WAX-ML: A Python library for machine learning and feedback loops on streaming data

Emmanuel Sérié   ESERIE@GMAIL.COM

Capital Fund Management
23 rue de l'Université
75007 Paris, France

Abstract

Wax is what you put on a surfboard to avoid slipping. It is an essential tool to go surfing ... We introduce WAX-ML a research-oriented Python library providing tools to design powerful machine learning algorithms and feedback loops working on streaming data. It strives to complement JAX with tools dedicated to time series. WAX-ML makes JAX-based programs easy to use for end-users working with pandas and xarray for data manipulation. It provides a simple mechanism for implementing feedback loops, allows the implementation of online learning and reinforcement learning algorithms with functions, and makes them easy to integrate by end-users working with the object-oriented reinforcement learning framework from the Gym library. It is released with an Apache open-source license on GitHub at `https://github.com/eserie/wax-ml`.

Keywords: machine-learning, streaming data, time series, feedback loops, Python, NumPy, JAX, Gym

1. Introduction

Since the advent of machine learning and its considerable development over the last 20 years many tools and computer programs have been developed in this field. The Python programming language (see Van Rossum and Drake (2009)), which is a very flexible and easy-to-use programming language, has contributed greatly to making machine learning programs more accessible.

Since its creation, a whole system of tools has been built to allow the implementation and access to efficient numerical computation methods. To begin with, the NumPy library, created in 2005 (see Harris et al. (2020) for a recent review), allows to perform numerical computations and exposes, with a simple API, multi-dimensional numerical arrays, and a set of functions implementing mathematical operations on these arrays.

Subsequently, multiple tools have been built in different and complementary directions to go further in the type of transformations that can be performed.

The Scikit-learn library, released in 2010 (see Buitinck et al. (2013)), has proposed a high-level object-oriented API for exposing a very large set of machine learning methods for supervised and unsupervised problems and has gathered a large international community around it.

On the other hand, the pandas library (see Wes McKinney (2010)), launched in 2009 , has built, on top of NumPy, a high-level powerful and flexible dataframe API that is better suited for data analysis and manipulation tools. In particular, it includes "time-series

functionalities such as date range generation and frequency conversion, moving window statistics, date shifting and lagging, creation of domain-specific time offsets and joining of time-series". It also strives to have highly performance-optimized implementations with critical code paths written in Cython or C.

Some time-series oriented machine learning tools have been released, such as Prophet (see Taylor and Letham (2017)) in 2017 which is dedicated to end-users working on time series and wanting to do prediction with black box statistical modeling techniques.

The TensorFlow library, released in 2015 (see Abadi et al. (2015)), proposed a set of tools to implement deep neural networks and has since built a large community of developers and users around it. The TensorFlow developers rebuilt a tensor API in Python, aiming to be compatible with NumPy but not completely, which made TensorFlow a separate ecosystem from that built around NumPy. In particular, this separation sometimes makes it difficult to develop research ideas in a short development cycle, often requiring reimplementation of NumPy code in TensorFlow. On the other hand, its widespread adoption in the research community and industry for the implementation of deep-learning algorithms has led to the development of very powerful optimization systems such as XLA (Accelerated Linear Algebra) (see xla), a domain-specific linear algebra compiler that can accelerate TensorFlow models with potentially no changes to the source code, resulting in improvements in speed and memory usage (see mlp (2020)).

Since then, other libraries have been built to allow the implementation of XLA programs outside of TensorFlow with other systems such as JAX (see Bradbury et al. (2018)), Julia (see Bezanson et al. (2017)), PyTorch (see Paszke et al. (2019)), Nx (see Team).

In particular, JAX, a library released in 2018 (see Bradbury et al. (2018)), has initiated a path to solving the "ecosystem separation" problem explained above, by providing a functional research-oriented API in Python for writing XLA programs with a NumPy API and primitive transformations to perform just-in-time XLA compilation, automatic differentiation, parallelization, and vectorization.

The approach followed by the JAX developers has been to propose some quite low-level primitives working with well-identified programming constraints, such as working with pure functions. This constraint seems to be a good compromise between maintaining the flexibility of Python and having the possibility to optimize numerical computations with systems like XLA. This approach suggests writing numerical Python programs in a functional programming style afterward. This is not very natural in a language like Python

which is at its core an object-oriented programming language. Once the constraints given by the functional programming approach of JAX are "digested", this can become a strength since functional programming allows one to write notably modular, reusable, testable programs, which aim at making JAX an efficient tool to implement research ideas.

This bet taken by JAX has somehow worked since a whole ecosystem has been built around it with the advent of domain-specific libraries. For instance, Optax (see Hessel et al. (2020)), "a gradient processing and optimization library for JAX", Haiku (see Hennigan et al. (2020)), "a simple neural network library for JAX", RLax (see Budden et al. (2020)), "a library built on top of JAX that exposes useful building blocks for implementing reinforcement learning agents", Flax (see Heek et al. (2020)), "a neural network library and ecosystem for JAX designed for flexibility", . . .

With the launch of WAX-ML, we want to complement this growing ecosystem with tools that make it easier to implement machine learning algorithms on streaming data.

We want to push forward the efforts initiated by JAX developers to reconcile the Python ecosystems around numerical computations by proposing tools allowing the use of JAX programs on time-series data thanks to the use of the pandas and xarray (see Hoyer and Hamman (2017)) libraries which are, for now, reference libraries for the manipulation of labeled datasets and in particular time-series data.

We even think that the TensorFlow and PyTorch ecosystem could also be made compatible with programs implemented with JAX ecosystem tools thanks to EagerPy (see Rauber et al. (2020)) which is a library that allows writing code that works natively with PyTorch, TensorFlow, JAX, and NumPy. In this first released version of WAX-ML, we use EagerPy to facilitate the conversion between NumPy tensors (which are used by pandas and xarray) and JAX tensors. We are also implementing some universal modules so that we can work with other tensor libraries, but for now, we remain focused on running JAX programs on streaming data.

2. WAX-ML goal

WAX-ML's goal is to expose "traditional" algorithms that are often difficult to find in standard Python ecosystem and are related to time-series and more generally to streaming data.

It aims to make it easy to work with algorithms from very various computational domains such as machine learning, online learning, reinforcement learning,

optimal control, time-series analysis, optimization, statistical modeling.

For now, WAX-ML focuses on time-series algorithms as this is one of the areas of machine learning that lacks the most dedicated tools. Working with time series is notoriously known to be difficult and often requires very specific algorithms (statistical modeling, filtering, optimal control).

Even though some of the modern machine learning methods such as RNN, LSTM, or reinforcement learning can do an excellent job on some specific time-series problems, most of the problems require using more traditional algorithms such as linear and non-linear filters, FFT, the eigendecomposition of matrices (*e.g.* Bouchaud et al. (2005)), principal component analysis (PCA) (*e.g.* Dong and Qin (2018)), Riccati solvers for optimal control and filtering,...

By adopting a functional approach, inherited from JAX, WAX-ML aims to be an efficient tool to combine modern machine learning approaches with more traditional ones.

Some work has been done in this direction in (Lee-Thorp et al. (2021)) where transformer encoder architectures are massively accelerated, with limited accuracy costs, by replacing the self-attention sublayers with a standard, non-parameterized Fast Fourier Transform (FFT).

WAX-ML may also be useful for developing research ideas in areas such as online machine learning (see haz (2021); Hazan (2019)) and development of control, reinforcement learning, and online optimization methods.

3. What does WAX-ML do?

Well, building WAX-ML, we had some pretty ambitious design and implementation goals.

To do things right, we decided to start small and in an open-source design from the beginning. We released it with an Apache open-source license (see Sérié (2021)).

In this section, we give a quick overview of the features we have developed in WAX-ML. They are described in more detail in section 3.7.

For now, WAX-ML contains transformation tools that we call "unroll" transformations allowing us to apply any transformation, possibly stateful, on sequential data. It generalizes the RNN architecture to any stateful transformation allowing the implementation of any kind of "filter".

We have implemented a "stream" module, described in section 3.3, permitting us to synchronize data streams with different time resolutions.

We have implemented some general pandas and xarray "accessors" permitting the application of any JAX-functions on pandas and xarray data containers.

We have implemented a ready-to-use exponential moving average filter that we exposed with two APIs. One for JAX users: as Haiku modules (see for instance our `EWMA` module). A second one for pandas and xarray users: with drop-in replacement of pandas `ewm` accessor.

We have implemented a simple module `OnlineSupervisedLearner` to implement online learning algorithms for supervised machine learning problems.

We have implemented building blocks for designing feedback loops in reinforcement learning, and have provided a module called `GymFeedback` allowing the implementation of feedback loop as the introduced in the library Gym (see Brockman et al. (2016)), and illustrated in Figure 4.

Finally, we have implemented some "universal" modules that can work with TensorFlow, PyTorch, JAX, and NumPy tensors. At the moment, we have only implemented a demonstration module for the exponential moving average that we have called `EagerEWMA`.

3.1 Why use WAX-ML?

If you deal with time-series and are a pandas or xarray user, but you want to use the impressive tools of the JAX ecosystem, then WAX-ML might be the right tool for you, as it implements pandas and xarray accessors to apply JAX functions.

If you are a user of JAX, you may be interested in adding WAX-ML to your toolbox to address time-series problems.

3.2 Functional programming

In WAX-ML, we pursue a functional programming approach inherited from JAX.

In this sense, WAX-ML is not a framework, as most object-oriented libraries offer. Instead, we implement "functions" that must be pure to exploit the JAX ecosystem.

We use the "module" mechanism proposed by the Haiku library to easily generate pure function pairs, called `init` and `apply` in Haiku, to implement programs that require the management of parameters and/or state variables. In this way, we can recover all the advantages of object-oriented programming but exposed in the functional programming approach.

This approach gives a lot of freedom in the type of ideas that can be implemented. For instance, JAX has been used recently to accelerate fluid dynamics

simulations (see Kochkov et al. (2021)) by two orders of magnitude.

WAX-ML does not want to reinvent the wheel by reimplementing every algorithm. We want existing machine learning libraries to work well together while trying to leverage their strength, which is easy to do with a functional programming approach.

To demonstrate this, in the current version of WAX-ML, we have constructed various examples, such as an exponential moving average (to serve as a toy example), the implementation and calibration of an LSTM architecture with a standard supervised machine learning workflow, and the implementation of online learning and reinforcement learning architectures. They are treated equally and laid out with flat organization in our sub-package `wax.modules`.

3.3 Synchronize streams

Physicists have brought a solution to the synchronization problem called the Poincaré–Einstein synchronization (see synchronisation). In WAX-ML we have implemented a similar mechanism by defining a "local time", borrowing Henri Poincaré terminology, to denominate the timestamps of the stream (the "local stream") in which the user wants to apply transformations and unravel all other streams. The other streams, which we have called "secondary streams", are pushed back in the local stream using embedding maps which specify how to convert timestamps from a secondary stream into timestamps in the local stream.

This synchronization mechanism permits to work with secondary streams having timestamps at frequencies that can be lower or higher than the local stream. The data from these secondary streams are represented in the "local stream" either with the use of a forward filling mechanism for lower frequencies or with a buffering mechanism for higher frequencies.

Note that this simple synchronization scheme assumes that the different event streams have fixed latencies.

We have implemented a "data tracing" mechanism to optimize access to out-of-sync streams. This mechanism works on in-memory data. We perform the first pass on the data, without actually accessing it, and determine the indices necessary to later access the data. Doing so we are vigilant to not let any "future" information pass through and thus guaranty a data processing that respects causality.

The buffering mechanism used in the case of higher frequencies works with a fixed buffer size (see the WAX-ML module `wax.modules.Buffer`) to allow the use of JAX / XLA optimizations and efficient processing.

We give simple usage examples in our documentation and in working example shown in Figure 2.

3.4 Working with streaming data

WAX-ML may complement JAX ecosystem by adding support for streaming data.

To do this, we implement a unique data tracing mechanism that prepares for fast access to in-memory data and allows the execution of JAX tractable functions such as `jit`, `grad`, `vmap` or `pmap` (see section 3.3).

This mechanism is somewhat special in that it works with time-series data.

The `wax.stream.Stream` object implements this idea. It uses Python generators to synchronize multiple streaming data streams with potentially different temporal resolutions. It works on in-memory data stored in `xarray.Dataset`.

To work with real streaming data, it should be possible to implement a buffer mechanism running on any Python generator and to use the synchronization and data tracing mechanisms implemented in WAX-ML to apply JAX transformations on batches of data stored in memory. We have not yet implemented such a mechanism but put it on our enhancement proposal list.

3.5 Adding support for time types in JAX

At the moment `datetime64` and `string_` types are not supported in JAX.

WAX-ML add support for these NumPy types in JAX by implementing an encoding scheme for `datetime64` relying on pairs of 32-bit integers similar to `PRNGKey` implemented in JAX for pseudo-random number generation. We also implement an encoding scheme for `string_` relying on `LabelEncoder` from the Scikit-learn library.

By providing these two encoding schemes, WAX-ML makes it easy to use JAX algorithms on data of these types.

Currently, the types of time offsets supported by WAX-ML are quite limited and we would like to collaborate with the pandas, xarray, and Astropy (see Astropy Collaboration et al. (2013)) teams to develop further the time manipulation tools in WAX-ML.

3.6 pandas and xarray accessors

WAX-ML implements pandas and xarray accessors to ease the usage of machine-learning algorithms implemented with JAX-functions within high-level data APIs such as `DataFrame` and `Series` in pandas and `Dataset` and `DataArray` in xarray.

To load the accessors, users should have to simply run `register_wax_accessors()` function and then use the "one-liner" syntax `<data-container>.stream(...).apply(...)`

3.7 Implemented modules

We have some modules (inherited from Haiku modules) ready to be used in `wax.modules`. They can be considered as "building blocks" that can be reused to build more advanced programs to run on streaming data. We have some "fundamental" modules that are specific to time series management, the 'Buffer' module which implements the buffering mechanism, the 'UpdateOnEvent' module which allows one to "freeze" the computations of a program and to update them on some events in the "local stream".

To illustrate the use of this module, we show in Figure 1 how it can be used to compute the trailing open, high, and close quantities of temperatures recorded during the day.
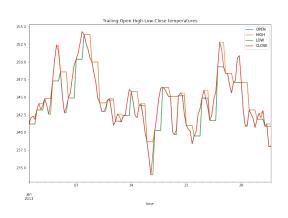

Figure 1: Computation of trailing Open-High-Low-Close temperatures.

We have a few more specific modules that aim to reproduce some of the logic that pandas users may be familiar with, such as `Lag` to implement a delay on the input data, `Diff` to compute differences between values over time, `PctChange` to compute the relative difference between values over time, `RollingMean` to compute the moving average over time., `EWMA`, `EWMVar`, `EWMCov`, to compute the exponential moving average, variance, and covariance of the input data.

Finally, we implement domain-specific modules for online learning and reinforcement learning such as `OnlineSupervisedLearner` (see section 4.2) and `GymFeedback` (see section 3.9).

For now, WAX-ML offers direct access to some modules through specific accessors for pandas and xarray users. For instance, we have an implementation of the "exponential moving average" directly accessible through the accessor `<data-container>.ewm(...).mean()` which provides a drop-in replacement for the exponential moving average of pandas.

We have implemented some working examples in our documentation. In Figure 2, we show an illustrative example where we apply exponential moving average filters with different time scales on a temperature data set (borrowed from the xarray library) while synchronizing two data streams ("air" and "ground") operating at different frequencies.
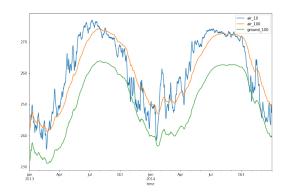

Figure 2: We apply exponential moving average filters with different time scales while synchronizing two data streams ("air" and "ground") operating at different frequencies.

3.8 Speed

The use of JAX allows for leveraging hardware accelerators that optimize programs for the CPU, GPU, and TPU architectures.

We have performed few benchmarks with WAX-ML on the computation of exponential moving averages on rather large dataframes (with 1 million rows and 1000 columns). For this size of dataframe, we observed a speedup of 3 compared to native pandas implementation. This is not a very important factor but it shows that even on such a simple algorithm already implemented in C, the XLA compilation can optimize things. The details of this benchmark are available in our documentation.

3.9 Feedback loops

Feedback is a fundamental notion in time-series analysis and has a wide history (*e.g.* Feedback). So, we believe it is important to be able to implement them well in WAX-ML.

A fundamental piece in the implementation of feedback loops is the delay operator. We implement it with the delay module `Lag` which is itself implemented with the `Buffer` module, a module implementing the buffering mechanism.

The linear state-space models used to model linear time-invariant systems in signal theory are a well-known place where feedbacks are used to implement for instance infinite impulse response filters. This is easily implemented with the WAX-ML tools and will be implemented at a later time.

Another example is control theory or reinforcement learning. In reinforcement learning setup, an agent and an environment interact with a feedback loop (see Figure 4). This generally results in a non-trivial global dynamic. In WAX-ML, we propose a simple module called `GymFeedBack` that allows the implementation of reinforcement learning experiments. This is built from an agent and an environment, both possibly having parameters and state (see Figure 3). The agent is in charge of generating an action from observations. The environment is in charge of calculating a reward associated with the agent's action and preparing the next observation from some "raw observations" and the agent's action, which it gives back to the agent.

A feedback instance `GymFeedback(agent, env)` is a function that processes the "raw observations" and returns a reward as represented in the Figure 4. Equivalently, we can describe the function `GymFeedback(agent, env)`, after transformation by Haiku transformation, by a pair of pure functions `init` and `apply` that we describe in Figure 5.

We have made concrete use of this feedback mechanism in the online learning application of section 4.2.

3.10 Compatibility with other reinforcement learning frameworks

In addition, to ensuring compatibility with other tools in the Gym ecosystem (see Brockman et al. (2016)), we propose a transformation mechanism to transform functions into standard stateful Python objects following the Gym API for agents and environments as implemented in the `deluca` framework (see LLC (2021)).
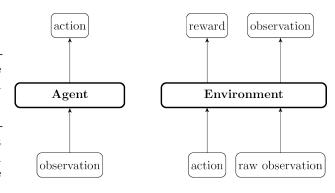

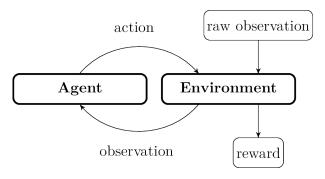Figure 3: Agent and Environment are simple functions (possibly with parameters and state).

Figure 4: Description of a Gym feedback loop.

4. Applications

4.1 Reconstructing the light curve of stars with LSTM

To demonstrate more advanced usages of WAX-ML with machine learning tools on time series we have reproduced the study (see Pere (2020)) on the reconstruction of the light curve of stars. We used the LSTM architecture to predict the observed light flux through time and use the Haiku implementation. We have then reproduced Haiku's case study for LSTM on this concrete application on the light curve of stars. We show an illustrative plot of the study in Figure 6.

Here WAX-ML is used to transition efficiently from time-series data with Nan values stored in a dataframe to a "standard" deep learning workflow in Haiku.

The study is available in our documentation through a reproducible notebook.

As a disclaimer, we emphasize that although our study uses real data, the results presented in it should not be taken as scientific knowledge, since neither the results nor the data source has been peer-reviewed by astrophysists. The purpose of the study was only
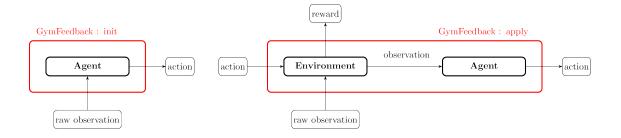
Figure 5: Description of a Gym feedback loop in term of pure functions `init` and `apply`.

to demonstrate how WAX-ML can be used when applying a "standard" machine-learning workflow, here LSTM, to analyze time series. However, we welcome further collaboration to continue the study.
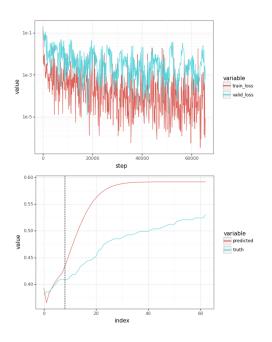

Figure 6: light curve of stars reconstruction with LSTM. First: training curve. Second: autoregressively predicted values and truth values.

4.2 Online linear regression in non-stationary environment

We performed a case study by implementing an online learning non-stationary linear regression problem following online learning techniques exposed in (Hazan (2019)).

We go there progressively by showing how a linear regression problem can be cast into an online learning problem thanks to the `OnlineSupervisedLearner` module developed in WAX-ML.

Then, to tackle a non-stationary linear regression problem (i.e. with a weight that can vary in time) we reformulate the problem into a reinforcement learning problem that we implement with the `GymFeedBack` module of WAX-ML.

We then define an "agent" and an "environment" using simple functions implemented with modules as illustrated in Figure 3. We have the agent which is responsible for learning the weights of its internal linear model and the environment which is responsible for generating labels and evaluating the agent's reward metric. We finally assemble them in a gym feedback loop as in Figure 4 and Figure 5.

We experiment with a non-stationary environment that flips the sign of the linear regression parameters at a given time step, known only to the environment. We show an illustrative plot of the final result of the study in Figure 7. More details about the study can be found in our documentation[1].

This example shows that it is quite simple to implement the online learning task with WAX-ML tools. In particular, the functional programming approach allowed us to compose and to reuse functions to design the final task.

5. Future plans

5.1 Feedback loops and control theory

We would like to implement other types of feedback loops in WAX-ML. For instance, those of the standard control theory toolboxes, such as those implemented in the SLICOT (see Benner et al. (1999)) library.

Many algorithms in this space are absent from the Python ecosystem and we aim to provide JAX-based implementations and expose them with a simple API.

An idiomatic example in this field is the Kalman filter, a now-standard algorithm that dates back to
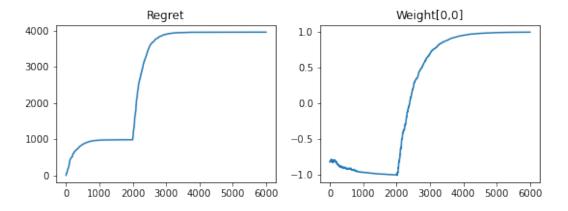
---

1. https://wax-ml.readthedocs.io/en/latest/

Figure 7: Online linear regression in a non-stationary environment. Left: The regret (cumulative sum of losses) first becomes concave, which means that the agent "learns something". Then, the regret curve has a bump at step 2000 where it becomes locally linear. It finally ends in a concave regime concave regime, which means that the agent has adapted to the new regime. Right: We see that the weights converge to the correct values in both regimes.

the 1950s (see Kalman (1960)). After 30 years of existence, the Python ecosystem has still not integrated this algorithm into widely adopted libraries. Some implementations can be found in Python libraries such as `python-control` (see pyt), statsmodels (see Seabold and Perktold (2010)), or SciPy (see Virtanen et al. (2020)). Also, some machine learning libraries such as Scikit-learn (see Buitinck et al. (2013)) and River (see Montiel et al. (2020)) have some closed and non-solved issues on this subject. Why has the Kalman filter not found its place in these libraries? We think it may be because they have an object-oriented API, which makes them very well suited to the specific problems of modern machine learning but, on the other hand, prevents them from accommodating additional features such as Kalman filtering. We think the functional approach of WAX-ML, inherited from JAX, could well help to integrate a Kalman filter implementation in a machine learning ecosystem.

It turns out that Python code written with JAX is not very far from Fortran (see Fortran), a mathematical FORmula TRANslating system. It should therefore be quite easy and natural to reimplement standard algorithms implemented in Fortran, such as those in the SLICOT library with JAX. It seems that some questions about the integration of Fortran into JAX have already been raised. As noted in an issue[2] reported on JAX's GitHub page, and it might even be possible to simply wrap Fortran code in JAX, which would avoid a painful rewriting process.

---
2. issue 3950

Along with the implementation of good old algorithms, we would like to implement more recent ones from the online learning literature which somehow revisit the filtering and control problems. In particular, we would like to implement the online learning version of the ARMA model developed in Anava et al. (2013) and some online-learning versions of control theory algorithms, an approach called "the non-stochastic control problem", such as the linear quadratic regulator developed in Hazan et al. (2020).

5.2 Optimization

The JAX ecosystem already has a library dedicated to optimization: Optax, which we use in WAX-ML. We could complete it by offering other first-order algorithms such as the Alternating Direction Multiplier Method (ADMM) (*e.g.* Boyd et al. (2011)). One can find functional implementations of proximal algorithms in libraries such as `proxmin` (see Melchior (2021)), `ProximalOperators` (see Stella et al. (2021)), or COSMO (see Garstka et al. (2019)), which could give good reference implementations to start the work.

Another type of work took place around automatic differentiation and optimization. In (Agrawal et al. (2019)) the authors implement differentiable layers based on convex optimization in the library `cvxpylayers`. They have implemented a JAX API but, at the moment, they cannot use the `jit` compi-

lation of JAX yet. We would be interested in helping to solve this issue[3].

Furthermore, in the recent paper by Blondel et al. (2021), the authors propose a new efficient and modular implicit differentiation technique with a JAX-based implementation that should lead to a new open-source optimization library in the JAX ecosystem.

5.3 Other algorithms

The machine learning libraries SciPy (see Virtanen et al. (2020)), Scikit-learn (see Buitinck et al. (2013)), River (see Montiel et al. (2020)), and ml-numpy (see Bourgin (2021)) implement many "traditional" machine learning algorithms that should provide an excellent basis for linking or reimplementing in JAX. WAX-ML could help to build a repository for JAX versions of these algorithms.

6. Conclusion

We think that the simple tools we propose in WAX-ML might be useful for performing machine-learning on streaming data and implementing feedback loops.

Some work could be continued to further develop the JAX ecosystem and implement some of the algorithms missing in the Python ecosystem in the wide adopted libraries.

We hope that this software will be well received by the open-source community and that its development will also be actively pursued.

7. Collaborations

The WAX-ML team is open to discussion and collaboration with contributors from any field who are interested in using WAX-ML for their problems on streaming data. We are looking for use cases around data streaming in audio processing, natural language processing, astrophysics, biology, finance, engineering ...

We believe that good software design, especially in the scientific domain, requires practical use cases and that the more diversified these use cases are, the more the developed functionalities will be guaranteed to be well implemented.

By making this software public, we hope to find enthusiasts who aim to develop WAX-ML further!

---

3. issue 103 JAX's GitHub page

References

Python Control Systems Library. URL `https://github.com/python-control/python-control`.

XLA: Accelerated Linear Algebra. URL `https://www.tensorflow.org/xla`.

MLPerf, 2020. URL `https://blog.tensorflow.org/2020/07/tensorflow-2-mlperf-submissions.html`.

Google Princeton AI and Hazan Lab @ Princeton University, 2021. URL `https://www.minregret.com/research/`.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable Convex Optimization Layers, 2019.

Oren Anava, Elad Hazan, Shie Mannor, and Ohad Shamir. Online Learning for Time Series Prediction. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *Proceedings of the 26th Annual Conference on Learning Theory*, volume 30 of *Proceedings of Machine Learning Research*, pages 172–184, Princeton, NJ, USA, 12–14 Jun 2013. Pmlr. URL `http://proceedings.mlr.press/v30/Anava13.html`.

Astropy Collaboration, T. P. Robitaille, E. J. Tollerud, P. Greenfield, M. Droettboom, E. Bray, T. Aldcroft, M. Davis, A. Ginsburg, A. M. Price-Whelan, W. E. Kerzendorf, A. Conley, N. Crighton, K. Barbary, D. Muna, H. Ferguson, F. Grollier, M. M. Parikh, P. H. Nair, H. M. Unther, C. Deil, J. Woillez, S. Conseil, R. Kramer, J. E. H. Turner, L. Singer, R. Fox, B. A. Weaver, V. Zabalza, Z. I. Edwards, K. Azalee Bostroem, D. J. Burke, A. R. Casey, S. M. Crawford, N. Dencheva, J. Ely, T. Jenness, K. Labrie, P. L. Lim, F. Pierfederici, A. Pontzen, A. Ptak, B. Refsdal, M. Servillat, and O. Streicher. Astropy: A community Python package for astronomy. *aap*, 558:A33, October 2013. doi: 10.1051/0004-6361/201322068.

Peter Benner, Volker Mehrmann, Vasile Sima, Sabine Van Huffel, and Andras Varga. *SLICOT—A Subroutine Library in Systems and Control Theory*, pages 499–539. Birkhäuser Boston, Boston, MA, 1999. ISBN 978-1-4612-0571-5. doi: 10.1007/978-1-4612-0571-5_10. URL `https://doi.org/10.1007/978-1-4612-0571-5%5F10`.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017. URL `https://doi.org/10.1137/141000671`.

Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and Modular Implicit Differentiation, 2021.

Jean-Philippe Bouchaud, Laurent Laloux, M. Augusta Miceli, and Marc Potters. Large dimension forecasting models and random singular value spectra, 2005.

David Bourgin. ml-numpy, 2021. URL `https://github.com/ddbourgin/numpy-ml`.

Stephen P. Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

URL `http://dblp.uni-trier.de/db/journals/ftml/ftml3.html#BoydPCPE11`.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

David Budden, Matteo Hessel, John Quan, Steven Kapturowski, Kate Baumli, Surya Bhupatiraju, Aurelia Guy, and Michael King. RLax: Reinforcement Learning in JAX, 2020. URL `http://github.com/deepmind/rlax`.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the Scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

Yining Dong and S. Joe Qin. A novel dynamic PCA algorithm for dynamic data modeling and process monitoring. *Journal of Process Control*, 67:1–11, 2018. ISSN 0959-1524. doi: https://doi.org/10.1016/j.jprocont.2017.05.002. URL `https://www.sciencedirect.com/science/article/pii/S095915241730094X`. Big Data: Data Science for Process Control and Operations.

Feedback. Feedback — Wikipedia, The Free Encyclopedia. URL `https://en.wikipedia.org/wiki/Feedback`. [Online; accessed 29-May-2021].

Fortran. Fortran — Wikipedia, The Free Encyclopedia. URL `https://fr.wikipedia.org/wiki/Fortran`. [Online; accessed 29-May-2021].

Michael Garstka, Mark Cannon, and Paul Goulart. COSMO: A conic operator splitting method for large convex problems. In *European Control Conference*, 2019. doi: 10.23919/ecc.2019.8796161. URL `https://arxiv.org/abs/1901.10887`.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David

Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

Elad Hazan. Introduction to Online Convex Optimization, 2019.

Elad Hazan, Sham M. Kakade, and Karan Singh. The Nonstochastic Control Problem, 2020.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL `http://github.com/google/flax`.

Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`.

Matteo Hessel, David Budden, Fabio Viola, Mihaela Rosca, Eren Sezener, and Tom Hennigan. Optax: composable gradient transformation and optimisation, in JAX!, 2020. URL `http://github.com/deepmind/optax`.

S. Hoyer and J. Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. doi: 10.5334/jors.148. URL `http://doi.org/10.5334/jors.148`.

Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35, 1960. doi: 10.1115/1.3662552. URL `http://dx.doi.org/10.1115/1.3662552`.

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning accelerated computational fluid dynamics, 2021.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. FNet: Mixing Tokens with Fourier Transforms, 2021.

Google LLC. Performant, differentiable reinforcement learning, 2021. URL `https://github.com/google/deluca`.

Peter Melchior. Proximal Minimization, 2021. URL `https://github.com/pmelchior/proxmin`.

Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. River: machine learning for streaming data in Python, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Christophe Pere. https://towardsdatascience.com/how-to-use-deep-learning-for-time-series-forecasting-3f8a399cf205, 2020.

Jonas Rauber, Matthias Bethge, and Wieland Brendel. EagerPy: Writing Code That Works Natively with PyTorch, TensorFlow, JAX, and NumPy, 2020. URL `https://eagerpy.jonasrauber.de`.

Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

Lorenzo Stella, Niccolò Antonello, and Mattias Fält. ProximalOperators.jl, 2021. URL `https://kul-forbes.github.io/ProximalOperators.jl/latest/`.

Einstein synchronisation. Einstein synchronisation — Wikipedia, The Free Encyclopedia. URL `https://en.wikipedia.org/wiki/Einstein%5Fsynchronisation`. [Online; accessed 29-May-2021].

Emmanuel Sérié. WAX-ML: A Python library for machine-learning and feedback loops on streaming data, 2021. URL `http://github.com/eserie/wax-ml`.

S. Taylor and Benjamin Letham. Forecasting at Scale. *PeerJ Prepr.*, 5:e3190, 2017.

Nx Team. Nx is a multi-dimensional tensors library for Elixir with multi-staged compilation to the CPU/GPU. URL https://github.com/elixir-nx/nx/tree/main/nx.

Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56–61, 2010. doi: 10.25080/Majora-92bf1922-00a.