Signal Processing Extracellular Neuron Recordings

Noam Almog

Abstract

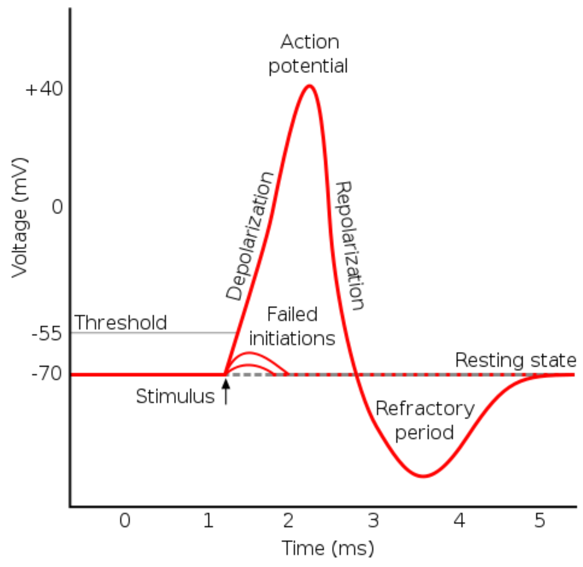
In this project we looked at an extra-cellular recording of neurons in the rat hippocampus with the goal of identifying and classifying individual neurons firing. The neural recordings constituted a signal of voltage over time on 16 different channels, each channel records from a 17 micron wire implanted in the rat's hippocampus. Given the 32 KHz resolution of the recording process across 16 channels, 1 second of recording generates 0.5 million 32bit integer samples per second. This amount of data necessitates the use of techniques in signal processing. For this project we process and analyze 60 seconds of recording utilizing methods in Fourier transform, high-pass filtering, PCA (SVD), and K-Means clustering. After applying these techniques we are able to clean and process the recording sufficiently to where it is possible to clearly identify the signatures of 3 distinct neurons.

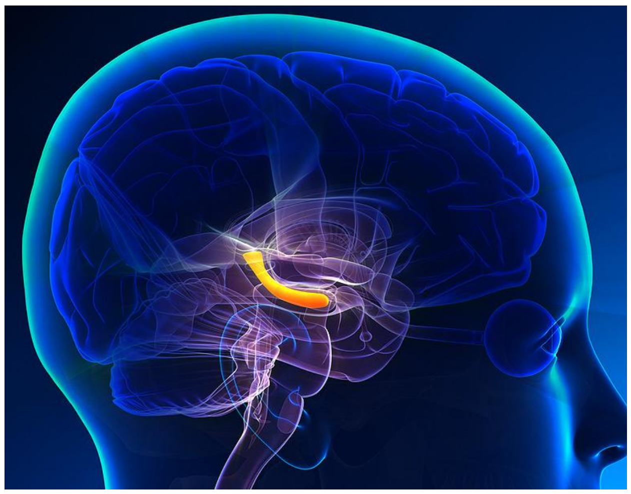
Input Signal

Background

A neuron is a specialized cell found in most multicellular-organisms that comprises of receptors for input from other neurons and a means of sending an output signal through the release of a molecular discharge. This chemical signal has a measurable voltage. Neurons' electrical activity and the connections between them known as neural networks are widely considered by modern science to be the mechanism by which humans derive consciousness, memory, learning and behavior. The hippocampus in particular is thought to be highly associated with learning and memory function in the brain.

Despite the immense potential for progress in understanding the mechanics of brain function, neuroscience continues to struggle to produce larger results. This is largely because of the sheer number of neurons in the brain and the difficulty in observing neurons in living organisms due to their location in the body. There are around 100 billion neurons in the human brain. To access these neurons one must first manage to circumvent the skull, which shields neurons both physically and electro magnetically from observation, and then navigate the densely layered, folded structure of the brain. A typical neuron firing, also called an *Action Potential*, occurs on the time order of 1-3 milliseconds, with a voltage on the order of 70-150 millivolts when observed from within the cell, and 70-150 microvolts when observed extracellularly. Given the amount of noise when trying to observe a biological signal on this voltage scale, signal processing is immensely useful in discerning neural activity.





(a) (b)

Figure 1: (a) Typical Neuron behavior. Note: this figure describes a neuron observed intracellularly while our recording are extracellular, where peaks are on the order of 70-150 microvolts instead of millivolts. Note the peak's duration of 1-2 milliseconds. (b) Location of hippocampus in the brain. Source: en.wikipedia.org/wiki/Action_potential, img.medscape.com/news/2015/dt_150204_hippocampus.jpg

Input Data

The input data for this project comes from the lab of Dori Derdikman, of the Technion medical faculty, whose work focuses on the function of memory and spatial mapping in the Hippocampus region of the brain. One of the lab's focuses is the study of Place Cells, a particular set of neurons in the rat hippocampus. These neurons are active only when the rat is in a particular 2D coordinate in an enclosure, encompassing the mechanism by which the rat stores a location mentally. Recordings of these neurons are made by implanting 17micron thick electrode wires into the rat's hippocampus near the neurons and measuring the voltage. Voltage is then recorded while the rat navigates an enclosure and neural activity is correlated to the rat's location. The YouTube video linked shows the process. The recordings are made from 16 electrodes at a sample rate of 32 KHz. The recording equipment (Neuralynx – Cheetah) saves the data as a raw binary which we converted to 32bit unsigned integers using a Java script we wrote. After storing the data in an ascii, comma-separated-value format, the rest of the processing is done in Matlab.

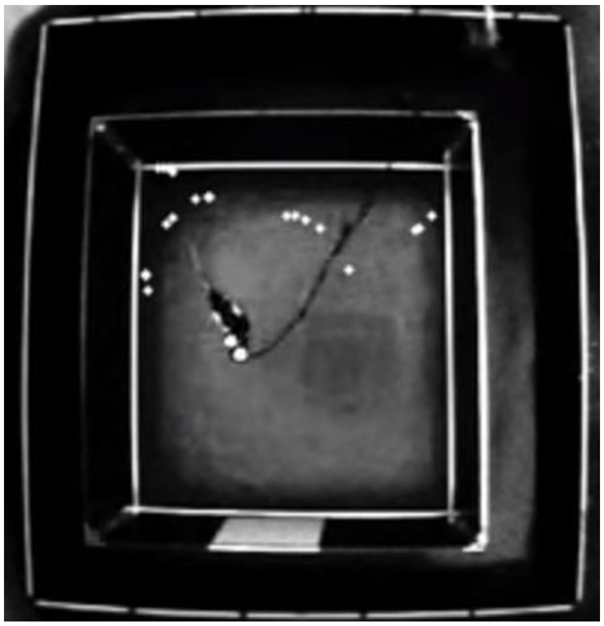


Figure 2: How recording is made. A still from a video of the neural recording process from the lab of Dori Derdikman, Technion medical faculty. Here a rat is seen moving in an enclosure with electrodes implanted in its hippocampus recording its neural activity. Particular neuron activity with respect to the rat's location are overlaid in the video. https://www.youtube.com/watch?v=i9GiLBXWAHI

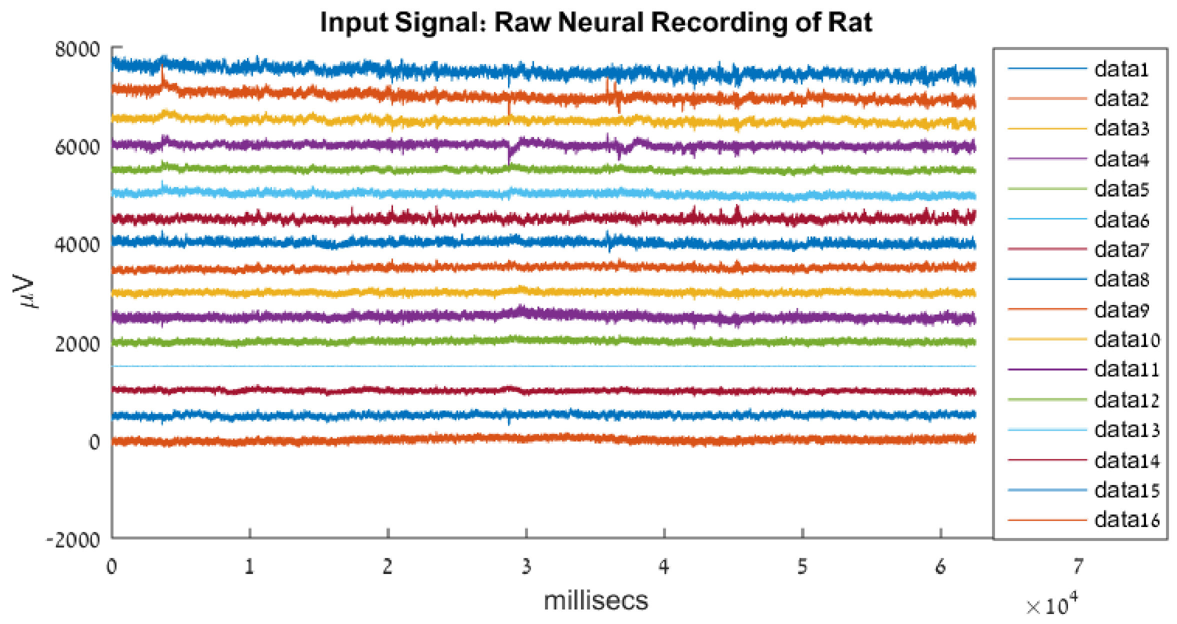


Figure 3: Input. 60 seconds of 16 channels of voltage recordings from the neurons of a rat hippocampus. To obtain this recording, preprocessing was done by writing a java script to read form a binary file produced by the recording equipment.

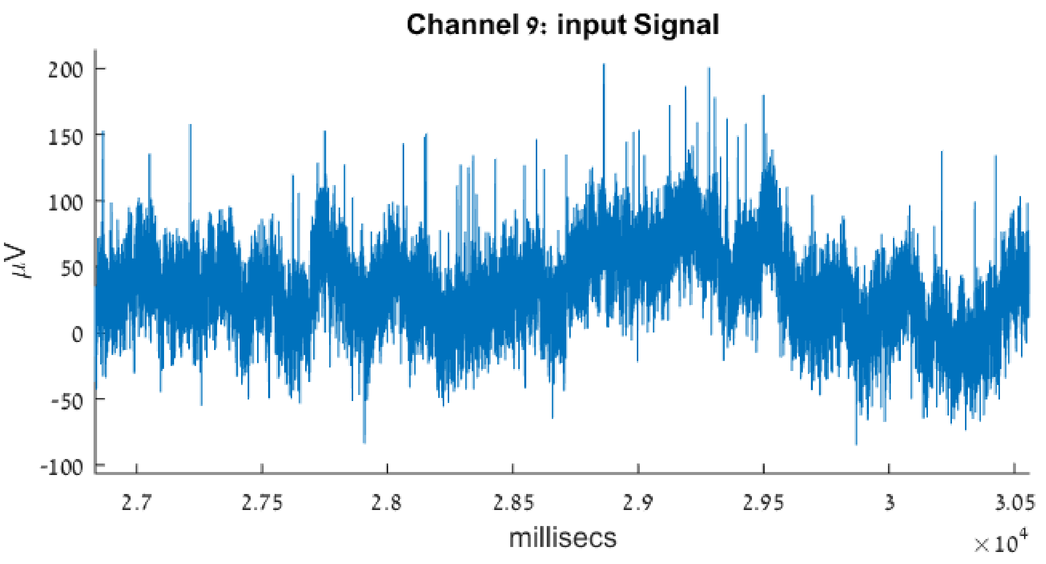


Figure 4: Input zoomed in. Another plot of the raw input zooming in to about 3 seconds of recording. Since a neuron action potential spike is on the order of 1-3ms with an extra cellular voltage of about 70-150 microvolts, it is clear that there is still a lot of unwanted noise in this signal.

Cleaning and Filtering

As described earlier, a typical extracellular recording of a neuron shows a voltage spikes of around 100 microvolts and about 1-3 milliseconds in length. Due to noise from other electro-chemical activity in the brain and errant electric activity from the recording process, cleaning and filtering is necessary. The two main methods used are manipulation in the Fourier domain and a High Pass Butterworth filter. The following section describes the Fourier process used. The High Pass filtering method is not thoroughly described as it was not discussed in the scope of the course.

Fourier Transform

The Fourier transform decomposes a function of time (signal) into the fundamental frequencies that make it up. The Fourier transform is a complex-valued function of frequency, whose absolute value represents the amount of that frequency present in the original function, the complex argument is the phase offset of the basic sinusoid in that frequency. Thus it is called the *frequency domain representation* of the original signal. The Fourier transform is not limited to functions of time only, but for a unified language, the domain of the original function is referred to as the *time domain*.
Linear operations performed on one domain have corresponding operation in the other domain, which are sometimes easier to perform. For example, the operation of differentiation in the dime domain corresponds to multiplication by the frequency, so some differential equations are easier to analyze in the frequency domain.

$$\underbrace{\varphi(x) = \dot{x}}_{Time\ domain} \rightarrow \underbrace{\hat{\varphi}(\xi) = s \cdot \xi}_{Frequency\ domain}$$

This is due to the convolution properties of the Fourier transformation. The convolution of two functions is defined as

$$\left(f(t) * g(t)\right) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$

In the Fourier transformation we get $F\left\{f(t)*g(t)\right\}=F\left\{f(t)\right\}\cdot F\left\{g(t)\right\}$

Proof:

$$F\left\{f(t)*g(t)\right\}=F\left\{\int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau\right\}=\int_{-\infty}^{\infty} f(\tau)F\left\{g(t-\tau)\right\}d\tau$$

$$=\int_{-\infty}^{\infty} f(\tau)\cdot e^{-2\pi i\xi t}F\left\{g(t)\right\}d\tau=F\left\{g(t)\right\}\cdot\int_{-\infty}^{\infty} f(\tau)\cdot e^{-2\pi i\xi t}d\tau$$

$$=F\left\{f(t)\right\}\cdot F\left\{g(t)\right\}$$

Functions that are localized in the time domain have Fourier transforms that spread out across the frequency domain and vice versa, a phenomenon known as the uncertainty principle. The critical case of this principal is the Gaussian function. The Fourier transform of a Gaussian function is another Gaussian function. The Fourier transform is defined as

$$\hat{\varphi}(\xi)=\int_{-\infty}^{\infty}\varphi(t)e^{-2\pi it\xi}dt \quad ; \quad \forall\xi\in\Box$$

The inverse transform is defined as

$$\varphi(t)=\int_{-\infty}^{\infty}\hat{\varphi}(\xi)e^{2\pi i\xi t}d\xi \quad ; \quad \forall t\in\Box$$

A sufficient, but not necessary, condition for existence of the transform-domain representation is that the time domain function is absolutely inferable $\int_{-\infty}^{\infty}\left|\varphi(t)\right|dt<\infty$.

Properties of the Fourier transform:

- Linearity – Given any $a,b\in\Box$, if $h(t)=a\cdot f(t)+b\cdot g(t)$ then the Fourier transformation will be $\hat{h}(\xi)=a\cdot\hat{f}(\xi)+b\cdot\hat{g}(\xi)$.
- Time-Shifting – Given any $t_0\in\Box$, if $h(t)=f(t-t_0)$ then the Fourier transformation will be $\hat{h}(\xi)=e^{-2\pi it_0\xi}\hat{f}(\xi)$.
- Frequency-Shifting – Given any $\xi_0\in\Box$, if $h(t)=e^{2\pi it\xi_0}f(t)$ then the Fourier transformation will be $\hat{h}(\xi)=\hat{f}(\xi-\xi_0)$.
- Time Scaling – Given any $a\in\Box$, $a\neq 0$, if $h(t)=f(a\cdot t)$ then the Fourier transformation will be $\hat{h}(\xi)=\frac{1}{|a|}\hat{f}\left(\frac{\xi}{a}\right)$.
- Conjugation – If $h(t)=f^*(t)$ then the Fourier transformation will be $\hat{h}(\xi)=\hat{f}^*(-\xi)$. If f is real then we get the *reality condition* $\hat{f}(-\xi)=\hat{f}^*(\xi)$, that is that f is a *Hermitian function*. If f is purely complex then $\hat{f}(-\xi)=-\hat{f}^*(\xi)$.
- Integration – Substituting $\xi=0$ in the definition we obtain $\hat{f}(0)=\int_{-\infty}^{\infty}f(t)dt$.
- Time-Frequency Duality – If $F\left\{f(t)\right\}=\hat{f}(\xi)$ then $F\left\{\hat{f}(t)\right\}=f(-\xi)$.

Proof:

$$f(t)=\int_{-\infty}^{\infty}\hat{f}(\xi)e^{2\pi i\xi t}d\xi \quad ; \quad f(-t)=\int_{-\infty}^{\infty}\hat{f}(\xi)e^{-2\pi i\xi t}d\xi$$

0

$$\Rightarrow \quad f(-\xi)=\int_{-\infty}^{\infty}\hat{f}(t)e^{-2\pi i\xi t}dt=F\left\{\hat{f}(t)\right\}$$

Reference:
- Course tutorial slides and class notes
- Wikipedia https://en.wikipedia.org/wiki/Fourier_transform

Using Fourier to Clean Data

Since the neural recording constitutes a continuous function localized in the time domain, the Fourier transform is useful in both analyzing and cleaning the signal. Firstly we plotted our signal in the frequency domain:

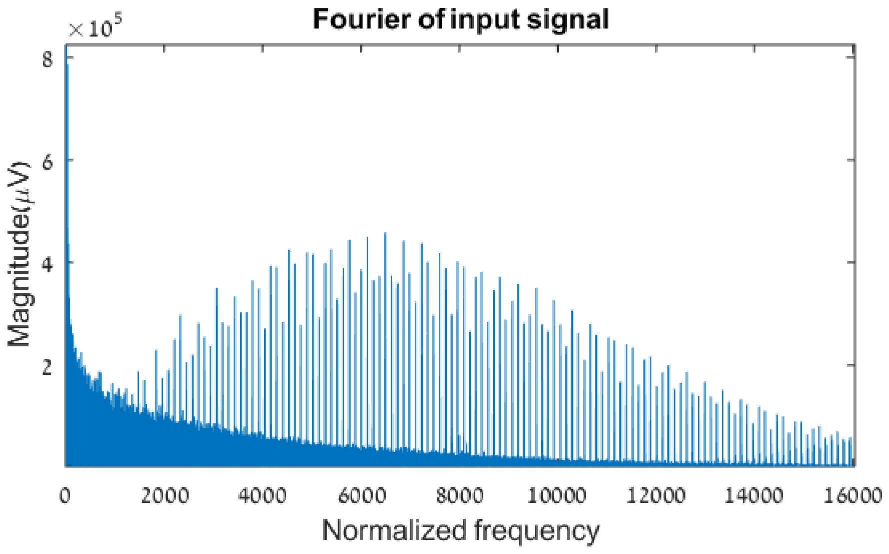


Figure 5: Signal in frequency domain after Fourier.

When viewing the signal in the frequency domain, it is clear that lot of harmonics are present in the signal. Harmonics occur when a frequency is an integer multiple of another frequency causing an additive effect in the energy of the source signals, this is common in electronic circuitry. Observing this in a biological signal is usually considered noise generated in the recording process. To remove these harmonics, we wrote code to take the average value for every 100 samples (3ms), and look for spikes more than 3 times the mean. The following is the result after changing those values to the mean.

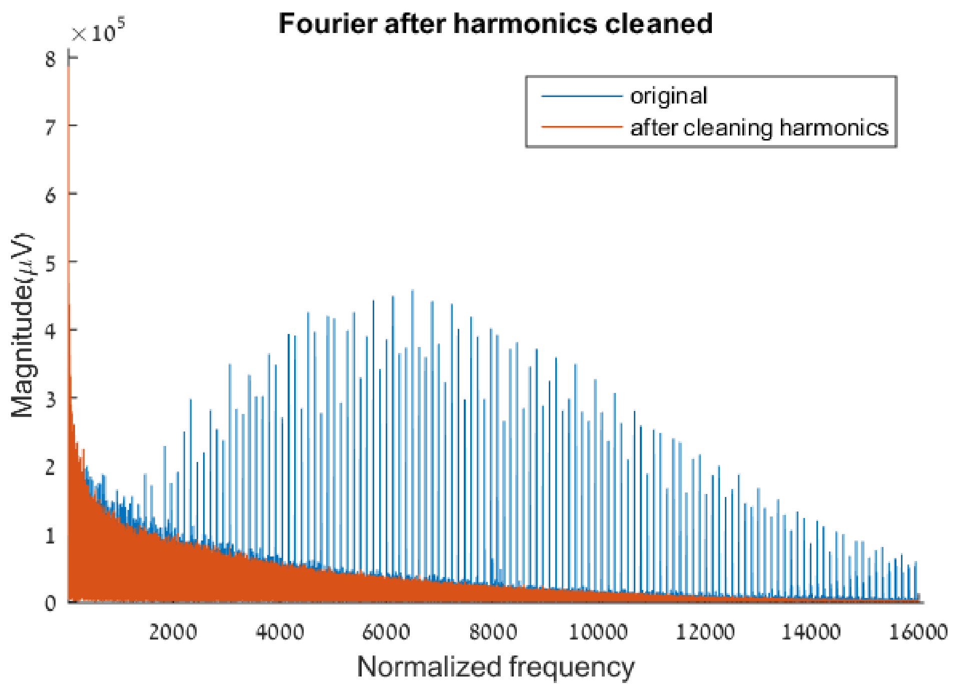


Figure 6: Fourier of signal after removing harmonics.

From the plot, it appears that some information was is lost in the cleaning process, however, the amount of loss is minimal compared to the noise energy removed.

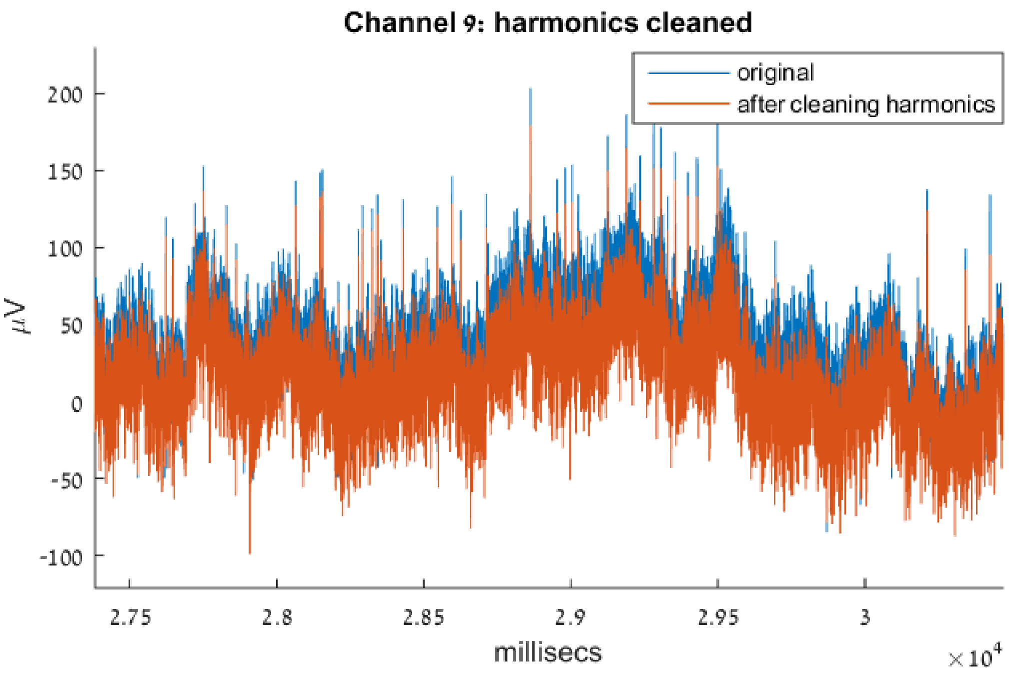


Figure 7: Single channel after harmonics cleaned.

Although removing the harmonics cleans up some errant energy in the signal, there remains a significant amount of unwanted oscillation, requiring further filtering.

High Pass Filtering

In order to be able to observe neuronal activity in the range of events on the order 1-3 ms, further cleaning of the signal is required. To accomplish this, a High Pass filter, specifically a Butterworth filter, is utilized. This technique is described briefly as it is outside the scope of this course. Essentially a high pass filter allows higher frequencies beyond a threshold to remain, while attenuating all frequencies below. In this case, we wanted to attenuate frequencies below 600 Hz because those frequencies in the signal directly interfered with voltage crests 0.8ms and longer, the range in which neuronal activity occurs. This also removes typical electronics oscillation in the 50 Hz range. The energy of these frequencies is high as seen in the original Fourier plot of the signal. We chose an implementation known as a Butterworth filter, which is characterized by a magnitude response that is maximally flat in the passband. To make sure that filtering was effective we plotted the Fourier of the signal after the high pass filtering.

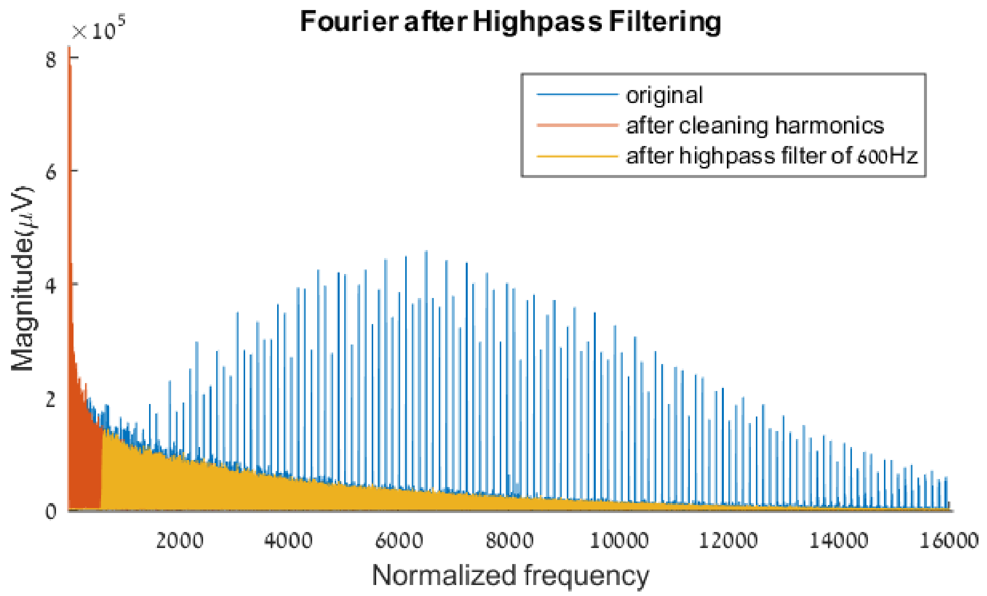


Figure 8: High Passed filter in Fourier domain. Looking at the Fourier of the signal it is clear that the high pass filter is functioning properly, minimal frequency in the signal is in the range of 0-600Hz.

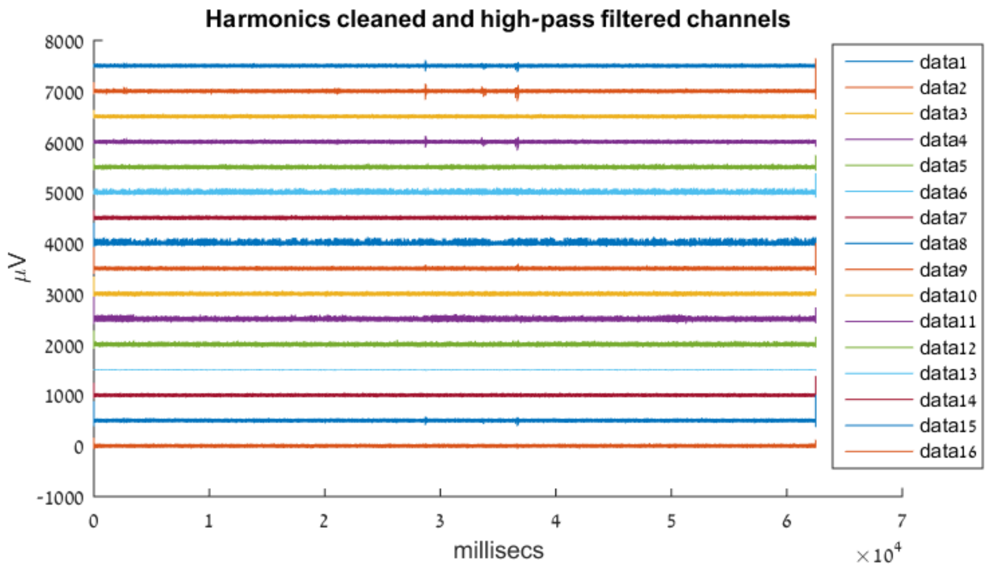


Figure 9: All channels after harmonics cleaning and high pass filter. The amount of noise and oscillation is significantly lower compared to figure 2 of the original input.

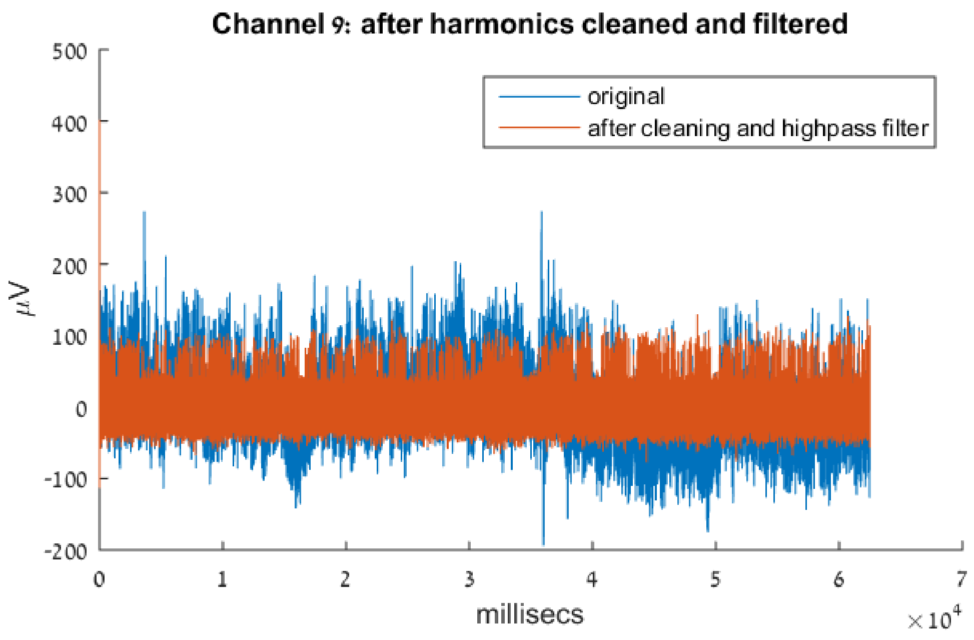


Figure 10: Channel 9 before and after harmonics cleaning and high pass filtering. Again it is clear the amount of noise and oscillation is significantly lower than the original input.

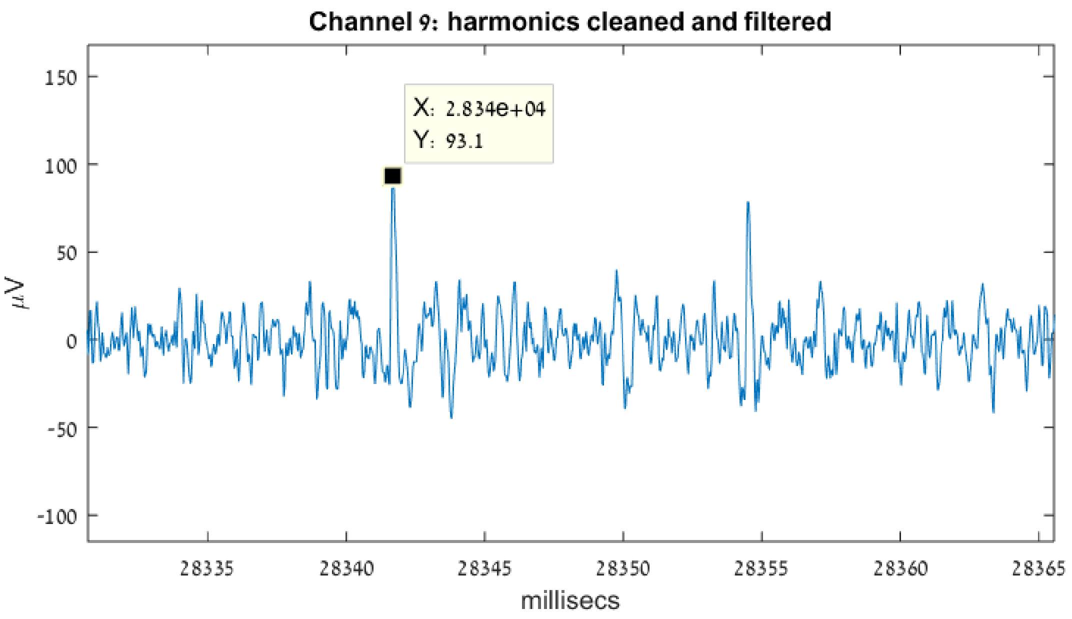


Figure 11: Channel 9 zoomed in after harmonics cleaning and high pass filtering. These peaks around 100 microvolts which are now visible are a clear indication of neuronal activity. This is verified by the lab of Dori Derdikman, Technion medical faculty.

Sampling and Compressing the Signal

Given the 32 KHz resolution of the recording process across all 16 channels, 1 second of recording generates 0.5 million 32bit integer. Our objective of analyzing 60 seconds consisted of more than 30 million samples. In order to efficiently analyze this data compression is necessary. For this we use PCA, specifically the SVD method. Prior to this however, it was necessary to create a data matrix of sample events to compress.

Sampling from Neural Recording

Before compression, we first create a collection of relevant samples of neuronal activity from our filtered channels. This is done by setting a voltage threshold for suspected neural activity, in our case 70 microvolts. Using the filtered channels as input, all samples on any channel surpassing the threshold, the entire set of values across all channels for 1ms before, to 2ms after are saved. This corresponds to the 3ms upper bound of the typical neuron action potential time. Saving data across all channels is done to see correlation between channels during the clustering phase. In order to avoid saving a specific peak multiple time, a 1ms break between peaks per channel is enforced. Since neurons have a refractory period of around 10ms this is sensible, although if the channel happens to be picking up signals from multiple sources relevant activity could be missed. When this process is complete we for this dataset, we end with a data matrix that is 1536x1603. 1603 samples x (3ms@32KHZ=96) x 16 channels in length.

PCA Analysis and Compression

Principal Component Analysis uses an orthogonal transformation to convert a set of sample signals of possibly correlated variables into a set of values of linearly uncorrelated variables, called principal components. The transformation is defined so the first principal component has the largest possible variance, accounting for as much of the variability in the data as possible. Each succeeding component has the highest variance possible after the previous components. Because PCA maps information this way, it is highly useful in data analysis and compression. Specifically it allows quantization of data to a minimal set of coefficients to achieve a specific error bound. PCA can be done by eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix, which allows for non-square matrices and is considered the standard method.

SVD Singular Value Decomposition

SVD is a factorization of a real or complex matrix. Any $m \times n$ matrix Γ can be decomposed to the form $\Gamma = U\Lambda^{1/2}V^*$, where U is an $m \times m$ unitary matrix, $\Lambda^{1/2}$ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V^* is the conjugate transpose of an $n \times n$ unitary matrix. Since U and V are unitary matrices, the columns of each of them form a set of orthonormal vectors, which can be regarded as basis vectors, this is also true for their conjugate transpose matrices U^* and V^* respectively. The diagonal entries $\lambda_i^{1/2}$ are known as the singular values of Γ. The m columns of U and the n columns of V are called left-singular vectors and right singular vectors of Γ, respectively. The left-singular vectors of $\Lambda^{1/2}$ are the orthogonal eigenvectors set of $\Gamma\Gamma^*$ and the left-singular vectors are the orthogonal eigenvectors set of $\Gamma^*\Gamma$. An interesting property of λ_i is that they are the non-zeros eigenvalues of both $\Gamma\Gamma^*$ and $\Gamma^*\Gamma$.
The proof that $\Gamma\Gamma^*$ and $\Gamma^*\Gamma$ have the same eigenvalues is very simple, we know that $\Gamma\Gamma^* = U\Lambda U^*$, and in an index representation it is shown as $\Gamma\Gamma^* u_i = \lambda_i u_i$. Multiplying from the left with Γ^* gives us $\Gamma^*\Gamma\Gamma^* u_i = \Gamma^* \lambda_i u_i$. Defining $\tilde{v}_i = \Gamma^* u_i$ we can write the expression as $\Gamma^*\Gamma\tilde{v}_i = \lambda_i \tilde{v}_i$. This means that λ_i

is also an eigenvalue of $\Gamma^*\Gamma$. This results gives us also the knowledge that V can be determined by U. Normalizing $\tilde{v}_i$ will give the corresponding member of V.

$$v_i = \frac{\tilde{v}_i}{\|\tilde{v}_i\|_2}$$

$$\|\tilde{v}_i\|_2^2 = \left(\Gamma^* u_i\right)^* \left(\Gamma^* u_i\right) = u_i^* \Gamma\Gamma^* u_i = u_i^* \lambda_i u_i = \lambda_i \|u_i\|_2^2 = \lambda_i$$

$$\Rightarrow v_i = \frac{1}{\lambda_i^{1/2}} \Gamma^* u_i$$

The matrix $\Lambda^{1/2}$ can be also represented in a compact form. Let's say that there are r nonzero on the diagonal of $\Lambda^{1/2}$. The compact form will be $\Gamma = U_r \Lambda_r^{1/2} V_r^*$, where now U_r is an $m \times r$ matrix and V_r is an $n \times r$ matrix. The compact form can be written also in index form, $\Gamma = \sum_{i=1}^{r} \lambda_i^{1/2} u_i v_i^T$ where u_i and v_i are the i^{th} column of U and V respectively. Each term is a rank-one matrix. This means that the k-approximation of Γ is given by

$$\hat{\Gamma}_k = \sum_{i=1}^{k} \lambda_i^{1/2} u_i v_i^T \quad ; \quad k \le r$$

This k-approximation is the best approximation in a squared-error sense for a given Γ.
References:
- Course tutorial slides
- Wikipedia, https://en.wikipedia.org/wiki/Singular_value_decomposition

SVD for Compression

After forming our data matrix as described in the previous section, the SVD k-approximation method described above is used to compress the data. To decide how many u vectors to use for compression, we consider the following.

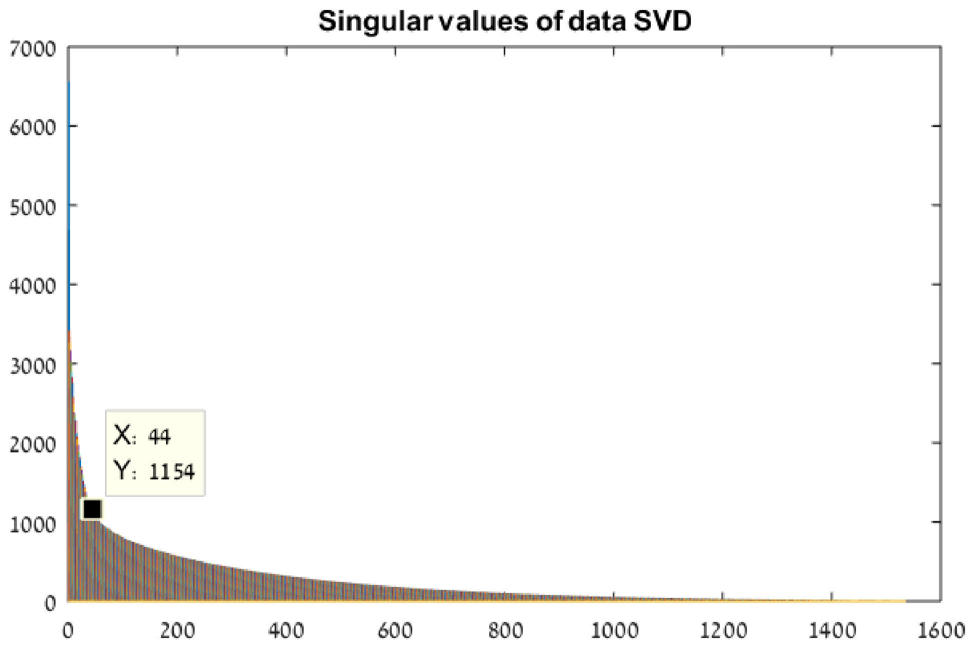


Figure 12: $\Lambda^{1/2}$ **values from SVD.** From the following plot, it seems that around 40th value there is a drop in information stored per value.

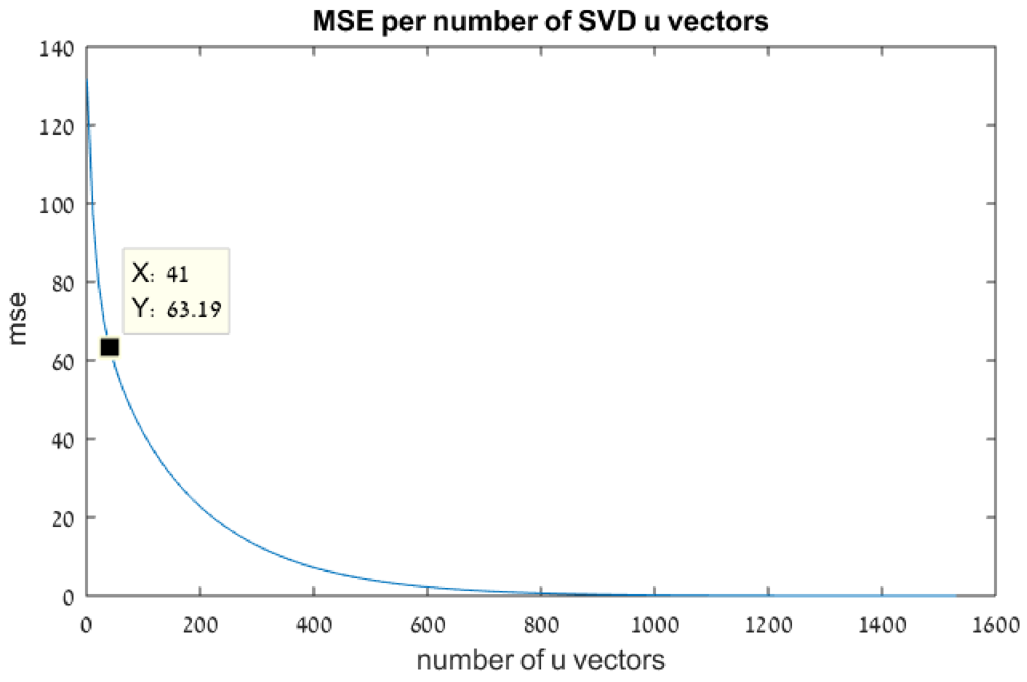


Figure 13: MSE vs number of U vectors used to compress signal. Although 40 seems relatively high in MSE, experimentally it appeared to function well during the clustering phase.

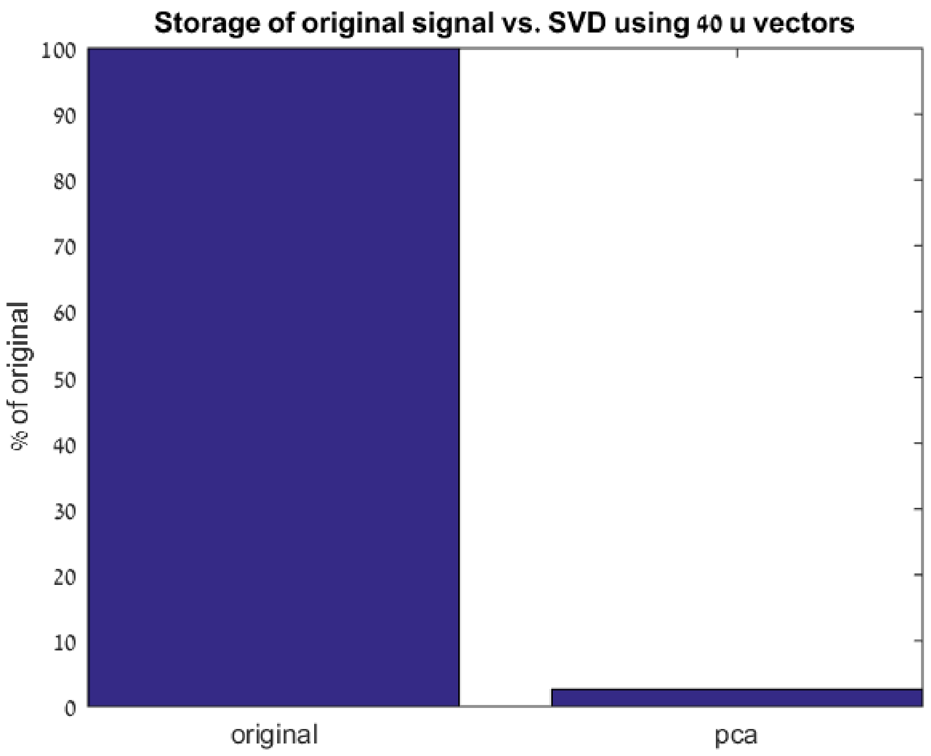


Figure 14: Storage space before and after SVD. After utilizing the SVD the size of our data matrix reduced to less than 5% of its original size.

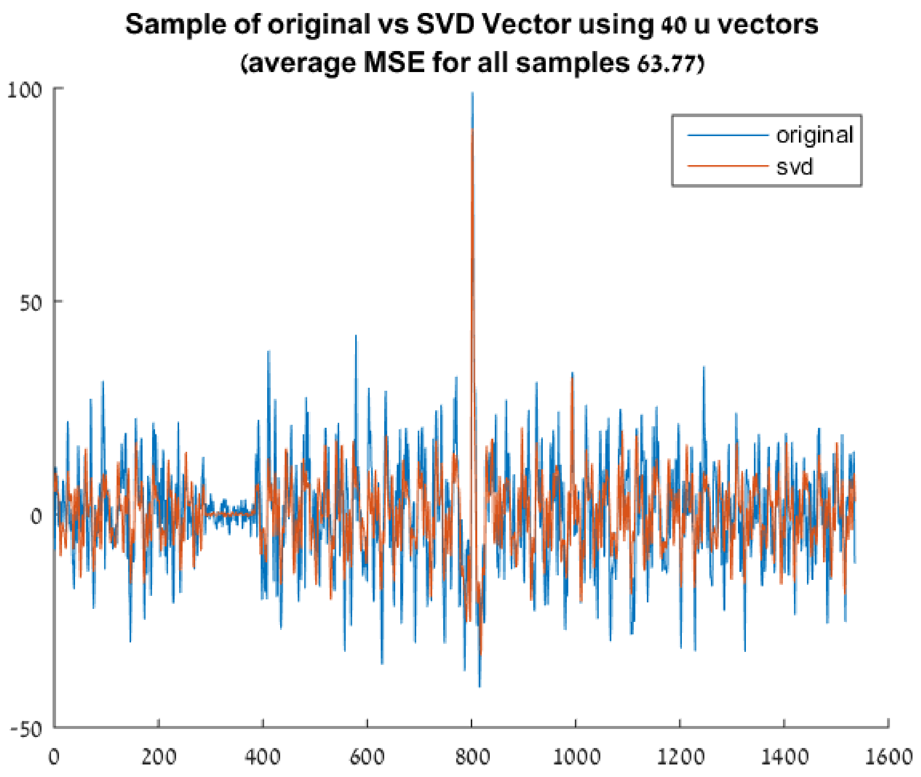


Figure 15: Sample before and after compression. The sample before and after (using 40 u vectors out of 1603). It is apparent some information is lost, but the overall shape of the signal is quite similar. The average MSE for the samples is 63.77 after compression at this u.

Clustering

Finding neural activity in a multi-channel cell recording involves more than identifying voltage spikes. The exact location of each wire in the electrode array used for recording with respect to the nearby neurons' locations is unknown. A neuron between two electrodes may show up as two smaller spikes on different channels as opposed to a large spike on a single channel. For this reason, clustering of the samples across *all* channels is utilized to identify neurons. For clustering in this project, the K-means method is used.

K-means clustering

k -means clustering is a method of vector quantization. It aims to partition n observations into k clusters; each observation belongs to the cluster with the nearest mean. Given a set of observations $(x_1, x_2, ..., x_n)$ where each observation is a m -dimensional real vector. k -means clustering aims to partition the n observations into k sets $\mathbf{S} = \{S_1, S_2, ..., S_k\}$ so as to minimize the within-cluster sum of squares (WCSS). Its objective is to find

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x}\in S_i} \left\| \mathbf{x} - \mu_i \right\|^2$$

Where μ_i is the mean of points in S_i. The most common algorithm uses an iterative refinement technique. Given an initial set of k means $m_1^{(1)},\dots,m_k^{(1)}$, the algorithm proceeds by alternating between two steps:

- Assignment Step: Assigns each observation to the cluster whose mean yields the least within-cluster sum of squares (WCSS). Since the sum of squares is the squared Euclidean distance, this is also the 'nearest' mean.

  $$S_i^{(t)} = \left\{ x_p : \left\| x_p - m_i^{(t)} \right\|^2 \le \left\| x_p - m_j^{(t)} \right\|^2 \ \forall j, 1 \le j \le k \right\}$$

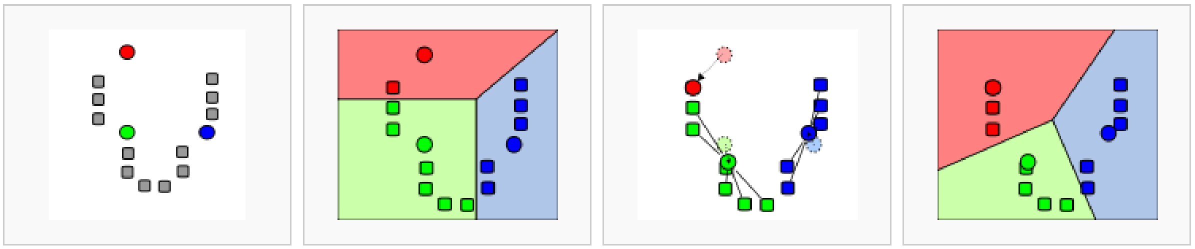
  Where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

- Update Step: Calculate the new means to be the centroids of the observation in the new clusters.

  $$m_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{x_j \in S_i^{(t)}} x_j$$

Since the arithmetic mean is a least-squares estimator, this also minimizes the WCSS objective. The algorithm converges when the assignments change no more. Since both steps optimize the WCSS objective, and there only exists a finite number of such partitioning, the algorithm must converge to a local optimum. There is no guarantee to find a global optimum by using this algorithm. Changing the distance function might stop the algorithm from converging.

The common way to initialize the algorithm is by random partitioning. This chooses k random observation from the date set and uses these as the initial means. Than it randomly assigns a cluster to each observation and proceeds to the update step, by that computing the initial mean to be the centroid of the randomly chosen cluster.



1. *k* initial "means" (in this case *k*=3) are randomly generated within the data domain (shown in color).

2. *k* clusters are created by associating every observation with the nearest mean. The partitions here represent the Voronoi diagram generated by the means.

3. The centroid of each of the *k* clusters becomes the new mean.

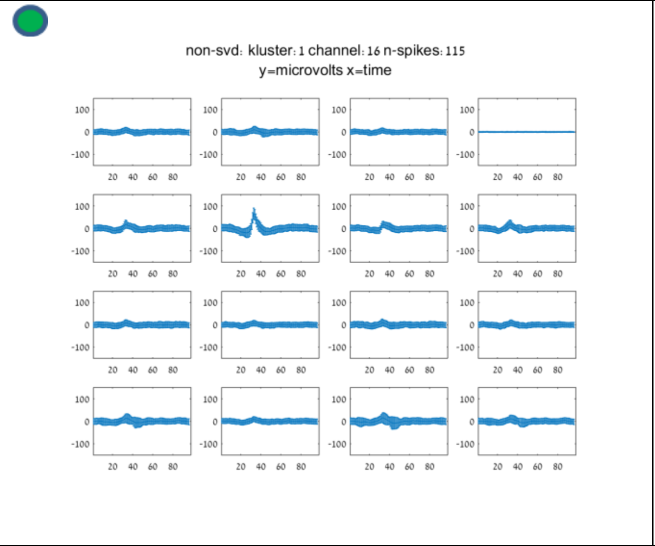
4. Steps 2 and 3 are repeated until convergence has been reached.

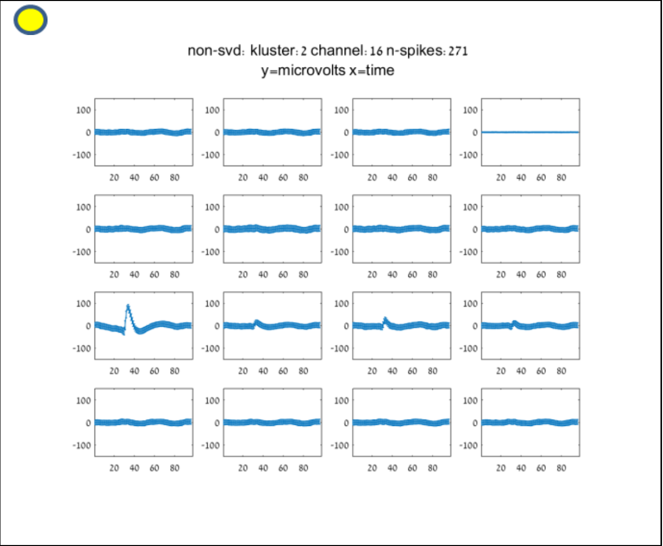
Figure 16: The K-Means algorithm visualized.

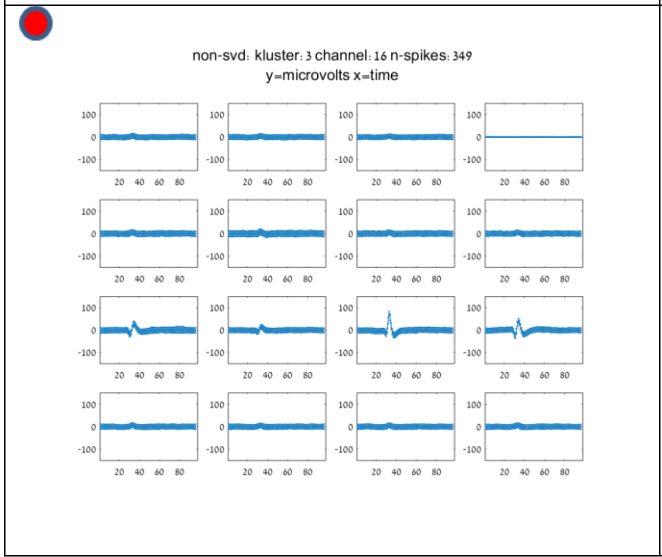
References:
- Wikipedia, https://en.wikipedia.org/wiki/K-means_clustering
- Matlab documentation, http://www.mathworks.com/help/stats/kmeans.html

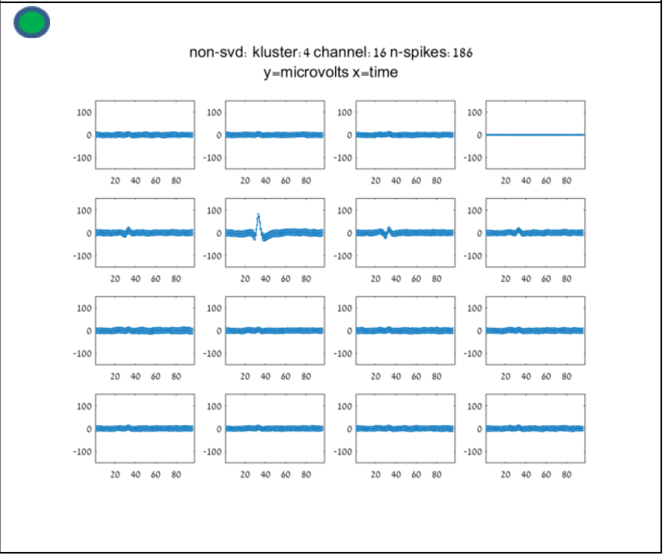
K-Means on Signal

After cleaning harmonics, filtering and extracting sample spikes from the full original signal, K-means is run and set to generate 6 clusters.









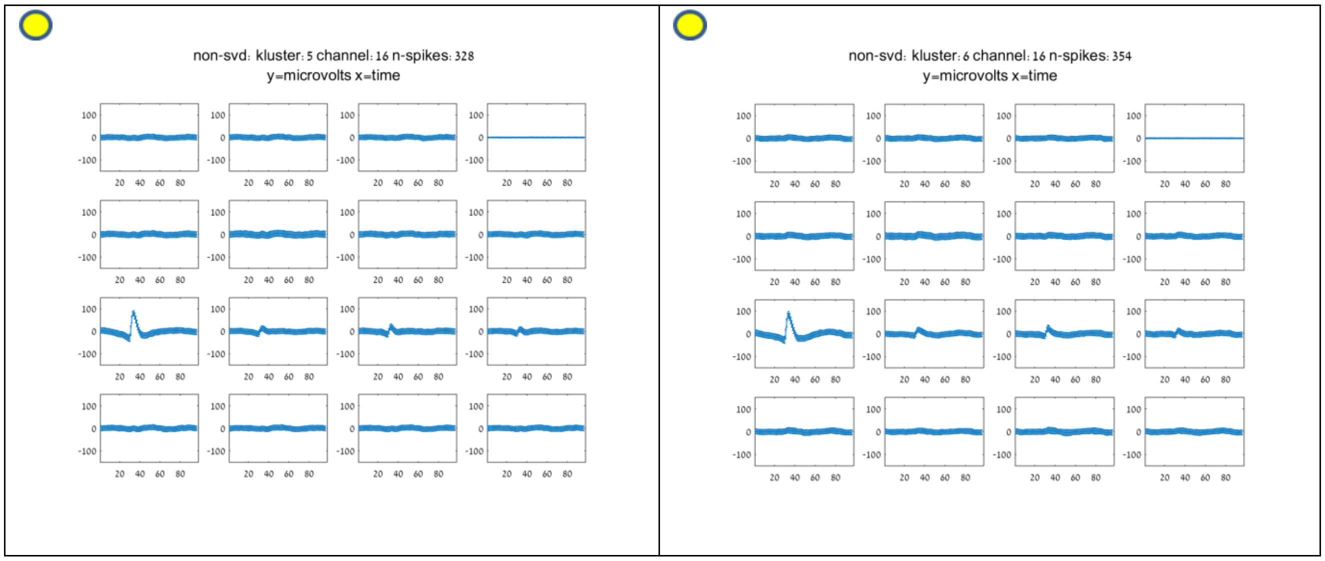


Figure 17: K-Means on filtered signal. The six clusters produced by k-means.

Clustering is done on 16 channels *simultaneously* per sample. Each cluster shows the mean value of the signal, with std error bar, per channel. Also included is the number of samples per cluster, x axis is time index in frequency domain (32KHz), y axis is microvolts. Colors indicate clusters that are similar to each other. Although K-means was initialized to find 6 clusters, visually it appears that some clusters are very similar, so that possibly the data is better divided into 3 unique clusters. These are shown by the color coding in figure 17.

Clustering after PCA compression

While this clustering on the uncompressed signal does successfully produce results, it is computationally expensive and would not be scalable for full neuron recordings which are 20 times larger than this dataset. Consequently k-means is run on the compressed data matrix which is, 40x1603, about 5% of the original size. The vectors are then converted back into the original base for plotting. Clustering on the compressed signal is much faster than the uncompressed signal.

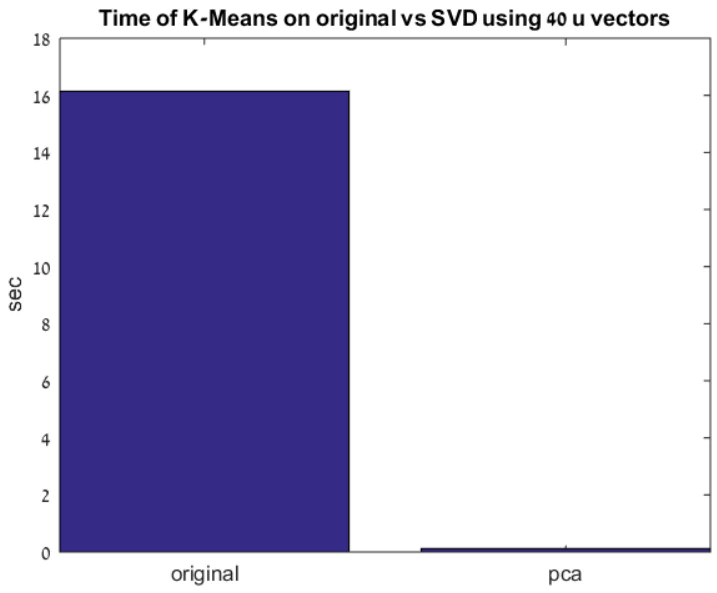


Figure 18: Timing of K-means process on original vs after SVD. Compression using 40 out of the 1603 u vectors, time is significantly lower.

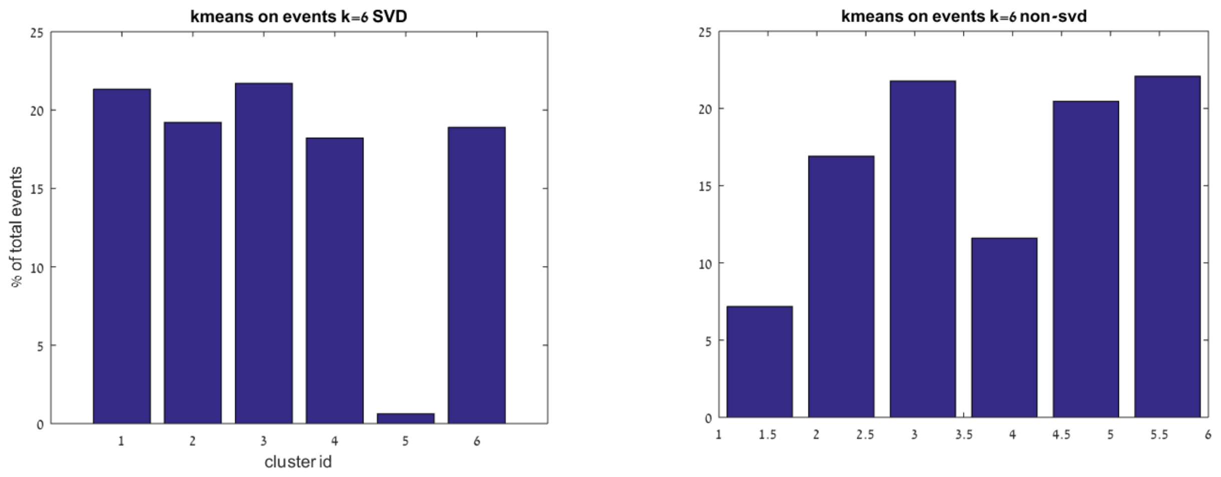
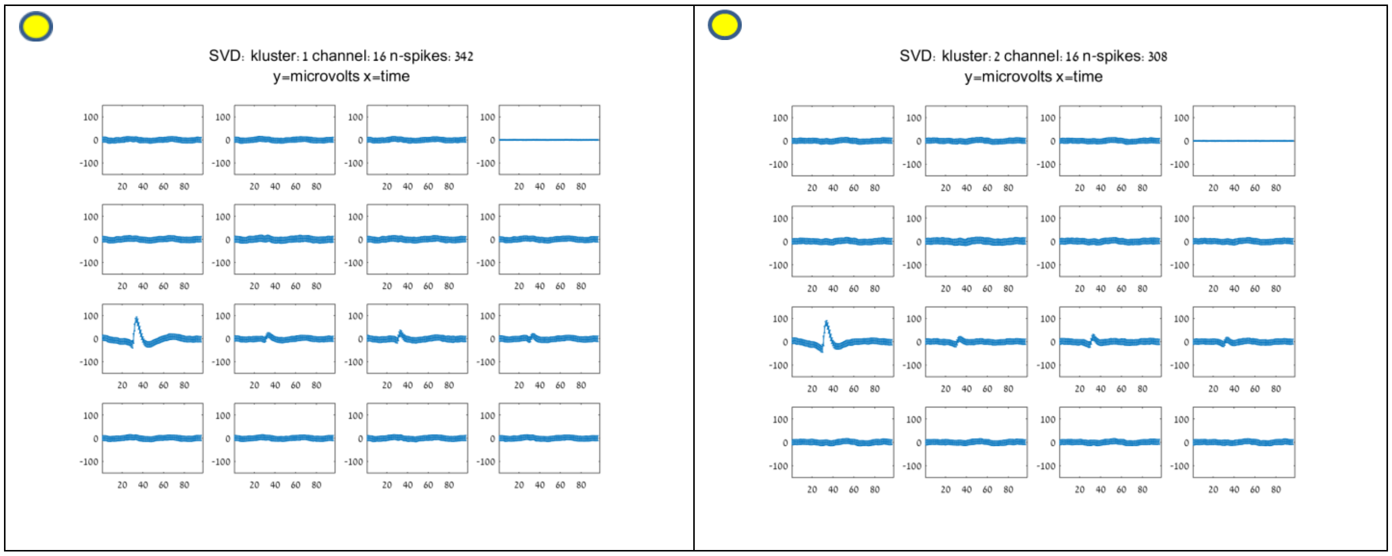


Figure 19: Clustering sample distribution on SVD compressed vs original. Clustering on the compressed data showed 5 distinct clusters as opposed to 6 on the uncompressed. Whether this is an indication that the compression is too lossy is a question we attempted to answer.



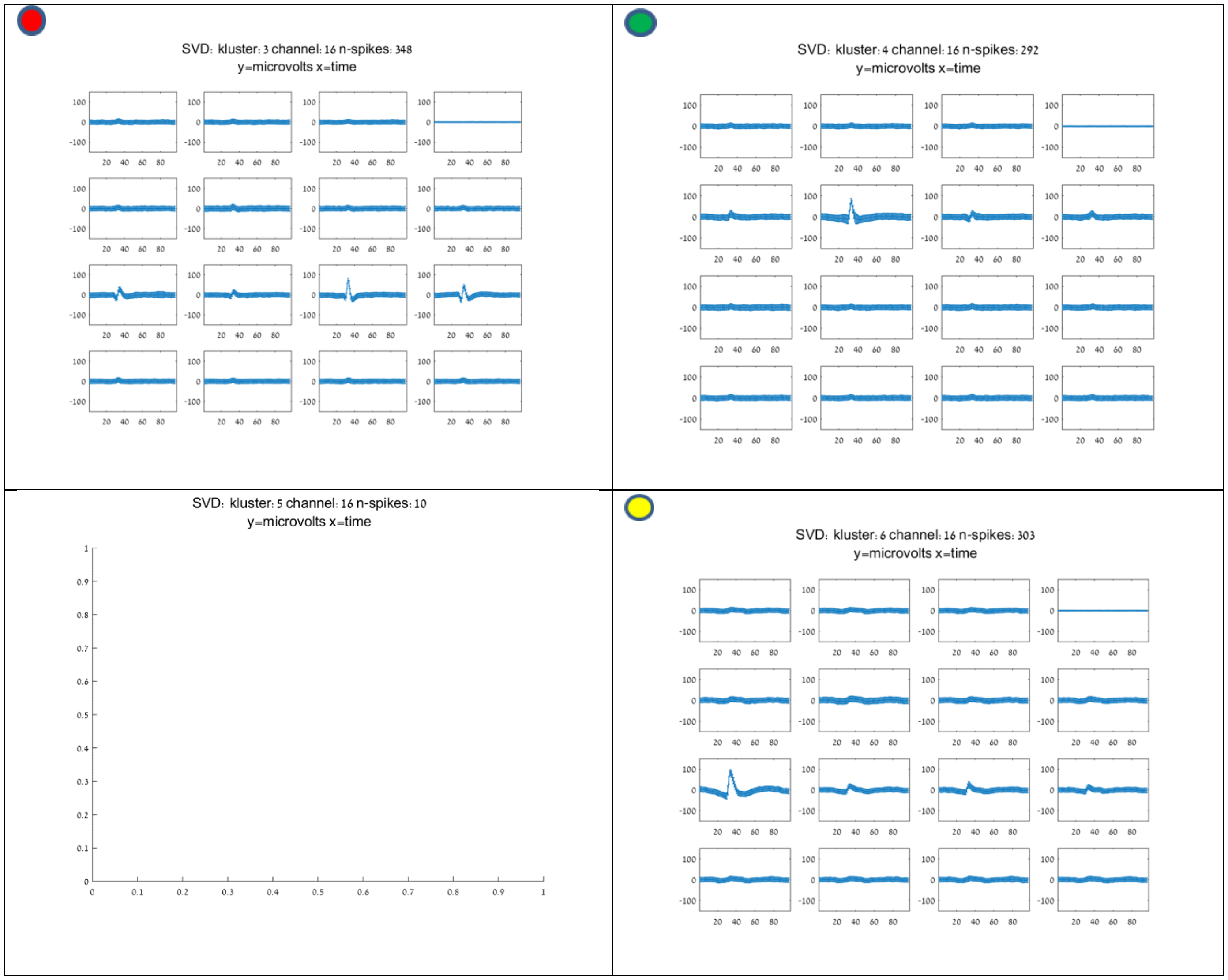


Figure 20: Clustering on SVD compressed signal. Note that cluster 5 is not included because it only comprises of 10 samples making it statistically insignificant.

Comparing these clusters to the uncompressed clusters in figure 19, the clusters appear to be very visually similar as shown by the color coding. Both appear to show approximately 3 unique clusters and in both cases the shape of the signals in those clusters are very similar. This finding is important because it is a sign that the SVD process does not appear to lower the integrity of the clustering significantly.

Conclusion

After cleaning, filtering, sampling, compressing, and clustering the data the results appear to show 3 unique clusters of signals, or three unique neuron signatures as we have interpreted it. Dori Derdikman's lab analysis shows 4 neurons, however they analyzed the full 31.25 minutes of recording while this analysis was done only over 60 seconds. It is very possible that the fourth neuron would be present if we analyzed the entire dataset.

Although we are able to more or less identify the same neurons as Dori's lab, the motivation of the project was to see if it was possible to try to discern more in the signal than what is already being identified. It is possible that the two clusters that appear to look similar in figure 17, and clustering boundaries in general, do actually signify distinct and brain activity. This potential is hard to investigate and needs to be done by association of cluster activity to the rat's behavior. That being said, being able to come up with the same results as the current methods is a good first step.

For the future it would be desirable to modify our methods to be able to process a full 30 minute recording, 1 billion elements instead of the 30 million done now (30x). Additionally, refining the clustering process to find the ideal number of clusters is a very interesting area to explore. Especially automation via hierarchical clustering, as well as other clustering algorithms. Perhaps most interesting would be to do more analysis into clusters that do not initially appear to be of interest, to see if perhaps they in fact signify undescribed brain activity. This is the ultimate goal.