ can come to different conclusions when presented with the same evidence[7].

This does not mean that in accepting Bayesian theory, one is adopting a philosophical stance which relativises truth. One can talk in the third person about the true state of affairs in the world, reasoners' differing experiences of the world, the subjective conclusions they each come to on the basis of that experience, and the objective accuracy of those conclusions. The subjectivity of Bayesian inference is akin to that of logical inference.

3.2.1.3. The controversy surrounding Bayesian methods

Until the 20th century, most writers on probability theory (from the Bernoullis to Maxwell) conceived of the field very much in terms of a search for laws of rationality, with the normative status of logical axioms—what would now be called a Bayesian

[5] Jaynes, *Probability Theory*, chapter 2; Cox, *Probability*

[6] Jaynes, *Probability Theory*, chapter 3

[7] For a striking example, see Jaynes, *Probability Theory*, chapter 3, p. 507.

programme.[8] But from the early 1900s on, with the foundation of modern statistics in the shape of the "frequentist" and related theories by Neyman, Pearson, Fisher, and others, the Bayesian account came to be considered extremely controversial. In part this was a consequence of a natural desire to avoid bringing "metaphysical" considerations about the nature of rationality into the solution of the very down-to-earth problems which drove the development of the field. Other criticisms focused on the element of subjectivity in Bayesian theory noted in section 3.2.1.2: surely a system which claimed that two rational people could draw different conclusions from the same statistical data was at best suboptimal, at worst absurd. And of course before Cox, *Probability* the foundations of Bayes theory did not seem as solid as they do now, while on the other hand the "adhockeries" and paradoxes of classical statistics pointed out by the Bayesians were not yet widely recognised.

In recent years the Bayesian approach to statistics has once again become fashionable, its popularity resting mainly on the very practical consideration that the answers obtained from frequentist methods in many problems of interest to industry seem too conservative, and only obliquely related to the questions asked. But whatever one's position on the admissibility of Bayesian methods in traditional applications of statistics, as a set of principles for programming a machine to reason and learn in an uncertain world, the theory is very compelling; and indeed many successful commercial applications ranging from speech recognition and expert systems to knowledge management, as well as recent advances in robotics, are or can be seen as essentially Bayesian.

3.2.2. Bayesian inference using models

Bayesian inference is well suited to the kind of application, common in engineering, in which the reasoner has a model of some process in the world, and wishes to use it in conjunction with her measurements of some observable quantities to make inferences or predictions about the values of other quantities.

As a trivial example, she might have a machine which, at each timestep t, picks at random from a bin containing several different kinds of object, and then says as the value r^t of its output R^t which kind it chose. The reasoner's knowledge Θ about the machine is enough for her to be able to tabulate the distribution

$$p(R^t = r^t \mid \Theta) = \text{proportion of } r^t\text{-type objects in bin} \tag{4}$$

Or, if the machine instead has several trays, and the value a^t of its input A^t tells it which to use, she can tabulate the conditional density

$$p(R^t = r^t \mid A^t = a^t, \Theta) = \text{proportion of } r^t\text{-type objects in bin } a^t \tag{5}$$

[8] Jaynes, *Probability Theory, passim*

which says what her degrees of certainty about the different possible outputs should be, if the machine is given a particular input. (As was noted in section 3.2.1.1, this equality between Bayesian probabilities and population proportions follows from considerations of symmetry.) If she knows A^t precisely then she can just read off the implied density of R^t. This distribution is in a sense her "estimate" of R^t, since it includes all the information she has about it—but it will not in general take the form of a single best estimate, or a range around some central value; it could be any legal probability density.

3.2.2.1. *Marginalisation*

If she does not know for sure what A^t is—*i.e.* her belief-density over its possible values is non-zero at more than one point—then she can still estimate R^t by performing what is called a "marginalisation". Representing all the knowledge she has which is relevant to the value of A^t by $\mathcal{A}$, she should compute

$$\begin{aligned} p(R^t = r^t \mid \Theta, \mathcal{A}) &= \int_{a^t} p(R^t = r^t, A^t = a^t \mid \Theta, \mathcal{A}) \quad \text{by sum rule} \\ &= \int_{a^t} p(R^t = r^t \mid A^t = a^t, \Theta)\, p(A^t = a^t \mid \mathcal{A}) \quad \text{by product rule} \end{aligned}$$

—the expectation of (5) over the degree of certainty she attaches to each possible value of A^t. (Note that $\mathcal{A}$ has been dropped when a particular $A^t = a^t$ is asserted on the right hand side of a conditional probability, since a firm hypothesis about the input renders irrelevant previous information about what it might otherwise have been.)

This is not the same as calculating an averaged "single best estimate" value for R^t: it is the whole distribution whose expectation is taken, so much more information is preserved. If the variables are discrete, and the distribution (5) takes the form of a table, the integration is a summation and the marginalisation is essentially a matrix multiplication. (Discussion of the case of continuous distributions is deferred until the big picture has been sketched in.)

3.2.2.2. *Bayes' rule*

Suppose, on the other hand, the modeller can measure the output R^t precisely and is interested in sharpening up her idea of what the input A^t must have been. She wants to know $p(A^t = a^t \mid R^t = r^t, \Theta, \mathcal{A})$, but her model tells her $p(R^t = r^t \mid A^t = a^t, \Theta)$—she needs to reverse the direction of conditionality, and to do that she has to use Bayes' rule[9]:

$$p(A^t = a^t \mid R^t = r^t, \Theta, \mathcal{A}) = \frac{p(R^t = r^t \mid A^t = a^t, \Theta)}{p(R^t = r^t \mid \Theta, \mathcal{A})}\, p(A^t = a^t \mid \mathcal{A})$$

[9] Bayes' rule is a straightforward consequence of the product rule (equation 2).

In this formula $p(A^t = a^t \mid \mathcal{A})$ is called the "prior", $p(R^t = r^t \mid A^t = a^t, \Theta)$ the "likelihood" and $p(R^t = r^t \mid \Theta, \mathcal{A})$ the "normaliser". The resulting density $p(A^t = a^t \mid R^t = r^t, \Theta, \mathcal{A})$ is called the "posterior".

The denominator can also be expanded by marginalising over A^t,

$$\cdots = \frac{p(R^t = r^t \mid A^t = a^t, \Theta)}{\int_{a^t} p(R^t = r^t \mid A^t = a^t, \Theta)\, p(A^t = a^{t\prime} \mid \mathcal{A})} p(A^t = a^t \mid \mathcal{A})$$

whence it can be seen that the posterior is obtained by simply adjusting the prior up for a^ts which make the observed r^t more likely than the other a^ts do on average, and down for those that make it less likely. The coefficient involved (likelihood ÷ normaliser) is sometimes called the "Bayes factor".

3.2.2.3. Cause and effect?

It is often convenient to call A^t the "cause" and R^t the "effect" it has by means of a process described by Θ. But nothing in the theory says that the relationship is really one of physical causation—it's only necessary that the reasoner's beliefs Θ should be such as to make her beliefs about the value of R^t depend on her beliefs about the value of A^t, and that allows for Θs specifying reverse causation (R^t causing A^t) or joint causation, or making no judgment about causation at all. So the terms should be seen as a useful shorthand.

3.2.3. Bayesian model learning

Section 3.2.2 showed how Bayesian methods can be used to predict effects from causes given a model, or conversely to infer causes from effects given a model using Bayes' rule. The Bayesian theory of learning simply involves applying Bayes' rule in a slightly different way to fill in the third side of the triangle and estimate the model from observed effects and causes.

3.2.3.1. Learning as inference

The idea is to treat the model as just another quantity which needs to be estimated, and write down what the estimate should be:

$$p(\Theta = \theta \mid A^t = a^t, R^t = r^t, \mathcal{H}) = \frac{p(R^t = r^t \mid A^t = a^t, \Theta = \theta)}{\int_{\psi} p(R^t = r^t \mid A^t = a^t, \Theta = \psi)\, p(\Theta = \psi \mid \mathcal{H})} p(\Theta = \theta \mid \mathcal{H})$$

Here $\Theta = \theta$ denotes the proposition that the process by which A^t causes R^t is as described by the specification θ; the observations a^t and r^t together comprise an example of the process in action; and $\mathcal{H}$ represents the prior knowledge the reasoner had before the observations came in which was relevant to θ. Bayes' rule says what her posterior belief-density for θ should be: what credence she should now give to each of the possible alternative models.

In practice, of course, she will want to base her estimate of θ on more than one A^t, R^t observation. Suppose she has a set d of readings a^t, r^t taken at different times t, and part of her background knowledge $\mathcal{H}$ is that they are independent. Then—adopting now the usual space-saving convention that the elided "proposition" a^t means $A^t = a^t$—she will conclude that

$$p(\theta \mid d, \mathcal{H}) = \frac{\prod_t p(r^t \mid a^t, \theta)}{\int_\psi \left(\prod_t p(r^t \mid a^t, \psi)\right) p(\psi \mid \mathcal{H})} p(\theta \mid \mathcal{H}) \tag{6}$$

3.2.3.2. Making predictions

How is the learner to use her estimate of θ, based on the observations $d = a^{[0,T)}, r^{[0,T)}$ to predict what the output of the process will be if it is given a new input a^T? If the information provided by d has enabled her to pin θ down precisely, she can just read off $p(r^T \mid a^T, \theta)$. Otherwise, she must *marginalise over the model* (*cf.* section 3.2.2.1), making use of the posterior distribution for θ derived in (6):

$$p(r^T \mid a^T, d, \mathcal{H}) = \int_\theta p(r^T \mid a^T, \theta)\, p(\theta \mid d, \mathcal{H}) \tag{7}$$

What's so impressive about this formula is that it makes quantitative allowance both for the learner's remaining uncertainty about the model (since she has learned it from a finite number of experiences—recall section 3.1.2.2), and for any element of randomness in the models themselves (section 3.1.2.1). If in addition she marginalises over a^T,

$$p(r^T \mid \mathcal{A}, d, \mathcal{H}) = \int_{\theta, a^T} p(r^T \mid a^T, \theta)\, p(a^T \mid \mathcal{A})\, p(\theta \mid d, \mathcal{H})$$

then she can also cope with situations in which she doesn't know the precise value of the model input (section 3.1.2.3).

3.2.4. Practical implementation of the approach

The importance of the Bayesian framework lies not only in the elegant answer it offers to the question of what learning is—*i.e.* a kind of inference—and the ease with which it draws together different kinds of uncertainty, but also in the context it provides for understanding the difficulties which the learner must confront when she comes to apply the theory in practice.

3.2.4.1. Model spaces

Chief among these is the requirement when performing the marginalisation (7) that she must consider *every* possible model θ. There is no way she can really do that—the space of all conceivable models for processes being hopelessly large and complicated—unless her background knowledge $\mathcal{H}$ rules all but a tractable subset of them out as entirely doubtful. This subset is called the model space.

The most convenient situation is when the model space comprises a small finite number of possibilities. Then the posterior distribution (6) is discrete and the marginalisation can be performed by enumeration.

If the space of possible models is large or continuous, but it and the prior distribution over its members ($p(\theta \mid \mathcal{H})$ in (6)) can be represented in a well-behaved parameterised form, then learning and the making of predictions can still be tractable. To take a trivial example, if the learner knows that the process being modelled generates the outputs from some Gaussian distribution (and just ignores the inputs), she can adopt "all Gaussians" as her model space, parameterised by their means and variances.[10] In very simple cases, it turns out that the marginalisation can be performed symbolically. Otherwise the integral involved will not have a closed form solution, and she will have to deploy a suitable numerical integration algorithm. (As the model space and prior are made more complicated, the posterior can become so badly behaved, by the normal standards of numerical analysis, that marginalisation is only possible using the more sophisticated Monte Carlo integration techniques, and tends to take a long time.[11])

3.2.4.2. (Local) maximum likelihood learning

However, the integration is not necessary if the uncertainty in the estimate of the model is negligible (section 3.2.3.2); and if the estimate has been made from a large number of observations, the learner is often able to establish that this is indeed the case. Then the whole posterior distribution is concentrated close to a single, "maximum *a posteriori*" (MAP) model, the full marginalisation is clearly overkill, and it is sufficient to consider the predictive distribution conditioned on the MAP model:

$$\begin{aligned} p(r^T \mid a^T, d, \mathcal{H}) &\approx p(r^T \mid a^T, \theta^*) \\ \text{where} \quad \theta^* &= \operatorname*{argmax}_{\theta} p(\theta \mid d, \mathcal{H}) \end{aligned} \tag{8}$$

In this case it will often also be true that the prior distribution $p(\theta \mid \mathcal{H})$ of the model has little influence on the posterior maximum, so that the MAP model is close to the one

[10] see also section 3.2.5.2

[11] Neal, *Probabilistic inference*

which maximises the "likelihood" term:

$$\begin{aligned}\theta^* &= \operatorname*{argmax}_\theta p(\theta \mid d, \mathcal{H}) \\ &= \operatorname*{argmax}_\theta \frac{p(d \mid \theta)}{p(d \mid \mathcal{H})} p(\theta \mid \mathcal{H}) \quad \text{see (6)} \\ &\approx \operatorname*{argmax}_\theta p(d \mid \theta) \\ &= \operatorname*{argmax}_\theta \prod_t p(r^t \mid a^t, \theta)\end{aligned}$$

Then the likelihood is said to "dominate" the prior—the latter will be approximately constant across the narrow range in which the former is not effectively zero. This is the Bayesian justification for the method of "maximum likelihood estimation". Because the maximisation involves finding a root of the derivative of the likelihood with respect to the parameter, it is convenient to look at the log of the likelihood

$$\cdots = \operatorname*{argmax}_\theta \sum_t \log p(r^t \mid a^t, \theta) \quad \text{since log is convex} \tag{9}$$

If the model distribution $p(r^t \mid a^t, \theta)$ is a member of the exponential family (all the obvious ones are), then its log will tend to have a friendly derivative which can then be summed over the whole training set to produce the overall value.

However, the maximisation may be very difficult to perform perfectly if the space is badly behaved, having many modes and making it impossible to find the globally best model using gradient-based search algorithms. So reasoning the Bayesianly correct way will often turn out to be impractical, or at least inconvenient. Then all the learner can do is take the *single most probable model* she has been able to find, and adopt it as a hypothesis. Given the constraints on her ability to entertain and draw the consequences from the much more complex beliefs which she ought ideally to be holding, this is the most rational way for her to proceed.

What if there is some residual uncertainty in the model—the observations were not sufficient to narrow the posterior down to a tiny part of the model space—but the learner does not wish to attempt a full marginalisation over the posterior? The distribution will take the form not of sharp peaks, dominated by that of the globally MAP model, but of finitely wide "islands" of probability mass. Sometimes she may be able to obtain a good local approximation to the posterior, and therefore of the shape of the island around each locally MAP model; and that may enable her to proceed by finding the best (point) model she can and performing a kind of local marginalisation around it.[12]

[12] MacKay, *A Practical Bayesian Framework*

3.2.4.3. Approximation using simple models

The learner still has to confront a problem deferred from section 3.2.4.1: what if she is not sure *a priori* that the process can be represented correctly by a model drawn from some known parameterised space—if very little is known about the the true model, except that it is likely to be rather complicated, and the one formalism for powerful models with which the analysis goes through cleanly, namely Gaussian process, is inappropriate? This is the situation that must be faced when designing a learning engine for an autonomous robot (section 3.1.1); and the only way forward is to adopt a space of models which are known to be too simple, but allow them to "paper over the cracks" by describing some phenomena in the environment as unpredictable when in fact a more powerful model would be able to capture their behaviour deterministically.

The reason this is an intuitively reasonable thing to do is that it appears to result in a "conservative" approximation, in the sense that it works by making the predictions more vague where necessary, rather than letting them be precise but inaccurate. It's possible that this feeling could be tightened up by some kind of variational argument akin to the maximum entropy principle,[13] but in any case, the evident efficacy of Bayesian methods which rely on it, such as the well-known Kalman filter and hidden Markov model which will be introduced shortly, suggests that in practice it proves to be correct: the performance of an over-simple model will degrade gracefully, as long as it is allowed to "learn" an element of random slop. *Cf.* section 3.1.2.4.

3.2.4.4. Rationally suboptimal reasoning

Overall it seems that learning, if it is to be perfectly rational, must throw a huge and frequently unsustainable weight back on the learner's knowledge of which models should be taken seriously as candidates, and how plausible each is *a priori*, as well as on her ability to work through uncooperative integrations or at least optimisations.

That Bayesian theory leads to this conclusion should not be counted against it as a failing. After all, it is a commonplace that no learner can absolutely guarantee the correctness of a specific prediction she makes on the basis of a generalisation from a finite number of previous examples. Inductive reasoning is acknowledged to be very difficult. Arguably, the Bayesian formulation succeeds both in expressing what the learner *could*

[13] Jaynes, *Probability Theory*, chapter 11

warrantably say—*i.e.* it actually solves the "problem of induction", in principle[14]—and also in explaining why what she *can* say is inevitably going to fall short of that ideal.

It also provides a context and motivation for the *ad hoc* way in which difficult tasks in machine learning are almost always tackled, namely by performing a fallible optimisation of a single model over a restrictive model space. The theory provides reassurance that the point hypothesis the machine ends up with is the best it can do given the real-world constraints on its algorithms, while also exposing the reasons why it is in principal suboptimal.[15]

3.2.4.5. Assessing model truth

Furthermore, it suggests a way of ameliorating the worst case consequences of the constraints on the learner's hypothesis space, by introducing a fallback hypothesis to the effect that something has gone wrong: the model optimisation did not work, or the training examples were unrepresentative, or the world has changed, or whatever. If these failure modes are real possibilities, then Bayes says the learner must keep them "in the back of her mind", and if at any point they seem likely to have occurred, she should respond by tempering the predictions from her model with the much vaguer prediction she must make if she concludes the model is useless—*e.g.* she may just predict nothing more specific than that the process outputs will lie in their legal range. This monitoring mechanism involves computing the posterior probability that the model is working versus the probability that a catastrophic failure has happened, from the respective prior probabilities and likelihoods, using Bayes' rule:

$$\begin{aligned} p(\theta^* \,|\, r^{[T,T+t)}, \mathcal{H}) &= \frac{p(r^{[T,T+t)} \,|\, \theta^*)\, p(\theta^* \,|\, \mathcal{H})}{p(r^{[T,T+t)} \,|\, \theta^*)\, p(\theta^! \,|\, \mathcal{H}) + p(r^{[T,T+t)} \,|\, \theta^!)\, p(\theta^* \,|\, \mathcal{H})} \\ \text{where} \quad \theta^* &= \operatorname*{argmax}_{\theta} p(\theta \,|\, r^{[0,T)}, \mathcal{H}) \\ \text{and} \quad \theta^! &= \text{fallback model} \end{aligned}$$

... and then adopting as her predictive distribution a linear combination ("mixture") of the predictive distributions implied by each possibility, weighted by her assessment that each holds:

$$p(r^{T+t} \,|\, r^{[0,T+t)}, \mathcal{H}) \approx p(r^{T+t} \,|\, \theta^*)\, p(\theta^* \,|\, r^{[T,T+t)}, \mathcal{H}) + p(r^{T+t} \,|\, \theta')\, \left(1 - p(\theta^* \,|\, r^{[T,T+t)}, \mathcal{H})\right)$$

[14] Bayesian theory even provides a quantitative justification for the principle of inductive reasoning known as Occam's Razor: that simpler models should be preferred over complex ones where both account equally well for the evidence This is ultimately because a class of complex models is necessarily going to be bigger than a class of simple ones, so that the prior probability of each member must be lower See for instance MacKay, *Bayesian Interpolation.*

[15] *Cf.* Neal & Hinton, *A New View of the EM Algorithm*

The model's predictions should be believed only to the extent that they have proved empirically to be better than those made from a position of ignorance. *Cf.* section 3.1.2.4.

3.2.5. Simple examples of models

Up to this point in the discussion, the question of the semantics/algebraic forms which Bayesian models can in practice take has been avoided or finessed. This section introduces the simple models which serve as building blocks for the more sophisticated constructions introduced later on.

3.2.5.1. Multinomials

It was noted in section 3.2.2.1 that inference using discrete models defined by a table or matrix is in principle not difficult (though the computation required can become prohibitively long-winded if the variables concerned can take a large number of distinct values). But how are these models to be learned in the first place?

Consider, as the simplest possible case, a single-bin selecting machine as in (4) but for which the modeller does not know the contents of the bin. Her model of the process can be parameterised by the proportions of the different kinds of object, say ω_i for type (and therefore output) i. Supposing that she has plenty of example outputs r^t available, so that the likelihood dominates the prior, (9) says that she should find the point ω in the parameter space to maximise the likelihood with which the process described by the model would produce the observed outputs:

$$\begin{aligned}\omega^* &= \operatorname*{argmax}_{\omega} \sum_t \log p(r^t \mid \omega) \\ &= \operatorname*{argmax}_{\omega} \sum_t \sum_i \delta_{i,r^t} \log \omega_i\end{aligned} \tag{10}$$

The derivative of the log likelihood is of course extremely simple, but its root must be found subject to the constraint that the proportions/probabilities must sum to unity. This can conveniently be achieved using a Lagrange multiplier.[16] The appropriate Lagrangian function is

$$L(\omega, \lambda) = \sum_{t,i} \delta_{i,r^t} \log \omega_i + \lambda \left(\sum_i \omega_i - 1 \right) \tag{11}$$

Setting its derivatives with respect to λ and the ω_is to zero yields the equations

$$\begin{aligned}\sum_i \omega_i &= 1 \\ \forall i. \sum_t \frac{\delta_{i,r^t}}{\omega_i} &= -\lambda\end{aligned} \tag{12}$$

[16] Bishop, *Neural Networks*, appendix C

whence

$$\omega_i^* = \frac{\sum_t \delta_{i,r^t}}{-\lambda}$$

and, solving for λ by substituting that into the constraint equation (12),

$$\begin{aligned}\sum_i \sum_t \frac{\delta_{i,r^t}}{-\lambda} &= 1 \\ \lambda &= -\sum_t \sum_i \delta_{i,r^t} \\ &= -\sum_t 1\end{aligned}$$

the estimation rule is

$$\omega_i^* = \frac{\sum_t \delta_{i,r^t}}{\sum_t 1} \tag{13}$$

Of course this is just the proportion of the output i observed in the training data (as it clearly has to be); but the method applies to less obvious cases considered later on.

If the process being modelled responds to an input a^t, like the multi-bin selecting machine of (5), then the corresponding estimation rule is

$$\begin{aligned}\omega_{i,j}^* &= \frac{\sum_t \delta_{i,r^t}\delta_{j,a^t}}{\sum_t 1} \\ \text{where} \quad \omega_{ij} &= p(R = i \mid A = j, \theta)\end{aligned}$$

This model is the simplest way of parameterising the relationship between two discrete variables; indeed according to the principle of maximum entropy (discussed briefly section 3.2.6.1) it is the model which should be adopted if nothing is known *a priori* about how the process actually works—though this doesn't absolve the learner from her (ideal) rational responsibility to consider more specialised models if the data displays some clear pattern.

Instead of pursuing the maximum likelihood policy adopted here, it is in fact possible to perform a MAP optimisation as in (8) if a prior $p(\theta \mid \mathcal{H})$ of a certain kind is adopted. This distribution (the Dirichlet) is of the same functional form with respect to the parameter as that of the likelihood, which means that the algebra above goes through undisturbed; it also yields a closed form for the full marginalisation over the model as in (7). Such a prior is called "conjugate" to its model.[17]

3.2.5.2. Gaussians

As the simplest illustration of a continuous model, suppose the learner knows that the process under consideration generates its outputs in such a way that she can represent her

[17] Lee, *Bayesian Statistics*, p. 59

expectations about their values as a Gaussian distribution of unknown mean and variance—or, more likely, she has very limited computing resources and wishes to make the best inferences she can while constraining her model to lie in that space[18], which is, as will now be shown, very tractable. Writing the model parameter as $\theta = \mu, \beta$, where μ is the unknown mean and β the precision (inverse (co)variance matrix) of the Gaussian, and noting that the input is ignored, the probability of each output r^t is

$$p(r^t \,|\, \theta) = p(r^t \,|\, \mu, \beta) = \left|\frac{\beta}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^t - \mu\right)' \beta \left(r^t - \mu\right)\right)$$

and the likelihood of the whole training set d is

$$p(d \,|\, \mu, \beta) = \prod_t p(r^t \,|\, \mu, \beta)$$

This means that the maximum likelihood approximation to the MAP parameter (section 3.2.4.2) is given by

$$\theta^* = \mu^*, \beta^* = \operatorname*{argmax}_{\mu,\beta} \sum_t \log p(r^t \,|\, \mu, \beta) \tag{14}$$

The terms in the sum have nice derivatives:

$$\log p(r^t \,|\, \mu, \beta) = \frac{1}{2}\log\left|\frac{\beta}{2\pi}\right| - \frac{1}{2}(r^t - \mu)'\beta(r^t - \mu)$$

$$\frac{\partial}{\partial\mu}\log p(r^t \,|\, \mu, \beta) = \beta(\mu - r^t) \tag{15}$$

$$\frac{\partial}{\partial\beta}\log p(r^t \,|\, \mu, \beta) = \frac{1}{2}\beta^{-1} - \frac{1}{2}(r^t - \mu)(r^t - \mu)' \tag{16}$$

so the optimisation (setting those to zero) reduces to

$$\mu^* = \frac{\sum_t r^t}{\sum_t 1} \tag{17}$$

$$\beta^* = \left(\frac{\sum_t (r^t - \mu^*)(r^t - \mu^*)'}{\sum_t 1}\right)^{-1} \tag{18}$$

As in the case of the discrete distribution of section 3.2.5.1, one can define a conjugate prior (here the normal/χ^2 distribution, or normal/Wishart in the multivariate case) which makes it easy to find the MAP parameter rather than the maximum likelihood one, or even carry out a full marginalisation.

It is easy to generalise (17) and (18) to model a process whose output is known to be sampled from one of several available Gaussians, the choice being determined by a discrete

[18] In fact there are often good reasons for modelling an unknown continuous distribution as a Gaussian: it is the vaguest distribution with a given mean and variance, in the "maximum entropy" sense of vagueness (Jaynes, *Probability Theory*, chapter 11).

input analogously with (5). The overall log likelihood in terms of the parameters μ_i and β_i of all the Gaussians i is

$$\log p(d \mid \theta) = \sum_t \sum_i \delta_{i,a^t} \log p(r^t \mid \mu_i, \beta_i) \tag{19}$$

and the estimation rules take the form of selective averages:

$$\mu_i^* = \frac{\sum_t \delta_{i,a^t} r^t}{\sum_t \delta_{i,a^t}} \tag{20}$$

$$\beta_i^* = \left(\frac{\sum_t \delta_{i,a^t} (r^t - \mu^*)(r^t - \mu^*)'}{\sum_t \delta_{i,a^t}} \right)^{-1} \tag{21}$$

This is the simplest (and the maximum entropy) model of the relationship between a discrete input and a continuous output.

3.2.5.3. Confidence regions for continuous inputs

Recall from section 3.2.4.5 that the learner can use the empirically observed performance of the model to decide (quantitatively) how confident that she is that data she is applying it to are similar in character to those from which she learned it. A finer-grained question of model trust comes up if the learner is not sure that the training set provided examples which covered every part of the input space, so that future inputs different from those on which the model was trained might provoke unexpected behaviour in the process under consideration.

Suppose the process under consideration takes a continuous input and produces a continuous output. The learner notes that it appears to be generating its outputs from a single Gaussian, as in section 3.2.5.2; but also that all the inputs she has been able to observe have fallen in roughly the same area. One way she can account for her worry that the process might respond to a different kind of input by producing a different kind of output would be to make her model treat the input and output together as being generated by a larger process, including whatever agency is responsible for providing the former as well as the machine (or whatever) which maps them to the latter. Then she could express the possibility that the input-output pairs, while mostly following the pattern manifested in the training data, will sometimes do something completely different. What she needs to do is model the distribution of the inputs seen in the training data as well as that of the outputs. At the simplest, she could treat the training outputs as being generated from a Gaussian $N(\mu, \beta)$, as in section 3.2.5.2, and the inputs symmetrically as being generated from a Gaussian $N(\nu, \gamma)$. From timestep T (the end of the training data) onwards, there are always two scenarios, represented by a variable Q^t:

$$A^t, R^t = \begin{cases} N(\nu,\gamma), N(\mu,\beta) & \text{if } t < T \text{ or } Q^t = 1 \\ \text{something random} & \text{if } t \geq T \text{ and } Q^t = 0 \end{cases}$$

Obviously the estimation rules for this model would just be two copies of (17) and (18). To make a prediction of what the output corresponding to a given input a^t will be, the learner need only see how well a^t fits with each of the two possible input-generating processes in play. The result is a weighted sum ("mixture") of Gaussians:

$$\begin{aligned} p(r^t \mid \theta, \mathcal{H}, a^t) &= p(r^t, Q^t = 0 \mid \theta, a^t) + p(r^t, Q^t = 1 \mid \theta, a^t) \\ &= p(Q^t = 0 \mid a^t, \nu, \gamma, \mathcal{H})\, p(r^t \mid Q^t = 0, \mu, \beta) + \\ &\quad\; p(Q^t = 1 \mid a^t, \nu, \gamma, \mathcal{H})\, p(r^t \mid Q^t = 1, \mathcal{H}) \end{aligned}$$

$$\text{where} \quad p(Q^t = 1 \mid a^t, \nu, \gamma, \mathcal{H}) = \frac{p(a^t \mid Q^t = 1, \nu, \gamma)\, p(Q^t = 1 \mid \mathcal{H})}{p(a^t \mid Q^t = 1, \nu, \gamma)\, p(Q^t = 1 \mid \mathcal{H}) + p(a^t \mid Q^t = 0, \mathcal{H})\, p(Q^t = 0 \mid \mathcal{H})}$$

$$\text{and} \quad p(Q^t = 0 \mid a^t, \nu, \gamma, \mathcal{H}) = 1 - p(Q^t = 1 \mid a^t, \nu, \gamma, \mathcal{H})$$

If a^t falls far from the centre of the confidence region $N(\nu, \gamma)$, the density $p(a^t \mid Q^t = 1, \nu, \gamma)$ will be very low; the learner will naturally conclude that the unknown input-output process was responsible, and end up with a mixture whose only significant component is the very broad output distribution $p(r^t \mid Q^t = 0)$. If, on the other hand, a^t lies in amongst the inputs previously seen in the training set, the mixture will be heavily biased towards the output distribution learned from the training data. Finally, in marginal cases the mixture will contain both components, the informative output being tempered by an element of uninformative noise. The exact proportions will depend to a degree on the learner's prior assessment $p(Q^t = 0 \mid \mathcal{H})$ of how often the unknown process occurs.

3.2.5.4. Linear/Gaussian regression

Section 3.2.5.3 discussed a model which described the distribution to be expected of the outputs given the training data, and that of the inputs, using the latter to assess the reliability of the former. What if instead the model goes after the joint distribution of the inputs and outputs together, instead of considering them independently? Since r^t and a^t are being treated symmetrically as if they were both outputs, the model can be learned simply by a version of (17) and (18) in which they have been concatenated. The mean of the resulting distribution is the concatenation of the means μ^* and ν^*, while its variance is made up of blocks comprising the variances β^{*-1} and γ^{*-1} along with the observed covariance between the inputs and outputs. Writing the blocks of its precision—which are not in general the inverses of the corresponding blocks in the variance—as

$$\begin{pmatrix} \phi_{RR} & \phi_{RA} \\ \phi_{AR} & \phi_{AA} \end{pmatrix}$$

it turns out after a little algebra that the predictive distribution of the model

$$\begin{aligned} p(r^t \mid a^t, \theta) &= \frac{p(r^t, a^t \mid \theta)}{p(a^t \mid \theta)} \\ &= N\left(r^t \mid \mu - \phi_{RR}^{-1} \phi_{RA} (a^t - \nu), \phi_{RR}\right) \end{aligned}$$

is another Gaussian, with precision ϕ_{RR} and mean $\mu - (\phi_{RR})^{-1}\phi_{RA}(a^t - \nu)$. The mean of the output prediction will vary linearly with the input.

Indeed this Gaussian joint input-output model is clearly equivalent to one in which the inputs are modelled as coming from a Gaussian, and being passed through a linear transformation κ before having zero-mean Gaussian noise added to produce the outputs:

$$A^t = N(\nu, \gamma)$$
$$R^t = \kappa A^t + N(0, \beta)$$

It is convenient to stipulate that every input a^t is supplemented with an element set to unity, thus providing the linear transformation with an intercept (*i.e.* μ) as well as a slope (*i.e.* $-(\phi_{RR})^{-1}\phi_{RA}$) folded up in the one parameter κ. In terms of the parameterisation $\theta = \kappa, \beta$, the predictive distribution is then

$$p(r^t \mid a^t, \theta) = \left|\frac{\beta}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^t - \kappa a^t\right)' \beta \left(r^t - \kappa a^t\right)\right)$$

This is a particular example of a "linear regressive" model (there is in general no need to assume, as is done here, that the output noise is Gaussian, or that the inputs are drawn from a Gaussian). It will be appropriate if the learner knows that the process's outputs do in fact vary smoothly with its inputs, and that the variation is linear—or she is prepared to restrict her model space to a first-order approximation of the trends she believes are present, in exchange for an efficient algorithm.

If she does not consider it necessary to obtain a confidence region (does not care about the distribution of the inputs) , and is thus happy with a conditional model of $p(r^t \mid a^t)$ as opposed to a joint model of $p(r^t, a^t)$, she can avoid some work by learning κ and β directly. The derivatives with respect to those parameters of each term in the log likelihood can be calculated as shown:[19]

$$\begin{aligned}
\log p(r^t \mid a^t, \kappa, \beta) &= \frac{1}{2}\log\left|\frac{\beta}{2\pi}\right| - \frac{1}{2}(r^t - \kappa a^t)'\beta(r^t - \kappa a^t) \\
\frac{\partial}{\partial \kappa_{ij}} \log p(r^t \mid a^t, \kappa, \beta) &= \frac{1}{2}\frac{\partial}{\partial \kappa_{ij}} a^t_k \kappa_{kl}\beta_{lm} r^t_m + \frac{1}{2}\frac{\partial}{\partial \kappa_{ij}} r^t_k \beta_{kl}\kappa_{lm} a^t_m - \frac{1}{2}\frac{\partial}{\partial \kappa_{ij}} a^t_k \kappa_{kl}\beta_{lm}\kappa_{mn} a^t_n \\
&= \frac{1}{2} a^t_i \beta_{jm} r^t_m + \frac{1}{2} r^t_k \beta_{ki} a^t_j - \frac{1}{2}\frac{\partial}{\partial \kappa_{ij}} a^t_k \kappa_{kl}\beta_{lm}\kappa_{mn} a^t_n \\
&= (\beta r^t a^{t\prime} - \beta\kappa a^t a^{t\prime})_{ij} && (22) \\
\frac{\partial}{\partial \beta} \log p(r^t \mid a^t, \kappa, \beta) &= \beta^{-1} - (r^t - \kappa a^t)(r^t - \kappa a^t)' && (23)
\end{aligned}$$

[19] In (22), the usual convention of summation over unbound subscripts in the same additive term is adopted, as the most convenient notation The subscripts $i\ j\ k\ l\ m\ n$ temporarily lose their usual meanings.

and the optimal solution is

$$\kappa^* = \left(\sum_t r^t a^{t'}\right)\left(\sum_t a^t a^{t'}\right)^{-1} \tag{24}$$

$$\beta^* = \left(\frac{\sum_t (r^t r^{t'} - \kappa^* a^t r^{t'})}{\sum_t 1}\right)^{-1} \tag{25}$$

Just like the output-only Gaussian model of section 3.2.5.2, the regressive model can easily be generalised to select one of several available mappings according to one, discrete (part of the) input, before passing another, continuous (part of the) input through that chosen. The estimation rules for each parameter pair κ_i^*, β_i^* are copies of (24) and (25), with the sums weighted by the same mask δ_{i,a^t} as in (20) and (21).

3.2.6. How Bayesian inference can influence behaviour

In section 3.2.1.1, it was explained that the semantics of Bayesian probabilities (levels of certainty) are grounded in the consistency and uniqueness of the axioms according to which they are manipulated. The only points at which they explicitly make contact with concepts expressible independently of the system are at the extremes: the propositions to which the reasoner assigns the probabilities zero and one are those she knows definitely are false and true respectively. But the whole purpose of the theory is to enable her to reason with propositions in which she has an intermediate degree of belief. So where are they to come from and how is she to going to use them? How is the reasoner's inference system embedded in her real life?

3.2.6.1. Breaking into the system

Consider first the question of where Bayesian probabilities have their ultimate roots. There are seven ways in which these numbers can meaningfully be assigned:

- Most straightforwardly, direct **observations** lead the reasoner to assign sharp zero/one probabilities to corresponding propositions.
- Where the reasoner knows the (discrete) outcomes a process might have, but has no information which leads her to expect one more than the others, considerations of **symmetry** (item 6 in section 3.2.1.1) require that she entertain each possibility with the same degree of certainty—with equal probability. For example, as was mentioned in section 3.2.1.1, this principle suffices to establish the connection between Bayesian probabilities and long-run frequencies in population-sampling experiments.

- If she does have further information about the process, she will want to factor it into her distribution over the outcomes while leaving the distribution as vague as possible in other respects. It can be shown that the only definition of "vagueness" which can be consistent with some clearly necessary constraints is the distribution's entropy, and that is the motivation for the principle of **maximum entropy**: set the probabilities so as to make up the distribution which has maximum entropy consistent with the known facts.[20]
- The trickiest probabilities to resolve unambiguously, but at the same time the ones with the least impact on the learner's judgements, are the prior probabilities assigned to the possible models ($p(\theta \mid \mathcal{H})$ in (6)). Before she has any observations, the learner can very often to be said to be in a state of great ignorance with respect to the value of the model parameters (where in the model space the right model is to be found), but it turns out to be unexpectedly difficult to derive a unique distribution which best represents that fact. However, this does not often matter, since the whole point of such a "**noninformative prior**" is that it should step aside and let the data-induced likelihood be the overwhelming influence on the posterior; the minor differences between the different conceivable priors will typically have a negligible effect on her conclusions.[21]
- It is often possible to represent the notion of the general character expected of the model parameters using what is called a "hierarchical" model.[22] The model prior is defined conditionally on a "**hyperprior**" or "regulariser" which determines the probability of (say) large parameters versus small ones, or the overall smoothness of the mapping implemented by the model. The hyperparameter is given its own prior, but to a greater or lesser extent is left to be determined from the observations. This approach can work well even with complicated models.[23]
- Some probabilities are implicitly "zeroed out" by **practical constraints** placed on the mechanisms used by the learner (section 3.2.4.4).
- Finally, intermediate results and the inferential conclusions which are the learner's goal are obtained by combining existing numbers according to the usual **laws of probability**. This is where the posterior distribution (6) of the model comes from.

[20] Jaynes, *Probability Theory*, chapter 11; in fact this principle subsumes the previous one

[21] Box & Tiao, *Bayesian Inference in Statistical Analysis*, section 1.3, priors can, however, become important when it comes to performing Bayesian model selection in the absence of a large supply of data

[22] Lee, *Bayesian Statistics*, p. 223

[23] MacKay, *A Practical Bayesian Framework*

3.2.6.2. Breaking out of the system: decision theory

The corresponding issue at the other end of the framework is how the learner can use her model and its predictions to help her make decisions. The principle of Bayesian decision theory is very simple: she must define a "gain function" g, which says how relatively good the outcome will be of performing each of the actions a^t in its repertoire if the state of the world is y^t. Then the action she should take is

$$a_*^t = \operatorname*{argmax}_{a^t} \int_{y^t} g(a^t, y^t)\, p(y^t \mid \mathcal{K}) \tag{26}$$

where $\mathcal{K}$ represents all her knowledge relevant to y^t. a_*^t is the action which maximises her expected gain, where the expectation is taken over her degree of certainty that the world is in each possible state.

This rule has many powerful properties.[20] Most importantly, it captures perfectly the notion of "cautiousness" in the face of uncertainty which was established as a requirement for the behaviour of an autonomous robot in section 3.1.2. Suppose some action generally gains a small reward for the agent, but in certain circumstances $y_!^t$ it provokes a large penalty. Then she will avoid the action whenever she suspects even slightly that those circumstances might obtain, since the product of the penalty with the small probability $p(y_!^t \mid \mathcal{K})$ will be sufficient to offset the product of the reward with the larger probability $p(\neg y_!^t \mid \mathcal{K})$. If she is completely unsure what the state of the world is—perhaps because she does not trust her model at all—she will choose an action which is reliably safe in all situations.

Note that a_*^t in (26) is not itself an average: the rule does not choose compromise, intermediate actions, but rather actions which balance the consequences across the agent's beliefs about the world.

If the agent has to plan ahead for more than one action into the future, a more general criterion will be appropriate, such as the greatest expected gain she could achieve up to some time horizon. This issue has been studied intensively within the field of reinforcement learning (section 2.3.1), and will be returned to in section 6.2.

[20] Berger, *Statistical Decision Theory*; Jaynes, *Probability Theory*, chapter 13

3.3. EM modelling

3.3.1. The *EM* algorithm

Many models used in the Bayesian learning of dynamical systems, such as the ones introduced in section 3.3.3, are particularly well suited to an optimisation technique called the *EM* or expectation-maximisation algorithm. This is really a meta-algorithm or algorithm schema, which can help find a maximum *a posteriori* model parameter even if some of the quantities on which the distribution depends are unknown—for instance, if they represent state in a dynamical system which is not directly observable.

3.3.1.1. The procedure

Call the known values d, the hidden quantities H, and the model Θ. The goal is to find a θ^* to maximise $p(\theta \,|\, d)$; a direct approach involves marginalisation over h, which is (it is supposed) hard; yet the more straightforward optimisation of $p(\theta, h \,|\, d)$ with respect to h as well as θ does not give the right answer—it just ignores the uncertainty in h. Happily, it turns out that given an estimate of the model, one can reliably generate a better one by maximising the *expected log* of $p(\theta, h \,|\, d)$, where the expectation is taken over the distribution of H implied by the *old* estimate. Writing the old parameter estimate as θ^n and the new one as θ^{n+1},

$$\begin{aligned}\theta^{n+1} &= \operatorname*{argmax}_{\theta} E_h\big[\, \log p(\theta, h \,|\, d) \,\big|\, \theta^n, d\,\big] \\ &= \operatorname*{argmax}_{\theta} \int_h p(h \,|\, \theta^n, d) \log p(\theta, h \,|\, d) \end{aligned} \tag{27}$$

The difficult marginalisation has been reduced to an integration which is often much easier, because it is conditioned on a fixed θ^n and involves the log of $p(\theta, h \,|\, d)$ rather than the distribution itself[21]. This yields an iterative algorithm which, if it converges at all, will always return the (or a locally) most probable θ^*.

Each iteration can notionally be divided into an "E-step", in which the distribution $p(h \,|\, \theta^n, d)$ is computed and a representation for $E_h[\log p(\theta, h \,|\, d)]$ as a function of θ is obtained, and an "M-step", in which this function is maximised. (In simple cases these stages can sometimes be symbolically conflated to a greater or lesser extent.)

[21] Many "exponential family" distributions have logs which are well behaved for this purpose

3.3.1.2. Correctness of the algorithm

To prove that the algorithm returns a (locally) MAP parameter, it is sufficient to show that each iteration finds a parameter which increases $p(\theta \mid d)$ as well as $E_h[\log p(\theta, h \mid d)]$, and that if it converges, the limiting parameter is a stationary point of $p(\theta \mid d)$. Then it follows that the limit is a posterior mode (or, conceivably, a saddle point).[22]

How does $p(\theta \mid d)$ relate to $p(\theta, h \mid d)$, and its expected log which is maximised in the M-step? By the product rule,

$$p(\theta, h \mid d) = p(h \mid \theta, d)\, p(\theta \mid d)$$
$$\text{and} \quad \log p(\theta \mid d) = \log p(\theta, h \mid d) - \log p(h \mid \theta, d)$$

Taking the same expectation over the unknowns as in the E-step (see (27)),

$$\log p(\theta \mid d) = \underbrace{\int_h p(h \mid \theta^n, d) \log p(\theta, h \mid d)}_{U(\theta)} - \underbrace{\int_h p(h \mid \theta^n, d) \log p(h \mid \theta, d)}_{W(\theta)} \tag{28}$$

The M-step always moves θ from θ^n to some θ^{n+1} chosen to increase $U(\theta)$. So $p(\theta \mid d)$ can only fail to go up if $W(\theta)$ is also increased. But that cannot happen, since

$$\begin{aligned} -\left(W(\theta^{n+1}) - W(\theta^n)\right) &= \int_h p(h \mid \theta^n, d) \log p(h \mid \theta^n, d) - \int_h p(h \mid \theta^n, d) \log p(h \mid \theta^{n+1}, d) \\ &= \int_h p(h \mid \theta^n, d) \log \frac{p(h \mid \theta^n, d)}{p(h \mid \theta^{n+1}, d)} \end{aligned}$$

is the Kullback-Leibler divergence[23] of $p(h \mid \theta^{n+1}, d)$ from $p(h \mid \theta^n, d)$, and is therefore nonnegative.

When (if) the algorithm has converged—so that θ^n is arbitrarily close to a fixed point θ^*—the derivative of U at θ^* is clearly zero, since otherwise U could be further maximised. And the derivative of W is given by

$$\frac{\partial}{\partial \theta} W(\theta) = \int_h p(h \mid \theta^*, d) \frac{\frac{\partial}{\partial \theta} p(h \mid \theta, d)}{p(h \mid \theta, d)}$$

so that

$$\begin{aligned} \frac{\partial}{\partial \theta} W(\theta^*) &= \int_h \frac{\partial}{\partial \theta^*} p(h \mid \theta^*, d) \\ &= \frac{\partial}{\partial \theta^*} \int_h p(h \mid \theta^*, d) \\ &= \frac{\partial}{\partial \theta^*} 1 \\ &= 0 \end{aligned}$$

[22] The proof sketch given here is adapted from Lee, *Bayesian Statistics*, p. 252.

[23] Bishop, *Neural Networks*, p. 59

as well. Since

$$\frac{\partial}{\partial\theta}\log p(\theta \mid d) = \frac{\partial}{\partial\theta}U(\theta) + \frac{\partial}{\partial\theta}W(\theta)$$

it follows that θ^* is a stationary point of the posterior, as required.

3.3.1.3. Generalisations of the EM *algorithm*

The *EM* algorithm was introduced in Dempster *et al.*, *Maximum likelihood from incomplete data* (in a maximum likelihood context, with a remark that it applies equally well under a Bayesian interpretation). The authors also point out that the M-step need not optimise θ^{n+1} all the way to a maximum in order to guarantee an increase in the posterior (although the proof in section 3.3.1.2 that the algorithm will converge to a posterior mode does not then go through). Procedures which exploit this fact are called generalised *EM* (*GEM*) algorithms.

Neal and Hinton give an interesting perspective on *EM*, in which the distribution over which the expectation is taken in the E-step is treated as a parameter ϖ—instead of being inferred as $p(h \mid \theta^n, d)$—and the algorithm as a procedure for maximising a quantity called the "variational free energy"

$$F(\theta,\varpi) = \underbrace{\int_h \varpi(h)\log p(h \mid \theta, d)}_{\dot{U}(\theta,\varpi)} - \underbrace{\int_h \varpi(h)\log\varpi(h)}_{\dot{W}(\varpi)}$$

with respect to ϖ as well as the model parameter θ.[24] Maximising F with respect to ϖ at iteration n, keeping θ^n fixed, does actually turn out to mean setting

$$\varpi^n(h) = p(h \mid \theta^n, d)$$

and then it can be seen from (28) that

$$\dot{U}(\theta,\varpi^n) = U(\theta)$$

Thus what the *EM* algorithm does (repeatedly maximise U) is equivalent to alternately maximising F with respect to ϖ and θ. Since additionally

$$\dot{W}(\theta^n,\varpi^n) = W(\theta^n)$$

it is clear that

$$F(\theta^n,\varpi^n) = \log p(\theta^n \mid d)$$

and it is not hard to show that approaching a maximum of F by any means whatsoever will yield a maximum of $\log p(\theta \mid d)$. This result assures the correctness of many variant *EM*-style

[24] Neal & Hinton, *A New View of the EM Algorithm*

algorithms in which only part of the distribution ϖ is updated at each E-step, or several M-steps are performed in between each E-step.

The variational free energy account also motivates algorithms in which the distribution ϖ is constrained to take some computationally convenient form,[25] and so-called "GEM" algorithms in which the M-step consists in increasing, but not necessarily maximising, U.

3.3.1.4. Ensemble learning

Taking this idea further, Hinton & vanCamp, *Keeping neural networks simple* and Waterhouse *et al.*, *Bayesian Methods for Mixtures of Experts* suggest a procedure called "ensemble learning" in which the whole posterior distribution of θ is approximated, not just its MAP value, thereby perhaps making it possible to carry out a full marginalisation (section 3.2.2.1). The strategy is to approximate the intractable joint posterior $p(\theta, h \mid d)$ by a separable distribution

$$\varpi(\theta, h) = \varpi_\theta(\theta)\, \varpi_h(h)$$

and then minimise the Kullback-Leibler divergence of the true posterior from the approximating distribution

$$\varpi^* = \underset{\varpi}{\operatorname{argmin}} \int_{\theta,h} \varpi(\theta, h) \log \frac{\varpi(\theta, h)}{p(\theta, h \mid d)}$$

By exploiting the separability of ϖ, the overall divergence can easily be reduced to the sum of divergences

$$\int_\theta \varpi(\theta) \log \frac{\varpi(\theta)}{E_h[\log p(\theta, h \mid d)]} + \int_h \varpi(h) \log \frac{\varpi(h)}{E_\theta[\log p(\theta, h \mid d)]}$$

where

$$E_\theta[\cdots] \propto \int_\theta \varpi(\theta) \cdots$$
$$E_h[\cdots] \propto \int_h \varpi(h) \cdots$$

This leads to an iterative algorithm in which repeated use of the updates

$$\varpi^{n+1}(\theta) \propto \exp E_h^n[\log p(\theta, h \mid d)]$$
$$\varpi^{n+1}(h) \propto \exp E_\theta^n[\log p(\theta, h \mid d)]$$

minimises the overall divergence error and gives an approximation to the posteriors of θ and h. If $\varpi^n(\theta)$ is constrained to be a (Dirac) delta function $\delta(\theta^n, \theta)$, an EM-style algorithm drops out as a special case.

[25] For instance, it explains the way in which the "k-means" clustering algorithm approximates the *EM* algorithm given in section 3.3.2

Clearly the method generalises to any number of parameters/unknown variables[26], and will be useful as long as the resulting update rules turn out tractable and the separability approximation does not break down too badly.

3.3.2. Mixture models

In section 3.2.5, various scenarios were considered in which the learner believed, or chose to believe, that the process she wished to model produced each of its outputs according to one of several possible methods, the choice being determined by a discrete input. The *EM* algorithm provides a neat solution to the problem of learning a "mixture" model, in which the output method is selected stochastically, either entirely at random or conditionally on a discrete or continuous input. Mixture models are useful for approximating the behaviour of systems whose behaviour exhibits clear, but not smooth (*e.g.* linear) patterns.

3.3.2.1. Unconditional Gaussian mixtures

Consider first an extension of the multi-Gaussian model of (20) and (21) to the case where the choice of output Gaussian is random, or invisible to the learner (or depends on an input which she cannot observe). Writing μ_i and β_i for the mean and precision of the Gaussian "component" i, Q^t for the choice of component used to generate r^t, and ω_i for the probability with which component i is chosen each time, the density of the outputs is given by

$$\begin{aligned} p(r^t \mid \theta) &= \sum_i p(r^t, Q^t = i \mid \mu, \beta, \omega) \\ &= \sum_i p(r^t \mid Q^t = i, \mu_i, \beta_i)\, p(Q^t = i \mid \omega) \\ &= \sum_i \omega_i \left|\frac{\beta_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^t - \mu_i\right)' \beta_i \left(r^t - \mu_i\right)\right) \end{aligned} \tag{29}$$

(As usual θ denotes the overall model parameter, in this case comprising μ, β, ω.) The log likelihood of the training set d

$$\begin{aligned} \log p(d \mid \theta) &= \log \prod_t p(r^t \mid a^t, \theta) \\ &= \log \prod_t \sum_i \omega_i \left|\frac{\beta_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^t - \mu_i\right)' \beta_i \left(r^t - \mu_i\right)\right) \end{aligned}$$

[26] Waterhouse *et al.*, *Bayesian Methods for Mixtures of Experts* uses it to infer the values of Bayesian hyperparameters, as well as the actual model parameters

is clearly differentiable, but the $\sum_i \cdots$ inside the product prevents the quadraticness of the Gaussian components' log likelihoods from bubbling to the top (*cf.* (14)), making it necessary to use an iterative algorithm for the optimisation of μ, β.

Of course any general purpose nonlinear maximiser will do the job, but there is also an elegant *EM* alternative, in which the set Q of the process's choices of Gaussian components over all the timesteps t is treated as an unknown variable (H in section 3.3.1.1), and the expected log posterior is repeatedly maximised with respect to the parameter $\theta = \mu, \beta, \omega$. In fact, if the training set d is big enough that the likelihood dominates the prior (section 3.2.4.2), then it is sufficient to work with the expected log likelihood; with this adjustment, the *EM* reestimation rule (27) becomes

$$\theta^{n+1} = \operatorname*{argmax}_{\theta} \sum_{q} p(q \mid \theta^n, d) \log p(d, q \mid \theta)$$

Note that the $\sum_i \cdots$ has moved outside the log to become the $\sum_q \cdots$. As one might expect, the loops over the training set inside the log likelihood can now be distributed through the *EM* expectation:

$$\begin{aligned}
\sum_{q} p(q \mid \theta^n, d) \log p(d, q \mid \theta) &= \sum_{q^0 \cdots q^{T-1}} \left(\prod_t p(q^t \mid \theta^n, r^t) \right) \left(\sum_t \log p(r^t, q^t \mid \theta) \right) \\
&= \sum_{q^0 \cdots q^{T-1}} \sum_t \log p(r^t, q^t \mid \theta) \prod_\tau p(q^\tau \mid \theta^n, r^\tau) \\
&= \sum_t \sum_{q^t} \log p(r^t, q^t \mid \theta)\, p(q^t \mid \theta^n, r^t) \sum_{q^0 \cdots \overset{\setminus}{\cdots} q^{T-1}} \prod_{\tau \neq t} p(q^\tau \mid \theta^n, r^\tau) \\
&= \sum_{t,i} \log p(r^t, Q^t = i \mid \theta)\, p(Q^t = i \mid \theta^n, r^t) \\
&= \sum_{t,i} p(Q^t = i \mid \theta^n, r^t) \log \left(p(r^t \mid Q^t = i, \theta)\, p(Q^t = i \mid \theta) \right) \\
&= \sum_{t,i} p(Q^t = i \mid \theta^n, r^t) \left(\log p(r^t \mid Q^t = i, \mu_i, \beta_i) + \log \omega_i \right) \qquad (30)
\end{aligned}$$

By analogy with the way (19) leads to (20) and (21), it should be clear that the reestimation formulas for for μ^{n+1} and β^{n+1} are just weighted averages:

$$\mu_i^{n+1} = \frac{\sum_t p(Q^t = i \mid \theta^n, r^t)\, r^t}{\sum_t p(Q^t = i \mid \theta^n, r^t)} \qquad (31)$$

$$\beta_i^{n+1} = \left(\frac{\sum_t p(Q^t = i \mid \theta^n, r^t)\, (r^t - \mu^*)(r^t - \mu^*)'}{\sum_t p(Q^t = i \mid \theta^n, r^t)} \right)^{-1} \qquad (32)$$

where the coefficients are given by

$$\begin{aligned}
p(Q^t = i \mid \theta^n, r^t) &= \frac{p(r^t \mid Q^t = i, \theta^n)}{p(r^t \mid \theta^n)} p(Q^t = i \mid \theta^n) \\
&\propto \omega_i^n \left| \frac{\beta_i^n}{2\pi} \right|^{\frac{1}{2}} \exp\left(-\frac{1}{2} \left(r^t - \mu_i^n\right)' \beta_i^n \left(r^t - \mu_i^n\right) \right)
\end{aligned}$$

(normalised so as to sum to unity). Each output r^t contributes to the new means and variances of each component i in proportion to the old estimate of how probable it was that i was responsible for generating r^t. Again, by analogy with (10) and (13), the reestimation rule is easily seen to be

$$\omega_i^{n+1} = \frac{\sum_t p(Q^t = i \mid \theta^n, r^t)}{\sum_t 1} \tag{33}$$

—the average of the posterior probabilities that i was responsible for each output.

Gaussian mixture models have been employed to good effect in "classification" or "clustering" applications[27], in which ellipsoidal clusters are discovered in the set of points r^t, and each point is classified according to the clusters which it is believed may have generated it. The well-established k-means algorithm can be seen as an approximation to this method (and can be justified by the arguments of section 3.3.1.3[28]).

3.3.2.2. Gaussian mixture confidence regions

It's also possible to use a mixture in just the same way on the other side to map out a confidence region for the model which is more flexible than that obtained in section 3.2.5.3. The procedure is simply to model the distribution of process inputs as a mixture of Gaussian (or whatever) components i, and then add another one, say component 0, to the mixture which merely says the input will lie somewhere in its legal range.

Furthermore, the learner can adjust the value of the frequency ω_0 with which she expects the model to break down in the light of its performance by using the *EM* update rule (33) with the other elements of ω clamped at their learned values. (Of course if she has the computing time to spare, she can keep on tweaking all her other model parameters as well.)

3.3.2.3. Input-output Gaussian mixtures

If the mixing is done over both the input and the output at the same time, then a very powerful model begins to emerge. Suppose the process is taken to be deciding at random between a number of rules i according to probabilities ω_i, each of which generates a^t and r^t from its own Gaussians $N(\nu_i, \gamma_i)$ and $N(\mu_i, \beta_i)$. Then the joint distribution is

$$p(r^t, a^t \mid \theta) = \sum_i \omega_i p(a^t \mid Q^t = i, \nu_i, \gamma_i)\, p(r^t \mid Q^t = i, \mu_i, \beta_i)$$

[27] The "AutoClass" system (Cheeseman *et al.*, *Autoclass*) is particularly interesting because it tries to take a fairly complete Bayesian approach to the problem It uses a rough but effective approximation to help find the best number of mixture components to use, as well as employing the *EM* algorithm given here to decide where each should be placed.

[28] Neal & Hinton, *A New View of the EM Algorithm*

The *EM* update rules will obviously take the form of two copies of (31) and (32), one for each pair of Gaussian parameters, plus the usual (33). The predictive distribution will be the Gaussian mixture

$$p(r^t \mid \theta, a^t) = \sum_i p(Q^t = i \mid \nu, \gamma, a^t)\, p(r^t \mid Q^t = i, \mu_i, \beta_i)$$
$$\text{where} \quad p(Q^t = i \mid \nu, \gamma, a^t) = \frac{\omega_i\, p(a^t \mid Q^t = i, \nu_i, \gamma_i)}{\sum_j \omega_j\, p(a^t \mid Q^t = j, \nu_j, \gamma_j)}$$

—which means that the placement of a^t will affect the probabilities with which r^t is predicted to lie in each of the "regions", loosely speaking, defined by the distributions $N(\mu_i, \beta_i)$. With enough components, this model is capable of capturing arbitrary input-output relationships.

3.3.2.4. Joint mixtures of experts

The input Gaussian/output Gaussian model of section 3.3.2.3 results in a predictive distribution which could be called "stochastically piecewise constant": given an input, one can infer one or more fixed points around which the corresponding output is likely to be found. If the joint input/output distribution of each mixture component is modelled as a single Gaussian, as in section 3.2.5.4, the outcome is a "stochastically piecewise linear" model: the points around which one expects to find the output move around linearly with the input, at the same time as their relative probabilities also change. This model can be seen as a member of the "mixture of experts" family: a different output-generating rule comes into play depending on the placing of the input.[29]

Like the unmixed joint-Gaussian model, each component i of the mixture of experts can also be parameterised in a slightly different way, with an input-generating distribution $N(\nu_i, \gamma_i)$, a linear transformation κ_i, and an output noise distribution $N(0, \beta_i)$. Under this alternative formulation, the *EM* reestimation rules for κ_i and β_i are weighted versions of (24) and (25):

$$\kappa_i^{n+1} = \left(\sum_t p(Q^t = i \mid \theta^n, r^t, a^t)\, r^t a^{t\prime}\right)\left(\sum_t p(Q^t = i \mid \theta^n, r^t, a^t)\, a^t a^{t\prime}\right)^{-1}$$
$$\beta_i^{n+1} = \left(\frac{\sum_t p(Q^t = i \mid \theta^n, r^t, a^t)\,(r^t r^{t\prime} - \kappa_i^{n+1} a^t r^{t\prime})}{\sum_t p(Q^t = i \mid \theta^n, r^t, a^t)}\right)^{-1}$$

$$\text{where} \quad p(Q^t = i \mid \theta^n, r^t, a^t) \propto p(r^t \mid Q^t = i, \kappa_i^n, \beta_i^n, a^t)\, p(Q^t = i, a^t \mid \omega^n, \nu^n, \gamma^n)$$
$$\text{and} \quad p(r^t \mid Q^t = i, \kappa_i^n, \beta_i^n, a^t) = \left|\frac{\beta_i^n}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\,(r^t - \kappa_i^n a^t)'\, \beta_i^n\, (r^t - \kappa_i^n a^t)\right)$$
$$\text{and} \quad p(Q^t = i, a^t \mid \omega^n, \nu^n, \gamma^n) = \omega_i^n \left|\frac{\gamma_i^n}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\,(a^t - \nu_i^n)'\, \gamma_i^n\, (a^t - \nu_i^n)\right)$$

[29] Xu *et al.*, *An Alternative Model*

The update rules for the input parameters ν, γ are of course similar to (31) and (32)

$$\nu_i^{n+1} = \frac{\sum_t p(Q^t = i \,|\, \theta^n, r^t, a^t)\, a^t}{\sum_t p(Q^t = i \,|\, \theta^n, r^t, a^t)} \tag{34}$$

$$\gamma_i^{n+1} = \left(\frac{\sum_t p(Q^t = i \,|\, \theta^n, r^t, a^t)\, (a^t - \nu^{n+1})(a^t - \nu^{n+1})'}{\sum_t p(Q^t = i \,|\, \theta^n, r^t, a^t)} \right)^{-1} \tag{35}$$

while that for ω is as (33). The predictive distribution is

$$p(r^t \,|\, \theta, a^t) = \sum_i p(Q^t = i \,|\, \omega, \nu, \gamma, a^t)\, p(r^t \,|\, Q^t = i, \kappa_i, \beta_i, a^t)$$

$$\text{where} \quad p(Q^t = i \,|\, \omega, \nu, \gamma, a^t) \propto p(Q^t = i, a^t \,|\, \omega, \nu, \gamma, a^t)$$

Like the other mixture of experts variants, the joint-Gaussian model can be seen as bringing together classification and regression[30]: it first classifies the inputs into Gaussian clusters, and then applies a different linear mapping to members of each class.

3.3.2.5. Conditional gating rules

It is also possible to define another kind of mixture of experts algorithm, in which no attempt is made to model the distribution of the inputs: instead of a^t being conditional (jointly with r^t) on the mixture component choice q^t, the latter is made conditional on the former. Although this "conditional mixture of experts" sounds semantically less ambitious than the joint mixture of experts of section 3.3.2.4, it turns out to be computationally more expensive.

In one possible parameterisation, the "gating rule" by which an expert i is chosen to map the input to the output is similar in form to a Gaussian mixture classification:

$$p(Q^t = i \,|\, \theta, a^t) = \frac{\omega_i\, g(a^t \,|\, \nu_i, \gamma_i)}{\sum_j \omega_j\, g(a^t \,|\, \nu_j, \gamma_j)} \tag{36}$$

$$\text{where} \quad g(a^t \,|\, \nu_i, \gamma_i) = \left| \frac{\gamma_i}{2\pi} \right|^{\frac{1}{2}} \exp\left(-\frac{1}{2} \left(a^t - \nu_i\right)' \gamma_i \left(a^t - \nu_i\right) \right)$$

This is the degree to which each Gaussian "receptive field" $N(\nu_i, \gamma_i)$ will "claim" the input a^t for its expert i, weighted by ω_i and normalised to sum to unity. Note that the Gaussians are being used here purely for their formal properties: there is no suggestion in this conditional mixture model that they express the actual distribution of the inputs (they don't). The M-step of the learning algorithm, in which the (conditional) expected log likelihood is maximised, takes the form

$$\theta^{n+1} = \underset{\theta}{\text{argmax}} \;\; \underbrace{\sum_{t,i} p(Q^t = i \,|\, \theta^n, r^t, a^t) \log p(r^t, Q^t = i \,|\, \theta, a^t)}_{U(\theta)} \tag{37}$$

[30] Jordan & Jacobs, *Hierarchical mixtures of experts* mentions similar, existing models in the statistical literature such as CART and MARS

Writing the logged term as

$$\log p(r^t \mid Q^t = i, \kappa_i, \beta_i, a^t) + \log p(Q^t = i \mid \omega, \nu, \gamma, a^t)$$

and expanding

$$\log p(Q^t = i \mid \omega, \nu, \gamma, a^t) = \log \frac{\omega_i \, g(a^t \mid \nu_i, \gamma_i)}{\sum_j \omega_j \, g(a^t \mid \nu_j, \gamma_j)} \tag{38}$$

it can be seen that the quadraticness of the Gaussian patches' exponents will be blocked by the $\log \sum \cdots$ from being exposed at the top level of the expected log likelihood. This means that its derivative with respect to the parameters ν and γ will not be very nice, and some sort of iterative optimiser will have to be deployed once per iteration of the *EM* algorithm to find a root of the derivative. The model of section 3.3.2.4 avoids this cost; one way of seeing why is to note that the troublesome denominator of 38, which is just like $p(a^t \mid \theta)$, is made part of the likelihood in the joint model and simply "cancels out".

However, the derivative is at least not hard to calculate: the identity

$$\begin{aligned}
\frac{\partial}{\partial \theta_k} \log \frac{f(\theta_i)}{\sum_j f(\theta_j)} &= \frac{\partial}{\partial \theta_k} \log f(\theta_i) - \frac{\partial}{\partial \theta_k} \log \sum_j f(\theta_j) \\
&= \frac{\partial}{\partial \theta_k} \log f(\theta_i) - \frac{\frac{\partial}{\partial \theta_k} f(\theta_k)}{\sum_j f(\theta_j)} \\
&= \left(\delta_{ik} - \frac{f(\theta_k)}{\sum_j f(\theta_j)} \right) \frac{\partial}{\partial \theta_k} \log f(\theta_k) \qquad (39)
\end{aligned}$$

(for any θ and differentiable f) implies that

$$\begin{aligned}
\frac{\partial}{\partial \theta_k} U(\theta) &= \sum_{t,i} p(Q^t = i \mid \theta^n, r^t, a^t) \left(\delta_{ik} - p(Q^t = k \mid \omega, \nu, \gamma, a^t) \right) \frac{\partial}{\partial \theta_k} \log \left(\omega_k \, g(a^t \mid \nu_k, \gamma_k) \right) \\
&= \sum_t \left(p(Q^t = k \mid \theta^n, r^t, a^t) - p(Q^t = k \mid \omega, \nu, \gamma, a^t) \right) \frac{\partial}{\partial \theta_k} \log \left(\omega_k \, g(a^t \mid \nu_k, \gamma_k) \right)
\end{aligned}$$

The Gaussian log-derivatives (15) and (16) then lead immediately to

$$\frac{\partial}{\partial \nu_i} U(\theta) = \sum_t \varepsilon^t_{\theta i} \gamma_i (\nu_i - a^t) \tag{40}$$

$$\frac{\partial}{\partial \gamma_i} U(\theta) = \sum_t \varepsilon^t_{\theta i} \left(\gamma_i^{-1} - (a^t - \nu_i)(a^t - \nu_i)' \right) \tag{41}$$

$$\text{where} \quad \varepsilon^t_{\theta i} = p(Q^t = i \mid \theta^n, r^t, a^t) - p(Q^t = i \mid \omega, \nu, \gamma, a^t) \tag{42}$$

An alternative parameterisation in terms not of the γ_is but of their square roots ψ_i yields a slightly different form of the derivative, which looks to be a little closer to linear and hence better matched with quadratic-approximation (conjugate gradients, quasi-Newton) optimisation methods:

$$\frac{\partial}{\partial \gamma_i} U(\theta) = \sum_t \varepsilon^t_{\theta i} \left(\psi_i^{-1\prime} - \psi_i (a^t - \nu_i)(a^t - \nu_i)' \right) \tag{43}$$
$$\text{where} \quad \psi_i' \psi_i = \gamma_i$$

Finally, the derivatives with respect to the weightings ω_i are

$$\frac{\partial}{\partial \omega_i} U(\theta) = \sum_t \frac{\varepsilon^t_{\theta i}}{\omega_i} \tag{44}$$

One obvious way to perform the M-step is to clamp one of the weightings at some arbitrary value, for instance setting $\omega_0 = 1$, and feed all the rest jointly to a conjugate gradients optimiser along with the derivatives given above.

The qualitative difference between a Gaussian-based gating rule and the Gaussian-mixture input distribution of section 3.3.2.4 can be appreciated most easily by considering the effect of the "extra" term $\cdots - p(Q^t = i \mid \theta, a^t)$ in the derivatives (40) and (41). In the joint-distribution model, the input-generating patches of the experts i are "influenced by" each input a^t in the training set precisely to the extent $p(Q^t = i \mid \theta^n, r^t, a^t)$ that they are judged responsible for having generated it (taking into account the accuracy with which the corresponding linear/Gaussian map predicts the output r^t). In the conditional-distribution model, the receptive fields of expert i are influenced by each input to an "extent" $\varepsilon^t_{\theta i}$ (42) which is positive if they appear to have been activated by it but are not currently "claiming" it, negative if the converse is true, and zero if there is agreement. At the fixed point of the training algorithm, this will mean that it is only the inputs around the margins of each receptive field, and those lying unexpectedly in the "wrong" field, which influence the placement and size of the fields. The precise positions of those inputs which are comfortably inside the "right" fields will not have a significant effect.

Another form of the conditional mixture of experts uses a simpler "softmax-on-linear" function to parameterise the gating probabilities:[31]

$$p(Q^t = i \mid \theta, a^t) = \frac{\omega_i \exp \eta_i' a^t}{\sum_j \omega_j \exp \eta_j' a^t} \tag{45}$$

This model's log likelihood has an η derivative which is quicker to calculate than (40) and (41), although its root must still be found by iterative methods.[32] However, its gating

[31] Jacobs *et al.*, *Adaptive mixtures of local experts*; the Gaussian receptive field gating scheme can be interpreted as a softmax-on-quadratic

[32] Jacobs *et al.*, *Adaptive mixtures of local experts* uses the technique of iteratively reweighted least squares.

rule is quite crude, effectively dividing the input space into regions divided by hyperplanes, with stochastically smooth transitions between expert choices at the boundaries—*i.e.*, if a^t is near a boundary, the decision could go either way. A more discriminating rule than the soft-hyperplane separator can be obtained by using a hierarchy of such classifiers[33], but the Gaussian-like scheme of (36) arguably gives locality properties at least as "sensible", in a simpler form.

3.3.2.6. Confidence regions for conditional mixture models

Just as one can use the input side of a joint—or joint mixture, or joint mixture of experts—model as a confidence region, mapping out the region of the input space for which it has seen example outputs during training (section 3.2.5.3), so it is possible to use the gating rule as a confidence region for a conditional mixture of experts model. However, the semantics will be subtly different.

In former case, the learner supposed that from time to time, a process quite different from that observed during training was brought into play, and both the input and the output would be unpredictable. Thus if an observed input was known from the confidence region to be very different from any seen in the training data, she would not want to make any certain prediction about the output; but even if the input was similar to previous ones, she would still have to bear in mind the possibility that it and the output were produced by the unknown process.

In the case of a conditional mixture of experts, the provision for model failure is expressed instead by including an expert in the mix which "sweeps up" inputs not claimed by the others, and has a completely uninformative output. For instance, it could conveniently be added to the Gaussian receptive field-based model of section 3.3.2.5 as an expert like the others but with a large receptive field and noisy output. Because the learner assumes during training that none of the example inputs trigger the uninformative expert, the other experts' receptive fields are forced to grow so that they exactly map out the region of the input space on which the model has been trained. When this assumption is relaxed, the model automatically takes account of this confidence region in its predictions—but inputs falling inside it can never trigger the uninformative expert, meaning that the model's corresponding output predictions will be less conservative than those of the joint mixture of experts. (It's interesting to note that as the weighting ω_0 of the uninformative expert is raised, the denominator in (38) which makes the difference becomes more nearly constant over the input space, and the remaining receptive fields become more like generating patches.)

[33] Jordan & Jacobs, *Hierarchical mixtures of experts*

Which of these kinds of confidence region is more appropriate will of course depend on the learner's knowledge about the likely ways in which the validity of the training set may in future break down.

3.3.3. Bayesian modelling of dynamical systems

The environment in which an autonomous robot exists (section 3.1) can be modelled, without loss of generality, as a stochastic discrete dynamical system (or Markov chain) of order one. Such a process has the defining characteristic that the probability of the system being in a particular state at each discrete timestep depends only the state it was in at the previous step. If the state cannot be directly measured, but must instead be inferred from the (probabilistic) effect it has on some other quantities such as sensor readings, the process is said to be partially observable.

3.3.3.1. Graphical model of a dynamical system; the Markov property

This situation can be summarised in the following diagram:

$$\begin{tikzcd} \cdots \arrow[r, "\delta"] & H^4 \arrow[r, "\delta"] \arrow[d, "\rho"] & H^5 \arrow[r, "\delta"] \arrow[d, "\rho"] & H^6 \arrow[r, "\delta"] \arrow[d, "\rho"] & H^7 \arrow[d, "\rho"] \\ & R^4 & R^5 & R^6 & R^7 \end{tikzcd} \tag{46}$$

Here H^t is the system's state at time t and R^t its output (given a double border to indicate that it is directly observable). The dynamics of the system, according to which H evolves from step to step, are expressed in the distribution $p(h^{t+1} \mid h^t, \delta)$, and the effect it has on R in the distribution $p(r^t \mid h^t, \rho)$.

"Graphical models" of this kind are convenient because they express the structure of the joint distribution of all the quantities involved in a way at once precise and visually intuitive. The graph may contain no directed cycles, so one can employ the obvious "family tree" terminology, and the assertion it embodies can be captured concisely (if somewhat cryptically) by saying that each node is conditionally independent of its non-descendents given its parents.[34] For example, because H^1 is the only node from which there is a link to H^2, $p(h^2 \mid h^1, x)$ is equal to $p(h^2 \mid h^1)$ for any X among H^2's other non-descendents (H^0, R^0 and R^1). This means that the joint distribution can be written in a form which factors it into separate terms for each timestep:

$$\begin{aligned} p(h^0, r^0, h^1, r^1, h^2, \cdots) &= p(h^1 \mid h^0, r^0, r^1)\, p(h^2 \mid h^1) \cdots \\ &= p(h^0)\, p(r^0 \mid h^0)\, p(h^1 \mid h^0)\, p(r^1 \mid h^1)\, p(h^2 \mid h^1) \cdots \end{aligned}$$

[34] Jordan, *Learning in Graphical Models*

or, writing this "Markov property" more formally,

$$p(h,r \mid \iota,\delta,\rho) = p(h^0 \mid \iota) \prod_t p(h^t \mid h^{t-1},\delta)\, p(r^t \mid h^t,\rho) \tag{47}$$

where ι denotes beliefs about the initial state H^0. (Obviously in this case the conditionality structure of the posterior arises from the causal directionality of the dynamical system, although the diagrammatic formalism can equally well be used for models in which causation plays no role.)

3.3.3.2. EM *learning of dynamical systems*

The Markov property is what makes dynamical systems models good subjects for the application of *EM* learning algorithms[35], with the parameters ι, δ and ρ jointly playing the role of θ and the system state over time H that of the unobserved quantity (also called H in section 3.3.1 above).

For a start, the temporal locality expressed in the Markov property can easily be seen to carry over into a convenient factorisation of the update rule. First expand the posterior $p(\theta, h \mid r)$ in (27) using the identity

$$p(\theta,h \mid r,\mathcal{H}) = \frac{p(h,r \mid \theta)\, p(\theta \mid \mathcal{H})}{p(r \mid \mathcal{H})}$$

where $\mathcal{H}$ includes prior beliefs about ι, δ and ρ (this simply in order to get the r on the left hand side of the conditioning bar). Then the reestimation rule becomes

$$\theta^{n+1} = \operatorname*{argmax}_{\theta} E_h\big[\, \log p(h,r \mid \theta) + \log p(\theta \mid \mathcal{H}) \,\big|\, \theta^n, r\,\big]$$

And from (47)

$$\log p(h,r \mid \theta) = \log p(h^0 \mid \iota) + \sum_t \left(\log p(h^t \mid h^{t-1},\delta) + \log p(r^t \mid h^t,\rho)\right)$$

So the rule decomposes into separate subrules for each part of the parameter; and none of them use more than two temporally adjacent h values at once:

$$\iota^{n+1} = \operatorname*{argmax}_{\iota} \left(\log p(h^0 \mid \iota) + \log p(\iota \mid \mathcal{H})\right) \tag{48}$$

$$\delta^{n+1} = \operatorname*{argmax}_{\delta} \left(\sum_t E_{h^t,h^{t-1}}\big[\, \log p(h^t \mid h^{t-1},\delta) \,\big|\, \theta^t, r\,\big] + \log p(\delta \mid \mathcal{H})\right) \tag{49}$$

$$\rho^{n+1} = \operatorname*{argmax}_{\rho} \left(\sum_t E_{h^t}\big[\, \log p(r^t \mid h^t,\rho) \,\big|\, \theta^t, r\,\big] + \log p(\rho \mid \mathcal{H})\right) \tag{50}$$

[35] Ghahramani, *Learning Dynamic Bayesian Networks* for an overview

Furthermore, the Markov property also suggests a convenient way of computing the pairwise distributions $p(h^t, h^{t-1} \,|\, r, \theta)$ over which the expectations in (49) are taken. Note first that

$$p(h^t, h^{t+1} \,|\, r, \theta) = \frac{p(h^t, h^{t+1}, r \,|\, \theta)}{p(r \,|\, \theta)}$$

the denominator $p(r \,|\, \theta)$ being a constant of no interest (at this stage); and further that

$$p(h^t, h^{t+1}, r \,|\, \theta) = \underbrace{p(r^{[0,t]}, h^t \,|\, \theta)}_{f(h^t)} \;\; p(h^{t+1} \,|\, h^t, \delta) \;\; \underbrace{p(h^{t+1}, r^{[t+1,T)} \,|\, \theta)}_{b(h^{t+1})} \tag{51}$$

or, in words, the probability the system ends up in state h^t having output $r^{[0,t]}$, then transitions to h^{t+1} and goes on to output $r^{[t+1,T)}$. By exploiting the conditionality structure of the posterior, f and b can be defined inductively by

$$\begin{aligned} f(h^{t+1}) &= p(r^{[0,t+1]}, h^{t+1} \,|\, \theta) \\ &= p(r^{t+1} \,|\, h^{t+1}, \rho) \int_{h^t} p(h^{t+1} \,|\, h^t, \delta)\, p(r^{[0,t]}, h^t \,|\, \theta) \\ &= p(r^{t+1} \,|\, h^{t+1}, \rho) \int_{h^t} p(h^{t+1} \,|\, h^t, \delta)\, f(h^t) \\ \text{with} \quad f(h^0) &= p(h^0 \,|\, \iota) \end{aligned} \tag{52}$$

$$\begin{aligned} b(h^{t-1}) &= p(h^{t-1}, r^{[t-1,T)} \,|\, \theta) \\ &= p(r^{t-1} \,|\, h^{t-1}, \rho) \int_{h^t} p(h^t \,|\, h^{t-1}, \delta)\, p(r^{[t,T)}, h^t \,|\, \theta) \\ &= p(r^{t-1} \,|\, h^{t-1}, \rho) \int_{h^t} p(h^t \,|\, h^{t-1}, \delta)\, b(h^t) \\ \text{with} \quad b(h^{T-1}) &= p(h^{T-1}, r^{[0,T)} \,|\, \theta) = f(h^{T-1}) \end{aligned} \tag{53}$$

These equations are special cases of those used for belief propagation in more general graphical models.[36]

The upshot is that if the distributions defining the model are well-behaved, an elegant and efficient algorithm drops out. Specifically:

1) If the functional form of $p(h^{t+1} \,|\, h^t, \delta)$ (the system dynamics) is closed under convolution with itself,

2) and with $p(r^t \,|\, h^t, \rho)$ (the output likelihood) and $p(h^0 \,|\, \iota)$ (the initial state), then f and b will have that functional form as well, and so will the pairwise distribution used in (49)'s expectation.

3) If the functional form of $p(h^{t+1} \,|\, h^t, \delta)$ is also closed under marginalisation, it will cover the distribution used in (50)'s expectation, since $p(h^t \,|\, r, \theta) = \int_{h^{t+1}} p(h^t, h^{t+1} \,|\, r, \theta)$.

4) If, finally, the distributions $p(h^{t+1} \,|\, h^t, \delta)$ and $p(r^t \,|\, h^t, \rho)$ are drawn from the common "exponential" family[37], the expectations in (50) and (49) may go through

[36] Lauritzen & Spiegelhalter, *Local computations*

[37] Lee, *Bayesian Statistics*, p. 63

cleanly and give rise to a simple optimisation.

Making predictions for observations r^T not yet made will also be easy, since the density required is

$$p(r^T \mid \theta, r^{[0,T)}) \tag{54}$$

and this can be obtained from f^T, as defined in (52) by marginalising over h^T.

There are two classes of model which meet these requirements completely, namely the "hidden Markov model" and the "Kalman filter".

3.3.3.3. The hidden Markov model

In a hidden Markov model (HMM)[37], the hidden state H^t is instantiated as a discrete quantity, say Q^t, and the dynamics parameter δ as an arbitrary transition matrix ω, so that

$$p(Q^{t+1} = i \mid Q^t = j, \omega) = \omega_{ij} \tag{55}$$

trivially satisfies the conditions 1 and 3 given in section 3.3.3.2 for $p(h^{t+1} \mid h^t, \delta)$. The output distribution $p(r^t \mid q^t, \rho)$ is a mixture, where the choice of mixing component is determined by the state q^t. Symbolically,

$$p(r^t \mid q^t = i, \rho) = p(r^t \mid q^t = i, \rho_i)$$

Clearly this arrangement also satisfies condition 2. The components $p(r^t \mid Q^t \;=\; i, \rho_i)$ which are mixed to form the output distribution are typically either discrete or Gaussian. In the latter case, the HMM is just a recursive (*i.e.* time-series) version of the Gaussian mixture model described in section 3.3.2, the only difference being that the unconditional probabilities ω_i with which each component i was chosen to generate the output are replaced by the Markov-conditional probabilities ω_{ij}.

Substituting this simple model into the framework of section 3.3.3.2, the equations (52) and (53) implementing the E-step at iteration n of the EM algorithm become the so-called "forward-backward" equations

$$\begin{aligned} f_i^0 &= \iota_i^n \\ f_i^{t+1} &= p(r^t \mid \rho_i^n) \sum_j \omega_{ij}^n f_j^t \end{aligned} \tag{56}$$

$$\begin{aligned} b_j^{T-1} &= f_j^{T-1} \\ b_j^{t-1} &= p(r^t \mid \rho_j^n) \sum_i \omega_{ij}^n b_i^t \end{aligned} \tag{57}$$

[37] Rabiner, *A Tutorial on Hidden Markov Models*

It is convenient to define also

$$\xi_{ij}^t = p(Q^t = j, Q^{t+1} = i, r \mid \theta^n) \quad \text{which is} \quad f_j^t \omega_{ij}^n b_i^{t+1} \tag{58}$$

(*cf.* (51)).

The update rule (49) for the dynamics parameter (transition matrix) becomes

$$\omega_{ij}^{n+1} = \sum_t p(q^t = j, q^{t+1} = i \mid \theta^n, r) = \sum_t \xi_{ij}^t \tag{59}$$

while the rule (48) for the initial state becomes

$$\iota_j^{n+1} = \sum_t p(q^0 = j \mid \theta^n, r) = \sum_i \xi_{ij}^0 \tag{60}$$

Both are easily established using Lagrange multipliers in exactly the same way as for (13). The priors $p(\omega \mid \mathcal{H})$ and $p(\iota \mid \mathcal{H})$ are ignored (assumed to be dominated by the likelihood).[38]

If, as is commonly the case, the components $p(r^t \mid \rho_i)$ of the output distribution are Gaussians, with means μ_i and precisions β_i, then their update rules are

$$\mu_i^{n+1} = \frac{\sum_t p(q^t = i)\, r^t}{\sum_t p(q^t = i)} \tag{61}$$

$$\beta_i^{n+1} = \left(\frac{\sum_t p(q^t = i)\, (r^t - \mu_i^{n+1})(r^t - \mu_i^{n+1})'}{\sum_t p(q^t = i)} \right)^{-1} \tag{62}$$

—*cf.* the similar weighted averages (31) and (32). (Again, the priors are treated as irrelevant.) The combination of the forward-backward equations with these or similar updates in an *EM* iteration sequence is called the Baum-Welch algorithm, after its discoverers.

The HMM's prediction for an observation r^T not yet made can, as was pointed out for (54), be evaluated by marginalising q^T out of f^T:

$$p(r^T \mid r^{[0,T)}, \theta) = \sum_i p(r^T \mid Q^T = i, \rho_i)\, p(Q^T = i \mid r^{[0,T)}, \theta) \tag{63}$$

Hidden Markov models have been applied successfully to problems in operations research[39], reinforcement learning[40] and many other fields including most saliently speech recognition[41]. One of their most useful properties is that they make no assumptions about the way successive outputs generated by the system are related to each other. This means

[38] although one can adopt a Dirichlet prior with little extra complication: see *e.g.* Koenig & Simmons, *Unsupervised learning* for a robotics application of this idea

[39] cited in Kaelbling *et al.*, *Planning and Acting*

[40] Kaelbling *et al.*, *Planning and Acting*

[41] Rabiner, *A Tutorial on Hidden Markov Models*

that they are well suited to capturing patterns of underlying temporal structure which express themselves in characteristic, but arbitrary, readings. HMMs can cope particularly well with the sort of randomness termed "nondeterminism" in section 3.1.2.1: if there are situations in which the process being modelled can follow any of several trajectories for a certain period, during which time the available observations are not sufficient to disambiguate them, the model can in effect maintain all the possible trajectories as hypotheses. In this case its "mixed" predictions will present the outcomes corresponding to each trajectory hypothesis, along with their assessed probabilities and their extra, localised "noise" uncertainty (β, in the case of an HMM with Gaussian output densities). The model is not constrained to output any single best guess or compromise estimate, and its mixture predictions are ideal for feeding into the Bayesian "greatest expected gain" decision rule (26).

3.3.3.4. Input-output HMMs

Just as the Gaussian mixture model of section 3.3.2 simply discards its inputs, so in its recursive analogue, the HMM, there is no way for external agencies to influence the trajectory of the dynamical system. In order to make a model suitable for use in control tasks (like robotics), the conditionality graph must be extended to include an input a^t for each timestep, thus:

$$\begin{tikzcd}
A^0 \arrow[dr] & A^1 \arrow[dr] & A^2 \arrow[dr] & A^3 \arrow[dr] & \\
Q^0 \arrow[r, "\delta"] \arrow[d, "\rho"] & Q^1 \arrow[r, "\delta"] \arrow[d, "\rho"] & Q^2 \arrow[r, "\delta"] \arrow[d, "\rho"] & Q^3 \arrow[r, "\delta"] \arrow[d, "\rho"] & \cdots \\
R^0 & R^1 & R^2 & R^3 &
\end{tikzcd}$$

In this "input-output hidden Markov model" or IOHMM[42], the dynamics are made conditional on the a^ts as well as the q^ts, in some form $p(q^{t+1} \mid q^t, a^t, \delta)$ more complicated than the simple transition matrix of (55); the overall likelihood becomes

$$p(q, r \mid a, \iota, \delta, \rho) = p(q^0 \mid \iota) \prod_t p(q^t \mid q^{t-1}, a^{t-1}, \delta)\, p(r^t \mid q^t, \rho)$$

—*cf.* (47). The dynamics distribution can, for instance, be adapted from the Gaussian receptive field "gating rule" (36):[43]

$$p(Q^t = i \mid Q^{t-1} = j, \theta, a^t) = \frac{\omega_{ij}\, g(a^t \mid \nu_i, \gamma_i)}{\sum_k \omega_{ik}\, g(a^t \mid \nu_k, \gamma_k)} \tag{64}$$

$$\text{where} \quad g(a^t \mid \nu_i, \gamma_i) = \left|\frac{\gamma_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(a^t - \nu_i\right)' \gamma_i \left(a^t - \nu_i\right)\right)$$

[42] Bengio & Frasconi, *An Input Output HMM Architecture*, alternatively called a partially observable Markov decision process or POMDP (Kaelbling *et al.*, *Planning and Acting*)

[43] alternatively, Bengio & Frasconi, *An Input Output HMM Architecture* represents the dynamics distribution using an MLP neural network

In this minimal generalisation of the static gating rule, each component i still has a single receptive field of its own which competes to "claim" the inputs a^t, but the components' bids are weighted differently according to the component j which was active at $t-1$ (hence ω_{ij} in place of ω_i).

The extension of the HMM model to a-conditional dynamics essentially preserves the forward-backward equations (56) and (57)—the only difference being that the transition probabilities are no longer just constant parameters ω_{ij} but rather a-dependent probabilities (64) which have to be evaluated—so the update rules for the initial state (60), output means (61) and output precisions (62) remain efficient. However, if A is a continuous variable, the update rule for δ inevitably becomes much more expensive, because of the requirement that the transition probabilities from each state at any given time must sum to unity: there is no function $a^t, q^t \to q^{t+1}$ which satisfies that constraint and has a log whose derivative is linear (*cf.* the discussion around equation (38)). Therefore the update rule (49)

$$(\omega,\nu,\gamma)^{n+1} = \operatorname*{argmax}_{\omega,\nu,\gamma} E_q\Big[\sum_t \log p(q^t \mid q^{t-1},\omega,\nu,\gamma,d^t)\,\Big|\, d,\theta^n\Big] \tag{65}$$

has to be implemented with an iterative optimiser, just like the M-step for the gating parameters of the (static) mixture of experts described in section 3.3.2.5.

Deploying the same arguments as for (40) and (41), the derivatives to be passed to the optimiser can be inferred from the template

$$\begin{aligned}\frac{\partial}{\partial\theta_k}U(\theta) &= \sum_{t,i,j} p(Q^t=i,Q^{t-1}=j\mid\theta^n,d)\,\big(\delta_{ik} - p(Q^t=k\mid Q^{t-1}=j,\omega,\nu,\gamma,d^t)\big)\\ &\qquad \frac{\partial}{\partial\theta_k}\log\big(\omega_{kj}\,g(d^t\mid\nu_k,\gamma_k)\big)\\ &= \sum_{t,j}\big(p(Q^t=k,Q^{t-1}=j\mid\theta^n,d) - p(Q^{t-1}=j\mid\theta^n,d)\,p(Q^t=k\mid Q^{t-1}=j,\omega,\nu,\gamma,d^t)\big)\\ &\qquad \frac{\partial}{\partial\theta_k}\log\big(\omega_{kj}\,g(d^t\mid\nu_k,\gamma_k)\big)\end{aligned}$$

so that

$$\frac{\partial}{\partial\nu_i}U(\theta) = \sum_t \varepsilon^t_{\theta i}\,\gamma_i(\nu_i - d^t) \tag{66}$$

$$\frac{\partial}{\partial\gamma_i}U(\theta) = \sum_t \varepsilon^t_{\theta i}\,\big(\gamma_i^{-1} - (d^t-\nu_i)(d^t-\nu_i)'\big) \tag{67}$$

$$\text{where}\quad \varepsilon^t_{\theta i} = p(Q^t=i\mid\theta^n,d) - \sum_j p(Q^{t-1}=j\mid\theta^n,d)\,p(Q^t=i\mid Q^{t-1}=j,\omega,\nu,\gamma,d^t) \tag{68}$$

Since the ω_{ij}s are attached to specific "previous" mixing states j, the derivative with respect to them lacks the summation over j:

$$\frac{\partial}{\partial\omega_{ij}}U(\theta) = \sum_t \frac{\varepsilon^t_{\theta ij}}{\omega_{ij}} \tag{69}$$

$$\text{where}\quad \varepsilon^t_{\theta ij} = p(Q^t=i,Q^{t-1}=j\mid\theta^n,d) - p(Q^{t-1}=j\mid\theta^n,d)\,p(Q^t=i\mid Q^{t-1}=j,\omega,\nu,\gamma,d^t)$$

As with a static gating rule, there is also the option of parameterising the derivative by the square root of γ rather than by γ itself, if it helps the optimiser.

IOHMM-like models are used extensively in high-level robot learning (under the name of Markov decision processes; see section 2.3.1), and have proved useful for low-level tasks as well—for instance, Meilă and Jordan learn a model for the development of the state of contact between a compliant robot arm and the objects it handles[44].

3.3.3.5. The Kalman filter

In a Kalman filter (KF)[45], the hidden state H is instantiated as a continuous quantity, say V, and the both the dynamics and output generation are linear with Gaussian noise:

$$\begin{aligned} V^{t+1} &= \lambda V^t + N(0,\alpha) \\ R^t &= \kappa V^t + N(0,\beta) \end{aligned}$$

so that

$$\begin{aligned} p(v^{t+1} \,|\, v^t, \lambda, \alpha) &= \left|\frac{\alpha}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(v^{t+1} - \lambda v^t\right)' \alpha \left(v^{t+1} - \lambda v^t\right)\right) \\ p(r^t \,|\, v^t, \kappa, \beta) &= \left|\frac{\beta}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^t - \kappa v^t\right)' \beta \left(r^t - \kappa v^t\right)\right) \end{aligned} \tag{70}$$

Because Gaussians are closed under convolution and marginalisation (as can be shown by combining/factorising their quadratic exponenents), as well as addition and linear transformation, the KF's linear/Gaussian model satisfies conditions 1 and 3 in section 3.3.3.2 just as the HMM does.

However the equations for computing f and b ((52) and (53)), known as the Kalman-Rauch recursions[46], are less straightforward than the corresponding "forward-backward equations" for the HMM (*cf.* (56) and (57)). Adopting the following notation

$$\begin{aligned} \bar{v}^t_\tau &= E[v^t \,|\, r^{[0,\tau]}, \theta^n] \\ \tilde{v}^t_\tau &= \text{Var}[v^t \,|\, r^{[0,\tau]}, \theta^n] \\ \tilde{v}^{t,t-1}_\tau &= \text{Cov}[v^t, v^{t-1} \,|\, r^{[0,\tau]}, \theta^n] \\ \bar{\iota} &= E[v^0 \,|\, \iota^n] \\ \tilde{\iota} &= \text{Var}[v^0 \,|\, \iota^n] \end{aligned} \tag{71}$$

[44] Meilă & Jordan, *Learning Fine Motion*

[45] Ghahramani & Hinton, *Parameter Estimation*

[46] Rauch, *Solutions to the linear smoothing problem*; Shumway & Stoffer, *An approach*

(all the distributions being Gaussian), the forward pass begins

$$\bar{v}^0_{-1} = \bar{\iota}$$
$$\tilde{v}^0_{-1} = \tilde{\iota}$$

and continues

$$\begin{aligned}
\bar{v}^t_{t-1} &= \lambda \bar{v}^{t-1}_{t-1} \\
\tilde{v}^t_{t-1} &= \lambda \tilde{v}^{t-1}_{t-1} \lambda' + \alpha^{-1} \\
K^t &= \tilde{v}^t_{t-1} \kappa' (\kappa \tilde{v}^t_{t-1} \kappa' + \beta^{-1})^{-1} \\
\bar{v}^t_t &= \bar{v}^t_{t-1} + K^t (r^t - \kappa \bar{v}^t_{t-1}) \\
\tilde{v}^t_t &= \tilde{v}^t_{t-1} - K^t \kappa \tilde{v}^t_{t-1}
\end{aligned}$$

The backward pass begins

$$\tilde{v}^{T-1,T-2}_{T-1} = (I - K^{T-1}\kappa)\lambda \tilde{v}^{T-2}_{T-2}$$

and continues (noting that $\tilde{v}^{T-1}_{T-1}$ has been computed at the end of the forward pass)

$$\begin{aligned}
J^{t-1} &= \tilde{v}^{t-1}_{t-1} \lambda' (v^t_{t-1})^{-1} \\
\bar{v}^{t-1}_{T-1} &= \tilde{v}^{t-1}_{t-1} + J^{t-1} (\bar{v}^t_{T-1} - \bar{v}^t_{t-1}) \\
\tilde{v}^{t-1}_{T-1} &= \tilde{v}^{t-1}_{t-1} + J^{t-1} (\tilde{v}^t_{T-1} - \tilde{v}^t_{t-1}) J^{t-1\prime} \\
\tilde{v}^{t-1,t-2}_{T-1} &= \tilde{v}^{t-1}_{t-1} J^{t-2\prime} + J^{t-1} (\tilde{v}^{t,t-1}_{T-1} - \lambda \tilde{v}^{t-1}_{t-1})
\end{aligned}$$

The expected log posteriors in the update rule schemas for $\delta = \lambda, \alpha$ (49) and $\rho = \kappa, \beta$ (50) are both quadratic in the KF, because every distribution involved is Gaussian. Setting all their derivatives to zero (using the same algebra as for (24) and (25)) yields the KF-specific update rules

$$\lambda^{n+1} = PB^{-1} \tag{72}$$

$$\alpha^{n+1^{-1}} = \frac{1}{T-1} \left(C - \lambda^{n+1} P' \right) \tag{73}$$

$$\kappa^{n+1} = \left(\sum_t r^t \bar{v}^{t\prime} \right) \left(\sum_t B^t \right)^{-1} \tag{74}$$

$$\beta^{n+1^{-1}} = \frac{1}{T} \sum_t (r^t r^{t\prime} - \kappa^{n+1} \bar{v}^t_{T-1} r^{t\prime}) \tag{75}$$

the sufficient statistics used in these formulas (which are given names here for later reference) being given by

$$\begin{aligned}
B &= \sum_{t \in [\,0, T-1\,)} B^t \\
C &= \sum_{t \in [\,0, T-1\,)} B^{t+1} \\
\text{where} \quad B^t &= E[\,v^t v^{t\prime} \,|\, r\,] = \bar{v}^t_{T-1} {\bar{v}^t_{T-1}}' + \tilde{v}^t_{T-1} \\
P &= \sum_{t \in [\,0, T-1\,)} P^{t+1,t} \\
\text{where} \quad P^{t+1,t} &= E[\,v^{t+1} v^{t\prime} \,|\, r\,] = \bar{v}^{t+1}_{T-1} {\bar{v}^t_{T-1}}' + \tilde{v}^{t+1,t}_{T-1}
\end{aligned} \tag{76}$$

The update rule (48) for ι simply sets the new estimate of the initial state to what it was computed to be at the end of the Kalman-Rauch recursions:

$$\bar{\iota}^n = \bar{v}^0_T$$

$$\tilde{\iota}^n = \tilde{v}^0_T$$

As in the case of the HMM (see equation 63), the prediction made by a KF for the incoming observation r^T can be obtained by marginalising v^T out of f^T, giving

$$p(r^T \mid r^{[0,T)}, \theta) = \int_{v^T} p(r^T \mid v^T, \rho_\iota)\, p(v^T \mid r^{[0,T)}, \theta)$$

which is a Gaussian with mean $\kappa\bar{v}^T_{T-1}$ and variance $\beta^{-1} + \kappa\tilde{v}^T_{T-1}\kappa'$.

Kalman filters are used widely in engineering for tasks in which the character of the data can be described accurately as "trends" linking variables and extending over time. They are particularly suitable for applications like tracking moving objects in which position/velocity/acceleration estimates play an important role (including robot localisation—section 2.1.1.1).

3.3.3.6. *Kalman filters with inputs*

If the model is augmented to include inputs a^t, as was done for the hidden Markov model in section 3.3.3.4, the KF can be used for process control tasks as well. Indeed, the extension causes much less disruption to the KF than it does to the HMM: making the dynamics linear in a^t as well as v^t still leaves the v distribution Gaussian, and the update rules retain their one-shot solutions.

Chapter 4

The Samovar model

4.1. Motivation

The hidden Markov model (section 3.3.3.3) and Kalman filter (section 3.3.3.5) can work very effectively in some domains, but the expressive power of both models falls short of that required for the robot environment modelling task. The "Samovar" model presented in this section is a hybrid which exploits the complementary strengths of each to obtain greater generality, while still inheriting some of their elegance and tractability.

4.1.1. A recursive mixed-linear model

The mixture of experts algorithm and its relatives show how linear regression, with its ability to capture trends, can be combined with cluster-classification, with its robustness in the face of "arbitrary" patterns, to make a piecewise linear model which has both desirable properties and can be learned reasonably quickly using a simple *EM* algorithm. It is natural to ask whether their time-series analogues, the Kalman filter and the hidden Markov model, can also be brought together in a similar way.

4.1.1.1. The architecture space

The basic idea can be expressed in the following table:

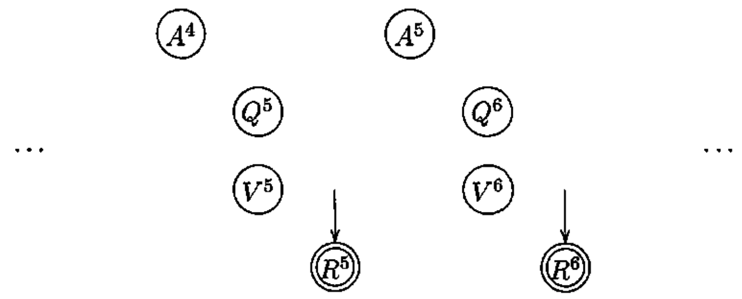
Mapping type	*Static version*	*Recursive version*
Constant	Gaussian	(Gaussian)
Linear	Linear regressive model	KF
Piecewise constant	Gaussian mixture model	HMM
Piecewise linear	Mixture of experts	?

To get a more detailed appreciation of the possibilities, and to put the models developed previously in perspective, it helps to consider them all as variants of the same graphical model, and characterise them according to which of its possible links (conditional dependencies) they incorporate and which they exclude. The quantities in play are

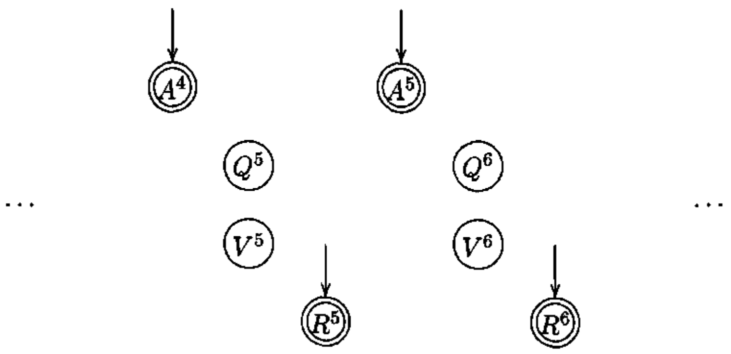
A^t	the process input; other quantities may be conditioned on this or modelled jointly to create a "confidence region" (section 3.2.5.3)
R^t	the process output
Q^t	hidden "mixing" variable: a discrete quantity, not directly observable, which determines the choice between possible mixing components in generating R^t; if R^t is not dependent on Q^t, then its distribution is not a mixture
V^t	hidden "linear" variable: a continuous quantity, not directly observable, on which R^t may depend linearly

In addition, recursive models such as the HMM and KF (section 3.3.3) can be expressed by introducing dependencies between hidden variables at successive timesteps. Some of the schemes mentioned previously are shown below:

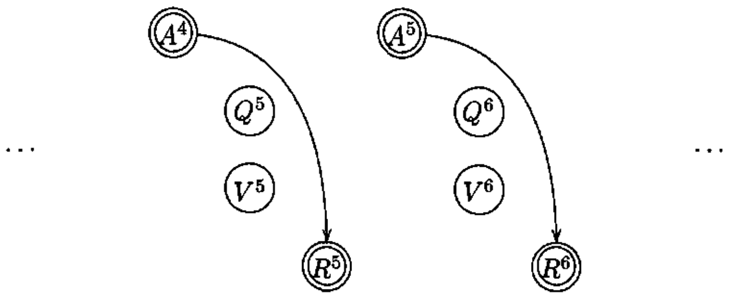
Gaussian (section 3.2.5.2). (R^t is independent of all other variables, but a stub dependency arc is included to draw attention to the distribution being modelled.)



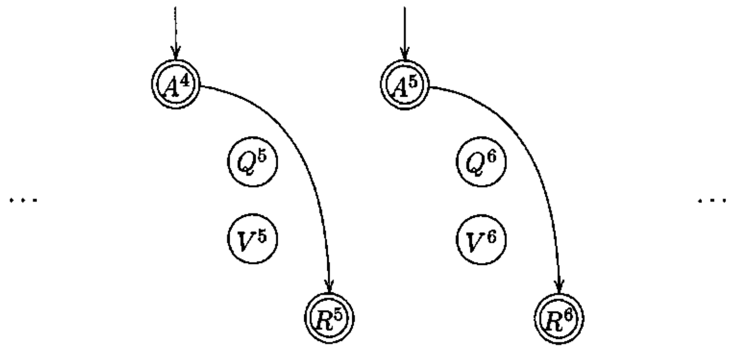
Input and output Gaussians (section 3.2.5.3). A simple Gaussian with a confidence region. Both R^t and A^t are modelled.



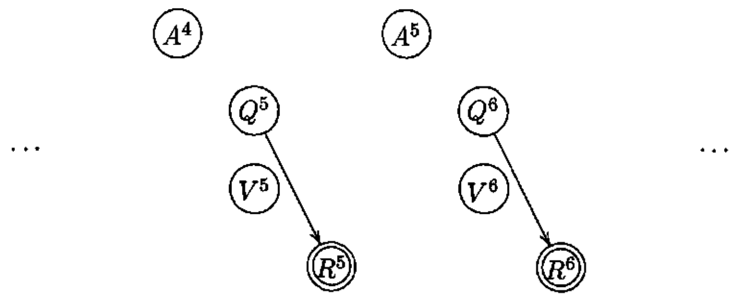
Linear/Gaussian mapping (section 3.2.5.4).



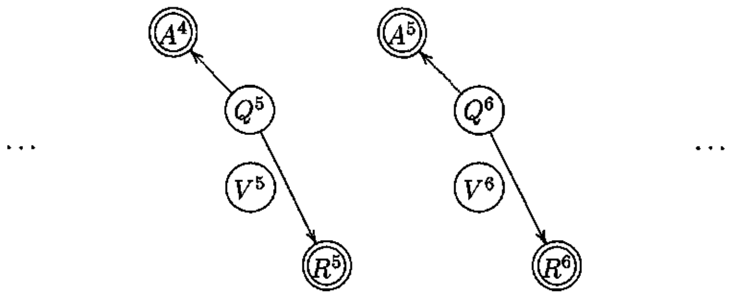
Joint Gaussian (section 3.2.5.4). Equivalent to a Linear/Gaussian mapping with a confidence region.



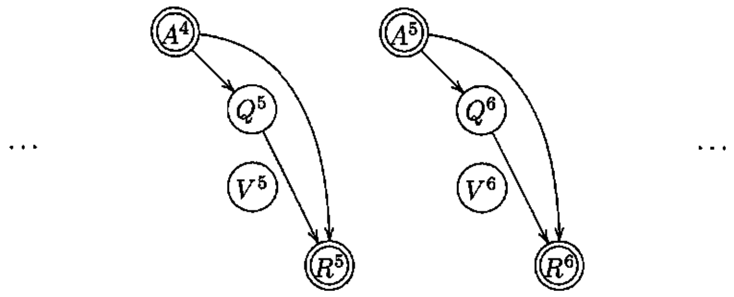
Unconditional Gaussian mixture (section 3.3.2.1).



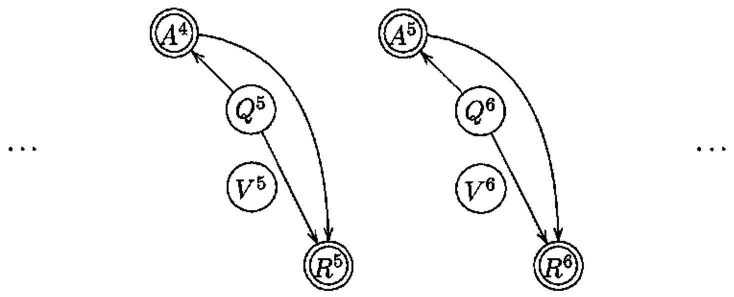
Joint input-output Gaussian mixture (section 3.3.2.3). This can be seen as a "piecewise constant" mixture of experts with a confidence region.



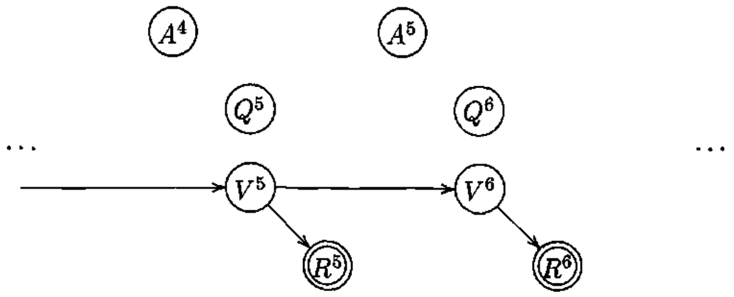
Conditional mixture of experts (section 3.3.2.5). The standard, "piecewise linear" mixture of experts.



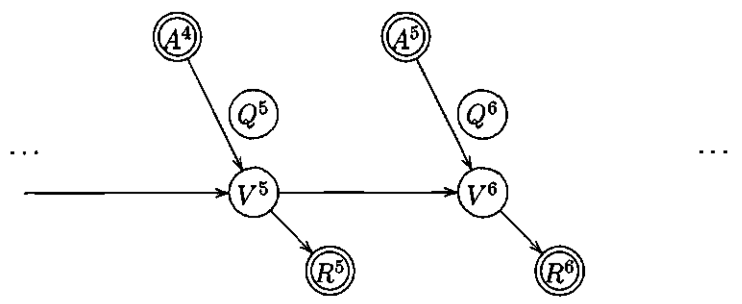
Joint mixture of experts (section 3.3.2.4). This can be seen as a (piecewise linear) mixture of experts with a confidence region.



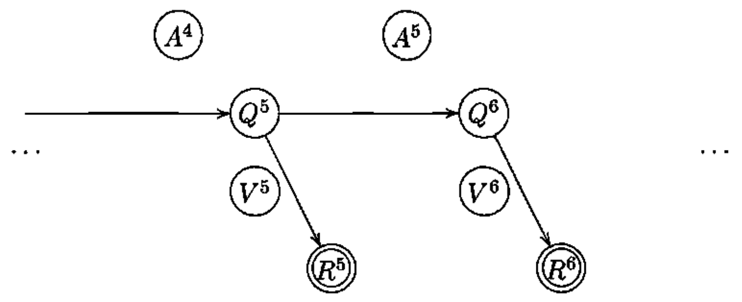
Kalman filter (section 3.3.3.5).



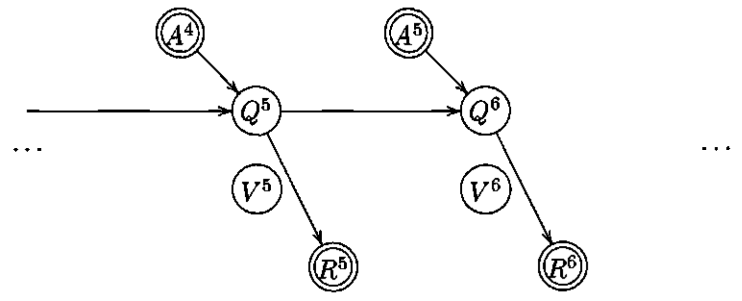
Kalman filter with input (section 3.3.3.6).



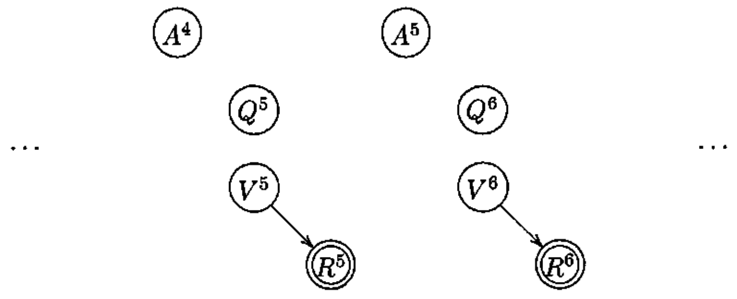
Hidden Markov model (section 3.3.3.3).



Input-output hidden Markov model (section 3.3.3.4).

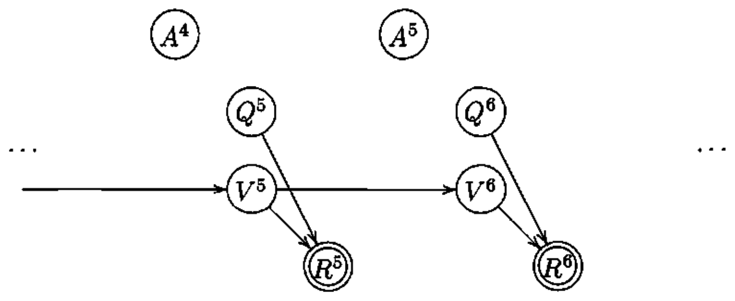


Many other variations are possible.[1] For instance, the method of factor analysis could be represented like this:

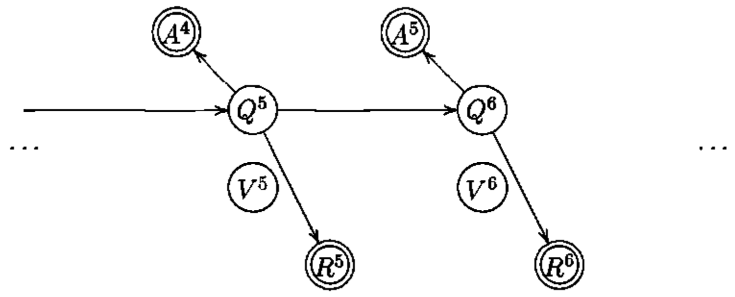


[1] Note that this architecture space only provides for continuous inputs, not a discrete one which could control a mixing process directly. However, it would be easy to include a discrete input in any of the models discussed below.

Or one could imagine a Kalman filter with a mixture output:



or an IOHMM with a confidence region for its inputs:

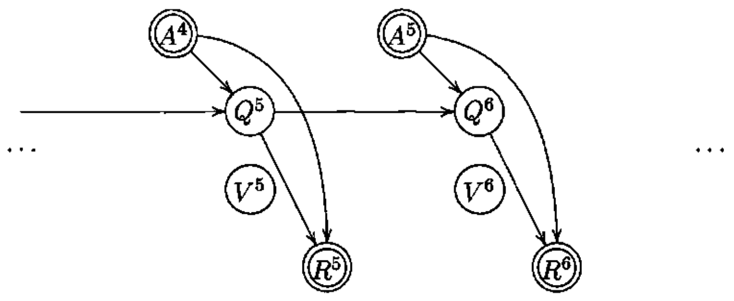


(77)

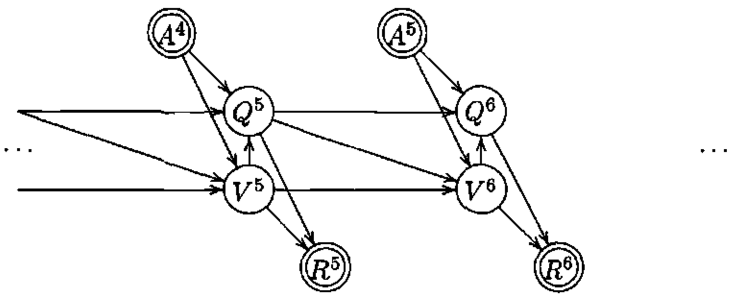
(which would have the added advantage of yielding a highly efficient reestimation rule for the "gating" parameters—*cf.* section 3.3.2.5).

4.1.1.2. The possibilities

In the context of the robot environment modelling task, the most interesting option is clearly something which could be called a "recursive mixture of experts"—a piecewise linear time-series model. But that term could be applied to any of a number of possible models, each of which makes a different tradeoff between the completeness of its conditionality structure and the cleanness of its learning algorithm. At the simplest, one could take a static mixture of experts and make its mixing probabilities (alone) recursive:



This is really an IOHMM with linear outputs; the corresponding *EM* algorithm is of essentially similar character to that of the IOHMM. At the other extreme, one could make the linear component of the model recursive as well, and even have the mixing probabilities depend on the linear hidden state:



(78)

This "denser" model is powerful—its state Q^t, V^t evolves according to a piecewise linear dynamics, with which it is easy to approximate a wide range of nonlinear systems—but it inherits none of the tractability of the HMM or KF, because the conditions defined in section 3.3.3.2 for the existence of a nice *EM* algorithm are not even approximately met: the mixing and linear processes are too closely intertwined:

1) **V-mixing problem.** The linear hidden state V^t will be rendered non-Gaussian by its mixed-linear relationship with V^{t-1} and V^{t+1}.

2) **V-normalising problem.** The distribution of V^t will also be disturbed by its role in determining the mixing hidden state Q^{t+1}—recall from section 3.3.2.5 that conditioning a multinomial variable on a continuous one will always lead to problems stemming from the need to normalise the distribution to sum to unity.

3) **Q-dynamics problem.** The relationships between the mixing hidden states Q^t will be made much more complicated by indirect influences via the V^ts, which will disturb their joint distribution and make the reestimation rule for the gating parameters somewhat problematic.

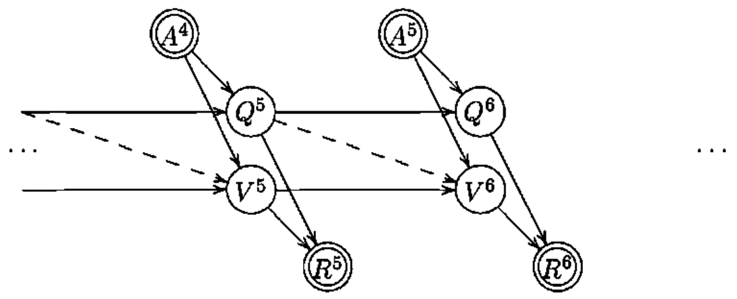
Consequently, nothing like the forward-backward equations (section 3.3.3.3) or the Kalman-Rauch recursions (section 3.3.3.5) will be available for computing the distribution of the hidden state variables V and Q.

4.2. A solution

To get past these difficulties, the Samovar model sacrifices a little of the "dense" model's expressiveness, either omitting or modifying the link $V^t \rightarrow Q^t$ which gives rise to the thorny V-normalising problem and Q-dynamics problem, and approximates the effect of the counterpart link $Q^t \to V^{t+1}$ in such a way that an accommodation can be made with the V-mixing problem. It is then able to deploy the fast KF and HMM learning algorithms as subroutines to obtain a reasonably efficient procedure for learning a mixed-linear model.

4.2.1. The conditional Samovar model

The "conditional" variant (see section 4.2.3 for the "joint" variant) of the Samovar model looks like this:



(79)

The linear hidden state V evolves according to mixed-linear (piecewise linear) dynamics, but the link $V^t \to Q^t$ has not been included: V does not in its turn affect the sequence Q of mixture components. As a result, V and Q remain sufficiently decoupled that the product-rule factorisation of the likelihood

$$p(r, v, q \mid a) = p(r \mid v, q, a)\, p(v \mid q, a)\, p(q \mid a)$$

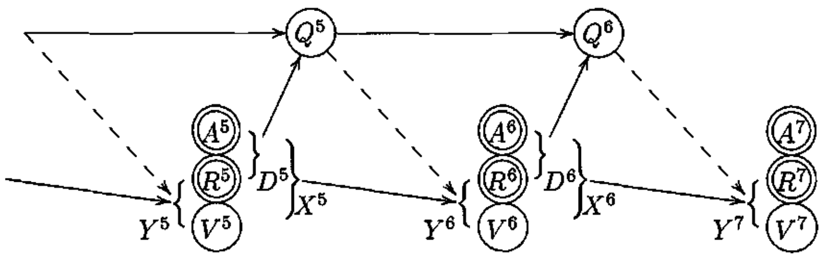
can be treated as a Kalman filter likelihood inside a hidden Markov model likelihood, leading (section 4.2.2) to a nested *EM* algorithm based on their respective recursions.

Of course, the simplification makes the model a little less expressive: there is no provision for Kalman filter-style inferences, made on the basis of a series of inputs and outputs and expressed in the linear hidden state, to influence the judgment as to which mapping component is active at each timestep. For instance, it might in principle be possible for the robot to judge its velocity from successive readings from a range sensor; and it might be the case that the model dynamics should be qualititatively different (using a different mapping) at different velocities; but there would not necessarily be any way in the simplified model for V^t to have the required effect on Q^t in a timely manner. Certainly, once the model has made a vague or incorrect prediction by failing to deploy the linear mapping component appropriate to the current velocity, it will be able to recognise *post hoc* that its Q^t estimate was poor, and thereafter use the discrete part of the dynamics (*i.e.* the transition matrix ω) to "remember" that it is in a certain qualitative velocity régime. However, there will be no avoiding that first, corrective mistake, because there is no other channel by which information can flow from the V^t estimate to the Q^t estimate.

4.2.1.1. *Assimilating the dynamics and the output function*

Before the exposition of a training algorithm for the Samovar model gets under way, a slight simplification will be made which reduces the number of model parameters in play.

Up to now, the discussion of dynamical systems models has taken for granted a schema in which the observations available to the learner are obtained via an output function from a hidden state which evolves according to a dynamics function (ρ and δ in (46)). However, for the purposes of the robot environment modelling task—in which the model is openly approximative[2] and no claim is made about the true ontology and conditionality structure of the process under consideration, *i.e.* the environment—the output function is redundant. The point of a piecewise linear model is that it can match a very wide range of dynamics functions; it should, therefore, be able to adapt itself during training to fit in with a fixed, simple output function. Indeed the natural "output function" to adopt is simply an identity projection of some elements of the hidden state, R^t being thereby assimilated directly into H^t. Then the model dynamics can be considered as approximating the mapping between one "world state" X^t and the next, each world state comprising a hidden part V^t and a visible part $D^t = R^t, A^t$. With this adjustment, the graphical model for the conditional Samovar model could be written thus:

 (80)

(In the notation of section 3.3.3.5, κ_i and β_i are fixed as follows:

$$\beta_i = \infty \quad \kappa_i = \begin{array}{c} \begin{array}{ccc} V & R & A \end{array} \\ \left(\begin{array}{ccc} 0 & I & 0 \end{array} \right) \end{array} \tag{81}$$

where the label V marks the columns of κ_i which operate on the linear hidden state, and R and A those which operate on the sensor readings and actions respectively.) Arranging the conditionality structure this way makes it easier to exploit obvious relationships between the known parts of adjacent world states, since they do not have to be mediated through a supposedly unknown value V^t. Indeed this model can be seen as a mixture of static linear mappings, augmented with just enough hidden linear state to capture longer-term dependencies; a model which followed the pattern of (46), in which all the linear state was

[2] in the sense of section 3.2.4.3

unobserved, would have to make more expensive inferences over a higher-dimensional V, as well as learning an output function to go with it.

The values r^t are those the robot can observe directly—*i.e.* the readings coming out of the back of its sensors, including the effects of noise, sensor failure or whatever. They are not supposed to correspond directly to whatever physical quantities the sensors are presumed to measure, and they cannot be considered to play a direct causal role in the future evolution of the robot's world. But this is not a problem, because the robot is not claiming to have learned a physical model of its environment, but only what it actually needs, according to Bayesian decision theory (section 3.2.6.2), for guiding its behaviour: a representation of how its knowledge of past sensor readings and actions should influence its beliefs about future readings, or, in other words, the probability distribution of the latter conditional on the former. This is all the model, within the constraints placed on the form of its representation, attempts to provide.

4.2.1.2. The probability model

Adopting a Gaussian receptive field-based mixing dynamics and confidence region for Q (section 3.3.2.5), the likelihood corresponding to the graphical model (80) is

$$\begin{aligned} p(v,q,r\,|\,\theta,a) &= p(v^0\,|\,\iota)\,p(q^0\,|\,\iota) \\ &\quad \prod_t p(v^{t+1},r^{t+1}\,|\,v^t,q^t,\theta,r^t,a^t)\,p(q^t\,|\,q^{t-1},\theta,r^t,a^t) \\ &= p(v^0\,|\,\iota)\,p(q^0\,|\,\iota) \\ &\quad \prod_t p(y^{t+1}\,|\,x^t,q^t,\theta)\,p(q^t\,|\,q^{t-1},\theta,d^t) \end{aligned} \tag{82}$$

$$\text{where}\quad p(y^{t+1}\,|\,Q^t=i,x^t,\theta) = \left|\frac{\alpha_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(y^{t+1}-\lambda_i x^t\right)'\alpha_i\left(y^{t+1}-\lambda_i x^t\right)\right) \tag{83}$$

$$\text{and}\quad x^t = \begin{pmatrix} v^t \\ r^t \\ a^t \\ 1 \end{pmatrix} \tag{84}$$

$$\text{and}\quad y^t = \begin{pmatrix} v^t \\ r^t \end{pmatrix} \tag{85}$$

$$\begin{aligned} \text{and}\quad p(Q^t=i\,|\,Q^{t-1}=j,\theta,d^t) &= \frac{\omega_{ij}\,g(d^t\,|\,\nu_i,\gamma_i)}{\sum_k \omega_{ik}\,g(d^t\,|\,\nu_k,\gamma_k)} \\ \text{and}\quad g(d^t\,|\,\nu_i,\gamma_i) &= \left|\frac{\gamma_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(d^t-\nu_i\right)'\gamma_i\left(d^t-\nu_i\right)\right) \\ \text{and}\quad d^t &= \begin{pmatrix} r^t \\ a^t \end{pmatrix} \end{aligned} \tag{86}$$

(The initial state estimates $p(v^0\,|\,\iota)$ and $p(q^0\,|\,\iota)$ will be discussed in section 4.2.2.7.)

Note that if the block of λ_i defining the slope of the relationship between V^t and R^{t+1} is zero, the model degenerates into a mixture of static experts[3]; if additionally the block

[3] "autoregressive HMM" in the terminology of Rabiner, *A Tutorial on Hidden Markov Models*

defining the slope of the relationship between D^t and R^{t+1} is zero, then only the bottom "intercept" row remains and the model becomes the Gaussian HMM of section 3.3.3.3. If, on the other hand, the number of components i is reduced to one, then the model is simply a Kalman filter.

4.2.2. A learning algorithm for Samovar

The strategy followed in developing a learning algorithm for the conditional Samovar model can be summarised as follows:

- exploit a nice factorisation of the likelihood to define an "*EEM*" algorithm with two nested expectation-steps
- perform the inner expectation using equations similar to the Kalman-Rauch recursions
- approximate the outer expectation, with the help of the HMM forward-backward equations, by leveraging some known properties of the distribution of Q

4.2.2.1. Nested EM

The reestimation rule for any *EM* algorithm with two unknown variables, such as V and Q, is

$$\theta^{n+1} = \underset{\theta}{\operatorname{argmax}}\ \underbrace{E_{v,q}\big[\ \log p(v,q,r\,|\,a,\theta)\,\big|\,r,a,\theta^n\,\big]}_{U(\theta)}$$

(θ being the parameter to be optimised and the outputs and inputs r, a jointly comprising the known data d of section 3.3.1). The double expectation can be rewritten as a nested pair of expectations:

$$\begin{aligned} U(\theta) &= E_{v,q}\big[\ \log p(v,q,r\,|\,a,\theta)\,\big|\,r,a,\theta^n\,\big] \\ &= \int_{v,q} p(v,q\,|\,r,a,\theta^n)\log p(v,q,r\,|\,a,\theta) \\ &= \int_q p(q\,|\,r,a,\theta^n)\int_v p(v\,|\,q,r,a,\theta^n)\log p(v,q,r\,|\,a,\theta) &\quad (87) \\ &= E_q\Big[E_v\big[\ \log p(v,q,r\,|\,a,\theta)\,\big|\,q,r,a,\theta^n\,\big]\,\Big|\,r,a,\theta^n\Big] &\quad (88) \end{aligned}$$

and the resulting procedure could be called an *EEM* algorithm. In the case of the Samovar model, this rearrangement is helpful, because the likelihood (82) comes ready-factored into almost exactly the terms $p(q\,|\,r,a,\theta)$ and $p(v\,|\,q,r,a,\theta)$ arising in (87) in such a way that the inner, v expectation is continuous, but can be carried out precisely using a version of the Kalman-Rauch recursions, while the outer, q expectation must be approximated, but is at least discrete. The following sections explain how this works, starting with the log likelihood at the heart of (88) and moving outwards.

4.2.2.2. The log likelihood

The likelihood (82) comprises two terms per timestep (plus two for the initial conditions which will be discussed later on). The first term $p(y^{t+1} \,|\, q^t, x^t, \lambda, \alpha)$, arising from the links $X^t \to Y^{t+1}$ and $Q^t \to Y^{t+1}$ in (80), expresses the continuous dynamics of the model, while the corresponding term $p(q^t \,|\, q^{t-1}, \omega, \nu, \gamma, d^t)$ arises from the mixing dynamics links $Q^{t-1} \to Q^t$ and $D^t \to Q^t$. The log likelihood thus falls into two parts, one involving the linear parameters λ, α and the other the mixing parameters ω, ν, γ:

$$\begin{aligned} U(\theta) &= U(\lambda,\alpha) + U(\omega,\nu,\gamma) \qquad (89)\\ \text{where} \quad U(\lambda,\alpha) &= E_q\Big[\sum_t E_v\big[\log p(y^{t+1} \,|\, q^t, x^t, \lambda, \alpha) \,|\, q, d, \theta^n \big] \,\Big|\, d, \theta^n \Big] \\ \text{and} \quad U(\omega,\nu,\gamma) &= E_q\Big[\sum_t E_v\big[\log p(q^t \,|\, q^{t-1}, \omega, \nu, \gamma) \,|\, q, d, \theta^n \big] \,\Big|\, d, \theta^n \Big] \end{aligned}$$

The reestimation rule for λ, α involves only the first part:

$$(\lambda,\alpha)^{n+1} = \operatorname*{argmax}_{\lambda,\alpha} U(\lambda,\alpha)$$

But this breaks down further into separate rules for each component i, because λ_i and α_i feature only in those terms corresponding to timesteps t for which $q^t = i$:

$$(\lambda_i,\alpha_i)^{n+1} = \operatorname*{argmax}_{\lambda_i,\alpha_i} E_q\Big[\sum_{t.q^t=i} E_{v^{t+1},v^t}\big[\log p(y^{t+1} \,|\, Q^t = i, x^t, \lambda_i, \alpha_i) \,|\, q, d, \theta^n \big] \,\Big|\, d, \theta^n \Big] \qquad (90)$$

Comparing (90) with (49), and (83) with (70), the only differences between this update rule and that of the Kalman filter are the appearance of x and y in place of v (section 4.2.1.1), the selective t-summation, and the extra outer expectation over q. The necessary computations are correspondingly similar to those performed for the KF M-step:

$$\lambda_i^{n+1} = P_i B_i^{-1} \qquad (91)$$
$$\alpha_i^{n+1} = \frac{1}{N_i}\left(C_i - \lambda_i^{n+1} P_i'\right) \qquad (92)$$

—*cf.* (72) and (73)—with the sufficient statistics obtained by taking the outer, q-expectation ...

$$\begin{aligned} P_i &= E_q\left[P_{i|q} \,\middle|\, r, a, \theta^n\right] \\ B_i &= E_q\left[B_{i|q} \,\middle|\, r, a, \theta^n\right] \\ C_i &= E_q\left[C_{i|q} \,\middle|\, r, a, \theta^n\right] \qquad (93)\\ N_i &= E_q\left[N_{i|q} \,\middle|\, r, a, \theta^n\right] \\ \text{where} \quad N_{i|q} &= \sum_{t:q^t=i} 1 \end{aligned}$$

... of the statistics (76) used in the KF algorithm, made t-selective and applied to x/y rather than h:

$$\begin{aligned}
B_{i|q} &= \sum_{t:q^t=i} B^{t|q} \\
\text{where}\quad B^{t|q} &= E[\,x^t x^{t\prime} \mid q, r, a\,] \\
&= \bar{x}^t_T \bar{x}^{t\,\prime}_T + \begin{pmatrix} \tilde{v}^t_T & \\ & 0 \end{pmatrix} \\
C_{i|q} &= \sum_{t\ q^t=i} C^{t+1|q} \\
\text{where}\quad C^{t+1|q} &= E[\,y^{t+1} y^{t+1\prime} \mid q, r, a\,] \\
&= \bar{y}^{t+1}_T \bar{y}^{t+1\prime}_T + \begin{pmatrix} \tilde{v}^{t+1}_T & \\ & 0 \end{pmatrix} \\
P_{i|q} &= \sum_{t:q^t=i} P^{t+1,t|q} \\
\text{where}\quad P^{t+1,t|q} &= E[\,y^{t+1} x^{t\prime} \mid q, r, a\,] \\
&= \bar{y}^{t+1}_T \bar{x}^{t\,\prime}_T + \begin{pmatrix} \tilde{v}^{t+1,t}_T & \\ & 0 \end{pmatrix}
\end{aligned}$$

The expectations and variances required are like (71):

$$\begin{aligned}
\bar{x}^t_T &= \begin{pmatrix} \bar{v}^t_T \\ r^t \\ a^t \\ 1 \end{pmatrix} \\
\bar{y}^t_T &= \begin{pmatrix} \bar{v}^t_T \\ r^t \end{pmatrix} \\
\bar{v}^t_\tau &= E[\,v^t \mid q, d^{[0,\tau]}, \theta^n\,] \\
\tilde{v}^t_\tau &= \mathrm{Var}[\,v^t \mid q, d^{[0,\tau]}, \theta^n\,] \\
\tilde{v}^{t,t-1}_\tau &= \mathrm{Cov}[\,v^t, v^{t-1} \mid q, d^{[0,\tau]}, \theta^n\,]
\end{aligned} \tag{94}$$

and they can be calculated in a similar way (see section 4.2.2.3).

Turning to the second, mixing part of the expected log likelihood (89), a reestimation rule is obtained which appears simpler, since the inner, v-expectation is irrevelant to the mixing dynamics (86), which was designed deliberately not to involve v.

$$(\omega,\nu,\gamma)^{n+1} = \operatorname*{argmax}_{\omega,\nu,\gamma} E_q\Big[\sum_t \log p(q^t \mid q^{t-1}, \omega, \nu, \gamma, d^t) \,\Big|\, d, \theta^n\Big]$$

However, this is exactly the update rule (65) for the dynamics of an IOHMM—which, it turned out, had to be implemented using an iterative optimiser.

The M-step for the linear dynamics parameters (equations 91 and 92) and the mixing dynamics parameters (equations 66, 67 and 69) are now in place; it remains to work out how the Q and $V|Q$ distributions on which they depend are to be computed, *i.e.*, how to perform the E-step.

4.2.2.3. *The inner expectation*

The expectation $E_v[\cdots]$ over the linear hidden state in (88) is taken conditional on a fixed value of Q, the sequence of mixing hidden states. And given a particular sequence of linear component choices q^t, the model looks very like a Kalman filter, albeit one with a degenerate output function, and in which a different (known) dynamics function is used at each timestep. Since nothing in the derivation of the Kalman-Rauch recursions (section 3.3.3.5) relies on λ and α remaining constant over time, they can be used with minor modifications to infer the familiar Gaussian distributions—but now conditional on q—for the pairs v^t, v^{t+1}.

In laying out these adapted formulas it is convenient to define a subscripted V to pick out those rows of a vector which correspond to the position of the hidden state v in the overall state x, as defined in (84); R similarly picks out r's rows. Thus, $x^t_V = v^t$ and $x^t_R = r^t$. A subscripted Y is used for the concatenation of v's rows and r's rows—*cf.* (85). A pair of these subscripts defines a subblock of the matrix to which they are applied. Combined with the usual matrix notation, this convention brings out both the relationship with the standard Kalman-Rauch recursions and the low dimensionality of most of the matrix arithmetic involved in the implementation.

The forward recursions begin

$$\bar{v}^0_{-1} = \bar{\iota}_q$$
$$\tilde{v}^0_{-1} = \tilde{\iota}_q$$

—separate estimates of the initial linear state being made for each q (see section 4.2.2.7)—and continue

$$\begin{aligned}
\bar{y}^t_{t-1} &= \lambda_{q^{t-1}} \begin{pmatrix} \bar{v}^{t-1}_{t-1} \\ r^{t-1} \\ a^{t-1} \\ 1 \end{pmatrix} \\
\tilde{y}^t_{t-1} &= \left(\lambda_{q^{t-1}}\right)_{YH} \tilde{v}^{t-1} \left(\lambda_{q^{t-1}}\right)'_{HY} + \alpha^{-1}_{q^{t-1}} \\
K^t &= \left(\tilde{y}^t_{t-1}\right)_{HR} \left(\left(\tilde{y}^t_{t-1}\right)_{RR}\right)^{-1} \\
\bar{v}^t_t &= \left(\bar{y}^t_t\right)_V + K^t(r^t - \left(\bar{y}^t_{t-1}\right)_R) \\
\tilde{v}^t_t &= \left(\tilde{y}^t_{t-1}\right)_{HH} - K^t \left(\tilde{y}^t_{t-1}\right)_{RH}
\end{aligned}$$

The backward iterations begin

$$\tilde{v}^{T,T-1}_T = \left(\left(\lambda_{q^{T-1}}\right)_{HH} - K^T \left(\lambda_{q^{T-1}}\right)_{RH}\right) \left(\tilde{y}^T_{T-1}\right)_{HH}$$

and continue

$$\begin{aligned} J^t &= \tilde{v}_t^t \left(\alpha'_{q^t}\right)_{HY} \left(\bar{y}_t^{t+1}\right)^{-1} \\ \bar{v}_T^t &= \bar{v}_t^t + J^t \left(\begin{pmatrix} \bar{v}_{t+1}^{t+1} \\ r^{t+1} \end{pmatrix} - \bar{y}_t^{t+1} \right) \end{aligned} \tag{95}$$

$$\tilde{v}_T^t = \tilde{v}_t^t + J^{t\prime} \left(\begin{pmatrix} \tilde{v}_{t+1}^{t+1} & \\ & 0 \end{pmatrix} - \tilde{y}_t^{t+1} \right) J^t \tag{96}$$

$$\tilde{v}_T^{t+1,t} = \left(J^{t+1} + I\right) \left(\begin{pmatrix} \tilde{v}_T^{t+2,t+1} \\ 0 \end{pmatrix} - \left(\lambda_{q^{t+1}}\right)_{YH} \tilde{v}_{t+1}^{t+1} \right) J^{t+1\prime} \tag{97}$$

4.2.2.4. The outer expectation

In theory, the outer expectation over q—which carries over from (88) into (93) and, in a slightly different form, into (68)—presents no great difficulty. The sequence-probabilities $p(q \mid r, a, \theta)$ by which the expectations are weighted can be obtained by multiplying the transition probabilities involved in the discrete evolution of q and the expected observation likelihoods conditioned on q

$$\begin{aligned} p(q \mid \theta, r, a) &\propto p(r \mid q, a, \theta)\, p(q \mid a, \theta) \\ &= \prod_t E_{v^t}\left[p(r^{t+1} \mid v^t, r^t, a^t, q^t, \theta) \mid q, \theta, r^{[1,t]}, a^{[1,t]}\right] \\ &\quad \prod_t p(q^{t+1} \mid q^t, \theta, r^t, a^t) \end{aligned} \tag{98}$$

Taking the expectation in the first term is easy, since the distribution $p(v^t \mid q, r^{[1,t]}, a^{[1,t]}, \theta)$ is computed as an auxiliary value by the modified Kalman-Rauch iterations (95)–(97):

$$E_{v^t}\left[p(r^{t+1} \mid v^t, q^t, r^t, a^t) \mid q, r, a\right] = \left|\frac{\phi}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^{t+1} - \lambda_i \bar{x}^t\right)' \phi \left(r^{t+1} - \lambda_i \bar{x}^t\right)\right) \tag{99}$$

$$\begin{aligned} \text{where} \quad & \phi^{-1} = (\lambda_i)_{RH}\, \tilde{v}_t^t\, (\lambda'_i)_{HR} + \alpha_i^{-1} \\ \text{and} \quad & i = q^t \\ \text{and} \quad & \bar{x}^t = \begin{pmatrix} \bar{v}_t^t \\ r^t \\ a^t \\ 1 \end{pmatrix} \end{aligned}$$

The second term is just the gating rule (86).

4.2.2.5. Approximating the outer expectation

However, it is in practice impossible to perform a summation over all the possible sequences q, because their number rises exponentially with their length T. Luckily, this q-space into which the intractability of the model has been concentrated is at least discrete;

and it is known to have some properties which can guide a greedy search towards its important regions:

1) *Sparseness.* Nearly all of the possible sequences will be of negligible probability, and will therefore contribute nothing to the expectation.

2) *Local consistency.* The most probable sequences are likely to be made up of locally near-optimal subsequences. If a section is taken out of a good assignment sequence, and considered in isolation, under minimal assumptions about what happens before and after it, it is hard to imagine that it will not fit the data at least tolerably well. Indeed, local consistency can be seen as a requirement placed on the kind of models which will be considered as acceptable.

3) *Finiteness of horizon.* Although the model is capable in principle of capturing the dynamics of a world in which events have consequences over a long timescale, there are good theoretical and empirical grounds to believe that such effects are unlikely either to arise in the target domain or, if they do, to be learned reliably by any timestep-based algorithm. So little will be lost if contiguous subsequences of the training data are considered in isolation, as long as they are not too short.

Put another way, it will normally be possible to tell with some certainty which component was active at a given timestep by considering the sensor readings and actions in its near temporal neighbourhood: most sub-sequences can be seen to be unlikely on the basis of local evidence. Knowledge of what happened further into the future/past may be needed to squeeze out all the ambiguity. But already the great bulk of the space of sequences can be pruned away, leaving one or more distinct (and narrow) islands of plausible hypotheses. (In fact, the learner will wish to reject models relative to which this assumption does not hold, so any bias introduced by it is benign.)

These considerations point towards a kind of heuristic dynamic programming procedure for generating a pragmatically adequate (though not statistically valid) "sample" of probable sequences. To begin with, short subsequences are evaluated separately from each other, using (98) restricted to the range covered by each; all but the most likely ones are rejected; and candidate sequences are made up by joining pairs of temporally adjacent survivors. Those longer sequences are in turn evaluated, pruned and joined, and so on until the horizon length of item 3 is reached. The number of candidate sequences in play is relatively large to start with—but they are short, and hence cheap to evaluate; as their length grows, their numbers fall. Item 1 says that the restriction on the number of full-length sequences remaining at the end does not necessarily weaken the approximation disastrously, while item 2 says that the early pruning decisions based on local considerations will mostly be correct.

In fact, it is possible to do better by selecting subsequences for joining on the basis not only of their respective local likelihoods, but also of the plausibility with which they fit together. Part of this inter-subsequence consistency can be assessed immediately by looking at the probability with which the gating rule (86) would in fact follow the final component choice in the first subsequence with the initial choice in the second. Since that probability is conditioned only on the known data r^t, a^t, and does not depend on the linear hidden state v^t, there is no difficulty in evaluating it as part of the outer expectation. Then the algorithm has at its disposal both local estimates of the likelihood of each subsequence, and estimates of the transition probabilities between them—so it can use the HMM forward-backward equations (56) and (57) to convert those local estimates into a globally informed estimate of the probability of each possible pair-up between subsequences. The table below shows the correspondence between the equations' original form in the HMM framework of section 3.3.3.3 and the way they are used here:

HMM value	*Symbol*	*Sequence-joining value*	*Symbol*
timestep	t	range of timesteps	u
HMM state	$Q^t = i$ or j	subsequence candidate	$S^u = l$ or m
state likelihood	$p(r^t \mid Q^t = j, \rho_j)$	cand. local likelihood	c_l^u
state transition prob.	ω_{ij}	cand. follow-on prob. est.	z_{lm}^u
global state pair prob.	ξ_{ij}^t	global cand. pair prob. est.	ζ_{lm}^u

The equations are iterated not over individual timesteps t, but over ranges u of timesteps of a given length L, so that subsection u covers the timesteps $[Lu, Lu + L)$. In place of the HMM states Q^t, the random variables of interest are the subsequences S^u, which denote the Samovar component choices within each range u:

$$\begin{aligned} S^{u,0} &= Q^{Lu} \\ S^{u,1} &= Q^{Lu+1} \\ &\dots \\ S^{u,L-2} &= Q^{L(u+1)-2} \\ S^{u,L-1} &= Q^{L(u+1)-1} \end{aligned} \tag{100}$$

Where the HMM states at each timestep have a likelihood $p(r^t \mid \rho_j)$ at which they predict each output, the component choice subsequences have a "local likelihood"

$$c_l^u = p(r^{[Lu, Lu+L)}, S^u = l \mid a^{[Lu,Lu+L)}, \theta) \tag{101}$$

that they describe correctly the outputs and component choices within their range, under no assumptions about what happens before and after them, obtained from running (98) with t ranging over $[Lu, Lu + L)$. The place of the state transition probabilities

$$\omega_{ij} = p(Q^t = j \mid Q^{t-1} = i, \theta)$$

is taken by estimates of the probabilities with which temporally adjacent subsequences look like they might have followed on from each other:

$$\begin{aligned} p(\text{transition from } S^{u-1} = l \text{ to } S^u = m) &\approx z^u_{lm} \\ &= p(Q^{Lu} = m^0 \mid Q^{Lu-1} = l^{L-1}, \theta, r^{Lu}, a^{Lu}) \end{aligned} \quad (102)$$

which is (86) with $t = Lu, i = m^0, j = l^{L-1}$. Instead of the output

$$\xi^t_{ij} = p(Q^t = j, Q^{t+1} = i, r \mid \theta)$$

—*cf.* (58)—the forward-backward equations yield

$$\begin{aligned} \zeta^u_{lm} &\approx p(S^u = m, S^{u+1} = l, r \mid \theta, a) \\ &\propto p(S^u = m, S^{u+1} = l \mid \theta, r, a) \end{aligned} \quad (103)$$

This is the Bayesianly correct estimate of the probability that the component choices in the range $[Lu, Lu + 2L)$ are as described by l, m, given of course the restriction that the distribution of the linear hidden state v is computed on the basis of local evidence inside each L-long subsequence, and is ignored in estimating the follow-on probabilities.

If the linear hidden state is actually irrelevant—if the blocks of the dynamics matrices λ_i which define the coefficients through which V^t affects Y^t are zero—then the first, $L = 1$ pass of the algorithm reduces, as it should, to the HMM's forward-backward computation. Subsequent passes have the effect of selecting a small, but (in this case) correctly distributed, sample of sequences. On the other hand, if the model comprises only a single component, then the algorithm reduces to the KF's Kalman-Rauch recursions.

In other, non-degenerate cases, it is clear that the algorithm will not in general produce an unbiased sample of mixing state sequences. The number of candidate subsequences it maintains at each stage must necessarily be limited in order to contain its consumption of resources, and this means that it will often have to reject large classes of possibilities on a heuristic basis at a relatively early stage. The issues of how the bias thereby introduced into the sample might be corrected, and indeed whether it matters much, are addressed in section 6.1.1.2.

4.2.2.6. V-matching

In principle, the candidate follow-on probability estimates (102) can be tightened up by taking into account the implications which the subsequences s^{u-1} and s^u each have for the hidden linear state V^{Lu}. The terminal $\bar{v}^L, \tilde{v}^L$ produced by the modified Kalman-Rauch recursions running inside s^{u-1}, and the initial $\bar{v}^0, \tilde{v}^0$ produced by those running inside s^u, are both estimates of this same quantity, so that the agreement between the two provides additional information about how well the sequences fit together. Turning the overlap into a probability is, however, not straightforward; the estimate would have to be calculated as

$$\begin{aligned} p(S^u = m \mid S^{u-1} = l) &\approx \int_{q^{Lu}, v^{Lu}} p(S^u = m \mid q^{Lu}, v^{Lu})\, p(q^{Lu}, v^{Lu} \mid S^{u-1} = l) \\ &= \int_{v^{Lu}} p(S^u = m \mid v^{Lu})\, p(v^{Lu} \mid S^{u-1} = l)\, p(Q^{Lu} = m^0 \mid Q^{Lu-1} = l^{L-1}, \theta) \end{aligned}$$

where the term

$$p(S^u = m \mid v^{Lu}) = \frac{p(v^{Lu} \mid S^u = m)}{\sum_m p(v^{Lu} \mid S^u = m)}$$

gives rise to a normalisation inside the integral, the resulting expression being not a Gaussian convolution but something far less easy to evaluate. In practice, however, all that is required is a comparison which causes really improbable couplings to be rejected. That can be achieved by considering the convolution of the two versions, or their cross-entropy.

4.2.2.7. Initial conditions

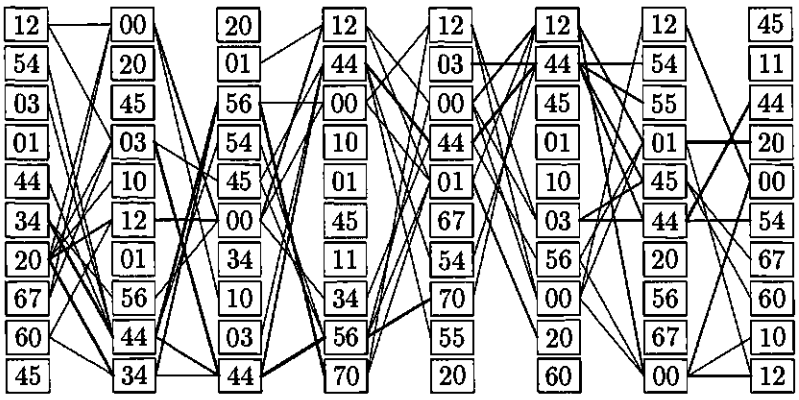
What, finally, of (82)'s initial conditions $p(v^0 \mid \iota)$ and $p(q^0 \mid \iota)$, discussion of which has so far been deferred? Since it only really makes sense to apply the Samovar algorithm to longish training sequences, the effect of the latter will be negligible, and it is not implemented (although it easily could be). The former, however, plays a more important role, since the subsequences over which the modified Kalman-Rauch recursions (section 4.2.2.3) are run are, by design, of at most moderate length. Each subsequence l is given its own initial linear hidden state estimate $\bar{\iota}_l, \tilde{\iota}_l$, so that the recursions can be run more than once, each time tightening $\bar{\iota}_l, \tilde{\iota}_l$ up, and therefore also the estimate of the subsequent V^ts.

4.2.2.8. Visualising the algorithm

The diagrams following are visualisations of the process by which short Q-subsequences are evaluated and joined to produce longer ones. They were obtained by running the algorithm on the test data of section 5.1.2. V-matching (section 4.2.2.6) was disabled, and the number of subsequence candidates maintained by the algorithm as possibilities for each section (*candsMax* of section 4.2.2.9) was limited to 10 to save space on the page; both these settings are suboptimal for the problem, but the fact that the algorithm takes longer to settle on the right answer actually makes its workings clearer.

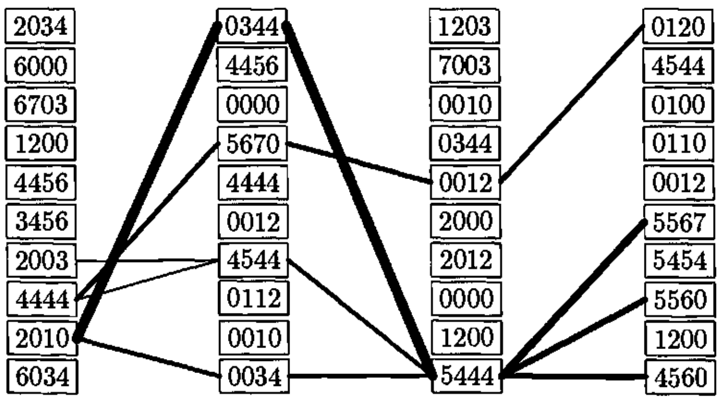
We join the algorithm when it is evaluating the candidates for the eight two-step subsections of a sixteen-step data series; the list of candidates s^u for each section u is represented by a column of boxed pairs of mixing state numbers $s^{u,t}$, the first column/section covering timesteps 0 and 1, the second timesteps 2 and 3, and so on.

The candidate subsequences are listed vertically in order of estimated probability, as determined by equation 103 during the previous round of the algorithm. Thus the box "20" at the top of the third column represents the working hypothesis that the most probable mixing states at timesteps 4 and 5 were numbers 2 and 0.

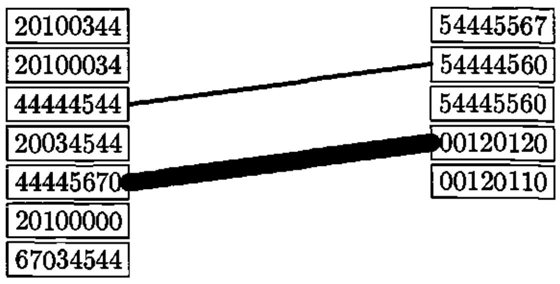


The black lines joining adjacent candidates represent pairings which look globally plausible on the basis of the candidates' local likelihoods and transition probabilities: the thickness of the line between candidate l in section u and candidate m in section $u+1$ is proportional to ζ^u_{lm} (equation 103), except that the lines indicating ζs below a cutoff threshold are omitted for clarity. At this stage, there are quite a few such lines present, indicating that many pairings have significant probability—the two-step sequences are short enough that almost any can be made consistent with the observable outputs by postulating an appropriate hidden linear state V.

Next, the algorithm takes a sample of the most probable-looking pairings, and obtains four-step candidates for the sections 0–3, 4–7, 8–11 and 12–15. At this stage, the picture is becoming clearer, with fewer serious pairing possibilities. Note the existence of an independent trajectory 4444, 5670, 0012, 0120 alongside a more ramified group of alternatives.



By the time the algorithm turns to considering the consequent eight-step subsequences, there are only two serious pairings to consider.



And these are the ones which are returned as hypotheses about the whole sixteen-step sequence:

$\boxed{4444454454444560}$
$\boxed{4444567000120120}$

4.2.2.9. The algorithm in pseudocode

In the following pseudocode summary of the whole conditional Samovar learning algorithm, maths italic letters such as r and α are used in the same way as in the text; other program variables are written in *slanted sans serif* characters, and control words in upright sans serif characters.

```
let learn $r, a$ =
```

- Start with an initial parameter containing a number of components, all set to the same default values. (The symmetry will be broken by the random element in the E-step.)

```
for $i \in [0, componentsNum)$
    $\lambda_i \leftarrow 0$
    $\alpha_i \leftarrow I$
    $\nu_i \leftarrow 0$
    $\gamma_i \leftarrow I$
```

- E-step: get a sample of component choice subsequences from the dynamic programming-style subsequence joining algorithm (see below).

```
let subsections = sequenceSample $r, a, \theta$
```

- Compute the sufficient statistics (93) for the linear dynamics parameter updates, averaged over all the subsequences found for each L-long range u.

```
for $i \in [0, |\theta|)$
  $B_i \leftarrow 0; C_i \leftarrow 0; P_i \leftarrow 0; N_i \leftarrow 0$
for $u \in [0, |subsections|)$
  for $seq \in subsections^u.seqs$
    for $t \in [0, seq.q)$
      let $i = seq.q^t$
      let $x = \begin{pmatrix} seq.\bar{v}^t \\ r^{Lu+t} \\ a^{Lu+t} \\ 1 \end{pmatrix}$
      let $y = \begin{pmatrix} seq.\bar{v}^{t+1} \\ r^{Lu+t+1} \end{pmatrix}$
      $B_i \overset{+}{\leftarrow} seq.prob \times \left( xx' + \begin{pmatrix} seq.\tilde{v}^t & \\ & 0 \end{pmatrix} \right)$
      $C_i \overset{+}{\leftarrow} seq.prob \times \left( yy' + \begin{pmatrix} seq.\tilde{v}^{t+1} & \\ & 0 \end{pmatrix} \right)$
      $P_i \overset{+}{\leftarrow} seq.prob \times \left( yx' + \begin{pmatrix} seq.\tilde{v}^{t+1,t} & \\ & 0 \end{pmatrix} \right)$
      $N_i \overset{+}{\leftarrow} seq.prob$
```

- M-step: update the linear dynamics parameters from their sufficient statistics (see equations 91 and 92), and invoke a root finder on the derivative of the expected log likelihood with respect to the mixing dynamics parameters.

```
for $i \in [0, |\theta|)$
  $\lambda_i \leftarrow P_i B_i^{-1}$
  $\alpha_i \leftarrow \left( \frac{C_i - \lambda_i P_i'}{N_i} \right)^{-1}$
$\omega, \nu, \gamma \leftarrow scgRoot\ (\omega, \nu, \gamma \rightarrow mixingErrorDeriv\ \omega, \nu, \gamma, sequences)$
```

scgRoot is a standard nonlinear root-finder, such as a scaled conjugate gradients routine[4]. The pseudocode for the function *mixingErrorDeriv* passed to it as an argument is given below; but first, the dynamic programming-style procedure described in section 4.2.2.4 for performing the approximate E-step:

```
let $sequenceSample\ r, a, \theta =$
```

- Start with every possible sub-sequence of length one.

```
$L \leftarrow 1$
for $u \in [0, T)$
  for $i \in [0, |\theta|)$
```

[4] Moller, *A scaled conjugate gradient algorithm*

```
        $subsections^u.seqs_i.q \leftarrow \langle i \rangle$
```

- Run the modified Kalman-Rauch recursions "inside" each candidate subsequence one or more times, to compute the distribution of v (equations 95–97) over the range covered by it, under no assumptions about what happens before and after it.

```
    join:
    for $u \in [0, |subsections|)$
        for $cand \in subsections^u.seqs$
            $\bar{\imath}, \tilde{\imath} \leftarrow 0, I$
            repeat a few times    (see section 4.2.2.7)
                $cand.v, \bar{\imath}, \tilde{\imath} \leftarrow$ kalmanRauch from $Lu$ to $L(u+1)-1$ given $cand.q, r, \bar{\imath}, \tilde{\imath}, \theta$
```

- Compute each candidate sequence's local likelihood.

```
    for $u \in [0, |subsections|)$
        for $l \in [0, |subsections^u.seqs|)$
            $c_l^u \leftarrow 1$
            let $cand = subsections^u.seqs_l$
            for $t \in [0, L)$
                let $i = cand.q^t$
                let $\phi = \left((\lambda_i)_{RH}\, cand.\tilde{v}^t\, (\lambda_i')_{HR} + \alpha_i^{-1}\right)^{-1}$
                let $x = \begin{pmatrix} cand.\bar{v}^t \\ r^{Lu+t} \\ a^{Lu+t} \\ 1 \end{pmatrix}$
                $c_l^u \overset{\times}{\leftarrow} \left|\frac{\phi}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^{Lu+t+1} - \lambda_i x\right)' \phi \left(r^{Lu+t+1} - \lambda_i x\right)\right)$
```

- Estimate the probability with which each pair of temporally adjacent candidates fits together.

```
    for $u \in [0, |subsections| - 1)$
        for $l \in [0, |subsections^u.seqs|)$
            for $m \in [0, |subsections^u.seqs|)$
                let $candPrev = |subsections^u.seqs|_l$
                let $j = candPrev.q^{L-1}$
                let $candNext = |subsections^u.seqs|_m$
                let $i = candNext.q^0$
                $z_{lm}^u \leftarrow \omega_{ij} \left|\frac{\gamma_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(d^{Lu} - \nu_i\right)' \gamma_i \left(d^{Lu} - \nu_i\right)\right)$
        for $m \in [\,0, |\mathit{subsections}^u.\mathit{seqs}|\,)$
          $z^u_{lm} \overset{\div}{\leftarrow} \sum_m z^u_{lm}$
```

- Run the HMM forward-backward equations to obtain global estimates of the probability of each pair of temporally adjacent candidates.

```
    let $\zeta = \mathit{forwardBackward}$ from 0 to $|\mathit{subsections}| - 1$ given $c, z$
    for $u \in [\,0, |\mathit{subsections}| - 1\,)$
      for $l \in [\,0, |\mathit{subsections}^u.\mathit{seqs}|\,)$
        for $m \in [\,0, |\mathit{subsections}^u.\mathit{seqs}|\,)$
          $\zeta^u_{lm} \overset{\div}{\leftarrow} \sum_{l,m} \zeta^u_{lm}$
```

- If a sample of long-enough sequences has been obtained, return them along with their estimated probabilities.

```
    if $L \geq$ sensible length
      for $u \in [\,0, |\mathit{subsections}| - 1\,)$
        for $l \in [\,0, |\mathit{subsections}^u.\mathit{seqs}|\,)$
          $\mathit{subsections}^u.\mathit{seqs}_l.\mathit{prob} \leftarrow \sum_m \zeta^u_{lm}$
      return subsections
```

- Otherwise, select up to *candsMax* (depending on available time/storage) of the most probable-seeming pairs, discarding the rest.

```
    let $\mathit{subsectionsPaired} = ()$
    for $u \in [\,0, |\mathit{subsections}| \div 2\,)$
      let $\mathit{candsPrev} = \mathit{subsections}^{2u}.\mathit{seqs}$
      let $\mathit{candsNext} = \mathit{subsections}^{2u+1}.\mathit{seqs}$
      $\mathit{subsectionsPaired}^u.\mathit{seqs}$ =
  { $\mathit{candsPrev}_l.q$ @ $\mathit{candsNext}_m.q$ :
    $l, m \in$ sample of size at most candsMax with $p(l,m) = \zeta^{2u}_{lm}$ }
```

- Repeat.

```
    $\mathit{subsections} \leftarrow \mathit{subsectionsPaired}$
    $L \overset{\times}{\leftarrow} 2$
    goto join
```

The output of this algorithm is, for each subsection of the timesteps $[0,T)$, a sample of likely mixing states sequences covering it. *kalmanRauch* is the routine implementing the modified Kalman-Rauch recursions of section 4.2.2.3, and *forwardBackward* the standard HMM forward-backward algorithm of section 3.3.3.3.

Finally, the derivative whose root is found to reestimate the mixing dynamics parameters:

```
let mixingErrorDeriv $\omega, \nu, \gamma$, sections =
```

- Initialise the derivatives.

```
for $i \in [0, |\theta|)$
  for $j \in [0, |\theta|)$
    $dByWeights_{ij} \leftarrow 0$
  $dByCentres_i \leftarrow 0$
  $dBySizes_i \leftarrow 0$
```

- Sum over time.

```
for $t \in [1, T)$
```

  - Find how probable the old parameter estimates made each pair of mixing states $Q^t = i, Q^{t-1} = j$.

```
let $u = \frac{t}{L}$
for seq $\in sections^u$.seqs
  let $i = seq.q^{t \bmod L}$
  let $j = seq.q^{t-1 \bmod L}$
  $pOld_{ij} \overset{+}{\leftarrow} seq.prob$
```

  - Find how likely the new parameter estimates make them.

```
for $i \in [0, |\theta|)$
  for $j \in [0, |\theta|)$
    $pNew_{ij} \leftarrow \omega_{ij} \left|\frac{\gamma_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(d^t - \nu_i)' \gamma_i (d^t - \nu_i)\right)$
  let $norm = \sum_j pNew_{ij}$
  for $j \in [0, |\theta|)$
    $pNew_{ij} \overset{\div}{\leftarrow} norm$
```

  - Compute the derivative contributions and add them in (see equations 66, 67 and 69).

```
for $i \in [0, |\theta|)$
  for $j \in [0, |\theta|)$
    let $\varepsilon^t_{ij} = \mathit{pOld}_{ij} - \mathit{pOld}_j \mathit{pNew}_{ij}$
    $\mathit{dByWeights}_{ij} \overset{+}{\leftarrow} \frac{\varepsilon^t_{\theta_{ij}}}{\omega_{ij}}$
  let $\varepsilon^t_i = \sum_j \varepsilon^t_{ij}$
  $\mathit{dByCentres}_i \overset{+}{\leftarrow} \varepsilon^t_{\theta_i} \gamma_i(\nu_i - d^t)$
  $\mathit{dBySizes}_i \overset{+}{\leftarrow} \varepsilon^t_{\theta_i} \left(\gamma_i^{-1} - (d^t - \nu_i)(d^t - \nu_i)'\right)$
```

- Return the overall derivative.

```
return $\mathit{dByWeights}$ @ $\mathit{dByCentres}$ @ $\mathit{dBySizes}$
```

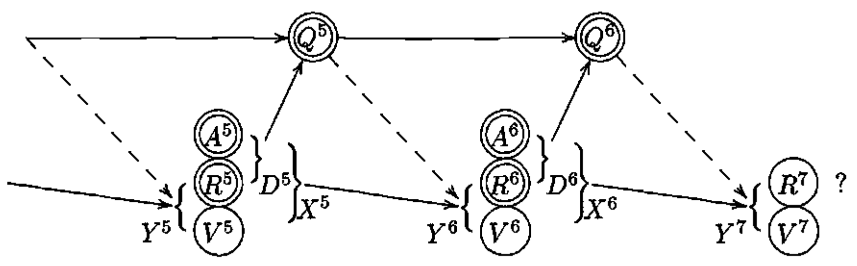
In practice, of course, the ordering of the algorithm and its storage requirements can be optimised, defensive measures need to be taken against numerical loss of significance (*e.g.* working in log space), and various boundary cases need to be taken into account.

4.2.2.10. Making predictions

The predictive distribution of the conditional Samovar model is[4]

$$\begin{aligned} p(r^T \mid \theta, d^{[0,T)}) &= E_{v,q}\left[p(r^T \mid v^{T-1}, q^{T-1}, \theta, d^{T-1}) \mid d, \theta\right] \\ &= \sum_q p(q \mid \theta, d^{[0,T)}) \int_{v^{T-1}} p(v^{T-1} \mid q, d^{[0,T)})\, p(r^T \mid v^{T-1}, q^{T-1}, \theta, d^{T-1}) \end{aligned}$$

Conditioned on each sequence q, and supposing without loss of generality that $T = 7$, the situation is as follows:



[4] Note the slight difference in the form of this expression as compared with that of the general rule (54) for the predictive distribution of a Bayesian dynamical systems model: it is a consequence of the switch from the usual graphical model (79) to the (80), in which the readings and actions have been brought into the world state.

Each sequence q gives rise to a Gaussian component in the overall mixture prediction, obtained by applying the dynamics function of q's terminal component q^6 to $X^6 \,|\, q$, and projecting the resultant distribution over Y^7 down to R^7 alone:

$$p(r^T \,|\, \theta, d^{[0,T)}) = \sum_q p(q \,|\, \theta, d^{[0,T)}) \left| \frac{\phi_i}{2\pi} \right|^{\frac{1}{2}} \exp\left(-\frac{1}{2} \left(r^T - (\lambda_i)_{RX}\, \bar{x}_q^{T-1}\right)' \phi_i \left(r^T - (\lambda_i)_{RX}\, \bar{x}_q^{T-1}\right) \right)$$

$$\text{where} \quad \phi_i = \left((\gamma_i^{-1})_{RR} + (\lambda_i)_{RV}\, \bar{v}_q^{T-1}\, (\lambda_i')_{VR} \right)^{-1}$$

$$\text{and} \quad i = q^{T-1}$$

$$\text{and} \quad \bar{x}_q^{T-1} = \begin{pmatrix} \bar{v}_q^{T-1} \\ r^{T-1} \\ a^{T-1} \\ 1 \end{pmatrix}$$

The expectation over q can be approximated by invoking the subsequence-joining algorithm of (4.2.2.5) to produce candidates l for the mixing state sequence over some reasonable time period leading up to $T - 2 = 5$ (inclusive—Q^5 determines the component used to generate r^6, the last known sensor reading). Part of its output will be estimates $\bar{v}_l^6, \bar{v}_l^6$ of what the linear hidden state implicated in the generation of R^7 would be, conditional on each of the sample ls. It remains only to compute the probability of each possible continuation to Q^7 of each of the sample sequences, conditional on the last known gating data d^6, and what each Q^7 would do with $X^7 \,|\, l$ to produce its output R^8. If desired, a confidence region along the lines of section 3.3.2.6 can be included with very little extra overhead.

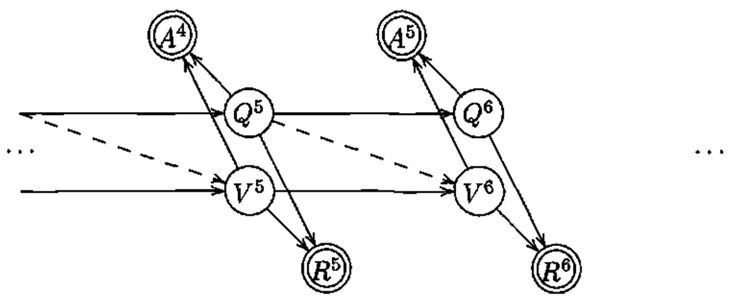
This simple procedure could obviously be optimised if a whole series of predictions is required (*e.g.* for a robot in a sense-think-act cycle): the set of subsequence candidates for the section [0,3] does not depend on the readings and actions from timestep 4 onwards, so there is no point in computing it for the timestep 4 prediction, only to forget it and recompute it at timesteps 5, 6, and 7—it could be cached. For instance, at timestep 7, the subsequence-joining algorithm can be started using cached candidate sets for the sections [0,3], [4,5] and the singleton [6,6].

4.2.3. The joint Samovar model

Just as the joint mixture of experts described in section 3.3.2.4 is closely related to the conditional mixture of experts of section 3.3.2.5, so the conditional Samovar model of section 4.2.1 has a "joint" counterpart with the broader aim of estimating the distribution of the robot's actions (process inputs) A^t, as they were seen in the training data, as well as that of its sensor readings (process outputs) R^t. As well as furnishing the model with an arguably more meaningful confidence region, the shift of perspective turns out to have other computational and semantic spin-offs.

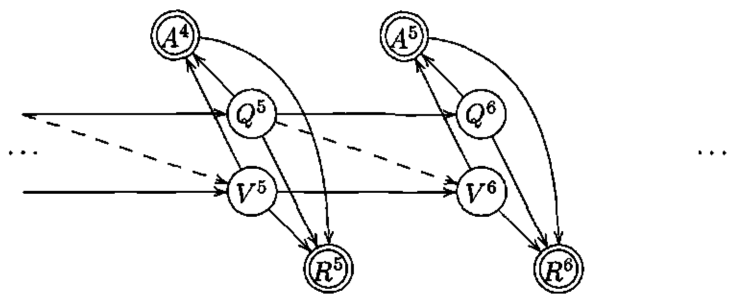
4.2.3.1. *Merging inputs and outputs*

One possible variation on this scheme would be to treat A^{t-1} and R^t symmetrically, as independent "outputs" with separate mixed linear/Gaussian distributions conditional on Q^t and V^t:

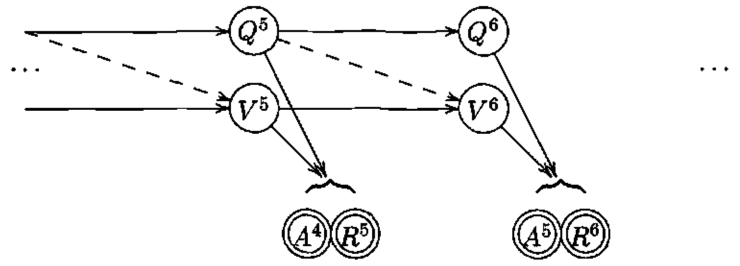


(104)

Compare (104) with (79): the only difference is that the sense of the links $A^{t-1} \to Q^t$ and $A^{t-1} \to V^t$ has been reversed. In fact, there is some advantage to be gained from making R^t depend also linearly on A^{t-1}:



—or, equivalently (*cf.* section 3.2.5.4), handling A^{t-1} and R^t not only symmetrically, but actually as subvectors of the same quantity:



(105)

Here A^{t-1} and R^t are shown as jointly conditional on V^t and Q^t—through a linear (input-)output mapping chosen by the mixing state

$$\begin{pmatrix} A^{t-1} \\ R^t \end{pmatrix} = \kappa_i V^t + N(0, \beta_i) \quad \text{where} \quad i = Q^t$$

—rather than V^t and Q^t being conditional on A^{t-1}, as in the conditional model. Since this arrangement is both simpler and more general than (104), it will be adopted henceforth as "the joint Samovar model". As usual the mixed-linear dynamics are

$$V^{t+1} = \lambda_i \begin{pmatrix} V^t \\ 1 \end{pmatrix} + N(0, \alpha_i)$$

(note the extra constant element appended to each V^t in order to support a nonzero intercept), while the mixing dynamics are discretely Markovian, according to a transition matrix ω. The joint Samovar model's likelihood is therefore

$$\begin{aligned} p(v,q,r,a\,|\,\theta) &= p(v^0\,|\,\iota)\,p(q^0\,|\,\iota) \\ &\quad \prod_t p(a^{t-1},r^t\,|\,v^t,q^t,\theta)\,p(v^t\,|\,v^{t-1},q^{t-1},\theta)\,p(q^t\,|\,q^{t-1},\theta) \end{aligned}$$

$$\text{where}\quad p(a^{t-1},r^t\,|\,v^t,Q^t=j,\theta) = \left|\frac{\beta_j}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(d^{t,t-1}-\kappa_j\begin{pmatrix}v^t\\1\end{pmatrix}\right)'\beta_j\left(d^{t,t-1}-\kappa_j\begin{pmatrix}v^t\\1\end{pmatrix}\right)\right)$$

$$\text{and}\quad d^{t,t-1} = \begin{pmatrix}r^t\\a^{t-1}\end{pmatrix}$$

$$\text{and}\quad p(v^t\,|\,v^{t-1},Q^{t-1}=i,\theta) = \left|\frac{\alpha_i}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(v^t-\lambda_i\begin{pmatrix}v^{t-1}\\1\end{pmatrix}\right)'\alpha_i\left(v^t-\lambda_i\begin{pmatrix}v^{t-1}\\1\end{pmatrix}\right)\right)$$

$$\text{and}\quad p(Q^t=j\,|\,Q^{t-1}=i) = \omega_{ij}$$

4.2.3.2. Semantics

The role of the joint model's hidden state Q^t, V^t in apparently *determining* the action A^{t-1} may perhaps require some explanation: are the actions not, then, generated by the robot, or even by some human agency? They are; but that does not mean that it is not possible or useful to approximate their distribution and draw inferences from it. Indeed, a fully responsible approach to the problem of model trust must surely take this distribution into account: if the model is asked to predict the consequences of an action which is clearly different from those it has previously had a chance to observe, or to observe in a situation similar to the current one, it is reasonable to expect it to express a suitable degree of caution in its answer. Certainly, the mixed linear/Gaussian generative model may be something of a travesty of the process by which the actions are really selected; but the same is after all true of the outputs—the justification of any finite and/or imperfectly optimised environment model must rest on an appeal to the approximative principle discussed in section 3.2.4.3. And the important thing from a Bayesian point of view is that it is a reasonable way of generalising from the training data to a density expressing what future actions (taken in each situation) are expected to be like. The semantics of the joint model could be glossed thus: "The processes responsible for generating the actions and readings in the training data behave differently in different situations. In each situation, we expect that the action which will be performed is like *this* and the corresponding reading like *that*. Unexpected actions (or readings) are evidence that some previously unseen process is at work."

One particularly interesting feature of the joint model is that the input-generating "noise" distribution associated with each component (represented by a block of β_i^{-1}) can be very tight, and was in fact observed to be so in practice during the simulated robot experiments reported in section 5.4. The reason for this is that the action A^{t-1} is modelled conditionally on V^t as well as Q^t, so that if the effect of the action on the linear hidden state is sufficiently deterministic, it may be possible to infer what the action must have been with some precision—relative to a candidate mixing state sequence and linear hidden state. And if each Q^t-choice in a candidate sequence has quite specific implications for A^{t-1} in the light of what is known about V^t, the observed a^{t-1} may provide strong evidence for distinguishing between them. What is especially significant is that the estimate of V^t can play a role in the model's judgement as to which mixing state was active, by modulating the corresponding A^{t-1}: in other words, the model can effectively perform "gating" on V^t. In section 4.2.1, it was noted that this was beyond the capability of the conditional Samovar model for reasons of tractability.

On the other hand, there is no possibility of folding the known data R^t, A^t directly into the continuous world state, as suggested in section 4.2.1.1 for the conditional model, and allowing the model direct access to linear relationships between successive actions and readings: in the joint model, all distributions must be mediated through the hidden linear state V. This may make it less trivial for the model to get a handle on the most robust and obvious phenomena in its environment. (Of course, if the dimensionality of R^t, A^t is large, mediation via a projection onto a lower-dimensional V will always be preferable if only for reasons computational tractability.[5])

Potentially the most important difference between the two models will become clear when the job for which they are ultimately intended—recommending good actions to the robot—is discussed. It will turn out to be very much easier in principle to obtain actions from the joint model (section 6.2).

4.2.3.3. Learning algorithm

As usual, the joint model including a generative "confidence region" will, in fact, be quicker and easier to learn than the conditional model (79): the gating receptive fields of (82), which must be optimised using a general-purpose iterative maximiser (section 3.3.2.5), have effectively disappeared, replaced by extra blocks in the better-behaved output-generating mappings κ_i, β_i. In their absence, the weightings ω revert to being a transition matrix, with a one-shot reestimation inherited from that for the transition matrix of an HMM (59). In other respects, the learning algorithm for the joint Samovar model is essentially similar to that for the conditional Samovar model (section 4.2.2.9), and it will not be specified in detail.

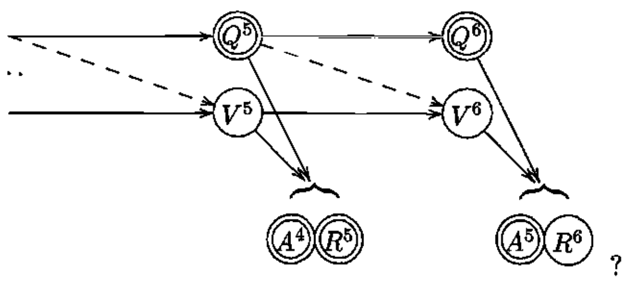
[5] *cf.* Vlassis & Kröse, *Robot Environment Modeling*

4.2.3.4. Predictions

Making predictions using the joint Samovar model is slightly simpler than with the conditional model in which the readings and actions have been folded into the world state (*cf.* section 4.2.2.10). The predictive distribution is just a specialisation of the general rule (54):

$$\begin{aligned} p(r^T \mid \theta, d^{[0,T)}) &= E_{v,q}\big[p(r^T \mid v^T, q^T, \theta) \mid d, \theta\big] \\ &= \sum_q p(q \mid \theta, d^{[0,T)}) \int_{v^T} p(v^T \mid q, d^{[0,T)})\, p(r^T \mid v^T, q^T, \theta) \end{aligned}$$

Conditioned on each sequence q, and supposing without loss of generality that $T = 6$, the situation is as follows:



The inference of V^6 and the assessment of $p(q)$ must be made without knowledge of the targetted R^6, but this merely requires a minor tweak to the formula by which the Kalman recursions incorporate the evidence. Each sequence q then gives rise to a Gaussian component in the overall mixture prediction, obtained by applying the output function of q's terminal component to $V^6 \mid q$, and projecting the resultant distribution over A^5 and R^6 down to R^6 alone while shifting it to take account of the known value a^5:

$$\begin{aligned} p(r^T \mid \theta, d^{[0,T)}) &= \sum_q p(q \mid \theta, d^{[0,T)}) \left|\frac{(\phi_i)_{RR}}{2\pi}\right|^{\frac{1}{2}} \exp\left(-\frac{1}{2}\left(r^T - \mu_i\right)' (\phi_i)_{RR} \left(r^T - \mu_i\right)\right) \\ \text{where} \quad \mu_i &= (\kappa_i)_{RV}\, \bar{v}_q^T + (\phi_i)_{RR}^{-1} (\phi_i)_{RA} \left(a^{T-1} - (\kappa_i)_{AV}\, \bar{v}_q^T\right) \\ \text{and} \quad \phi_i &= \left(\gamma_i^{-1} + \kappa_i \bar{v}_q^T \kappa_i'\right)^{-1} \\ \text{and} \quad i &= q^T \end{aligned}$$

As usual, the outer summation over q can be approximated by running the subsequence-joining algorithm (section 4.2.2.5) to produce a sample of candidates for the mixing state sequence over some reasonable time period leading up, in this case, to T (inclusive).

4.3. Other approaches to mixed-linear modelling

The learning algorithm set out above, which could be summarised as a KF nested inside an approximated HMM, is not the only way to handle the basic intractability of recursive mixed-linear models.

4.3.1. An aggressive variational approximation

One might, for example, observe that the mixing choice sequence Q is often known more or less exactly, *a posteriori* and given the true model. In that case, the posterior distribution of the linear hidden state sequence V is, like that of the hidden state of a Kalman filter, pairwise Gaussian (*cf.* section 4.2.2.3). So the question arises of a whether a cheap and dirty variational approximation can be made, along the lines of section 3.3.1.4, in which the ϖ takes the form

$$\varpi(\theta, h) = \varpi_\theta(\theta) \prod_t \varpi_{Q^t}(q^t)\, \varpi_{V^t, V^{t+1}}(v^t, v^{t+1})$$

where the Q^ts are treated as if they were independent. This allows the expectation over q in (88), which makes it necessary to resort to some relatively expensive approximation such as Samovar's subsequence-joining algorithm of section 4.2.2.5, to be replaced by individual expectation over the separate q^ts.

Unfortunately, it turns out in practice that although the process does tend to settle on parameter values which make the approximation true (*i.e.* the $p(Q^t = i \,|\, \theta, d)$s become indicator variables), the linear hidden state V is inevitably driven close to zero, so that the end result is merely a wastefully implemented recursive mixture of static linear maps[6]. The reason this happens is that the distributions $\varpi(v^t, v^{t+1})$ are reestimated using

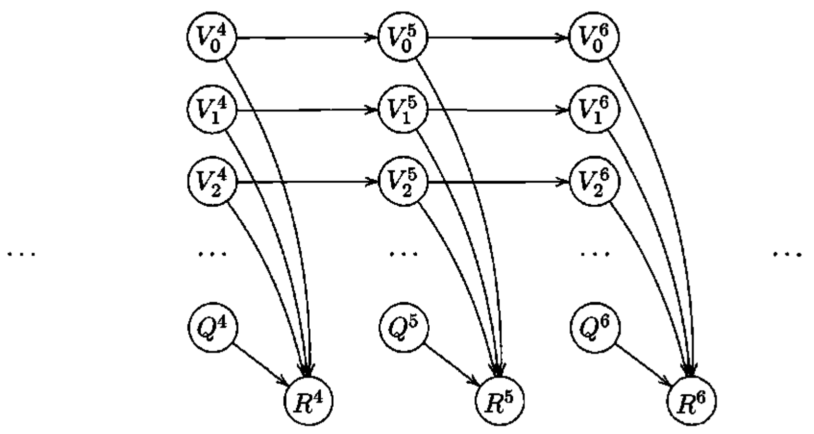
$$\begin{aligned}\varpi^{n+1}(v) &\propto \exp E^n_Q \left[\log p(q, v \,|\, \theta)\right] \\ &\propto \exp \sum_t \sum_i \varpi_{Q^t}(i)\, p(v^{t+1}, r^t \,|\, v^t, \theta) \\ &= \prod_t \prod_i p(v^{t+1}, r^t \,|\, v^t, \theta)^{\varpi_{Q^t}(i)}\end{aligned}$$

Instead of keeping the q-relative Kalman filter distributions separate so that the parameters can be updated on the basis of statistics derived from those q-sequences in which they are likely to have participated, as the Samovar's learning is careful to do, this procedure combines them by taking their product (an "and" rather than an "or"). The resulting mean is naturally near zero, and the variance very small.

[6] like the "autoregressive HMM" mentioned in Rabiner, *A Tutorial on Hidden Markov Models*

4.3.2. Switching state space models

Another variational approach[7] works by considering a slightly different graphical model than (79), namely



where Q^t selects both a linear/Gaussian output function and a linear hidden state V_i^t to which it is applied

$$R^t = \kappa_i V_i^t + N(0, \beta_i) \quad \text{where} \quad Q^t = i$$

while each of the V_is evolves according to its own dedicated linear/Gaussian dynamics

$$V_i^{t+1} = \lambda_i V_i^t + N(0, \alpha_i)$$

The likelihood is[8]

$$p(v, q, r, a \,|\, \theta) = \prod_t p(r^t \,|\, Q^t = i, v_i^t, \kappa_i, \beta_i)\, p(Q^t = i \,|\, q^{t-1}) \prod_j p(v_j^t \,|\, v_j^{t-1}, \lambda_j, \alpha_i)$$

This can best be described as a "recursive mixture of recursive linear models"—a bank of Kalman filters—as opposed to the "recursively mixed recursive linear model" of (79).[9]

Because each component of the model has a whole Kalman filter to itself, rather than sharing the same linear hidden state with the others, a variational approximation in which the Q^ts are treated as independent does not have the crippling effect it has on the Samovar model (section 4.3.1). The consequent learning algorithm takes the form of applications in turn (or in some order or other) of

- for each filter i in the bank, the standard Kalman-Rauch recursions (section 3.3.3.5), weighted at each timestep t by the current estimate of the

[7] Ghahramani & Hinton, *Variational Learning for Switching State-Space Models*

[8] omitting the terms for the initial conditions

[9] Ghahramani & Hinton, *Variational Learning for Switching State-Space Models* points out that similar models have been described in the engineering and econometrics literatures

probability $\varpi_{Q^t}(i)$ that i was responsible for generating the output r^t, yielding distributions $\varpi^n(v)$

- for each filter i in the bank, the standard Kalman filter M-step (72)/(73) and (74)/(75), the latter also being weighted by the "responsibilities" $\varpi_{Q^t}(i)$
- the standard HMM forward-backward recursions (section 3.3.3.3) to make fresh "responsibility" estimates $\varpi_{Q^t}^{n+1}(i)$, making use of the likelihoods

$$E_v^n p(r^t \mid h_i^t, Q^t = i, \kappa_i, \beta_i)$$

This is clearly much more efficient than the Samovar learning algorithm, since there is no need for the expensive expectation over longish q-subsequences (section 4.2.2.5), but it is only appropriate for applications in which the system being modelled can be relied on to stay in the same KF régime for a significant period, rather than potentially choosing a different rule at each timestep. There is no way for inferences, made on the basis of observations, about the Kalman state V_i^t of one filter in the bank to influence the estimate of the subsequent state V_j^{t+1} of another—and indeed if a filter is (believed to be) unused for any length of time, the variance of the estimate of its Kalman state will rapidly become such as to make it effectively useless. If, therefore, the motivation behind adopting a mixed linear model is to be able to approximate the rapidly changing dynamics of a robot's environment, the extra separability of the switching state-space model, from which it gains its superior efficiency, comes at a considerable price paid in the expressiveness thereby foregone.

4.3.3. Monte Carlo methods

There is an entirely different way to approach the problem of learning and using dynamical systems models which go beyond the small class of those with a tractable, closed form hidden state distribution $p(H)$. The overall form of the learning algorithms is still the same—they could still be called *EM* algorithms. But the way the E-step is implemented is fundamentally different.

4.3.3.1. E-*step by random sampling*

Variational learning algorithms, and the Samovar algorithm of section 4.2.2.9, work by building up an explicit (if approximate) representation of $p(H)$, and computing the sufficient statistics—generally, moments of that distribution—required for the M-step from that distribution. Monte Carlo methods, on the other hand, maintain a set of particular possible hidden states, simulating the dynamics of the model in such a way that their sampling distribution is $p(H)$.[10] If enough point hypotheses are tracked, they can be used to compute sample-based versions of the moments (or whatever other statistics the M-step wants) which will be good estimators of the true values.[11] The best known such technique in the robotics community is the "condensation" algorithm[12], developed for use in visual tracking problems and since applied to robot localisation[13].

The forward part of the Condensation algorithm proceeds as follows:

- generate a sample $\{h_i^0\}$ of "particles" from the initial state distribution $p(h^0 \mid \iota)$
- repeat for each timestep t
    - compute the likelihood $p(r^t \mid h_i^t, \rho)$ of each particle (possible hidden state) h_i^t in the current sample
    - choose a subsample $\{h_{*i}^t\}$ using the (normalised) likelihoods as sampling probabilities
    - generate the successor sample set $\{h_i^{t+1}\}$ by applying the stochastic model dynamics to each h_{*i}^t, *i.e.* sample from the distribution $p(h^{t+1} \mid h_{*i}^t, \delta)$

It can be shown that the moments computed from the sets h_*^t, weighting each particle by its likelihood, converge stochastically to the moments of the distributions $p(h^t \mid r^{[0,t)})$. Early versions of the algorithm based their parameter updates on statistics estimated from this approximation. With the addition of a smoothing backward pass[14] similar to the Rauch recursions and the HMM backward equations, it is possible to reweight the particles so as to obtain estimators of the moments of $p(h^t \mid r^{[0,T)})$—conditioned on the future as well as the past. As one might expect, this extension is reported to improve the effectiveness of the learning algorithm considerably.

4.3.3.2. Comparison with Samovar

Apart from its simplicity, the great benefit of the Monte Carlo approach is that it relies on no special properties of the distribution $p(H)$; this means, for instance, that the V-normalising problem which arises in the design of the conditional Samovar model (section 4.1.1.2), and is very hard to solve within a closed-form framework, becomes

[10] Doucet, *On sequential simulation-based methods* for a general review

[11] From a strictly Bayesian point of view, it would perhaps be possible to account quantitatively for the uncertainty in the model parameters introduced by the finiteness of the sample.

[12] Isard & Blake, *Condensation*

[13] see section 2.1.1.3

[14] Isard & Blake, *A smoothing filter*

irrelevant, so that it would be possible to use a condensation-type algorithm to learn a very "intertwined" model like that shown in (78). Indeed, the form of the model becomes within reason irrelevant; one interesting line of work even uses a "density tree" representation of the dynamics, borrowed from nonparametric statistics.[15] For the kind of tasks on which Samovar is intended to be useful, the most intriguing possibility thus opened up is perhaps a recursive Gaussian process model, or mixture of Gaussian processes.

On the other hand, for many applications one is likely to feel that slightly separable mixed-linear models like (79) remain a reasonable way of expressing one's ignorance, in a maximum entropy sense, about the true dynamics of the system—and within this class, the Samovar algorithm appears to offer a substantial advantage in terms of computational complexity. Blake[16] points out that the complexity of the condensation algorithm rises quadratically with the size of the sample set of particles if, as has been found necessary in practice, a smoothing version of the particle filtering E-step is employed. If the hidden state space is at all large, with a high-dimensional linear part or a large number of discrete mixing states, a large set of particles will be required to cover it adequately, and the running time will rapidly become prohibitive. The Samovar algorithm, which folds all the sampling at the V level into closed-form, Kalman filter-like recursions, and uses hidden Markov model-like recursions at the Q level to direct its sample generation, takes time quadratic in the much smaller number of mixing states (because of the HMM forward-backward equations) and cubic in the dimensionality of the linear hidden state (because of the Kalman-Rauch recursions). It has been found to perform acceptably on quite complicated problems.

There is no reason in principle why a Monte Carlo-based method could not be used for *making predictions* with a model learned with the Samovar algorithm, since there is then (obviously) no need to condition on future data. However, although a particle filter could probably be made to *infer* (a sample of) *good actions* in much the same way as the Samovar model (section 6.2), it would then be subject to the same rapid increase in computational complexity as in the essentially similar E-step inference.

4.3.4. Dynamic Bayes nets

The whole field of Bayesian time-repeated models with more or less sparse conditionality graphs (dynamic Bayes nets or DBNs) is a open research area. Among the most interesting recent results is Boyen & Koller, *Tractable Inference*, which investigates the extent of the error introduced when approximating assumptions are made about the hidden state distribution. This work reinforces the intuitively reasonable hope that the

[15] Thrun & Langford, *Monte Carlo Hidden Markov Models*

[16] Isard & Blake, *A smoothing filter*, p. 8; North *et al.*, *Learning and classification*, p. 26

perturbing effect of an imperfect representation at a particular timestep does not persist for all timesteps thereafter, or get magnified, but instead decays exponentially: the errors do not build up and render the H estimate useless.

4.3.5. Geometrical hypothesis tracking

Cox and Leonard's navigational method[17] of constructing a tree of hypotheses about the structure of the world around the robot, and using an extended Kalman filter to estimate the robot's position conditional on each branch of the tree, is a way of getting around the KF's representational limitations strikingly similar to that adopted by the Samovar model and algorithm. Note that because they are only interested in generalising the KF's forward-in-time fusion ability to a mixed-hypothesis distribution, they need only store the leaves of the tree; the Samovar algorithm, on the other hand, has to make inferences backwards to states which are still relevant for its purposes because it needs to use them for

[17] see section 2.1.1.1

Chapter 5

Evaluation

5.1. Synthetic data

Before assessing the performance of the Samovar model on a mobile robot environment modelling task, it is interesting to investigate its properties by running it against a controlled dataset.

5.1.1. The framework

The Samovar model is intended to be useful for applications where the system being modelled exhibits significant temporal structure of both "trend-like" and "qualitative" kinds, handled respectively by the model's Kalman filter and hidden Markov model elements. Different "situations" can succeed each other in an arbitrary order; the process dynamics in each situation is taken to be (approximately) linear/Gaussian, and the possibility of trend continuity between situations is left open by the incorporation of the linear hidden state. The synthetic data which helps elucidate the basic behaviour of the model is sampled from a process which genuinely does have the mixed-linear dynamics (82), separating the question of the model's effectiveness on its own terms from the issue of how useful this scheme proves to be as an approximation of more realistic dynamical systems.

5.1.1.1. *Situation structure*

One of the capabilities which it is most desirable that the model should possess is that of recognising the existence of two whole categories of situations, which appear the same but in fact form separate complexes within the temporal structure of the system. For instance, there might be two parts of a robot's environment between which it cannot distinguish using directly available sensory information, but border on detectibly different neighbouring regions. A simple example of this phenomenon can be defined using the following transition diagram:

$$\begin{tikzcd}
A' \arrow[r, bend left, "0.4"] \arrow[d, "0.2"'] & B' \arrow[l, bend left, "0.4"] \arrow[r, "0.2"] & P' \arrow[ll, bend right=50, "1.0"'] & \\
L \arrow[d, "1.0"'] & & & \\
A \arrow[r, bend left, "0.4"] & B \arrow[l, bend left, "0.4"] \arrow[r, "0.2"] & P \arrow[r, "0.2"] \arrow[uull, "0.8"] & G \arrow[uulll, "1.0"']
\end{tikzcd} \qquad (106)$$

Note that this is not a graphical model, like other similar-looking figures in previous chapters, but a depiction of a finite state-machine, the arcs representing permissible transitions between states, and the numbers transition probabilities. (Arcs from A, B, A', B' back to themselves are omitted for clarity.)

Here, the mixing states (situations) A', B', P' have the same linear dynamics as their counterparts A, B, P, and the transition probabilities between A, B are the same as those between A', B'. The only differences are that from the "true prompt" situation P it is possible sometimes to reach a "goal" situation G, while from the "false prompt" situation P' the only destination is A'; and that a transition into the A, B, P complex is always signalled by a "landmark" L. Because of the aliasing of the linear dynamics between the two groups of mixing states, it is impossible to resolve the ambiguity between them without making long-range inferences from the relatively rare L and G occurrences.

5.1.1.2. *Intra-situation characteristics*

The most difficult kind of data sequence which the algorithm could be asked to segment[1] is one in which the linear state is more hidden than visible. For instance, suppose that there are no process inputs—A is zero-dimensional—and the outputs R have smaller

[1] in the terminology of Ghahramani & Hinton, *Variational Learning for Switching State-Space Models*

dimensionality than the linear hidden states V. Then, other things being equal, the model will not be able to tell which component i was used at a given timestep t merely by checking how accurately each would map r^t to r^{t+1} through its dynamics matrix λ_i, because, unless λ_i is degenerate, different values of the linear hidden state H^t could produce any r^{t+1} whatsoever. Instead, it will be forced to use non-local information, *i.e.* the way the readings behave over several timesteps either side of t. For these experiments, the outputs R were made one-dimensional and the linear states V two-dimensional.

On top of that, the difficulty of finding the true segmentation will rise as the precisions α_i of the components' mappings fall, or their dynamics matrices λ_i become more similar. For this experiment, the dynamics matrices were sampled from an elementwise uniform distribution in the range $(-\frac{1}{2}, \frac{1}{2})$, which effectively guarantees that the hidden linear state will not diverge towards infinity and cause the program to abort.

$$(\lambda_i)_{kl} = \text{Uniform}\left(-\frac{1}{2}, \frac{1}{2}\right) \tag{107}$$

The noise variances (inverse precisions) were generated by drawing their square roots from an elementwise uniform distribution of variable width:

$$\begin{aligned} \alpha_i &= (\sigma'\sigma)^{-1} \\ \text{where} \quad (\sigma)_{kl} &= \text{Uniform}(-\mathcal{E}, \mathcal{E}) \end{aligned} \tag{108}$$

By adjusting $\mathcal{E}$, it is possible to control the solubility of the segmentation task.

5.1.2. Experimental results

The experiments performed with data sets generated from synthetic environments are directed towards answering the key question: does the subsequence-joining procedure implementing the E-step of the Samovar model's learning algorithm (section 4.2.2.5) work? Given the true values of the parameters of the model from which a data sequence was drawn, does the small set of proposed mixing states sequences Q match the true sequence in which the components were used?

In assessing the performance of the Q-inference algorithm, there are two measures which seem worth considering. One is the probability $p(r \mid q_?, \theta_*)$ of the observed outputs conditional on the sequence $q_?$ which it recommends as the single most likely one[2], compared

[2] The maximum likelihood mixing state sequence can, given the assumptions made by the subsequence-joining procedure, be obtained using the standard Viterbi algorithm in a way analogous to how the forward-backward equations were employed in section 4.2.2.5.

with their probability given the true sequence q_*. This can conveniently be expressed on a log scale as

$$\Delta\mathcal{L} = \frac{1}{T}\sum_t \left(\log p(r^t \mid q_?, \theta_*) - \log p(r^t \mid q_*, \theta_*)\right)$$

Another statistic which drops naturally out of the problem setup is the probability with which the algorithm suggests the system was in the discrete state complex A, B, P when in fact it was in A', B', P', or *vice versa.*

$$\mathcal{M} = \frac{1}{T}\sum_t \begin{cases} p^{\text{alg}}(q^t \in \{A', B', P'\}) & \text{if } q_*^t \in \{A, B, P\} \\ p^{\text{alg}}(q^t \in \{A, B, P\}) & \text{if } q_*^t \in \{A', B', P'\} \\ 0 & \text{otherwise} \end{cases}$$

This is a reasonable measure of its success in inferring the overall temporal structure of the data; in order to score well, the algorithm must correctly identify occurrences of the states L, G and P—which can only be achieved by a careful analysis of contextual clues—and use them as "punctuation marks" in conjunction with the structure of the transition matrix to resolve the ambiguity of what happens in between.

The following graphs show how these two statistics $\mathcal{M}$ and $\Delta\mathcal{L}$ vary with the noise level $\mathcal{E}$, with V-matching (section 4.2.2.6) enabled and disabled. Their data points were obtained as follows:

- Create a model θ_* with
  - the mixing state structure of (106)
  - random linear dynamics, as defined in (107)
  - random Gaussian noise terms depending on the setting of $\mathcal{E}$, as defined in (108);
- sample for 1000 timesteps from this model, recording the true mixing state sequence q_* and readings r_*[3]
- ask the algorithm of section 4.2.2.5 to reconstruct a sample of hypothetical mixing state sequences, given knowledge of r_* and θ_* (but not, of course, q_*);
- compute the statistic of interest ($\Delta\mathcal{L}$ or $\mathcal{M}$).
- Repeat 60 times, and report the overall average of the statistic.[4]

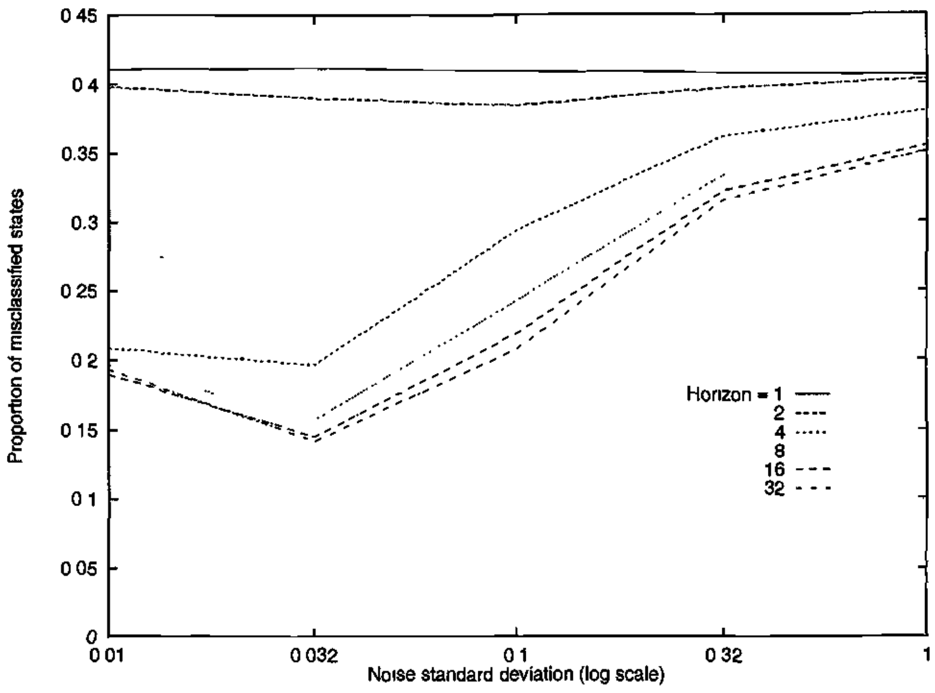
[3] Note that 1000 is not a model parameter, but simply a large enough number of example timesteps to get a low-variance estimate of each model's performance.

[4] Again, the repetitions are simply there to help estimate the model class's performance precisely; they do not affect the inference process itself.

The maximum number of candidate subsequences which the algorithm was allowed to maintain for each subsection (*candsMax* of section 4.2.2.9) was set at 30. Other settings of this parameter were not explored formally; as will be demonstrated shortly, the setting was high enough for the algorithm to achieve good results with V-matching enabled.

(For a visualisation of the subsequence-joining algorithm in action on this data set—albeit with a smaller *candsMax* setting—refer to section 4.2.2.8. Numerals 0–7 in the diagrams correspond to mixing states $A', B', P', L, A, B, P, G$ respectively.)

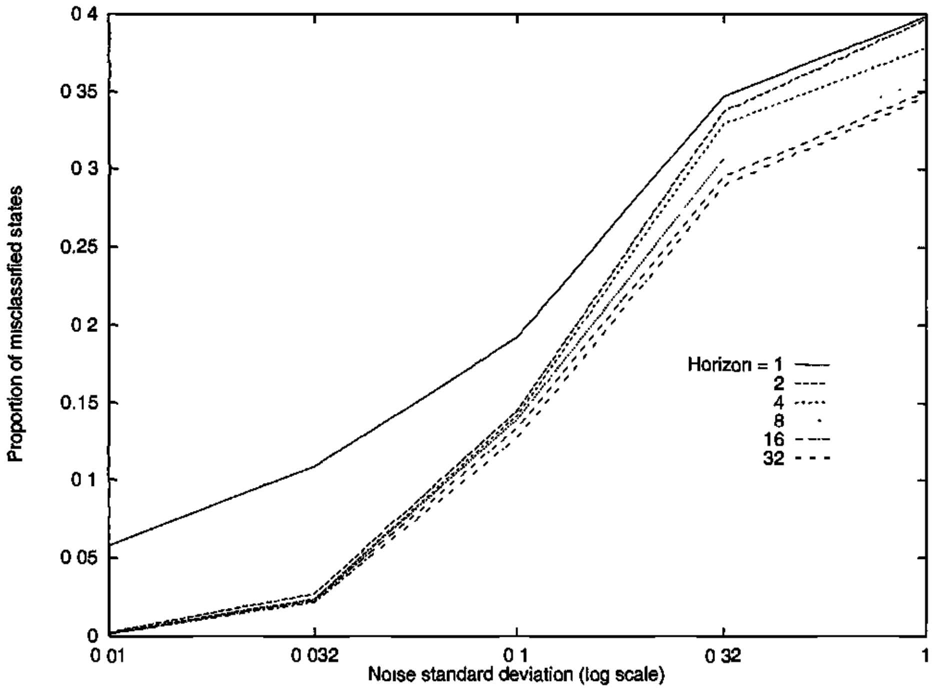
Turning first to the measure $\mathcal{M}$, it turns out as expected that the algorithm's inferences as to which group of mixing states (A, B, P or A', B', P') was active at each timestep improve as its horizon for linear state estimation is lengthened. This graph shows its performance when V-matching (section 4.2.2.6) is disabled:



The abcissa denotes the noise-determining $\mathcal{E}$ and the ordinate $\mathcal{M}$. When the horizon length is one, the algorithm is attempting to infer the linear state at each timestep only from its immediate influence on a single observable output; since that is impossible, there is no way for it to get any grip on what is going on, and its classification can only be random. Because the L and G states are not subject to misclassification under the scoring scheme, this means that $\mathcal{M}$ evens out at around 41% (rather than 50%) irrespective of the noise in the system. As the horizon is lengthened, the algorithm's resolving power improves and the proportion of misclassified states falls at all noise levels. Of course, in a really noisy system it is ultimately impossible to infer the mixing state sequence effectively; at $\mathcal{E} = 1$ the error rate has climbed to 35%.

Note that even in the best case, the average misclassification probability is still nontrivial at 15%, because of "possibility lossage". The *candsMax* = 30 memory slots available are sufficient to keep alive as hypotheses all the 16 two-step subsequences permitted by the transition matrix—the number of links in the diagram (106), including the omitted self-links of A, B, A' and B'. But, as the graph above makes clear, this horizon is still too short to support accurate Q-inferences, so the selection of 30 of the possible 625 candidates for covering each four-step section is necessarily somewhat arbitrary. Many of the right answers will be thrown away at this point, so that they are simply not available for consideration as part of the longer subsequences which would demonstrate their value.

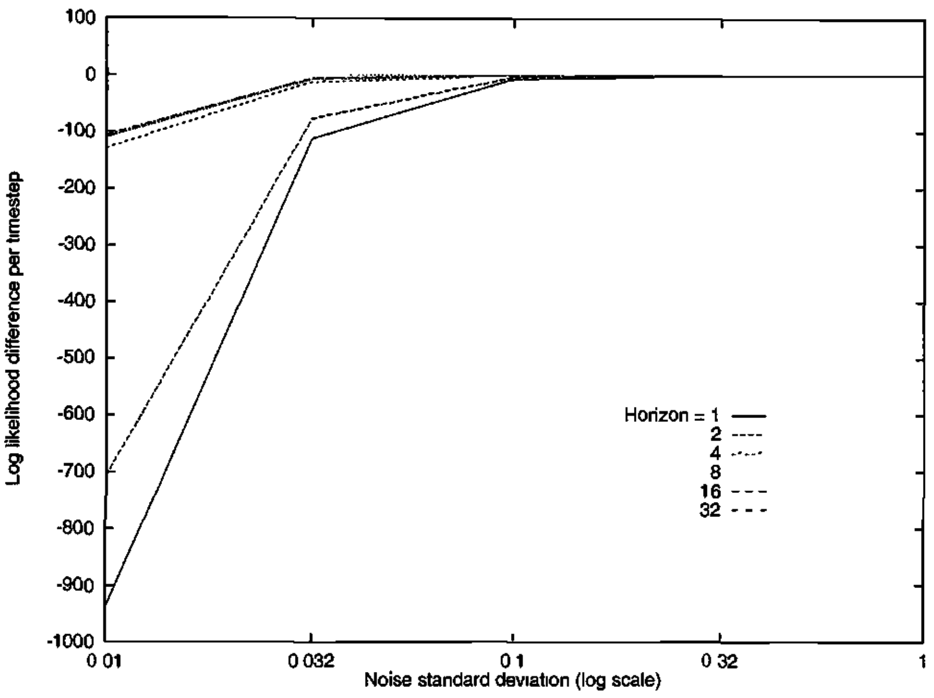
With V-matching enabled, the algorithm performs much better:



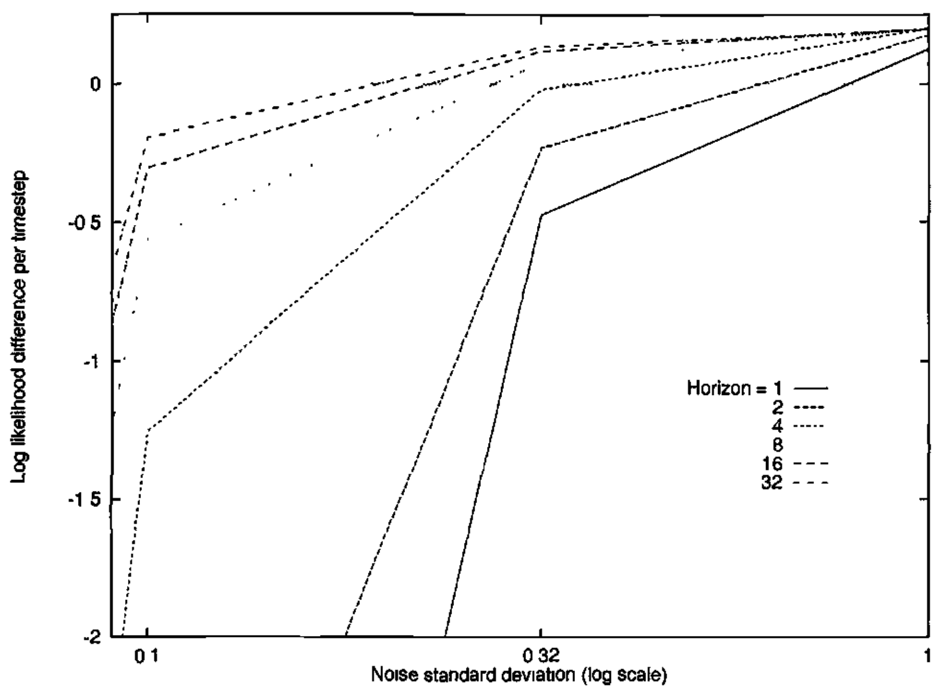
V-matching allows inferences about the linear hidden state made on the basis of a single observation in isolation to propagate forwards and backwards sufficiently well that in the absence of serious disturbing noise, the algorithm is immediately able to achieve an average misclassification probability around 6%. This in turn means that it is able to concentrate its hypothesis space on the most likely subsequences well before combinatorial explosion sets in and forces it to start pruning blindly. By the time it has raised its horizon to four timesteps, $\mathcal{M}$ has fallen below 0.3%[3]. As the noise level is increased, the estimates of V—and therefore the V-matching judgments—made on the basis of short sections become less precise; once $\mathcal{E}$ reaches 0.32, the longer horizons (eight and up) have a clear advantage.

[3] With a 32-timestep horizon, this is further reduced to 0.14%

The story told by the other measure $\Delta\mathcal{L}$ is similar. Here it is plotted against $\mathcal{E}$ with V-matching disabled:

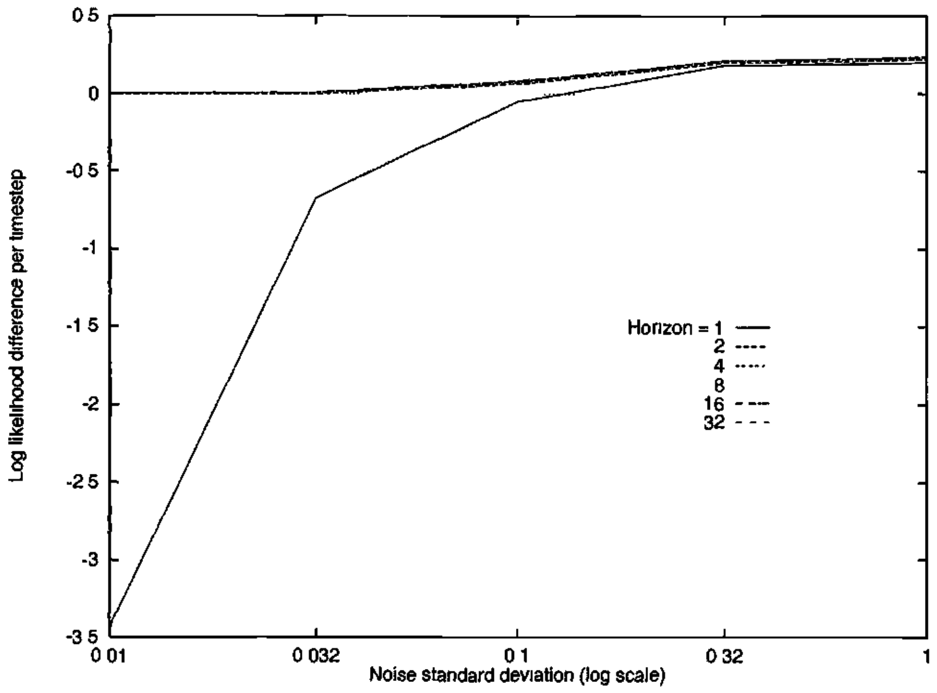


With a horizon length of one, the suggested best sequence $q_?$ turns out always to consist of repetitions of the single state A', the unconditionally most probable state in the Markov process defined by (106)—a fair choice, given that consideration of the individual process outputs does not give the algorithm any other information to work on, but one which is necessarily going to yield an essentially arbitrary prediction of the process outputs. It is to be expected that the log probability of those predictions relative to those made conditional on q_* will simply depend on the variance of the process output noise, as determined by $\mathcal{E}$; and indeed the graph shows $\Delta\mathcal{L}$ at horizon length one evening out around -1000, -100 and -10 as $\mathcal{E}^2$ increases from $\frac{1}{10000}$ via $\frac{1}{1000}$ to $\frac{1}{100}$. Given longer horizons, the algorithm does better, but "possibility lossage" means that the fit between $q_?$ and r_* is still somewhat strained: even a relatively small number of wrong mixing state choices can make it impossible to find a linear state trajectory which accounts satisfactorily for the observed outputs. It is only at higher noise levels that $q_?$'s likelihood reaches and exceeds that of q_*, as an expanded view of the graph shows:



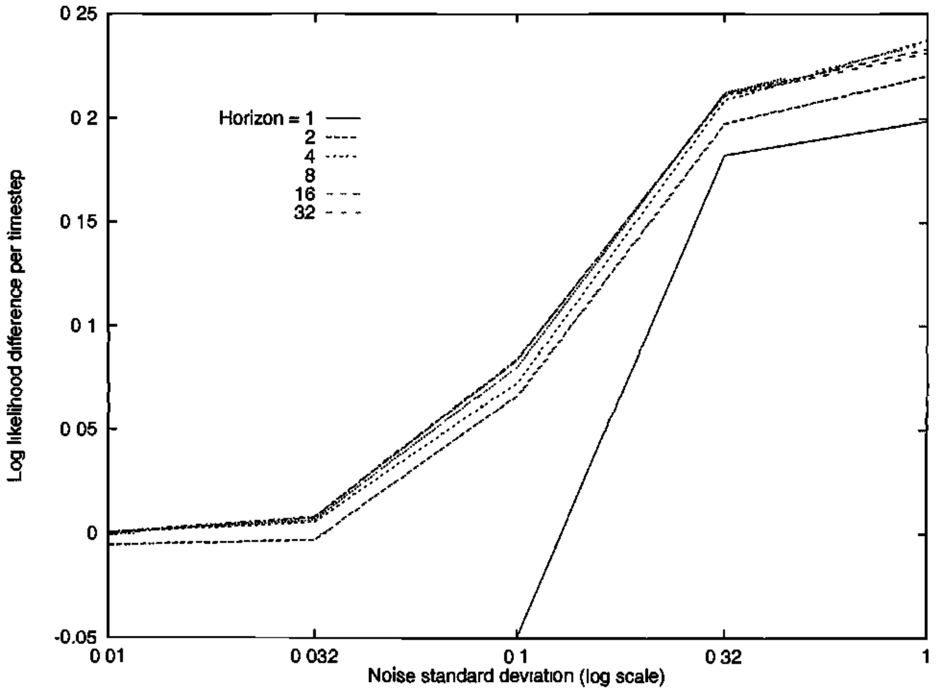
But this is no great achievement; under these conditions, little hard information can be gleaned from the outputs, so that the main determinant of the likelihood of a mixing sequence is the unconditional probability of its state transitions, and even the sequence $A', A', A' \ldots$ (with an unbeatable average transition probability of 0.4) comes to seem more probable than the truth.

With V-matching enabled, on the other hand, the algorithm can produce $q_?$s which are at least comparable in likelihood with q_* for all horizons longer than a single step:



(109)

The improvement in $\Delta\mathcal{L}$ gained by considering longer subsequences is marginal but consistent[4]; here is a detail from the same graph:



5.2. Simulations

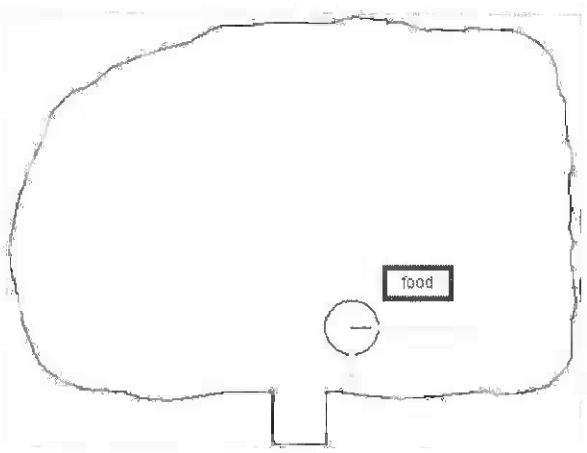
In order to provide a means of assessing the effectiveness of the Samovar model on a reasonably realistic, but still well-controlled, learning task, two simple simulated robot environments were constructed which exemplify some of the phenomena which the model was designed to handle.

5.2.1. Environment 1

As is traditional in autonomous robotics, the setup is a two-dimensional navigation task for a robot equipped with range sensors. In the following diagram, the robot is represented

[4] —except that it is reversed at the highest noise levels, with horizon lengths 4, 8, 16 yielding very slightly better likelihoods than 32. This effect was found to be repeatable but no clear explanation for it could be found.

by a circle; the irregular outer rectangle delineates the boundary of its enclosure, and the rectangle labelled 'food' marks a "target" region which is of special interest to the robot.



(110)

5.2.1.1. The robot

For simplicity, the robot is modelled as being perfectly round. It has two range sensors, one pointing forwards and one to the right—the minimum which could reasonably be required for modelling its anticlockwise wall-following behaviour. These sensors provide accurate measurements R^t_{ahead} and R^t_{right}, on a $[0,1]$ scale, of the distance to the nearest obstacle in their respective directions up to a range of 80 pixels, which (as can be seen from (110)) is quite small compared with the overall size of the enclosure. An additional sensor is provided which outputs $R^t_{\text{food}} = 1$ if the centre of the robot is positioned inside the 'food', and zero otherwise.

Action commands to the robot take the form of two variables A^t_0 and A^t_1, both $\in [0,1]$, which determine the distance and bearing in which the robot travels between timesteps t and $t+1$:

$$\begin{aligned} pos.x^{t+1} &= pos.x^t + dist^t \cos bearing^{t+1} \\ pos.y^{t+1} &= pos.y^t + dist^t \sin bearing^{t+1} \\ \text{where} \quad dist^t &= 40 \text{ pixels} \times \frac{a^t_0 + a^t_1}{2} \\ \text{and} \quad bearing^{t+1} &= bearing^t + 70^\circ \times (a^t_0 - a^t_1) \end{aligned}$$

The maximum distance the robot can move is therefore 40 pixels, or half the reach of of its range sensors. (This is intended to be reminiscent of how a tracked vehicle might respond to commands to move its left and right tracks foward by given amounts.[5])

[5] c.f. Pierce & Kuipers, *Map learning*

5.2.1.2. The world

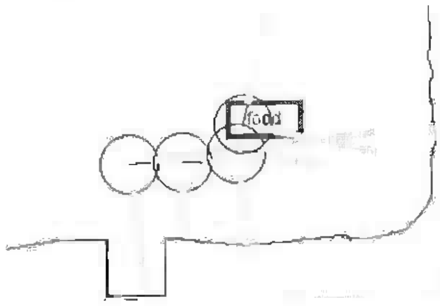
Most of the enclosure consists of straight or gently curving sections of wall. In these areas, the readings from the range sensors will exhibit trend-like relationships with the previous readings and the robot's actions, as the wall approaches and recedes at a rate dependent on the lengths of the steps made by the robot and its relative orientation. There is a (limited) legitimate role for hidden linear quantities, for instance where the curvature of the wall could be measured by comparing successive range measurements. These phenomena are probably best modelled using linear rules.

Other aspects of the environment are of a more qualitative and less smoothly regular nature, which could be captured using mixture rules. The abrupt changes in the direction of the wall at the corners of the arena fall into this category, as do the placements of the "food" area and the large notch in the wall positioned a little way clockwise from it.

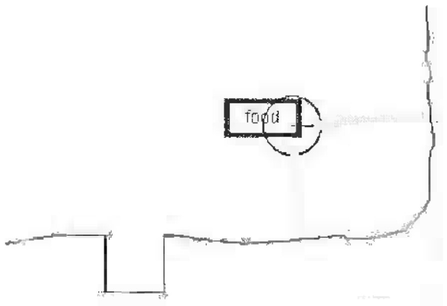
5.2.1.3. The data

The training set on which the model is trained, and the test set against which its performance is evaluated, were generated by steering the robot by hand around its enclosure in an anticlockwise direction, its distance from the wall varying from near zero up to about the limit of its range sensors. The trained model is expected to be able to predict the range sensor readings with fair accuracy.

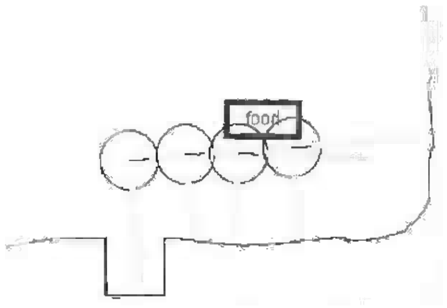
Each time the robot passed through the bottom right-hand corner of the arena, one of several behavioural variations was chosen, according to which it either did or does not at some point move into the "food" area and hence trigger the food-detecting sensor. When it reached the notch in the outer wall, which is designed to be a reliable landmark for the region near the food, it was always moved more or less directly forwards by the maximum distance for the next two steps. It could then be caused to make a sharp turn to the left, and—if not too close to the lower edge of the enclosure—would then find itself on the "food" area. Or it would simply be moved forwards, missing the "food" altogether. The possible scenarios are represented in the following diagrams:



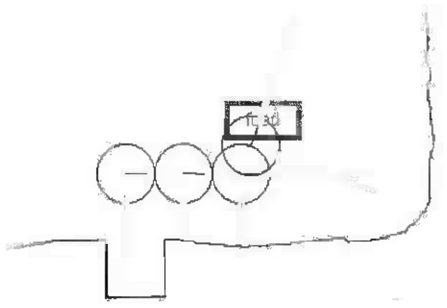
The robot turns and encounters the food (situation 1)



The robot turns back parallel to the wall, still on the food (situation 2)

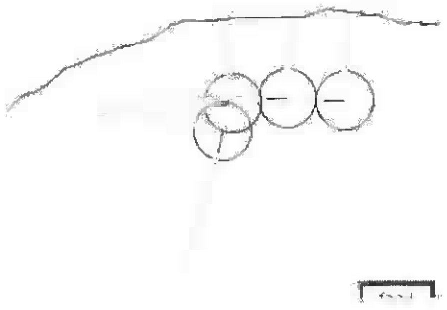


The robot misses the food because it does not turn (situation 3)

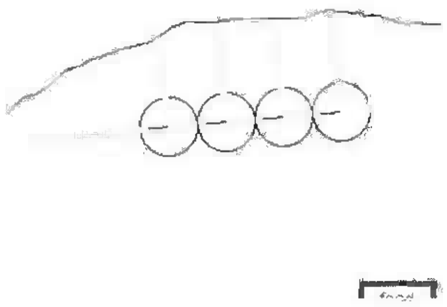


The robot turns, but misses the food because it was too close too the wall (situation 4)

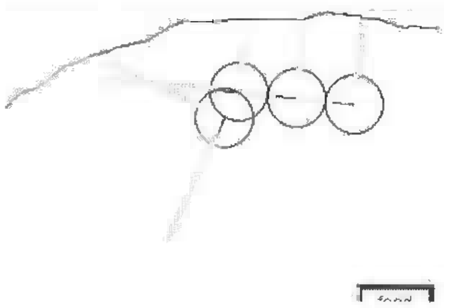
Similar sequences were played out when the robot passed through the opposite corner of the arena, except of course that the robot did not encounter the food since was in the wrong place.



The robot turns, but it is not in the right place to find the food (situation 6)



The robot is in the wrong place and does not turn (situation 7)



The robot would be too close to the wall even if it were in the right place (situation 8)

To these were added situation 5, covering a single timestep in the training data at which the robot was driven onto the food in a way which never otherwise happened (in the test data or in the training data), and situation 0, covering all otherwise unclassified timesteps. Over the 22 circuits of the arena which were made as the training data was being collected, and the 30 (separate) circuits from which the evaluation data was gathered, several examples of each combination of possibilities were included:

Situation	N^o *in training set*	N^o *in test set*
0	522	705
1	11	10
2	11	10
3	6	10
4	6	10
5	1	0
6	6	10
7	16	10
8	4	10

(111)

Overall, the training circuits took 583 timesteps to complete and the test circuits 775. No great effort was expended on making the test data identical in character to the training data on measures such as the robot's average distance from the wall; part of the point of a probabilistic model is that it should degrade more or less gracefully in the face of this kind of variability.

5.2.1.4. The model's task

Each of the robot's three sensors presents the model with a different kind of problem.

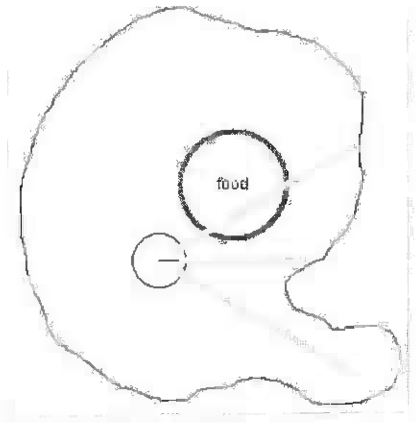
Because of the (manually generated) behaviour patterns from which the training and test data sets resulted—mostly, wall-following of a reasonably smooth boundary—the readings from the **right-pointing range sensor** could be predicted quite successfully by approximating linearly their relationship with their predecessor(s) and the robot's actions. However, the applicability of the linear rule (or rules) will vary depending on the context. There is also, potentially, scope for the model to learn the detailed shapes of characteristically irregular sections of the wall, and make sharper predictions than it would if it had to treat their effects as "noise".

Except when the robot comes to a corner, the **forward-pointing range sensor** will always read unity, to indicate that the way ahead is clear for more than 80 pixels. The general lack of obvious landmarks will perhaps make it difficult for the model to predict when these corners are going to occur, but the immediate successor to a non-unity reading should be easy to guess, since it will depend strongly and smoothly on the robot's current action.

The environment is, of course, carefully set up so that the model cannot predict accurately when the **food sensor** will read high unless it has taken on board some fairly subtle temporal phenomena. If it is to tell the food area in the lower right hand corner from the otherwise similar upper part of the arena, it needs to recognise the "notch" landmark, and remember it for three timesteps (by moving into a specific chain or subnet of mixing states Q). Then it has to judge whether the robot is sufficiently far from the wall to be able to reach the food, and has made a move which will actually take it there. If it fails to take all three factors into account, it will necessarily make either false positive or false negative predictions for the reading at several timesteps in the test data.[6]

5.2.2. Environment 2

Complementing the navigation task of section 5.2.1, a second environment was constructed with the aim of comparing the performance of the Samovar model on two problems involving different dynamics. As before, the robot exists in a two-dimensional enclosure in which is situated a designated "food" region.



(112)

[6] Note too that on top of this intrinsic difficulty, the model is not told that the food sensor's reading is a discrete rather than a continuous variable.

5.2.2.1. The robot

This time, the robot has three range sensors, providing accurate measurements on a $[0, 1]$ scale of the distance to the nearest obstacle straight ahead and 30° to either side, up to a maximum of 160 pixels (more than half of the internal width of the enclosure). The sensor indicating by means of a 0/1 output whether the robot is positioned inside the 'food' region is retained.

In the new environment, the robot's actions determine not its speed as before but its acceleration, via a (rough) simulation of momentum. However, the robot's angular motion is damped. In terms of the variables A_0^t and A_1^t, both $\in [-\frac{1}{2}, \frac{1}{2}]$,

$$\begin{aligned}
\mathit{pos.x}^{t+1} &= \mathit{pos.x}^t + \mathit{dist}^t \cos \mathit{bearing}^t + \frac{1}{2}\mathit{turn}^t \\
\mathit{pos.y}^{t+1} &= \mathit{pos.y}^t + \mathit{dist}^t \sin \mathit{bearing}^t + \frac{1}{2}\mathit{turn}^t \\
\mathit{speed}^{t+1} &= \mathit{speed}^t + \mathit{accel}^t \\
\mathit{bearing}^{t+1} &= \mathit{bearing}^t + \mathit{turn}^t \\
\text{where}\quad \mathit{dist}^t &= 40 \text{ pixels} \times (\mathit{speed}^t + \frac{1}{2}\mathit{accel}^t) \\
\text{and}\quad \mathit{accel}^t &= \frac{a_0^t + a_1^t}{2} \\
\text{and}\quad \mathit{turn}^t &= 70° \times (a_0^t - a_1^t)
\end{aligned}$$

These responses are intended to mirror roughly the effect of sending commands to lateral propellors fitted to a submersible robot (with a stabilising fin). An action $(0, 0)$ leaves the robot's speed and orientation unchanged, so that it coasts forward for a distance of 40 pixels times its previous speed. Turns can be accomplished via an asymmetric pair of motor commands, and the robot can speed up or slow down by issuing net positive or net negative pairs.

Environment 1 exhibits phenomena which are dynamic, but discrete, and linear, but static; the point of bringing momentum effects into environment 2 is to bring dynamic-linear phenomena into the picture as well—the kind of relationship which cannot be handled by the standard models, and which motivated the development of Samovar in section 4.1.

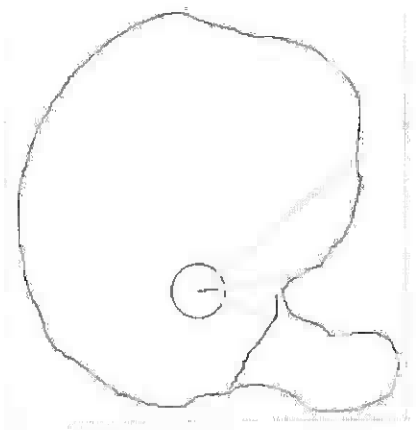
5.2.2.2. The world

As before, the enclosure is mostly quite regular in shape, so that there should be clear, near-linear relationships to be discovered between the range sensor readings and actions. However, they will depend critically on the robot's speed; if the action at particular time is the no-op $(0, 0)$, the distance by which the obstacles the robot is facing approach it, and hence the amount by which its range sensor readings decrease, will be determined entirely by its prior speed (as well as on their relative orientation). Of course, this is precisely the kind of situation for which Kalman filters were designed, and the model should be able to cope well by using the linear hidden state V^t to represent the robot's speed—along with whatever else it finds useful.

In addition, the environment does have some discrete features, including the food and the characteristic bulge in the outer wall, which will be best modelled by deploying different mixture components.

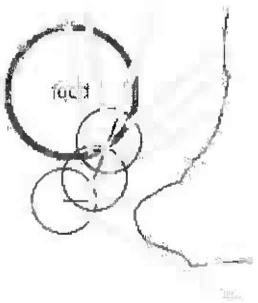
5.2.2.3. The data

A slightly different strategy was adopted in the generation of the training and test data sets for this environment. In the previous experiment, the robot was steered around the walls of the enclosure (110), performing a characteristic "food-encounter" manoeuvre in two similar places, one of which actually gave access to the food and was signalled (earlier in the circuit) by a landmark notch. Here, the robot is simply steered onto the food, or not, each time it passes anticlockwise past the landmark bulge. However, during half these passes, the state of the enclosure is changed from that shown in (112) so that the food and the landmark both disappear:

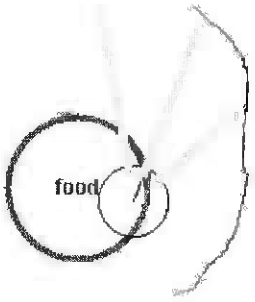


(113)

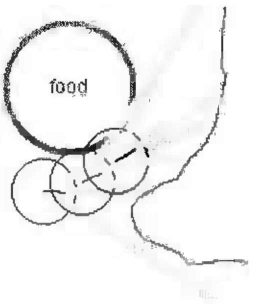
Both the training and test runs contained five instances with the enclosure in state (112), and five with it in state (113), of each of the following scenarios:



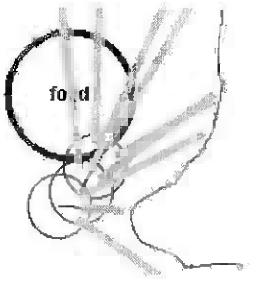
The robot turns and encounters the food (situation 1)



The robot is still on the food (situation 2)



The robot misses the food because it does not turn enough (situation 3)



The robot turns, but misses the food because it was too far away (situation 4)

Situations 6–8 are, for this environment, labels for the same parts of the robot's anticlockwise cycle through the environment, but with the landmark bulge closed off and the food absent. Situation 5 is, as before, a category for timesteps at which the robot encounters the food in an unusual way, which happened once in the test data for this environment but not in the training data, and situation 0 covers all the otherwise unlabelled timesteps.

The robot was driven at a variety of speeds ranging from below ten pixels per step to above fifty, sometimes moving at a steady speed for a number of steps and sometimes accelerating or decelerating. This, and the food's large size and circular shape, made it quite difficult to predict how long the food sensor was going to remain high once triggered (situation 2); in environment 1, the robot in practice always stayed on the food for precisely two steps. Distinguishing between situations 1 and 4 was also made more difficult, because the range of trajectories which the robot followed through the food area was considerably greater.

Overall, the training run comprised 645 timesteps and the test run 698, broken down as follows:

Situation	*N° in training set*	*N° in test set*
0	598	659
1	6	5
2	16	8
3	5	5
4	5	5
5	1	1
6	5	5
7	5	5
8	4	5

(114)

5.2.2.4. The model's task

If the model is to predict the **range sensor** readings accurately, it must learn how they depend on their predecessors and the robot's speed & direction, and how to estimate its speed by comparing successive readings, taking the accelerating effect of its actions in account, and so on. It could achieve reasonable results by representing its speed discretely, in the mixing hidden state Q, with separate linear components coming into play in different, quantised speed ranges; but it would appear to be easier and more effective for it to use instead the linear hidden state V. However, V may not be sufficient on its own as a means of addressing the structural differences between different parts of the enclosure: between, for instance, the corner with the landmark bulge and the smooth surface down the left hand side.

As before, the environment has been carefully arranged so that the **food sensor** readings cannot be predicted correctly without noting, and then remembering, a landmark characteristic for the presence of the food, and subsequently monitoring the robot's readings and actions to determine whether it is close enough and carrying out the right procedure for ending up in the designated region.

5.3. Models evaluated

This section details the pragmatic answers developed to the questions left open by the theoretical framework of section 4 and those which arose while running the experiments.

5.3.1. Training the conditional models

Most of the issues will be discussed in the context of the conditional Samovar models, which include as a special case the hidden Markov and linear (or autoregressive) hidden Markov models used as references against which to judge the behaviour of the models proposed here.

5.3.1.1. Regularisation

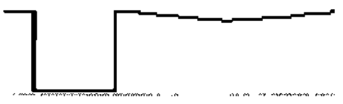
The question of how to control the complexity of the model falls outside the scope of the work reported here (see section 6.1.3). However, the issue of regularisation, which is (or is treated in the Bayesian theory as) an aspect of this problem, cannot in practice be ignored.

If any of the model's components are judged to have been used only at a few timesteps, it is possible that the maximum likelihood solution for its dynamics matrix λ to become degenerate or nearly so, with a value or subspace of values capable of accounting very accurately for all the readings the component is supposed to have been responsible for. A straightforward maximum likelihood learning algorithm will naturally seek to exploit this phenomenon in order to obtain a likelihood which is very high or even unbounded, by increasing the component's noise precision α_i—the theory that there is no noise in the process modelled by the component will be logically consistent with the data. But then the program may encounter a numerical singularity and fail; and even if it does not, the solution will most likely account poorly for future instances of the same situation, since in all probability there will in fact be unpredictable variations in the precise values of the readings, which will be given a very low probability by the over-precise predictive distribution. The same phenomenon can happen on the input side if a component's gating patches are made very small.

To avoid this problem, it's necessary to add some terms to the objective function targeted by the optimiser which penalises over-specificity when there is little data to go on. In the Bayesian formulation, these terms are nicely interpreted as priors expressing a more or less strong belief that there is some noise to be taken into account; they achieve the desired effect of bringing the maximum *a posteriori* parameters back from the degenerate maximum likelihood point, while fading into the background if there is more data available. For practical reasons, it's obviously desirable to choose the form of the prior in such a way as to avoid upsetting the tractability of the update rules. One possibility is to put a gamma distribution on the determinant of the noise precisions α_i and patch precisions γ_i: it turns out that the extra terms introduced slip unobtrusively into the likelihood (82). Even more straightforward is a conjugate prior taking the form of offsets to the sufficient statistics, with an effect equivalent to that of some previously observed data with particular characteristics.

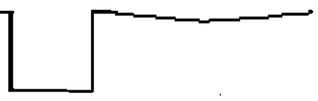
However, when the system was tried out on the simulated robot data, a serious problem was uncovered in the way either of these kinds of regularisers interacted with the procedure for making predictions (section 4.2.2.10). One way in which the model can account for the temporal structure of the environment between the landmark notch and the "food" is

$$\begin{tikzcd}
 & & & \boxed{\uparrow G} \arrow[r] & \cdots \\
 & & A \arrow[r] & B \arrow[u] & & \\
\cdots \arrow[r] & L \arrow[ur] \arrow[dr] & & & X \arrow[r] & \cdots \\
 & & X \arrow[r] & X \arrow[ur] & &
\end{tikzcd} \tag{115}$$



This is a transition diagram like (106), showing possible trajectories of the mixing state Q over the timesteps following the one at which is encounters the landmark notch. The state L is specialised for the situation where the robot is next to the notch; from L, Q can evolve either through a similarly specialised complex A, B, G of states, leading to one in which the "food" sensor reads high, or through more normal states, X say, which account generically for the typical situation in which the "food" sensor stays low. Of course, there is a more economical structure available:

$$\begin{tikzcd}
 & & & \boxed{\uparrow G} \arrow[r] & \cdots \\
\cdots \arrow[r] & L \arrow[r] & A \arrow[r] & B \arrow[u] \arrow[r] & X \arrow[r] & \cdots
\end{tikzcd}$$



Nevertheless, the model does generally converge on the solution (115), and there is no obvious reason of principle why it should not work adequately. Indeed, when the model considers its experiences retrospectively, it can easily tell which branch Q must have taken $(L, A, B, G$ or $L, X, X, X)$, because the "food" sensor reading of state G is immediately recognisable and otherwise very unusual. The problem comes when the model wants to make a *prediction* for what is about to happen three steps after it has seen the notch, *i.e.* when it has to decide whether it is going to encounter the food or not. One would hope that it will consider both branches as possibilities, so that its prediction can take the form of a mixture, one component arising from the $L, A, B, (G)$ possibility, and the other from the $L, X, X, (X)$ hypothesis; the former will postulate a "food" sensor reading of 1.0 and the latter one of 0.0.

What happens in practice, however, is that the former theory is entertained with a very low probability or not at all. The problem is simply that the state(s) X are just as good as A, B at accounting for the generic, but very consistent and predictable, situation in which the robot is merely moving along a roughly straight wall. In fact, they can always account perfectly for the readings of both the forward-pointing range sensor (one) and the "food" sensor (zero); and this means that if the regulariser is based on a prior subject to domination by the likelihood, their output noise variances will tend asymptotically to zero with the number of timesteps at which they are considered to have been active. And since they are used very commonly in similar situations all over the arena, while the alternatives A, B are deployed only occasionally, this means that their predictions will necessarily be much more precise—whatever the setting of the prior's parameters. This is what accounts for the model's preference for the $L, X, X, (X)$ branch: it really does look more likely.

At root, it is a mismatch between the model and reality that underlies the phenomenon. To model the "food" sensor output as a continuous variable is, of course, asking for trouble, and the model could easily enough be extended to handle discrete outputs in a more informed way (section 6.1.2.1). However, the problem would remain with regard to the range sensor outputs, which are sometimes perfectly predictable and other times subject to considerable and uncertain variation.

The solution adopted was simply to add a fixed offset to the model's output variances themselves, effectively capping the precision with which any reading can be predicted. A, B and the X states are thereby placed on the same footing and the prediction goes through successfully. Offsetting the output variances is not equivalent to supposing some fixed number of previously seen training examples (the number would have to be variably equal to the amount of actual data); it is more like blurring all the data points with some fixed Gaussian uncertainty, or indeed adding Gaussian sensor noise—which makes the environment closer in character to how the model assumes it will be, at the cost of losing some information. For the experiments reported here, this artificial variance was set to a diagonal 0.0001. No further regularisation-related difficulties were encountered.

5.3.1.2. Structural adaptation

The other major task associated with complexity control is deciding the number of components in the model. Like the standard Baum-Welch HMM learning procedure (section 3.3.3.3), the Samovar learning algorithm as presented in section 4.2.2.9 focusses entirely on the problem of training a model with a previously fixed number of mixing states. Bayesian theory does in principle also give a formula for the estimate of the "right" number σ of states, but actually implementing it is another matter. The issue will be discussed in section 6.1.3. For now, its most trouble manifestation—namely, the phenomenon whereby a model with a large number of states can "overfit" the noise in the training data in the way described in section 5.3.1.1, and perform poorly on further data drawn from the same distribution—will be sidestepped. The number of components will be limited to a level which is known to be sufficient but not wildly excessive, and likelihoods quoted in test results will be computed using an independently generated data sequence, not with the sequence used for training the model. Depending on the character of the application, such a pragmatic approach can prove effective in practice, especially since many mixture models are to some extent self-regulating: they do not always find "work" for all their components merely because they are there, instead allowing the weighting given to "unnecessary" ones to decay to zero.

Indeed, when working with a mixture of experts model (static or recursive), it is not uncommon to encounter the opposite difficulty. This is because the chief challenge in training such models is overcoming a kind of bootstrapping problem. Given the true parameters according to which expert generates its outputs, an *EM* algorithm can reliably infer the gating parameters which determine the probabilities of each expert being chosen given a certain input (and *vice versa*). However, before the algorithm has worked out what experts are in play, it is liable to underestimate the number of classes into which the inputs may need to be divided: if the trigger ranges of two components in the true model overlap, but the learned model does not yet handle either case accurately, then there is no likelihood-increasing "incentive" for it to make what appears to be a distinction without a difference by placing two receptive fields in the same part of the input space; and, overfitting notwithstanding, it will often settle on a solution where one expert deals poorly with both kinds of input.

One way to work around this is

- start with a small number of components (*e.g.* one)
- train them for a while using the *EM* algorithm
- remove the ones which are judged to have been active at a very small (expected) number of timesteps:

  $$\left\{ i : \sum_q p(q \,|\, \theta, d) \sum_t \delta(q^t, i) \ll 1 \right\}$$

  —the numbers $p(q \,|\, \theta, d)$ being calculated as ***seq.prob*** by section 4.2.2.9
- duplicate the remaining ones
- repeat a few times, until there are "enough" components

There are several obvious free parameters involved in this procedure, which have no inherent semantic importance but may nevertheless influence the nature of the solution on which the model eventually settles as well the time it takes to do so. Furthermore, the "duplication" step can be implemented in a variety of ways: as a copy with a random perturbation of fixed magnitude; or by going back to the posterior distribution of the component's parameters and constructing two replacements from either end of its major axis; or with a straight copy, relying on the random element of the sequence-joining algorithm to break the symmetry.

5.3.1.3. Training procedure

The training régime used for the simulation experiments was chosen after a little trial and error development as one that seemed to work reasonably well:

1) start with a single component, with regularisers fixed to bias it towards a constant output
2) *Map out where the outputs fall*
    - do once or twice
        - E-step
        - M-step outputs
        - duplicate the components
    - do five times
        - E-step
        - M-step outputs
3) *Get a rough idea of the discrete temporal relationships between the different parts of the space*
    - do once or twice
        - duplicate the components *(to allow for a degree of aliasing)*
    - do ten times
        - prune unused components
        - E-step
        - M-step outputs and background weightings

4) *Bring linear mappings into play where there is a demand for them*
    - relax regularisers to allow linear mappings
    - do 20 times
        - prune unused components
        - E-step
        - M-step outputs and background weightings

5) *Finalise the discrete temporal relationships*
    - duplicate the components
    - do 20 times
        - prune unused components
        - E-step
        - M-step background weightings
    - at last,
        - E-step
        - M-step gating patches

6) *Settle down*
    - do 20 times
        - prune unused components
        - E-step
        - M-step outputs and background weightings

The pattern followed is one of increasing model sophistication: the training script moves from a single Gaussian (step 1), via an unconditional mixture (step 2) and a hidden Markov model (step 3), to a recursively mixed recursive linear model (step 4), before finally throwing in the gating conditionality. It helps to reserve this last step to the end, because a naive implementation of the conjugate gradients-based optimisation (section 3.3.2.5) is necessarily rather slow; it is possible, however, that the number of discrete states left at the end of training is larger than it would be if the gating parameters were optimised at various earlier points as well. If the steps before step 4 are omitted, so that the model is allowed to introduce temporal linear relationships early on in training, it immediately tries to use these to account for all the phenomena in its world—even those which are much better explained in terms of the discrete dynamics, such as its regular encounters with the food or with the corners of the arena, both of which are essentially discontinuous in their effects on the robot's sensor readings.

By varying the number of times the model's components are duplicated in steps 2 and 3, different sizes of model can be created. For instance, a model of size 32 can be obtained by performing two duplications each time, and one of size 8 by performing them just once. In step 2, where the aim was to get good coverage of the data, each component i was duplicated by computing the posterior of its mapping λ_i (a matrix-variate Gaussian) and substituting two components with mappings drawn from the opposite 0.5 standard deviation points of that distribution's major axis. In later steps, where the aim was to create possible aliases for situations, the components were simply duplicated precisely (though they were allowed to diverge thereafter). When a component was split, the transition probabilities to and from it were divided approximately equally between its offspring, creating a 2×2 square block in ω. In fact, it would be better to spread the probability more widely, allowing the new states to be plugged in in more flexible ways: the present implementation was observed to settle on quite wasteful structures such as (115).

5.3.1.4. Optimisation of the gating receptive fields

When the Gaussian receptive field-based gating rule was introduced in section 3.3.2.5, it was suggested that the parameters for the rule could be optimised by applying a conjugate gradients minimiser to the derivative with respect to them of the log likelihood. However, no suggestion was made as to how a good starting point (seed) could be found; and in practice, it turns out that the quality of the initial seed has a big influence on the rate at which the algorithm is able to converge and on the optimality of the solution it finds: the space seems to be quite a difficult one in which to search, with a strange shape and many local minima.

Good results were obtained by initialising the components' receptive fields as if they were actually a generative model of the readings and actions which were supposed by the model to have triggered them, and shrinking them by a small constant factor. (Recall from section 3.3.2.5 that the receptive fields of a conditional model are not intended to represent a mixture probability density, as those of a joint model do; here, the density is being used as a convenient starting point.) The conjugate gradients algorithm is then able to move the fields around and reduce the initial error (negative log likelihood) by a factor of two or so.

However, it turns out that it almost never manages to change the size of the fields significantly: they stay very close to the size to which they were initialised. In consequence, the confidence region semantics are somewhat compromised. A closer investigation of the shape of the error landscape, the way it depends on the parameterisation of the receptive fields, and the workings of the optimiser would be needed to explain exactly why. The conjugate gradients optimiser was found to work much better on synthetic test data, which differed from the simulated robot data most obviously in that it was uniformly and relatively densely distributed.

Because the conjugate gradients optimiser tended to leave the gating fields' sizes alone, the regulariser adopted to avoid them becoming too small was rendered irrelevant. In principle, however, any differentiable regulariser could be used with negligible cost, since the parameter optimisation is already being performed iteratively. During the seeding phase, which in practice determined the fields' size, 0.0001 was added to the diagonal of the inferred field "variances" in order to work around the problems mentioned in section 5.3.1.1.

5.3.2. Training the joint models

Most of the pragmatic issues raised in section 5.3.1 carry over from the conditional variant of the Samovar model to the joint one. The training régime described in section 5.3.1.3 needs a little adaptation, however, since the joint model, being unable to represent immediate linear relationships between readings and actions at successive timesteps, is not prone to over-eager use of linear interpolation, and also requires no expensive iterative optimisation of a conditional gating rule. Steps 4 and 5 are thereby vitiated. On the other hand, the joint model was from time to time observed to settle quickly on large output variances, which would decay only very slowly. In order to avoid that, the variances in question were clamped during the first part of training. The procedure adopted was, therefore, patterned after but not identical to that for the conditional model:

1) start with a single component, with output noise variance clamped to 0.0025
2) *Map out where the outputs fall*
    - do once or twice
        - E-step
        - M-step everything, except for output noise
        - duplicate the components
    - do five times
        - E-step
        - M-step output matrix, but not output noise
3) *Get a rough idea of the discrete temporal relationships between the different parts of the space*
    - do once or twice
        - duplicate the components

- do eight times
    - E-step
    - M-step everything, except for output noise

4) *Release the output noise*

- do thity times
    - E-step
    - M-step
    - prune unused components

5) *Finalise the temporal relationships*

- duplicate the components
- do 10 times
    - E-step
    - M-step transition matrix
    - prune unused components
- do 30 times
    - E-step
    - M-step
    - prune unused components

5.4. Observations

Each of the models described in section 5.3 was evaluated on each of the two tasks, with a view to investigating the following issues:

- How well do the Samovar models perform compared with baseline HMMs and linear HMMs, with respect to
    - predicting when the robot will encounter the "food", and
    - predicting range sensor readings?

- Is it possible to extrapolate from performance on the training data to performance on the test data?
- Does the introduction of a confidence region improve the models' effectiveness?
- How does the Samovar model use the mixing and linear states available to it?

5.4.1. Procedure

The evaluation covered conditionally gated HMM and linear HMM, and conditional and joint Samovar, models with 8, 16, and 32 mixing (Q) states; the dimensionality of the linear state (V) was varied between 1 and 3 with the conditional Samovar models, and between 2 and 4 with the joint ones (which need more since they cannot directly exploit relationships between temporally adjacent actions and sensor readings—see section 4.2.3.1). Instances of each model were first trained on one set of data, and then evaluated both against that and against a separate training series; this process was repeated twenty times for each of the two simulation environments, in order to get a reasonable idea of the variability in the results. Furthermore, separate models were trained and tested with the model failure probability set to zero (in which case the confidence region is wholly disabled) and 0.1. Models trained with a confidence region were also re-run against the training and test data sets with the model failure probability set to 0.5. Note that these probabilities did not reflect the frequency with which the fallback component was actually judged to have been the most probable, because of the mismatch between its static Gaussian predictive distribution and the bounded $[0, 1]$ range of the sensor and action spaces.

5.4.1.1. Criteria: "food" sensor

Each model's success at predicting whether it was going to encounter the "food" at a given timestep was measured by considering the probability with which it suggested that the food sensor reading would exceed 0.5. Ideally this should be unity in situations 1 and 2 (as defined in section 5.2.1.3 and section 5.2.2.3 for the first and second simulated environments respectively), and zero in situations 0 and 3–8.

For simplicity, the predictions were obtained by running the algorithm of section 4.2.2.10 from scratch using the preceding eight experiences as raw material, rather than using the "candidate set cache" optimisation suggested in that section. Extending the "history" window more than eight steps into the past was not found to affect the predictions significantly, which is as it should be given the way the environments were designed.

The criterion was extended to the whole of the model's training or test run by averaging over all the timesteps in each situation category: the average predicted food probability for situation i is

$$F_i(\theta \,|\, r, a) = \frac{1}{|\mathrm{sit}_i|} \sum_{t \in \mathrm{sit}_i} p(R^t_{\mathrm{food}} > 0.5 \,|\, r^{[t-8,t)}, a^{[t-8,t]}, \theta)$$
$$\text{where} \quad \mathrm{sit}_i = \{t : t\text{'s situation category is } i\}$$

From these quantities it is possible to compute the robot's gains given any reward matrix covering the cases of true positive, false positive, true negative and false negative predictions; the point of separating out the different situations is to show whether there are any in which it consistently makes mistakes (*e.g.* the deliberately difficult situation 6).

The experimental food prediction results will be presented in the form of graphs showing at a glance how each of the twenty models in each class promised to do—how it performed during the pass over the training set—and how well it actually did on the test set.

5.4.1.2. *Criteria: range sensors*

When it comes to assessing the models' performance at predicting the range sensor data, the natural measure of how well the actual measurements agree with what the models say they should be is

$$p(r^t \,|\, r^{[0,t)}, a^{[0,t]}, \theta)$$

—the predictive pdf implied by the model θ for the sensor readings R^t, evaluated at the actually measured point r^t. Loosely, this could be called the "predicted likelihood" of the sensor readings, but because the pdf is continuous, the numbers can range from zero up to arbitrarily high levels, so to avoid confusion the term "nonfood predictive pdf value (or *NF*)" is adopted.

Note that an accuracy measure based on, say, the difference between the mean of the model's prediction on the one hand, and the observed value on the other, would unfairly penalise multimodal predictive pdfs.

To obtain a summary performance measure for a class of models, the predictive pdf is geometrically averaged over t in the whole test run, with the predictions based on a window of the preceding eight experiences, and geometrically averaged again over θ in the class:

$$NF(\text{models}) = \exp\left(\frac{1}{20} \sum_{\theta \in \text{models}} \log NF(\theta \,|\, \text{test data})\right)$$
$$\text{where} \quad NF(\theta \,|\, r, a) = \exp\left(\frac{1}{T} \sum_t \log p(r^t \,|\, r^{[t-8,t)}, a^{[t-8,t]}, \theta)\right)$$

Note that this is not a decision-theoretic measure, but rather a predictive analogue of the objective targetted during training.

Again, the experimental food prediction results will be presented in the form of graphs showing a sample of the performance, and, just as importantly, the generalisation success of each model class.

5.4.1.3. Summary of the protocol

- repeat for each model class in the list: conditionally-gated HMM; conditionally-gated linear ("autoregressive") HMM; conditional Samovar with 1, 2, 3 dimensions of linear hidden state; joint Samovar with 2, 3, 4 dimensions of linear hidden state
  - repeat with instances of the model class with 8, 16, 32 mixing states
    - repeat for model failure probability = 0.0, 0.1
      - repeat twenty times
        - repeat for each of the two environments
          - train the model on the training data for the environment
          - get the model to predict each experience in the training set from its eight predecessors, and record its performance according to the criteria of section 5.4.1.1
          - in the same way, evaluate the model's predictions over the test data for the environment
      - plot the numbers obtained from all twenty trained models on the scatter plots presented below, to show both performance and generalisation chacteristics

5.4.2. Environment 1

The observations made from the experiments are broken down by task—those collected from environment 1 being reported here and those from environment 2 in section 5.4.3—and subgrouped according to sensor class (range or food), model variant (conditional or joint) and model size (number of components). For conciseness, the model types are abbreviated *e.g.* HMM/8 for an eight-component hidden Markov model or Samovar/3/16 for a sixteen-component Samovar model with three-dimensional linear hidden state.

5.4.2.1. Nonfood

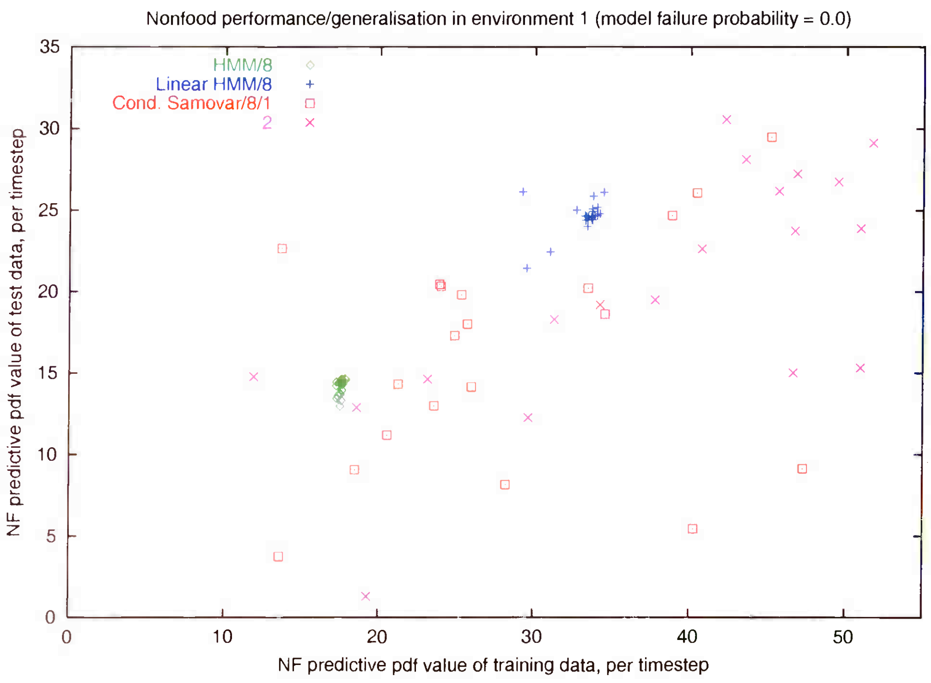
Turning first to the quality of the models' range sensor predictions, it will be convenient to present the results not through summary statistics, but in scatter graphs of the performance of all twenty of the trained models in each batch on both the training data and the test data. Showing all the data at a glance in this way makes it possible to judge the intra-class variability, and, crucially, the degree to which the potential shown by each model translates into success on unseen data. Each marker on the graph corresponds to one of the twenty models trained from each class; the model's *NF* predictive pdf values[7] on the training set becomes the abcissa of the marker, and its *NF* predictive pdf value on the test set the ordinate. By using differently shaped markers, a single diagram can be used to compare several different model classes.

One important reason for showing several classes of model on the same plot is that it's hard to give a simple interpretation of "how good" a (geometric) mean predictive pdf value of, say, 25 actually is, in absolute rather than comparative terms. As a rough indication of the general accuracy of the models, one can bear in mind that 25 is the best predictive pdf value theoretically achievable by a model which quantised the two sensor readings onto a five-by-five histogram.

The first scatter graph presents the training and test *NF* values for the HMM, linear HMM and Samovar models with eight discrete states, with the model failure probability

[7] See section 5.4.1.1

controlling the confidence region set to zero both during training and during testing:

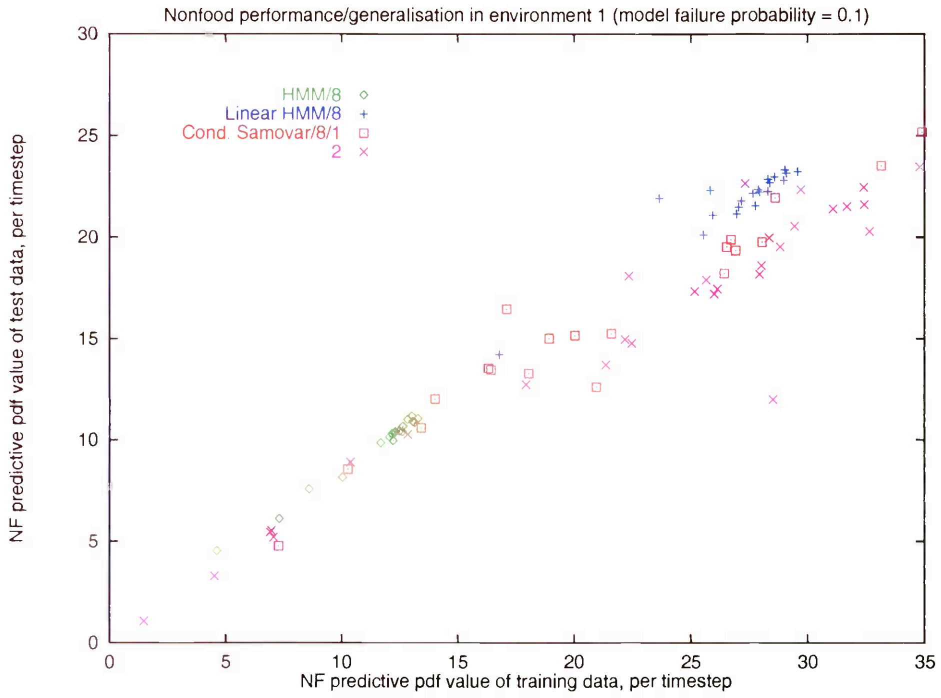


The linear HMMs are clustered in the top right hand corner (right for most promising on training data, top for most successful on test data). The plain HMMs are clustered further down and to the left. Many of the Samovar models perform better on the training data than the linear HMMs, but only one of the Samovar/8/1s and a few of the Samovar/8/2s fulfil this promise when evaluated against the test data: the others differ wildly, and unpredictably in the their training data predictive pdf values correlate poorly with their test data predictive pdf values.

The reason why the the Samovars mostly find it difficult to beat the linear HMMs on the *NF* measure in this environment is simply that there is quite little useful role for linear hidden state here. Note, though, that the *NF* success of the linear HMMs is achieved at some cost in reduced accuracy in predicting the "food" sensor readings: see section 5.4.2.2.

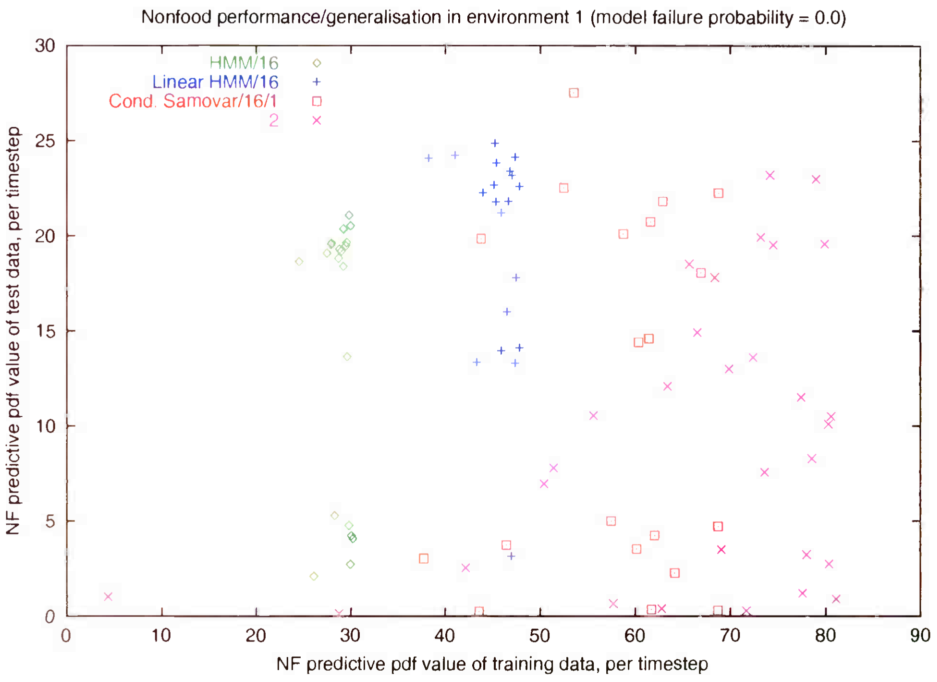
One surprising feature of the plot above is that two of the models score better on the test data than on the training data (the Samovar/8/2 with $NF_{\text{train}} = 11.9$ and $NF_{\text{test}} = 14.8$, and the Samovar/8/1 with $NF_{\text{train}} = 13.7$ and $NF_{\text{test}} = 22.7$). On investigation, this turned out to be due to a single extremely poor prediction which both models made at the same point in the training run, contributing a log pdf hit of -381 in the first place and -514 in the second and depressing their overall NF_{train} score enough, given that no corresponding "prediction disasters" occurred in the test run, to produce this anomaly. The models' mistake is an extremely surprising one: starting from a position in which the robot's forward-pointing range sensor is reading 0.81, and the robot moves directly forwards, they predict that the new reading will be very close to 1.0, whereas in fact of course it falls further (to 0.60). Underlying this failure is an incorrect choice of mixing component ϕ. During training, the mixing component probabilities are computed using information from the future as well as from the past, while when making predictions they are perforce conditioned only on the past; normally, the learning algorithm ensures that the foresight ϕ-estimates are consistent with the hindsight ones by adjusting the transition matrix and gating patches. But the situation in question appears to be sufficiently uncommon that the algorithm can sometimes get stuck in a local maximum in which this does not happen: the reading/action combination falls well outside any of the available gating patches, and the least distant one, which is activated *faute de mieux*, is attached to an expert which yields a poor prediction. This is in fact the kind of problem which the confidence region alleviates, although it is of course motivated by the issue of novel data rather than of known data which the training algorithm fails to handle effectively.

Indeed, the anomaly just discussed does not appear in the following plot, which shows the test and train *NF* scores for the same model classes but with the confidence region enabled (model failure probability = 0.1):

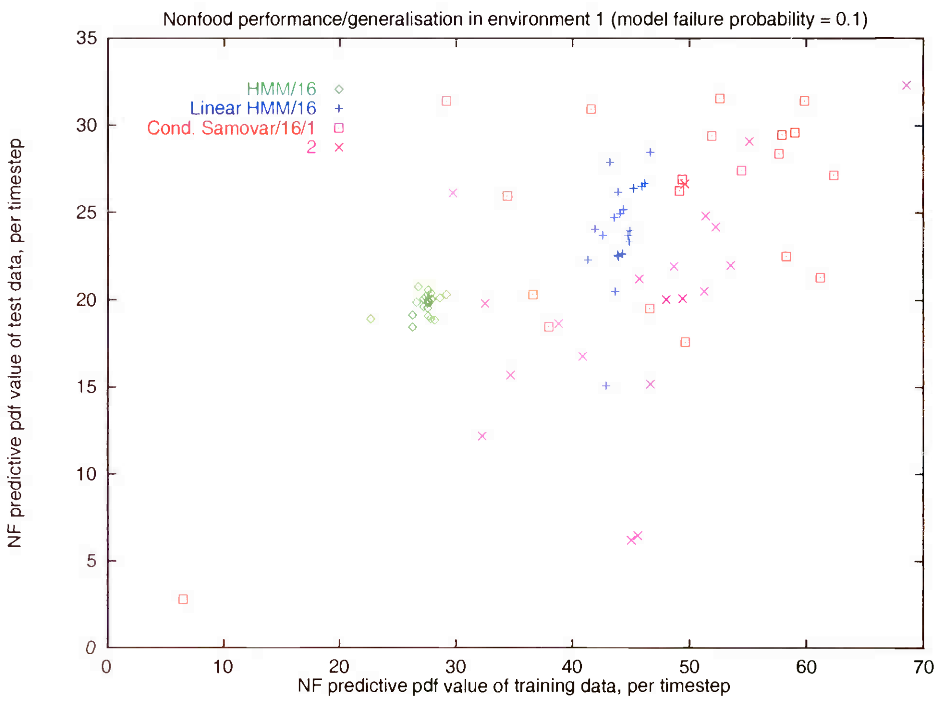


The train-test correlation has improved for all the model classes, but the test data predictive pdf values have fallen somewhat (note the change in y scale). Increasing the model failure probability to 0.5 (not shown) reduces them still further. This comes about because each component of an eight-component model must cover quite a broad range of situations, so its gating field must be significantly large compared with that of the fallback "model failure" component; hence the latter will often be judged to be active with a finite probability, and the definiteness of the predictive pdf will be tempered with its broad "don't know" output.

With sixteen mixing states available, the Samovar model achieves some good test predictive pdf values, but they are not at all correlated with the training predictive pdf values:



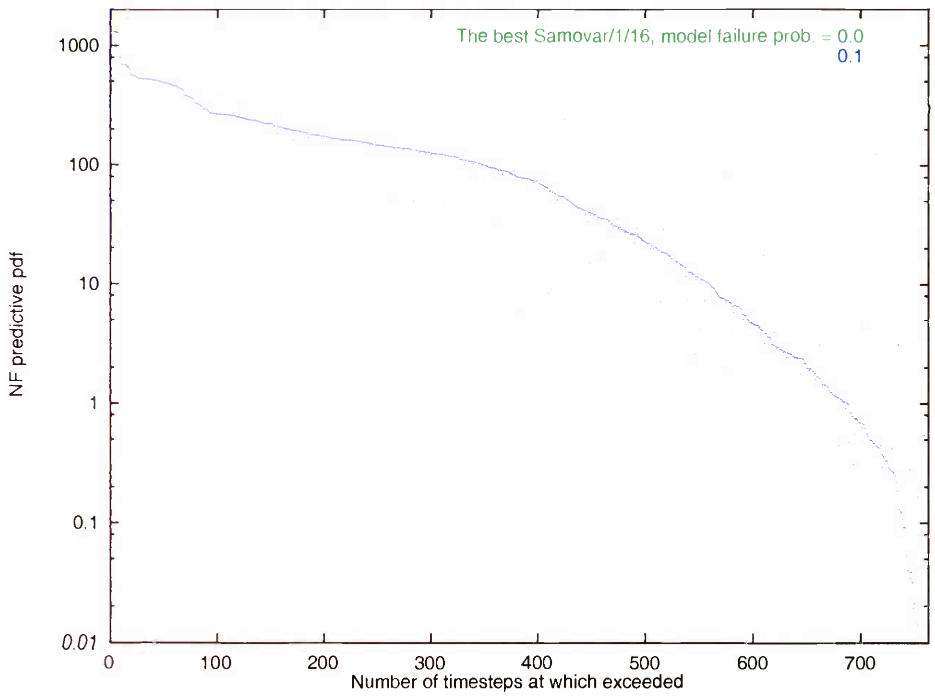
However, when the confidence region is enabled, their predictive pdf values all improve considerably, yielding the best nonfood predictions of any class of model:



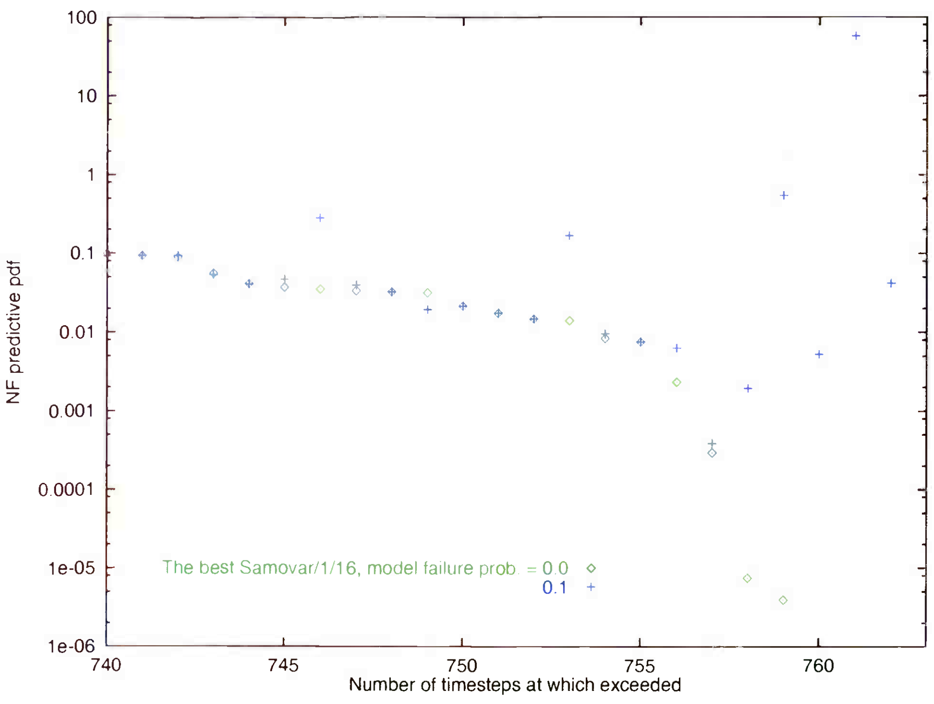
It is even (just) the case that the five most promising Samovar/16/1 models do better than the five most promising linear HMM/16 models.

The reason why the confidence region is an unmitigated plus here is that the 16-component models' experts are more situation-specific. One consequence of this is that their predictions outside the narrow areas on which they have been trained are more "precisely misleading", so that tempering them with a "don't know" predictive distribution is more of a gain; another is that their gating patches are smaller, so that the "model failure" component's broad patch interferes with them less. Increasing the model failure probability to 0.5 (not shown) lifts some of the remaining poor predictive pdf values somewhat, but otherwise has little effect.

Some insight into the effect of the confidence region can be gained by considering the profile of nonfood predictive pdf values obtained across all the timesteps in a run. The following diagram shows, simply, all the *NF* predictive pdf values from the best Samovar/1/16 model without a confidence region, arranged in descending order, together with the corresponding predictive pdf values from the same model at the same timesteps, but with $p(Q^t = 0)$ set to 0.1:



Mostly, the two predictive pdf values are identical (which is why there is no green showing in the graph—the blue has covered it up almost everywhere). Sometimes, the confidence region makes a good prediction more cautious, thereby reducing the achievable predictive pdf value (the blue dots below the main line); occasionally the effect is to improve it (the dots above the line). Note that either way, at most of the timesteps the predictive pdf value exceeds 9, which would be the maximum theoretically obtainable from a 3×3 quantising model over the $[0, 1]$ legal range of the sensors, and only 10% of the time are they very poor in the sense of falling below unity (which would be the predictive pdf value of a uniform "model" over that range). However, some of those 10% are really terrible, as can be read off the y-axis of this detail of the missing bottom right hand corner of the graph,[8]

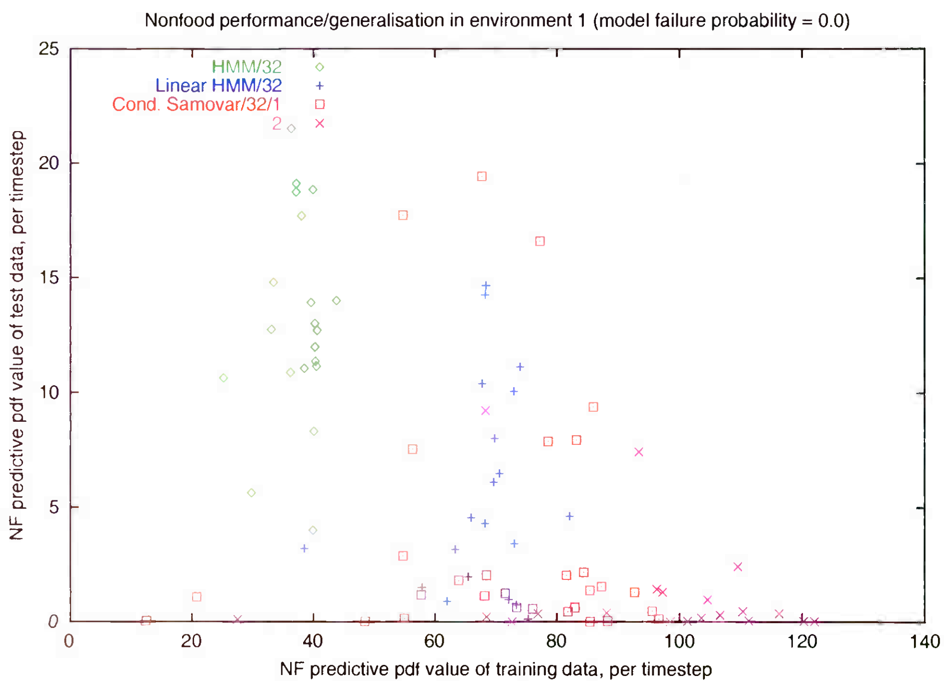


and it is here that the confidence region has its beneficial effect, detecting situations different from any in the training data, so that the fallback component cuts in with its vague "predictions" and caps the badness of all but one of the model's overall, mixed predictions. Note that the second-worst prediction is turned into quite a success, with a predictive pdf value above 56. This falls at a timestep whose immediate predecessors are handled very poorly; if the model tries to assign them its learned components, it remains confused for several timesteps, while if it can write them off as unpredictable, it recovers in time to make the good prediction noted here.

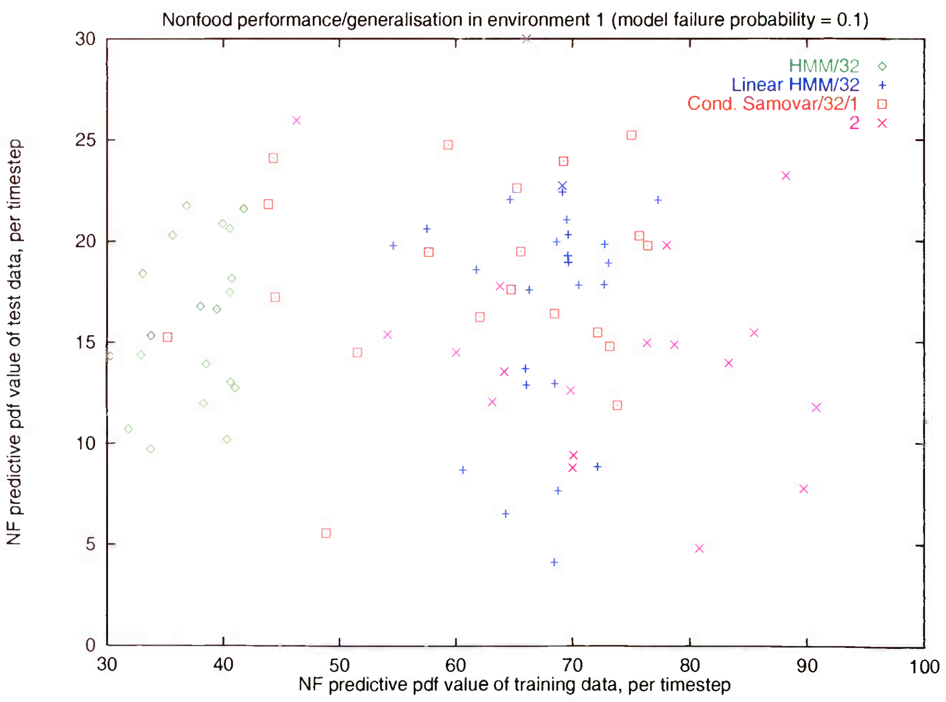
Finally, here are the promised and achieved *NF* predictive pdf values for models with

[8] The four worst predictive pdf values (not shown) made in the absence of a confidence region range from around 10^{-7} down to around 10^{-33}.

32 mixing states:



It's clear that overfitting has set in badly even amongst the plain HMM models—their predictive pdf values over the training set (x-axis) are poorly correlated with those over the test set (y-axis). The confidence region helps, but still, both the models' predictive pdf values and the correlation between what they promise and what they deliver are lower than they are when sixteen states are used:



5.4.2.2. Food

Of the seven scenarios defined in section 5.2.1.3, the ones that gave the models the most trouble were, as expected, numbers 1 and 6—the true and false food situations. A few models were also persistently inaccurate in number 4, where the robot misses the food after starting from too close to the wall. All the others were handled correctly,[9] so it is on 1, 4 and 6 that the analysis presented here concentrates.

The first group of scatterplots displays the food-prediction accuracy of all the conditional, eight-component models generated during the experiments. (Corresponding graphs for the 16- and 32-state models, and for the joint-variant Samovar models, will be given later on.) The left hand column contains the plots for models trained and tested with the model failure probability $p(Q^t = 0)$, or MFP, set to zero, which effectively disables the confidence region (section 3.3.2.6); for the right hand column, the confidence region is enabled with $p(Q^t = 0)$ set to 0.1. Each row holds the plots for models of a different class, in increasing order of complexity from plain HMMs at the top to Samovars with two-dimensional linear hidden state at the bottom.

For each of the 20 individual models θ in each class/MFP combination, a marker is placed in the appropriate scatterplot to indicate what the model's mean food prediction was in each of the scenarios under consideration, during the training run and during the test run. The colour and shape of the marker indicate a situation $i \in 1, 4, 6$, as shown in the key in the top left-hand plot, and its coordinates are

$$\big(F_i(\theta \,|\, \text{training data})\,,\; F_i(\theta \,|\, \text{test data})\big)$$

in the notation of section 5.4.1.1. Points further to the right denote higher mean food probabilities during the training run, while points further to the top denote the same during the test run. Ideally all the red +s, corresponding to scenario 1 in which the robot finds the food, should be in the top right-hand corner, while all the green $\times$s and blue $*$s, denoting mean food probabilities for situations in which in which the robot misses the food because it is in the wrong corner of the arena or starts too close to the wall, should be in the bottom left-hand corner.

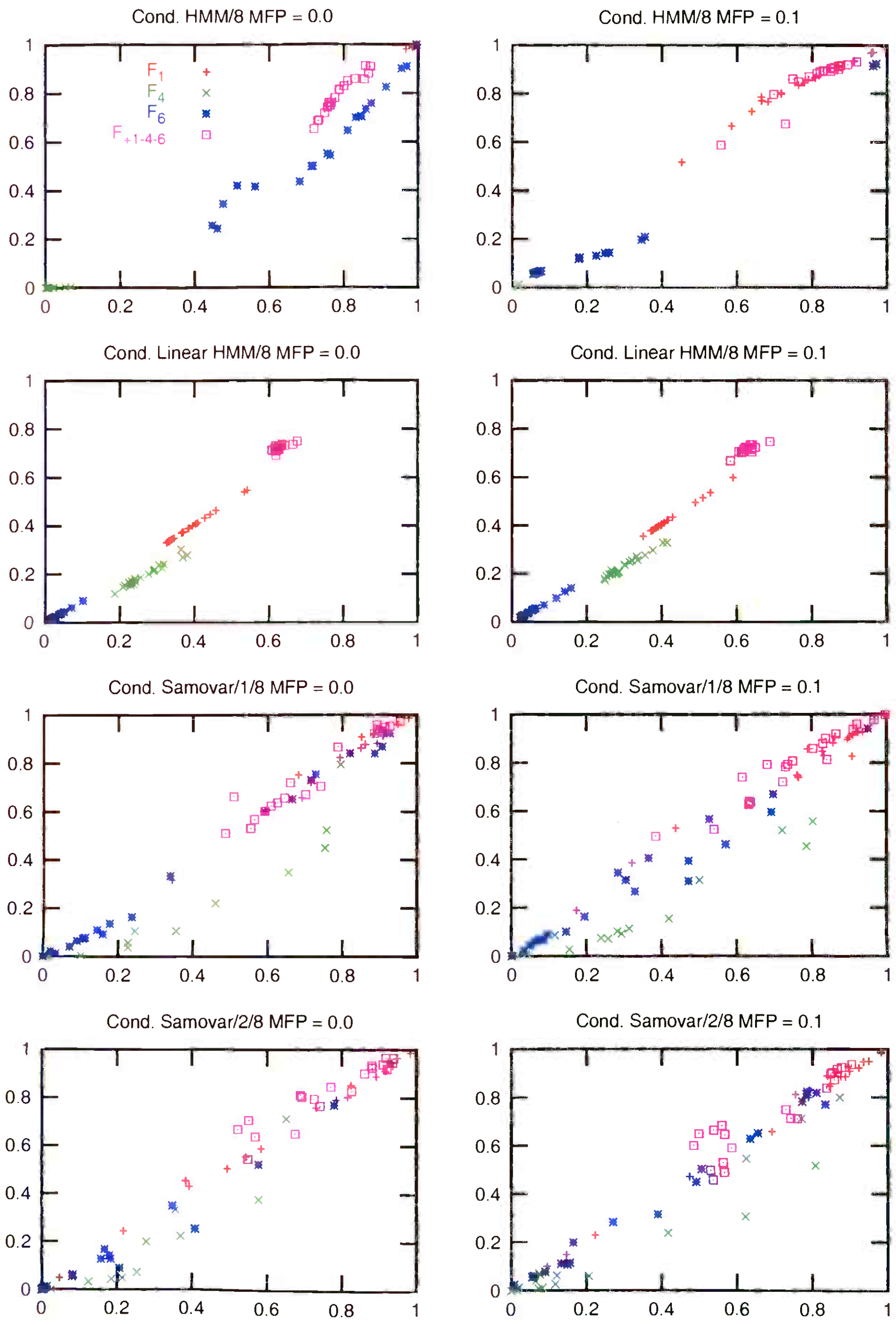
Finally, a fourth marker (purple box) is added to summarise the model's overall accuracy on these three scenarios: the probability with which it predicted the right food reading in situation 1, 4 or 6, averaged over the training run (abcissa) and test run (ordinate). This measure is called F_{+1-4-6} since the true reading is high (1.0) in situation 1 and low (0.0) in 4 and 6. The target position for its markers is the top right-hand corner,

[9] Except by the Samovar/2/32 when overfitting sets in (see below).

denoting perfect accuracy on both runs; but when interpreting the absolute value of the summary F_{+1-4-6}, it should be born in mind (a) that only the most "difficult" situations are being considered, and (b) that the proportions in which these situations occur differ between the two runs, so that there is no *a priori* reason why the value should be the same in each case.

Mean $p(r_{\mathrm{food}} > 0.5)$ *and accuracy in environment 1*

x-axis: training run; y-axis: test run



From the top left-hand plot, showing results for the reference conditionally-gated hidden Markov model 'Cond. HMM/8' without a confidence region, it can be seen that many of these simple models are able to predict food occurrences quite effectively. All the F_1 markers are clustered in the extreme top right-hand corner: the models all predict food occurrences with probability near unity every time. All the F_4 markers are clustered in the opposite corner: the models all predict, with similar accuracy, non-occurrence of the food due to the robot starting its turn from too close to the wall. However, none of the F_6 markers are correctly placed in the lower-left corner. This is because the HMM/8s are not powerful enough to handle every situation with complete success; their food sensor predictions around the "true food" situation are good, but those around the "false food" scenario 6 are not so good. As a result, their "hard case" food sensor accuracies F_{+1-4-6} mostly do not exceed 85% on the test data (y-axis). Recall too that their range sensor predictions are less informative than those of the more complex models—see section 5.4.2.1.

HMM/8s with a confidence region are often better at refraining from predicting a food event in the "false food" case, especially during the test run (in the top right-hand plot, the blue *s are lower down than they are in the top left-hand plot). This is partly an artifact of the phenomenon, already noted in section 5.4.2.1, of the confidence region cutting in frequently when applied to this class of model: the food sensor component of the "don't know" prediction is deliberately biased towards zero to reflect the general rarity of food events, so that any activation of the confidence region tends to depress the predicted food probabilities—and indeed the scenario 1 mean probabilities are also lower with the confidence region than without. Nevertheless, the effect is an overall improvement in accuracy which is quite pronounced in the test run (the purple boxes are higher up), which suggests that the confidence region may actually be playing a positive role. Increasing the model failure probability further (not shown) reduces the predicted food probabilities again, and this time also the net accuracy.

As an aside, note that the scenario 6 predictions of the HMM/8s are on average more accurate (*i.e.* lower) in the test run than in the training run. The reason for this that the test run happens to include three steps, out of the 10 assigned to scenario 6, which fall on the borderline between scenarios 6 and 8: the robot would in fact have turned onto the food if the food was there, but only just. The training run only includes two such steps out of 16. Since in these marginal cases the model has an immediately obvious reason (the distance to the wall) to discount somewhat the possibility of encountering the food, it will tend to handle them better than more central scenario 6 steps—albeit for the wrong reason. The net effect is that the food prediction accuracy on the test run is biased upwards relative to that on the training run. Similar effects can be seen in some other plots presented in this section, although there is no clear pattern: in general, different models behave in surprisingly different ways on the two data sets, probably because they have found fundamentally different representations for solving the prediction problem.

Turning now to the second row of the graph, it can be seen that the linear HMM/8s perform much less well on the food prediction task than the plain HMMs. For instance, they fail to produce unequivocal predictions of food encounters in scenario 1 (the red crosses are not in the top right hand corner); and the confidence region does not help. It seems that these models are concentrating on predicting the range sensors accurately—a task which they can perform well in this environment (section 5.4.2.1)—to the detriment of their food sensor predictions.

Some of the Samovar/1/8s (third row) are able to predict the food occurrences almost perfectly (some purple boxes in the extreme top right-hand corner). The reason they can do better than the plain HMMs turns out to be that each of their mixing states can account for a wider range of range-sensory phenomena, leaving more for mapping out the approach to the food area. It turns out that the average *NF* range sensor predictive pdf value achieved by the five best Samovar/1/8s, chosen according to food prediction accuracy, is, at 17.2, also better than that achieved by the plain HMMs (section 5.4.2.1): some of these models are able to perform both prediction tasks at once.

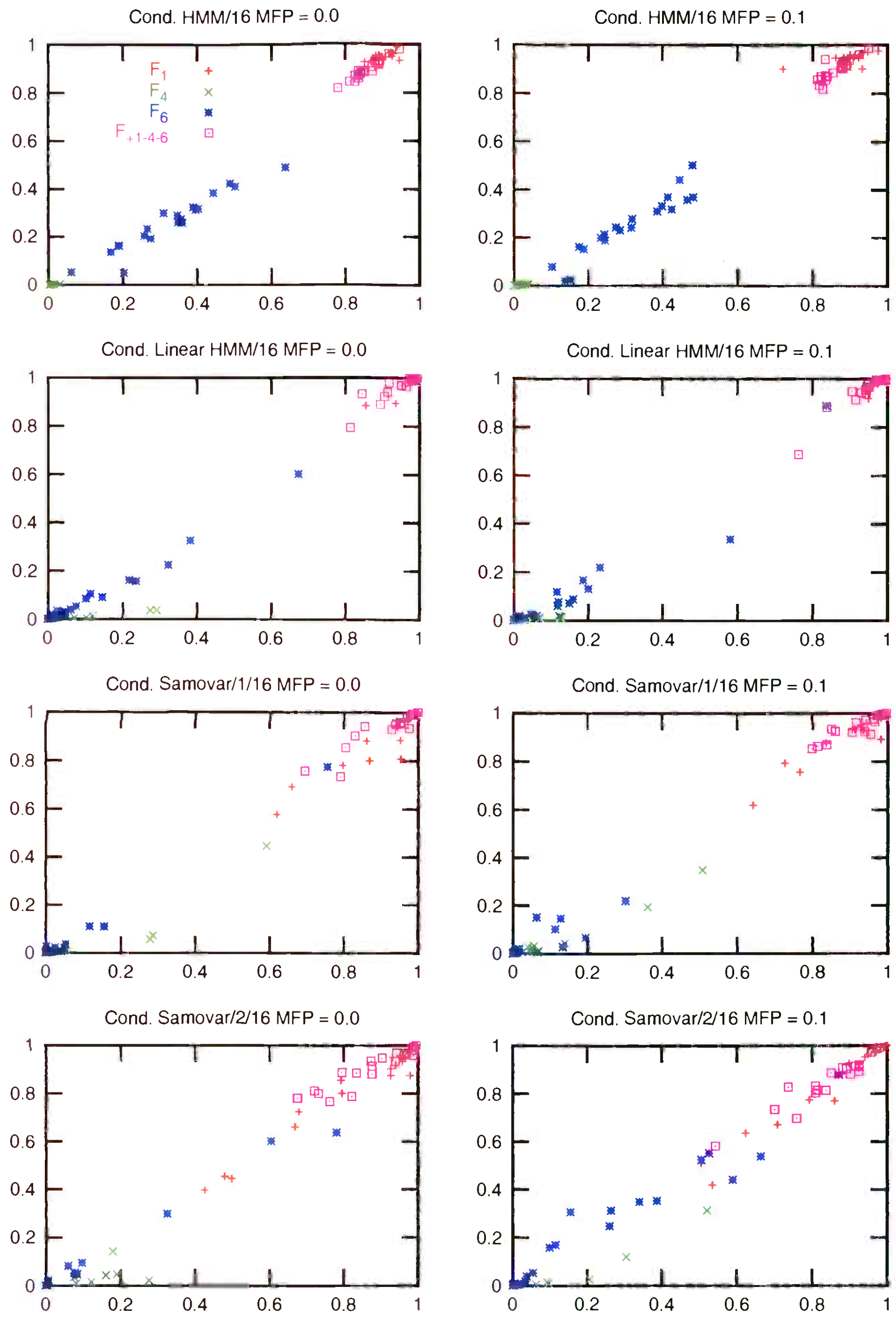
However, the intra-class variability is considerable, and some are actually worse than the plain HMMs; in order to obtain a single Samovar/1/8 which predicts the food sensor readings well, it is necessary to train several and choose the best. At least the correlation between the models' performance on the training data and their performance on the test data is strong, so that the choice can be made with some confidence.

The Samovar/2/8s, and the Samovar/3/8s (not shown), do not yield such good results as the Samovar/1/8s: they are powerful enough to be prone to overfitting.

Turning now to the 16-component models, the main feature to emerge from the scatterplots is that their food predictions are generally better than those of the eight-component models; they have enough states to map out the entire coarse structure of the environment:

Mean $p(r_{\text{food}} > 0.5)$ *and accuracy in environment 1*

x-axis: training run; y-axis: test run



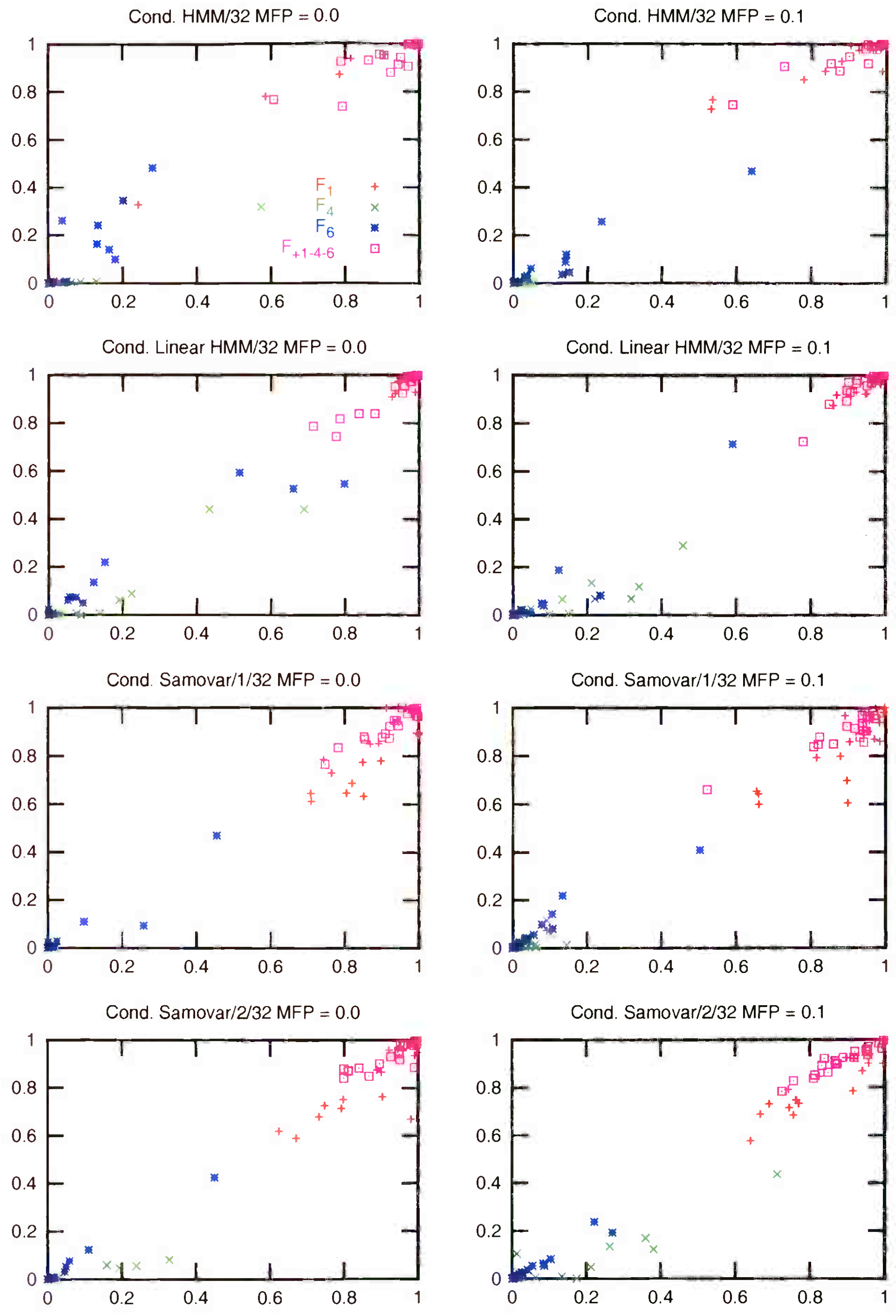
The Samovar/1/16s suffer from a slightly higher intra-class variability than the simpler models. But with the confidence region enabled, they may again have a slight edge when it comes to combining good food predictions with good range sensor predictions: the five food-best Samovars with a confidence region return an average *NF* score of 26.83, against 23.79 for the linear HMMs.

Interestingly, the even greater intra-class variability of the Samovar/2/16s is not helped by the confidence region but rather made worse: from the bottom right-hand plot it can be seen that more of them make false positive food predictions in scenario 6. This apparently comes about when the confidence region intervenes to prevent a model from making an over-precise range sensor prediction, and effectively resets its hidden state—including its memory of where it is in the arena.

Just as in the *NF* scatterplots (section 5.4.2.1), although to a lesser extent, there is evidence of overfitting in the food-prediction plots for the 32-component models: the hitherto tight relationship between F_{+1-4-6} performance on the training and test data sets has weakened, especially in the case of the Samovar/1/32:

Mean $p(r_{\text{food}} > 0.5)$ and accuracy in environment 1

x-axis: training run; y-axis: test run

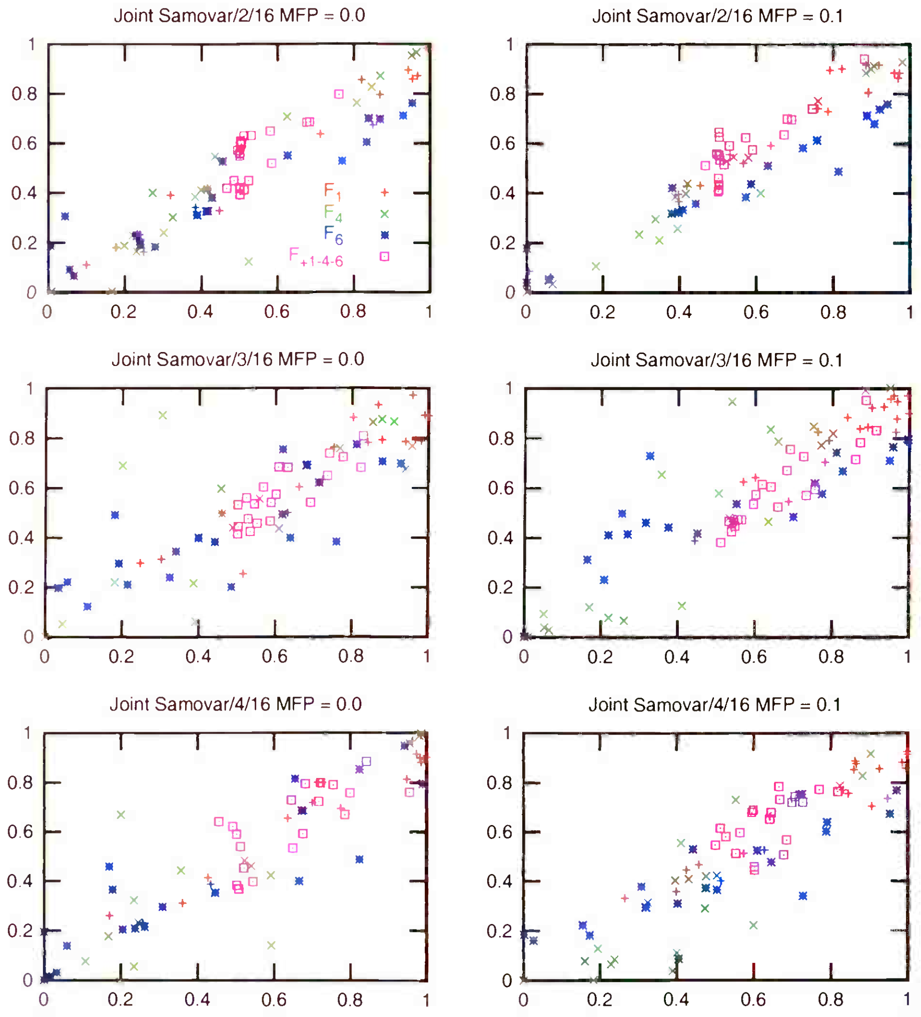


In fact some of the Samovar/2/32s and Samovar/3/32s even make unusual false positive predictions for scenario 3 (not shown).

The joint Samovar model was unable to get as good a grip on the food occurrences as the conditional models with fewer than 32 mixing components at its disposal. In fact it's easy to see that the "off by one" nature which the conditional model acquires when the output function is assimilated into the dynamics (section 4.2.1.1)—with the mixing state at time t directly determining the sensor readings at time $t + 1$—is advantageous when it comes to storing a fact over several timesteps. Effectively, the conditional models only have to learn to remember the landmark notch of 110 for three timesteps, while the joint models have to remember it for four. Sixteen components do not, therefore, seem to be enough to get the true positive food probabilities up in situation 1 and the false positives down in situations 4 and 6:

Mean $p(r_{\text{food}} > 0.5)$ *and accuracy in environment 1*

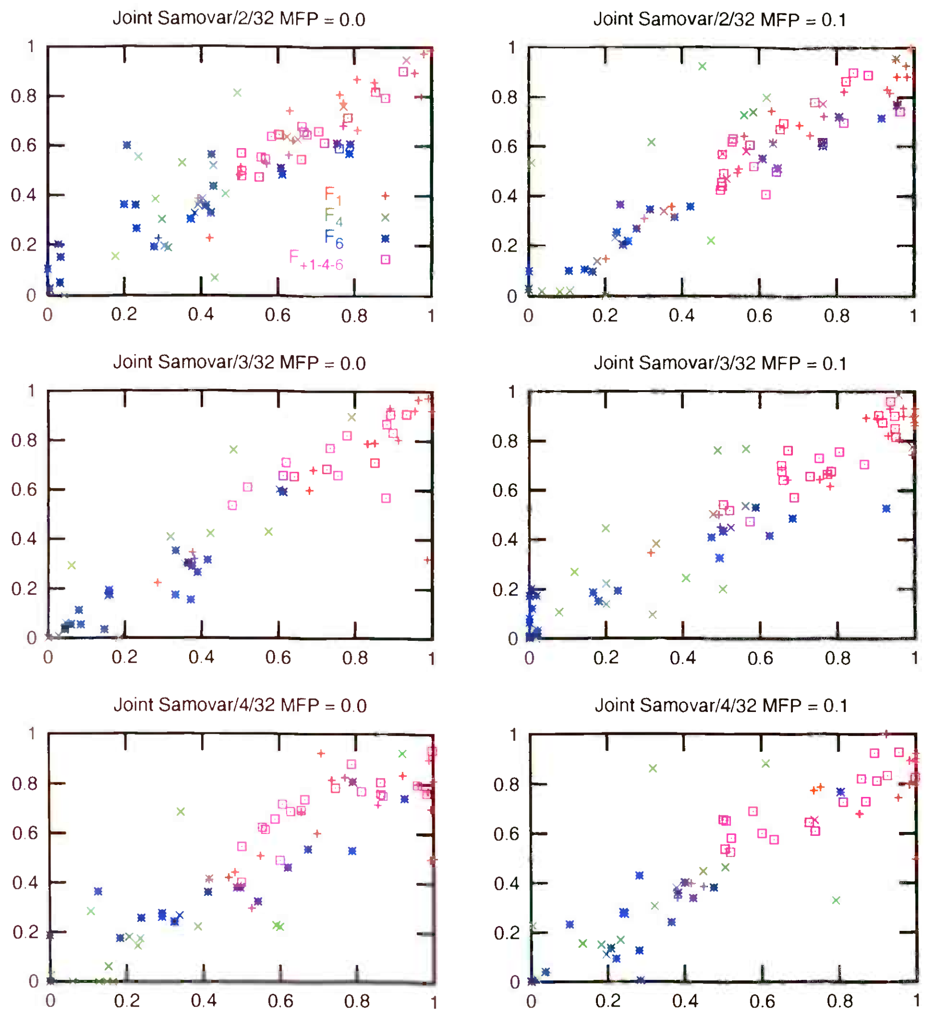
x-axis: training run; y-axis: test run



Even with 32 discrete states in play, three dimensions of linear hidden state are needed finally to achieve a few F_{+1-4-6} scores above 90%:

Mean $p(r_{\text{food}} > 0.5)$ *and accuracy in environment 1*

x-axis: training run; y-axis: test run

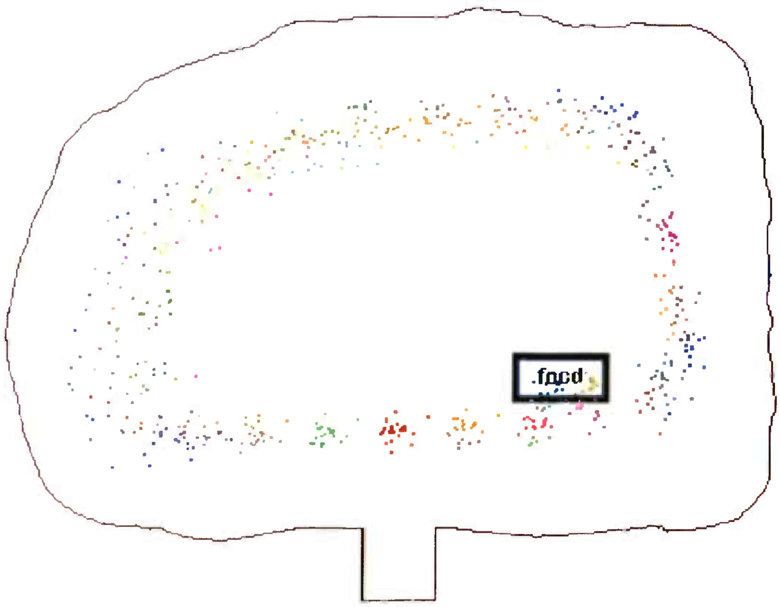


5.4.2.3. Qualitative

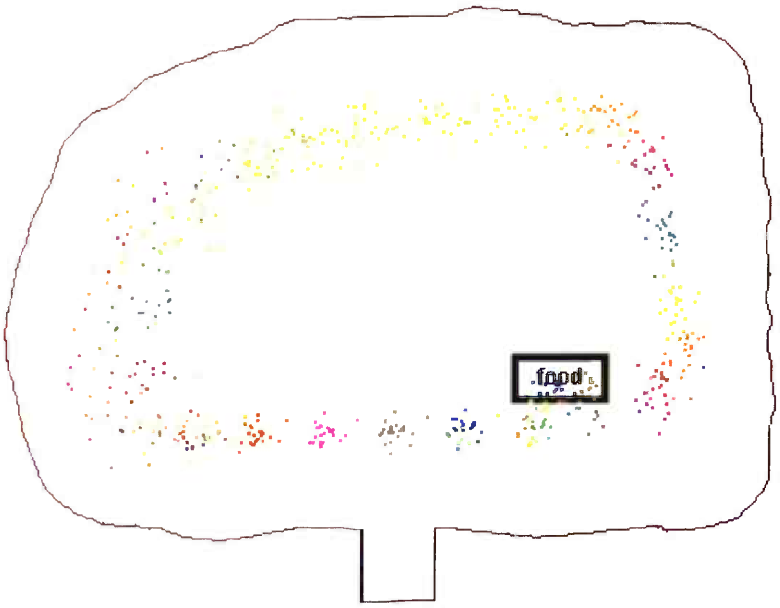
It's interesting to look at the way each class of model categorises the situations it encounters in the simulated robot environment (*i.e.* the way it distributes them between its components). The diagrams that follow show by mean of colour coding[10] the component judged most probably to have been active at each position the robot occupied, in retrospect

[10] apologies to those reading in black and white

(conditioned on the whole data set). Here is the best sixteen-component HMM, measured by NF:



And for comparison the NF-best conditional Samovar/16/1 ...



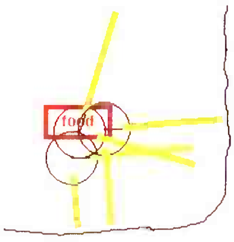
(116)

Both models classify the steps covering the "notch" landmark uniquely or almost so. However, the Samovar is able to expend fewer states on handling the more uniform parts of the arena—for instance, it doesn't have to devote separate components to the situations where it is near and far from a flat wall—leaving it enough left over to mark out uniquely the region between the notch and the "food"[11]. The HMM shown here can't, in fact, predict

[11] it's using the structure shown in *c.f.* (115)

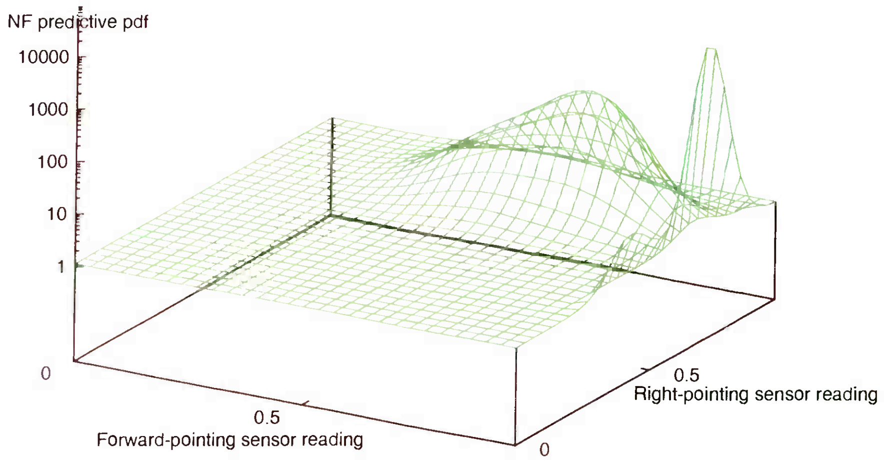
the food sensor readings accurately (although others of the same size, selected by their promise on this task, could—see section 5.4.2.2). Corners are also treated more or less regularly: note the sequence grey-blue, purple, orange, yellow in (116). The *NF*-best linear HMM showed a similar pattern, though not quite as clean.

Although environment 1 offered no opportunity for the Samovar models to employ their linear hidden state for Kalman-style speed estimation, they did manage to discover some useful things to do with it. For example, in the following scenario ...



it is obviously advantageous for the robot to remember how far away from the wall it was before it turned onto the food and lost contact with it, so that it knows what the right-pointing range sensor reading will be when it turns back. Analysis of the way the V-related blocks of the dynamics matrices stacked up (given, of course, the ordering of the components used in the most probable Q-sequences), followed by empirical testing, showed that a Samovar/1/16 model was indeed making this inference: changing the initial wall-distance sensor reading over the range 0.4–0.7 yielded an exactly matching variation in the (highly confident) reading predicted two steps later. A similar encoding was used to remember the distance from the wall while the robot passed through the landmark "notch" (at which point the right-pointing sensor was generally returning an uninformative 1.0). Various other apparently significant relationships were apparent, especially in the approach to the top right-hand corner of the arena, but it was not possible to interpret them with any confidence.

The models' predictions were found sometimes to be significantly multi-modal; for instance, the following graph shows the predictive density obtained at a particularly confusing timestep from a Samovar/1/16 model:



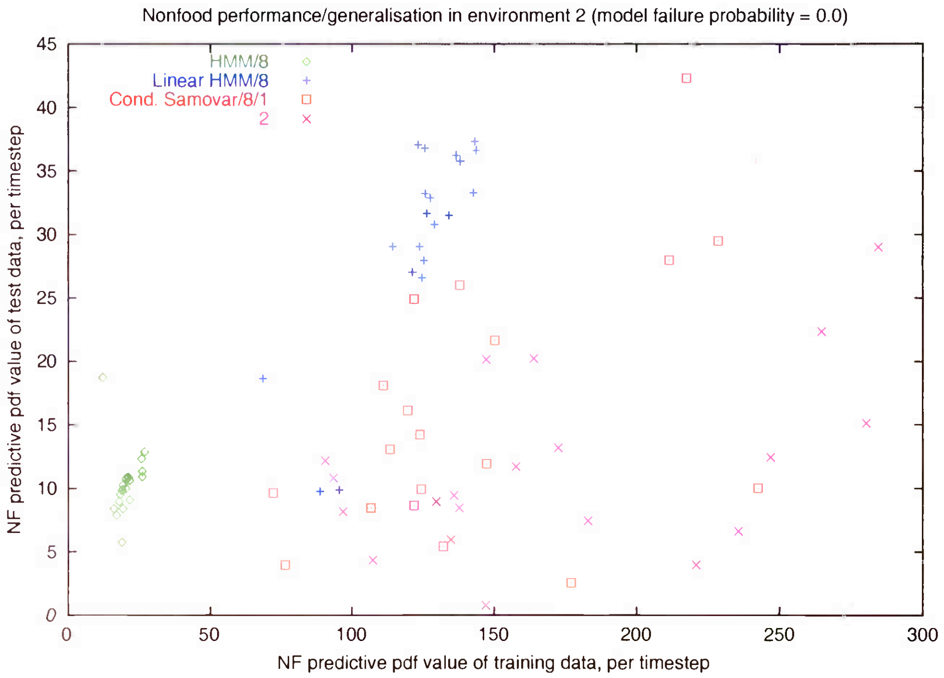
The peaks in the graph indicate several well-separated hypotheses about which combination of forward- and right-pointing range sensor readings seem plausible on the basis of the experiences leading up to this timestep. Each mode corresponds to a possible mixing state of the model.

5.4.3. Environment 2

The story is fairly similar in the other environment (section 5.2.2), except that, as expected, the element of momentum in the robot's dynamics gives an advantage to the Samovar models which are able to represent it more or less exactly.

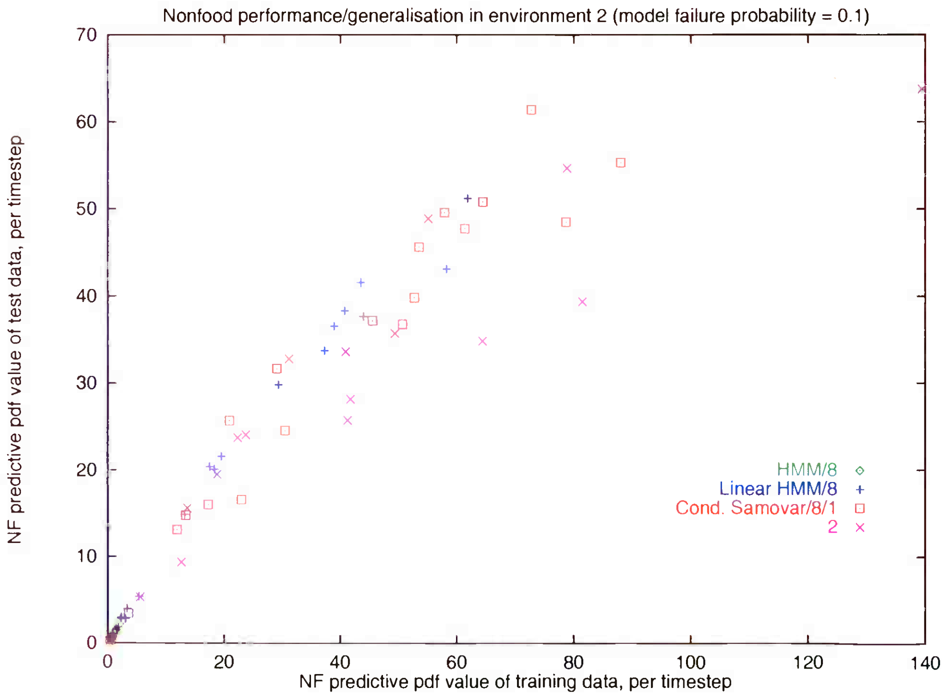
5.4.3.1. Nonfood

Surprisingly, the *NF* predictive pdf values achieved by the conditional models in environment 2 are very different on the test as against the training data even with as few as eight components:



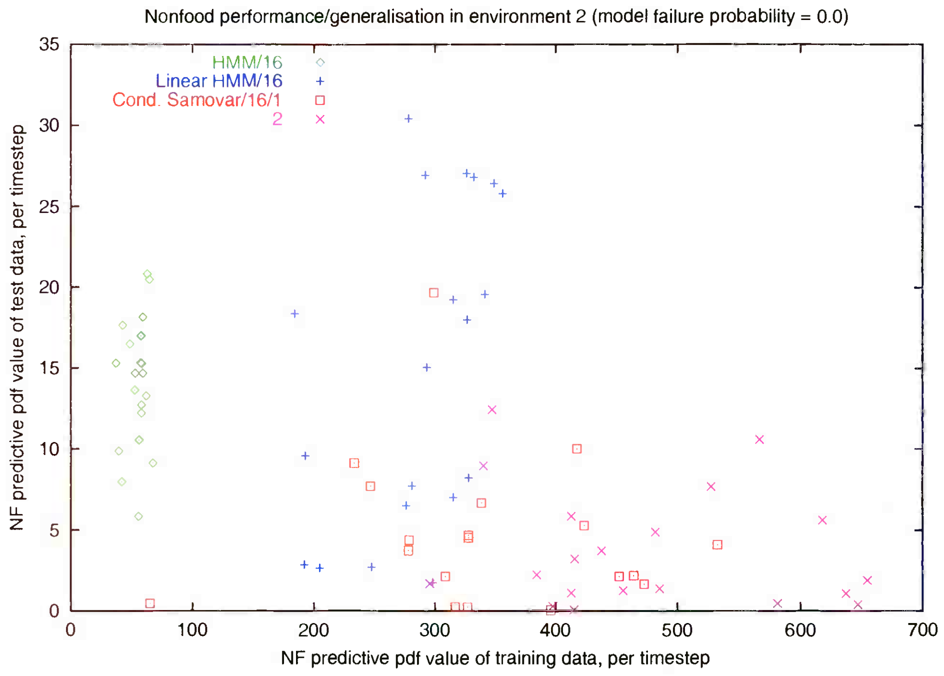
(117)

This applies even to the linear HMMs, which promise an *NF* of around 140 and deliver around 35. One possible reason for this is that they have relied (as they must) on piecewise approximation to capture the effect of the robot's speed on the development of its sensor readings, and therefore fail to generalise naturally to the test run in which the robot's trajectory is sometimes novel. When the confidence region is enabled, the test predictive pdf values of the linear HMMs and Samovars improve:

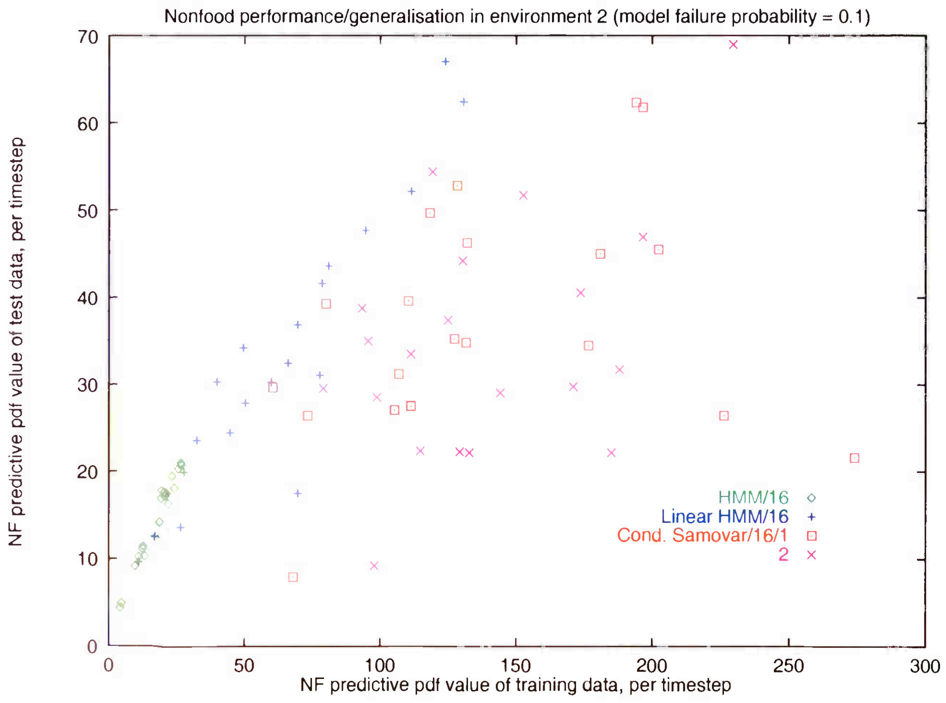


Note that the plain HMMs react very badly to the confidence region: there is no way for them to understand this more complicated environment, so their components hardly specialise, their gating patches are large, the confidence region takes over too often, and the resulting predictive pdf values are low.

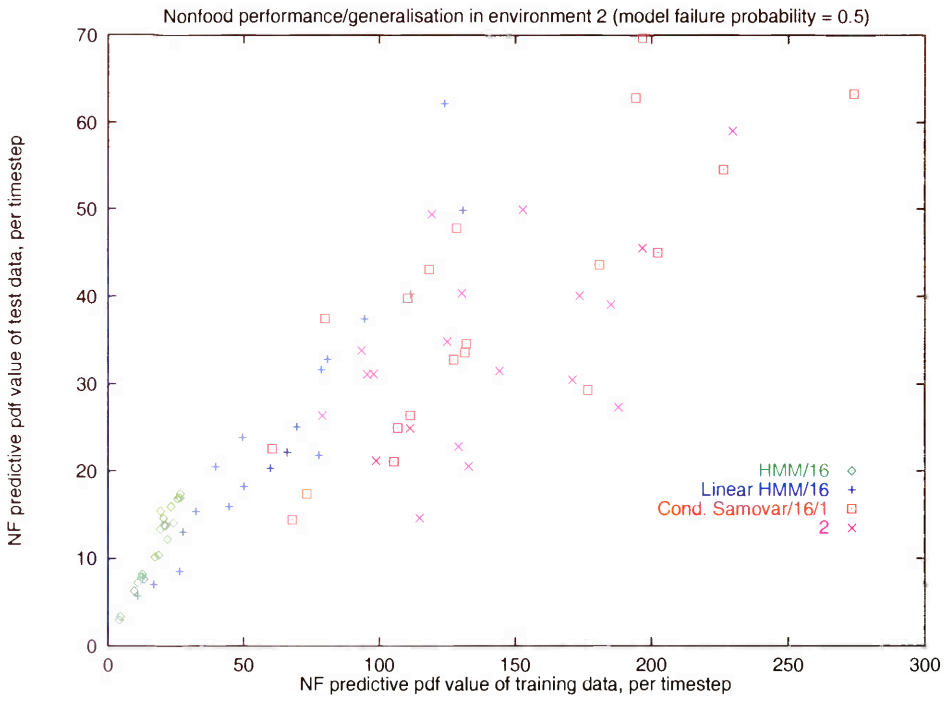
The absolutely highest nonfood predictive pdf value for environment 2 is obtained from a Samovar/16/1. However, these models depend heavily on an aggressive confidence region; with the confidence region disabled, most of their predictive pdf values are very low:



and even with a model failure probability of 0.1, their training-set predictive pdf values correlate poorly with their test-set predictive pdf values, which are in any case no better than those of the linear HMMs:

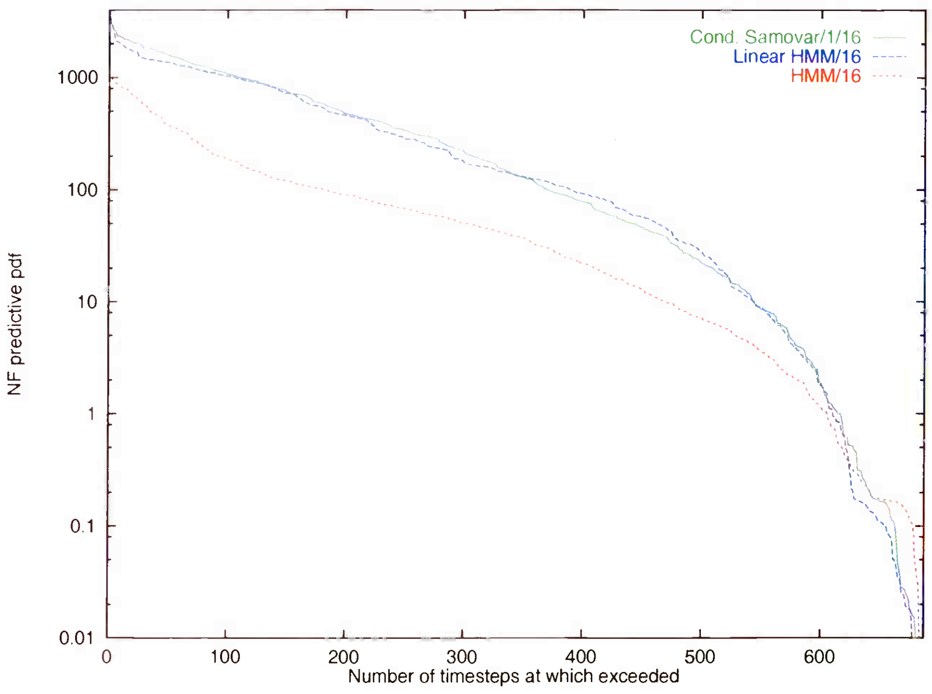


It is only when $p(Q^t = 0)$ is raised to 0.5 that the Samovar/1/16s do well . . .



. . . and even then the *NF* predictive pdf value of the five most promising Samovar/16/1s is only slightly higher, at 58.35, than that of the five most promising linear HMM/16s (with

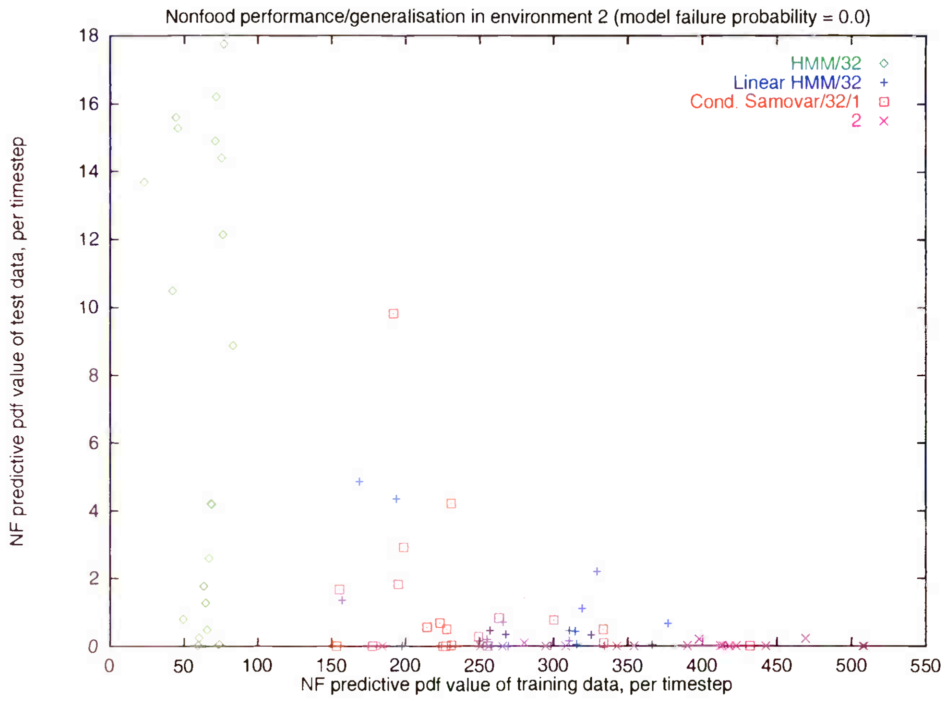
$p(Q^t = 0) = 0.1)$, at 53.75; it's clear from the graphs that this difference is unlikely to be robust. A plot of the profile of all predictive pdf values obtained over a single run by each of three different models ...



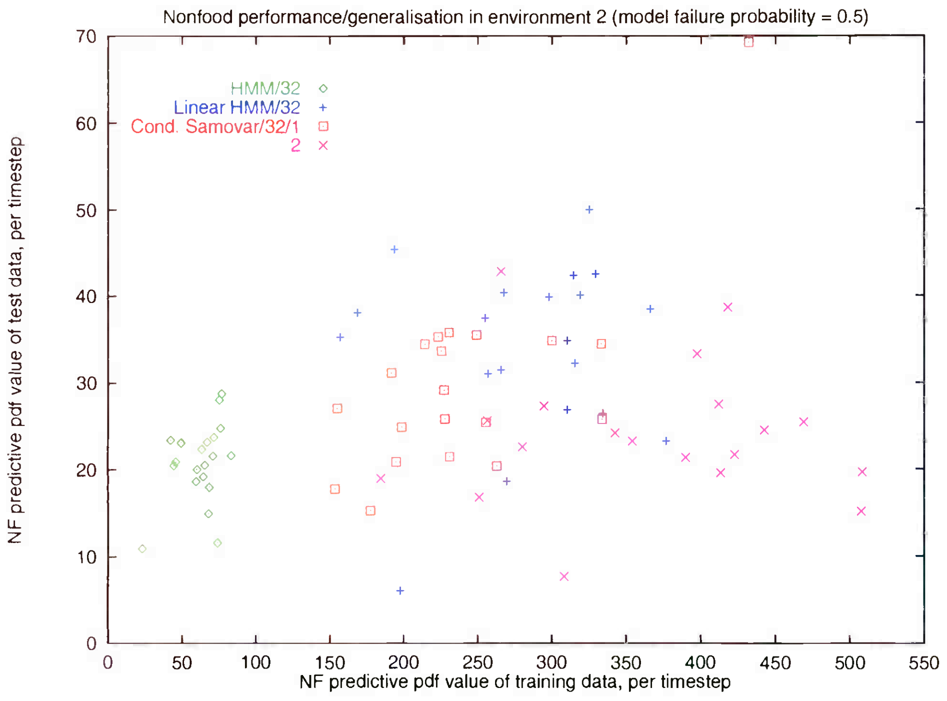
... shows the Samovar achieving even higher predictive pdf values than the linear HMM at the top end (left), and also producing fewer poor predictions with predictive pdf values in the range 0.1–1 (right), but losing out in the middle. The vanilla HMM, shown for comparison as the lowest curve, produces the fewest high predictive pdf value predictions and also the fewest very low predictive pdf value ones. (The models used to produce this plot were the *NF*-best in their respective classes.)

32 states seem to be too many; generalisation fails completely in the absence of a confidence region, with models of all classes except the plain HMMs reporting very high

predictive pdf values on the training set and very low ones on the test set ...

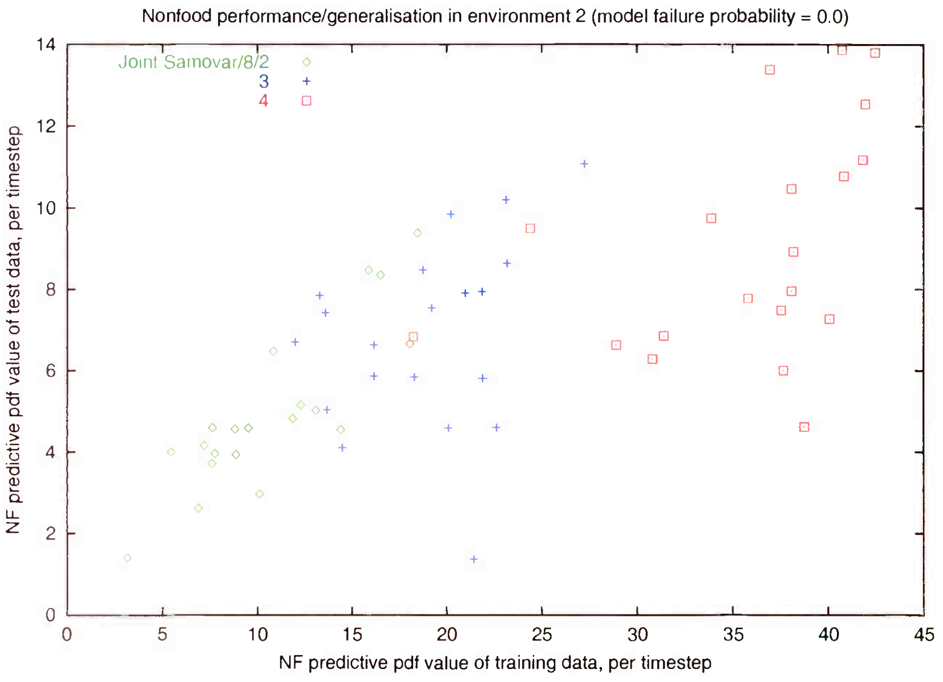


... and even with an aggressive model failure hypothesis, nearly all the models turn out to be mediocre:



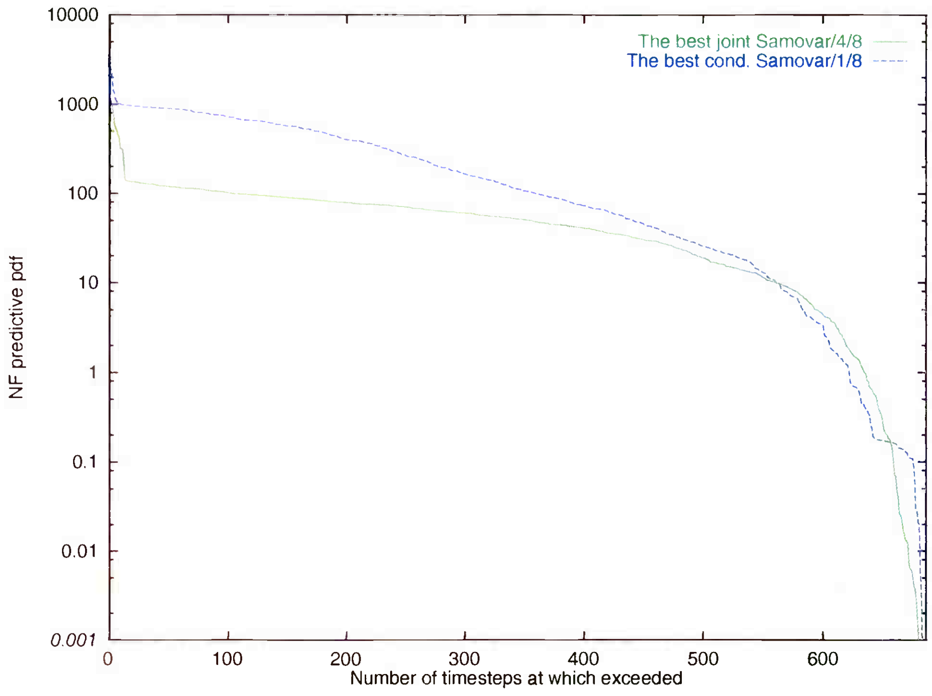
As in environment 1, the joint models achieve considerably lower *NF* predictive pdf values than the conditional ones, and require a higher-dimensional linear hidden state.

However, they are not subject to the same degree of generalisation failure with eight components (*cf.* (117)), perhaps because the reading/action density modelling by which the components' activation probabilities is judged leaves less room for poor optimisation than the conditional models' gating rule (section 5.3.1.4):



Adding a confidence region (not shown) improves both the train/test correlation and the test predictive pdf values. When more components are added, the models achieve higher predictive pdf values at some steps, but this gain is more than offset by an increase the number of steps at which poor predictions are made.

Comparison of the predictive pdf value profiles for conditional Samovar/8/1 and joint Samovar/8/4 shows that the latter produces if anything a slightly larger number of genuinely informative predictions, but fewer very precise ones and more extremely poor ones:



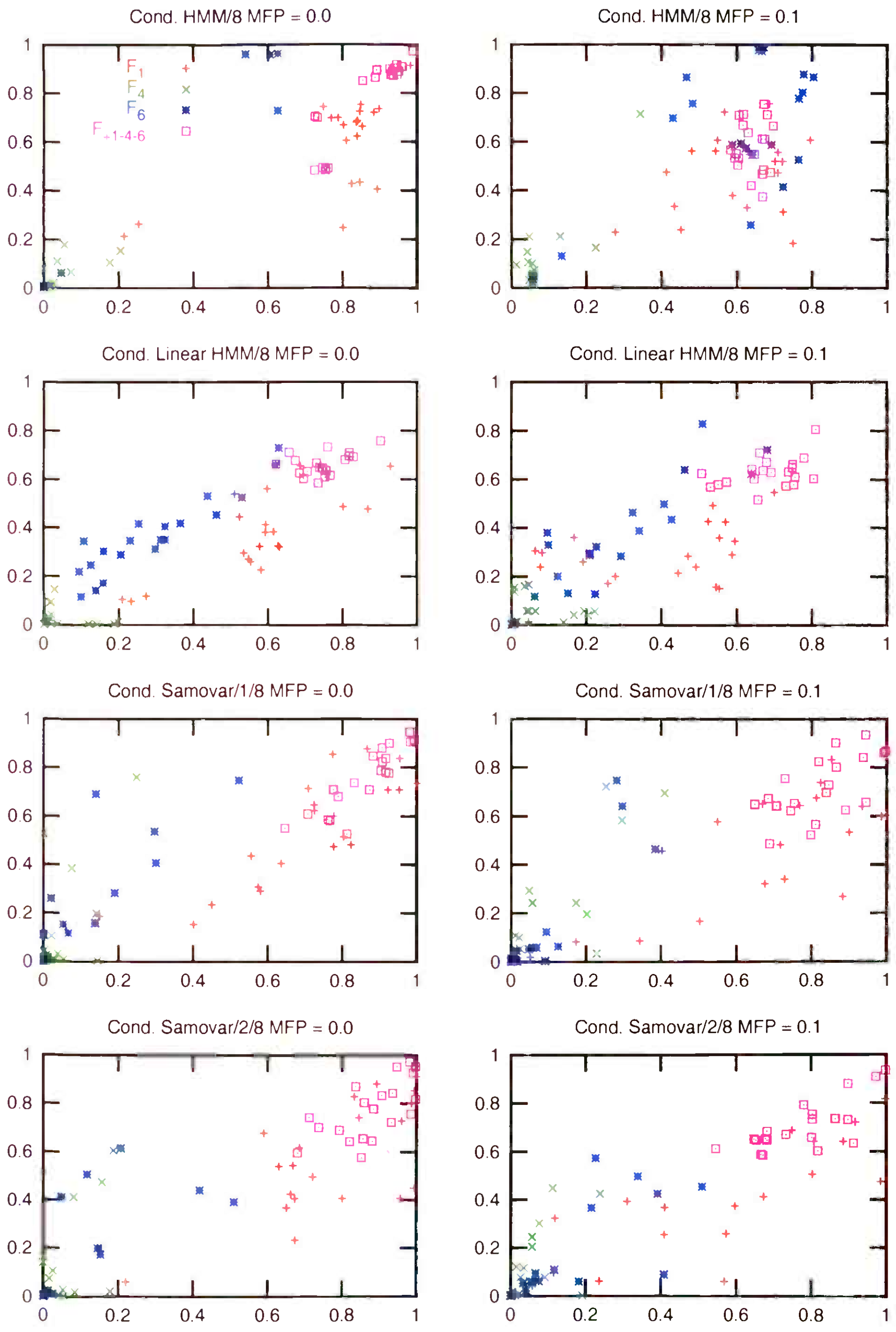
The model failure probability was fixed here at 0.1, but the same general pattern holds for different $p(Q^t = 0)$ and different numbers of mixing states. It's easy to account for the lower cap on the joint model's best predictive pdf values as a side-effect of the fact that all of its inferences must be channelled through a linear dynamics noise process as well as an output noise process, both with nonzero (and indeed regularised) variance. The joint model's increased propensity to make very misleading predictions arises from a relative inability of its confidence region to truncate them—the predictive pdf values grouped in an obvious ledge around 0.25 in the data from the conditional model reflect timesteps at which the prediction comes purely from its fallback component, and the data from the joint model lacks this feature. What the joint model appears to be doing is exploiting the actions' and readings' dependence on the linear hidden state (section 4.2.3.1) to make one of its learned components fit the former by adjusting the latter; the noise variances provide sufficient flexibility for this mostly to seem preferable to the alternative of invoking the model failure hypothesis.

5.4.3.2. Food

Although the number of timesteps over which the models have to remember the landmark "tongue" in environment 2 is smaller than the time distance between the "notch" and the food in environment 1, the less regular shape of the enclosure in the area around the food and greater variability in the trajectories with which the robot approaches it makes it very difficult to achieve the near-perfect food predictions of section 5.4.2.2:

Mean $p(r_{\text{food}} > 0.5)$ *and accuracy in environment 2*

x-axis: training run; y-axis: test run

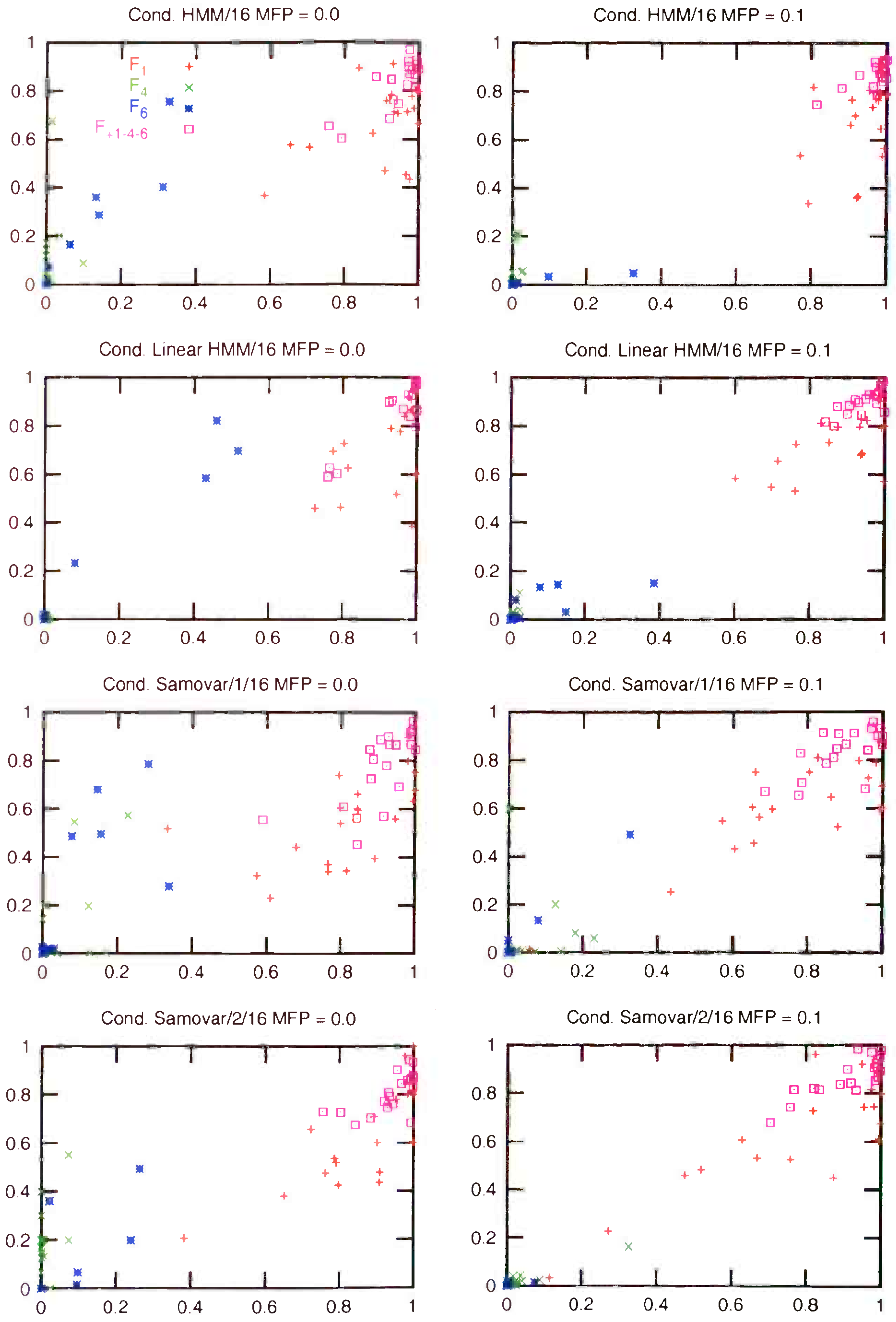


Some of the HMM/8s account well for the food sensor readings, but (as was seen in section 5.4.3.1) their range sensor predictions are very poor, whereas when the confidence region was enabled, the five best Samovar/2/8s as measured by food prediction accuracy achieved an average *NF* score of 41.35.

When the models are given sixteen components, many of them achieve food sensor accuracies above 90%. Note that the confidence region (right hand column) helps all the models cut down on false positives in scenario 6 and distinctly improves the net F_{+1-4-6} accuracy of all except the plain HMMs:

Mean $p(r_{\text{food}} > 0.5)$ *and accuracy in environment 2*

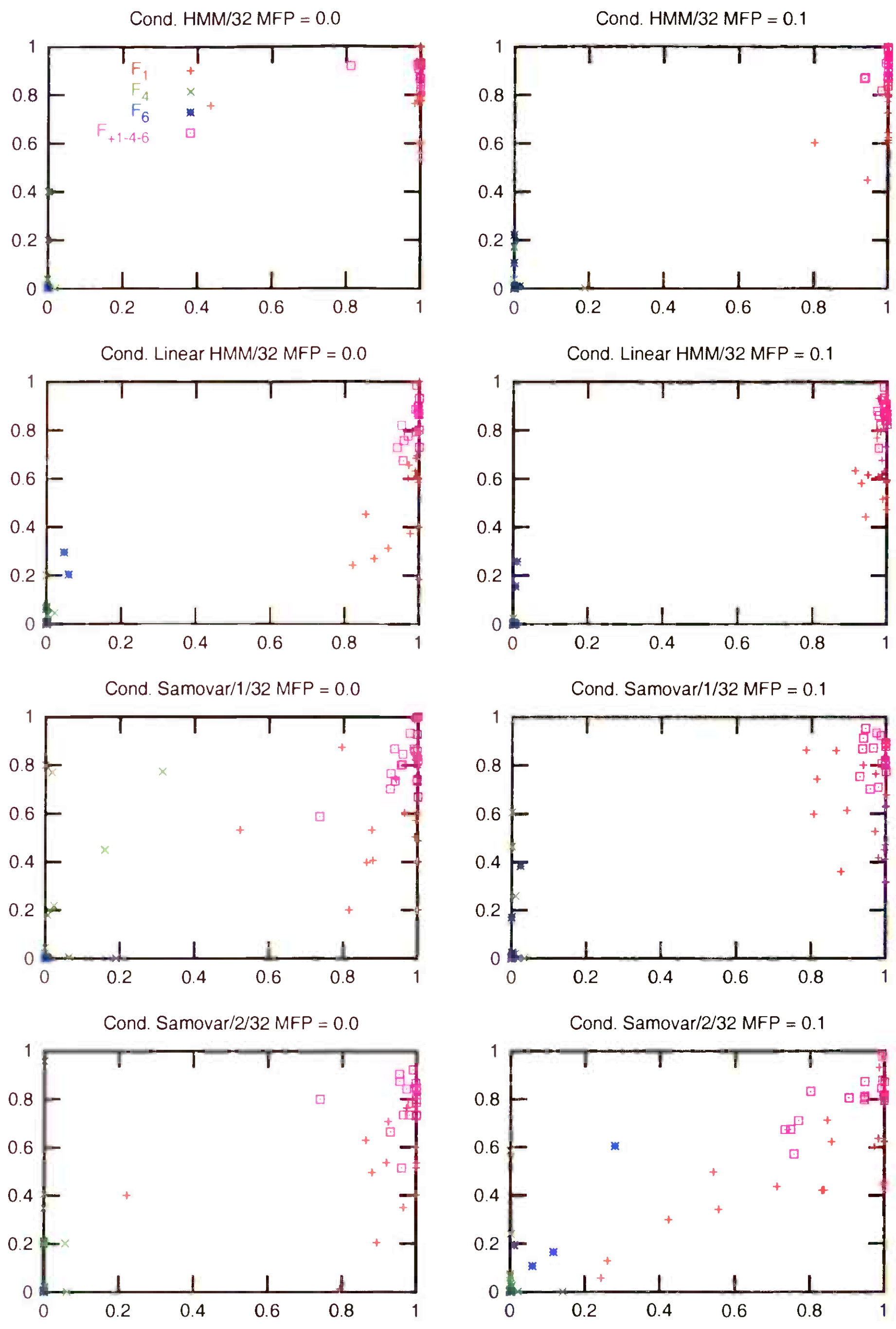
x-axis: training run; y-axis: test run



With 32 components, overfitting has set in, with most of the models promising near-perfect predictions on the training data and delivering relatively poor ones on the test data:

Mean $p(r_{\text{food}} > 0.5)$ and accuracy in environment 2

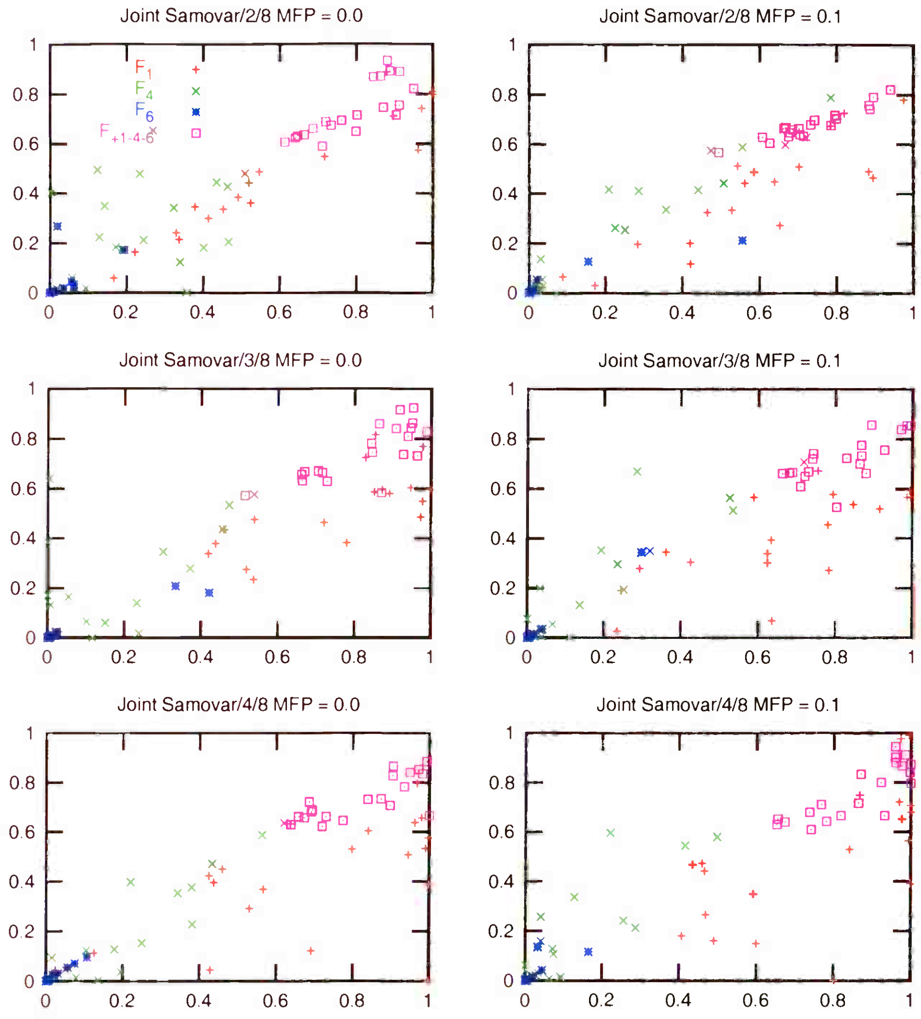
x-axis: training run; y-axis: test run



Food prediction in environment 2 proves less challenging for the joint Samovar models than it does in environment 1, but most of the joint models of whatever size suffer from generally low food predictions in scenario 1, and false positives in scenario 4: this appears to be because their gating rule is less discriminating that that of the conditional models, so that they are unable to use range sensor readings to judge precisely whether they will land on the food. Nevertheless, the best models in each class offer reasonable accuracy, and a few achieve F_{+1-4-6} rates above 90%:

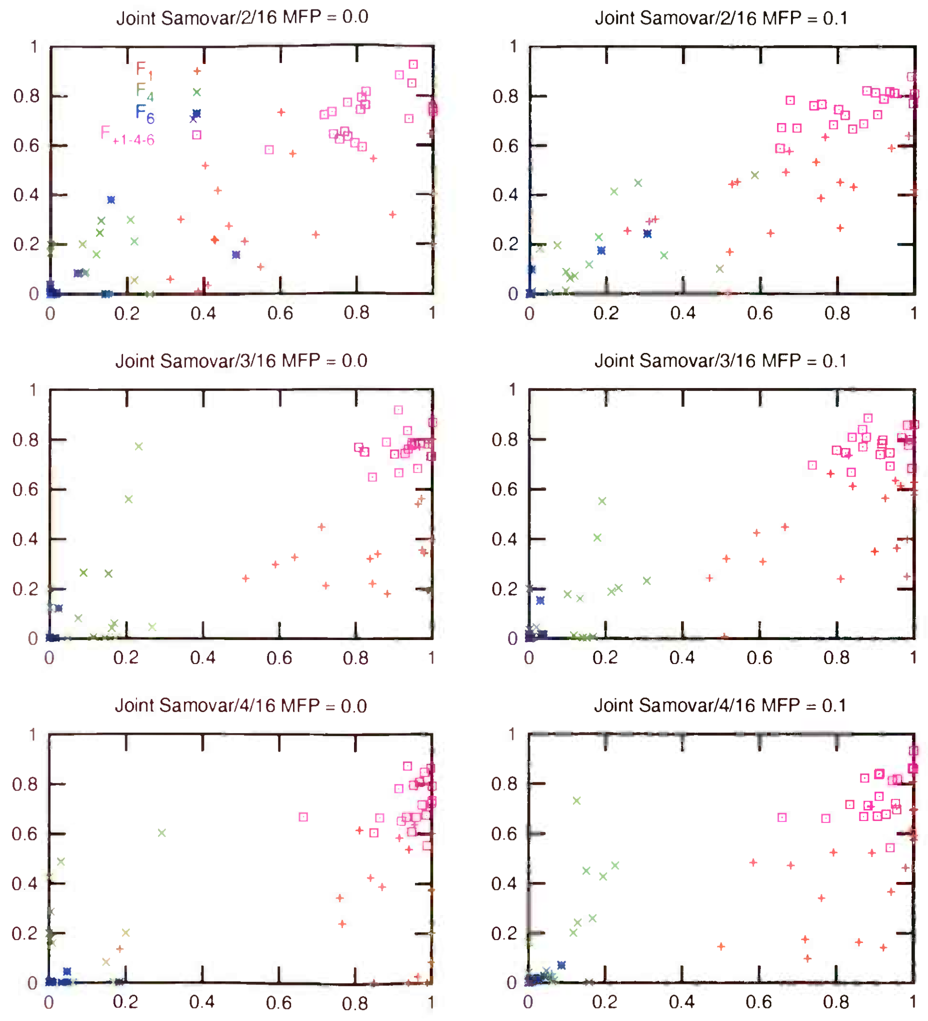
Mean $p(r_{\text{food}} > 0.5)$ *and accuracy in environment 2*

x-axis: training run; y-axis: test run



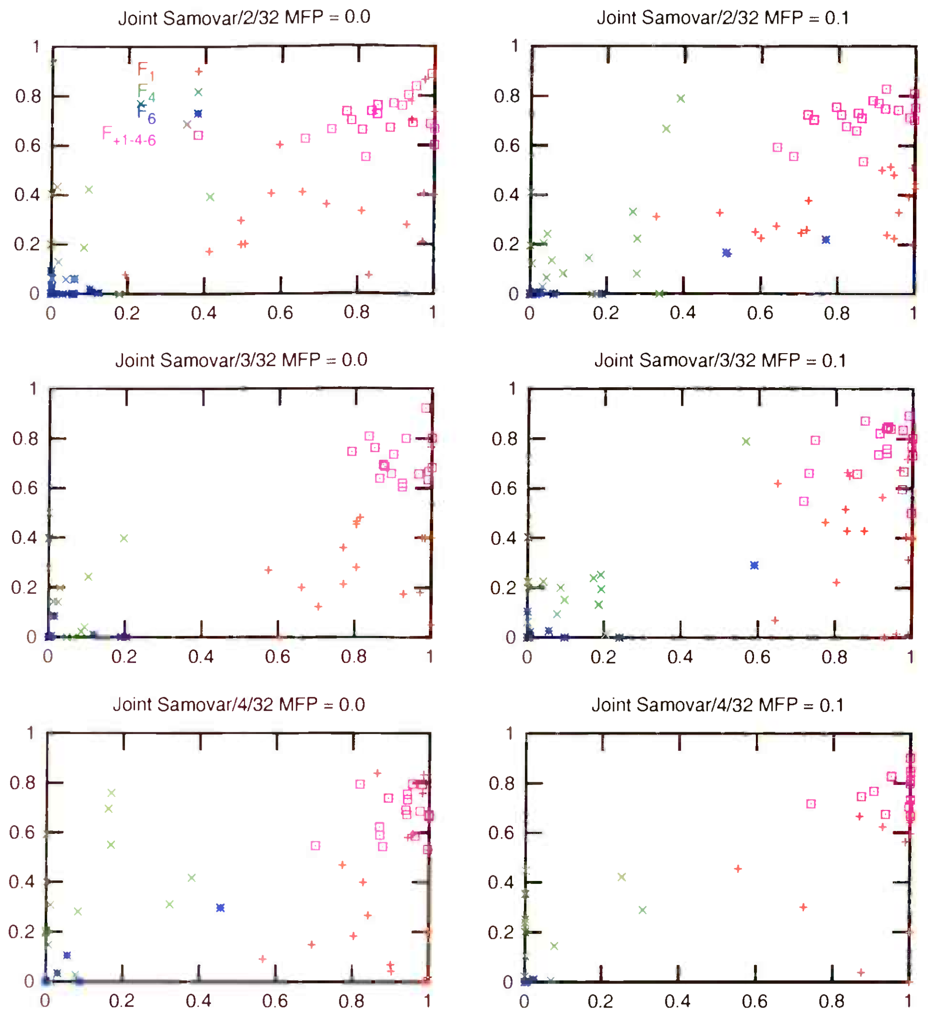
Mean $p(r_{\text{food}} > 0.5)$ and accuracy in environment 2

x-axis: training run; y-axis: test run



Mean $p(r_{\text{food}} > 0.5)$ and accuracy in environment 2

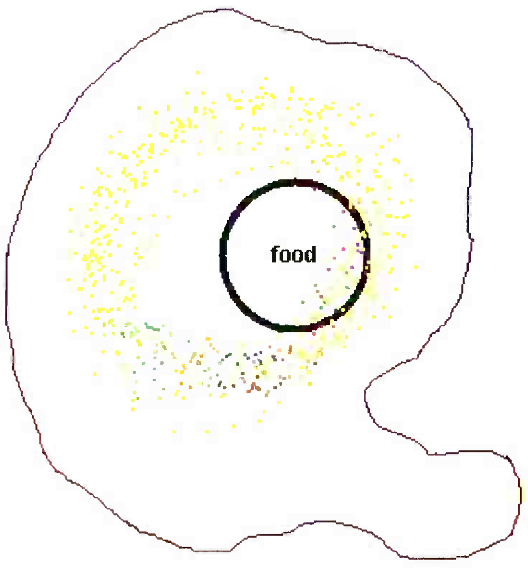
x-axis: training run; y-axis: test run



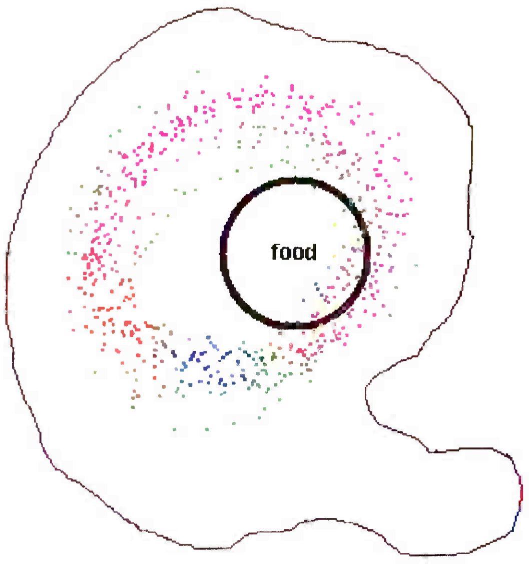
5.4.3.3. *Qualitative*

As one might hope, the Samovar models are able to use a single component to handle the whole of the fairly regular area in the top and left hand parts of environment 2. The

NF-best joint Samovar/8/4, for example, segments its experiences like this:[12]



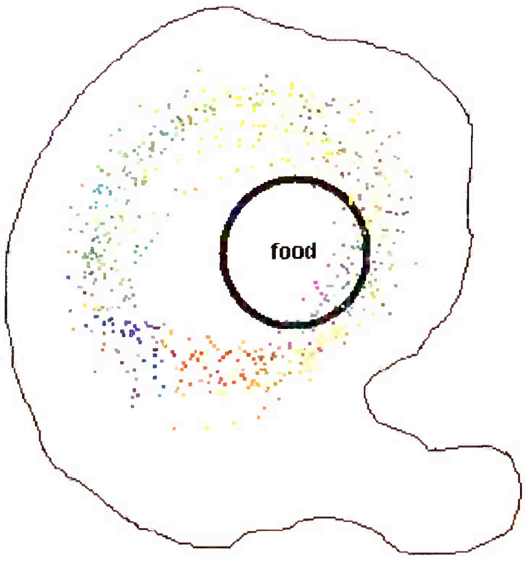
In fact, when the landmark hole in the bottom right-hand corner is open, the component represented by the yellow dots is the only one used. When it is closed, it triggers a separate sequence of mixing states which end, potentially, in the "food" area. The *NF*-best eight-component HMM doesn't really try to predict the range sensor readings, using mostly the same fixed Gaussian magenta, or, when it is either very close to the wall or far away from it, the even vaguer green:



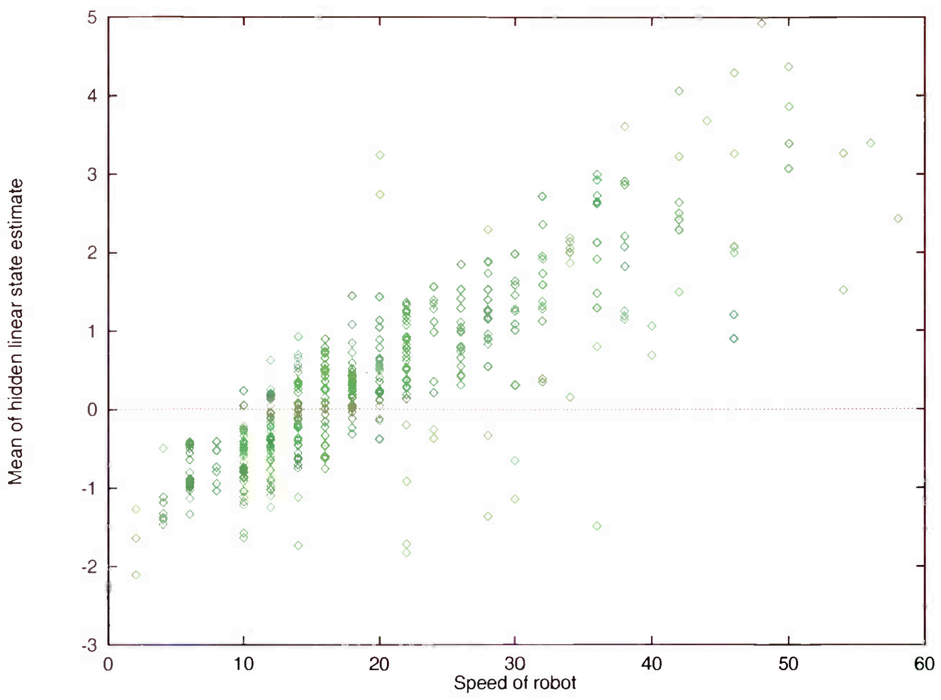
The *NF*-best linear HMM is inclined to use up some of its components on discretising the state information which the Samovar can represent numerically (though this turns out not

[12] see section 5.4.2.3 for a description of what these diagrams mean

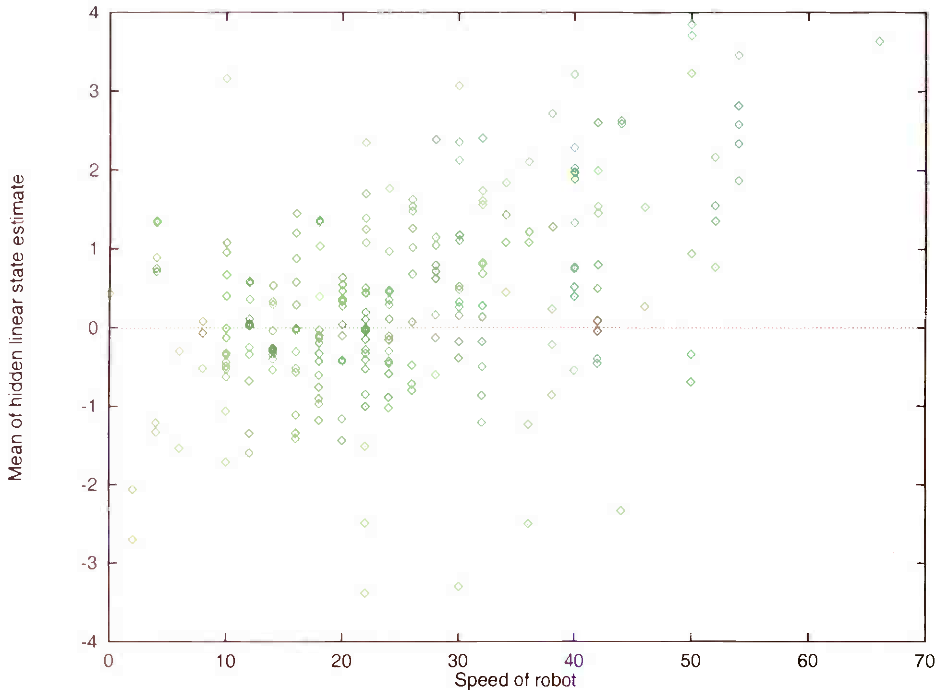
to be a matter simply of associating different states with different speeds), leaving it without enough to throw at the region around the food:



In the case of a Samovar model with one element of linear hidden state, it is possible to plot V_0^t to verify that it really is being used by at least some of the components to represent the robot's speed. Here is a graph showing $\bar{v}_0^t$ (judged *a posteriori*) against the speed at timestep t for those ts at which the most probable component (out of this model's 16, also judged *a posteriori*) is one of a group for which the two quantities were found to be strongly correlated:



The states outside this group do not so obviously use the linear hidden state to represent the robot's speed:



5.4.4. Computational resources required

The time taken to train a Samovar model on a 360MHz UltraSPARC-II varied from around 25 minutes for Samovar/1/8 to around 90 minutes for Samovar/4/32. Little effort was made to optimise the training régime of section 5.3.1.3, the program used to implement it (a native-code executable written in the ocaml language), or indeed the quantity of training data it was asked to process, so there seems to be no reason why larger models could not be learned in an acceptable amount of time. The limiting factor turned out to be the amount of memory required for caching the transition matrix once per timestep—recall that in the conditional models, it varies depending on the sensor readings and actions—but this could easily be circumvented by imposing a time horizon on mixing state inferences as well as for those over the linear hidden state.

For comparison, training times for linear HMMs varied from around five minutes for a linear HMM/8 to around 40 for a linear HMM/32—substantially less, of course, than that of the Samovars. Furthermore, it be seen from the scatterplots in sections 5.4.2.1 and 5.4.3.1 that the performance of the linear HMMs produced by different runs of the training algorithm is much more consistent than that of the Samovar HMMs: to produce a single good Samovar model requires more applications of the training algorithm than are required to produce a single good linear HMM model.

Training a plain HMM/8 took around five minutes and an HMM/32 about 25, and the HMM/8s and HMM/16s were both very consistent from run to run.

5.4.5. Summary

The Samovar models learned to deploy their linear hidden state effectively, both to represent the robot's speed in environment 2 and for various other less obvious purposes. They also generated interesting and natural-seeming classifications of the coarse structure of their environments.

Conditional Samovar/1/16 models produced the best nonfood predictive pdf values in both environments, although their advantage over similar-sized linear HMMs was slight (and their computational demands considerably greater).

In environment 2 especially, the conditional Samovar required only eight components to do a good job both at predicting the food sensor readings and at predicting the range sensor readings, while the other models required sixteen components.

The nonfood predictive pdf values achieved by the joint Samovars were considerably lower than those obtained from the conditional models—because of their longer and more uncertain information path—although the number of basically informative predictions they made was at least as great. They required 32 components to be able to learn to predict the food sensor readings in environment 1, while the conditional models could manage with 8.

Successful generalisation from the training data to the test series was possible, for all but the smallest models of each class, only with the help of the confidence region, which was found to do a fair job of selectively allowing the fallback component to cut in and produce a vague prediction in areas outside the learned components' competence.

Chapter 6

Future Work

6.1. Improving the model

In this section, ways are identified in which the Samovar model and algorithm could be improved.

6.1.1. Better training algorithms

For a start, there are some obvious problems with the training algorithm described in section 4.2.2.

6.1.1.1. Alternatives to Baum-Welch

One of the most significant bottlenecks encountered in the application of the Samovar learning algorithm to the robot environment learning problem (section 5) was the memory cost of the Baum-Welch algorithm (section 3.3.3.3) for inferring the discrete structure of the system being modelled. Baum-Welch is simple and effective on moderate-sized problems, and hence widely used, but it does (in a naive implementation) involve computation time quadratic in the number of the model's discrete states. The situation is made worse in the conditional Samovar model by the fact that the transition matrix is not constant, and must be either stored or repeatedly calculated, at relatively high cost either way, for each timestep. This means that there is a fairly hard limit on the number of mixing states which the Samovar learning algorithm can keep in play at any one time. Since the way it discovers significant chains or sparse networks of states, such as the sequence leading up to the encounters with the "food" region in the simulation, is by experimenting with doubling (or quadrupling) all the states up and then thinning out the resulting transition matrix, a serious constraint is placed on the length of the alternate mixing state trajectories which it can learn. Most of the doubled states will eventually disappear, but in the mean time the algorithm is forced to consume large amounts of time and/or memory. More discriminating ways of encoding the variable transition matrix would help with the latter but compound the former.[1]

Since the same problem has of course been encountered by researchers into speech recognition and indeed robotics, there are a variety of promising techniques available for uncovering the structure of discrete dynamical systems more efficiently. These work by making structural adaptations in a more directed way, for instance greedily pruning down an initially huge model.[2] Heuristics for deciding when to split components of static mixture models and when to merge them may also be of use.[3]

6.1.1.2. Statistical correctness

In the description of the subsequence-joining procedure which comprises the E-step of the Samovar learning algorithm (section 4.2.2.5), it was noted that it is possible for the number of possible-seeming candidate sequences to overwhelm the buffers available for storing them, in which case whole classes of overall sequences—all those including the subsequences discarded on the basis of estimates of their probability made on the basis of inadequate local information—would be pruned out and lost to later consideration. Indeed, even if the probability assessments used in making the early sampling decisions are correct, the fact that large blocks of similar solutions are being rejected means that the eventual sample is bound to be rather lumpy and still won't have the right distribution.

[1] In fact a cheaper representation using eight bits to encode each entry in matrix against a row-specific baseline was found not to degrade the effectiveness of the algorithm significantly, and since the forward-backward equations were in any case programmed in log space, the extra running time required was not great.

[2] Stolcke & Omohundro, *Hidden Markov model induction*

[3] Ueda *et al.*, *SMEM Algorithm for Mixture Models*

One response to this is to argue, with Hinton[4], that the correctness of the E-step is secondary in importance to its adequacy as a platform for raising the complete data log likelihood. In the specific case of the Samovar algorithm, provided that the early M-steps are successful in finding increasingly probable model parameters, the number of plausible short subsequences (which is a reflection of the sparsity of the transition matrix and the tightness of the linear dynamics uncertainty) will fall quite rapidly, so that subsequent E-steps will have fewer blind guesses to make. In the course of testing the algorithm on the fairly sparse FSM of section 5.1.1.1, it was found that the true model parameters were close to being fixed points; although, as is to be expected when learning from finite data, a slight drift was sometimes observed, this was invariably accompanied by an increase in the likelihood.[5]

If, however, it was reckoned to be desirable to obtain a correctly distributed set of state sequences from the algorithm, it could in principle be obtained by importance sampling. The procedure would involve running the algorithm many times—which would progressively eliminate the effect of the blockiness due to early pruning—and correcting the probabilities it assigned to each of the sample sequences it produced by comparing their true likelihoods with their probability under whatever the algorithm's sampling distribution turns out to be. (The tree of decisions made by the algorithm provides a kind of audit trail for assessing how probable each of its eventual outputs was *a priori*.) Ghahramani applies this idea to his variational algorithm for inference over mixtures of factor analysers[6]; of course the feasibility of an importance sampling approach to correcting the Samovar E-step would depend on how close the subsequence-joining algorithm already was to producing the right answers.

6.1.2. The character of the model

Although the mixed-linear/Gaussian form of the Samovar model is quite general and has some desirable properties, it also has a number of limitations.

6.1.2.1. Discrete and non-Gaussian outputs

In section 5.3.1.1, it was noted that the recursive mixed-linear Samovar model was, with its continuous outputs, structurally a poor choice for predicting the plainly discrete "food" sensor readings of the simulated robot tasks; and indeed this mismatch put some obstacles in the way of the model's successful application, although the performance ultimately obtained was not unacceptably compromised. It would, in fact, be very easy to extend the model so

[4] Hinton, *Products of experts*; section 3.3.1.3

[5] unless the priors/regularisers were set in a way inconsistent with the true system

[6] Ghahramani & Beal, *Variational inference . . . factor analysers*

as to cope more correctly with outputs (and inputs) which it was told were discrete in nature and were conditional only on the discrete hidden state Q^t. The Samovar model's likelihood would of course have to be extended with new terms to express the extra dependencies, but they would figure in the learning and prediction algorithms only at the level of the outer, Q expectation (section 4.2.2.4), as an unproblematic extra factor in (99). It would also be straightforward to extend Q with extra values whose role was not to determine the model's current outputs but to influence the future development of Q (a practice widely adopted in the speech recognition community under the name "tying").

Even if the outputs are continuous, it may not be appropriate to model their Q^t-conditional distribution using an unconstrained mean and Gaussian variance. Any of the exponential family distributions commonly used for the outputs of discrete HMMs could be employed. Of course, the reliance placed by the Samovar model on the Kalman filter formalism means that it can only capture continuous, V^t-conditional trends via a linear/Gaussian output distribution.

6.1.2.2. Grouping outputs

The Samovar model is also suboptimal in the way it assumes that when the situation changes (*i.e.* a different Q^t cuts in), the linear rules for *all* the sensor outputs change along with it. For an example of a situation in which some sensor characteristics change but others do not, consider the robot simulation of section 5: around the "food" area, the range sensors behave as they always did, but because the mixing state is different (in order to bring about the correct food sensor reading), the model must learn that pattern all over again. It would not be difficult to specify a model in which the mixing state was grouped so that several linear (or whatever—see section 6.1.2.1) mappings could be asked to predict different sensor readings at each timestep. But it is not obvious how such a model could be learned. The issue is discussed briefly in section 6.1.3; note also that Pierce has addressed a similar issue of grouping uninterpreted sensors in his work on mobile robot localisation and mapbuilding[7].

6.1.2.3. Products of experts

Hinton and colleagues have recently drawn attention[8] to a generic failing of mixture models, namely that they scale poorly to high dimensional spaces. The root of the problem is this: mixture models proceed from the assumption that the features observed in each data point can all be traced back to the parameters of a single generator. Considerable effort is expended in using *EM* or similar techniques to identify the component responsible. But in

[7] Pierce & Kuipers, *Map learning*

[8] Hinton, *Products of experts*

fact, most multidimensional densities are more structured than that, in ways that we tend to express linguistically using words like 'and' and 'but': "The readings are always arranged in this region, and also fit this other pattern, but in that situation this part is impossible." As a result, the distribution's highest density region can be at the same time small in overall volume, but impossible to represent using a tractable number of generically shaped fixed or movable generators (kernels or mixture components); in a high dimensional space, the cardinality of the mixture may have to be multiplied until it approaches that of the data set in order of magnitude.

The suggested solution is to model the distribution not as a sum of local sub-distributions, but as a *product* of expert distributions specialised in detecting features (and not necessarily localised in feature space). Where the values of these densities are simultaneously high ('and'), the combined, product probability will be high as well; where one or more of them is close to zero ('but'), the probability will be low. This gives rise to the generative model

$$p(r^t \mid \theta) = \frac{\prod_i \mathbf{p}(r^t \mid \theta_i)}{\int_{r^t} \prod_i \mathbf{p}(r^t \mid \theta_i)} \tag{118}$$

where the product is scaled to enforce the constraint that it integrate to unity. Actually, it is not clear that the expert "distributions" are really best viewed as probability densities at all; the whole point of the model is that they cannot be used individually to predict anything, but only collectively (which is why they have been written using the bold $\mathbf{p}(\cdots)$ above). Note that there is no requirement that they should themselves be normalised; in fact it seems plausible that they are closely related to the "potentials" of undirected graphical models.[9].

At first sight it appears impossible to learn a product of experts model efficiently, because the normaliser in (118) gives rise to an awkward extra term in the derivative of the log likelihood:

$$\frac{\partial}{\partial \theta_i} \log p(r^t \mid \theta) = \frac{\partial}{\partial \theta_i} \log \mathbf{p}(r^t \mid \theta_i) - \int_{r^t} p(r^t \mid \theta) \frac{\partial}{\partial \theta_i} \log \mathbf{p}(r^t \mid \theta_i) \tag{119}$$

The "purpose" of subtracting the expected derivative of *all* the possible data with respect to each expert's parameter could be seen as encouraging the experts to diversify: it should actively avoid agreeing with other experts where its voice would be redundant in order to concentrate on its own speciality. Computing the expectation exactly is impossible, and indeed it would seem to demand the use of a heavyweight Markov chain Monte Carlo method. However, the authors present good results obtained with derivatives calculated on the basis of a very short chain of Gibbs samples—quick to simulate, but much too short

[9] Jordan, *Learning in Graphical Models*; indeed Hinton, *Products of experts* explicitly compares the product model with a Boltzmann machine (a variety of undirected graphical model)

to yield a genuinely independent sample. The most plausible reason they suggest for this felicitous phenomenon is that the diversification term is simply not critically important; it does not much matter whether the experts specialise as much as they optimally might.

One might speculate that some kind of product of experts would be the ideal resolution of the issues of sensor grouping raised in section 6.1.2.2.

6.1.3. Selecting the right structure

Like the standard Baum-Welch HMM learning procedure (section 3.3.3.3), the Samovar learning algorithm as presented in section 4.2.2.9 focusses entirely on the problem of training a model with a previously fixed number of mixing states. The question of how many such states there really ought to be is left unaddressed. Bayesian theory does, in fact, provide an extremely compelling answer, which is simply that one should construct a posterior distribution which includes mixtures of all (conceivably relevant) sizes in exactly the usual way: specifying a prior density which expresses one's beliefs about how plausible all the different possible models, with various different numbers of states, are, and then applying Bayes' rule in the light of the observed data. What makes this idea so attractive is that it naturally embodies an "Occam's razor" filter which automatically rejects large mixtures, unless the data furnish strong evidence in their favour. The reason is simply that mixtures with more components have a higher-dimensional parameter space; and the prior over that space must integrate to unity; so the prior probability available to be spread over each possible mixture model falls off exponentially with their size. Even if the reasoner thinks *a priori* that a big model is a real possibility, and just isn't quite sure which one, the attenuation of the prior density means that all of them will start off with a substantial penalty—which nicely balances the large likelihoods which some of them may give rise to by just happening to (over)fit the data very closely. This notion of complexity control by prior attenuation is perhaps the deepest contribution Bayesian thinking has made to understanding of ideal rationality; it applies not only to mixture models[10], but also to model structure and "power" generally[11].

The problem is that adding models with different structures makes the task of doing inference with the posterior; already intractable in general when a single structure is considered, even more difficult. One approach is to enumerate all the structures which are considered at all plausible, and attempt to assess their probabilities by marginalising out the

[10] Cheeseman *et al.*, *Autoclass*

[11] MacKay, *Bayesian Interpolation*; Dellaportas *et al.*, *On Bayesian Model and Variable Selection*

remaining parameters. For instance, if one is looking for an estimate of the "right" number σ of states for a mixture:

$$\begin{aligned} p(\sigma \mid r, a, \mathcal{H}) &\propto p(r \mid \sigma, a, \mathcal{H})\, p(\sigma \mid \mathcal{H}) \\ &= \int_{\theta \cdot |\theta| = \sigma} p(r \mid \theta, a)\, p(\theta \mid \mathcal{H})\, p(\sigma \mid \mathcal{H}) \end{aligned}$$

It is sometimes possible to find usable approximations to the inevitable integral over θ[12]. Ghahramani has recently given attention to the problem of performing full Bayesian inference over mixtures of factor analysers;[13] since Kalman filtering can be seen as the recursive analogue of the method of factor analyis, and Samovar is a recursively mixed linear model (with output mappings akin to the "factor loading" matrix), it may be possible to extend his variational algorithm to Samovar-type dynamical models as well. Otherwise, there may be no simple and reliable alternative to the time-consuming process of generating a sample of model parameters drawn from the posterior $p(\theta \mid d)$ using (probably) the hybrid Markov chain Monte Carlo method.[14]

Alternatively, it is possible to follow a Markov Chain through the space of model structures. Because this technique can, with ingenuity[15], be used to sample from the space of model structures even if it is more complicated than just a chain of possible "sizes", it would perhaps help with the problem of learning which output distributions were appropriate for each sensor (section 6.1.2.1). Similarly, Boyen has recently demonstrated[16] a "structural *EM*" algorithm for exploring a space of conditionality graphs by incremental adjustment; it could potentially be applied to the problem of factorising the Samovar model's discrete hidden state and grouping its outputs.

6.2. Action selection

The purpose of constructing a model of the robot's environment is to provide it with a flexible way of deciding which actions it ought to perform in order to further its ends. One

[12] see for instance Cheeseman *et al.*, *Autoclass* (mixture models), MacKay, *A Practical Bayesian Framework* (MLPs)

[13] Ghahramani & Beal, *Variational inference ... factor analysers*

[14] Neal, *Probabilistic inference*

[15] Dellaportas *et al.*, *On Bayesian Model and Variable Selection*

[16] Boyen *et al.*, *Discovering the Hidden Structure*

of the attractive aspects of a recursive mixed-linear model is that, in its joint variant (though not in its conditional one), it is sufficiently invertible to permit something reasonably close to optimal planning to be performed over a continuous space of actions.

6.2.1. The theory

According to Bayesian decision theory (section 3.2.6.2), the aim when acting under uncertainty is to find an action which maximises the expected goodness of the future state of the world, where the expectation is taken conditional on the agent's belief distribution about each action's consequences, and the goodness is some predefined assignment of relative costs and benefits to each state.

6.2.1.1. Planning

It is easy to extend the decision rule 26 to the case where one is choosing an action with a view to what will happen over more than one timestep into the future, as a robot clearly must. By the sum rule (marginalisation), what is needed at time T is

$$a_*^t = \operatorname*{argmax}_{a^T} \int_{h^T} \int_{r^{[T,T+\tau)}} p(r^{[T,T+\tau)} \,|\, a^T, h^T, \theta) \sum_{t \in [T,T+\tau)} g(r^t)$$

where $g(r^t)$ is the gain from obtaining sensor readings r^t. (There is no reason why g could not also depend on the action a^t if that was desired. Note that rewards not easily defined purely in terms of r^t can be supported by adding an extra "reward" pseudo-sensor, which is fed manually determined values during training; the model will hopefully develop any hidden state necessary for predicting it just as it does with the real sensors.[17])

6.2.1.2. Planning as inference

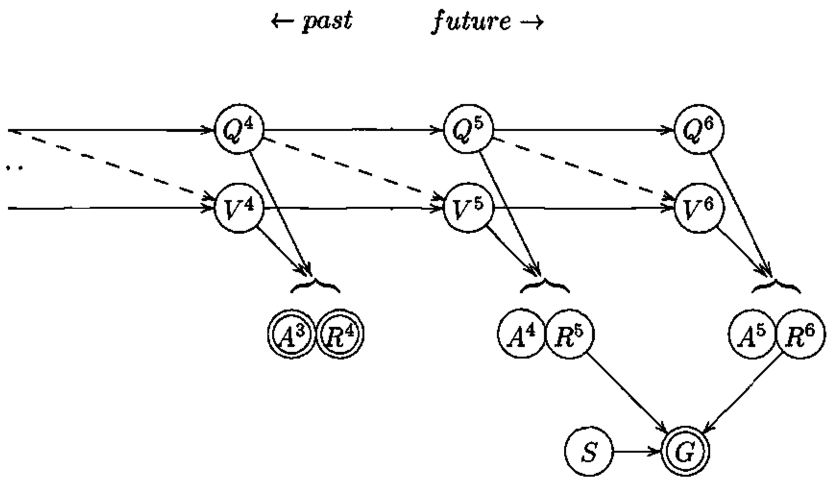
It is possible to consider the search for a good action as a kind of probabilistic inference, by encoding a goodness landscape over sensor readings as the likelihood $p(G = 0 \,|\, r^t, \mathcal{G})$ that a "goal" variable achieves some designated value, *w.l.o.g.* zero, as follows:

$$a_*^t = \operatorname*{argmax}_{a^T} \int_{h^T} \int_{r^{[T,T+\tau)}} p(r^{[T,T+\tau)} \,|\, a^T, h^T, \theta) \sum_{t \in [T,T+\tau)} \frac{1}{\tau} p(G = 0 \,|\, r^t, \mathcal{G}) \qquad (120)$$

Conceptually, there is a distinction between a known, variable goodness for each outcome and a variable probability of a fixed goodness, but once expectations are taken, the numbers will work out the same. Note that weightings $\frac{1}{\tau}$ have been inserted in order to turn the summation over future gains into a legal mixture (it is only the relative gains that matter);

17 *c.f.* Chrisman, *Reinforcement learning*

time-dependent weightings such as a discount rate could also be used. The contribution this makes to the likelihood is precisely that obtained by extending the model to include a noisy measurement taken from *one* of the sensor readings, depending on an unknown selector variable; this means that (120) can be interpreted as maximum likelihood estimation of a^T given this extra "evidence". So a good action can be chosen by projecting a scenario forward into the future which includes that "measurement", and then inferring a distribution for what the actions "must have been", or "must be going to be". The following graph shows the state of play when this approach is applied to the joint Samovar model (section 4.2.3):



(121)

Here, timestep 4 is the point at which the agent is making a decision, and G, whose value is known to be (going to be) zero, is supposed be generated by adding a Gaussian noise sample to either R^5 or R^6, depending on the unknown value of the discrete "selector" S.

6.2.1.3. A possible algorithm

Setting the problem up this way makes it possible to apply inference techniques from the graphical models literature. In fact, conditioned on a known value of S, only one of the links $R^5 \to G$ and $R^6 \to G$ is effectively present, so that the effect of G is merely to induce a Gaussian distribution over one of the readings. Inference of the hidden state Q, V, and hence A^4, can then be performed using the usual sequence-joining E-step algorithm. It is, for instance, not difficult to find an action intended to yield a payoff at a single, specified future timestep. But of course in general S is unknown. The suggested solution is to adopt the variational approximation from Ghahramani and Hinton's learning algorithm for the switching state-space model (section 4.3.2), where the same problem of accounting for the effect of a selector variable is solved by a kind of mean field approximation: estimates ρ^t are kept of the probability that each of the state-space models in the bank was responsible for generating the reading observed at time t, and they are alternately used for inference of the models' states and recalculated based on those state estimates. Adapting this idea, one might introduce variables ρ^t as channels for information about the value of S in (121); their function in terms of the decision rule (120) would be to represent the degree to which the planner is attempting to achieve a nonzero gain at each timestep. The algorithm would look something like this:

- initialise ρ^t to an even distribution
- repeatedly
    - treat $G = 0$ as a noisy observation made from each sensor reading R^t through the Gaussian distribution representing the goodness landscape, with its precision scaled down by ρ^t
    - use the subsequence-joining algorithm to infer the model's hidden state distribution
    - reset the ρ^ts in proportion to the likelihood with which each of the R^ts accounts for $G = 0$, marginalised over the current estimate of the hidden state
- finally, read off a Gaussian mixture distribution over A^4, and suggest the mode as the recommended action

Note that it would not be possible to apply this procedure to the conditional Samovar model, because it needs to be provided with specific actions on which to condition its mixing state inferences; even if one were simply to neglect the effect of the actions on the mixing state trajectory—which would be unsound since the action-independent weightings ω_{ij} of a conditional gating model are not transition probabilities[18]—one would still be left with the problem of reading off the maximum likelihood A^4 at the end, and its distribution under the conditional model would not be a mixture of Gaussians but something much more complicated.

6.2.2. Taking account of the robot's belief state

Although the algorithm sketched above would be reasonably efficient, it has a theoretical weakness in that it does not take account of the way the robot's belief state will develop.

6.2.2.1. Ballistic actions

As noted in section 2.3.2, it is often useful to try and follow a trajectory through the world in which one is likely to receive sensor information which helps pin down what the world state is; but in the graph (121), nothing is presumed known about future sensor

[18] see section 3.3.3.4

readings except their (intended) goodness. The procedure will, therefore, concentrate on finding actions which would work well if one were to carry them out blind—one might say ballistically. It's impossible to fix this problem straightforwardly, because to do so would mean providing the model with information about what the future readings and actions are going to be—and that, of course, depends on the plan the robot ultimately comes up with. It would, however, be possible to add a penalty term to the subsequence-joining algorithm which caused it to prefer sequences with low entropy.

6.2.2.2. Monte Carlo planning

While exact inference of the correct "non-ballistic" action is probably impossible, one might perhaps work up a more principled approach to performing genuine POMDP planning using a Monte Carlo optimisation method. For instance,

- to plan forwards from time t,
    - repeatedly
        - start with an arbitrary action $a^{t,0}$
        - repeatedly
            - sample a candidate $a^{t,n+1}$ from some proposal distribution conditional on $a^{t,n}$
            - assess its expected goodness by recursively planning forwards from time $t+1$ under the supposition that $A^t = a^{t,n+1}$
            - adopt $a^{t,n+1}$ or not according to a Metropolis-Hastings acceptance rule
    - return the best $a^{t,0}$ found

By applying itself recursively to assess the future effect of an action, this procedure would automatically be employing realistic estimates of the succeeding actions and readings on the robot's belief state. For this reason it would, however, take a long time to run (exponential, in fact, in the length of the time window considered). If one wished to have the robot follow a policy of optimising a given sensor goodness measure over a reasonably extended period, it would pay to use this procedure to learn (offline) a policy oracle which would take a belief state as input and map it to a suggested action; however, the belief state is a very high-dimensional space, and it's not clear how this oracle should be parameterised.

Chapter 7

Conclusions

The present thesis has proposed the use of a mixed-linear probabilistic state-space model for learning the dynamics of a robot's interaction with the world. A novel algorithm has been presented for training and interrogating such a model, which compares favourably in computational complexity with existing techniques.[1] Results collected from two simulated mobile robot environments support the claim that mixed-linear models can capture both discontinuous and continuous structure in the world in an intuitively natural manner; while they were not proved to perform significantly better than simpler autoregressive hidden Markov models on these simple tasks, it is possible to claim tentatively that they might scale more effectively to environments in which trends over time played a larger role. Two types of probabilistic confidence region, including a generative one which as a side-effect considerably simplified the learning algorithm, were quite effective at preventing both HMM and mixed-linear models from making over-confident but wrong predictions.[2]

The near-invertibility of the mixed-linear model with generative confidence region made it possible to suggest a reasonably efficient algorithm (not yet tried out) for planning the robot's future actions so as to optimise its expected reward; however, it could allow at best only inexactly for the effect of the robot's future experiences on its belief state.

In placing the work presented here in the context of other approaches to similar problems, parallels have been drawn with probabilistic techniques for robot localisation and

[1] chapter 4

[2] chapter 5

mapping, neural network system identification, and visual tracking.[3] Mixed-linear dynamics models might find applications at the borders of these areas, subsuming several tasks often treated as different in kind into a single framework:

- **Categorising distinguished locations** in the environment, or more generally regions in the abstract state space of the robot-world dynamical system, and discovering how they connect to one another. The mixed-linear formalism provides a natural way of extending appearance-based methods[4] to what could be called the "look and feel" of the environment, based on linear relationships between sensor readings and dynamical quantities such as the robot's speed. Note that it would be easy in the probabilistic framework to incorporate a known model of noisy odometry information, or more complicated sensor models such as those developed for computer vision, which the robot could learn to exploit as appropriate.
- **Learning the response to actuator commands** as a "black box" system, from the insider's point of view. A mixed-linear state-space model is a plausible generalisation of the hidden Markov and Kalman models previously applied to this problem, while being both easier to understand and work with, and no less powerful in principle, than the multi-layer perceptron.
- **Accounting for dynamic external phenomena.** Mixed-linear models are already used successfully for visual tracking, and they ought to be able to capture the effect of moving objects on, for example, a robot's range sensor readings, in a powerful and robust way.

Before this last possibility in particular has a serious chance of coming to fruition, some more work will have to be done on introducing extra structure into the model, breaking the one-to-all link between the discrete state at each timestep and the linear mapping given the task of predicting all of that step's sensor readings.[4] Otherwise, the unsophisticated, but in essence domain-neutral, training régime adopted for the simulation experiments[5] may plausibly provide an adequate basis for the application of mixed-linear models to other robots and worlds. Implementing the learning algorithm in a more memory- and time-efficient way than it was in the experiments reported above should allow it to handle models with at least 64 components and several dimensions of linear hidden state.[6] Some care

[3] chapter 2; section 4.3

[4] section 2.1.2.6

[4] section 6.1.2.2

[5] section 5.3.1.3; section 5.3.2

[6] section 5.4.4

certainly has to be taken in setting up roughly appropriate priors to fend off regularisation-related problems[7], and in exploring the space of model sizes (number of mixing states and dimensionality of linear hidden state); since the methods presented here are based on maximum likelihood model estimation and not full Bayesian inference, the latter can only be accomplished by validating models' promise against their performance on unseen data.[8]

[7] section 5.3.1.1

[8] The recommended procedure for getting something working after as few experiences as possible is to train a fair sample of models on some test data, and then choose between them based on their success at predicting new data as it comes in (section 3.2.4.5).

Appendix A

Notation

All continuous quantities, such as process outputs and model parameters, are assumed to be vector-valued; no distinguishing notation is adopted. Transposition is denoted by $\cdots'$ and matrix inversion by $\cdots^{-1}$. Vector and matrix values are used freely in the standard grid notation to denote subblocks of larger assemblies.

Model parameters are written using Greek letters (θ). Random variables are denoted by capital letters (Θ, R^t), and their values by the corresponding lower case letter (θ, r^t). The latter can also be used on their own to assert a value of the former, so that $p(a^t)$ means $p(A^t = a^t)$.

Temporal indexing is indicated by superscripts (r^t, S^u). "Missing out" an indexing superscript or subscript can be used to mean "the whole sequence/set" (r, μ). Range superscripts select the corresponding subsequence out of the variable they are applied to ($r^{[0,T)}$). Values computed at a particular iteration of an *EM* algorithm are identified by superscripts involving the iteration number n (θ^n).

Indexing into a discrete set of simultaneous possibilities, such as mixture components, is indicated by subscripts ($\mu_\imath$, c_l^u). Subscripts are also used to select rows and/or columns from vectors and matrices ($(\cdots)_A$), and have a special meaning in section 3.3.3.5.

A.1. Greek symbols

Symbol	*Usage*	*See also*	*First use*
α	The precision (inverse variance) of the Gaussian noise term in the inter-state dynamics of a Kalman filter. Symbolically, $V^{t+1} = \lambda V^t + N(0, \alpha)$.		section 3.3.3.5
α_i	The precision (inverse variance) of the Gaussian noise term in component i of the inter-linear-state dynamics of a Samovar model. Symbolically, if $Q^t = i$ then $Y^{t+1} = \lambda_i X^t + N(0, \alpha_i)$. Note that in the Samovar model, the robot sensor readings and actions are subsumed into the world states X and Y.		section 4.2.1.2
β	The precision (inverse variance) of the Gaussian noise which a model—Gaussian, Kalman filter, or whatever—adds to each output R^t. In the Samovar model, β is subsumed into α.	μ, κ	section 3.2.5.2
β_i	The precision (inverse variance) of the Gaussian noise which component i of a mixture model—Gaussian mixture, mixture of experts, or whatever—adds to each process output R^t. In the Samovar model, β_i is subsumed into α_i.	μ_i, κ_i	section 3.3.2.1
γ	The precision (inverse variance) of the Gaussian patch from which a joint-Gaussian model generates each process input A^t. Or, the precisions of the Gaussian patches of all the components of a joint mixture of experts, or of the Gaussian receptive fields of all the components of a conditional mixture of experts or Samovar model—both symbolically $\cup_i\{\gamma_i\}$.	ν	section 3.2.5.3 section 3.3.2.4 section 3.3.2.5

γ_i	The precision (inverse variance) of the Gaussian patch from which the component i of a joint mixture of experts generates each process input A^t, or of the Gaussian receptive field according to which the component i of a conditional mixture of experts claims inputs for itself.	γ, ν_i	section 3.3.2.4 section 3.3.2.5
$\delta_{i,j}$	Not a model parameter, but the standard Kronecker delta: unity if $i = j$, zero otherwise.		
δ	Unsubscripted, the model parameters generically governing the inter-state dynamics of whatever dynamical system model is being treated.	ρ	section 3.3.3.1
ι	The "model parameter" expressing the distribution of the initial hidden state H^0 of a dynamical systems model ($= V^0$ in the Kalman filter, Q^0 in the hidden Markov model).		section 3.3.3.1
ι_i	In the hidden Markov model, $\iota_i = p(Q^0 = i)$ is the "model parameter" specifying the probability that the initial state at timestep t = 0 is i.		section 3.3.3.3
$\bar{\iota}, \tilde{\iota}$	In the Kalman filter, these "model parameters" define the Gaussian distribution of the initial state. Symbolically, $V^0 \sim N(\bar{\iota}, \tilde{\iota})$.		section 3.3.3.5
κ	The output linear mapping (output matrix) of a linear regressive model or Kalman filter. Symbolically, $R^t = \kappa A^t + N(0, \beta)$, or $R^t = \kappa V^t + N(0, \beta)$ in the case of a KF.		section 3.2.5.4 section 3.3.3.5
κ_i	The output linear mapping (output matrix) of the component i of a mixture of experts model. Symbolically, if $Q^t = i$ then $R^t = \kappa_i A^t + N(0, \beta_i)$.		section 3.3.2.4

λ	The inter-state linear mapping (dynamics matrix) of a Kalman filter. Symbolically, $V^{t+1} = \lambda V^t + N(0, \alpha)$.	α	section 3.3.3.5
λ_i	The linear mapping (dynamics matrix) of component i of the inter-linear-state dynamics of a Samovar model. Symbolically, if $Q^t = i$ then $Y^{t+1} = \lambda_i X^t + N(0, \alpha_i)$. Note that X^t includes a bias element which is always set to unity, enabling λ_i to include the intercept as well as the slope of the mapping.		section 4.2.1.2
μ	The mean around which a Gaussian model generates process outputs R^t.	β	section 3.2.5.2
μ_i	The mean around which component i of a Gaussian mixture model generates process outputs R^t.	β_i	section 3.3.2.1
ν	The mean of the Gaussian patch from which a joint-Gaussian model generates each process input A^t. Or, the means of the Gaussian patches of all the components of a joint mixture of experts, or the centres of the Gaussian receptive fields of all the components of a conditional mixture of experts or Samovar model—both symbolically $\cup_i\{\nu_i\}$.	γ	section 3.2.5.3 section 3.3.2.4 section 3.3.2.5
ν_i	The mean of the Gaussian patch from which the component i of a joint mixture of experts generates each process input A^t, or the centre of the Gaussian receptive field according to which the component i of a conditional mixture of experts claims inputs for itself.	ν, γ_i	section 3.3.2.4 section 3.3.2.5

ω_i	The probability with which an unconditional mixture model chooses a particular component i to generate each output R^t, or with which a joint mixture model chooses i to generate each input-output pair A^t, R^t. Or the "background" weighting which a conditional mixture model gives to i in choosing a component to map A^t to R^t.		section 3.2.5.1 section 3.3.2.5
ω_{ij}	The probability with which an HMM transitions to state i from state j—symbolically, $p(Q^{t+1} = i \mid Q^t = j, \theta)$. Or, the "background" weighting which the Samovar model gives to component i when choosing a component to map X^t to Y^{t+1} when the previously chosen component was j.		section 3.3.3.3 section 4.2.1.2
$\tilde{\pi}$	In the variational free energy/ensemble learning view of the *EM* algorithm, a parameter defining a probability density function which is optimised so as to approximate the distribution of the quantities being estimated.		section 3.3.1.3
ρ	The model parameters generically governing the output function of whatever dynamical system model is being treated.	δ	section 3.3.3.1
Θ	The reasoner's beliefs about the process under consideration (generally a random variable whose values are model parameters θ). Strictly speaking, relative to the prior $\mathcal{H}$; but the latter is mostly neglected.		section 3.2.2
θ	Generically, all the parameters of whatever model is currently being treated.		section 3.2.3.1
τ	Not a model parameter, but an alternative to t where two timestep variables are needed.		

ξ^t_{ij}	Not a model parameter, but the probability that a hidden Markov model generated the training outputs and was in states j and i at timesteps t and $t+1$ respectively.		(58)
ζ^u_{lm}	Not a model parameter, but, in the subsequence-joining algorithm for the Samovar model's E-step, an estimate of the probability that the model generated the training outputs and made the component choices described by the subsequences m and l during the time ranges u and $u+1$ respectively.		(103)

A.2. Roman symbols

Symbol	*Usage*	*See also*	*First use*
A^t	Random variable: the input to the process at time t, or the robot's actions at time t.		section 3.2.2
a^t	A value of A^t, *i.e.* the observed process input at time t; or (as a proposition in a probability) an abbreviation for $A^t = a^t$.		section 3.2.2
A	Random variable: all the process inputs, or robot actions, in the training set, which are taken to be all those occurring before timestep T. Symbolically, $\{ A^t : t \in [0,T) \}$.	D	section 3.2.2
$(\cdots)_A$	A is sometimes used as a subscript to select rows and/or columns corresponding to the position of A^t in X^t.	$(\cdots)_R, (\cdots)_V$	
a	A value of A. Symbolically, $\{ a^t : t \in [0,T) \}$; or (as a proposition in a probability) an abbreviation for $A = a$.	d	section 3.2.2

$\mathcal{A}$	The learner's prior information or assumptions about the process input.		section 3.2.2.1
B	A statistic used in the (re)estimation rule for the Kalman filter and Samovar models: the expected sum of the outer products of the bases for the inter-state linear mapping.	C, P	(76) section 4.2.2.3
b	$b(h^t)$ is the probability that a dynamical process is in state h^t and then produces the outputs $r^{[t,T)}$. If the process is an HMM or a KF, it can be computed efficiently using the backward half of the forward-backward equations or the Rauch recursions, respectively.	f	(51) 57 section 3.3.3.5
C	A statistic used in the (re)estimation rule for the Kalman filter and Samovar models: the expected sum of the outer products of the targets for the inter-state linear mapping.	B, P	(76) section 4.2.2.3
c_l^u	In the subsequence-joining algorithm for the Samovar model's E-step, the likelihood that the components of the mixed-linear dynamics chosen during the time range $[Lu, Lu+L)$ are those proposed in the subsequence l, and that the outputs $r^{[Lu,Lu+L)}$ are generated.	S^u, z_{lm}^u	(101)
D^t	Random variable: the observable values at timestep t, comprising the process input and output (or robot action and sensor reading). Symbolically, $\begin{pmatrix} R^t \\ A^t \end{pmatrix}$		section 4.2.1.1
d^t	A value of D^t, *i.e.* the observed process output and input (or robot action and sensor reading) at time t—symbolically, $\begin{pmatrix} r^t \\ a^t \end{pmatrix}$—or (as a proposition in a probability) an abbreviation for $D^t = d^t$.		section 4.2.1.2

D	Random variable: the process inputs and outputs (or robot actions and sensor readings) on which the model is trained, which are taken to be all those occurring before timestep T. Symbolically, $D = A \cup R = \{A^t, R^t : t \in [0, T)\}$.		section 3.2.3.1
d	A value of D: the observations of process inputs and outputs (or robot actions and sensor readings) used to train the model, which are taken to be all those occurring before timestep T. Symbolically, $d = a \cup r = \{a^t, r^t : t \in [0, T)\}$.		section 3.2.3.1
$E[\cdots]$	$E_x[f(x) \mid y]$ is the expectation of $f(x)$ given y: $\int_x p(x \mid y)\, f(x)$. "*E*-step" is the name given to the first half of each iteration of the *EM* algorithm.		section 3.3.1.1
F	"Variational free energy".		section 3.3.1.3
F_i	Average probability at which the robot's "food" sensor is predicted to exceed 0.5 in situations of category i		section 5.4.1.1
f	$f(h^t)$ is the probability that a dynamical process produces the outputs $r^{[0,t]}$, finishing in state h^t. If the process is an HMM or a KF, it can be computed efficiently using the forward half of the forward-backward equations or the Kalman recursions, respectively.	b	(51) 56 section 3.3.3.5
$g(\cdots \mid \mu, \beta)$	The Gaussian pdf with mean μ and precision (inverse variance) β.		section 3.3.2.5
$g(a^t, y^t)$	Gain function defining goodness of performing action a^t in world state y^t.		section 3.2.6.2

H^t	Random variable: the process's hidden state at timestep t. This can comprise a continuous quantity V^t on which other quantities depend linearly, or a discrete quantity Q^t which controls a choice between mixture components, or a combination of both.		section 3.3.3
h^t	A value of h^t *i.e.* the process's hidden state at time t; or (as a proposition in a probability) an abbreviation for $H^t = h^t$.	v^t, q^t	section 3.3.3
H	Random variable: the process's hidden state at every timestep over the training period. Symbolically, $\{ H^t : t \in [0, T) \}$.	V, Q	section 3.3.3
h	A value of H. Symbolically, $\{ h^t : t \in [0, T) \}$; or (as a proposition in a probability) an abbreviation for $H = h$.	v, q	section 3.3.3
$\mathcal{H}$	The learner's prior information or assumptions about the process.	Θ	section 3.2.3.1
I	The identity matrix.		section 3.3.3.5
i	A particular component of a mixture model—generally a value of Q^t for some t. $Q^t = i$ or j or k is used instead of q^t where the superscript would get in the way.		section 3.2.5.1
J^t	A working value used in the Kalman-Rauch recursions.		section 3.3.3.5
j	(See i.)		
K^t	A working value used in the Kalman-Rauch recursions.		section 3.3.3.5
$\mathcal{K}$	Proposition: all the agent's knowledge relevant to her decision as to which action to take.		section 3.2.6.2
k	(See i.)		

L	In the subsequence-joining algorithm for the Samovar model's E-step, the length of the subsequences currently being evaluated. Samovar models: the expected sum of the outer product of each base and target for the inter-state linear mapping.	u, S^u	section 4.2.2.5
$L(\cdots, \lambda)$	A Lagrangian function used in a constrained optimisation.	$L(a^t, y^t)$	section 3.2.5.1
l	In the subsequence-joining algorithm for the Samovar model's E-step, a particular subsequence of mixture component choices, *i.e.* a value of S^u for some u. l^0 is the first and l^{L-1} the last. $S^u = l$ or m is used instead of s^u where the superscript would get in the way.		section 4.2.2.5
m	(See l.)		
$N(\mu, \beta)$	Denotes the Gaussian distribution with mean μ and precision (inverse variance) β. Also used as an anonymous Gaussian random variable.		
n	Index of *EM* algorithm iterations. θ^n is the estimate of the overall model parameters at iteration n; the same notation is used on all the sub-parameters λ^n, ι^n *etc.*		section 3.3.1.1
P	A statistic used in the (re)estimation rule for the Kalman filter and Samovar models: the expected sum of the outer product of each base and target for the inter-state linear mapping.	B, C	(76) section 4.2.2.3

Q^t	Random variable: the process's mixing hidden state at timestep t, *i.e.* a discrete-valued, unobservable quantity inside the process (or in the robot's environment), which can control the choice of mixture component in generating, for instance, R^t, and/or influence its successor Q^{t+1} through a transition matrix.	ω, μ, β	section 3.3.2.1 section 3.3.3.3
q^t	A value of Q^t *i.e.* the process's mixing hidden state at time t; or (as a proposition in a probability) an abbreviation for $Q^t = q^t$.		section 3.3.2.1 section 3.3.3.3
Q	Random variable: the process's mixing hidden state at every timestep over the training period. Symbolically, $\{ Q^t : t \in [0,T) \}$.		section 3.3.2.1 section 3.3.3.3
q	A value of Q. Symbolically, $\{ q^t : t \in [0,T) \}$; or (as a proposition in a probability) an abbreviation for $Q = q$.		section 3.3.2.1 section 3.3.3.3
R^t	Random variable: the output of the process at time t, or the robot's sensor readings at time t.		section 3.2.2
r^t	A value of R^t, *i.e.* the observed process output at time t; or (as a proposition in a probability) an abbreviation for $R^t = r^t$.		section 3.2.2
R	Random variable: all the process outputs, or robot sensor readings, in the training set, which are taken to be all those occurring before timestep T. Symbolically, $\{ R^t : t \in [0,T) \}$.	D	section 3.2.2
$(\cdots)_R$	R is sometimes used as a subscript to select rows and/or columns corresponding to the position of R^t in X^t.	$(\cdots)_A, (\cdots)_V$	
r	A value of R. Symbolically, $\{ r^t : t \in [0,T) \}$; or (as a proposition in a probability) an abbreviation for $R = r$.	d	section 3.2.2

S^u	Random variable: the choices of mixture component made by the process during the time range u. Symbolically, $\{Q^t : t \in [Lu, Lu+L)\}$. Values of S^u are written either s^u or l.		100
s^u	A value of S^u, *i.e.* a sequence of choices of mixing hidden state made by the process during the time range u.	l	100
T	The size of the training set D, or equivalently the timestep at which the learner is asked to make a prediction using her model.		section 3.2.2
t	A timestep. The training set D for the model is taken to comprise observations of the process made over the first T timesteps, in the range $[0, T)$. Timestep T is taken to be the one at which the learner is asked to make a prediction using her model.		section 3.2.2
$U(\theta)$	The expected log likelihood maximised in the M-step of an EM algorithm.		section 3.3.1.2
u	In the subsequence-joining algorithm for the Samovar model's E-step, a range of timesteps (period) over which various possible mixing state subsequences s^u are being locally evaluated. The current length of the subsequences is called L, so range u covers the steps $[Lu, Lu+L)$.		section 4.2.2.5
V^t	Random variable: the process's linear hidden state at timestep t, *i.e.* a continuous-valued, unobservable quantity inside the process (or in the robot's environment), on which other quantities, such as the output (or sensor readings) R^t, or V^t's successor V^{t+1}, are generally supposed to depend through a linear/Gaussian mapping, so that its distribution stays Gaussian.	$\lambda, \alpha, \kappa, \beta$	section 3.3.3.5

v^t	A value of V^t *i.e.* the process's linear hidden state at time t; or (as a proposition in a probability) an abbreviation for $V^t = v^t$.		section 3.3.3.5
V	Random variable: the process's linear hidden state at every timestep over the training period. Symbolically, $\{ V^t : t \in [0,T) \}$.		section 3.3.3.5
$(\cdots)_V$	V is sometimes used as a subscript to select rows and/or columns corresponding to the position of V^t in X^t.	$(\cdots)_R, (\cdots)_A$	
v	A value of V. Symbolically, $\{ v^t : t \in [0,T) \}$; or (as a proposition in a probability) an abbreviation for $V = v$.		section 3.3.3.5
$W(\theta)$	A quantity used in the proof sketch of the correctness of the *EM* algorithm.		section 3.3.1.2
X^t	Random variable: in the Samovar model, the continuous state of the world at timestep t, comprising the robot's sensor readings, the action it takes, and the unknown linear hidden state, along with an element fixed at unity which serves as a bias term. Symbolically, $(H^{t'}, R^{t'}, A^{t'}, 1)$. The world also has some discrete state Q^t.	λ, α	section 4.2.1.1
x^t	A value of X^t *i.e.* a possible continuous world state at time t—symbolically, $(h^{t'}, r^{t'}, a^{t'}, 1)$—or (as a proposition in a probability) an abbreviation for $X^t = x^t$.		section 4.2.1.2
Y^t	Random variable: in the Samovar model, the non-robot-dependent continuous state of the world at timestep t, comprising the robot's sensor readings and the unknown linear hidden state. Symbolically, $(H^{t'}, R^{t'})$.	λ, α	section 4.2.1.1

y^t	A value of Y^t—symbolically, $(h^{t'}, r^{t'})$—or (as a proposition in a probability) an abbreviation for $Y^t = y^t$.	section 4.2.1.2	z^u_{lm}

References

Andrieu, Christophe. Robust Full Bayesian Methods for Neural Networks. In *Advances in Neural Information Processing Systems 12*, 1999.

Basye, Kenneth, Kirman, Thomas Dean Jak and Lejter, Moises. A Decision-Theoretic Approach to Planning, Perception, and Control. *IEEE Expert*, 7(4):58–65, 1992.

Basye, Kenneth, Dean, Thomas and Kaelbling, Leslie. Learning Dynamics: System Identification for Perceptually Challenged Agents. *Artificial Intelligence*, 72:139–171, 1995.

Bengio, Y. and Frasconi, P. An Input Output HMM Architecture. In *Advances in Neural Information Processing Systems 7*. MIT Press, 1995.

Berger, J. O. *Statistical Decision Theory and Bayesian Analysis*. Springer-Verlag, 1985.

Bishop, Christopher M. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

Box, George E. P. and Tiao, George C. *Bayesian Inference in Statistical Analysis*. Wiley Classics Library. Wiley, 1992.

Boyen, Xavier and Koller, Daphne. Tractable Inference for Complex Stochastic Processes. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in AI (UAI-98)*, pages 33–42, 1998.

Boyen, Xavier, Friedman, Nir and Koller, Daphne. Discovering the Hidden Structure of Complex Dynamic Systems. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI-99)*, pages 91–100, 1999.

Brooks, Rodney A. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

Burgard, W., Fox, Dieter, Jans, H., Matenar, C. and Thrun, Sebastian. Sonar-based mapping of large-scale mobile robot environments using EM. In *Proc. of the 16th International Conference on Machine Learning (ICML'99)*, 1999.

Cassandra, Anthony R., Kaelbling, Leslie Pack and Kurien, James A. Acting under Uncertainty: Discrete Bayesian Models for Mobile-Robot Navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1996.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W. and Freedman, D. Autoclass: A Bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning*. Ann Arbor, MI, 1988.

Chrisman, Lonnie. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 183–188. AAAI Press, 1992.

Cox, I. J. and Leonard, J. J. Modeling a dynamic environment using a Bayesian multiple hypothesis approach. *Artificial Intelligence*, 66:311–344, 1994.

Cox, D. R. Probability, Frequency and Reasonable Expectation. *Am. Jour. Phys.*, 14:1–13, 1946.

de Freitas, J.F.G., Niranjan, M. and Gee, A.H. The EM Algorithm and Neural Networks for Nonlinear State Space Estimation. Technical Report CUED/F-INFENG/TR 313, Cambridge University Department of Engineering, 1998a.

de Freitas, J.F.G., Niranjan, M., Gee, A.H. and Doucet, A. Sequential Monte Carlo methods for optimisation of neural network models. Technical Report CUED/F-INFENG/TR 328, Cambridge University Department of Engineering, 1998b.

Dellaert, Frank, Burgard, Wolfram, Fox, Dieter and Thrun, Sebastian. Using the CONDENSATION Algorithm for Robust, Vision-based Mobile Robot Localization. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. IEEE, 1999.

Dellaportas, P., Forster, J. J. and Ntzoufras, I. On Bayesian Model and Variable Selection Using MCMC. Working paper, available from `http://stat-athens.aueb.gr/~ptd`, 1998.

Dempster, A. P., Laird, N. M. and Rubin, D. B. Maximum likelihood from incomplete data via the *EM* algorithm. *J. Roy. Statist. Ser.*, B, 39:1–38, 1977.

de Verdière, V. C. and Crowley, J. L. Local Appearance Space for Recognition of Navigation Landmarks. In *Proceedings of the 6th International Symposium on Intelligent Robotic Systems (SIRS'98)*, pages 261–269, 1998.

Doucet, A. On sequential simulation-based methods for Bayesian filtering. Technical Report CUED/F-INFENG/TR_310, Department of Engineering, Cambridge University, 1998.

Duckett, Tom and Nehmzow, Ulrich. Performance Comparison of Landmark Recognition Systems for Navigating Mobile Robots. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, 2000.

Durrant-Whyte, H. F. Consistent integration and propagation of disparate sensor observations. *International Journal of Robotics Research*, 6(3):3–24, 1987.

Elfes, A. Sonar Based Real World Mapping and Navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, 1987.

Fox, Dieter and Burgard, Wolfram. Active Markov Localization for Mobile Robots. *Robotics and Autonomous Systems*, 25:195–207, 1998.

Fox, Dieter, Burgard, Wolfram and Thrun, Sebastian. Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

Fusiello, A. and Caprile, B. Synthesis of indoor maps in presence of uncertainty. *Robotics and Autonomous Systems*, 22(2):103–114, 1997.

Ghahramani, Z. and Beal, M. J. Variational inference for Bayesian mixture of factor analysers. In *Advances in Neural Information Processing Systems 12*, 1999. To appear; www.gatsby.ucl.ac.uk/~zoubin.

Ghahramani, Zoubin and Hinton, Geoffrey E. Parameter Estimation for Linear Dynamical Systems. Technical Report CRG-TR-96-2, Department of Computer Science, University of Toronto, 1996.

Ghahramani, Zoubin and Hinton, Geoffrey E. Variational Learning for Switching State-Space Models. *Neural Computation*, (in press), ress. www.gatsby.ucl.ac.uk/~zoubin.

Ghahramani, Z. Learning Dynamic Bayesian Networks. In Giles, C. L. and Gori, M. (eds), *Adaptive Processing of Sequences and Data Structures*, Lecture Notes in Artificial Intelligence, pages 168–197. Springer-Verlag, 1998.

Gutmann, J.-S. and Konolige, K. Incremental Mapping of Large Cyclic Environments. In *International Symposium on Computational Intelligence in Robotics and Automation (CIRA'99)*, 1999.

Gutmann, J.-S., Burgard, W., Fox, D. and Konolige, K. An Experimental Comparison of Localization Methods. In *International Conference on Intelligent Robots and Systems (IROS'98)*, 1998.

Hermann, Michael, Pawelzik, Klaus and Geisel, Theo. Self-Localization by Hidden Representations. In *Proceedings of the Eighth International Conference on Artificial Neural Networks*, pages 1103–1108. Springer Verlag, 1998.

Hertz, John, Krogh, Anders and Palmer, Richard G. *Introduction to the theory of neural computation.* Addison Wesley, 1991.

Hinton, G. E. and van Camp, D. Keeping neural networks simple by minimizing the description length of the weights. In *Proceedings of COLT*, 1993.

Hinton, G. E. Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, volume 1, pages 1–6. Springer Verlag, 1999.

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

Isard, Michael and Blake, Andrew. A smoothing filter for CONDENSATION. In *Proceedings of the 5th European Conference on Computer Vision*, pages 767–781. Springer Verlag, 1998b.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J. and Hinton, G. E. Adaptive mixtures of local experts. In *Advances in Neural Information Processing Systems 3*, pages 79–87. Morgan Kaufman, 1991.

Jaynes, E. T. Probability Theory: The Logic of Science. Available from `ftp://bayes.wustl.edu`, 1995.

Jordan, M. I. and Jacobs, R. A. Hierarchical mixtures of experts and the EM algorithm. In *Advances in Neural Information Processing Systems 6*, pages 181–214. Morgan Kaufman, 1994.

Jordan, Michael I. (ed). *Learning in Graphical Models.* MIT Press, 1999.

Julier, S. J. and Uhlmann, J. K. A General Method for Approximating Nonlinear Transformations of Probability Distributions. "Internet Publication"; see `http://www.robots.ox.ac.uk/~siju/work/work.html#UnscentedFiltering`, 1996.

Kaelbling, Leslie Pack, Littman, Michael L. and Cassandra, Anthony R. Planning and Acting in Partially Observable Stochastic Domains. *Artificial Intelligence*, 101, 1998.

Koenig, S. and Simmons, R. G. Unsupervised learning of probabilistic models for robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2555–2560, 1996a.

Koenig, S. and Simmons, Reid. Passive Distance Learning for Robot Navigation. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, pages 266–274, 1996b.

Kristensen, S. Sensor planning with Bayesian decision theory. *Robotics and Autonomous Systems*, 19(3–4):273–286, 1997.

Kröse, B. J. A. and Bunschoten, R. Probabilistic localization by appearance models and active vision. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2255–2260, 1999.

Kröse, Ben, Bunschoten, Roland, Vlassis, Nikos and Motomura, Yoichi. Appearance based robot localization. In *IJCAI-99 Workshop Adaptive Spatial Representations of Dynamic Environments*, pages 53–58. Morgan Kaufman, 1999.

Kuipers, B. and Levitt, T. Navigation and mapping in large-scale spaces. *AI Magazine*, 9:25–43, 1988.

Kurz, A. Constructing maps for mobile robot navigation based on ultrasonic range data. *IEEE Trans. Systems, Man and Cybernetics B*, 26(2), 1996.

Lauritzen, S. L. and Spiegelhalter, D. J. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statististical Society B.*, pages 157–224, 1988.

Lee, Peter M. *Bayesian Statistics: An Introduction.* Arnold, 1997. Second edition.

Leonard, J. J., Moran, B. A. and Cox, I. J. Underwater Sonar Data Fusion Using an Efficient Multiple Hypothesis Algorithm. In *IEEE Int. Conf. on Robotics and Automation.* IEEE, 1995.

Lu, F. and Milios, E. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

MacKay, David J. C. Bayesian Interpolation. *Neural Computation*, 4(3):415–447, 1992a.

MacKay, David J. C. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3), 1992b.

Meilă, Marina and Jordan, Michael I. Learning Fine Motion by Markov Mixtures of Experts. In *Advances in Neural Information Processing Systems 8*, pages 1003–1009. Morgan Kaufman, 1996.

Moller, M. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):535–533, 1993.

Moravec, H. P. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9:61–74, 1988.

Narendra, K. Neural networks for control: Theory and practice. *Proceedings of the IEEE*, 84(10):1385–1406, 1996.

Neal, Radford M. and Hinton, Geoffrey E. A New View of the EM Algorithm that Justifies Incremental and Other Variants. In Jordan Jordan, *Learning in Graphical Models.*

Neal, R. M. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

Neal, R. M. *Bayesian Learning for Neural Networks.* Springer-Verlag, 1996.

Nelson, Alex T. and Wan, Eric A. Neural Speech Enhancement Using Dual Extended Kalman Filtering. In *Proceedings of the Seventh International Conference on Artificial Neural Networks*, 1997.

North, B., Blake, A., Isard, M. and Rittscher, J. Learning and classification of complex dynamics. Technical Report draft report, Department of Engineering Science, University of Oxford, 1999.

Oore, S., Hinton, G. and Dudek, G. A mobile robot that learns its place. *Neural Computation*, 9:683–699, 1997.

Pierce, David and Kuipers, Benjamin. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92:169–229, 1997.

Pourraz, F. and Crowley, J. L. Continuity Properties of the Appearance Manifold for Mobile Robot Position Estimation. In *Proceedings of the 6th International Symposium on Intelligent Robotic Systems (SIRS'98)*, pages 251–260, 1998.

Puskorius, G. and Feldkamp, L. Decoupled extended Kalman filter training of feedforward layered networks. In *Proceedings of the Fourth International Joint Conference on Neural Networks*, pages I–771–777, 1991.

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

Racz, J. and Dubrawski, A. Artificial neural network for mobile robot topological localization. *Robotics and Autonomous Systems*, 16, 1995.

Rauch, H. E. Solutions to the linear smoothing problem. *IEEE Transactions on Automatic Control*, 8:371–372, 1963.

Sarle, Warren S. Why Statisticians Should Not FART. ftp://ftp.sas.com/pub/neural/fart.txt, 1995.

Shatkay, Hagit and Kaelbling, Leslie Pack. Learning Topological Maps with Weak Local Odometric Information. In *Fifteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufman, 1997.

Shatkay, Hagit and Kaelbling, Leslie Pack. Heading in the Right Direction. In *Proceedings of the Fifth International Conference on Machine Learning (ICML)*, 1998.

Shumway, R. H. and Stoffer, D. S. An approach to time series smoothing and forecasting using the EM algorithm. *J. Time Series Analysis*, 3(4):253–264, 1982.

Stolcke, A. and Omohundro, S. Hidden Markov model induction by Bayesian model merging. Technical Report TR-94-003, International Computer Science Institute, 1994.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Tani, Jun and Fukumura, Naohiro. A Dynamical Systems Approach for a Learnable Autonomous Robot. In *Advances in Neural Information Processing Systems 8*, pages 989–995. Morgan Kaufman, 1996.

Thrun, Sebastian and Langford, John. Monte Carlo Hidden Markov Models. Technical Report CMU-CS-98-179, School of Computer Science, Carnegie Mellon University, 1998.

Thrun, Sebastian. Bayesian Landmark Learning for Mobile Robot Localization. *Machine Learning*, 33(1), 1998a.

Thrun, Sebastian. Learning Metric-Topological Maps for Indoor Mobile Robot Navigation. *Artificial Intelligence*, 99(1):21–71, 1998b.

Thrun, S., Gutmann, S., D.Fox, Burgard, W. and Kuipers, B. Integrating Topological and Metric Maps for Mobile Robot Navigation: A Statistical Approach. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press, 1998a.

Thrun, Sebastian, Burgard, Wolfram and Fox, Dieter. A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots. *Machine Learning*, 31(5):1–25, 1998b.

Thrun, S., Burgard, W., and Fox, D. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. "Submitted for publication"; see `http://www.cs.cmu.edu/~thrun`, 1999.

Tutschku, K. Recurrent Multilayer Perceptrons for Identification and Control: The Road to Applications. Technical Report 118, University of Würzburg Institute of Computer Science, 1995.

Ueda, N., Nakano, R., Ghahramani, Z. and Hinton, G. E. SMEM Algorithm for Mixture Models. *Neural Computation*, in press.

Vlassis, Nikos and Kröse, Ben. Robot Environment Modeling via Principal Component Regression. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1999.

Vlassis, Nikos, Motomura, Yoichi and Kröse, Ben. An information-theoretic localization criterion for robot map building. In *Proc. ACAI'99, Int. Conf. on Machine Learning and Applications*, 1999.

Waterhouse, Steve, MacKay, David and Robinson, Tony. Bayesian Methods for Mixtures of Experts. In *Advances in Neural Information Processing Systems 8*. Morgan Kaufman, 1996.

Whitehead, Steven D. and Ballard, Dana H. Active Perception and Reinforcement Learning. In *Proc. of the 7th International Conference on Machine Learning (ICML'90)*, 1990a.

Whitehead, Steven D. and Ballard, Dana H. Learning to Perceive and Act. Technical Report Technical Report 331, Dept. of Computer Science, University of Rochester, 1990b.

Xu, Lei, Jordan, Michael I. and Hinton, Geoffrey E. An Alternative Model for Mixtures of Experts. In *Advances in Neural Information Processing Systems 7*, pages 633–640. Morgan Kaufman, 1995.