3

Empirical probabilistic reweighting algorithms

Concept. To extend the formalism that we developed in previous chapters to enable the empirical emulation of real-world data via a probabilistic reweighting. This technique should enable a researcher to model complex dynamical trends in the data very well; at the cost of making the abstract interpretation of the model less immediately comprehensible than the statistical inference models in some proceeding chapters. For the mathematically-inclined, this chapter will take a detailed look at how our formalism can be extended to focus on probabilistic reweightings and their optimisation using real-world data. For the programmers, the software described in this chapter lives in the public Git repository: https://github.com/umbralcalc/learnadex.

3.1 Probabilistic formalism

The key distinction between the methods that we will develop in this chapter and the ones in the proceeding chapters is in their utility when faced with the problem of attempting to model real-world data. In the proceeding chapter, we shall describe some powerful techniques that can be used most effectively when the researcher is aware of the family of models that generated the data. In the present chapter, we will go into the details of how a more 'empirical' approach can be derived for dynamical process modeling in a probabilistic framework which locally adapts the model to the data through time.

While we think that it's worth going into some mathematical detail to give a better sense of where our formalism comes from; we want to emphasise that the framework we discuss here is not new to the technical literature at all. Our overall framework draws on influences from Empirical Dynamical Modeling (EDM) [1], some classic nonparametric local regression techniques — such as LOWESS/Savitzky-Golay filtering [2] — and also Gaussian processes [3] as well. The novelties here, instead, lie more in the specifics of how we combine some of these ideas together when referencing the stochadex formalism, and how this manifests in designing more generally-applicable software

for the user. In addition, readers well-versed in machine learning will note that our software design and mathematical formalism reflects our preference for 'online' learning[1] in the context of time series prediction, in contrast to some of the more standard frameworks.

When trying robustly assess how far a model is from accurately describing a set of real-world data, trying to use only generated samples of the model process can be diffcult. Instead, in this section, we are going to extend this formalism to look at how probability theory can help with this data comparison problem in a systematic way. In order to do this, we need to return to the probabilistic formalism which we discussed in the previous chapter.

Given the general master equation $P_{\mathsf{t}+1}(X|z) = P_{\mathsf{t}}(X'|z)P_{(\mathsf{t}+1)\mathsf{t}}(x|X', z)$, if we wanted to compute the mean $M_{\mathsf{t}+1}(z)$ of the distribution over the matrix row corresponding to time $(\mathsf{t}+1)$, it would be straightforward to just multiply both sides by x and integrate over it in its $\omega_{\mathsf{t}+1}$ sub-domain. However, there is another similar expression for the mean that we can derive under certain conditions which will be valuable to us when developing the empirical reweighting. 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 derive

$$M_{\mathsf{t}+1}(z) = \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, x\, P_{\mathsf{t}+1}(x|z) = \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x'\, x'\, P_{\mathsf{t}'}(x'|z) \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)\,. \tag{3.1}$$

To see that Eq. (3.1) 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

$$\begin{aligned}
C^{ij}_{\mathsf{t}+1}(z) &= \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, [x - M_{\mathsf{t}+1}(z)]^i [x - M_{\mathsf{t}+1}(z)]^j P_{\mathsf{t}+1}(x|z) \\
&= \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} \int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x'\, [x' - M_{\mathsf{t}+1}(z)]^i [x' - M_{\mathsf{t}+1}(z)]^j\, P_{\mathsf{t}'}(x'|z) \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)\,.
\end{aligned} \tag{3.2}$$

While they look quite abstract, Eqs. (3.1) and (3.2) express the core idea behind how our 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

[1] We'll explain this later in the chapter.

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

It may seem needlessly more complex to deal with probability distributions over matrices instead of marginal distributions over separate vectors which represent the rows of these matrices. However, the matrix description is more general, and it turns out to be quite neat to describe correlations across time (which would be lost by marginal distributions over row vectors). In order to study these out-of-time-order correlations, we need only consider using the statistical moments computed across the columns of X. For example, pairwise correlations can be analysed through the out-of-time-order covariance matrix elements

$$C^{ij}_{(\mathsf{t}+1)\mathsf{t}'}(z) = \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x \int_{\omega_{\mathsf{t}'}} \mathrm{d}^n x' [x - M_{\mathsf{t}+1}(z)]^i [x' - M_{\mathsf{t}'}(z)]^j P_{\mathsf{t}+1}(X|z)\,. \tag{3.3}$$

- Need to finish the descriptions in this section to move onto the next...

3.2 Online learning the optimal reweighting

Probabilistic reweighting depends on the stationarity of $P_{\mathsf{t}+1}(x|z) = P_{\mathsf{t}'}(x|z)$ such that, e.g., Eq. (3.1) 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

$$\mathsf{w}_{\mathsf{t}'}(y, z) = \int_{\omega_{\mathsf{t}+1}} \mathrm{d}^n x\, P_{(\mathsf{t}+1)\mathsf{t}'}(x|x'{=}y, z)\,, \tag{3.4}$$

we can apply this as a *reweighting* of the historical time series samples to estimate any statistics of interest. Taking Eqs. (3.1) and (3.2) as the examples; we are essentially approximating these integrals through weighted sample estimations like this

$$M_{\mathsf{t}+1}(z) \simeq \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} Y_{\mathsf{t}'} \mathsf{w}_{\mathsf{t}'}(Y_{\mathsf{t}'}, z) \tag{3.5}$$

$$C^{ij}_{\mathsf{t}+1}(z) \simeq \frac{1}{\mathsf{t}} \sum_{\mathsf{t}'=0}^{\mathsf{t}} [Y_{\mathsf{t}'} - M_{\mathsf{t}+1}(z)]^i [Y_{\mathsf{t}'} - M_{\mathsf{t}+1}(z)]^j \mathsf{w}_{\mathsf{t}'}(Y_{\mathsf{t}'}, z)\,, \tag{3.6}$$

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.

Our goal in this section will 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. (3.1), 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. (3.2). These are just a few simple examples of distributions that can link the estimated statistics from Eqs. (3.1) and (3.2) 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 $P_{\mathsf{t}+1}[y; M_{\mathsf{t}+1}(z), C_{\mathsf{t}+1}(z), \dots]$ as representing the likelihood of $y = Y_{\mathsf{t}+1}$ given the estimated statistics from Eqs. (3.1) and (3.2) (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

$$\ln \mathcal{L}_{\mathsf{t}+1}(Y|z) = \sum_{\mathsf{t}'=0}^{\mathsf{t}+1} \ln P_{\mathsf{t}'}[y; M_{\mathsf{t}'}(z), C_{\mathsf{t}'}(z), \dots] \,, \tag{3.7}$$

where the summation continues until all of the past measurements $Y_{\mathsf{t}+1}, Y_{\mathsf{t}}, \dots$ 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. 3.1.

In order to specify what $P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z)$ 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

$$P_{(\mathsf{t}+1)\mathsf{t}'}(x|x', z) \propto \mathcal{K}(z, \mathsf{t}+1, \mathsf{t}') \,, \tag{3.8}$$

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

$$P_{\mathsf{t}+1}[y; M_{\mathsf{t}+1}(z), C_{\mathsf{t}+1}(z), \dots] = \mathsf{MultivariateNormalPDF}[y; M_{\mathsf{t}+1}(z), C_{\mathsf{t}+1}(z)] \,. \tag{3.9}$$

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.[2] 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) [5]; 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. (3.8) and (3.9) 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 very similar to (but not quite the same as) evaluating maximum a posteriori (MAP) of a Gaussian process regression. The main differences here are that the mean of a Gaussian process as a function of time is typically included within z, and hence must be obtained through optimisation. In

[2] One can think of using this neural network-based reweighting scheme as similar to constructing a normalising flow model [4] with an autoregressive layer. Invertibility and further network structural constraints mean that these are not exactly equivalent, however.

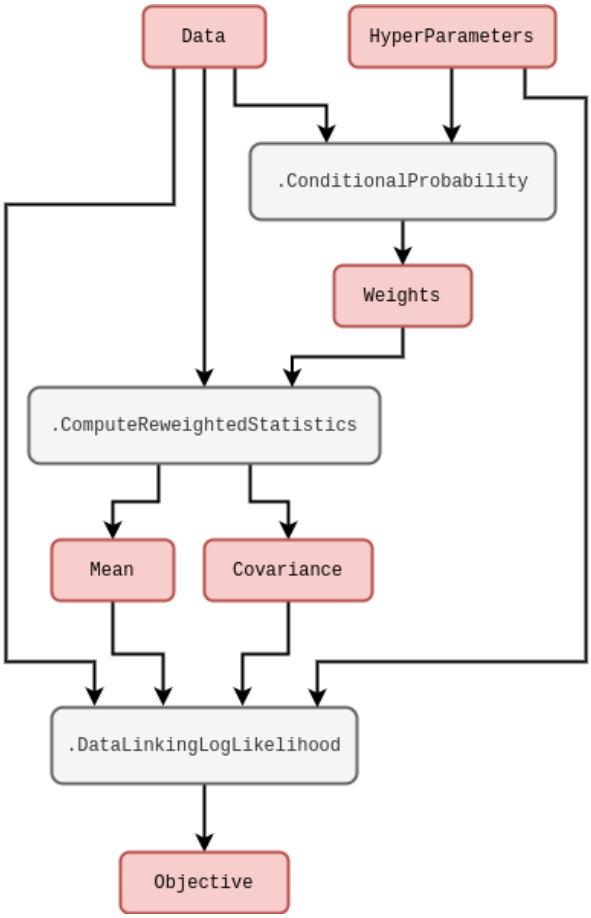

Figure 3.1: Code schematic of the probability reweighting optimisation.

contrast, our methodology relies on the fact that the mean estimator can be computed directly by weighted sample estimation and can then be fed to some data-linking distribution. By doing it this way, we enable many different kinds of data to be described by the same underlying conditional probability-based reweighting and also make incorporating future data into the current model much easier then is the case for a standard Gaussian process (which would require re-optimising with respect to the next data point every time). The time-dependent kernel we have chosen in Eq. (3.8) also only represents one particular choice, but we could consider a wide range of state-dependent conditional probability weightings for the algorithm as well.

As another form of flexibility; we could also try adapting the data-linking distributions to include an intercept term and linear coefficients for the statistics which are passed to it. These could then be treated as additional hyperparameters and optimised jointly with the others if there is sufficient constraining power in the data.

The optimisation approach that we choose to use for obtaining the best hyperparameters in the conditional probability of Eq. (3.7) will depend on a few factors. For example, if the number of hy-

perparameters is relatively low, but their gradients are difficult to calculate exactly; then a gradient-free optimiser (such as the Nelder-Mead [6] method or something like a particle swarm [7, 8]) 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 [9] (SGD) or Adam [10].

If the gradients of Eq. (3.7) are needed, we can always factorise each derivative with respect to hyperparameter z^i in the following way through the chain rule

$$\frac{\partial}{\partial z^i} \ln \mathcal{L}_{\mathtt{t}+1}(Y|z) = \sum_{\mathtt{t}'=0}^{\mathtt{t}+1} \frac{\partial M_{\mathtt{t}'}}{\partial z^i} \frac{\partial}{\partial M_{\mathtt{t}'}} \ln P_{\mathtt{t}'}[y; M_{\mathtt{t}'}(z), C_{\mathtt{t}'}(z), \dots] + \sum_{\mathtt{t}'=0}^{\mathtt{t}+1} \frac{\partial C_{\mathtt{t}'}}{\partial z^i} \frac{\partial}{\partial C_{\mathtt{t}'}} \ln P_{\mathtt{t}'}[y; M_{\mathtt{t}'}(z), C_{\mathtt{t}'}(z), \dots] \,. \tag{3.10}$$

By factoring derivatives in this manner, the computation can be separated into two parts: the derivatives with respect to $M_{\mathtt{t}'}$ and $C_{\mathtt{t}'}$, 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 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.

Before moving on to the software design aspects, we need to consider how we might structure learning by optimisation of Eq. (3.7) for a sequence of observations in time. 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 solve this issue, our learning algorithm should track an up-to-date optimal value for z as data is continually passed into it. Iteratively updating the optimal parameters as new data is ingested into the objective function is typically called 'online learning' [11, 12], in contrast to 'offline learning' which would correspond to learning an optimal z only once with the entire dataset provided upfront.[3]

3.3 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

[3] This book will mostly be focussed on using online learning techniques due to the inherently sequential nature of stochastic processes.

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, a few minor extensions to the typing structures and code abstractions are necessary; as we show in Fig. 3.2. 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 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.

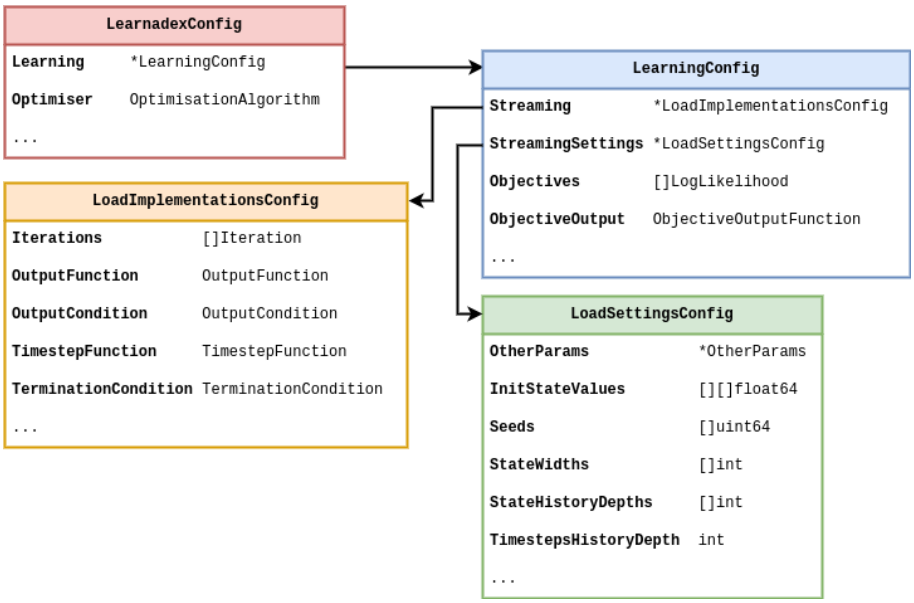

Figure 3.2: A relational summary of the core data types in the learnadex.

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. Similarly, each iterator also gets a data streamer configuration which defines where the data is streaming from — e.g., from a file on disk, from a local database instance or maybe via a network socket — and also some inherited abstractions from the stochadex which define the time stepping function and when the data stream ends. In Fig. 3.3 below, we provide a schematic of the method calls of (and within) each data iterator.

- introduce the β past discounting factor in this section and explain what it's for
- refactor the code so that it's always doing online learning under the hood — this can either

be rolling refits in blocks on a refitting schedule with any optimisation algorithm of choice or full online learning Adam optimisation

$$z_*(\mathtt{t}+1) = -\alpha[\mathtt{t}+1, \text{stats of gradient history like Adam}]\frac{\partial}{\partial z}\ln\mathcal{L}_{\mathtt{t}+1} + z_*(\mathtt{t})$$

- 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. (3.10) 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 [13], 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

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. 3.2 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.

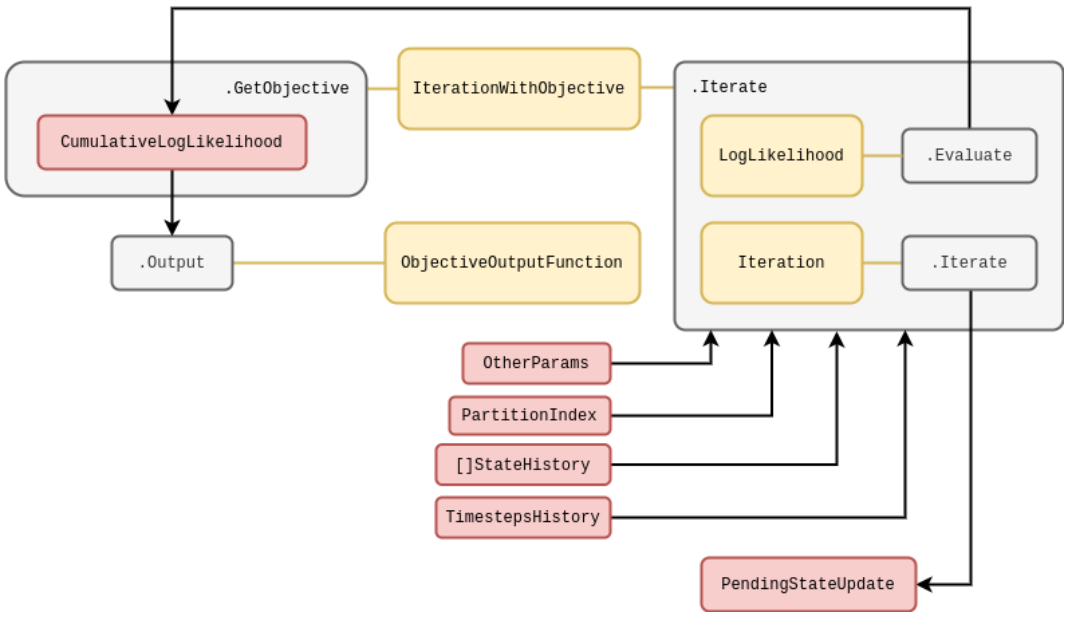

Figure 3.3: A schematic of an iteration with an objective function evaluation.

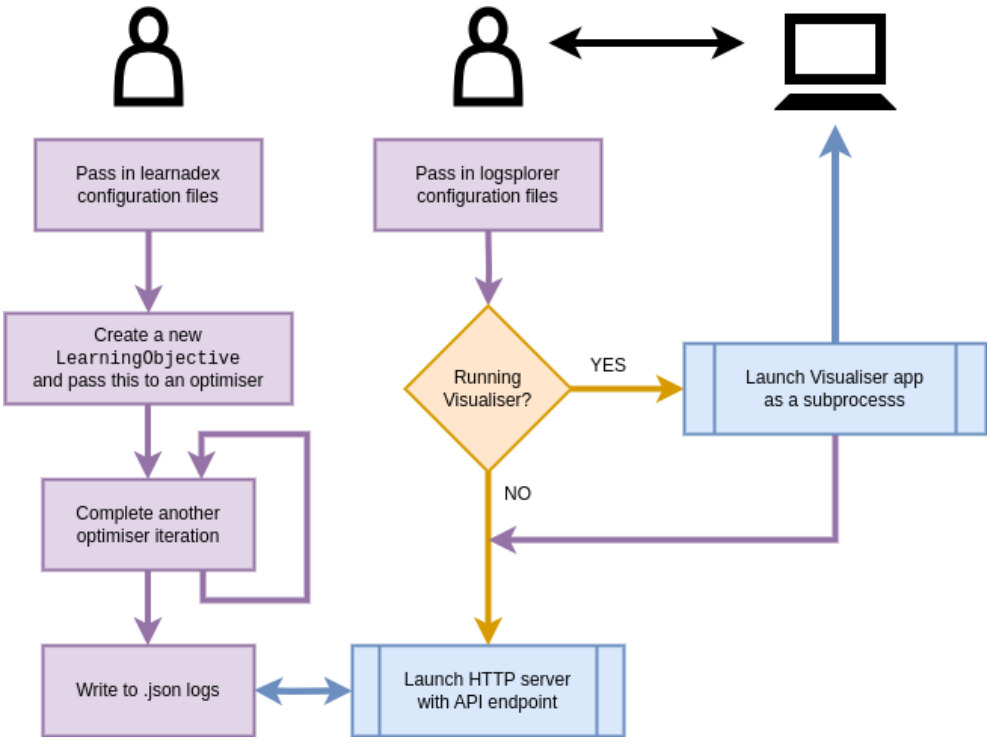

Figure 3.4: A diagram of the main learnadex and logsplorer executables.

Bibliography

[1] G. Sugihara and R. M. May, *Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series*, *Nature* **344** (1990) 734–741.

[2] A. Savitzky and M. J. Golay, *Smoothing and differentiation of data by simplified least squares procedures.*, *Analytical chemistry* **36** (1964) 1627–1639.

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

[4] 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* **43** (2020) 3964–3979.

[5] 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*, pp. 111–149. Springer, 2021.

[6] J. A. Nelder and R. Mead, *A simplex method for function minimization*, *The computer journal* **7** (1965) 308–313.

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

[8] 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)*, pp. 69–73, IEEE, 1998.

[9] H. Robbins and S. Monro, *A stochastic approximation method*, *The annals of mathematical statistics* (1951) 400–407.

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

[11] E. Hazan et al., *Introduction to online convex optimization*, *Foundations and Trends® in Optimization* **2** (2016) 157–325.

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

[13] "The MongoDB Webpage." https://www.mongodb.com/.