Flexible spike sorting in Python

Bartosz Telenczuk 1,2, Dmytro Belevtsoff 2,3

1 Insitute for Theoretical Biology, Humboldt University, Germany, Berlin
2 Neurology, Medical Uninversity Charite, Germany, Berlin
3 Bernstein Center for Computation Neuroscience, Germany, Berlin

Spike sorting is a common pre-processing step in the analysis of single or multi-unit activity (SUA or MUA). The goal of the procedure is to detect the times at which a single cell generated action potentials based on the extracellular recordings of an electric potential close to the cell. Since multiple cells can be active simultaneously special methods must be used to discriminate the responses of just a single cell. In many situations this is possible, because activities of different cells usually differ in wave shapes. Therefore it is possible to isolate them by comparing the shapes and amplitudes of detected extracellular spikes and grouping the ones which look similar (using automatic or manual clustering procedure).

There are dozens of different (commercial and free) software packages aimed at spike sorting. However, many of them are controlled only via graphical user interface (GUI) making them very inflexible. Testing new algorithms or adding support for new data formats usually requires in depth knowledge of the source code of the package (if available) and time-consuming development of extensions (if possible).

Therefore, we developed a new spike sorting library based on dynamic and interactive language (Python) called SpikeSort. While still very early in development, it offers many standard and not-so-standard algorithms for spike detection, feature extraction and spike classification. The main design goals of the library is the flexibility and extendibility allowing user to add new filters/algorithms/etc. without need to recompile or to understand the entire code base. In addition, the algorithms available in SpikeSort may be easily used in own scripts and programs.

We put much effort to make working with SpikeSort very interactive and intuitive. To this end, we employ a modular approach based on a set of components that may be flexibly combined to develop customized processing workflows.

Last but not least, we also take care of memory efficiency and performance. This is achieved among others by leveraging a set of third-party Python libraries (NumPy, PyTables and scikits.learn).