Nerve-On-A-Chip Analyses

Problem statement

We are given a multi-electrode array (MEA) with P electrodes with neurons stretched across uniformly distributed electrodes. The neurons is anchored and electrode e_1 from which action potentials originate, and terminates at electrode e_p, and traverses electrodes $e_2, e_3, \ldots e_{p-1}$.

As an action potential propagates across the axon, each electrode will detect an increase in voltage as a function of time. The action potential will fist be detected by e_1 at t_1, e_2 at t_2, etc. As such, we can describe an action potential, A as a series of coordinates:

$$A = [(v_1, t_1), (v_2, t_2), \ldots, (v_p, t_p)] \tag{1}$$

Our goal is to identify the pairs (v, t) for a single (but generalized to any number of) action potential in order to determine compute the conduction velocity of a neuronal signal.

Initial approach

As a quick solution for the grant proposal that Bonnie and Alex were putting together, I wrote some *ad hoc* code to identify homologous action potentials across multiple electrodes. In brief, the approach can be summarized via the following steps:

1. Identify peaks in each of the P electrode recordings
2. Generate "peaklets" (P-tuples of peak indices, composed of one peak from each of the P electrodes)
3. Filter peaklets by some temporal criteria
4. Plot peaklet waveforms

Assumptions

We assume that the input data exists as a CSV file with the following structure:

Example CSV schema for electrode recordings.

Time (s)	Electrode 1	Electrode 2	$\cdots$	Electrode P
$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$

One column must be named "Time (s)". However, the others can be named in any format. The user must specify the order of electrode traversal.

Files are read in using `Pandas`, with peak-finding, peaklet-finding, and plotting performed using `numpy`, `scipy`, `pandas`, `matplotlib` and `seaborn`.

Code

Functions used for the analysis are contained in the `utils.py` file. A demonstration of the code can be found in the `noc_analysis.ipynb` file.

1. `Peaks`: Class containing peak-finding code. Similar to code utilized by Pulse3D.
    1. **Args**
        1. `max_twitch_frequency` (*required*): Defines the maximum frequency with which peaks can occur.
        2. `prominence_factor` (*required*): Scaling factor for peak prominences. Larger values result in more sensitive peak finding **more** sensitive (e.g. more peaks will be identified).
        3. `width_factor` (*required*): Scaling factor for peak widths. Larger values result in more sensitive peak finding **more** sensitive (e.g. more peaks will be identified).
        4. `height_factor` (*required*): Number of standard deviations above temporal signal mean required for a peak to be identified. Larger values make peak finding **less** senstive (e.g. fewer peaks will be identified).
    2. **Methods**:
        1. `fit`:
            1. **Args**
                1. `recordings` (*required*): Pandas DataFrame of recordings. One column must be named "Time (s)". The other columns are named by electrode names.
                2. `electrodes` (*optional*): List of electrode column names for which to compute peaks. If not provided, computes peaks for all electrodes.
            2. **Returns**:
                1. `electrode_peaks`: Dictionary, where keys are electrode names, and values are np.arrays of peak indices.
2. `Peaklets`: Class for identifying valid peaklets. Sequentially iterates over electrodes to build up 1-tuples, 2-tuples, ... P-tuples, filtering out invalid peaklets after each iteration. For large P, this approach is faster than building all possible P-tuples and then filtering.
    1. **Args**
        1. `max_peaklet_distance` (*required*): Maximum allowed time (in seconds) between peaks within a peaklet.

    2. `traversal_order` (*required*): List of electrode names, containing traversal order of electrodes that a neuronal signal takes.
2. **Methods**
    1. `fit` :
        1. **Args**
            1. `peaks` (*required*): Dictionary of previously identified electrode peaks.
            2. `recordings` (*required*): Pandas DataFrame of recordings. One column must be named "Time (s)". The other columns are named by electrode names.
        2. **Returns**:
            1. `peaklets` : Dictionary where keys are `indices` , `time` , and `voltage` , and values are the Pandas DataFrames of peaklet indices, the time of each peaklet index, and voltage of each peaklet index.
3. `plot_peaklets` : Method to plot waveforms around peaklets.
    1. **Args**:
        1. `peaklets` (*required*): Previously identified peaklet indices, times, and voltages.
        2. `recordings` (*required*): Pandas DataFrame of recordings. One column must be named "Time (s)". The other columns are named by electrode names.
        3. `index` (*required*): Peaklet index to plot.
        4. `window` (*required*): Number of time-samples before and after peaklet indices to include in waveforms.
        5. `cmap` (*optional*): Colormap dictionary mapping each electrode name to a color.
    2. **Returns**
        1. `fig` : matplotlib figure object

Possible future work: beamforming

Approach

Another way to solve the homologous peak identification problem is via the delay-and-sum beamforming algorithm (DAS). The DAS algorithm is most frequently used in the ultrasound imaging field, but is applicable to other settings related to directional signal transmission or reception (for example, in the case of multi-electrode recordings).

For each electrode, $e_1 \ldots e_p$, we apply an incremental time shift to the time signal, and then average the shifted signals. We assume that the action potential travels with a constant speed across the axon. As such, if the time-shift applied to the signal mirrors the time-delay required to travel between

electrodes, we'll see constructive interference when we sum the signals. Constructive interference will be maximized when the time-shift accounts entirely for the time-delay between electrodes. Otherwise, we'll see destructive interference.

In order to apply a time-delay, we compute the Fourier transform of our time-signal, $f(x)$ as

$$F(x) = \int_{-\infty}^{\infty} f(x) e^{-2\pi j x \omega} dx \tag{2}$$

and inverse Fourier transform as:

$$f(y) = \int_{-\infty}^{\infty} F(y) e^{-2\pi j y \omega} d\omega \tag{3}$$

and then multiply $F(x)$ by a complex exponential:

$$F_{sh}(x) = F(x) \times e^{-j\omega t_o} \tag{4}$$

In other words, we have:

$$x(t - t_o) \longleftrightarrow F(x) \times e^{-j\omega t_o} \tag{5}$$

Assume that the distance between any adjacent pair of electrodes is a constant distance d. For simplicity, let's assume that the time it takes a neuronal signal to travel to one electode to the next is t_o. Assume we are given a matrix of recordings $X \in \mathbb{R}^{N \times p}$ with N time-samples and P electrodes. We can then apply the DAS algorithm as follows:

$$DAS(X; t_o) = \frac{1}{P} \sum_{i=1}^{P} \int_{-\infty}^{\infty} F(x_i) e^{-j\omega(P-i) * t_o} d\omega \tag{6}$$

where the algorithm is parameterized by a constant time-shift t_o. Signals are shifted relative to the last electrode in the series (e.g. the electrode that the action potential traverses last). As such, we time-shift the first electrode the most, and don't time-shift the last electrode at all.

We can then begin to examine which time-shifts, t_0 result in the largest constructive interference, especially related to peak indices.

Similar approaches to beamforming

One could similar identify an optical time-shift by cross-correlating the (possibly windowed) electrode recordings and identifying the time-lag resulting in the peak signal correlation.

Things to consider

I've found that signal attenuation across electrodes is a noticeable artifact. If we can identify peaks corresponding to the same action potential across electrodes, we can model signal attentuation, possibly using some linear or exponential curve fitting procedure.