Convolutional Dictionary Learning with Grid Refinement for Spike Sorting - MVA 2023/2024

Theïlo Terrisse theilo.terrisse@eleves.enpc.fr
Florent Pollet florent.pollet@ens-paris-saclay.fr

1 Introduction and contributions

With the recent and steady increase of high-density neural recordings, for instance thanks to new probes like Neuropixels [1], fully automated spike sorting, a technique to identify and categorize individual action potentials from multiple neurons, has become more and more crucial to be able to track recorded neurons, and therefore deepen our understanding of the brain as well as assess probe's quality. In this regard, Convolutional Dictionary Learning (CDL) represents a promising framework to help solve this task, and many existing spike sorters resort to this for template matching, like MountainSort [2], Kilosort [3], or Spyking Circus [4]. However, these tools are far from being perfect and manual curation is often needed. In the paper [5], the authors suggest a potential improvement to decrease the time errors and increase template's quality thanks to a new Convolutional Orthogonal Matching Pursuit (COMP) algorithm with interpolation (INTERP).

In this mini-project, we aim at assessing if this grid refinement technique brings significant improvement for spike sorting applications. Granted, the authors compare their algorithm to other CDL algorithms like Alternating Descent Conditional Gradient (ADCG), Continuous Basis Pursuit (CBP) and standard COMP with electrophysiological signals, but they do not assess the overall impact on spike sorting performance. Therefore, we focus on applying this new COMP-INTERP algorithm as a part of a spike sorting pipeline and we measure the classification and matching performances on a synthetic and a real electrophysiological dataset. The code is made available at `https://github.com/florian6973/tsproj`.

[5] provides a part of its source code at `https://github.com/ds2p/srcdl`: the core of the COMP-INTERP algorithm is already implemented in its univariate variant, but it is only sparsely commented, not all the extensions and the tests mentioned in the article are present; the 2D variant is not complete either. We therefore used this (roughly 60%) as a starting point to build a spike sorting pipeline with *hydra* and *spikeinterface*. We also implemented many plotting and metric functions. We both worked equally on the code and on the report, Theilo focused on synthetic experiments, metrics and plotting, whereas Florent mainly investigated the integration of the existing code and the real dataset. If we reused the same design framework for the experiments (synthetic and real spike sorting datasets), we tested other datasets and we focused on global spike sorting performance instead of on a comparison of CDL methods (time and accuracy). We added preprocessing steps for the real dataset, and postprocessing steps to automatically compare the results to the groundtruths: confusion matrix, template reconstruction. The final code could be described as a single channel spike sorter prototype, compatible with any *spikeforest* or *spikeinterface* recording [6] [7].

2 Method

Recall on the framing of CDL CDL is interested in signals of duration $T > 0$ consisting in a noisy baseline sparsely scattered with patterns which are occurrences of a small number of possible sources (in spike-sorting, these are called *spikes* and are typically peak-shaped patterns, each corresponding to the activation of a specific neuron). We denote by $C \in \mathbb{N}^*$ the number of such sources, by h^c the normalized continuous-time pattern associated to source c, by $N^c \in \mathbb{N}^*$ the number of occurrences of source c in the signal under scrutiny, and by $\{\tau_i^c\}_{i\in[\![1,N^c]\!]}$ the (continuous) times of these occurrences. Denoting x_i^c the code for source c at occurrence i, the observed signal writes: $y(t) = \sum_{c=1}^{C}\sum_{i=1}^{N^c} x_i^c h_c(t-\tau_i^c) + \varepsilon(t)$ where ε is some *i.i.d.* white noise. In practice, the signal is sampled using a sampling frequency f_s resulting in a sampling grid of step $\Delta \in (0, T]$ composed of $N = \lfloor \frac{T}{\Delta} \rfloor$ points. The sources become of length $L < N$ and the observed event times are of length $n_i = \lfloor \frac{\tau_i^d}{\Delta} \rfloor$. Most CDL methods then consist in solving the following optimization problem:

$$\min_{\{\mathbf{h}_c\}_c, \mathbf{x}} \|\mathbf{y} - \mathbf{H}\mathbf{x}\|_2^2 \quad \text{s.t. } \|\mathbf{x}\|_0 \leq \beta \tag{1}$$

where β is some sparsity threshold, $\mathbf{x} \in \mathbb{R}^{C(N-L+1)}$ is the concatenation of the C sparse codes and $\mathbf{H} = \left[\mathbf{H}^1 \middle| \dots \middle| \mathbf{H}^C\right] \in \mathbb{R}^{N\times C(N-L+1)}$ encodes the convolution operation with the sources: $\mathbf{H}^c\mathbf{x}^c = \mathbf{x}^c * \mathbf{h}_c$. Note that other sparsity constraints can be applied, for instance using the relaxed, convex ℓ^1 norm; but ℓ^0 has the advantage of encouraging fast greedy methods.

Principle of COMP Since the problem is not convex, it is usually tackled by alternating between Convolutional Sparse Coding (CSC), which is the optimisation on the codes $\mathbf{x}$, and Convolutional Dictionary Update (CDU), which is the optimization on the dictionary $\mathbf{H}$. In the case of COMP, CSC alternates between

1. a "selection step", in which at iteration t', one selects the time-shifted source $(c^{t'}, i^{t'}) \in [\![1, C]\!] \times [\![1, N-L+1]\!]$ such that $\mathbf{H}^{(c^{t'},i^{t'})}$ has the highest cross-correlation with the current residual $r^{(t')}$ and adds it to the current set $S^{(t')}$ of selected sources. To avoid storing $\mathbf{H}$ and speed up computations, this is typically done by working out the $h_c \star r^{(t')}[i]$ on-the-run.
2. a "projection step", in which the new residual $r^{(t'+1)}$ and associated code $\mathbf{x}|_{t'+1}$ are obtained by projecting the signal onto the span of $\mathbf{H}|_{t'+1}$ (*i.e.* $\mathbf{H}$ restricted to its columns in $S^{(t'+1)}$): $\mathbf{x}|_{t'+1} = (\mathbf{H}|_{t'+1}^{\mathrm{T}}\mathbf{H}|_{t'+1})^{-1}\mathbf{H}|_{t'+1}^{\mathrm{T}}\mathbf{y}$ and $r^{(t'+1)} = y - \mathbf{H}|_{t'+1}\mathbf{x}|_{t'+1}$.

Limitations of COMP This approach to CDL restrains to working on the sampling grid, which has two limitations. Firstly, an event from source c may occur off the sampling-grid, meaning that $n_i \neq \tau_i^c$. As a result, the sampled segment may differ from the segment obtained by sampling the same source, but occurring exactly on the sampling grid, resulting in a miss-identification. Secondly, even if the correct source is identified, traditional COMP can only locate the event on the sampling-grid, inducing a temporal error (see Figure 1). Additionally, The CSC step can be slow for high-dimensional matrices, due to the pseudo-inversion of $\mathbf{H}|_{t'}$.

Improvements brought by COMP-INTERP To solve these issues, Andrew H. Song *et al.* [5] introduce an interpolation of the templates of the dictionary and a sped-up projection step of COMP. More precisely, for a given sub-sampling step $\Delta_K := \frac{\Delta}{K}$ with $K \in \mathbb{N}^*$, the dictionary $\mathbf{H}$ is augmented into a larger dictionary $\mathbf{H}_{\text{INTERP}} \in \mathbb{R}^{N\times CK(N-L+1)}$, obtained by duplicating and shifting each h_c by $k\Delta_K$, for $k \in [\![1, K]\!]$. Such a shift is obtained by using a smooth interpolator function f (for instance,

the sinc interpolator) to interpolate h_c into a continuous-time function $\tilde{h}_c$, then shifting it by $k\Delta_K$, and finally resampling it on the original sampling grid to get $\mathbf{h}_{(c,k)} = \tilde{h}_c(n\Delta - k\Delta_K)$. Denoting $\mathbf{f}^k[n] = f(n\Delta - k\Delta_K)$ for $n \in [\![-\frac{L-1}{2}, \frac{L-1}{2}]\!]$ (that is, f shifted by $k\Delta_K$ and resampled), this writes: $\forall n \in [\![-\frac{L-1}{2}, \frac{L-1}{2}]\!], \mathbf{h}_{(c,k)}[n] = \mathbf{h}_c * \mathbf{f}^k[n]$. In other words, denoting $\mathbf{F}^k \in \mathbb{R}^{L\times L}$ the Toeplitz matrix encoding the convolution with (zero-padded) $\mathbf{f}^k$, $\mathbf{h}_{(c,k)} = \mathbf{F}^k\mathbf{h}_c$. With this interpolated dictionary in hand, problem (1) rewrites:

$$\min_{\{\mathbf{h}_{(c,k)}\}_{(c,k)},\mathbf{x}} \|\mathbf{y} - \mathbf{H}_{\text{INTERP}}\mathbf{x}\|_2^2 \quad \text{s.t.}\|\mathbf{x}\|_0 \leq \beta \tag{2}$$

where $\mathbf{x}$ now lives in $\mathbb{R}^{CK(N-L+1)}$. The algorithm proposed in [5] is then similar to COMP, up to a few variations. The CSC step is the same, but using $\mathbf{H}_{\text{INTERP}}$ instead of $\mathbf{H}$. Furthermore, since this new dictionary can be of high dimension, some optimisations are used by the authors. To speed up the selection step, the authors re-use an idea from locOMP [8], namely that since templates are much shorter than the residual, $r^{(t')}$ and $r^{(t'+1)}$ only differ locally, and thus cross-correlations only have to be re-computed for segments on which the two differ. Additionally, using an idea given in [9], the authors speed up the projection step by replacing all matrix multiplications involving $\mathbf{H}|_{t'}$ by cross-correlations, and by using the Cholesky factorisation $\mathbf{H}|_{t'}^{\mathrm{T}}\mathbf{H}|_{t'} = L^{(t')}(L^{(t')})^{\mathrm{T}}$, with $L^{(t')}$ a lower-triangular matrix, to ease its inversion. This factorisation can be performed efficiently, since $\mathbf{H}|_{t'}$ is obtained from $\mathbf{H}|_{t'-1}$ simply by adding one column. Therefore, knowing that $\|h^{(t')}\|_2 = 1$:

$$\mathbf{H}|_{t'}^{\mathrm{T}}\mathbf{H}|_{t'} = \begin{pmatrix} \mathbf{H}|_{t'-1}^{\mathrm{T}}\mathbf{H}|_{t'-1} & \mathbf{H}|_{t'-1}^{\mathrm{T}}\mathbf{h}^{(t')} \\ (\mathbf{h}^{(t')})^{\mathrm{T}}\mathbf{H}|_{t'-1} & 1 \end{pmatrix} \quad \text{so} \quad \mathbf{L}^{(t')} = \begin{pmatrix} \mathbf{L}^{(t'-1)} & 0 \\ \mathbf{w}^{\mathrm{T}} & \sqrt{1 - \|\mathbf{w}\|_2^2} \end{pmatrix} \tag{3}$$
$$\text{where } \mathbf{L}^{(t'-1)}\mathbf{w} = \mathbf{H}|_{t'-1}^{\mathrm{T}}\,\mathbf{h}^{(t')}$$

CDU step CDU then consists in solving (1) with fixed $\mathbf{x}$. [5] solves this using shift-invariant K-SVD [10]. This method solves the problem one template h_d at a time; the solution is given by

$$\widehat{\mathbf{h}}_d = \left(\sum_{k=0}^{K-1}\sum_{i=1}^{N^{(d,k)}}\sum_{m=1}^{N^{(d,k)}} x_i^{(d,k)} \left(\mathbf{F}^k\right)^{\mathrm{T}} \left(\mathbf{S}_i^d\right)^{\mathrm{T}} \mathbf{S}_m^d\, \mathbf{F}^k\, x_m^d\right)^{-1} \times \left(\sum_{k=0}^{K-1}\sum_{i=1}^{N^d} x_i^d \left(\mathbf{F}^k\right)^{\mathrm{T}} \left(\mathbf{S}_i^d\right)^{\mathrm{T}} \mathbf{E}^{\mathbf{d}}\right) \tag{4}$$

where $\mathbf{S_i}$ is the matrix that shifts $\mathbf{h}^d$ to its position n_i, and $E^d = \mathbf{y} - \sum_{c\neq d}^{C}\sum_{i=1}^{N^c} x_i^c \mathbf{S}_i^c \mathbf{h}_c$. Remark that problem (1) would traditionally include a constraint on the norm of the sources to ensure convergence of the algorithm. In practice, [5] solves this issue by normalizing the obtained dictionary after each update. The detailed algorithm can be found in [5].

Window processing [5] introduces the notion of windows and segments, to speed up processing. The signal can be split equally in non-overlapping windows, or in short segments that contain the events of interest. With windows or segments, the CSC step can be parallelized, even though it adds some complexity related to boundary conditions and an additional external sum for the CDU step. In practice, we do not apply this parallelization as it brings no time gain for a small number of short segments.

3 Data

To experiment, we rely on two datasets. The first, synthetic one is used to explore in-depth the capabilities of the new algorithm, similar to the one used in the original paper [5]. The second, real one is used to assess the performance of the algorithm in concrete spike sorting applications.

Synthetic dataset The synthetic dataset is based on a very simple model that could account for electrophysiological recordings. It is based on a white Gaussian noise signal, in which at times uniformly selected at random we add some events based on short exponentially oscillating templates h_i:

$$h_1(t) \propto (10^3 t) \exp\left(-(10^3 t)^2\right) \cos\left(\frac{\pi}{2}(10^3 t)\right) \quad \text{and} \quad h_2(t) \propto (10^3 t) \exp\left(-(10^3 t)^2\right)$$

We construct a signal of length $T = 3s$ by generating 10 events of length $10ms$ per template, adding random noise and sampling with a frequency of $8kHz$. We select amplitudes and the noise level such that the Signal-to-Noise Ratio (SNR) is around 25 dB, which allows us to easily distinguish the templates as spikes above the average noise. We respect the refractory period as we want to model real firing patterns of neurons, so events are not overlapping. The signal has only one channel. In Figure 2, from the appendix, we plot the signal and the two templates. No pre-processing will be applied, as the SNR is set manually and used to test the algorithm's robustness to noise.

Real dataset This real dataset is available through the *spikeforest* Python package [7], and we can easily manipulate it with the *spikeinterface* library [6]. It comes from a paired intracellular and extracellular recording, so it has one groundtruth unit (neuron) among all the spikes that are recorded by the probe. However, other spikes may be present and come from other neurons that are not in the groundtruth. It is 300 seconds long and it has 64 channels.

First and foremost, once we download the dataset, we denoise and filter the traces. To denoise, we can use the common noise removal strategy which aims at removing shared artifacts between channels, and we can add whitening to remove spurious spatial correlations. Moreover, the spikes are high frequency components of the signal, so we want to remove all low frequency components (like the local field potential), that can be caused by some lights or electric sources (60 Hz) for instance. We plot the spectogram of the first 50 seconds in Figure 8 to check that it makes sense to perform a Fourier Transform in this regard, even though we know the signal is not stationary and ergodic, because of the very short spikes at random. We notice in fact that the recording is already filtered and relatively denoised. We also check there is no specific trend by a linear fit. Since the implemented algorithm is only 1D, we select the channel where the spikes related to the groundtruth unit have maximum SNR (around 25 dB, like the synthetic dataset). In Figure 7, from the appendix, we plot the first seconds of the signal, and the spikes corresponding to the groundtruth unit, as a reference for later.

4 Results

The goal of the following experiments are twofold: to check the relevance of the CDL framework for spike sorting, and to assess the potential benefits of using grid refinement in this regard.

To answer these questions, we define several metrics:

- the accuracy, along with a confusion matrix. A detected event matches the groundtruth event if the time difference between the two is below a threshold, defined as 4 ms for the real dataset. Each unit is matched with a groundtruth unit based on the Hungarian algorithm.
- the template reconstruction error: [5] introduces the error metric $\text{err}(\hat{\boldsymbol{h}_c}, \tilde{\boldsymbol{h}_c}) = \sqrt{1 - \langle \hat{\boldsymbol{h}_c}, \tilde{\boldsymbol{h}_c} \rangle^2}$. We did not use the Dynamic Time Warping (DTW) metric, since we want an exact matching of the templates. The issue with the chosen norm is that is not invariant to small time-shifts, as shown in Figure 3. Therefore, we generalize this metric and take the minimum of the error for all possible shifts (with rolling and opposite) of $\tilde{h}_c$.

- the preprocessing, CSC and CDU times.

Synthetic data To get a better sense of all tunable hyperparameters, we start with synthetic experiments. While the article [5] focuses on performance comparison with other methods, we run the algorithm in various conditions, for 10 iterations each, without and with interpolation ($K = 10$), following the parameters listed in Table 1. For each experiment, the dictionary is learnt on a training dataset, where we extract segments of the signals when a given threshold is crossed, proportional to the median absolute deviation. The accuracy and time are then measured on a separate test dataset, here without splitting into segments. The results are summarized in Table 2. Experiments 1 to 4 illustrate the sensitivity to initial conditions, as a random initialization degrades performance when compared to the noisy true templates. Both algorithms also preserves initial horizontal shifts (see Figure 3), which is fine for reconstruction as the sparse code can compensate for the offset, but it makes the interpretation of event detection less reliable in a broader context. Experiment 5 shows that performance are degraded when adding 3 sources (shown on Figure 4), notably because units 2 and 4 are similar, simple and very thin, thus inducing false positives. Experiments 6 and 7 are the only ones where interpolation gives a real advantage: it appears to be slightly more robust to background noise; more importantly the added-value of COMP-INTERP arises when the sampling frequency is low, notably thanks to a finer reconstruction as illustrated on Figure 5. Lastly, experiment 8 shows the importance of the choice of the interpolator, as the cubic interpolator brings poor reconstruction error, notably due to oscillations visible on Figure 6. Further details on the experiments can be found in the notebook of the project.

Spikeforest data Contrary to what is done in the paper [5], we have selected a dataset for which we have benchmarks from other spike sorters on the spikeforest website [7]. Here again, segments centered on peaks (shown in Figure 7) are extracted from the signal for processing. Since we roughly see two potential clusters, we set $C = 2$. We tested with $C = 1$, but it brings many false positives, since all the other spikes are assigned to the single unit. To mitigate that, we can increase the peak detection threshold, and then it works well with $C = 1$. We tested with $C = 3$ too, but in this case the templates are not well initialized by KMeans, so the single unit is split between two dictionary templates. With $C = 2$, there is no significant difference between COMP and COMP-INTERP, as Table 3 and Figure 9 show. Table 3 also presents the running times of both COMP and COMP-INTERP, for which CSC is significantly longer (roughly ten times). Based on Spikeforest benchmarks, traditional spike sorters have an accuracy of roughly 0.95, performance which is almost reached by our simple pipeline demonstrating an accuracy of 0.91.

Discussion Considering the aforementioned experiments, grid refinement brings no significant improvement for spike sorting, given the additional computational cost. The main pathways to improve spike sorting do not lie in grid refinement but in other aspects such as pre-processing, post-processing and precise clustering: template matching through CDL is only successful when the dictionary is initialized with the right number of templates, each one belonging to a different unit. Then, CDL runs very well, and can efficiently perform matching on selected segments only (not the whole signal) because considering the whole high-frequency signal would be too computationally intensive. Moreover, there are many hyperparameters to tune to get relevant results, and some methods should be investigated to try to simplify this potentially intensive task. Grid refinement is much more relevant for signals which have lower sampling frequency relative to the duration of the patterns of interest. Further improvements could consist in testing COMP-INTERP with a multivariate signal, parallelizing more the code (CSC) and using new graph signal processing tools to take into account the probe geometry.

References

[1] J. J. Jun, N. A. Steinmetz, J. H. Siegle, *et al.*, "Fully integrated silicon probes for high-density recording of neural activity," *Nature*, vol. 551, pp. 232–236, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:205262002.

[2] J. E. Chung, J. F. Magland, A. H. Barnett, *et al.*, "A fully automated approach to spike sorting," *Neuron*, vol. 95, no. 6, 1381–1394.e6, 2017, ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2017.08.030. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0896627317307456.

[3] M. Pachitariu, S. Sridhar, and C. Stringer, "Solving the spike sorting problem with kilosort," *bioRxiv*, 2023. DOI: 10.1101/2023.01.07.523036. eprint: https://www.biorxiv.org/content/early/2023/01/07/2023.01.07.523036.full.pdf. [Online]. Available: https://www.biorxiv.org/content/early/2023/01/07/2023.01.07.523036.

[4] P. Yger, G. L. Spampinato, E. Esposito, *et al.*, "A spike sorting toolbox for up to thousands of electrodes validated with ground truth recordings in vitro and in vivo," *eLife*, vol. 7, D. Kleinfeld, Ed., e34518, Mar. 2018, ISSN: 2050-084X. DOI: 10.7554/eLife.34518. [Online]. Available: https://doi.org/10.7554/eLife.34518.

[5] A. H. Song, F. J. Flores, and D. Ba, "Convolutional Dictionary Learning With Grid Refinement," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2558–2573, 2020, Conference Name: IEEE Transactions on Signal Processing, ISSN: 1941-0476. DOI: 10.1109/TSP.2020.2986897. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9064880 (visited on 12/31/2023).

[6] A. P. Buccino, C. L. Hurwitz, S. Garcia, *et al.*, "Spikeinterface, a unified framework for spike sorting," *Elife*, vol. 9, e61834, 2020.

[7] J. Magland, J. J. Jun, E. Lovero, *et al.*, "Spikeforest, reproducible web-facing ground-truth validation of automated neural spike sorters," *eLife*, vol. 9, M. Meister, R. L. Calabrese, and M. Meister, Eds., e55167, May 2020, ISSN: 2050-084X. DOI: 10.7554/eLife.55167. [Online]. Available: https://doi.org/10.7554/eLife.55167.

[8] B. Mailhé, R. Gribonval, P. Vandergheynst, and F. Bimbot, "Fast orthogonal sparse approximation algorithms over local dictionaries," *Signal Processing*, Advances in Multirate Filter Bank Structures and Multiscale Representations, vol. 91, no. 12, pp. 2822–2835, Dec. 2011, ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2011.01.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165168411000053 (visited on 01/01/2024).

[9] R. Rubinstein, M. Zibulevsky, and M. Elad, "Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit," en, 2008.

[10] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Transactions on Signal Processing*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006, Conference Name: IEEE Transactions on Signal Processing, ISSN: 1941-0476. DOI: 10.1109/TSP.2006.881199. [Online]. Available: https://ieeexplore.ieee.org/document/1710377 (visited on 12/31/2023).

A Supplementary tables

#	Num. sources	Events/src	Noise std	Init.	Sampl. freq.	Interpolator
1	2	10	0.1	Random	8kHz	sinc
2	2	10	0.1	GT	8kHz	sinc
3	2	10	0.1	Noisy GT	8kHz	sinc
4	2	10	0.1	Offset GT	8kHz	sinc
5	5	3	0.1	Noisy GT	8kHz	sinc
6	2	10	0.3	Noisy GT	8kHz	sinc
7	2	10	0.1	Noisy GT	4kHz	sinc
8	2	10	0.1	Noisy GT	8kHz	cubic

Table 1: Parameters for the experiments on the synthetic dataset
"Noisy GT" is the ground truth with added noise of std. 0.2, and "Offset GT" with an offset of $+2ms$

Experiment	Accuracy		Distance to GT templates		Temporal shift (ms)	
	Interp	No interp	Interp	No interp	Interp	No interp
1	0 ± 0	0.12 ± 0.24	0.92 ± 0.21	0.82 ± 0.35	0.95 ± 2.63	1.05 ± 2.01
2	1 ± 0	1 ± 0	0.07 ± 0.03	0.07 ± 0.03	0 ± 0	0 ± 0
3	0.80 ± 0.40	1 ± 0	0.21 ± 0.31	0.11 ± 0.05	-0.19 ± 0.27	-0.20 ± 0.10
4	1 ± 0	1 ± 0	0.08 ± 0.02	0.07 ± 0.02	2 ± 0	2 ± 0
5	0.39 ± 0.42	0.42 ± 0.42	0.35 ± 0.34	0.34 ± 0.30	-0.11 ± 0.58	0.09 ± 0.27
6	0.96 ± 0.09	0.75 ± 0.38	0.21 ± 0.03	0.22 ± 0.06	-0.06 ± 0.08	0.16 ± 0.37
7	1 ± 0	0.91 ± 0.11	0.07 ± 0.03	0.13 ± 0.08	-0.10 ± 0.12	0.03 ± 0.13
8	0.85 ± 0.32	0.9 ± 0.3	0.28 ± 0.30	0.11 ± 0.05	-0.04 ± 0.11	-0.09 ± 0.41

Table 2: Results for the synthetic dataset (averaged over 5 runs and over the sources)
Running time per iteration of CSC (with and without interpolation respectively):
$631 \pm 19ms$ and $66 \pm 2ms$ with 2 sources and $3863 \pm 110ms$ and $406 \pm 65ms$ with 5 sources.

Experiment	Accuracy		Distance to avg GT waveforms		Time CSC - CDU (s)	
	Interp	No interp	Interp	No interp	Interp	No interp
1 (2 sources)	0.91	0.91	0.02	0.01	15.0 - 0.8	1.4 - 0.6
2 (3 sources)	0.00	0.50	0.05	0.02	23.5 - 1.5	2.0 - 0.8
3 (1 source)	0.91	0.91	0.03	0.04	7.8 - 0.5	0.8 - 0.4

Table 3: Summarized results for the experiments on the real dataset

B Supplementary figures

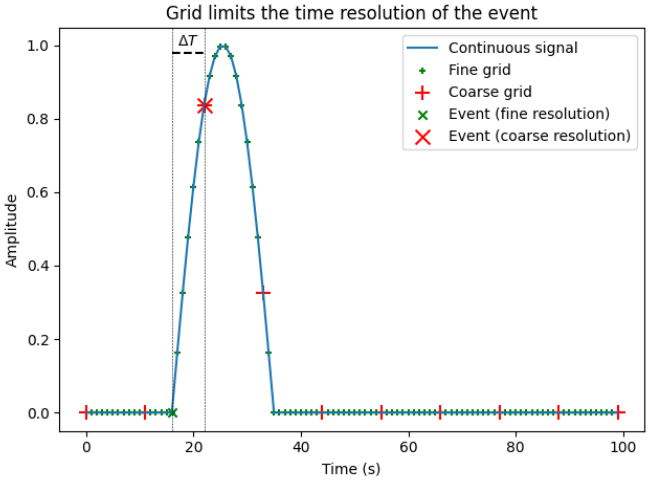
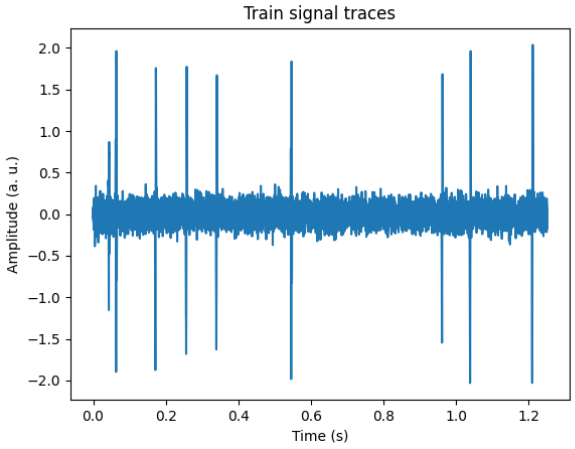
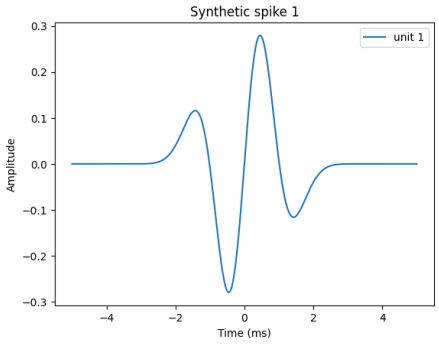


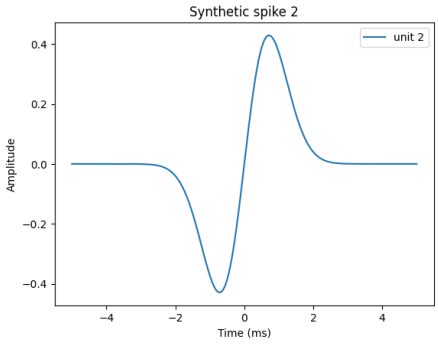
Figure 1: Discretization generates a time-quantization error ΔT if the grid is too coarse



(a) First second of the signal

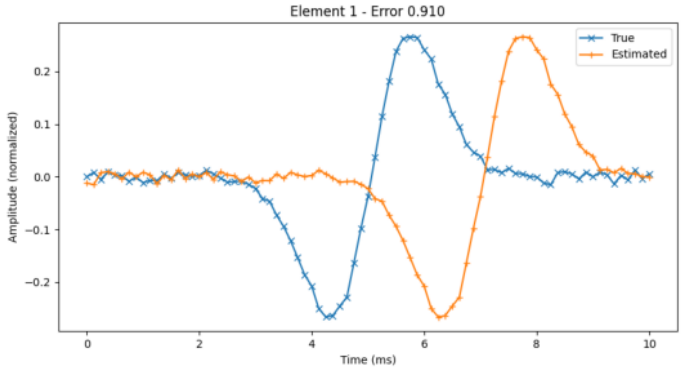


(b) Synthetic spike 0

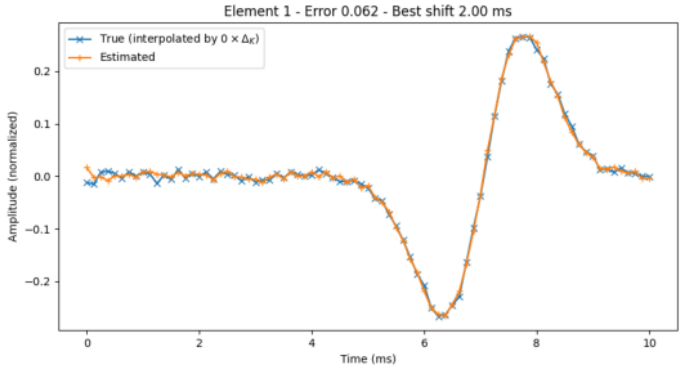


(c) Synthetic spike 1

Figure 2: The synthetic dataset



(a) With horizontal shift



(b) Without horizontal shift

Figure 3: Distance to the true template with and without shift, for the synthetic dataset

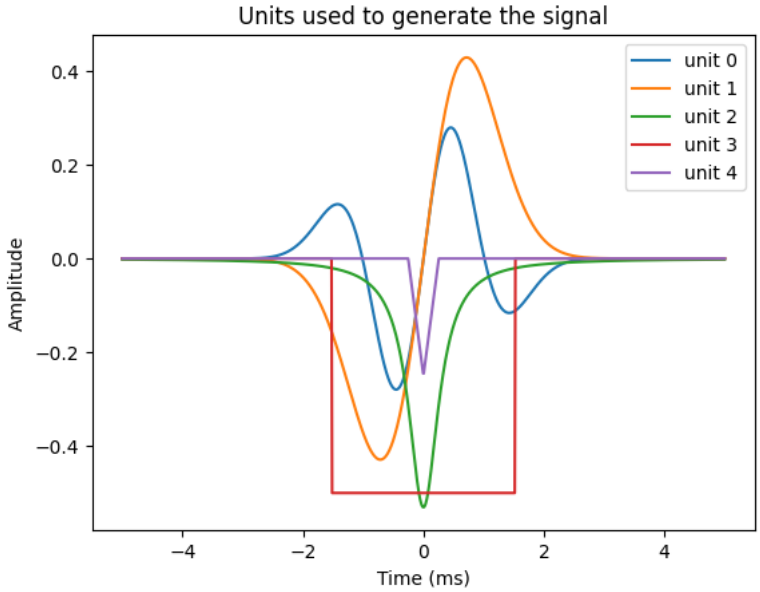
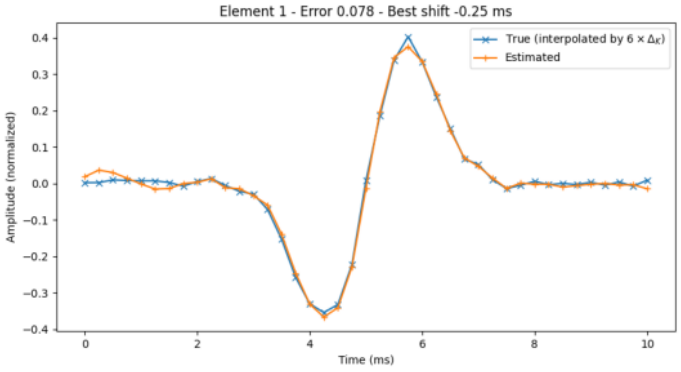
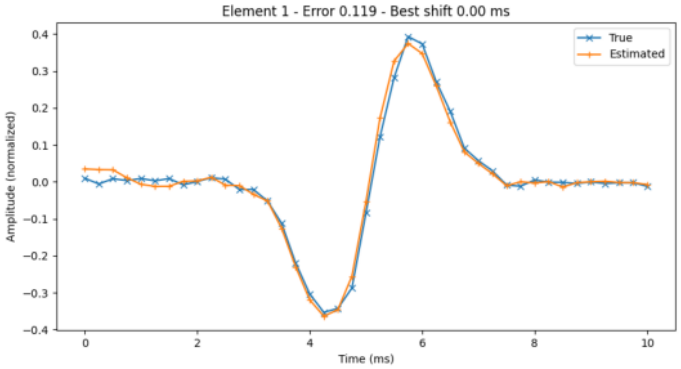


Figure 4: Sources used for the experiment 5 on the synthetic dataset



(a) With grid refinement



(b) Without grid refinement

Figure 5: Reconstruction of unit 1 for low sampling frequency (experiment 7 on synthetic dataset)

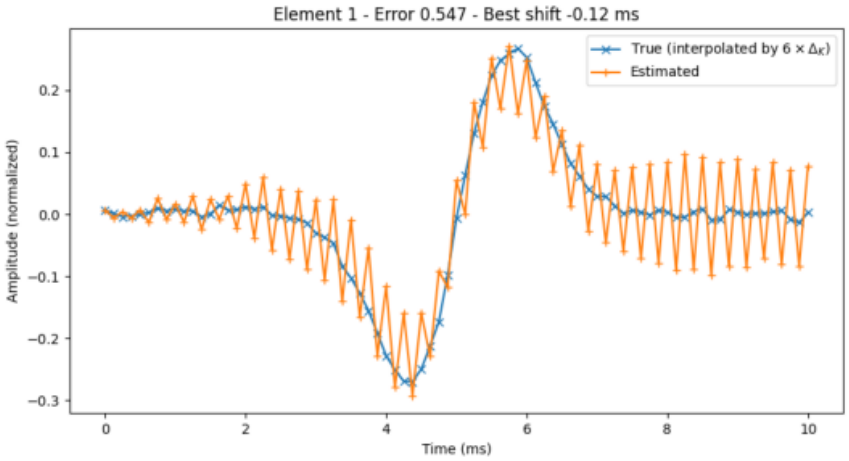
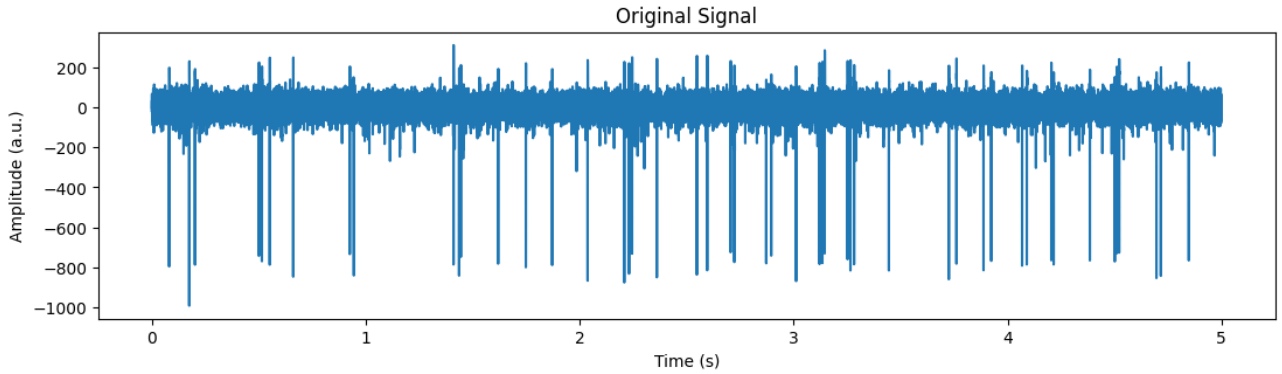
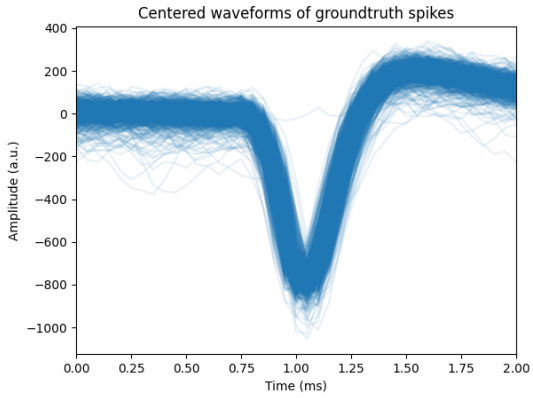


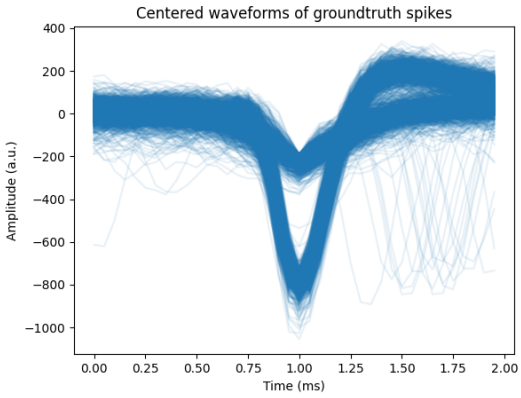
Figure 6: Reconstruction of unit 1 using cubic interpolator (experiment 8 on synthetic dataset)



(a) First seconds of the real signal. We clearly distinguish the neuron spikes



(b) Waveforms of the groundtruth unit



(c) Detected peaks to be classified

Figure 7: Real signal overview

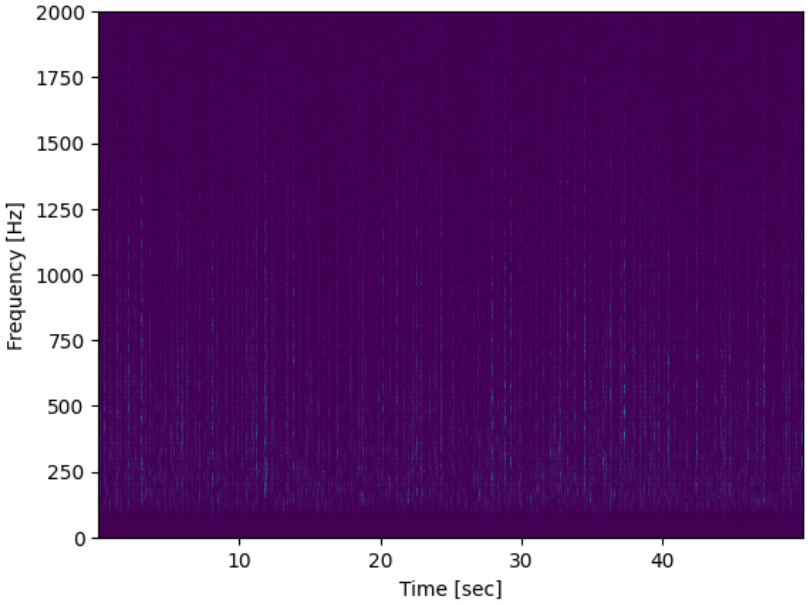
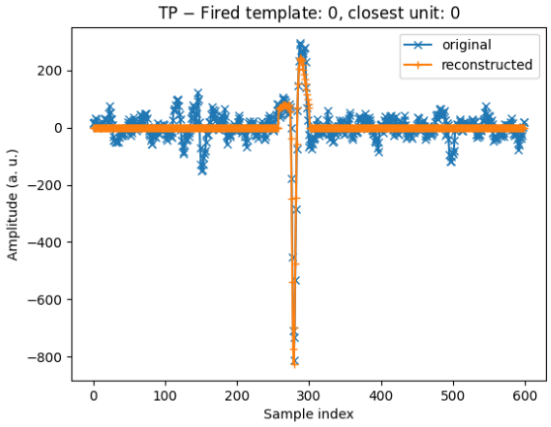
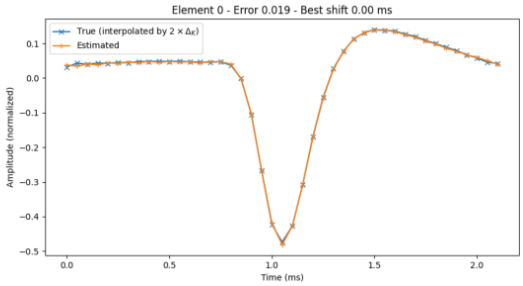


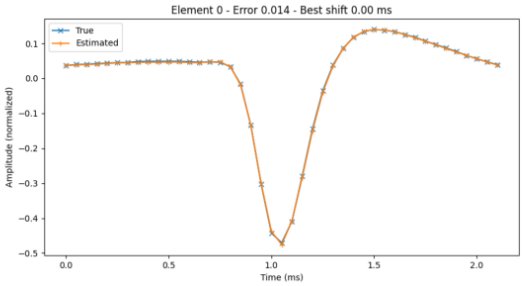
Figure 8: Spectrogram of the real denoised signal



(a) Example of well detected and classified spike



(b) Learnt template *vs* ground truth (No interp)



(c) Learnt template *vs* ground truth (Interp)

Figure 9: Spike sorting on real dataset with two templates