

Probabilistic online learning software

Concept. To outline the design characteristics of a probabilistic online learning framework, built to interface directly with our generalised simulation engine. The probabilistic algorithms we introduced in the previous chapter will form the basic set of tools in this framework. As well as its own implementations of useful algorithms, this software is also designed to make use of other machine learning APIs, e.g., Libtorch (via Gotch), and optimisation libraries, e.g., Gonn and eoapt. For the mathematically-inclined, this chapter will motivate online learning as an essential framework for our use case and how our probabilistic algorithms can work within this context. For the programmers, the software described in this chapter lives in the public Git repository: <https://github.com/umbralcalc/learnadex>.

Got to [here](#) in rewrite...

3.1 Online learning with any algorithm

Before discussing the software design aspects, we need to consider how we might structure learning by optimisation of Eq. (??) 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’ [1, 2], in contrast to ‘offline learning’ which would correspond to learning an optimal z only once with the entire dataset provided upfront.¹

¹This book will mostly be focussed on using online learning techniques due to the inherently sequential nature of stochastic processes.

3.2 Software design

Let’s now take a step back from the specifics of the probabilistic reweighting algorithm to introduce our new software 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, a few minor extensions to the typing structures and code abstractions are necessary; as we show in Fig. 3.1. 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.

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. 3.2 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

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

- refactor the code and integrate the reweighting algorithm with Libtorch models for the conditional probabilities — describe how this is supported

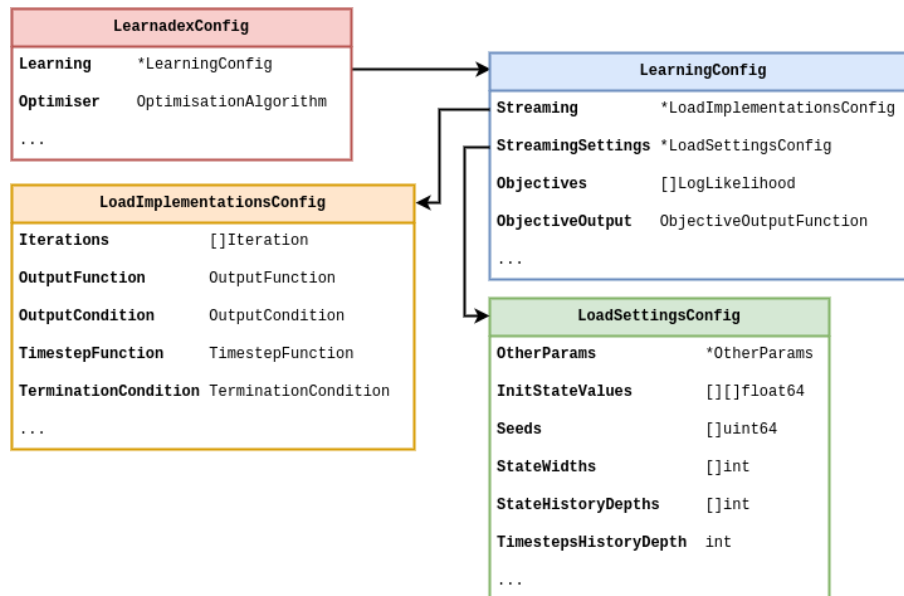


Figure 3.1: A relational summary of the core data types in the learnadex.

- 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. (??) 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 [3], 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 `logexplorer` - 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 `logexplorer` API and visualised

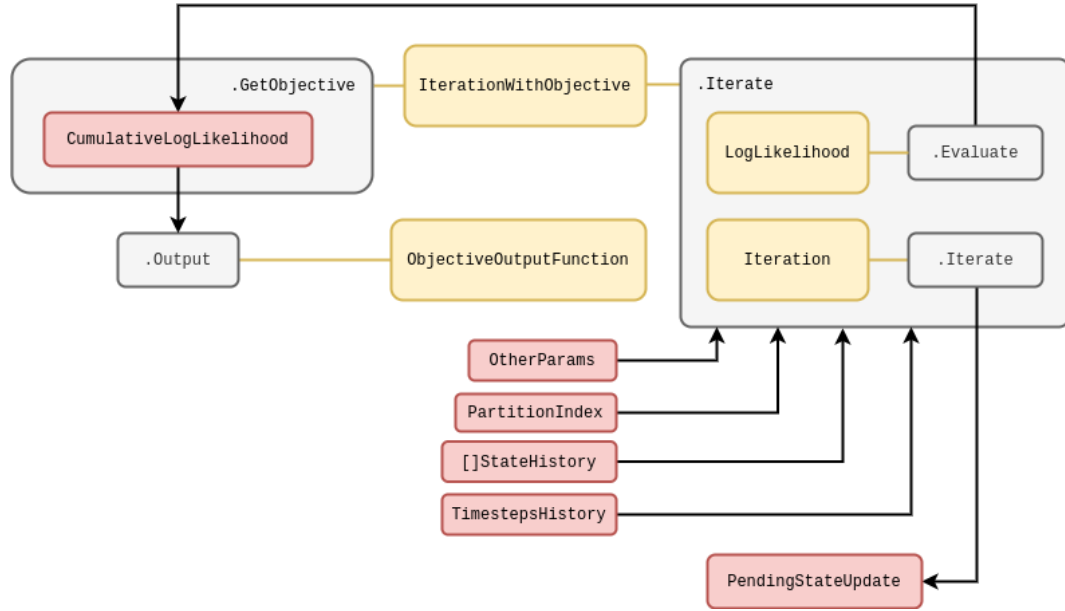


Figure 3.2: A schematic of an iteration with an objective function evaluation.

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. 3.1 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.

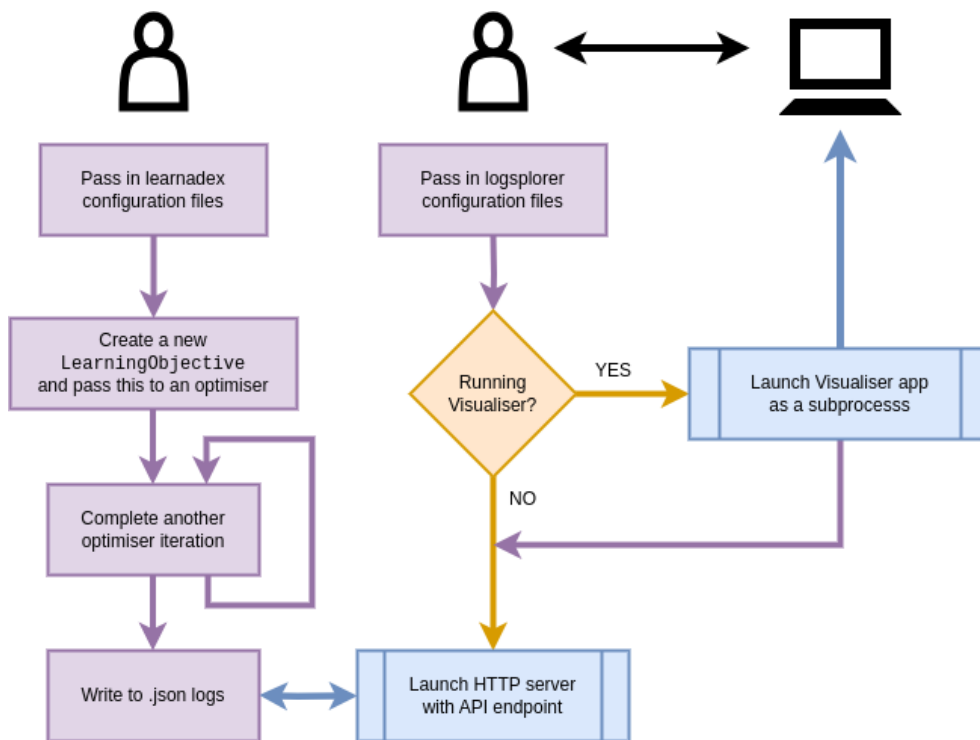


Figure 3.3: A diagram of the main learnadex and logsporer executables.

Bibliography

- [1] E. Hazan *et al.*, “Introduction to online convex optimization,” *Foundations and Trends® in Optimization*, vol. 2, no. 3-4, pp. 157–325, 2016.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [3] “The MongoDB Webpage,” <https://www.mongodb.com/>, accessed: 2023-08-17.