Spark-Advanced Analytics

Analyzing Neuroimaging Data with PySpark and Thunder

Naga Jyothi Kunaparaju

Saint Peter's University jkunaparaju@gmail.com

1-347-341-3307

ABSTRACT

This report is a study of the neuron activity of Zebra Fish to determine if changes in that activity relates to various functions of the brain. As size and complexity of neural data increases everyday due to advancement in technology, analyzing this data set improve chances of understanding the behavior of brain when it is still active. As data size is growing there must be different ways to analyze the different datasets. This could be done either trying to experiment on different parameters or building new models. As memory is another limitation for single workstations analyzing in a distributed environment is necessary. Hadoop MapReduce and its ecosystem like HDFS, HIVE, PIG are one solutions to such kind of problems. However data should be loaded from disk for each operation which is a weakness as it makes iterations considerably slower during computation. When we use Apache spark, which is a distributed cluster computing system

with a rich set of of libraries, along with python provides a cost effective solution for doing such analysis. It extends MapReduce functionality with a resilient distributed dataset(RDD). In this model user can cache wither entire dataset or work with intermediate results which makes computations faster. Along with Spark there is another open source library, which well suited for Image data especially for Neuroimaging, called Thunder. Thunder is used along with the python implementation of spark. Using python gives advantages such as using its rich set of advanced libraries which are well suited for data analysis and Visualization. We examined K-means algorithm to cluster the various fish time series into multiple clusters to describe Neural behavior. The positive results show us that learned cluster summarizes certain elements of Zebrafish brain. Analysis showed how these tools helped finding patterns of neural data.

1. INTRODUCTION

As technology emerging data growing very fast with increasing complexity making the need for wider variety of analysis. One hour of photon imaging of a larval zebrafish brain can yield 1TB. It's not possible to run and analyze results in single machine with this data. Data include recordings from all neurons consist of Time-series from ~10power 8 voxels(Ref1). MapReduce is a widely adopted programming model. It has a weakness, data must be loaded from disk for each operation,(Ref6) which can slow iterative computations and makes interactive, exploratory analysis difficult. Apache Spark platform extends and generalizes the MapReduce model while solving this weakness(Ref1), by introducing a data sharing technic called a resilient distributed data set (RDD). One of the key advantage of spark over other systems is ability to cache large data sets and efficiently manage data from disks when memory completely occupied which an essential for doing repeatedly query and do multiple analysis is critical for analyzing neural activity.

Apache Spark framework with extension Pyspark and Thunder project made possible implements many algorithms for processing large amounts of spatial / temporal data sets. The tools enable rapid analysis of large neural sets. Thunder includes several univariate and multivariate analyses, organized into regression,

factorization, clustering and time-series statistics(Ref3). Thunder and Spark could replace earlier analysis tools such as Matlab, which will cost money and time compared to Spark. Matlab has very less support for distributed computing and also even more basic modern features like functional and object-oriented programming(Ref 4). As Spark is open-source library which makes large-scale neural analytics available to a broad community. It depends on the scientific computing procedures in Python (through Numpy and Scipy), which are also freely available and which are familiar tools for data science.

1.1 PYTHON BASICS

Python is easy to learn and a highly productive language where we can do more data analytics tasks quickly. Scala is more complex in terms code understanding due to its high level functional features. When we compare code conciseness, both are very concise depending on how efficiently you can code. Reading Python is more easy, it shows you step-by-step what your code execution is and the state of each variable. Scala, compared to python, will focus more on final result hiding most of the implementation details. Whilst pattern matching is good way to extract variables, advanced features like implicit(custom UDIs) can be confusing to the non-expert user.

Summarizing advantages

- Python is easy to understand and learn.
- It has rich set of tools to do mathematical and statistical analysis like numpy, Scipy
- Pandas and nltk
- Integrates well with other languages like c, c++ and matlab.
- Python also has rich set of development tools like IPYTHON.

1.2 SPARK INTERNALS

Spark is cluster computing framework designed to handle large datasets. What makes Spark unique is the type of computing. It is designed to cover wide range of workloads in one place. It reduces burden of maintaining different tools (Like in Hadoop systems).

Spark can run on clusters and access Hadoop data. It provides API in python, java, Scala and SQL. Spark in memory computing makes applications run 20% faster, in comparison to other applications, for certain jobs.

RDD is a core data structure in Spark which provides in memory computation for many generic usages and also provides fault tolerance by making RDD coarse-grained. RDDs are read only created either from Storage or from other transformation. RDD's are not always materialized. Through the lineage graph they have enough information about how it was derived. Spark written on Scala comes to around 14000 lines of code which is very small code compared to Hadoop. Spark runs on Mesos cluster manager. Each spark program runs as a separate Mesos application, with its own driver and workers. Resource sharing between these applications is handled by Mesos.

1.3 Advantages of RDD

There is one more technique in industry similar to RDD is called Distributed Shared Memory. DSM handles general abstraction but allows every time reads and writes to memory location. This feature makes DSM not fault tolerant. RDD only allows bulk writes through map operations, makes it easier to achieve fault tolerance. RDD is immutable in nature allows system can have slow nodes by running backup copies. Backup copies are not easy to implements in DSM because two copies of a task would access the same memory location and interface allowing each other's updates override. RDD can degrade gracefully when there is not enough memory

1.4 Py-Spark Architecture

Pyspark can be started just like Spark. When Pyspark shell opened it creates spark context object which will communicate with cluster. After spark context has been created Pyspark is similar to spark. Pyspark supports IPYTHON browser which is an ideal development environment with fast program building. One major disadvantage with python over Scala is, objects which need to be serializable should go through Cloudpickle. Any external library for application should be included in PYTONPATH or distributed Zip/EGG file externally.

When Pyspark interpreter starts it starts JVM which will create java spark context. Python uses Project Py4j for communication. The data associated to the RDD actually lives as java objects in JVM. Any API call related to RDD from python is serialized with cloud pickle and send to cluster where python interpreter will interpret into java code and resulted RDD stored in JVM.

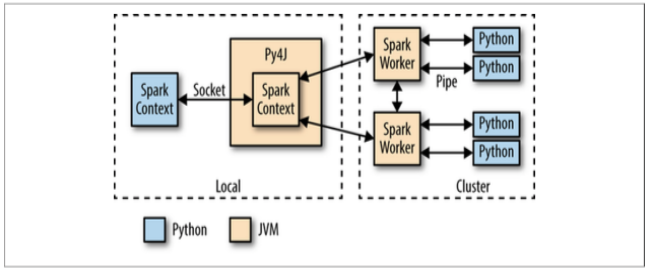


Figure 11-1. PySpark internal architecture

Like for example call made for loading text file from python interpreted and Java string objects loads into memory.

When PySpark's Python interpreter starts, it also starts a JVM with which it communicates through a socket. The Spark executors on the cluster start a Python interpreter for each core, with which they communicate data through a pipe when they need to execute user code. Running sc.textFile() in the Python interpreter will call the JavaSparkContexts textFile method, which loads the data as Java String objects.

Python's built-in support for serializing executable code is not as powerful as Scala's. PySpark had to use a custom module called "cloudpickle". Thunder use of NumPy for matrix computations and MLlib library for distributed implementations of some statistical techniques. Using Thunder API, we classify some neural traces into a set of patterns using MLlib's K-means implementation. After installation, and setting the SPARK_HOME environment variable, we can invoke the Thunder shell similar to spark shell opening.

1.5 THUNDER INSTALLATION

Thunder is a collection of tools for the analysis of image and time series data in Python. It provides data structures and algorithms for loading, processing, and analyzing these data. Thunder tries to use the fastest data structures like ndarray which can available with local as numpy and distributed as bolt. There are well

set of tools developed also integrated well with sophisticated tools like numpy scipy.

Thunder can be installed as: pip install thunder-python

When we start spark Thunder context available automatically. There are lot of changes from the initial version which was written in book to existing one. One of major architecture change is Spark code separated with Thunder. Earlier spark was integrated along with thunder. Now thunder image processing can be done standalone.

2 DATA

Analysis starts from loading, parsing data and saving results to disk. The data for this project is images of Zebrafish brain loaded from Thunder repository, at python/thunder/utils/data/fish/tif-stack. Zebrafish is great model for neuroscience, because it is transparent and the brain is small enough to image at a high resolution to identify individual neurons. Data for Thunder is a persisted set of images, with individual image filenames.

2.1 IMAGE DATA TERMS:

Pixel: Each element in a 2D image matrix corresponds to a pixel

Voxel: A Voxel is a volume element in 3D matrix

Stack: A stack of 2D images will generate a 3D information.

When we start analyzing brain 2d images are not sufficient to analyse whole brain. A stack of 2d images are taken and which will create like a 3d effect which provides most of the information from brain. Thunder can load file from .tiff which represents 3D stacks. Databricks allows to run spark cluster with ease and code without difficulty of installing external IDE. Thunder library intergrated with Databricks

There are two core data types in Thunder, Series and Images, both are subclasses of Data class. The Data class like RDDs of key-value pairs, where the key represents some type of variable and the value is a NumPy array of actual data. For the Images object, the key could be a time point and the value is the image at that time point formatted as a NumPy array. Both wrapped around in a rdd will expose RDD methods. Each one is a high-level wrapper for an ndarray, and can be used in either a local like numpy or a distributed form with spark. These data types are exposed to same kind of methods to load from a wide variety of local or remote data. Both have statistical operators, and can be written to disk in several different formats. These data types are classes in core thunder package.

2.2 IMAGES

Example binary and text, and also be constructed directly from a list or ndarray. Methods are available for filtering, preprocessing, and statistical aggregation conditional on the index.

Thunder Images data type is used for representation of collection of images or volumes. This collection suits well to movie data, when images come from different points in time. It can be used for generic image collections. Images can be loaded from different formats like png, tiff and binary. They also can be created from a list or ndarray. Many domain-specific methods are available for processing images, filtering and writing to external formats.

2.3 SERIES

The Series data type in Thunder is similar to time series data. It is used to represent a array of one dimensional records and share a common index. The index is time. It can also be used for normal series data. Series data also can be loaded from major disk formats.

2.4 BOTH

In addition, above methods, there are many other methods shared by Series and Images for aggregations functions like mean and std deviation and generic methods like map and reduce. These methods are from a data class which is a subclass of Base. Images to series and series to images are converted via an intermediate data class called blocks which is also a subclass of base. Map function can be applied to each record for both Data and Image

```
class Data:
    property dtype:
        # The dtype of the numpy array in this RDD's value slot
    # lots of RDD methods, like first(), count(), cache(), etc.
    # methods for aggregating across arrays, like mean(),
    # variance(), etc., that keep the dtype constant

class Series(Data):
    property dims:
        # lazily computes Dimension object with information about
        # the spatial dimensions encoded in the keys of this RDD
    property index:
        # a set of indices into each array, in the style of a
        # Pandas Series object
    # lots of methods to process all of the 1D arrays in parallel
    # across the cluster, like normalize(), detrend(), select(),
    # and apply(), that keep the dtype constant
```

```
    # methods for parallel aggregations, like seriesMax(),
    # seriesStdev(), etc., that change the dtype
    def pack():
        # collects the data at the client and repacks from the
        # sparse representation in the RDD to a dense
        # representation as a NumPy array with shape corresponding
        # to dims

class Images(Data):
    property dims:
        # the Dimension object corresponding to the NumPy shape
        # parameter of each value array
    property nimages:
        # number of images in RDD; lazily executes an RDD count
        # operation
    # multiple methods for aggregating across images or processing
    # them in parallel, like maxProjection(), subsample(),
    # subtract(), and apply()
    def toSeries():
        # reorganize data as a Series object
```

Here is the code to load the data set:

```
path_to_images = ('path/to/thunder/python/thunder/utils/data/fish/tif-stack')
imagesRDD = tsc.loadImages(path_to_images, inputformat='tif-stack')
print imagesRDD
print imagesRDD.rdd
...<thunder.rdds.images.Images object at 0x109aa59d0>
```

PythonRDD[8] at RDD at PythonRDD.scala:43

The objects stored in Images are key-value pairs:

print imagesRDD.first()

(0, array([[[26, 25],

[26, 25],

[26, 25],

The imagesRDD like dictionary with key value pairs. In this print statement, it prints first record with key 0 corresponds to the zeroth image in the set, and the value is a NumPy array corresponding to the image. All of the core data types in Thunder are Python RDDs of key-value pairs. Even though PySpark generally allows RDDs of heterogeneous collections, the keys and values always have a homogeneous type across the RDD. The images Object exposes .dims property because of homogeneity which will return dimensions of the array.

print imagesRDD.first()[1].shape

(76, 87, 2)

print imagesRDD.dims

Dimensions: min= (0, 0, 0), max= (75, 86, 1), count= (76, 87, 2)

Our data set is composed of 20 "images" where each image is a 76 × 87 × 2 stack.

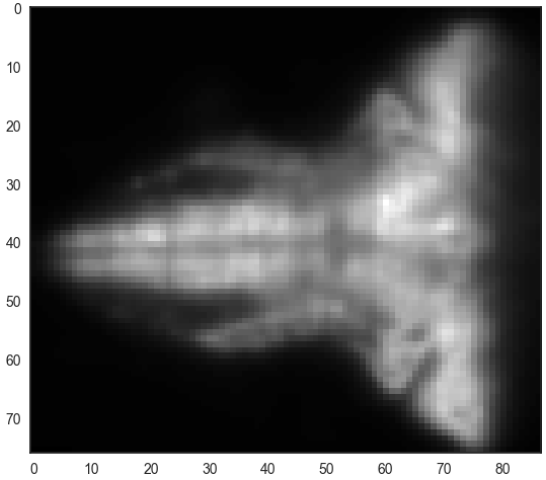
Data can be easily visualized using matplotlib library (see Figure 1):

Import matplotlib.pyplot as plt img = imagesRDD.values().first() plt.imshow(img[:, : ,0], interpolation='nearest', aspect='equal', cmap='gray'

We can subsample this image and also we can convert into time series to do more analysys. Which has a key corresponds to each image and value is array of time series values.

seriesRDD = imagesRDD.toSeries()

print seriesRDD.dims



A single slice from the raw zebrafish data

Dimensions: min=(0, 0, 0), max=(75, 86, 1), count=(76, 87, 2)

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

13224

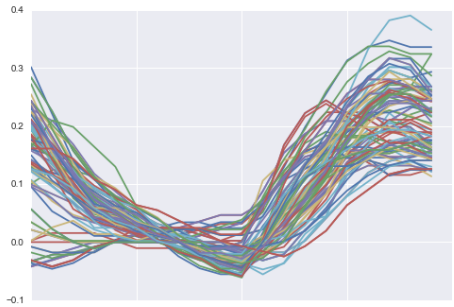
ImageRDD is collection of images of 20 and seriesRDD is a collection of 13,224 time-series objects of length 20. SeriesRDD is representation of data in to s time-series object with key is a tuple with coordinates of each image and value is a one dimensional NumPy array with time-series values.

With this data we can do more analysis.

3 METHOD:

Thunder supports univariate and multivariate analyses, categorized as factorization, regression, clustering and time-series. The components designed for analysis are scalable and extendable, making it easy to use. Univariate analyses can be done on single neural channel like, individual voxel summary statistics or regression. This can be done by using spark map function which will executes in all worker nodes of the clusters. This map function again fallowed by a 'collect' and it will get results to the driver. Thunder use a high-level Regression Model class which implements map and subclasses for models to fit. This kind abstraction

```
examples = series.filter(lambda x: x.std() > 6).normalize().sample(100).toarray()
plt.plot(series.index, examples.T);
```



allows more flexibility the with minimum knowledge able to implement most models with ease.

Doing analysis on multiple channels at the same time on entire data set is called Multivariate analysis. Combined map and reduce operations are necessary, where the map step distributes tasks across cluster by partitioning the data, and the reduce step combines the result by commutative and associative functions. As multivariate analyses use singular value decomposition (SVD), which seeks to approximate an n×t matrix as the product of n×k and k×t matrices, where n is the number of channels and t the number of points. There is online note book for thunder library available which will be very useful for executing code without burden of any installation process. You can test the code for Normal basic operations.

First we start with loading images :

from thunder import images

data = images.fromrandom()

data

>> Images

>> mode: local

>> dtype: float64

>> shape: (10, 10, 50)

Thunder documentation available in http://thunder-project.org/ website, and github in, https://github.com/thunder-project/thunder. Spark can be integrated with Ipython used for development application easily.

path_to_images = (

'path/to/thunder/python/thunder/utils/data/fish/tif-stack')

imagesRDD = tsc.loadImages(path_to_images,

inputformat='tif-stack')

print imagesRDD

print imagesRDD.rdd

...

<thunder.rdds.images.Images object at 0x109aa59d0>

PythonRDD[8] at RDD at PythonRDD.scala:43

This is older version of Python code for loading mages and creatingRDD.

Now

data = td.images.fromexample("fish" , sc)

plt.imshow(data[0][:,:,1])

imagesRDD = data.tordd()

imagesRDD

First Images are loaded as array. Then its converted in RDD.

print imagesRDD.first()[1].shape

print imagesRDD.count()

```
> print imagesRDD.first()[1].shape
  print imagesRDD.count()
```

▸ (2) Spark Jobs

```
(2, 76, 87)
20
```

20

There are 20 images. Each image is stack of 2 76X87 images.

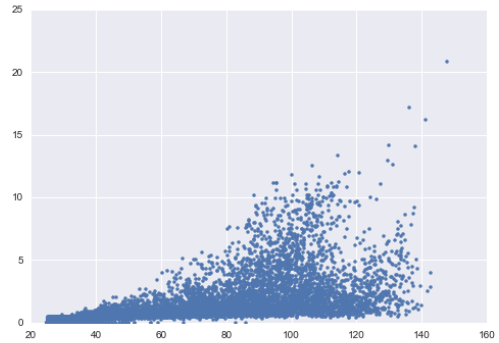
$ thunder

3.1 CREATING TIME-SERIES

The final goal of this analysis is measuring whole brain activity in a larval zebrafish while simultaneously presenting a moving visual stimulus and monitoring intended behavior. By converting image data into time series objects and cluster neurons by fallowing

```
14]: means = series.map(lambda x: x.mean()).flatten().toarray()
     stds = series.map(lambda x: x.std()).flatten().toarray()

15]: plt.plot(means, stds, '.');
```



To compute statistics within records, we can make use of the map method, which executes an arbitrary function on each record

classification techniques observing patterns is the task we have to perform using these pyspark and thunder.

Time-series RDD object we can do computations across per series level or across all series.

seriesRDD = imagesRDD.toSeries()

print seriesRDD.max()

…

array([158, 152, 145, 143, 142, 141, 140, 140, 139, 139, 140, 140, 142, 144, 153, 168, 179, 185, 185, 182], dtype=uint8)

computes the maximum value across all voxels at each time point, while:

stddevRDD = seriesRDD.seriesStdev()

print stddevRDD.dims

…

[((0, 0, 0), 0.4), ((1, 0, 0), 0.0), ((2, 0, 0), 0.0)]

Dimensions: min=(0, 0, 0), max=(75, 86, 1), count=(76, 87, 2)

computes the standard deviation of each time series and returns the result as an RDD, preserving all the keys. Any user-defined function to each series (including

lambda functions), can be applied using the apply method,

seriesRDD.apply(lambda x: x.argmin())

Categorizing Neuron Types with Thunder

Data used to do this series data which is already persisted. We examine K-means algorithm to cluster the various fish time series into multiple clusters to describe Neural behavior.

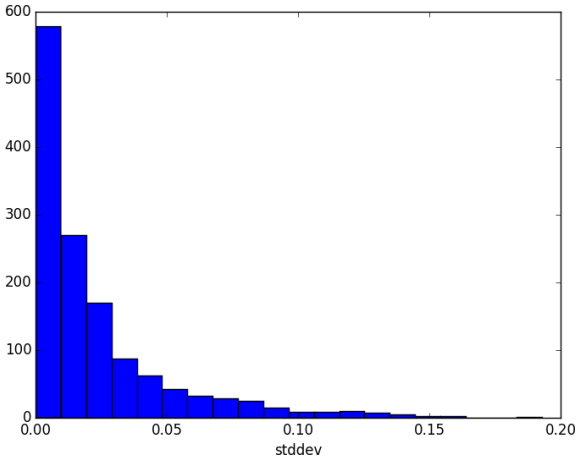
normalizedRDD = seriesRDD.normalize(baseline='mean')

Now our goal is to select most active timeseries to find out the behavior of Neurons.

NormallizeRDD can be subset based on certain criteria. Now plot histogram of standard deviations to find most active with a threshold Of 10%

stddevs = (normalizedRDD.seriesStdev().values().sample(False, 0.1, 0).collect())

plt.hist(stddevs, bins=20)



Distribution of the standard deviations of the voxels

Now plot most active series to cluster according to their behavior.

plt.plot(normalizedRDD.subset(50, thresh=0.1, stat='std').T)

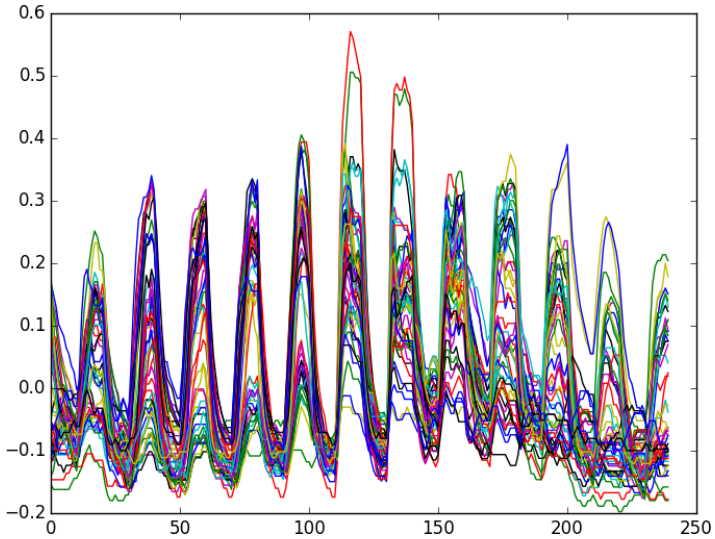


Figure 2. Fifty of the most active time series, based on standard deviation

Thunder's K-means API calls out to the MLlib Python API. K-means for multiple values of k: from thunder import KMeans

```
ks = [5, 10, 15, 20, 30, 50, 100, 200]
models = []
for  k in ks:
models.append(KMeans(k=k).fit(normalizedRDD))
```

From this we built two error models first one will manually simplify sum across all-time series of the Euclidean distance from time series to cluster center.

Second one use built in metric

```
def model_error_1(model):
  def series_error(series):
    cluster_id = model.predict(series)
    center = model.centers[cluster_id]
    diff = center - series
    return diff.dot(diff) ** 0.5
  return
(normalizedRDD .apply(series_error().sum())
  def model_error_2(model):
   return                    1.                    /
model.similarity(normalizedRDD).sum()
```

Results

We will compute both error metrics for each value of k and plot them (see Figure 11-8):

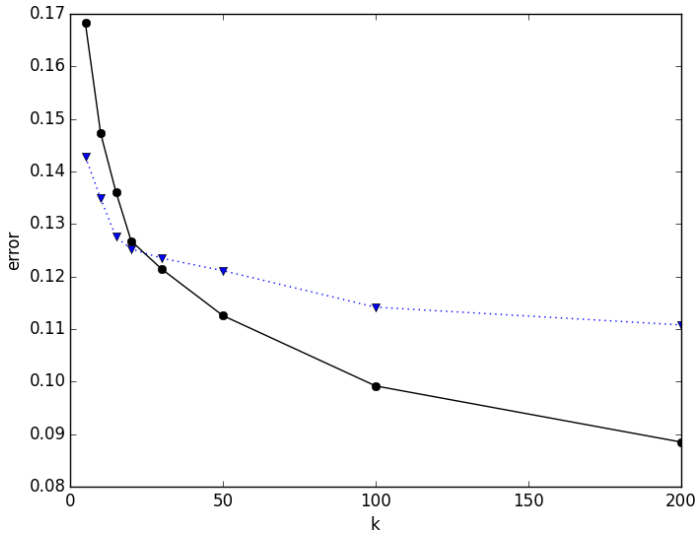


Figure 4. K-means error metrics as a function of k (black circles are model_error_1 and blue triangles are model_error_2)

We'd expect these metrics to generally be monotonic with k; it seems like k=20 might be a sharper elbow in the curve. Let's visualize the cluster centers that we've learned from the data (see Figure 4):

model20 = models[3]

plt.plot(model20.centers.T)

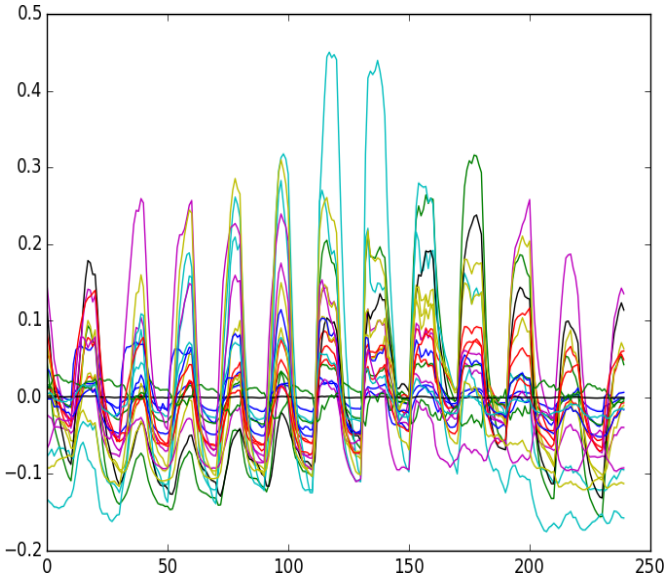


Figure 5. Model centers for k=20

It's also easy to plot the images themselves with the voxels colored according to their assigned cluster (see Figure 6):

from matplotlib.colors import ListedColormap

by_cluster = model20.predict(normalizedRDD).pack()

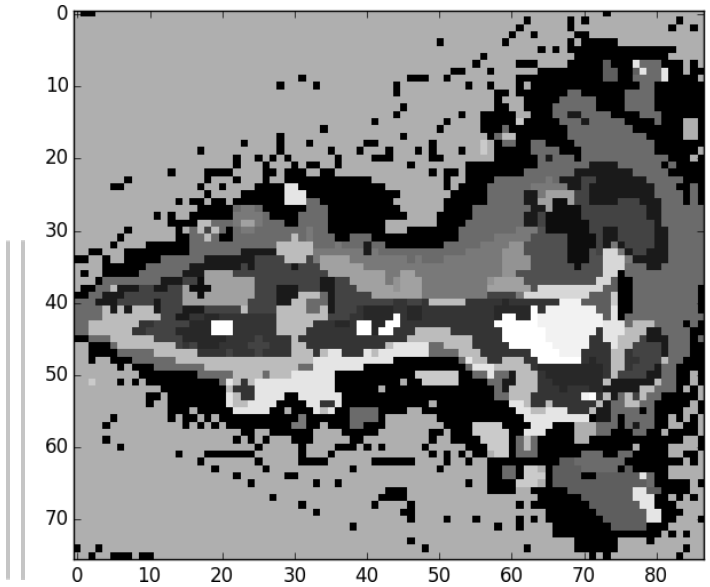
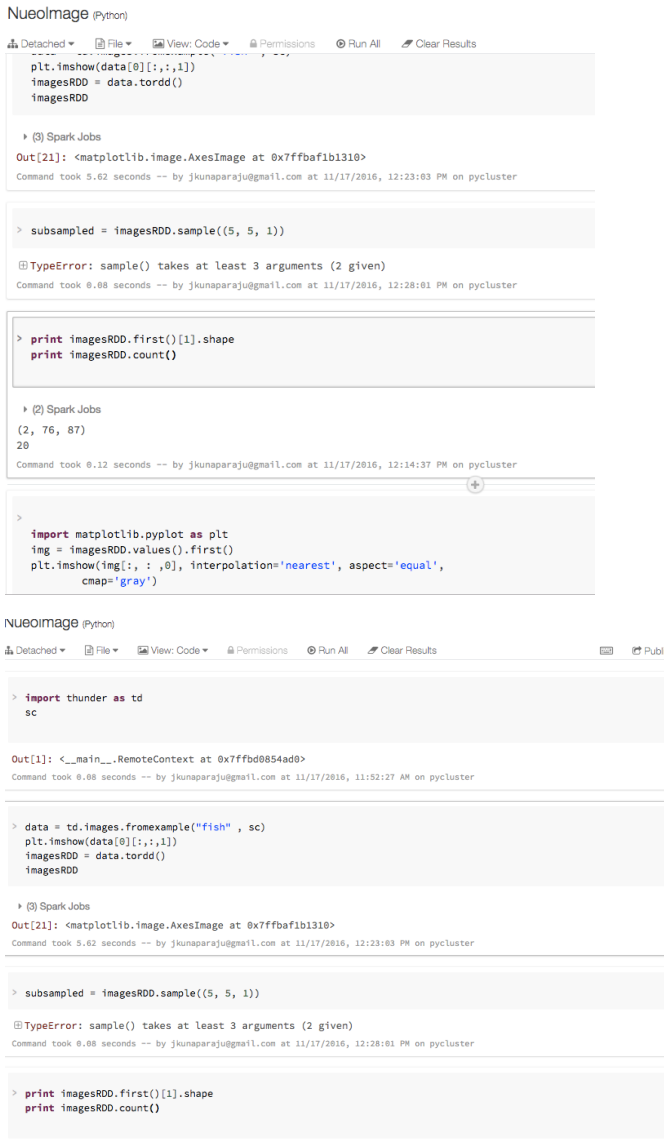


Figure6. Voxels colored by cluster

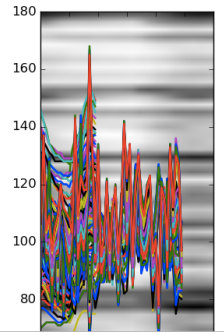
cmap_cat = ListedColormap(sns.color_palette("hls", 10), name='from_list')

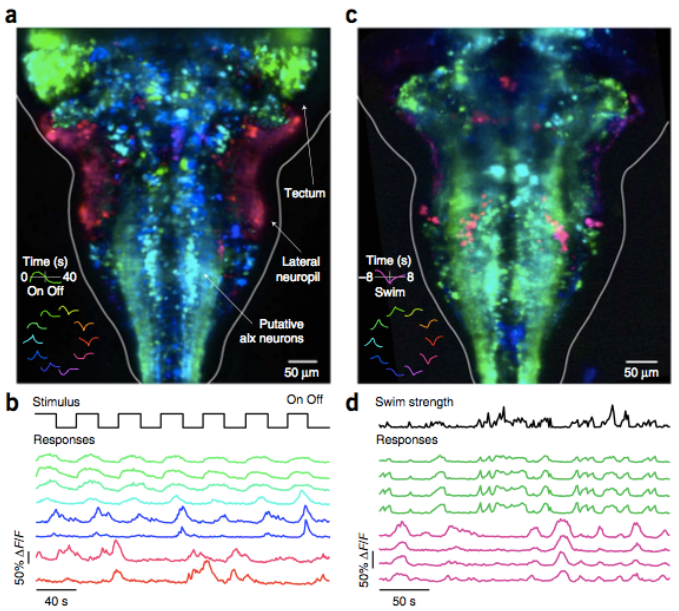
plt.imshow(by_cluster[:, :, 0], interpolation='nearest',

aspect='equal', cmap='gray')



```
examples = series.filter(lambda x: x.std() > 4).sample(n=50).toarray()
plt.imshow(examples, extent=[0,60,50,180 ]);
fig = plt.draw()
display(fig)
```





Maps of sensorimotor responses in larval zebrafish.

(a) Maps of response dynamics obtained by reducing each voxel to a pair of numbers using PCA. In the color wheel, the first two principal components are the red and yellow-green traces, and various linear combinations details a family of dynamics angle (hue) indicates response shape, and radius (brightness) indicates response strength. These values determine the hue and brightness for each voxel in the map

(b) Stimulus sequence and calcium responses of individual neurons; examples highlight different response types.

(c) Map derived from an experiment in which the fish swam sporadically in response to a constantly slowly moving stimulus.

(d) Swimming strength (from electrophysiological recordings) and calcium responses of individual neurons during self-driven swimming.

3.2

4 DISCUSSION

From the above results it is clear that cluster we made for analysis will capture certain behavior of zebra fish brain. If we have data with high resolution enough to resolve subcellular structures, we could first perform clustering of the voxels with k equal to an estimate of the number of neurons in the imaged volume. That would allow us to effectively map out the entire neuron cell bodies. If we do Time-series for each neuron, then it could be possible to determine different functional behavior more clearly. Analysis showed how these tools helped finding patterns of neural data. Whole brain maps of direction selectivity revealed differences in the organization of stimulus selectivity across brain areas.

Animal organisms have both sensory neurons and behaviors tuned to the direction of visual motion. In the zebrafish, mechanisms of direction selectivity have been characterized both at the single cell and the overall level. In this project neural responses are measured, while presenting fish with a whole-field moving stimulus that changed direction.

Direction selectivity is mostly characterized by measuring responses to moving patterns and fitting the responses with a model. In Thunder, a combination of mass-univariate tuning and regression can model ~10^8 time series in parallel, by first estimating the response to each direction and then fitting. Differences in tuning heterogeneity may signify qualitative differences in the types of computations that these areas perform. In particular, the coarse biases, likely reflect circuits underlying motor coordination, whereas the heterogeneity in visually responsive areas may reflect fine-scale visual computation. This analysis assumes unimodal tuning to direction, so responses only tuned to orientation, but not in direction, will appear untuned . By extending Thunder with bimodal tuning would be better.

Several factors motivated to build a library on Spark. Primary reason is Spark's data caching will solve some extent problems of data loading. It's is fundamental limitation of single-workstation solution. And it also solves the problem with distributed computing solutions built on the Hadoop MapReduce engine,

which Spark performs well in many computations. Iterative computing is most frequently occurring technic in machine learning. Caching is very important for iterative computations. Spark can be easily adaptable when compared to other existing system. Lowering learning curve because of its APIs, the abstraction of RDDs and optimized job scheduling, writing analyses in Spark is more concise than in other platforms and faster. It requires minimal control over the distribution and execution of work. Thunder can be implemented with Python with very minimal experience to do analysis. As spark along with Python, thunder work well with IPYTHON notebook which will be very useful in development for immediate visualization and results.

5 CONCLUSION

The analyses describes examined stimulus to behavior. This work stands out an important innovation in modern neuroscience. Recording from large number of neurons and using distributed computing to find patterns in very large data sets. Direction selectivity is commonly characterized by measuring responses to moving pattern In some advanced experiments and techniques, in which the fish swam in response to a constantly slowly moving stimulus(Ref1). Thunder offers variety of set of functionality like statistics on time series, clustering, tools for visualization and also it has modules for matrix factorizations, regression/classification. Without use of these tools zebrafish brain recordings would have been very difficult to analyze. Neurons in hindbrain and midbrain performed a variety of dynamics in response to the stimulus.

At present in computational field where large data set sizes are becoming normal. Like other domains neuroscience also rely on distributed computing. Thunder along with Python, spark is a major milestone in neuroscience analytics. For example, to use Matlab is also expensive when compared to spark apache open source projects. Thunder provides rich documentation and tools to launch in Amazon Ec2that creates a spark cluster with preinstalled thunder. Provided

benchmarking will give guide lines to select cluster size for Kmeans.

6 REFERENCES

Thunder in action, see the recent article by Thunder authors in Nature Methods (July 2014).

Freeman, Jeremy, et al. "Mapping brain activity at scale with cluster computing." Nature methods 11.9 (2014): 941-950.

Riza, Sandy, et al. Advanced Analytics with Spark: Patterns for Learning from Data at Scale. " O'Reilly Media, Inc.", 2015.

Freeman, Jeremy. "Open source tools for large-scale neuroscience." Current opinion in neurobiology 32 (2015): 156-163.

Ghatge, Mrs Dipali Dayanand. "Review: Apache Spark and Big Data Analytics for Solving Real World Problems."

Shoro, Abdul Ghaffar, and Tariq Rahim Soomro. "Big data analysis: Apache spark perspective." Global Journal of Computer Science and Technology15.1 (2015).

Armbrust, Michael, et al. "Scaling spark in the real world: performance and usability." Proceedings of the VLDB Endowment 8.12 (2015): 1840-1843.

Vladimirov, Nikita, et al. "Light-sheet functional imaging in fictively behaving zebrafish." Nature methods (2014).