Lessons Learned in Spike Sorting: The $n = 1$ Perspective

Eddie Yan

June 21, 2013

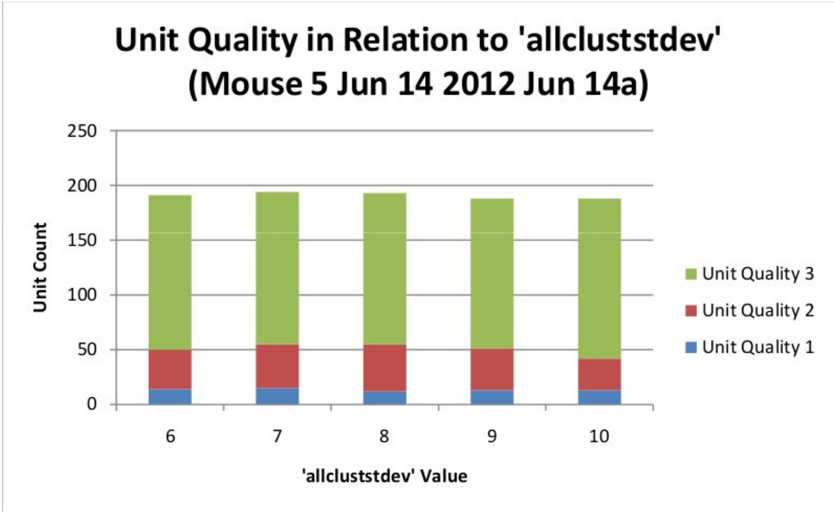
What I did

- October 2012–November 2012: Changing parameters (`allcluststdev`) and doing unit quality by hand
- November 2012–December 2012: Automatic unit quality via SNR
- January 2013: Changing `mergeclusterstdev`
- January 2013–February 2013: Trying to optimize
- March 2013–June 2013: Improving merge deliberation

Changing `allcluststdev` (Mouse 5 Jun14a)

- It doesn't seem to affect the quality of units produced at the end of sorting, at least with the range of parameters tried $\{6, 7, 8, 9, 10\}$
- Doing unit quality by hand on the same dataset again and again is tedious and prone to inconsistency

This Figure is Really Old (Mouse 5 Jun14a)



Auto-Unit Quality/Semi-automatic Unit Quality

- Doing unit quality by hand was extremely inconsistent and unreliable
- Find a metric to allow computers to do it automagically!
- What worked? SNR!
  - Sort of

Auto-Unit Quality/Semi-automatic Unit Quality and Quasi-SNR

Steps:

1. Interpret spikes at face value: a series of voltages in discrete time
2. Compute their root-mean-square (RMS) power
   $\sqrt{\frac{1}{n}\left(x_1^2 + x_2^2 + \ldots + x_n^2\right)}$
3. Why is it quasi-SNR? We treat the mean as the "signal" and simply subtract it from each of the spikes to get the "noise"
4. Get the signal to noise ratio
5. Decide unit quality
   - We can choose percentiles, further qualifiers, etc.
   - Qualifier that works well: restricting consideration to points near the peak of the spike

Pitfalls in Auto-Unit Quality

- Even with only 3 grades of unit quality, the number of empirically derived parameters grows quickly
- The process can be confused by high-SNR artifacts/non-units that would be caught by a human
- Best use case for auto-unit quality?
    - Consistent scoring of different sorting algorithms

Changing mergeclustdev (Mouse 5 Jun14a)

mergecluststdev	1	2	3
Unit Quality 1	45	43	40
Unit Quality 2	139	148	146
Unit Quality 3	103	104	106
Total	287	295	292
Merges in get_penultimate	260	61	24

- Observations
  - get_penultimate merges are usually not very significant
  - bulk of merges are done in get_final_units

Applying lessons learned with `mergecluststdev`: merges in `get_final_units`

- Goal is to improve merges in `get_final_units`
- Techniques tried:
  1. ~~Mahalanobis Distance~~
  2. Principal Component Analysis

Principal Component Analysis in One Slide

- **Motivation**: Units are messy to compare, as spikes each have ≈ 47 sampled points of amplitude
- Principal component analysis (PCA) allows us to transform each spike into 47 components of decreasing significance, so a comparing e.g. only the first three dimensions becomes reasonable (we go from $\mathbb{R}^{47}$ to $\mathbb{R}^{3}$)

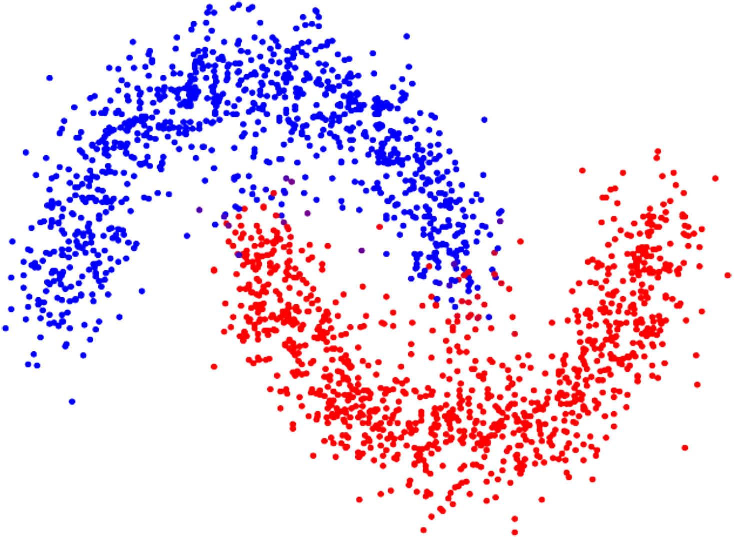
How PCA Was Used in Sorting

- Used in `get_final_units` as an alternative to the current Euclidean-distance based merge process
- For each unit **i**:
  1. Consider each PCA-transformed spike in $\mathbb{R}^3$ space
  2. Form a cluster of points corresponding to the spikes of that unit in $\mathbb{R}^3$
  3. Compare this unit with every other unit
  4. For the other unit **j**, also form cluster of points corresponding to spikes in $\mathbb{R}^3$
  5. Consider the distance between the clusters to decide if the two units should be merged (the smaller the distance between the clusters of two units, the more likely they should be merged)

Problems Encountered with PCA and Their Attempted Solutions

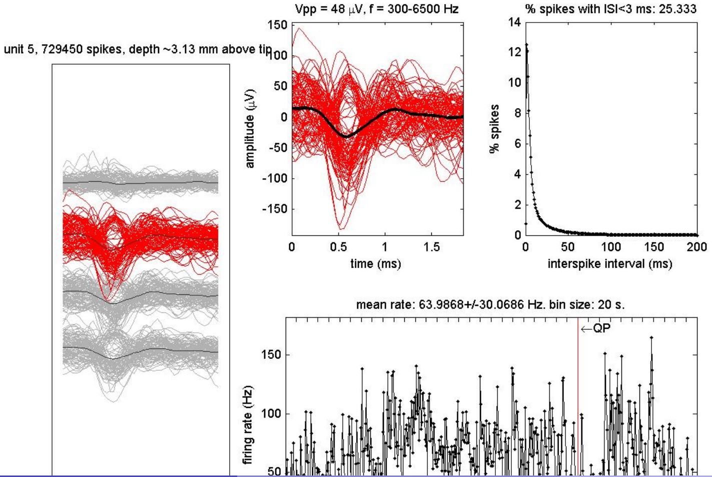
- The PCA merge process is not inherently scale-invariant
  - Normalize the data using z-scores
- The PCA merge process is more sensitive to “garbage units” than the old Euclidean-distance based merge process
  - Use intensive garbage-discarding/“sanity-checks”—`get_sane` before the merge process

A Toy Cluster



get_sane: An Exploration of Garbage Collection in Spike Sorting

- The old merge process was somewhat tolerant of “garbage units,” because it formed “mass graves,” of bad units



get_sane's sanity checks

1. If the peak-to-peak voltage of a unit is above a certain threshold, discard it
2. If the absolute maximum voltage of a unit is above a certain threshold, discard it
3. If the maximum `diff()` of the unit is below a certain threshold, discard it (it doesn't actually have a spike)
4. If the peak-to-peak voltage of a unit is below a certain threshold, discard it
5. Take the coefficient of variation of the minimum of each unit: $\frac{\sigma}{\mu}$ and discard this unit if it exceeds a certain threshold

Performance with and without get_sane (Mouse 48) Evaluated Manually

Process	get_sane,pca	get_sane,orig	orig	orig[1]
Qual. 1	43	25	20	38
Qual. 2	59	40	38	62
Qual. 3	119	57	128	227
Total	221	122	186	327

[1] updated merge process, not the copy of code I was working with

Performance with and without get_sane Evaluated Automatically

Process	get_sane,pca	pca	get_sane,orig	orig	orig [2]
Qual. 1	47	45	27	25	30
Qual. 2	67	75	38	28	64
Qual. 3	107	239	57	133	233
Total	221	359	122	186	327

[2] updated merge process, not the copy of code I was working with

How many units does get_sane discard?

Another way to phrase this question: How many units at the get_final step are viable?

- The short answer is: very few.
- On a typical dataset (Mouse 48), **95%** - **98%** units are discarded
  - Why does get_sane still produces a comparable number of units with this many being discarded? It does not overmerge bad units.

Further Ideas

- Use “fashionable” clustering techniques”
  - similarity-graphs
  - *k*-means
  - spectral clustering
- Unit Maturity

Spike Sorting from Scratch, Some Lofty Principles

- Use version control e.g. `git`
- Document!
- Reduce the number of parameters
- Build an iterative test suite for each "subroutine," with test cases and clearly defined expected output
- Design with parallelism in mind, consider exploring MATLAB's cluster functionality or UCLA's Hoffman2
- Use classes! Or in the very least, structs and functions, to organize data into predictable pieces
- Consider scope and variable names carefully–avoid making everything globally accessible and naming conflicts and know what the state of each variable should be at every step