2

Online simulation inference

Concept. To extend the formalism that we developed in the previous chapter to describe the time evolution of state probabilities. Having introduced the basic concepts, we then use this formalism to motivate some important methods for probabilistic learning, which we then use in developing a framework for online simulation inference. For the mathematically-inclined, this chapter will take a detailed look at how our formalism can be extended to focus on the time evolution of probabilities, with a view to online simulation inference later on. For the programmers, all of the relevant software lives in this public Git repository: https://github.com/umbralcalc/learnadex.

2.1 Probabilistic formalism

This book is about building more realistic training environments for machine learning systems in the real world. In the last chapter we formalised, designed and built a generalised simulation engine which could serve as the essential scaffolding for these environments. So why then, in this chapter, do we want to extend our formalism to talk about probabilistic learning methods?

For many of these realistic environments where only partial state observations are possible, probabilistic learning tools can immediately become an essential tool to robustly infer the state of a system and predict its future states from any given point in time. Given that these tools are often so important for extending the capability of learning algorithms to deal with partially-observed domains; we propose to extend the idea of what is more conventionally considered a 'machine learning environment' to include tools for system state inference and future state prediction. In this sense, this chapter, and the remaining chapters of this book, will consider probabilistic learning as an essential part of the environment for a machine learning system.

In this chapter, we're going to begin by discussing the formal connections between some probabilistic learning methods and the simulation formalism we introduced in the last chapter. Let's start by returning to the formalism that we introduced in that chapter. As we discussed at that point; this formalism is appropriate for sampling from nearly every stochastic phenomenon that one can think of. We are going to extend this description to consider what happens to the probability that the state history matrix takes a particular set of values over time.

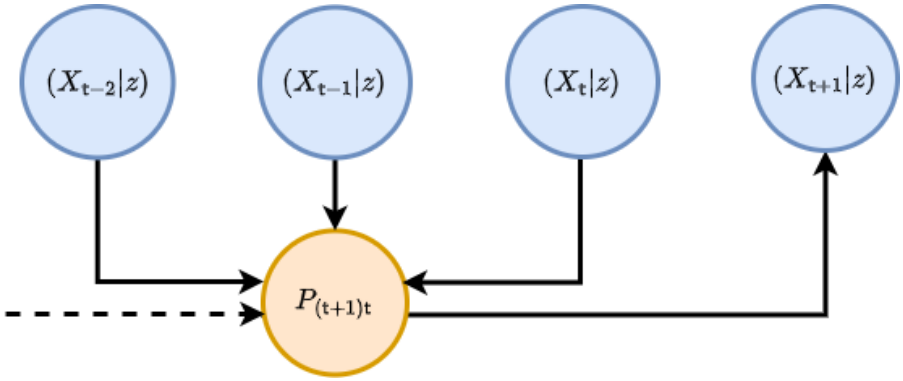

Figure 2.1: Graph representation of Eq. (2.1).

So, how do we begin? Previously, we defined the general stochastic process with the formula $X^i_{t+1} = F^i_{t+1}(X_{0:t}, z, t)$. This equation also has an implicit *master equation* associated to it that fully describes the time evolution of the *probability density function* $P_{t+1}(X|z)$ of $X_{0:t+1} = X$ given that the parameters of the process are z. This can be written as

$$P_{t+1}(X|z) = P_t(X'|z)P_{(t+1)t}(x|X', z)\,, \tag{2.1}$$

where for the time being we are assuming the state space is continuous in each of the matrix elements and $P_{(t+1)t}(x|X', z)$ is the conditional probability that $X_{t+1} = x$ given that $X_{0:t} = X'$ at time t and the parameters of the process are z. To try and understand what Eq. (2.1) is saying, we find it's helpful to think of an iterative relationship between probabilities; each of which is connected by their relative conditional probabilities. We've also illustrated this kind of thinking in Fig. 2.1.

Consider what happens when we extend the chain of conditional probabilities in Eq. (2.1) back in time by one step. In doing so, we retrieve a joint probability of rows $X_{t+1} = x$ and $X_t = x'$ on the right hand side of the expression

$$P_{t+1}(X|z) = P_{t-1}(X''|z)P_{(t+1)t(t-1)}(x, x'|X'', z)\,. \tag{2.2}$$

Since Eqs. (2.1) and (2.2) are both valid ways to obtain $P_{t+1}(X|z)$ we can average between them without loss of generality in the original expression, like this

$$P_{t+1}(X|z) = \frac{1}{2}\left[P_t(X'|z)P_{(t+1)t}(x|X', z) + P_{t-1}(X''|z)P_{(t+1)t(t-1)}(x, x'|X'', z)\right]. \tag{2.3}$$

Following this line of reasoning to its natural conclusion, Eq. (2.1) can hence be generalised to consider all possible joint distributions of rows at different timesteps like this

$$P_{t+1}(X|z) = \frac{1}{t}\sum_{t''=0}^{t} P_{t''}(X''|z)P_{(t+1)t\ldots t''}(x, x', \ldots|X'', z)\,. \tag{2.4}$$

If we wanted to just look at the distribution over the latest row $X_{t+1} = x$, we could achieve this through marginalisation over all of the previous matrix rows in Eq. (2.1) like this

$$P_{t+1}(x|z) = \int_{\Omega_t} \mathrm{d}X' P_{t+1}(X|z) = \int_{\Omega_t} \mathrm{d}X' P_t(X'|z)P_{(t+1)t}(x|X', z)\,. \tag{2.5}$$

But what is Ω_{t}? You can think of this as just the domain of possible matrix X' inputs into the integral which will depend on the specific stochastic process we are looking at.

The symbol $\mathrm{d}X'$ in Eq. (2.5) is our shorthand notation throughout the book for computing the sum of integrals over previous state history matrices which can further be reduced via Eq. (2.4) into a product of sub-domain integrals over each matrix row

$$P_{\mathsf{t}+1}(x|z) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}''=0}^{\mathsf{t}}\left\{\int_{\omega_{\mathsf{t}'}}\mathrm{d}^n x' ... \int_{\Omega_{\mathsf{t}''}}\mathrm{d}X''\right\} P_{\mathsf{t}''}(X''|z)P_{(\mathsf{t}+1)\mathsf{t}...\mathsf{t}''}(x,x',...|X'',z) \tag{2.6}$$

$$= \frac{1}{\mathsf{t}}\sum_{\mathsf{t}''=0}^{\mathsf{t}}\int_{\Omega_{\mathsf{t}''}}\mathrm{d}X'' P_{\mathsf{t}''}(X''|z)P_{(\mathsf{t}+1)\mathsf{t}''}(x|X'',z)\,, \tag{2.7}$$

where each row measure is a Cartesian product of n elements (a Lebesgue measure), i.e.,

$$\mathrm{d}^n x = \prod_{i=0}^{n}\mathrm{d}x^i\,, \tag{2.8}$$

and lowercase $x, x', \ldots$ values will always refer to individual rows within the state matrices. Note that $1/\mathsf{t}$ here is a normalisation factor — this just normalises the sum of all probabilities to 1 given that there is a sum over t'. Note also that, if the process is defined over continuous time, we would need to replace

$$\frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}} \to \frac{1}{t(\mathsf{t})}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\delta t(\mathsf{t}')\,. \tag{2.9}$$

Let's go through some examples. Non-Markovian phenomena with continuous state spaces can have quite complex master equations. A relatively simple example is that of pure diffusion processes which exhibit stochastic resetting at a rate r to a remembered location from the trajectory history [1]

$$P_{\mathsf{t}+1}(x|z) = (1-r)P_{\mathsf{t}}(x|z) + \sum_{i=0}^{n}\sum_{j=0}^{n}\frac{\partial}{\partial x^i}\frac{\partial}{\partial x^j}\Big[D_{\mathsf{t}}(x,z)P_{\mathsf{t}}(x|z)\Big] + r\sum_{\mathsf{t}'=0}^{\mathsf{t}}\delta t(\mathsf{t}')K[t(\mathsf{t})-t(\mathsf{t}')]P_{\mathsf{t}'}(x|z)\,, \tag{2.10}$$

where here K is some memory kernel. For Markovian phenomena which have a continuous state space, Eqs. (2.1) and (2.5) no longer depend on timesteps older than the immediately previous one, hence, e.g., Eq. (2.5) reduces to just

$$P_{\mathsf{t}+1}(x|z) = \int_{\omega_{\mathsf{t}}}\mathrm{d}^n x'\, P_{\mathsf{t}}(x'|z)P_{(\mathsf{t}+1)\mathsf{t}}(x|x',z)\,. \tag{2.11}$$

A famous example of this kind of phenomenon arises from approximating Eq. (2.11) with an expansion (Kramers-Moyal [2, 3]) up to second-order, yielding the Fokker-Planck equation

$$P_{\mathsf{t}+1}(x|z) = P_{\mathsf{t}}(x|z) - \sum_{i=0}^{n}\frac{\partial}{\partial x^i}\Big[\mu_{\mathsf{t}}(x,z)P_{\mathsf{t}}(x|z)\Big] + \sum_{i=0}^{n}\sum_{j=0}^{n}\frac{\partial}{\partial x^i}\frac{\partial}{\partial x^j}\Big[D_{\mathsf{t}}(x,z)P_{\mathsf{t}}(x|z)\Big]\,, \tag{2.12}$$

which describes a process undergoing drift-diffusion.

An analog of Eq. (2.5) exists for discrete state spaces as well. We just need to replace the integral with a sum and the schematic would look something like this

$$P_{\mathsf{t}+1}(x|z) = \sum_{\Omega_{\mathsf{t}}} P_{\mathsf{t}}(X'|z) P_{(\mathsf{t}+1)\mathsf{t}}(x|X', z)\,, \tag{2.13}$$

where we note that the P's in the expression above all now refer to *probability mass functions*. In what follows, discrete state space can always be considered by replacing the integrals with summations over probability masses in this manner; we only use the continuous state space formulation for our notation because one could argue it's a little more general.

Analogously to continuous state spaces, we can give some examples of master equations for phenomena with a discrete state space as well. In the Markovian case, we need look no further than a simple time-dependent Poisson process

$$P_{\mathsf{t}+1}(x|z) = \lambda(\mathsf{t})\delta t(\mathsf{t}{+}1) P_{\mathsf{t}}(x{-}1|z) + \big[1 - \lambda(\mathsf{t})\delta t(\mathsf{t}{+}1)\big] P_{\mathsf{t}}(x|z)\,. \tag{2.14}$$

For such an example of a non-Markovian system, a Hawkes process [4] master equation would look something like this

$$\begin{aligned} P_{\mathsf{t}+1}(x|z) = {} & \mu\delta t(\mathsf{t}{+}1) P_{\mathsf{t}}(x{-}1|z) + \big[1 - \mu\delta t(\mathsf{t}{+}1)\big] P_{\mathsf{t}}(x|z) \\ & + \sum_{x'=0}^{\infty} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \phi[t(\mathsf{t}) - t(\mathsf{t}')]\delta t(\mathsf{t}{+}1) P_{\mathsf{t}\mathsf{t}'(\mathsf{t}'-1)}(x{-}1, x', x'{-}1|z) \\ & + \sum_{x'=0}^{\infty} \left\{1 - \sum_{\mathsf{t}'=0}^{\mathsf{t}} \phi[t(\mathsf{t}) - t(\mathsf{t}')]\delta t(\mathsf{t}{+}1)\right\} P_{\mathsf{t}\mathsf{t}'(\mathsf{t}'-1)}(x, x', x'{-}1|z)\,, \end{aligned} \tag{2.15}$$

where we note the complexity in this expression arises because it has to include a coupling between the rate at which events occur and an explicit memory of when the previous ones did occur (recorded by differencing the count between adjacent timesteps by 1).

2.2 Probabilistic learning algorithms

So now that we are more familiar with the notation used by Eq. (2.5), we can use it to motivate some useful probabilistic learning methods. While it's worth going into some mathematical detail to give a better sense of where each technique comes from, we should emphasise that the methodologies we discuss here are not new to the technical literature at all. We draw on influences from Empirical Dynamical Modeling (EDM) [5], some classic nonparametric local regression techniques — such as LOWESS/Savitzky-Golay filtering [6] — and also Gaussian processes [7].

Let's begin our discussion of algorithms by integrating Eq. (2.5) over x to obtain a relation for the mean of the distribution

$$M_{\mathsf{t}+1}(z) = \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, x\, P_{\mathsf{t}+1}(x|z) = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}''=0}^{\mathsf{t}} \int_{\Omega_{\mathsf{t}''}} \mathrm{d}X'' P_{\mathsf{t}''}(X''|z) M_{(\mathsf{t}+1)\mathsf{t}''}(X'', z)\,, \tag{2.16}$$

where you can view the $M_{(\mathsf{t}+1)\mathsf{t}''}(X'', z)$ values as either terms in some regression model, or derivable explicitly from a known master equation. The latter of these provides one approach to statistically

infer the states and parameters of stochastic simulations from data: one begins by knowing what the master equation is, uses this to compute the time evolution of the mean (and potentially higher-order statistics) and then connects these $\mathtt{t}$ and z-dependent statistics back to the likelihood of observing the data. This is what is commonly known as the 'mean-field' inference approach; averaging over the available degrees of freedom in the statistical moments of distributions. Though, knowing what the master equation is for an arbitrarily-defined stochastic phenomenon can be very difficult indeed, and the resulting equations typically require some form of approximation.

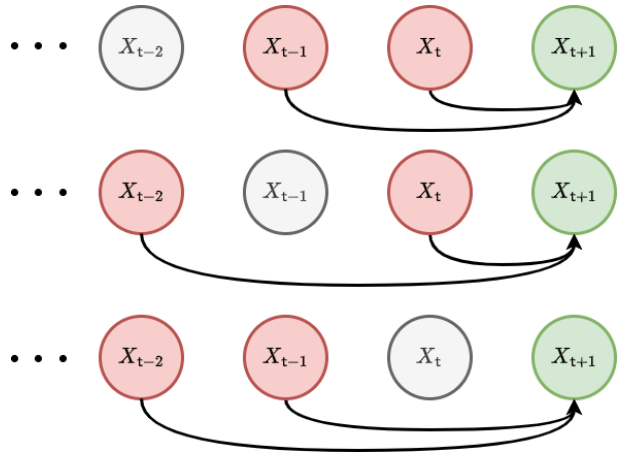

Figure 2.2: A graph representation of the correlations in Eq. (2.21).

Given that the mean-field approach isn't always going to be viable as an inference method, we should also consider other ways to describe the shape and time evolution characteristics of $P_{\mathtt{t}+1}(X|z)$. For continuous state spaces, it's possible to approximate this whole distribution with a logarithmic expansion like so

$$\ln P_{\mathtt{t}+1}(X|z) \simeq \ln P_{\mathtt{t}+1}(X_*|z) + \frac{1}{2}\sum_{\mathtt{t}'=0}^{\mathtt{t}+1}\sum_{i=0}^{n}\sum_{j=0}^{n}(x-x_*)^i \mathcal{H}^{ij}_{(\mathtt{t}+1)\mathtt{t}'}(z)(x'-x'_*)^j \tag{2.17}$$

$$\mathcal{H}^{ij}_{(\mathtt{t}+1)\mathtt{t}'}(z) = \frac{\partial}{\partial x^i}\frac{\partial}{\partial (x')^j}\ln P_{\mathtt{t}+1}(X|z)\bigg|_{X=X_*}, \tag{2.18}$$

where the values for X_* (and its rows $x_*, x'_*, \dots$) are defined by the vanishing of the first derivative, i.e., these are chosen such that

$$\frac{\partial}{\partial x^i}\ln P_{\mathtt{t}+1}(X|z)\bigg|_{X=X_*} = 0\,. \tag{2.19}$$

This logarithmic expansion is one way to see how a Gaussian process regression is able to approximate $X_{\mathtt{t}}$ evolving in time for many different processes. By selecting the appropriate function for the kernel given by Eq. (2.18), a Gaussian process regression can be fully specified.

If we keep the truncation up to second order in Eq. (2.17), note that this expression implies a

pairwise correlation structure of the form

$$P_{t+1}(X|z) \to \prod_{t'=0}^{t} P_{(t+1)t'}(x, x'|z) = \prod_{t'=0}^{t} P_{t'}(x'|z) P_{(t+1)t'}(x|x', z)\,. \tag{2.20}$$

Given this pairwise temporal correlation structure, Eq. (2.7) reduces to this simpler sum of integrals

$$P_{t+1}(x|z) = \frac{1}{t} \sum_{t'=0}^{t} \int_{\omega_{t'}} \mathrm{d}^n x' P_{t'}(x'|z) P_{(t+1)t'}(x|x', z)\,. \tag{2.21}$$

We have illustrated these second-order correlations with a graph visualisation in Fig. (2.2).

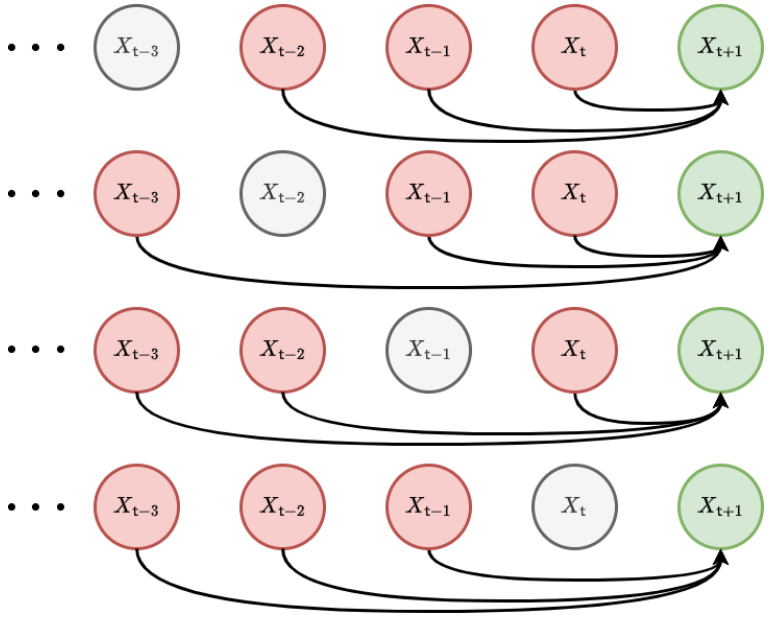

Figure 2.3: A graph representation of the correlations in Eq. (2.23).

In a similar fashion, we can increase the expansion order of Eq. (2.17) to include third-order correlations such that

$$P_{t+1}(X|z) \to \prod_{t'=0}^{t} \prod_{t''=0}^{t'-1} P_{t't''}(x', x''|z) P_{(t+1)t't''}(x|x', x'', z)\,, \tag{2.22}$$

and, in this instance, one can show that Eq. (2.7) reduces to

$$P_{t+1}(x|z) = \frac{1}{t} \sum_{t'=0}^{t} \frac{1}{t'-1} \sum_{t''=0}^{t'-1} \int_{\omega_{t'}} \mathrm{d}^n x' \int_{\omega_{t''}} \mathrm{d}^n x'' P_{t't''}(x', x''|z) P_{(t+1)t't''}(x|x', x'', z)\,. \tag{2.23}$$

We have also illustrated these third-order correlations with another graph visualisation in Fig. (2.3). Using $P_{\mathsf{t}'\mathsf{t}''}(x', x''|z) = P_{\mathsf{t}''}(x''|z)P_{\mathsf{t}'\mathsf{t}''}(x'|x'', z)$ one can also show that this integral is a marginalisation of this expression

$$P_{(\mathsf{t}+1)\mathsf{t}''}(x|x'', z) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x' P_{\mathsf{t}'\mathsf{t}''}(x'|x'', z)P_{(\mathsf{t}+1)\mathsf{t}'\mathsf{t}''}(x|x', x'', z)\,, \tag{2.24}$$

which describes the time evolution of the conditional probabilities. Note how this implies that the Gaussian process kernel itself can be evolved through time to replicate these higher-order temporal correlations for a regression problem, if desired.

Another probabilistic learning algorithm that we can consider is what we shall call 'empirical probabilistic reweighting'. There is another expression for the mean of the distribution, that we can derive under certain conditions, which will be valuable to motivating this algorithm. If the probability distribution over each row of the state history matrix is *stationary* — meaning that $P_{\mathsf{t}+1}(x|z) = P_{\mathsf{t}'}(x|z)$ — it's possible to go one step further than Eq. (2.16) and assert that

$$M_{\mathsf{t}+1}(z) = \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, x\, P_{\mathsf{t}+1}(x|z) = \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x'\, x'\, P_{\mathsf{t}'}(x'|z)\int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)\,. \tag{2.25}$$

To see that Eq. (2.25) is true, first note that a joint distribution over both x and x' can be derived like this $P_{(\mathsf{t}+1)\mathsf{t}'}(x, x'|z) = P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)P_{\mathsf{t}'}(x'|z)$. Secondly, note that this joint distribution will always allow variable swaps trivially like this $P_{(\mathsf{t}+1)\mathsf{t}'}(x, x'|z) = P_{\mathsf{t}'(\mathsf{t}+1)}(x', x|z)$. Then, lastly, note that stationarity of $P_{\mathsf{t}+1}(x|z) = P_{\mathsf{t}'}(x|z)$ means

$$\begin{aligned}
\frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x \int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x'\, x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x, x'|z) &= \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x'\, x\, P_{\mathsf{t}'(\mathsf{t}+1)}(x, x'|z) \\
&= \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x' \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, x'\, P_{(\mathsf{t}+1)\mathsf{t}'}(x, x'|z) \\
&= \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x'\, x'\, P_{\mathsf{t}'}(x'|z)\int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)\,,
\end{aligned}$$

where we've used the trivial variable swap and integration variable relabelling to arrive at the second equality in the expressions above.

The standard covariance matrix elements can also be computed in a similar fashion

$$\begin{aligned}
C^{ij}_{\mathsf{t}+1}(z) &= \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, [x - M_{\mathsf{t}+1}(z)]^i [x - M_{\mathsf{t}+1}(z)]^j P_{\mathsf{t}+1}(x|z) \\
&= \frac{1}{\mathsf{t}}\sum_{\mathsf{t}'=0}^{\mathsf{t}}\int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x'\, [x' - M_{\mathsf{t}+1}(z)]^i [x' - M_{\mathsf{t}+1}(z)]^j\, P_{\mathsf{t}'}(x'|z)\int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)\,.
\end{aligned} \tag{2.26}$$

While they look quite abstract, Eqs. (2.25) and (2.26) express the core idea behind how the probabilistic reweighting will function. By assuming a stationary distribution, we gain the ability to directly estimate the statistics of the probability distribution of the next sample from the stochastic

process $P_{\mathsf{t}+1}(x|z)$ from past samples it may have in empirical data; which are represented here by $P_{\mathsf{t}'}(x'|z)$.

Probabilistic reweighting depends on the stationarity of $P_{\mathsf{t}+1}(x|z) = P_{\mathsf{t}'}(x|z)$ such that, e.g., Eq. (2.25) is applicable. The core idea behind it is to represent the past distribution of state values $P_{\mathsf{t}'}(x'|z)$ with the samples from a real time series dataset. If the user then specifies a good model for the relationships in this data by providing a weighting function which returns the conditional probability mass

$$\mathsf{w}_{\mathsf{t}'}(y,z) = \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x \, P_{(\mathsf{t}+1)\mathsf{t}'}(x|x'{=}y,z)\,, \tag{2.27}$$

we can apply this as a *reweighting* of the historical time series samples to estimate any statistics of interest. Taking Eqs. (2.25) and (2.26) as the examples; we are essentially approximating these integrals through weighted sample estimations like this

$$M_{\mathsf{t}+1}(z) \simeq \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} Y_{\mathsf{t}'} \mathsf{w}_{\mathsf{t}'}(Y_{\mathsf{t}'}, z) \tag{2.28}$$

$$C^{ij}_{\mathsf{t}+1}(z) \simeq \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} [Y_{\mathsf{t}'} - M_{\mathsf{t}+1}(z)]^i [Y_{\mathsf{t}'} - M_{\mathsf{t}+1}(z)]^j \, \mathsf{w}_{\mathsf{t}'}(Y_{\mathsf{t}'}, z)\,, \tag{2.29}$$

where we have defined the data matrix Y with rows $Y_{\mathsf{t}+1}, Y_{\mathsf{t}}, \ldots$, each of which representing specific observations of the rows in X at each point in time from a real dataset.

The goal of a learning algorithm for probabilistic reweighting would be to learn the optimal reweighting function $\mathsf{w}_{\mathsf{t}'}(Y_{\mathsf{t}'}, z)$ with respect to z, i.e., the ones which most accurately represent a provided dataset. But before we think about the various kinds of conditional probability we could use, we need to think about how to connect the post-reweighting statistics to the data by defining an objective function.

If the mean is a sufficient statistic for the distribution which describes the data, a choice of, e.g., Exponential, Poisson or Binomial distribution could be used where the mean is estimated directly from the time series using Eq. (2.25), given a conditional probability $P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)$. Extending this idea further to include distributions which also require a variance to be known, e.g., the Normal, Gamma or Negative Binomial distributions could be used where the variance (and/or covariance) could be estimated using Eq. (2.26). These are just a few simple examples of distributions that can link the estimated statistics from Eqs. (2.25) and (2.26) to a time series dataset. However, the algorithmic framework is very general to whatever choice of 'data linking' distribution that a researcher might need.

We should probably make what we've just said a little more mathematically concrete. We can define $P_{\mathsf{t}+1}[y; M_{\mathsf{t}+1}(z), C_{\mathsf{t}+1}(z), \ldots]$ as representing the likelihood of $y = Y_{\mathsf{t}+1}$ given the estimated statistics from Eqs. (2.25) and (2.26) (and maybe higher-orders). Note that in order to do this, we need to identify the x' and t' values that are used to estimate, e.g., $M_{\mathsf{t}+1}(z)$ with the past data values which are observed in the dataset time series itself. Now that we have this likelihood, we can immediately evaluate an objective function (a cumulative log-likelihood) that we might seek to optimise over for a given dataset

$$\ln \mathcal{L}_{\mathsf{t}+1}(Y|z) = \sum_{\mathsf{t}'=0}^{\mathsf{t}+1} \ln P_{\mathsf{t}'}[y; M_{\mathsf{t}'}(z), C_{\mathsf{t}'}(z), \ldots]\,, \tag{2.30}$$

where the summation continues until all of the past measurements $Y_{t+1}, Y_t, \ldots$ which exist as rows in the data matrix Y have been taken into account. The code to compute this objective function follows the schematic we have provided in Fig. 2.4.

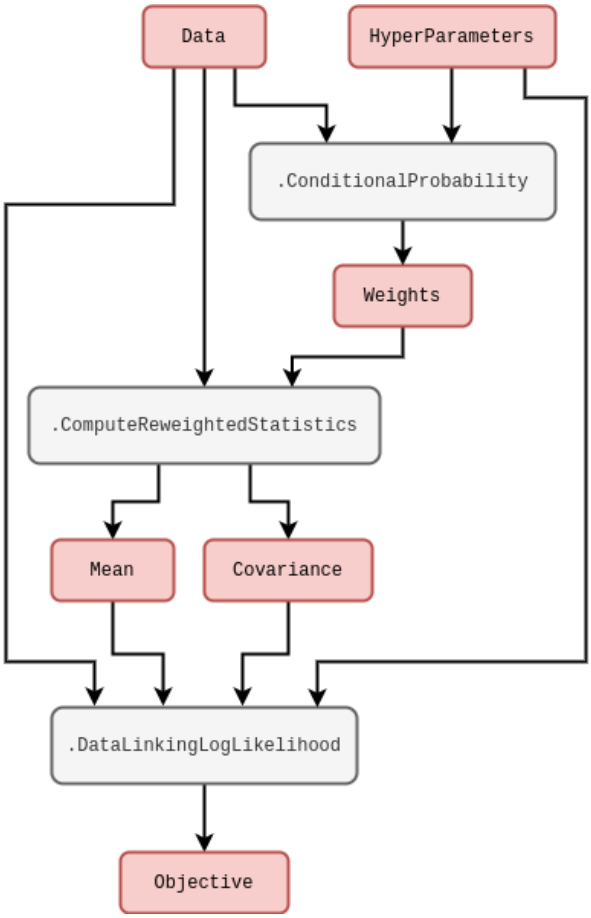

Figure 2.4: Code schematic of the probability reweighting objective computation.

In order to specify what $P_{(t+1)t'}(x|x', z)$ is, it's quite natural to define a set of hyperparameters for the elements of z. To get a sense of how the data-linking function relates to these hyperparameters, it's instructive to consider an example. One generally-applicable option for the conditional probability could be a purely time-dependent kernel

$$P_{(t+1)t'}(x|x', z) \propto \mathcal{K}(z, t+1, t'), \tag{2.31}$$

and the data-linking distribution, e.g., could be a Gaussian

$$P_{t+1}[y; M_{t+1}(z), C_{t+1}(z), \ldots] = \mathsf{MultivariateNormalPDF}[y; M_{t+1}(z), C_{t+1}(z)]. \tag{2.32}$$

It's worth pointing out that other machine learning frameworks could easily be used to model these conditional probabilities. For example, neural networks could be used to infer the optimal

reweighting scheme and this would still allow us to use the data-linking distribution.[1] It would still be desirable to keep the data-linking distribution as it can usually be sampled from very easily — something that can be quite difficult to achieve with a purely machine learning-based representation of the distribution. Sampling itself could even be made more flexible by leveraging a Variational Autoencoder (VAE) [9]; these use neural networks not just on the compression (or 'encode') step to estimate the statistics but also use them as a layer between the sample from the data distribution model and the output (the 'decode' step).

In the case of Eqs. (2.31) and (2.32) above, the hyperparameters that would be optimised could relate to the kernel in a wide variety of ways. Optimising them would make our optimised reweighting similar to (but very much *not* the same as) evaluating maximum a posteriori (MAP) of a Gaussian process regression. In a Gaussian process regression, one is concerned with inferring the the whole of X_t as a function of time using the pairwise correlations implied by Eq. (2.17). Based on this expression, the cumulative log-likelihood for a Gaussian process can be calculated as follows

$$\ln \mathcal{L}_{\mathsf{t}+1}(Y|z) = -\frac{1}{2}\sum_{\mathsf{t}'=0}^{\mathsf{t}+1}\sum_{\mathsf{t}''=0}^{\mathsf{t}'}\left[n\ln(2\pi) + \ln\left|\mathcal{H}_{\mathsf{t}'\mathsf{t}''}(z)\right| + \sum_{i=0}^{n}\sum_{j=0}^{n} Y^i_{\mathsf{t}'}\mathcal{H}^{ij}_{\mathsf{t}'\mathsf{t}''}(z)Y^j_{\mathsf{t}''}\right]. \tag{2.33}$$

As we did for the reweighting algorithm, in Fig. 2.5 we have illustrated a schematic of the basic code needed to compute the objective function of a learning algorithm based on Eq. (2.33). Note that, in the expression above, we have assumed that the data has already been shifted such that its values are positioned around the distribution peak. Knowing where this peak will be a priori is not possible. However, for Gaussian data, an unbiased estimator for this peak will be the sample mean and so we have included an initial data standardisation in the steps outlined by Fig. 2.5.

The optimisation approach that we choose to use for obtaining the best hyperparameters in the conditional probability of Eq. (2.30) will depend on a few factors. For example, if the number of hyperparameters is relatively low, but their gradients are difficult to calculate exactly; then a gradient-free optimiser (such as the Nelder-Mead [10] method or something like a particle swarm [11, 12]) would likely be the most effective choice. On the other hand, when the number of hyperparameters ends up being relatively large, it's usually quite desriable to utilise the gradients in algorithms like vanilla Stochastic Gradient Descent [13] (SGD) or Adam [14].

If the gradients of Eq. (2.30) are needed, we can always factorise each derivative with respect to hyperparameter z^i in the following way through the chain rule

$$\begin{aligned}\frac{\partial}{\partial z^i}\ln\mathcal{L}_{\mathsf{t}+1}(Y|z) = &\sum_{\mathsf{t}'=0}^{\mathsf{t}+1}\frac{\partial M_{\mathsf{t}'}}{\partial z^i}\frac{\partial}{\partial M_{\mathsf{t}'}}\ln P_{\mathsf{t}'}[y; M_{\mathsf{t}'}(z), C_{\mathsf{t}'}(z), \dots] \\ &+ \sum_{\mathsf{t}'=0}^{\mathsf{t}+1}\frac{\partial C_{\mathsf{t}'}}{\partial z^i}\frac{\partial}{\partial C_{\mathsf{t}'}}\ln P_{\mathsf{t}'}[y; M_{\mathsf{t}'}(z), C_{\mathsf{t}'}(z), \dots]\,.\end{aligned} \tag{2.34}$$

By factoring derivatives in this manner, the computation can be separated into two parts: the derivatives with respect to $M_{\mathsf{t}'}$ and $C_{\mathsf{t}'}$, which are typically quite straightforward; and the derivatives with respect to z elements, which typically need a more involved calculation depending on the model. Incidentally, this separation also neatly lends itself to abstracting gradient calculations as having a

[1] One can think of using this neural network-based reweighting scheme as similar to constructing a normalising flow model [8] with an autoregressive layer. Invertibility and further network structural constraints mean that these are not exactly equivalent, however.

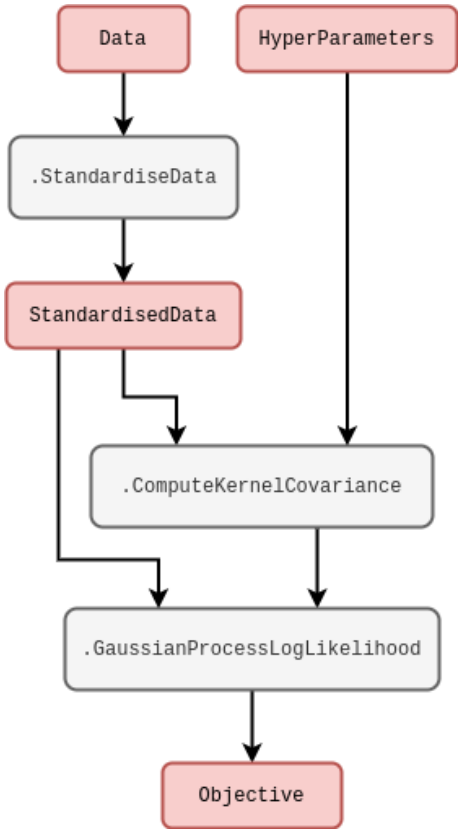

Figure 2.5: Code schematic of the Gaussian process objective computation.

simpler, general purpose component that can be built directly into a library of data models and a more complex, model-specific component that the user must specify.

The same logic should apply to a learning algorithm which optimises z to obtain the MAP for a Gaussian process. If the gradients of Eq. (2.33) are required, the user would have to specify derivatives of the kernel matrix (and its determinant) with respect to each hyperparameter z^k in order to calculate

$$\frac{\partial}{\partial z^k} \ln \mathcal{L}_{\mathrm{t}+1}(Y|z) = -\frac{1}{2} \sum_{\mathrm{t}'=0}^{\mathrm{t}+1} \sum_{\mathrm{t}''=0}^{\mathrm{t}'} \left[\frac{\partial}{\partial z^k} \ln \left| \mathcal{H}_{\mathrm{t}'\mathrm{t}''}(z) \right| + \sum_{i=0}^{n} \sum_{j=0}^{n} Y^i_{\mathrm{t}'} \frac{\partial}{\partial z^k} \mathcal{H}^{ij}_{\mathrm{t}'\mathrm{t}''}(z) Y^j_{\mathrm{t}''} \right] . \tag{2.35}$$

2.3 Online learning with any algorithm

We have discussed the importance of probabilistic learning in the context of environments where only partial state observability is possible and, in the previous chapter, we motivated the use of some specific probabilistic learning methods. However, we haven't yet discussed how we might implement a learning algorithm in practice. In particular, before covering the various aspects of

software design, it's important to consider how we want to structure learning by optimisation of an objective with respect to a stream of time series data.

One of the issues that can arise when learning streams of data is 'concept drift'. In our context, this would be when the optimal value for z does not match the optimal value at some later point in time. In order to mitigate this, our learning algorithms should be able to track an up-to-date optimal value for z as data is continually passed into them. Iteratively updating the optimal parameters as new data is ingested into the objective function is typically called 'online learning' [15, 16], in contrast to 'offline learning' which would correspond to learning an optimal z only once with the entire dataset provided upfront.

The reader may recall that this book is about building more realistic environments for machine learning systems. An important part of learning from environments in a robust manner is ensuring *adaptability to new data*. In addition to this, stochastic processes are inherently sequential. Many types of system evolve not just their states, but also dynamical description, over time. Online learning is the natural framework to use in this context.

Let's return to the models we discussed in the previous chapter which optimise the cumulative log-likelihood of the data matrix Y with respect to z at a particular point in time, i.e., which optimise $\ln \mathcal{L}_{\mathtt{t}+1}(Y|z)$ with respect to z at time $\mathtt{t}+1$. The simplest (and most generally applicable) way to implement an online learning approach with any machine learning model is to rerun the whole optimisation algorithm for z after each new datapoint y has been received. Each time the optimisation is rerun, it will be using the entire batch of training data — here represented by the data matrix Y. This kind of 'batch' online learning will work for most of the standard machine learning algorithms, but the re-training process can take a long time to run in each instance.

One way to speed things up is to assume that the optimal value for z which was obtained from the previous data iterations is close to the one we will find in the most recent iteration. Hence, by inserting this previous value into the next run of the optimisation procedure as an initial guess, the algorithm will typically converge much more quickly to the optimum. This is more of a practical insight, but are there any quantitative methods which can reduce the amount of computation required to update z when the latest datapoint in the series has been received?

This is where 'pure' online learning comes in. When gradients of the log-likelihood are available, we can make things much more efficient. Consider $\mathcal{L}_{\mathtt{t}+1}(y|z)$ as the log-likelihood term for datapoint y such that the cumulative log-likelihood of the data matrix Y can be written as the following summation of terms

$$\ln \mathcal{L}_{\mathtt{t}+1}(Y|z) = \sum_{\mathtt{t}'=0}^{\mathtt{t}+1} \ln \mathcal{L}_{\mathtt{t}'}(y|z)\,. \tag{2.36}$$

If we now denote the z which we have learned from the data up to timestep $\mathtt{t}$ as $z_*(\mathtt{t})$, we may write the following expression which uses the gradient of the log-likelihood to learn from each new datapoint arrival

$$z^i_*(\mathtt{t}+1) = z^i_*(\mathtt{t}) - \alpha(\mathtt{t}+1,\dots)\frac{\partial}{\partial z^i}\ln \mathcal{L}_{\mathtt{t}+1}(y|z)\,, \tag{2.37}$$

which is based on a stochastic gradient descent (SGD) algorithm approach. In contrast to the more standard offline SGD approach — which would be applied to mini-batches of the data — this update is applied to z_* using the log-likelihood at each point in time in sequence.

In Eq. (2.37) the $\alpha(\mathtt{t}+1,\dots)$ function, or 'learning rate' function, returns a value which controls the step size towards the optimal value. As indicated by the arguments to this function, the learning

rate can be time-dependent, which allows the user to set a schedule of steps which help convergence in certain problems. Other inputs can depend on the specific flavour of stochastic gradient descent algorithm that is being run. For example, an Adam optimiser [14] makes use of the statistics computed from the history of gradient values obtained as the algorithm progresses.

Still need to discuss:

- batch learning algorithms — more like Gaussian processes
- 'pure' online learning algorithms — more like empirical probabilistic reweighting

Got to here in rewrite...

Note that other excellent online machine learning frameworks are available — see, e.g., River [17] and Vowpal Wabbit [18]. The motivation for designing our own probabilistic online learning software is to ensure maximal integration with the stochadex simulation engine. We'll aim to achieve this by designing the code to use the same data structures and and concepts as we used when building the stochadex, where possible. In the next section on software design, we will show how this can be done while still maintaining extensibility and interoperability with other machine learning libraries and APIs. So let's get on with it!

2.4 Simulation inference formalism

In Bayesian inference, one applies Bayes' rule to the problem of statistically inferring a model from some dataset. This typically involves the following formula for a posterior distribution

$$\mathcal{P}_{\mathsf{t}+1}(z|Y) \propto \mathcal{L}_{\mathsf{t}+1}(Y|z)\mathcal{P}(z)\,. \tag{2.38}$$

In the formula above, one relates the prior probability distribution over a parameter set $\mathcal{P}(z)$ and the likelihood $\mathcal{L}_{\mathsf{t}+1}(Y|z)$ of some data matrix Y up to timestep $\mathsf{t}+1$ given the parameters z of a model to the posterior probability distribution of parameters given the data $\mathcal{P}_{\mathsf{t}+1}(z|Y)$ up to some proportionality constant. All this may sound a bit technical in statistical language, so it can also be helpful to summarise what the formula above states verbally as follows: the initial (prior) state of knowledge about the parameters z we want to learn can be updated by some likelihood function of the data to give a new state of knowledge about the values for z (the 'posterior' probability).

From the point of view of statistical inference, if we seek to maximise $\mathcal{P}_{\mathsf{t}+1}(z|Y)$ — or its logarithm — in Eq. (2.38) with respect to z, we will obtain what is known as a maximum posteriori (MAP) estimate of the parameters. In fact, we have already encountered this metholodology in the previous chapter when discussing the algorithm which obtains the best fit parameters for the empirical probability reweighting. In this case; while it appears that we optimised the log-likelihood directly as our objective function, one can easily show that this is also technically equivalent obtaining a MAP estimate where one chooses a specfic prior $\mathcal{P}(z) \propto 1$ (typically known as a 'flat prior').

How might we calulate the posterior in practice with some arbitrary stochastic process model that has been defined in the stochadex? In order to make the comparison to a real dataset, any stochadex model of interest will always need to be able to generate observations which can be directly compared to the data. To formalise this a little; a stochadex model could be represented as a map from z to a set of stochastic measurements $\mathsf{Y}_{\mathsf{t}+1}(z), \mathsf{Y}_{\mathsf{t}}(z), \ldots$ that are directly comparable to the rows in the real data matrix Y. The values in Y may only represent a noisy or partial

measurement of the latent states of the simulation X, so a more complete picture can be provided by the following probabilistic relation

$$P_{\mathsf{t}+1}(\mathsf{y}|z) = \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x \, P_{\mathsf{t}+1}(\mathsf{y}|x) P_{\mathsf{t}+1}(x|z) \,, \tag{2.39}$$

where, in practical terms, the measurement probability $P_{\mathsf{t}+1}(\mathsf{y}|x)$ of $\mathsf{Y}_{\mathsf{t}+1} = \mathsf{y}$ given $X_{\mathsf{t}+1} = x$ can be represented by sampling from another stochastic process which takes the state of the stochadex simulation as input. Given that we have this capability to compare like-for-like between the data and the simulation; the next problem is to figure out how this comparison between two sequences of vectors can be done in a way which ensures the the statistics of the posterior are ultimately respected.

For an arbitrary simulation model which is defined by the stochadex, the likelihood in Eq. (2.38) is typically not describable as a simple function or distribution. While we could train the probability reweighting we derived in the previous chapter to match the simulation; to do this well would require having an exact formula for the conditional probability, and this is not always easy to derive in the general case. Instead, there is a class of Bayesian inference methods which we shall lean on to help us compute the posterior distribution (and hence the MAP), which are known as 'Likelihood-Free' methods [19, 20, 21, 22].

'Likelihood-Free' methods work by separating out the components of the posterior which relate to the closeness of rows in Y to the rows in Y from the components which relate the states X and parameters z of the simulation stochastically to Y. To achieve this separation, we can make use of chaining conditional probability like this

$$\mathcal{P}_{\mathsf{t}+1}(X, z|Y) = \int_{\Upsilon_{\mathsf{t}+1}} \mathrm{d}\mathsf{Y} \, \mathcal{P}_{\mathsf{t}+1}(\mathsf{Y}|Y) P_{\mathsf{t}+1}(X, z|\mathsf{Y}) \,, \tag{2.40}$$

where $\Upsilon_{\mathsf{t}+1}$ here corresponds to the domain of the simulated measurements matrix Y at time $\mathsf{t}+1$.

As we demonstrated in the previous chapter, it's possible for us to also optimise a probability distribution $\mathcal{P}_{\mathsf{t}'}(\mathsf{y}|Y) = P_{\mathsf{t}'}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}, \mathcal{C}_{\mathsf{t}'}, \dots)$ for each step in time to match the statistics of the measurements in Y as well as possible, given some statistics $\mathcal{M}_{\mathsf{t}'} = \mathcal{M}_{\mathsf{t}'}(Y)$ and $\mathcal{C}_{\mathsf{t}'} = \mathcal{C}_{\mathsf{t}'}(Y)$. Assuming the independence of samples (rows) in Y, this distribution can be used to construct the distribution over all of Y through the following product

$$\mathcal{P}_{\mathsf{t}+1}(\mathsf{Y}|Y) = \prod_{\mathsf{t}'=0}^{\mathsf{t}+1} P_{\mathsf{t}'}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}, \mathcal{C}_{\mathsf{t}'}, \dots) \,. \tag{2.41}$$

We do not necessarily need to obtain these statistics from the probability reweighting method, but could instead try to fit them via some other objective function. Either way, this represents a lossy *compression* of the data we want to fit the simulation to, and so the best possible fit is desirable; regardless of overfitting. This choice to summarise the data with statistics means we are using what is known as a Bayesian Synthetic Likelihood (BSL) method [20, 21] instead of another class of methods which approximate an objective function directly using a proximity kernel — known as Approximate Bayesian Computation (ABC) methods [19].

Let's consider a few concrete examples of $P_{\mathsf{t}'}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}, \mathcal{C}_{\mathsf{t}'}, \dots)$. If the data measurements were well-described by a multivariate normal distribution, then

$$P_{\mathsf{t}'}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}, \mathcal{C}_{\mathsf{t}'}, \dots) = \mathsf{MultivariateNormalPDF}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}, \mathcal{C}_{\mathsf{t}'}) \,, \tag{2.42}$$

Similarly, if the data measurements were instead better described by a Poisson distribution, we might disregard the need for a covariance matrix statistic $\mathcal{C}_{\mathsf{t}'}$ and instead use

$$P_{\mathsf{t}'}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}, \mathcal{C}_{\mathsf{t}'}, \dots) = \mathsf{PoissonPMF}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}) \,. \tag{2.43}$$

The more statistically-inclined readers may notice that the probability mass function here would require the integrals in Eq. (2.40) to be replaced with summations over the relevant domains.

Eq. (2.40) demonstrates how one can construct a statistically meaningful way to compare the sequence of real data measurements $Y_{\mathsf{t}+1}, Y_{\mathsf{t}}, \dots$ to their modelled equivalents $\mathsf{Y}_{\mathsf{t}+1}(z), \mathsf{Y}_{\mathsf{t}}(z), \dots$. But we still haven't shown how to compute $P_{\mathsf{t}+1}(X, z|\mathsf{Y})$ for a given simulation, and this can be the most challenging part. To begin with, we can reapply Bayes' rule and the chaining of conditional probability to find

$$P_{\mathsf{t}+1}(x, z|\mathsf{Y}) \propto P_{\mathsf{t}+1}(\mathsf{y}|z) P_{\mathsf{t}}(z|\mathsf{Y}') = P_{\mathsf{t}+1}(\mathsf{y}|x) P_{\mathsf{t}+1}(x|z) P_{\mathsf{t}}(z|\mathsf{Y}') \,, \tag{2.44}$$

where here $P_{\mathsf{t}}(z|\mathsf{Y}')$ is the probability of $\mathsf{Y}_{\mathsf{t}} = \mathsf{Y}'$.

The relationship between $P_{\mathsf{t}+1}(X|z)$ and previous timesteps can be directly inferred from the probabilistic iteration formula that we introduced in the previous chapter. So we can map probabilities of $X_{0:\mathsf{t}+1} = X$ throughout time and learned information about the state of the system can be applied from previous values, given z. But is there a similar relationship we might consider for $P_{\mathsf{t}+1}(z|\mathsf{Y})$? Yes there is! The marginalisation

$$P_{\mathsf{t}+1}(z|\mathsf{Y}) \propto \left[\int_{\Omega_{\mathsf{t}+1}} \mathrm{d}^n x \, P_{\mathsf{t}+1}(\mathsf{y}|x) P_{\mathsf{t}+1}(x|z) \right] P_{\mathsf{t}}(z|\mathsf{Y}') \,, \tag{2.45}$$

shows how the z updates can occur in an iterative fashion. The reader may also recognize the factor above in brackets as Eq. (2.39). To complete the picture, one can combine the X and z updates into a joint distribution update which takes the following form

$$P_{\mathsf{t}+1}(X, z|\mathsf{Y}) \propto P_{\mathsf{t}+1}(\mathsf{y}|x) P_{(\mathsf{t}+1)\mathsf{t}}(x|X', z) P_{\mathsf{t}}(X', z|\mathsf{Y}') \,. \tag{2.46}$$

We can also marginalise this distribution over the past state history rows to get a distribution over the latest state row $X_{\mathsf{t}+1} = x$ like this

$$P_{\mathsf{t}+1}(x, z|\mathsf{Y}) = \int_{\Omega_{\mathsf{t}}} \mathrm{d}X' P_{\mathsf{t}+1}(X, z|\mathsf{Y}) \propto P_{\mathsf{t}+1}(\mathsf{y}|x) \int_{\Omega_{\mathsf{t}}} \mathrm{d}X' P_{(\mathsf{t}+1)\mathsf{t}}(x|X', z) P_{\mathsf{t}}(X', z|\mathsf{Y}') \,. \tag{2.47}$$

In the next section, we're going to discuss how to translate all of this probabilistic language into some MAP inference algorithms. Before we do this, however, it will be instructive (particularly for 'online' learning algorithms) to consider what happens if the model changes over time and z needs to change in order to better represent the real data. In such situations, we propose to apply the same formula as Eq. (2.47) but instead replace the distribution over (X', z) on the right hand side with its 'past discounted' version[2] Formula below is wrong — reformulate in terms of a $\beta^{\mathsf{t}-\mathsf{t}'}$

[2]In the continuous-time version, this past-discounting factor can depend on the stepsize such that we replace

$$\beta^{\mathsf{t}-\mathsf{t}'} \longrightarrow \frac{1}{\beta[\delta t(\mathsf{t})]} \prod_{\mathsf{t}''=\mathsf{t}'}^{\mathsf{t}} \beta[\delta t(\mathsf{t}'')] \,.$$

Bayesian evidence weighting factor for each past timestep and the result will be a $\prod$ expression here not a $\sum$...

$$\int_{\Omega_{\mathsf{t}}} \mathrm{d}X' P_{\mathsf{t}}(X', z|\mathsf{Y}') \longrightarrow \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x' \beta^{\mathsf{t}-\mathsf{t}'} P_{\mathsf{t}'}(x', z|\mathsf{Y}') \,, \tag{2.48}$$

where $0 < \beta < 1$ and we recall the notation which considers distributions over the individual rows x' within the matrix X' in this new version. This time-dependent discount factor could be used to reduce the dependence of the update on data which is much further in the past, and hence will ultimately lead to a more responsive algorithm. This responsiveness would have to be balanced with the tradeoffs associated with discounting potentially valuable data that may offer greater long-term stability. Readers who are familiar with reinforcement learning may be starting to feel in familiar territory here — they will have to wait for the latter parts of the book to see more on discounting though!

2.5 Online learning the MAP

Eq. (2.46) tells us how to probabilistically translate the current state of knowledge about (x, z) forward through time in response to the arrival of new data. We also know how to connect the simulated measurements to the real data because Eq. (2.40) essentially gives us an objective function to maximise for each step in time. This is all great in theory; but in practice, this optimisation problem typically has several layers of difficulty to it. Since the model has been defined by its stochastically generated samples of measurements $\mathsf{Y}_{\mathsf{t}+1}(z), \mathsf{Y}_{\mathsf{t}}(z), \ldots$, the objective function will manifestly be stochastic too. Another layer of difficulty is that gradients of the objective function are not immediately computable and so navigation around the optimisation domain could be difficult, especially in high-dimensional problems. Lastly, given that the simulation model in the stochadex needs to be running multiple times for each timestep, we need a way of mitigating computational expense.

So how should we proceed? To solve this problem in the general case, Eqs. (2.40) and (2.46) tell us we need to synthesize the following components into a single algorithm:

1. A process $P_{(\mathsf{t}+1)\mathsf{t}}(x|X', z)$ which iterates the state matrix of the simulation X forward in time.
2. A process $P_{\mathsf{t}+1}(\mathsf{y}|x)$ which generates a simulated measurement from the simulated state x.
3. A probability distribution $P_{\mathsf{t}'}(\mathsf{y}; \mathcal{M}_{\mathsf{t}'}, \mathcal{C}_{\mathsf{t}'}, \ldots)$ which represents the posterior distribution of the simulated measurement vector y given an optimised compression of the real data into summary statistics.
4. Some way of representing samples from the distribution $P_{\mathsf{t}+1}(X, z|\mathsf{Y})$ so that their distribution can be updated and will coverge towards the posterior over (X, z).

Cite this nice paper which outlines all the recent kinds of simulation inference: [23].

Keep the heuristic Bayes posterior estimator method as it is an example of recursive Bayes estimation - it can also be used to filter the ensemble at every step to make a particle filter [24]. Amortize this online learning process by training a neural net to produce the best estimates for the filter from the input real data!

Before writing this up, should read this paper on efficient amortized inference using neural networks with `BayesFlow` here in particular: [25]. But also, should cite other works to make amortized inference more efficient by using neural networks to learn convenient functions of the Bayes factor in Evidence networks [26].

- amortized online inference of the posterior update over just z can be achieved by running lots of simulations and solving the inverse problem with the `y` outputs i.e., neural network modelling of the update in Eq. (2.45)

The algorithm is specifically: 1. if this is a refit step, sample new values for (X, z) for all members of the ensemble from the current (X, z) distribution points and run the iterations for all of these ensemble members from the back of the window all the way up to the current point in time (hence the full matrix X is sampled) 2. take all of the ensemble members a step forward in time 3. approximate the mode by computing the average values of z within the q-th percentile of the sampled probability mass (where q is set by the user and is ideally $< 68\%$) — this idea comes from nested sampling 4. stream in the data for the next point in time and go to 1.

As such an algorithm converges, we can recompute (and hence iteratively improve) the MAP estimate with respect to each iteration of the posterior.

Readers with some machine learning experience may be familiar with the classic exploration vs exploitation tradeoffs. It's clear that these tradeoffs will manifest in our case here when trying to strike a balance between iterating the posterior distribution and optimizing the current posterior with respect to (X, z) to compute the MAP.

Readers of the previous section may also have recognized that Eq. (2.46) contains the same conditional probability $P_{(\mathtt{t}+1)\mathtt{t}}(x|X', z)$ as the reweighting algorithm. This structure enables us to reuse all of the exposition we provided for the probabilistic reweighting and highlights how the reweighting itself can be used in the algorithm to optimise the posterior.

If we now synthesize both of these observations together, we can see how a stochastic variant of the well-known Expectation-Maximisation Algorithm [27, 28, 7] naturally emerges.

2.6 Software design

Let's now take a step back from the specifics of the probabilistic reweighting algorithm to introduce our new sofware package for this part of the book: the 'learnadex'. At its core, the learnadex algorithm adapts the stochadex iteration engine to iterate through streams of data in order to accumulate a global objective function value with respect to that data. The user may then choose which optimisation algorithm (or write their own) to use in order to leverage this objective for learning a better representation of the data.

As we discussed at the end of the last section, the algorithms in the learnadex are all applied in an 'online' fashion — refitting for the optimal hyperparameters z as new data is streamed into them. A challenging aspect of online learning is in managing the computational expense of recomputing the optimal value for z after each new datapoint is sent. To help with this; the user may configure the algorithm recompute the optimum value after larger batches of data have been ingested. The last value of optimum z will also frequently be close to the next optimum in the sequence, so using the former as the initial input into the optimisation routine for the latter is typically very valuable for aiding efficiency.

Reusing the `PartitionCoordinator` code of the stochadex to facilitate online learning makes neat use of software which has already been designed and tested in earlier chapters of this book.

However, in order to fully achieve this, an additional configuration type is necessary; as we show in Fig. 2.6. To start with, we separate out 'learning' from the kind of optimiser in the overall config so as to enable multiple optimisation algorithms to be used for the same learning problem. The hyperparameters that define that optimisation problem domain can be determined by the user with an extension to the `OtherParams` object in the stochadex so that it includes some optional Boolean masks over the parameters, i.e., `OtherParams.FloatParamMask` and `OtherParams.IntParamMask`. These masks are used to extract the parameters of interest, which can then be flattened and formatted to fit into any generic optimisation algorithm.

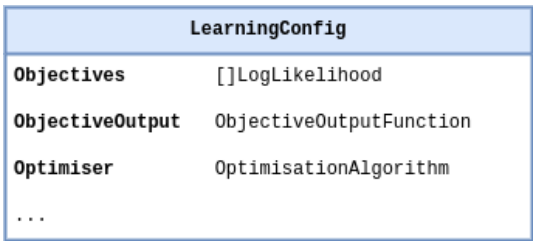

Figure 2.6: The learning config data type.

On the learning side; in order to define a specific objective for each data iterator to compute while the data streams through it, we have abstracted a 'log-likelihood' type. Each data iterator is defined as a standard stochadex iteration, hence it can stream data into the learning algorithm from any user-defined source — e.g., from a file on disk, from a local database instance or maybe via a network socket. In Fig. 2.7 below, we provide a schematic of the method calls of (and within) each data iterator.

- refactor the code and integrate the reweighting algorithm with Libtorch models for the conditional probabilities — describe how this is supported
- describe the method calls diagram in more detail — in particular, point out how it can replace the `Iteration.Iterate` method which is called when the `StateIterator` is asked for another iteration from the `PartitionCoordinator` of the stochadex
- then talk about the optimiser! starting with non-gradient-based: the two packages that are supported out of the box are gonum and eaopt (still need to do gago — see here: github.com/maxhalford/eaopt)
- also need to then support gradient-based algorithms (like vanilla SGD) by implementing Eq. (2.34) for the current basic implementations in the learnadex — shouldn't be too difficult!
- then talk about the output - talk about the possibilities for output and what the default setting to json logs is for
- could also be written to, e.g., a locally-hosted database server and the best-suited would be a NoSQL document database, e.g., MongoDB [29], but building something bespoke and simpler is more aligned with the use-case here and with the principles of this book

- describe the need for log exploration and visualisation and then introduce logsplorer - a REST API for querying the json logs (with basic filtering and selection capabilities but could be extended to more advanced options) and optionally also launches a visualisation React app written in Typescript
- note how this could be scaled to cloud services easily and remotely queried through the logsplorer API and visualised

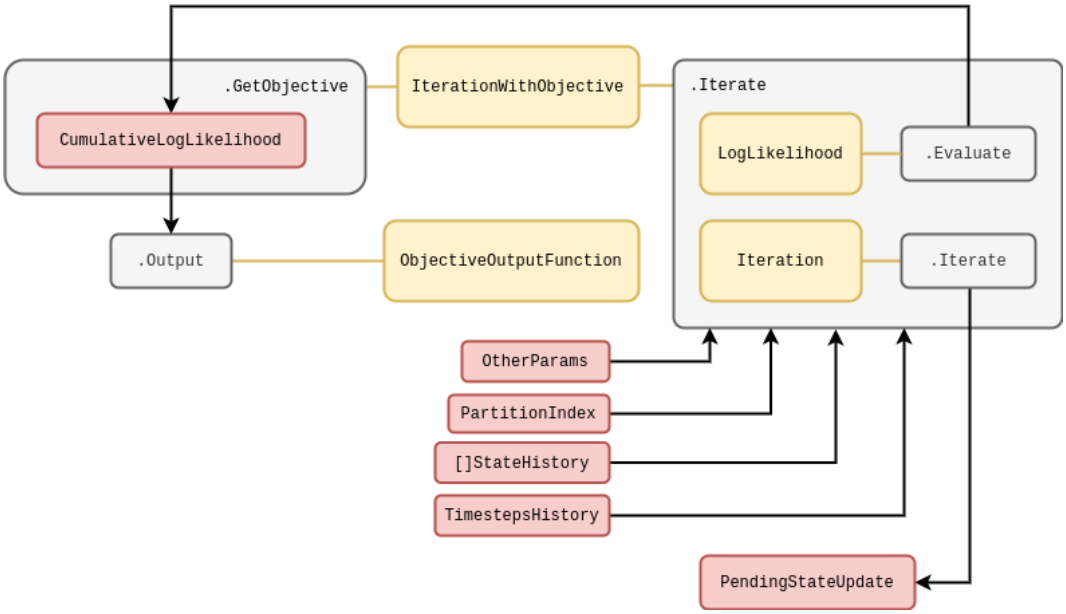

Figure 2.7: A schematic of an iteration with an objective function evaluation.

As with the software we wrote for the stochadex, the learnadex main binary executable leverages templating to enable full configurability of all the implementations and settings of Fig. 2.6 through passing configs at runtime. Users can alternatively use the learnadex as a library for import, if they desire more control over the code execution.

Take a step back at this point and consider all the use-cases for the learnadex:

- Core functionality is to enable iterative updates to the $P_t(X, z)$ distribution at every timestep. There must also be flexibility in how this distribution can represented, i.e., either:
  - a set of Monte Carlo samples, or
  - a set of distribution parameters
- For Monte Carlo samples, we must also keep the flexibility to use either a BSL or ABC-style data-to-sim comparison in order to facilitate the update
- For distribution parameters, the update can be custom-built by the user with online likelihood-based inference/Bayes estimator methods

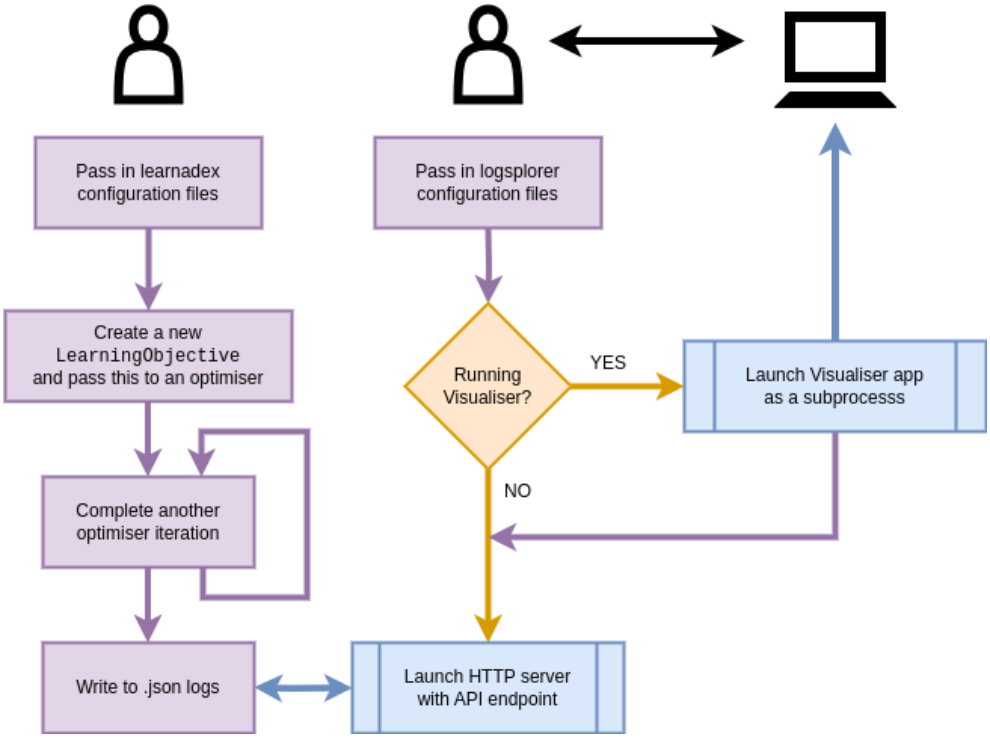

Figure 2.8: A diagram of the main learnadex and logsplorer executables.

- Separate thread context runs of online learning methods using an update method, e.g.,
  - Gradient-free batch optimisation with any chosen learning algorithm
  - Gradient-based parameter updates
  - Arbitrary parameter updates from a different method

Bibliography

[1] D. Boyer, M. R. Evans, and S. N. Majumdar, "Long time scaling behaviour for diffusion with resetting and memory," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2017, no. 2, p. 023208, 2017.

[2] H. A. Kramers, "Brownian motion in a field of force and the diffusion model of chemical reactions," *Physica*, vol. 7, no. 4, pp. 284–304, 1940.

[3] J. Moyal, "Stochastic processes and statistical physics," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 11, no. 2, pp. 150–210, 1949.

[4] A. G. Hawkes, "Spectra of some self-exciting and mutually exciting point processes," *Biometrika*, vol. 58, no. 1, pp. 83–90, 1971.

[5] G. Sugihara and R. M. May, "Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series," *Nature*, vol. 344, no. 6268, pp. 734–741, 1990.

[6] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[7] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[8] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 3964–3979, 2020.

[9] L. Pinheiro Cinelli, M. Araújo Marins, E. A. Barros da Silva, and S. Lima Netto, "Variational autoencoder," in *Variational Methods for Machine Learning with Applications to Deep Networks*. Springer, 2021, pp. 111–149.

[10] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[11] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. IEEE, 1995, pp. 1942–1948.

[12] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*. IEEE, 1998, pp. 69–73.

[13] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] E. Hazan *et al.*, "Introduction to online convex optimization," *Foundations and Trends® in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.

[16] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[17] "River: Online machine learning in Python," https://riverml.xyz/latest/, accessed: 2023-12-30.

[18] "Vowpal Wabbit: Your go-to interactive machine learning library," https://vowpalwabbit.org/, accessed: 2023-12-30.

[19] S. A. Sisson, Y. Fan, and M. Beaumont, *Handbook of approximate Bayesian computation*. CRC Press, 2018.

[20] L. F. Price, C. C. Drovandi, A. Lee, and D. J. Nott, "Bayesian synthetic likelihood," *Journal of Computational and Graphical Statistics*, vol. 27, no. 1, pp. 1–11, 2018.

[21] S. N. Wood, "Statistical inference for noisy nonlinear ecological dynamic systems," *Nature*, vol. 466, no. 7310, pp. 1102–1104, 2010.

[22] C. Drovandi and D. T. Frazier, "A comparison of likelihood-free methods with and without summary statistics," *Statistics and Computing*, vol. 32, no. 3, p. 42, 2022.

[23] K. Cranmer, J. Brehmer, and G. Louppe, "The frontier of simulation-based inference," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 055–30 062, 2020.

[24] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking," *IEEE Transactions on signal processing*, vol. 50, no. 2, pp. 174–188, 2002.

[25] S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe, "Bayesflow: Learning complex stochastic models with invertible neural networks," *IEEE transactions on neural networks and learning systems*, vol. 33, no. 4, pp. 1452–1466, 2020.

[26] N. Jeffrey and B. D. Wandelt, "Evidence networks: simple losses for fast, amortized, neural bayesian model comparison," *arXiv preprint arXiv:2305.11241*, 2023.

[27] H. O. Hartley, "Maximum likelihood estimation from incomplete data," *Biometrics*, vol. 14, no. 2, pp. 174–194, 1958.

[28] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society: series B (methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[29] "The MongoDB Webpage," https://www.mongodb.com/, accessed: 2023-08-17.