

Open source tools for large-scale neuroscience

Jeremy Freeman

Please address correspondence to:

Jeremy Freeman

19700 Helix Drive
Ashburn, VA 20147
716-465-5303

freemanj11@janelia.hhmi.org
jeremyfreeman.net

Abstract

New technologies for monitoring and manipulating the nervous system promise exciting biology but pose challenges for analysis and computation. Solutions can be found in the form of modern approaches to distributed computing, machine learning, and interactive visualization. But embracing these new technologies will require a cultural shift: away from independent efforts and propriety methods, and towards an open source and collaborative neuroscience.

"I am absolutely convinced that in a few decades, historians of science will describe the period we are in right now as one of deep and significant transformations to the very structure of science. And in that process, the rise of free openly available tools plays a central role."

— Fernando Perez, creator of iPython ¹

Understanding the brain has always been a shared endeavor. But thus far, most efforts have remained individuated: labs pursuing independent research goals, slowly disseminating information via journal publications, and, when analyzing their data, repeatedly reinventing the wheel.

New experimental technologies are forcing a paradigm shift. Data sets are getting both larger and more complex. Many labs have more data than they have time to analyze, even for basic processing, let alone rich data exploration. The scale and complexity of the problems we want to tackle demands shared solutions.

Large-scale, high-resolution optical recordings of neural activity present a particularly exciting, and challenging, case study, and will be the focus of this essay. As encapsulated in an earlier review, the “operational principles of a neural circuit must be deduced through analysis of its structure and function”², and crucial to this effort is monitoring neural activity, at single-neuron resolution, in large populations, across multiple brain areas, or even the entire brain, during behavior.

Imaging methods — including two-photon laser scanning microscopy³, light-sheet imaging^{4,5}, and light-field imaging^{6,7} — monitor neural activity via protein sensors that convert changes in neural state, like changes in Ca²⁺ concentration, into changes in fluorescence (the case of two-photon imaging in head-fixed behaving mice is described in detail in another review in this same issue, Peron et al.).

The raw recorded data are time-varying images. Currently, a two-photon imaging experiment monitoring a region of mouse visual cortex can yield 512 x 512 x 4 pixels at 8 Hz, resulting in ~60 GB per hour, while a whole-brain light-sheet imaging experiment in a larval zebrafish can yield 1000 x 2000 x 40 pixels at 2 Hz, resulting in ~1.2 TBs per hour. These numbers describe one recording session from one animal, whereas most experiments involve many of each. Improvements in the spatial extent and temporal resolution of these technologies^{8,9} will only make these data sizes larger.

To understand the analytical challenges posed by imaging data, it is worth first understanding the typical data analysis steps. In its abstract form, this sequence shares much in common with data analytics in many industry settings (Figure 1).

Images must first be preprocessed, by registering across time to compensate for motion, the form of which may differ across experimental preparations and imaging modalities. Typically, this is followed by some kind of extraction of identified neuronal signals, for example: segmentation through morphological analysis of image structure^{10,11}, activity-based identification and demixing of correlated fluorescence patterns^{12,13}, or some combination of the two¹⁴. Appropriateness of methods will depend on the model system, resolution and sampling in both space and time, the indicator of neural activity, and the area imaged. For large data sets

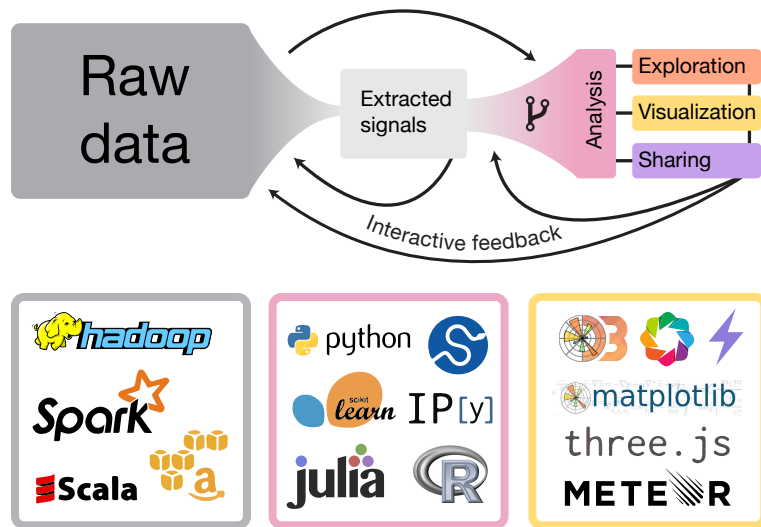


Figure 1. Most large-scale analytics, whether in industry or neuroscience, involve common patterns. Raw data are massive in size. Often, they are processed so as to extract signals of interest, which are then used for statistical analysis, exploration, and visualization. But raw data can be analyzed or visualized directly (top arrow). And the results of each successive step inform how to perform the earlier ones (feedback loops). Icons below highlight some of the technologies, discussed in this essay, core to the modern large-scale analysis workflow.

covering diverse morphological structures, voxel-wise analyses may provide a complementary alternative^{4,15}. In either case, temporal filtering is required to remove artifacts (e.g. trends), and deconvolution can be used to try and identify spikes¹⁶. Having identified neurons and their responses, we want to understand them. This process is more exploratory, and can include relating neural responses to properties of the stimulus or behavior of an animal¹⁷, identifying topological or low-dimensional structure in the data¹⁸, or inferring functional coupling¹⁹.

The first challenge is that there is currently little agreement as to how to solve these problems. Many existing approaches are ad-hoc, especially for basic data processing. Analyses are often more focused on suiting the needs of individual labs than the community, and algorithmic sophistication is valued above ease of implementation -- both unsurprising given the ordinary incentive structures in academia. Little is available in the way of vetting or benchmarking or standardization, partly due to the lack of curated, “ground-truth” data sets, in formats that are easily accessible from modern, distributed computing environments.

The second challenge is that all aspects of analysis must scale to potentially massive data sets, but single workstation solutions designed for smaller datasets remain the norm. To process raw data efficiently, we need to both load and operate in parallel across a dataset. Many operations are “embarrassingly parallel” -- we apply the exact same function to different portions of the data -- but require different partitioning depending on whether operations act locally in time, or space, or both. So it can become complicated to distribute an entire sequence of steps. Some algorithms are also more scalable than others: for example, parallelizing an image registration algorithm that applies an operation to the image at each time point might be trivial, but parallelizing an algorithm that examines pairs of time points at a time, and updates parameters after examining each pair, might be impossible. Even for algorithms that scale well, complete processing pipelines usually require multiple passes over the same data -- e.g. image filtering, registration, temporal filtering, factorization etc. Especially when data do not fit in the memory of a single machine, it becomes essential to minimize unnecessary reloading, and to efficiently combine sequences of operations.

After reducing a data set to, say, the time series of hundreds or thousands of neurons, scalability remains a challenge, but in a different form. The data can be loaded into the memory

of one machine, but fitting a complex model to every neuron becomes frustratingly slow, and fitting network models with coupling across neurons becomes intractable because they can create -- in the process of analysis -- objects that no longer fit in memory.

In approaching these challenges, we must balance the need for standardization and scalability with the importance of flexibility and interactivity. Different stages of analysis inform one another, with the results of one step suggesting a change to another (indicated by the feedback loops in Figure 1). It may prove beneficial to focus less on particular algorithms, and rather on the access patterns and forms of useful distribution common to *all* algorithms, yielding modular frameworks into which new algorithms can be incorporated and compared.

Solving these challenges will require new tools, but also a new culture. Most labs develop custom analysis strategies, using proprietary tools like Matlab poorly suited to collaborative development, inventing creative algorithms but only applying them to data from the lab in which they were developed, because they are hard to reproduce, require complex configuration, and barely run on single workstations.

Imagine, instead: fast open-source libraries for common analyses of imaging data, available to anyone and developed by all, with intuitive, modular code bases supporting customization, comparison, and benchmarking of pipelines and parameters, implemented in distributed systems that can run in cloud computing environments, with web-based interfaces for interactively exploring data and visualizing results.

An exciting new ecosystem of open-source computing technologies (Figure 1), described below, will make this vision a reality.

Distributed computing and storage

The first challenge of any large data set is, simply, its sheer size. Even if the goal is extracting a smaller representation, we must load and process the raw data, sometimes all at once. This is difficult when data do not fit the memory of a single machine, especially if we want to try analyses in an exploratory fashion, and, as discussed above, work with different distributed representations of the data.

There is a long history of solving such problems by parallelizing both storage and computation across large clusters of computers. Especially over the last decade, the trend in such distributed systems has been towards ever higher levels of abstraction, to let a user specify computations at the level of algorithms and queries, while hiding complexities like data distribution, load balancing, and parallelization strategies.

Early efforts in high-performance computing, including the message passing interface (MPI), emphasized inter-process communication and fast computation over the processing of massive data volumes, and were fairly complex to program. In the early 2000s, researchers at Google developed a set of abstractions for distributed storage and parallel computing of massive data sets on commodity hardware: the Google File System, and a programming model called MapReduce²⁰. The central idea of MapReduce is to divide a workflow into two steps: a map step, which applies a function to partitions of the data in parallel, and a reduce step, which aggregates across partitions using an associative operator. These abstractions were inspired by

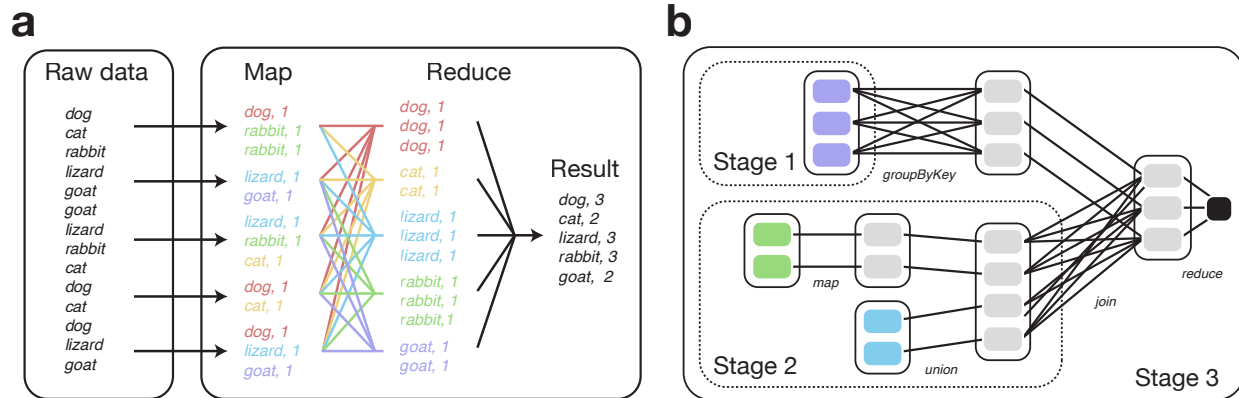


Figure 1. Distributed computing approaches. **(a)** The MapReduce programming model applied to counting words. Raw data, a potentially very large list of words, are processed into key-value pair records of the form (word, 1) in parallel during the “Map” step. During the “Reduce”, records with the same key (the word) are combined and an associative operator, addition, computes a sum for each word. **(b)** Diagram showing how Spark sequences more complex operations (adapted from Ref. X). Each large rounded rectangle is a resilient distributed dataset (RDD); colored or gray bubbles represent partitions. RDDs are related to one another by transformations. Simple ones, like map, apply an operation to each partition independently and in parallel; other transformations, like groupByKey, require moving (or shuffling) data among partitions. Colored bubbles represent data loaded directly from disk; gray bubbles are intermediate objects, which can be cached into memory. When a user requests the output from the final RDD — the black bubble on the far right — the graph of operations is compiled into three stages.

related concepts from functional programming, where many of the ideas underlying parallel computing first originated.

A now-classic example of MapReduce is counting words in a potentially massive corpus of text (Figure 2a). Here, as in most cases, the data are represented as (key, value) pairs. In the map step, the “mappers” work in parallel to convert words from the input data into (key, value) pairs, where the key is the word and the value is 1. In the reduce step, the “reducers” aggregate all the values labeled with each key using the addition operator, resulting in a list of counts for each word. This approach elegantly scales — with enough computers — to massive data sets, and a surprisingly wide variety of analyses and machine learning algorithms can be expressed using the MapReduce abstraction²¹. An open source version of MapReduce developed at Yahoo! around 2006, called Hadoop²², made the programming model accessible to a broad audience, and it was widely adopted in industry.

But the Hadoop MapReduce approach has limitations, many of which are particularly relevant to neuroscience. First, in typical Hadoop MapReduce workflows, data are loaded from disk during each analysis, so it can be slow to implement operations that require repeatedly querying the data, including the iterative computations common in machine learning algorithms and neural analyses (e.g. k-means clustering, independent component analysis, etc.). Second, it can be difficult to chain operations together in the kinds of complex workflows mentioned above; for example, registering images across time, computing a summary statistic on each voxel’s time series, performing an image-based morphometric analysis of neuronal shape, and then extracting time series within sub-regions from the original data. Such complex sequences can be both hard for a user to express, and inefficient to implement.

A new platform, called Spark, developed by researchers at Berkeley's AmpLab around 2009²³, addresses many of these limitations through a new abstraction, the resilient distributed dataset (RDD), and an associated execution engine. An RDD is a distributed collection of records— e.g. words, image patches, neuron traces — and can be manipulated through a family of high-level operators that process the data in parallel in a variety of ways. When sequences of operations are chained together by a user, the underlying implementation compiles the “graph” of desired operation into a small number of efficient tasks (Figure 2b). From the perspective of a user, this makes it easy to specify *where* you want to load data from, and *what* you want to do with the data -- as a sequence of RDD operations -- and Spark handles the complex task of executing that entire sequence, efficiently, across a cluster. Spark also allows data to be cached into the distributed RAM of a cluster, enabling faster repeated queries. This is especially important because data loading and reloading is often a key bottleneck in complex pipelines. Raw imaging data, for example, can be loaded and cached, and the data can then be exposed to a complex sequence of operations, or repeated, interactive analyses, without ever reloading from disk. Finally, Spark's APIs in Java, Scala, and Python allow complex operations to be expressed intuitively, with minimal amounts of code. The Python API is particularly appealing for neuroscience, for reasons discussed below.

All these technologies must load raw data at some point, and different options are available for storing that data. Conventionally, Hadoop MapReduce runs alongside the Hadoop Distributed File System (HDFS), which stores partitions of data on the hard drives of compute nodes. When running on Amazon's EC2, Hadoop or Spark can load data from Amazon's Simple Storage Service (S3), which is an appealing approach to storing massive amounts of compute-accessible data (for archival purposes, data can be moved from S3 to a longer-term low-accessibly storage layer called Glacier). Tools for working with Spark on Google's cloud computing environment, Google Compute Engine, and the associated Google Cloud Storage, are currently in development (https://github.com/broxonix/spark_gce). Spark can also load data directly from shared network drives, such as a Networked File System (NFS), making it appealing for academic high-performance computing environments with existing NFS (but not necessarily HDFS). The most common input formats for these tools, such as text files or Hadoop Sequence Files, are not well-suited to neural data, but we and others have been developing support for custom input formats, e.g. images and binary data; see, for example, these recent additions to the Spark and Thunder projects (<https://github.com/apache/spark/pull/1658> and <https://github.com/freeman-lab/thunder/releases/tag/v0.4.0>).

Deploying these distributed systems in a cloud-based environment — such as Amazon's EC2 or Google's Compute Engine — provides significant computing power without the requirement of an academic high-performance cluster or industrial data center. Cloud deployment also makes it easier to build tools that run identically for all users, especially with virtual machine platforms like Docker. However, cloud deployment for neuroscience does require transferring data to cloud storage, which may become a bottleneck. Deploying on academic clusters requires at least some support from cluster administrators, but keeps the data near the computation.

Despite the diversity of deployment and storage options, most of these tools are designed so that analyses and data formats can be used identically in many environments, facilitating standardization and sharing.

Languages for statistical computation

Once we can load and process data, we want to perform statistical computations, including everything from basic image or time series processing and signal extraction, to fitting complex network models in large population recordings.

Many open source tools exist for statistical computing, and many are superior to their proprietary alternatives. Python is an appealing choice. As a general purpose language, Python can interface with many external languages and components (e.g. C/C++ code, web servers). Python also has a rich ecosystem of libraries: NumPy (for numerical computing), SciPy (for statistical computing), scikit-learn (for machine learning), scikit-image (for image processing), and Pandas (for data manipulation)²⁴⁻²⁸. And iPython provides a powerful, interactive Python shell²⁹⁻³¹. Existing libraries in Python that can be used with neural imaging data including SIMA (motion registration and image segmentation for fluorescence imaging)³², nipy (analysis of fMRI data)³³, pycortex (surface rendering for fMRI data), and Thunder (distributed analytics for images and time series using Spark)¹⁵. Especially among statisticians, R is a popular alternative to Python, and offers a suite of libraries for complex statistical modeling, but lacks some of Python's generality and extendability. Julia is a particularly exciting new technical computing language, and outperforms R, Python, and Matlab for many local computations³⁴. It is early in development, and has yet to feature libraries for neural analysis, or even basic statistical modeling, but has promise for this use case.

These languages and libraries are still primarily used on single machines, but there are several strategies for use at scale. The Hadoop Streaming API supports writing Hadoop MapReduce jobs in Python, and Spark has extensive support for Python through its Python API, PySpark. Thunder, for example, is built on top of PySpark, making extensive use of Python's scientific computing libraries^{15,35}. Thunder offers a variety of data manipulation and analytic approaches directly relevant to spatial and temporal neural data. Simple, embarrassingly parallel operations can often be scaled up trivially, but more complex analyses, especially those examining joint relationships across a data set, require new implementations. A set of more general-purpose, highly-optimized machine learning algorithms have been implemented directly in Spark's MLlib machine learning library, including k-means clustering, collaborative filtering, and logistic regression, though MLlib has yet to support the same variety of operations available in, say, R or scikit-learn. As alternatives, iPython offers its own strategies for parallelization, including compatibility with MPI, and Julia offers user-controllable forms of parallelization.

Why is Matlab not on this list? Although Matlab is likely the most widely used platform among neuroscientists today³⁶, it is hard to recommend as a primary analysis tool in a future of open and collaborative science. User-developed Matlab code can be shared, but Matlab itself is closed-source and expensive. Although easy to use for beginners, and perhaps useful for educational purposes, Matlab has limited support for distributed computing, or even more basic modern features like functional and object-oriented programming or continuous automated testing, which make software easier to maintain, test, collaborate on, share, and integrate with other services, especially web-based ones³⁷⁻³⁹. More fundamentally, as we look towards a future where research and data are to be publicly shared, it seems inappropriate to allow a single, for-profit entity to effectively tax the reproducibility of results — much in the same way for-profit journals tax the distribution of knowledge. Finally, outside of niche applications, Matlab is much less widely used in industry data science; if we want to train students to succeed within and outside academia, we should teach them more than just Matlab.

Visualizing and interacting

For complex data sets, visualization is just as important as analysis. Graphical representations inform both our understanding of data and our development of analyses; a visualization can be the best guide for how an algorithm works ⁴⁰. Given the rich variety of data in neuroscience -- images, time series, networks, behavior -- there is a need for both custom visualizations suited to these data types, but also sufficient flexibility for exploration and interactivity. For effective communication and reproducibility, any scientific finding should be presented through visualizations that anyone can generate from raw data.

Several open-source tools exist for visualizing analysis results within the computing environments described above, including matplotlib and seaborn for Python, ggplot2 for R, and gadfly for Julia; some of these libraries purely support visualization, whereas others integrate visualization with statistical computations and data manipulation, in particular the ggplot2 library, which is based on a set of abstractions for generating and manipulating graphics from data known as “the grammar of graphics” ^{41,42}. Javascript-based platforms emphasize web visualization and interactivity, including D3.js (“data-driven documents”) and three.js or the X toolkit (for 3D visualizations with WebGL). These libraries are powerful, but it can be hard for new users to implement custom visualizations from scratch. Particularly useful would be ways to plot interactive visualizations from within analysis environments; at least three open-source projects in active development are filling this gap (mpld3, bokeh, and lightning), and could be combined with web app frameworks like Meteor.js to build powerful, interactive visualization environments.

Rethinking visualizations as interactive environments is especially exciting because it frees us from the constraint of communicating fixed and potentially biased representations of data. The typical neuroscience journal article shows, in sequence, a single example in rich detail (e.g. a single neuron or animal), a summary of a small population, and then a summary across the dataset. Imagine instead an interactive representation with all levels dynamically accessible. Such visualizations will require flexibly rendering large datasets and analyses at multiple scales, which is an exciting active area of development, and may benefit from techniques used in astronomy and geospatial analysis; spatial querying systems designed for cosmology, for example, are being adapted to query anatomical neural data ⁴³.

There is also rapidly growing interest in the “data analysis notebook”. These notebooks — the iPython notebook being a particularly popular example²⁹⁻³¹ — combine executable code blocks, notes, and graphics in an interactive document, that runs in a web browser, and provides a seamless front-end to a computer, or a large cluster of computers. Notebooks are a particularly appealing way to disseminate information; a recent neuroimaging paper⁴⁴, for example, provided all of its analyses in a github repo with iPython notebooks that generate all the figures in the paper ⁴⁵ — a clear model for the future of reproducible science.

Summary

Technological developments are rapidly pushing the boundaries of what we can measure and manipulate in the brains of awake-behaving animals. Our understanding of the brain will depend, ultimately, on our ability to design the right experiments, and ask the right questions. But the ability to rapidly manipulate, analyze, and visualize our data, in shared and collaborative

environments, with tools developed and vetted by and for the community, will be essential to that effort. The family of open-source tools described above suggests an exciting future for modern neuroscience.

This essay provided only a sketch; to make it a reality, several challenges remain, both technical and cultural:

Standardizing data formats

Collaborative development of analyses would benefit greatly if data sets were routinely shared and vetted by the community, and made available in standard formats. A priori standardization, however, is difficult because analyses and access patterns — which are constantly evolving — affect the way data should be represented and stored. And formats useful for local storage (e.g. HDF5) may not be compatible with distributed systems⁴⁶. Perhaps instead we should design data formats in lock step with the design of our analysis or visualization tools; there is precedence for this in many other communities.

Vetting and benchmarking

Too many existing analysis approaches are ad-hoc, developed for a particular kind of data, and not sufficiently generalizable or flexible. As we integrate more of our analysis approaches into distributed systems, and share more of our data, we can use this opportunity to systematically benchmark and vet our approaches. For example, to solve problems in imaging data analysis like source extraction / segmentation, and spike inference, we can hold data analysis challenges or competitions on standardized data sets with ground truth, and develop common frameworks that integrate multiple algorithms (see also review in this issue by Peron et al.)

Properly valuing contributions

To support the development of shared, open-source tools, the standards by which academic science values work will need to change. In particular, our community should consider significant contributions to code repositories — as documented and community-reviewed on Github — as on par with published, peer-reviewed journal articles.

Unifying local and distributed computing

In neuroscience applications, we often analyze data at different scales; for example, testing an analysis extensively on a subset of data, and then scaling the same analysis out to a massive data set. Due to overheads, distributed solutions rarely outperform, and may underperform, local implementations for small data sets. We need new abstractions that automatically perform computations as efficiently as possible in either regime.

Incorporating GPUs

Most of the tools discussed above emphasize CPU-based computation and parallelization; computing with GPUs is a powerful alternative, especially for some image processing and linear algebra routines^{7,10}, but may be less useful for loading and processing massive volumes of data due to demands on RAM and IO. How to fully combine the advantages of CPU and GPU-based approaches in parallel computing is an exciting area for future work.

Moving toward streaming analytics

As spatial scales and temporal resolutions increase, even with massive, industry scale storage, data may become too large to store permanently in any form. Especially with new technologies for manipulating neural activity, we will want analyses performed online, for guiding targeted, closed-loop manipulations of both neural activity and experimental paradigms. To perform analyses online, we will need scalable, streaming computing architectures, and programming models and algorithms suitable for data streams, something we are actively exploring with the distributed streaming platform Spark Streaming⁴⁷(<https://databricks.com/blog/2015/01/28/introducing-streaming-k-means-in-spark-1-2.html>). Working with streaming data will ultimately require not only new tools and algorithms, but entirely new ways of thinking about our experiments.

Acknowledgments

Much thanks to Jascha Swisher, Nicholas Sofroniew, and Karel Svoboda for helpful comments on earlier versions of this essay.

Reference highlights

Amat et al., 2014

* Describes a hybrid CPU and GPU-based algorithm for segmenting and tracking cell locations and lineages, derived from terabyte-scale high-resolution anatomical imaging data. Demonstrates the benefits of incorporating GPUs in image processing routines.

Mukamel et al., 2009

* An approach for automatically segmenting and demixing neural signals from imaging data using spatio-temporal independent component analysis. Provides a thorough account of not only the core computations involved (PCA and ICA) but the necessary preprocessing and post-processing steps.

Pnevmatikakis et al., 2014

* A broad review and implementation of matrix-factorization based approaches to extracting regions of interest from imaging data. Considers both spatial demixing and temporal deconvolution, and provides efficient, parallelizable algorithms.

Miri et al., 2011

* Comprehensive and detailed overview of how to use regression analyses to relate calcium imaging signals to behavioral events. Emphasizes a variety of practical statistical and signal processing issues, and considers both pixel-wise and ROI analyses.

Chu et al., 2007

* Shows how a variety of analyses and machine learning algorithms can all be expressed using the MapReduce programming model. Implementation details may not be relevant to more modern versions, but conceptually useful in understanding the basics of the approach.

Freeman et al., 2014

** Presents an open-source library for analyzing massive neural data sets using the Spark distributed computing platform. Along with describing the library itself — Thunder — this paper provides a useful overview of the challenges and considerations that arise when using a distributed computing engine to analyze scientific data.

Dean et al., 2004

** Foundational paper describing the MapReduce programming abstraction and its implementation at Google. Technical, but incredibly readable, this is an important paper for anyone beginning to work with or think about massively parallel analytics, even if not directly using the MapReduce engine model.

Zaharia et al., 2012

** Presents the architectural and design principles behind the Spark distributed computing platform, its Resilient Distributed Dataset abstraction, and its execution engine, and is full of intuitive examples and diagrams. Given Spark's rapidly growing adoption in both industry and science, this will surely come to be considered a foundational paper.

Buitinck et al., 2013

** Explains the thinking behind the API of scikit-learn, a popular, usable, and extendable machine learning library for Python. Provides an inside look into the design decisions underlying the library's development, and its elegant API, which emphasizes a set of key operations — fitting, estimating, and transforming — that have proven influential and have been adopted by other libraries.

Wickham et al., 2010

** Presents the concept behind a “grammar of graphics” and its application in a graphics library called ggplot2 for the R statistical computing language. A powerful and inspiring set of concepts about statistical graphics, elegantly explained by a modern pioneer of the field.

https://github.com/WagnerLabPapers/Waskom_JNeurosci_2014

** An impressive example of how to make science reproducible. This github repository contains all the code required to reproduce the results in the companion paper, as well as interactive iPython notebooks that make it easy for readers to generate the figures.

References

1. Perez, F. In Memoriam, John D. Hunter III: 1968-2012. *blog.fperez.org* at <<http://blog.fperez.org/2013/07/in-memoriam-john-d-hunter-iii-1968-2012.html>>
2. O'Connor, D. H., Huber, D. & Svoboda, K. Reverse engineering the mouse brain. *Nature* **461**, 923–929 (2009).
3. Grewe, B. F., Langer, D., Kasper, H., Kampa, B. M. & Helmchen, F. High-speed in vivo calcium imaging reveals neuronal network activity with near-millisecond precision. *Nat. Methods* **7**, 399–405 (2010).
4. Ahrens, M. B., Orger, M. B., Robson, D. N., Li, J. M. & Keller, P. J. Whole-brain functional imaging at cellular resolution using light-sheet microscopy. *Nat. Methods* **10**, 413–420 (2013).
5. Vladimirov, N. *et al.* Light-sheet functional imaging in fictively behaving zebrafish. *Nat. Methods* **11**, 883–884 (2014).
6. Prevedel, R. *et al.* Simultaneous whole-animal 3D imaging of neuronal activity using light-field microscopy. *Nat. Methods* (2014).
7. Broxton, M. *et al.* Wave optics theory and 3-D deconvolution for the light field microscope. *Opt Express* **21**, 25418–25439 (2013).
8. Lecoq, J. *et al.* Visualizing mammalian brain area interactions by dual-axis two-photon calcium imaging. *Nature Neuroscience* **17**, 1825–1829 (2014).
9. Stirman, J. N., Smith, I. T., Kudenov, M. W. & Smith, S. L. Wide field-of-view, twin-region two-photon imaging across extended cortical networks. *bioRxiv* (2014).
10. Amat, F. *et al.* Fast, accurate reconstruction of cell lineages from large-scale fluorescence microscopy data. *Nat. Methods* **11**, 951–958 (2014).
11. Pachitariu, M., Packer, A. M. & Pettit, N. Extracting regions of interest from biological images with convolutional sparse block coding. *Advances in neural information processing systems* (2013).
12. Mukamel, E. A., Nimmerjahn, A. & Schnitzer, M. J. Automated analysis of cellular signals from large-scale calcium imaging data. *Neuron* **63**, 747–760 (2009).
13. Pnevmatikakis, E. A. *et al.* A structured matrix factorization framework for large scale calcium imaging data analysis. *arXiv q-bio.NC*, (2014).
14. Podgorski, K., Dunfield, D. & Haas, K. Functional clustering drives encoding improvement in a developing brain network during awake visual learning. *PLoS Biol* **10**, e1001236 (2012).
15. Freeman, J. *et al.* Mapping brain activity at scale with cluster computing. *Nat. Methods* **11**, 941–950 (2014).
16. Vogelstein, J. T. *et al.* Fast nonnegative deconvolution for spike train inference from population calcium imaging. *Journal of Neurophysiology* **104**, 3691–3704 (2010).
17. Miri, A., Daie, K., Burdine, R. D., Aksay, E. & Tank, D. W. Regression-based identification of behavior-encoding neurons during large-scale optical imaging of neural activity at cellular resolution. *Journal of Neurophysiology* **105**, 964–980 (2011).
18. Cunningham, J. P. & Yu, B. M. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience* **17**, 1500–1509 (2014).
19. Pillow, J. W. *et al.* Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature* **454**, 995–999 (2008).
20. Dean, J. & Ghemawat, S. MapReduce: simplified data processing on large clusters. *Communications of the ACM* (2004).
21. Chu, C.-T. *et al.* Map-reduce for machine learning on multicore. *Advances in neural information processing systems* **19**, (2007).
22. Shvachko, K., Kuang, H., Radia, S. & Chansler, R. The Hadoop Distributed File System. in *IEEE 26th Symposium on Mass Storage Systems and Technologies* 1–10 (IEEE, 2010).

23. Zaharia, M., Chowdhury, M., Das, T., Dave, A. & Ma, J. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. in *NSDI* (2012).
24. van der Walt, S., Colbert, S. C. & Varoquaux, G. The NumPy array: a structure for efficient numerical computation. *arXiv cs.MS*, (2011).
25. Pedregosa, F. *et al.* Scikit-learn: Machine Learning in Python. *arXiv cs.LG*, (2012).
26. Buitinck, L. *et al.* API design for machine learning software: experiences from the scikit-learn project. *arXiv cs.LG*, (2013).
27. Abraham, A. *et al.* Machine learning for neuroimaging with scikit-learn. *Front Neuroinform* **8**, 14 (2014).
28. van der Walt, S. *et al.* scikit-image: Image processing in Python. *arXiv cs.MS*, e453 (2014).
29. Perez, F. & Granger, B. E. IPython: a System for Interactive Scientific Computing. *Computing in Science and Engineering* **9**, 21–29 (2007).
30. Rossant, C. *Learning IPython for Interactive Computing and Data Visualization*. (Packt Publishing Ltd, 2013).
31. Rossant, C. *IPython Interactive Computing and Visualization Cookbook*. (2014).
32. Kaifosh, P., Zaremba, J. D., Danielson, N. B. & Losonczy, A. SIMA: Python software for analysis of dynamic fluorescence imaging data. *Front Neuroinform* **8**, 80 (2014).
33. Gorgolewski, K. *et al.* Nipype: a flexible, lightweight and extensible neuroimaging data processing framework in python. *Front Neuroinform* **5**, 13 (2011).
34. Bezanson, J., Karpinski, S., Shah, V. B. & Edelman, A. Julia: A Fast Dynamic Language for Technical Computing. *arXiv cs.PL*, (2012).
35. Freeman, J. Spark as a platform for large-scale neuroscience. *databricks.com* at <<https://databricks.com/blog/2014/10/01/spark-as-a-platform-for-large-scale-neuroscience.html>>
36. Wallisch, P. *et al.* *MATLAB for Neuroscientists*. (Academic Press, 2010).
37. Tjoa, S. I used Matlab. Now I use Python. *stevetjoa.com* at <<https://stevetjoa.com/305/>>
38. Rossant, C. Why use Python for scientific computing? *cyrille.rossant.net* at <<http://cyrille.rossant.net/why-using-python-for-scientific-computing/>>
39. Feldman, P. Eight advantages of Python over Matlab. *phillipmfeldman.org* at <http://phillipmfeldman.org/Python/Advantages_of_Python_Over_Matlab.html>
40. Bostock, M. Visualizing Algorithms. *bost.ocks.org* at <http://bost.ocks.org/mike/algorithms/>
41. Wickham, H. A Layered Grammar of Graphics. *Journal of Computational and Graphical Statistics* **19**, 3–28 (2010).
42. Wilkinson, L., Wills, D., Rope, D., Norton, A. & Dubbs, R. *The Grammar of Graphics*. (Springer Science & Business Media, 2006).
43. Burns, R., Vogelstein, J. T. & Szalay, A. S. From cosmos to connectomes: the evolution of data-intensive science. *Neuron* **83**, 1249–1252 (2014).
44. Waskom, M. L., Kumaran, D., Gordon, A. M., Rissman, J. & Wagner, A. D. Frontoparietal representations of task context support the flexible control of goal-directed cognition. *The Journal of Neuroscience* **34**, 10743–10755 (2014).
45. Waskom, M. L. https://github.com/WagnerLabPapers/Waskom_JNeurosci_2014
46. Heber, G., Folk, M. & Koziol, Q. A. Big HDF FAQs. www.hdfgroup.org (2014). at http://www.hdfgroup.org/pubs/papers/Big_HDF_FAQs.pdf
47. Zaharia, M., Das, T., Li, H., Shenker, S. & Stoica, I. Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. in *Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing* (2012).