Clustering Hippocampal Neuron Action Potentials Using Autoencoders and Autoencoder-Kalman Filtering for Noise Reduction

A Major Qualifying Project submitted to the faculty at
Worcester Polytechnic Institute
in partial fulfilment of the requirements for the
Degree of Bachelor of Science

By:
Imogen Cleaver-Stigum
Erik Reimert Burro

April 6th, 2021

Therese M Smith, PhD, Project Advisor
Assistant Teaching Professor
Department of Computer Science, WPI

Abstract

In this project, we applied the deep learning methods of autoencoder-Kalman filtering as well as the autoencoder preprocessing from Ciecierski [3] to improve clustering on action potentials by filtering noise. We used multiple types of clustering, including k-means clustering, mean-shift clustering, and agglomerative hierarchical clustering. We evaluated the performance of each clustering algorithm after using each filtering method, as well as no filtering, to analyze which of the filtering methods has the best effect on clustering action potentials.

Acknowledgements

We would like to thank the following individuals for their help over the course of this project:

- Professor Therese M Smith, for advising our project and connecting us with other invaluable people and resources;
- Konrad Ciecierski, for letting us conduct a replication study on his paper “Neural Spike Sorting Using Unsupervised Adversarial Learning” [3], including by providing a sample of the data he used for training and by giving us tips and reference material for implementation; and
- Professor Matthew Liam Weiss, Professor Randy Clinton Paffenroth, and Joshua R. Uzarski for letting us work from their study “The Autoencoder-Kalman Filter: Theory and Practice” [21] and sharing with us their implementation.

Table of contents

Table of Figures

Table of Tables

Executive Summary

In computational neuroscience, a useful task is clustering neuron action potentials. This can allow researchers to determine whether the action potentials have been affected by myelination, which is caused by some diseases and can slow down action potentials. action potentials are very noisy signals, and they are affected by several types of noise, including Gaussian noise, additive noise, and multiplicative noise. Reducing the noise in the signals should make the clustering more effective. There have been multiple types of methods that aim to reduce the noise in these signals. One such method is described in the paper *Neural Spike Sorting Using Unsupervised Adversarial Learning* by Konrad Ciecierski [3]. This method uses autoencoders with a custom loss function that is specific to action potential signals to filter the noise in the data before clustering. Another such method is the autoencoder-Kalman filter (AEKF), which is comprised of a Kalman filter to filter the noise, in between the encoder and decoder layers of the autoencoder neural network [21]. However, before this paper, there had not yet been any research done on which of these two methods results in better clustering performance.

In this project, we applied autoencoder-Kalman filtering [21] preprocessing as well as the autoencoder preprocessing method from [3], attempting to improve clustering on hippocampal action potentials by filtering noise. We used multiple types of clustering, including k-means clustering, mean-shift clustering, and agglomerative hierarchical clustering. We evaluated the performance of each clustering algorithm after using each filtering method, as well as no

filtering, to analyze which of the filtering methods has the best effect on clustering action potentials. After performing both types of noise filtering, as well as a control group with no filtering, on the data, we used the clustering algorithms on the data and evaluated the performance of each. Our results showed that the autoencoder method described in Cieciersi's paper [3] resulted in better clustering performance measures for most of the algorithms than the AEKF and no clustering, and the AEKFgenerally performed better than no filtering.

Chapter 1: Problem Description

We have reason to believe that analysis of action potentials through clustering could show whether the individual is suffering from a disease that might be affecting the neuron performance (Therese M. Smith, Personal Communication, 16 September 2020). The action potentials in myelinated neurons are slowed down by demyelination caused by diseases like multiple sclerosis. For this reason, performing clustering on action potentials is a useful task. The recent papers by Weiss and Paffenroth [22] and Ciecierski [3] deal with clustering neuron action potentials using different methods for noise filtering and clustering. Weiss and Paffenroth use an Autoencoder-Kalman filter, and Ciecierski uses unsupervised adversarial learning, also using autoencoders. However, as far as we know, the computational neuroscience community has yet to do research on which of these methods performs best for clustering action potentials.

To verify this research gap, we used a database of academic journals to find all the journals that might have relevant papers, including *Brain Informatics, Brain-Computer Interfaces, Computational Intelligence and Neuroscience, i-Perception, Network Neuroscience, Neural Network World, Neural Networks, Neurocomputing, and Neuroinformatics.* In each of these, we used various keywords such as “clustering,” “artificial intelligence,” “deep learning,” “Kalman filter,” et cetera to try to find some work that had already combined these two papers. We did not find any such work in this search. Another reason we believe that this research gap exists because the two papers are very recent.

We are aiming to use autoencoder-Kalman noise filtering and clustering on hippocampus action potentials as well as Ciecierski’s method of unsupervised adversarial learning using

autoencoders to determine which of these methods is more successful at preparing action potentials to be clustered. The noise filtering in combination with the machine learning will serve to reduce the dimensionality of the data so it can more easily be clustered. We will perform the clustering on real data collected from hippocampi. This will reveal which of the two methods performs best, which will help in future research concerning clustering action potentials to find out whether they are affected by demyelination from diseases.

Chapter 2: Related Work

In this chapter, we will discuss the background information about computational neuroscience, clustering, and noise filtering that this project is building upon.

2.1 Prior Research on Brain-Computer Interfaces

In the field of computational neuroscience, research on brain-computer interfaces (BCI's) in the past has shown the practical applications of BCI's, as well as what the current technology is capable of. For example, BCI's have been used for patients with nervous system injuries or diseases that affect the nervous system [2]. This is especially relevant when the BCI's are being used to treat patients with diseases like multiple sclerosis. Multiple sclerosis is accompanied by demyelination of neurons [20]. Demyelination can be expected to slow the propagation of neuron signals from one neuron to another [19].

Another use of BCI's has been for people with severe physical disabilities [12]. [12] developed a BCI that detects eye movements and determines whether they can be categorized as either voluntary or involuntary by examining the action potentials and classifying them. In another study, Khairullaha et al. developed a BCI that takes action potential data and creates human-readable writing from them [10]. And one of the most recent uses of a BCI has been Neutralink's BCI that measures action potentials from a live pig and analyzes the action potentials when the pig is responding to a stimulus [14].

There is also work that deals specifically with clustering in computational neuroscience. Shah et al. performed clustering on action potentials based on the action potential spikes, specifically from visual neurons [18]. Their work shows that clustering by the spikes is an effective way to cluster action potentials.

Yet another example of clustering being used in computational neuroscience is Alashwal et al.'s work that looks into which clustering algorithms perform best on neural data for Alzheimer's disease [1]. This study found that k-Means, k-Means-Mode, multi-layer clustering, and hierarchical agglomerative clustering algorithms (discussed in section 2.2 below) have all been used successfully to perform clustering on neural data and draw conclusions about Alzheimer's patients.

2.1.1 Unsupervised Adversarial Learning: Clustering with Autoencoders

One paper in computational neuroscience that will have a significant influence on the methodology in this project is *Neural Spike Sorting Using Unsupervised Adversarial Learning* by Konrad Ciecierski [3]. In this paper, Ciecierski constructs an autoencoder to eliminate noise and reduce the dimensionality of the action potential data and clusters the data based on action potential spike shape. This reveals information about the different types of spikes, separated by shapes, as well as how similar the spikes in that cluster are to the average spike in that cluster. This paper is also discussed further in Chapter 3, where we go into more detail about the methods that we replicated from Ciecierski's work.

Autoencoder neural networks can be used for clustering data with no attributes, such as signal data that only has time series data [3]. An autoencoder consists of an encoder that

compresses the data to reduce the dimensionality, and a decoder that transforms the encoded data back into its full form. The process of using an autoencoder reduces the noise in the data. The data should be similar when it comes out of the decoder to how it was when it was inputted into the encoder - these two states of the data can be compared using mean-squared error. Performing clustering between the two layers of the autoencoder is beneficial because the dimensionality of the data is reduced, which makes the clustering smoother and prevents overfitting of a model with too many dimensions. This is important for action potential time series data, which might have thousands of voltage measurements over time for each action potential.

In Ciecierski's paper, he uses adversarial learning to implement more effective clustering. The input (an N by 48 tensor) is inputted into a neural network that is an encoder. The encoder puts the input through two dense layers, and then outputs a categorical head (an N by 10 tensor - 10 being the maximum number of classes that the model will find) and a Gaussian head (an N by 3 tensor). The decoder does the reverse, also having two dense layers.

We define the loss function as the mean-squared error (MSE) of the input that was fed into the encoder and the output that comes out of the decoder. This tells us how well the autoencoder has re-formed the data after encoding or compressing it.

Once the autoencoders have been trained, there is an adversarial phase in which the discriminators are trained, because they use the categorical and Gaussian heads that were outputted from the encoder. The discriminators are also neural networks. There is one for each head - categorical and Gaussian.

Finally, the generation phase is when the categorical head is made by the encoder into a categorical distribution and the Gaussian head is made into a normal distribution.

2.2 Clustering Algorithms

There are a variety of clustering algorithms, each with some benefits and faults [17]. We researched several different clustering algorithms to determine which ones might be most appropriate for the data we are working with.

One of the simplest clustering algorithms is k-means clustering [17]. It is also one of the fastest, running in O(n) time. It is a centroid-based algorithm, so it approaches the clustering problem by searching for the best points to place the centroids for each cluster and assigning the data points to clusters based on which centroid they belong with. A description of the algorithm is below:

1. Randomly place k centroids among the data points.
2. Assign each data point to the centroid closest to it.
3. Calculate the mean position of all the data points in each cluster and reposition the centroid to be at that mean.
4. Repeat steps 2 and 3 until the centroids no longer move during the repositioning in step 3 (or until they move very little).

One disadvantage of k-means clustering is that it requires the user to select a value of k, the number of clusters. It could be run for multiple values of k to determine which value results in the best clusters, but this would increase the runtime. K-means clustering also is not completely repeatable since the standard implementation begins with randomly placed centroids. The most significant disadvantage is that this algorithm's use of the mean as the centroid of each cluster is an over-simplified approach. This approach assumes that the clusters are close to circular, so it does not perform well with irregularly shaped clusters.

Another clustering algorithm is mean-shift clustering [17]. Like k-means clustering, it is centroid-based. However, mean-shift clustering automatically determines how many clusters there are in the dataset, which is an advantage over k-means clustering. Rather than having to choose the number of clusters k, the user instead has to define the radius r of the "window" of points the algorithm considers at once. The algorithm of $O(n^2)$ time is described below:

1. Place centroids among the data points, 2r apart from each other.
2. Reposition each centroid at the densest point (the point with highest concentration of data points) inside the "window" of radius r around the centroid.
3. Repeat step 2 until the centroids no longer move, so they are each at the densest area within radius r.

The radius of the window directly affects how the clusters form, so it is an important decision.

Density-based spatial clustering of applications with noise (DBSCAN) is a clustering algorithm that finds clusters by considering the density of the data points relative to each other, rather than seeking the location of the centroid like k-means and mean-shift clustering [17]. This algorithm automatically determines the number of clusters in the data, and functions better than k-means and mean-shift clustering in situations where the clusters are irregularly shaped or different sizes. Unlike k-means clustering, it selects the number of clusters automatically. It does require an input of the radius ε within which two data points would be considered to be in the same cluster. This algorithm is as described below:

1. Beginning with any data point, group this data point with any other points within distance ε from it.
    a. If there are not enough other points near this point, it is marked as noise and marked "visited."

   b. Otherwise, it is the considered beginning of a new cluster and marked as "visited."

2. Repeat step 1 for each point in the data set, finding the points closest to it and placing them in the same cluster of marking them as noise, until all the nodes are visited.

One main advantage of DBSCAN is that it identifies data points that are noise and do not belong in any cluster. This is particularly relevant in computational neuroscience, where there is a significant amount of noise in data from real neurons. One fault of this algorithm is that it does not perform as well when clusters vary in density.

OPTICS (Ordering Points To Identify Cluster Structure) clustering is similar to DBSCAN clustering, but it also adds the concepts of core distance and reachability distance [7]. Core distance is the minimum radius needed to classify a point as a "core point" of a cluster. Reachability distance is the distance between two core points. OPTICS clustering clusters the data based on the reachability distances between pints.

A fifth clustering algorithm is expectation–maximization clustering using Gaussian mixture models [17]. Where k-means clustering performs poorly with clusters that are not close to a circular shape, this algorithm expects clusters to have Gaussian distributions, so it tends to perform better than k-means clustering when the clusters are oblong rather than circular. This algorithm is O(n) time.

Agglomerative hierarchical clustering is another $O(n^3)$ algorithm that does not require any user input other than the data itself [17]. Each data point starts as its own cluster and they iteratively combine clusters. The algorithm is described below:

1. Begin by considering each data point as its own cluster.
2. Combine the two clusters that are closest to each other into one cluster.

3. Repeat step 2 until the desired number of clusters is reached.

This algorithm also does not require a user input of the number of clusters or “window’ radius. However, it does let the user select how many clusters to use, since the algorithm can stop combining clusters at any point.

Another technique is simulated annealing, where the data points are redistributed probabilistically so the clustering reveals the groups that would not exist with randomly distributed data points, therefore they are statistically significant [11]. This method can be used in combination with other clustering algorithms.

2.3 Clustering Performance Measures

Once the chosen algorithm has formed some clusters, it is necessary to evaluate its performance on the data. One way to evaluate the algorithm is by analyzing its runtime. We also have to analyze the results of the algorithm. Ideally, good clusters should have a high density of data points within the clusters, and be spaced out between different clusters [23]. The number of clusters and overlap between clusters are also factors [5].

One measure of the performance of a clustering algorithm is the Davies-Bouldin index [11, 23]:

$$DB = \frac{1}{n}\sum_{i=1}^{n} \quad max\left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right)$$

n is the number of clusters and σ_i is the average distance of all the data points in cluster i from the centroid c_i. This index considers all the clusters in the dataset and reflects how well spaced out from each other they are, as well as how dense the individual clusters are. A smaller value is better.

The Dunn index also considers how well spaced out the clusters are and how dense the individual clusters are [11, 23]. However, it does not equally consider all the clusters like the Davies-Bouldin index does. Instead it considers the "worst" clusters - the ones that are least dense and least spaced out from other clusters.

$$D = \frac{min_{\,1 \le i < i \le n}\; d(i,j)}{max_{\,1 \le k \le n}\; d'(k)}$$

where i, j, and k are indices for clusters, d is the distance between points in the cluster, and d' is the distance between clusters. A larger value of the Dunn index indicates better clusters.

However, we have chosen to use the Davies-Bouldin index over the Dunn index because they take into account the same characteristics of the clusters, so both are not necessary.

A third measure of cluster quality is the Silhouette index [23]. Rather than considering each cluster like the two indices above, the Silhouette coefficient reflects how well assigned each data point is. It ranges from -1, indicating that the data point should be in another cluster other than the one it is assigned to, to +1, indicating that the point is in the correct cluster.

$$S(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

a(i) is the average distance between the data point *i* and all other points in that cluster, and *b(i)* is the smallest average distance between the data point *i* and all the points in another cluster. This equation must be repeated over each data point in the dataset.

Because each of these three indices measures the quality of the clustering differently, all of them are useful in combination for comparing the results from various clustering algorithms.

2.4 Existing Clustering Tools

Several tools already exist for Python that implement the clustering algorithms described above. Klusta is a program that takes for input a flat binary file that contains analog multi-channel signals composed of neural spikes [15]. The program then detects neural spikes in the data and sorts them accordingly with a flood-fill algorithm and subsequently clusters them into groups stemming from the same neuron. This program is directly linked to our project and studying it is greatly valuable in aiding our understanding and design of our project, especially understanding the flood-fill algorithm.

Klusters is a tool distributed under the GNU public license that sorts independent action potentials into clusters [8]. This tool is directly linked to our project as it is clustering independent action potentials which is a crucial part of our application. Studying this program will prove beneficial to our understanding and design of our own program.

Sci-kit Learn is a Python library that implements many different clustering algorithms and other machine learning tools.

2.5 Noise Filtering

Removing the noise in the action potential recording is essential when using the data in AI because it reduces the dimensionality of the data, which makes it easier to deal with using AI methods (Therese M. Smith, Personal Communication, 5 October 2020). Low-pass filtering has been used in the past to filter neuron action potential data. However, the Kalman filter is more effective than low-pass filtering for neuroscience data because while low-pass filtering uses a simplified approach that eliminates all frequencies above some threshold, Kalman filtering uses a more advanced approach [21]. Kalman filtering detects a larger variety of noise types, including Gaussian, bimodal, and Cauchy noise. Therefore, it is appropriate for very noisy data from neurons. Weiss et al. further improve the filtering by using autoencoder-Kalman filtering (AEKF), which uses an autoencoder before the Kalman filtering. Autoencoders serve to encode the data into a more compacted format, which reduces the dimensionality of the data and eliminates noise. The data can then be restored to a state similar to its original state, but with less noise.

If we model the noise with a normal probability distribution function, then with additive white Gaussian noise, the variance is constant and the mean is always 0, making it simple to eliminate (Therese M. Smith, Personal Communication, 27 October 2020). However, when the variance changes, this becomes multiplicative noise. Multiplicative noise is characterized by a normal probability distribution function - small amounts of noise happen often, while large amounts of noise happen more rarely. When the frequency of the action potential increases, the noise variance might also increase. Multiplicative noise must also be considered when dealing with action potential data.

There are several libraries and tools that exist for Python that are useful for implementing an AEKF for noise filtering, including the Python Keras library from Tensorflow, which includes tools for making neural networks and autoencoders.

2.6 How This Project Builds Upon Related Work

This project is using existing clustering tools and clustering performance measures that are already established and accepted. We are replicating the use of the autoencoder by Ciecierski [3] and also working off the methodology by Wiess et al. [21] for the AEKF. We are improving upon what has been done before by using Kalman filtering on action potential data in combination with AI and clustering, which has never been done before.

Chapter 3: Methodology

To reach our goal of using clustering on hippocampal neuron action potentials to compare the AEKF and unsupervised adversarial learning methods, we have several sub-objectives:

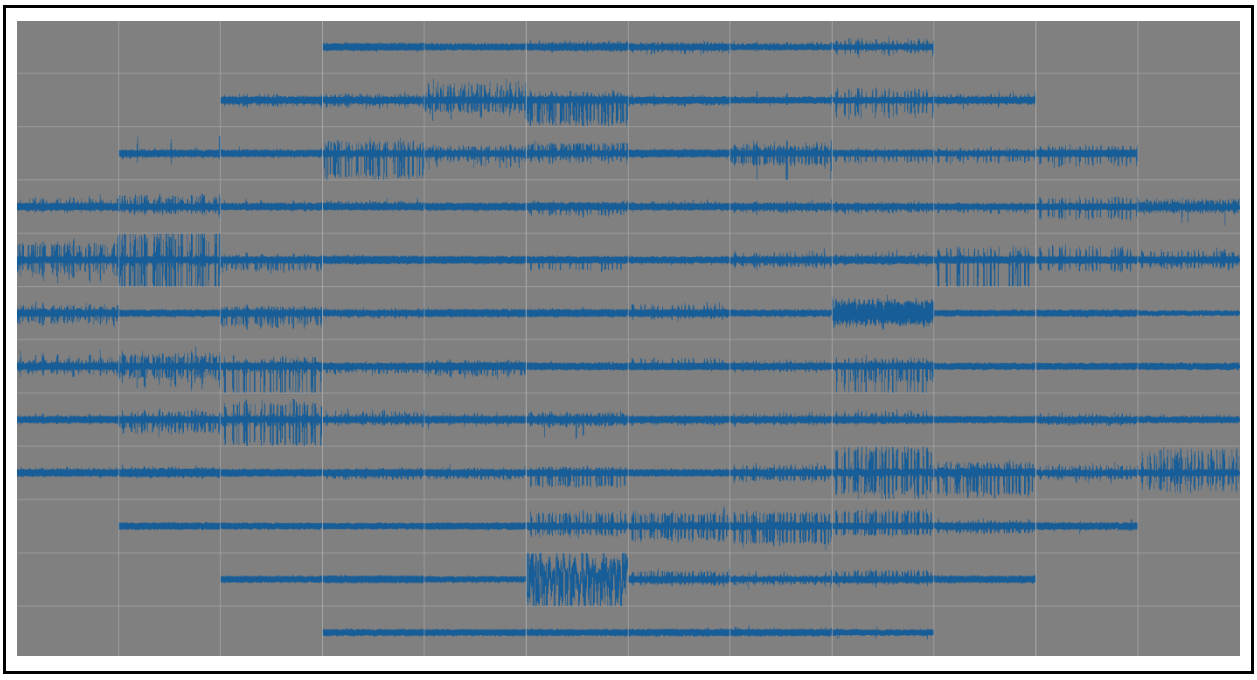
1. Explore and clean the data.
2. Perform noise filtering using autoencoder-Kalman filtering (AEKF) as described by Weiss and Paffenroth.
3. Perform autoencoder filtering as described by Ciecierski.
4. Perform clustering using various algorithms on the AEKF output, the autoencoder output, and the original (unfiltered) data.
5. Compare the results from each type of clustering for each dataset using clustering performance measures for unsupervised learning.

This section is outlined according to these sub-objectives.

3.1 The Data

We used two datasets for this project: a dataset from Collaborative Research in Computational Neuroscience Data Sharing (CRCNS), and a dataset used by Konrad Ciecierski in *Neural Spike Sorting Using Unsupervised Adversarial Learning*.

From CRCNS, we chose to use the dataset called hc-3. The data from CRCNS contains action potentials measured from the hippocampi of rats. More can be read about this data from CRCNS. This data contains many recordings over time of action potentials, each recording containing numerous spikes. Figure 3.1 shows a visualization of the action potential spikes in one file from the hc-3 dataset:



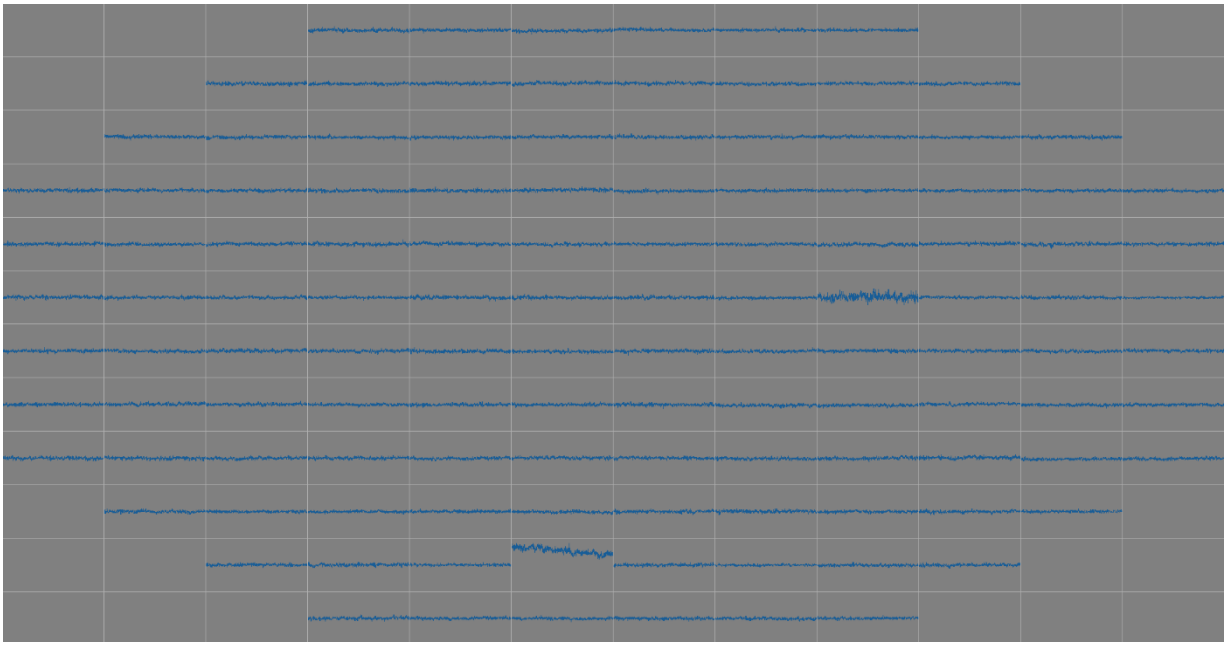


Figure 3.1: Action Potential Spikes in Data from CRCNS

These visualizations were produced using the MEA Tools software by Dan Bridges, available at https://github.com/dbridges/mea-tools.

The data we used to train the models is the same dataset used by Ciecierski in *Neural Spike Sorting Using Unsupervised Adversarial Learning* [3]. This dataset also contains recordings over time of action potentials, each recording containing multiple spikes. Visualizations of the recordings are shown in Figures 3.2 and 3.3:

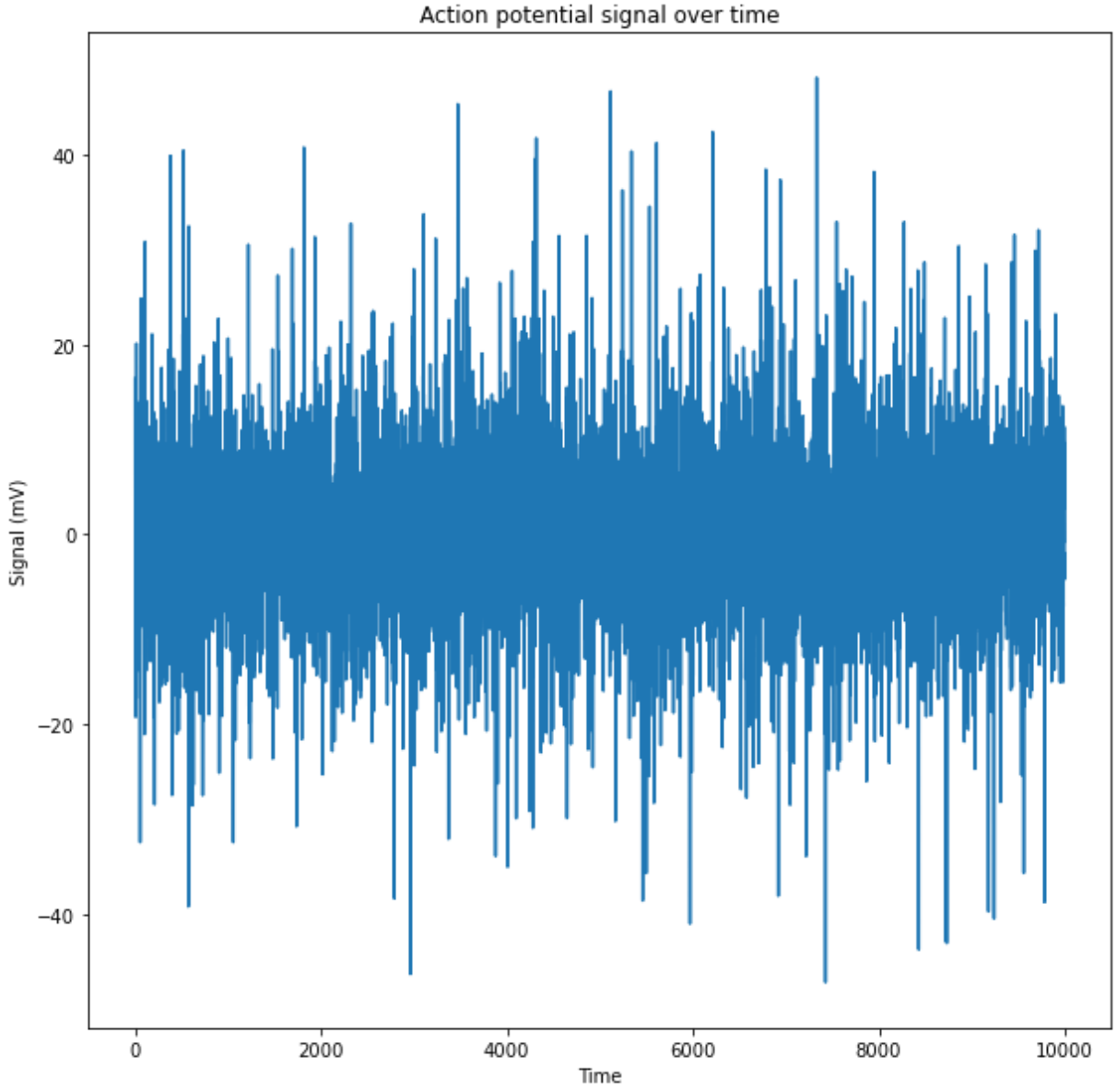


Figure 3.2: The Action Potential Spikes in Ciecierski Data: One Full Recording

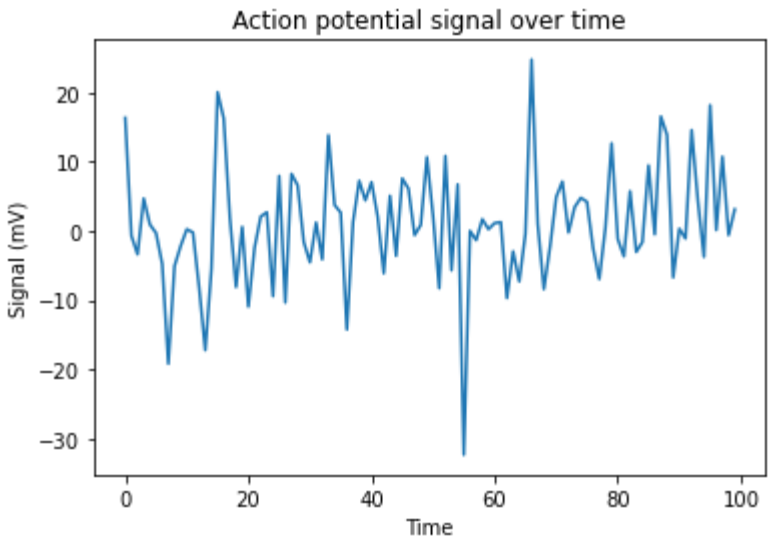


Figure 3.3: The Action Potential Spikes in Ciecierski Data: A Partial Recording for Detail

Before performing clustering on the data, we did preliminary data cleaning and data exploration to understand the data and put it into a format that makes it easier to cluster.

Another step before clustering the data was to split the data into training, testing, and validation sets, in proportions of 80%, 16%, and 4%, respectively.

3.2 Autoencoder-Kalman Filtering

To filter the noise out of our action potential data before clustering, we implemented an autoencoder-Kalman filter (AEKF), referencing *The Autoencoder-Kalman Filter: Theory and Practice* by Weiss et al. from 2020 [21].

The AEKF is essentially a Kalman filter inside of an autoencoder neural network. It consists of two layers of encoders, followed by the Kalman filter, followed by two layers of decoders. Performing the Kalman filtering inside the autoencoder is beneficial because the dimensionality of the data is reduced so the filtering can be more accurate and efficient.

We implemented this by using the Python library Keras for the autoencoder and writing the Kalman filter separately. The autoencoder is made up of the encoder and the decoder. The encoder is one Functional model in Keras with several layers [9]. For each Keras model we have created, we have generated a diagram to show the neural network layers using the Keras function "keras.utils.plot_model()". Figure 3.4 shows the model for the autoencoder:

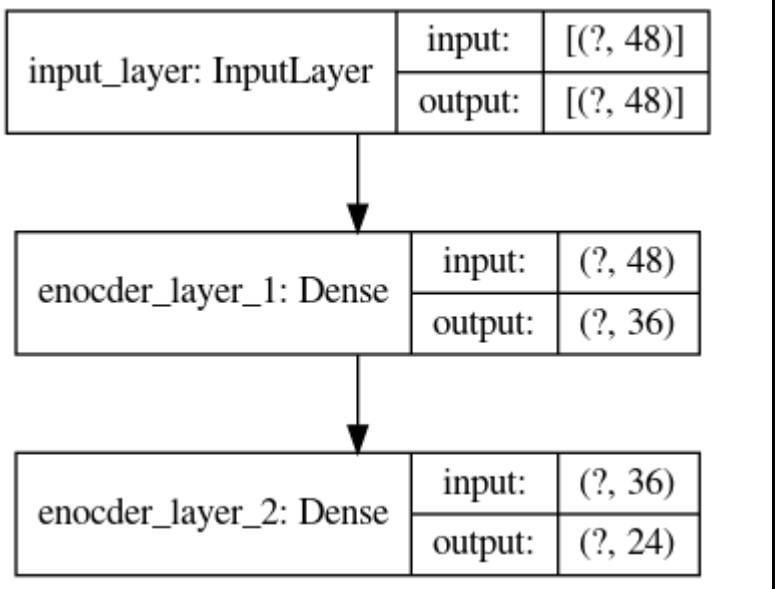


Figure 3.4: The Layers of the Encoder for the AEKF

The encoder takes an input and puts it through 2 neural network layers. Next, the data passes through the Kalman filter. We used [16] as a base for our Kalman filter implementation, as well as the method used by Weiss et al. each sample tuple of data is passed through the Kalman filter.

Next, we made the decoder. This is also part of the neural network using Keras. The layers of the decoder are shown in Figure 3.5:

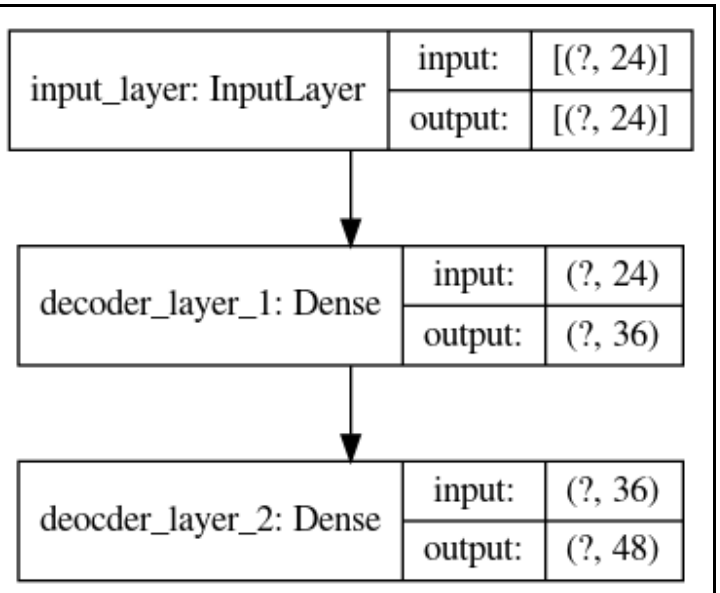


Figure 3.5: The Layers of the Decoder for the AEKF

The decoder is similar to the encoder and passes the input, which has just been filtered through the Kalman filter, through 3 layers of the neural network. The output from this decoder has been filtered, and it can then be passed into the clustering algorithms.

3.3 Unsupervised Adversarial Learning Autoencoder

First, we implement the autoencoder. Our implementation follows those of Ciecierski [3] and Weiss et al. [21]. We used the Python library keras for the layers of the autoencoder. The autoencoder consists of an encoder and a decoder. The encoder is a neural network into which we input a tensor (N, 48) and output a tensor (N, 13). We can then split that output into two into two tensors: a categorical head (N, 10) and a Gaussian head (N, 3). Inside the encoder, there are two dense layers (meaning all the neurons in each layer are connected to all the other neurons in that layer). Thus, the inputs are encoded and condensed.

The decoder takes the tensors of dimensions (N, 10) and (N, 3) and outputs a tensor of dimensions (N, 48). The output is restored to an approximation of the original data, but with dimensionality reduced and noise eliminated.

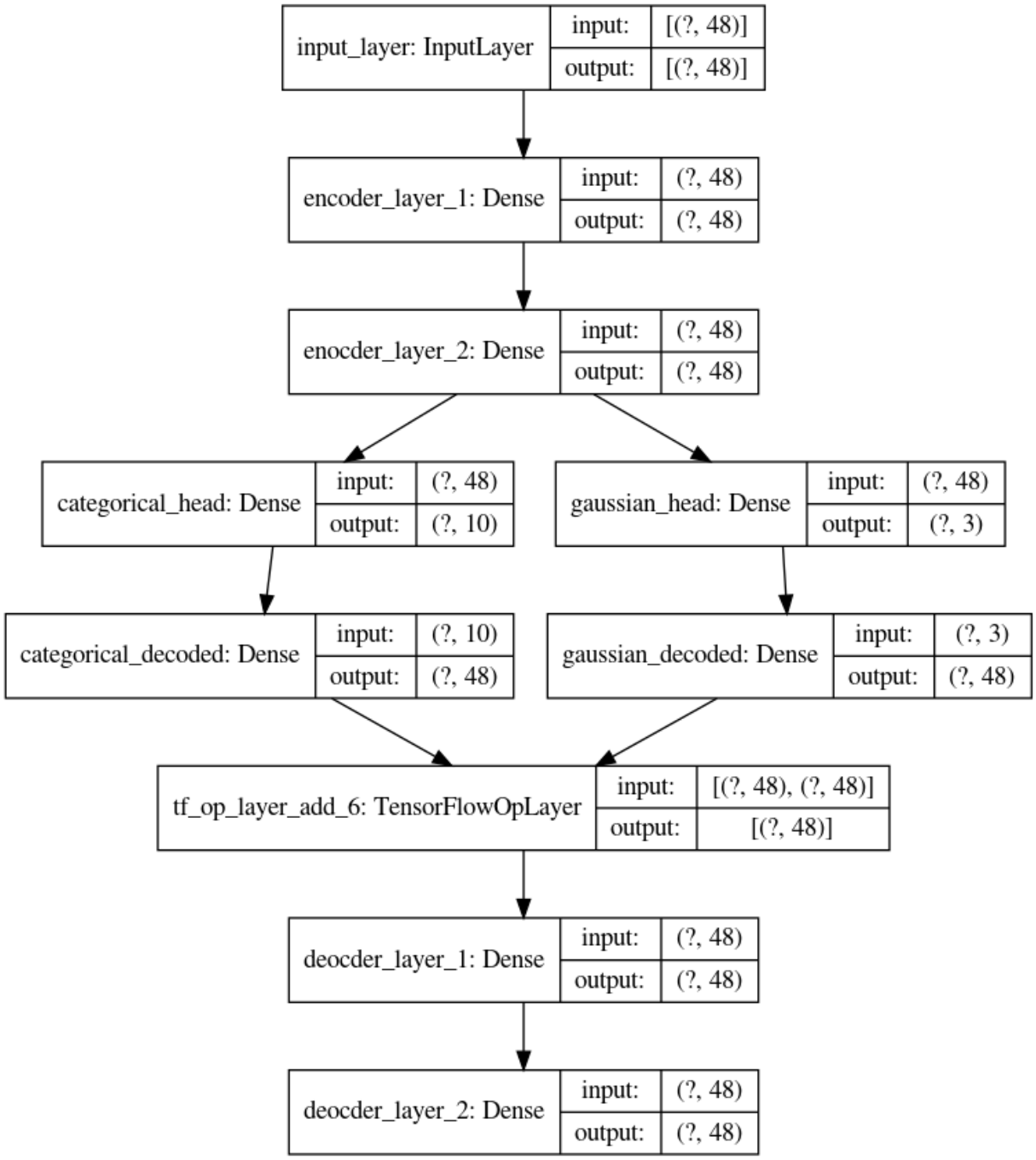


Figure 3.6: The Autoencoder for Reducing Noise in Unsupervised Adversarial Learning

We also implemented the custom loss function to optimize the deep learning autoencoders:

$$in = [x_{i,j}],\ i = 0 \ldots N-1,\ j = 0 \ldots 47; \quad out = dec(enc(in))$$

$$SE_p(in, out, b, e) = \sum_{i=0}^{N-1} \sum_{j=b}^{e} (in_{i,j} - out_{i,j})^2$$

$$SE_w(in, out) = SE_p(in, out, 0, 5) + 50\, SE_p(in, out, 6, 17) + SE_p(in, out, 18, 47)$$

$$MSE_w(in, out) = \frac{SE_w(in, out)}{48\ N}$$

(Ciecierski 2020)

This method is also described in *Neural Spike Sorting Using Unsupervised Adversarial Learning* by Ciecierski [3].

3.4 Clustering Algorithms

We used multiple clustering algorithms so we could compare the results from each one. First, we used k-means clustering, since it is simple and quick, so we can get some preliminary results. For k-means and any other algorithms that require an input of the number of clusters, we used a range of cluster numbers from 2 to 10, based on the results of Ciecierski [3] that showed that there were no more than 9 clusters in the dataset, as well as the fact that 2 is the minimum number of clusters that k-means can create. We also used mean-shift clustering because it determines the optimal number of clusters itself. Then, we used DBSCAN clustering, since this type of clustering is designed for applications with noise, which the neuron action potentials have. We also used hierarchical agglomerative clustering, since this has been shown to be effective in neuroscience clustering applications [1]. We used all of these clustering algorithms from the scikit-learn Python library.

3.5 Performance Evaluations and Visualizations

To evaluate the performance of each clustering algorithm for each type of filtering (autoencoder, AEKF, and no filtering), we used the performance measures of DB Score and Silhouette Index from the sklearn implementations. We also compared the filtering methods for each algorithm by making plots that compare the values - for example, a plot comparing the DB scores for each filtering method on the training and testing set, plotted along values of k for the k-means algorithm. In addition to the performance measures of the clustering algorithms, we also evaluated our implementations of the autoencoder and AEKF by visualizing the data at multiple steps in the filtering process - for example, so we can see what the data looks like when it has been encoded, and how it turns out after being decoded. We made the visualizations using the matplotlib library in Python.

Chapter 4: Results

In this chapter, we will present and analyze the results of our research, which include the performance measures and visualizations. We will discuss the training of the autoencoder and AEKF on the data, and compare the performances of the three filtering methods (the autoencoder-Kalman filter (AEKF), the autoencoder, and no filtering) for each of the clustering types we used.

4.1 Autoencoder Training

The results of the autoencoder testing are shown in Figure 4.1. We used 32 training epochs with 2 stages per epoch. The loss function continued to decrease until the 32nd epoch but the difference per epoch became very small after 22 epochs.

Figure 4.1: Autoencoder Loss Function Value Across Training Epochs

This means that as the autoencoder was training, the custom loss function described in section 3.3 was successfully being minimized.

4.2 AEKF Training

The AEKF training phase resulted in the loss function values shown in Figure 4.3. The loss function value continued to decrease across the 32 epochs, but it decreased more slowly towards the end.

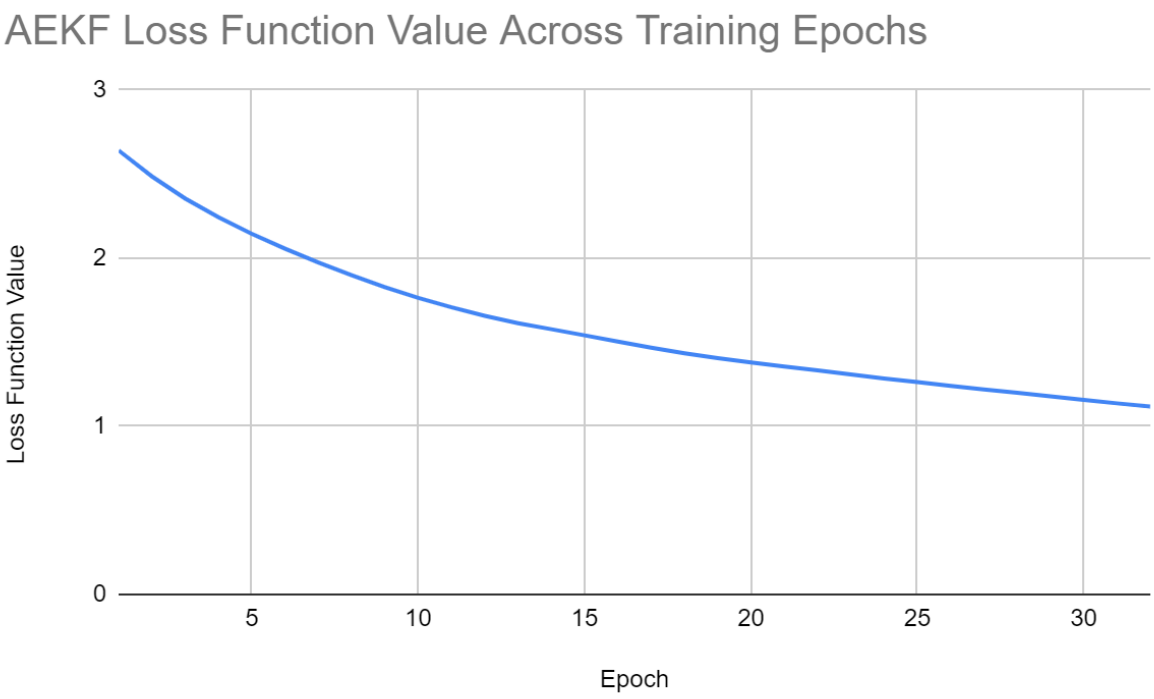
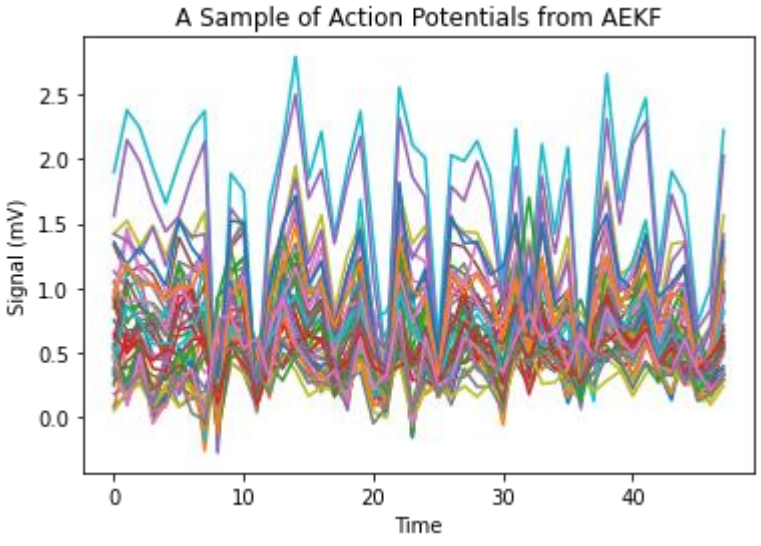


Figure 4.2: AEKF Loss Function Value Across Training Epochs

This indicates that the loss function for the AEKF was successfully being minimized. It might have become slightly smaller with more training epochs, but we determined that this number of training epochs was sufficient because function was decreasing slowly by the time it had trained on 32 epochs.

The following plots made in matplotlib show the difference in the data before and after it was processed through the AEKF. The AEKF made the data more comprehensible as well as eliminating the noise. It also made almost all of the values positive, compared to the approximately normal distribution of values before filtering with the mean centered around 0. It also got rid of some outliers which were likely affected by noise.

(a)



(b)

Figure 4.3: A Sample of Action Potentials from Unfiltered (a) and AEKF (b) Data

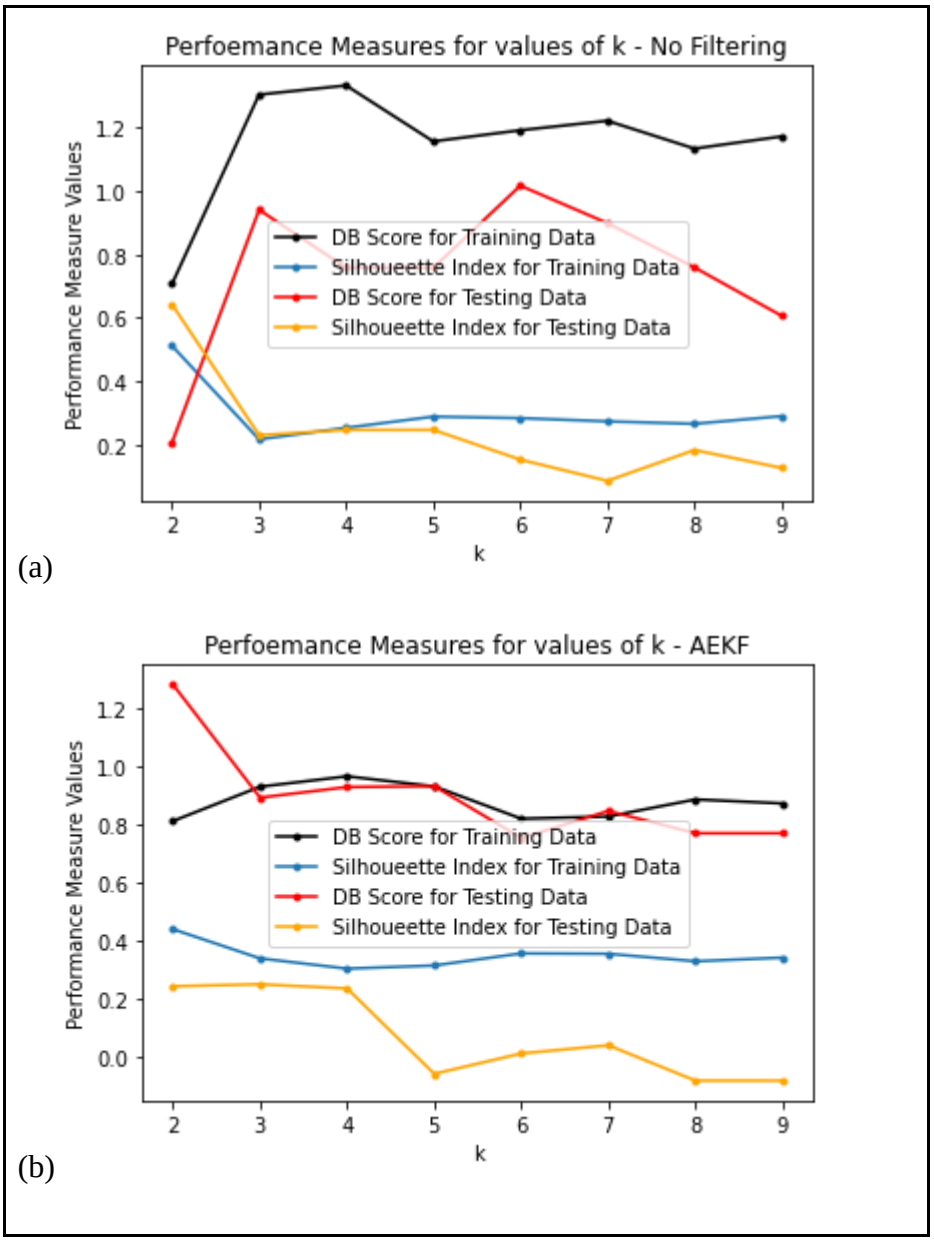
These visualizations show that the AEKF was filtering the data in a way that might make it easier to cluster.

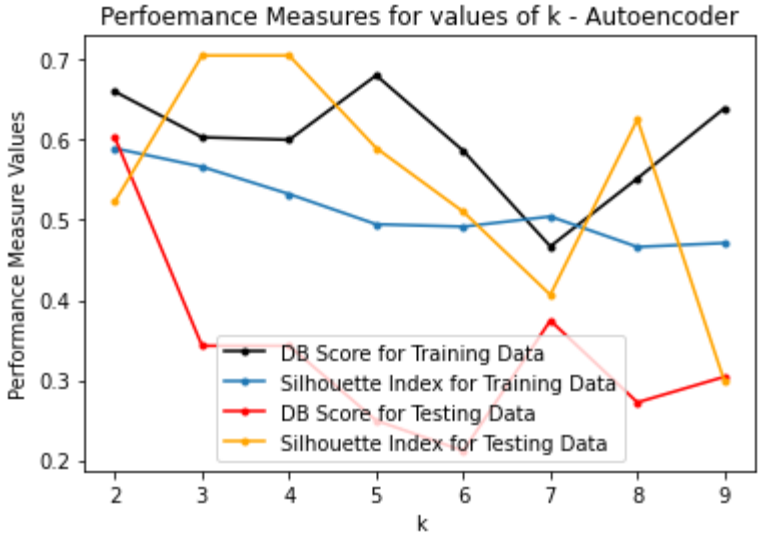
4.3 Comparison of Filtering Methods for Clustering

For each clustering algorithm, we will provide visualizations of the performance measures to show visually the differences in performance of the different algorithms. As a reminder, for the performance measures of the Davies-Bouldin (DB) score, lower values (closer to 0) signify better clusters. The silhouette index ranges from -1 to 1, and higher values are better.

4.3.1 k-Means Clustering

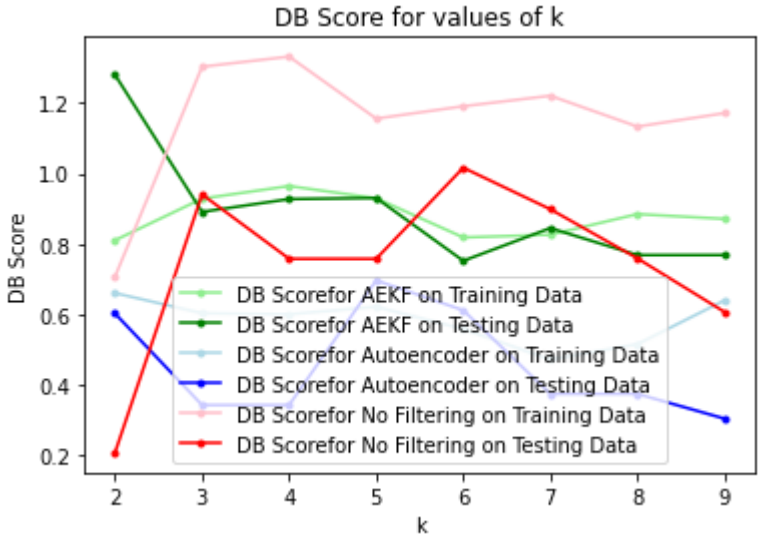
Figure 4.4 shows the DB scores and silhouette indices for training and testing data sets after using k-means clustering on the outputs from each of the filtering methods, plotted against k, the number of clusters used for the k-means clustering algorithm. In addition, Figure 4.5 shows the same values plotted, but with each plot representing a different score rather than a different filtering method.


Perfoemance Measures for values of k - No Filtering
Performance Measure Values
k
DB Score for Training Data
Silhoueette Index for Training Data
DB Score for Testing Data
Silhoueette Index for Testing Data
(a)
Perfoemance Measures for values of k - AEKF
Performance Measure Values
k
DB Score for Training Data
Silhoueette Index for Training Data
DB Score for Testing Data
Silhoueette Index for Testing Data
(b)

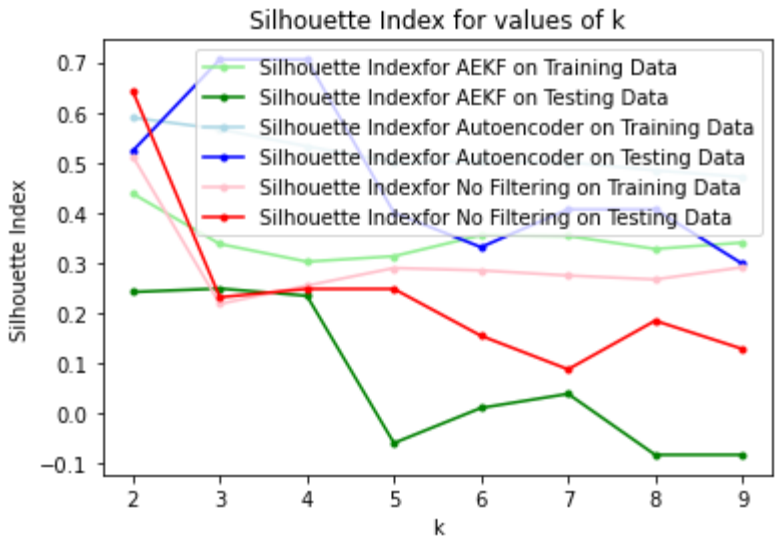


(c)

Figure 4.4: The DB Scores and Silhouette Indices for Training and Testing Data for Each Filtering Method Using k-Means Clustering



(a)



(b)

Figure 4.5: Comparisons of the DB Scores (a) and Silhouette Indices (b) for Each Filtering Method on Training and Testing datasets

Overall for k-means clustering, the silhouette indices were highest (best) for the autoencoder method (testing and training), and worst for no filtering (testing) and the AEKF (training). The DB scores were lowest (best) for the autoencoder method (training and testing) and worst for no filtering (training and testing).

4.3.2 Mean-Shift Clustering

Table 4.1 shows the DB scores and silhouette indices for mean-shift clustering:

Table 4.1: Performance Measure for Each Filtering Method using Mean-Shift Clustering, for Training and Testing Datasets

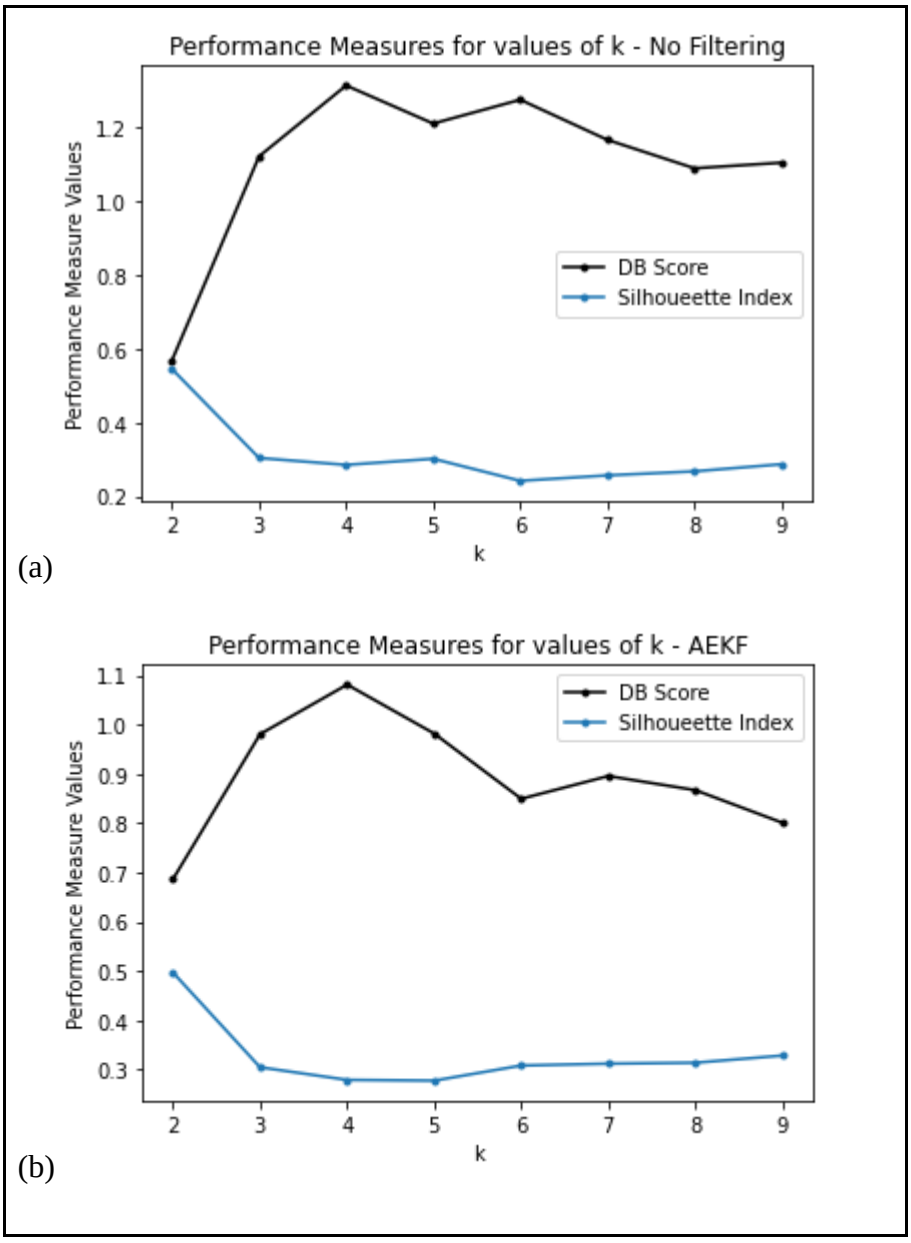
Filtering Method	**Dataset**	**k**	**DB Score**	**Silhouette Index**

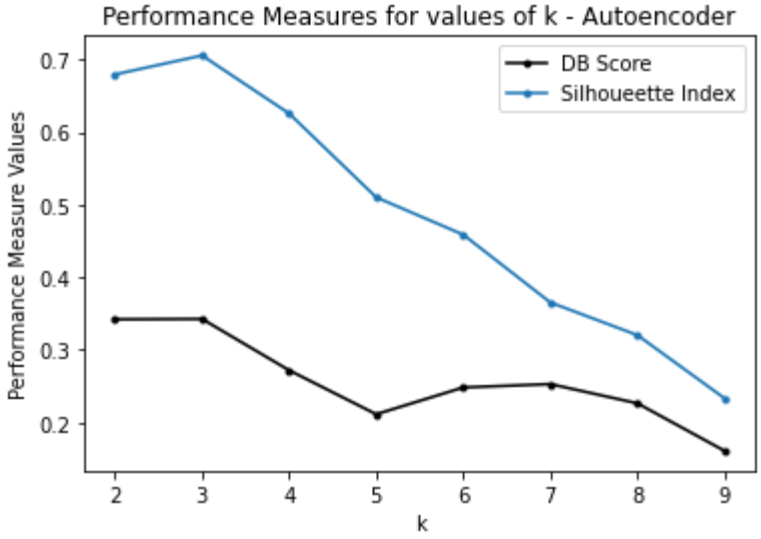
No Filtering	Training	2	0.56811	0.544272
	Testing	2	0.219408	0.626561
AEKF	Training	3	0.775299	0.261609
	Testing	3	0.801855	0.249039
Autoencoder	Training	5	0.441522	0.548717
	Testing	3	0.34275	0.705601

These values show that the highest silhouette indices overall were for the autoencoder (testing) and no filtering (beating the autoencoder by a small margin in the training set), while the lowest were for the AEKF. The lowest DB scores were for the autoencoder (training) and no filtering (testing).

4.3.3 Agglomerative Hierarchical Clustering

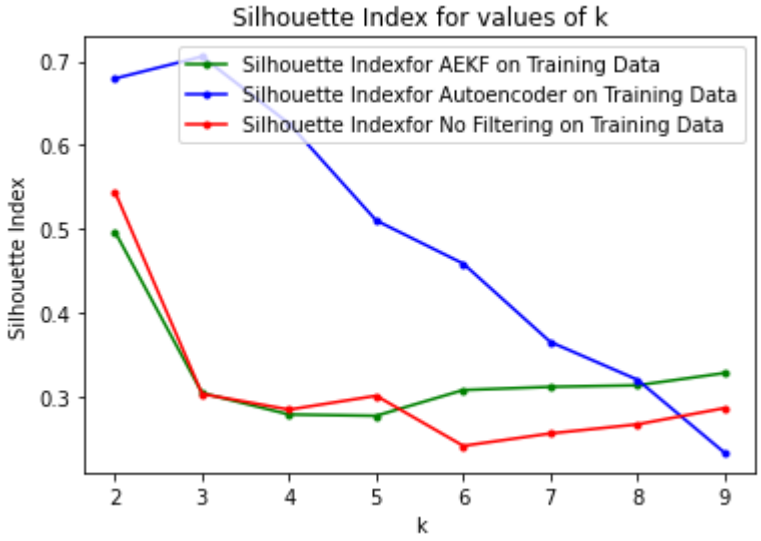
The plots here are equivalent to the plots showing performance measures for k-means clustering except that there is no testing or predicting in this type of clustering, so there is no test performance measure.


Performance Measures for values of k - No Filtering
Performance Measure Values
k
DB Score
Silhoueette Index
(a)
Performance Measures for values of k - AEKF
Performance Measure Values
k
DB Score
Silhoueette Index
(b)

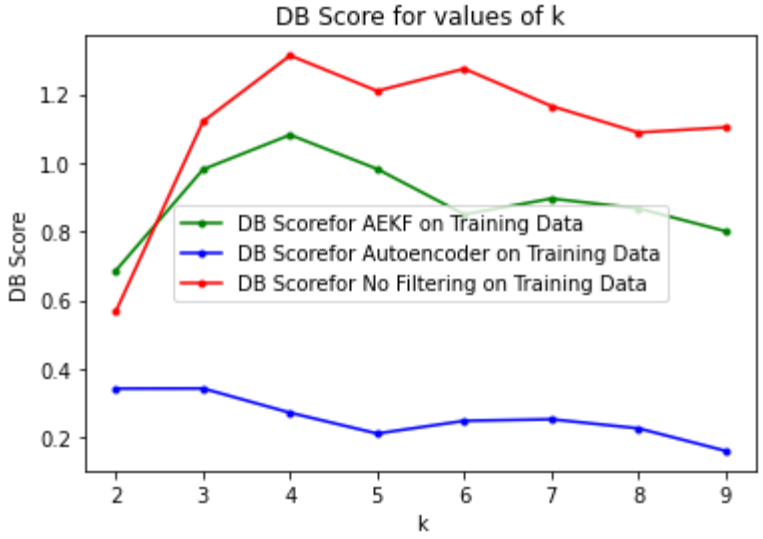


(c)

Figure 4.6: The DB Scores and Silhouette Indices for Each Filtering Method Using Agglomerative Hierarchical Clustering



(a)



(b)

Figure 4.7: Comparisons of the DB Scores (a) and Silhouette Indices (b) for Each Filtering Method and Agglomerative Hierarchical Clustering

These results show that the autoencoder performed significantly better in both DB scores and silhouette indices using agglomerative hierarchical clustering.

4.3.4 OPTICS Clustering

Table 4.2 shows the performance measure values for each filtering method using OPTICS clustering.

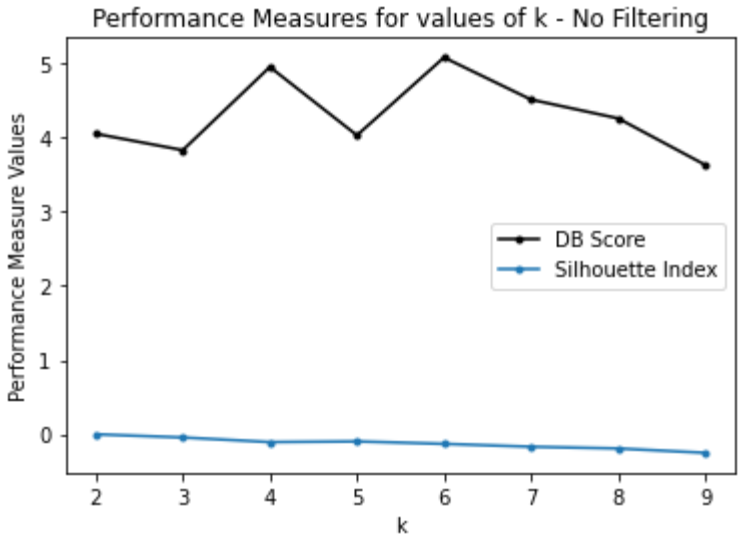
Table 4.2: Performance Measure for Each Filtering Method using OPTICS Clustering

Filtering Type	**Number of Clusters**	**DB Score**	**Silhouette Index**
No Filtering	4	2.332557	0.122551
AEKF	3	2.669474	0.052706
Autoencoder	2	1.44418	-0.299292

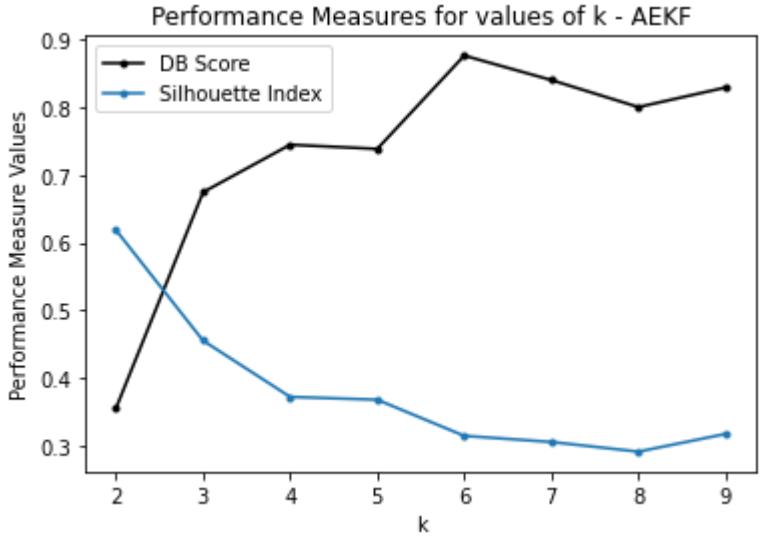
The autoencoder has the best DB score and the unfiltered data has the best silhouette index. However, this table shows that the filtering methods all performed worse with OPTICS clustering compared to other types of clustering. This could indicate that OPTICS clustering is not well suited to the action potential data itself.

4.3.5 Spectral Clustering

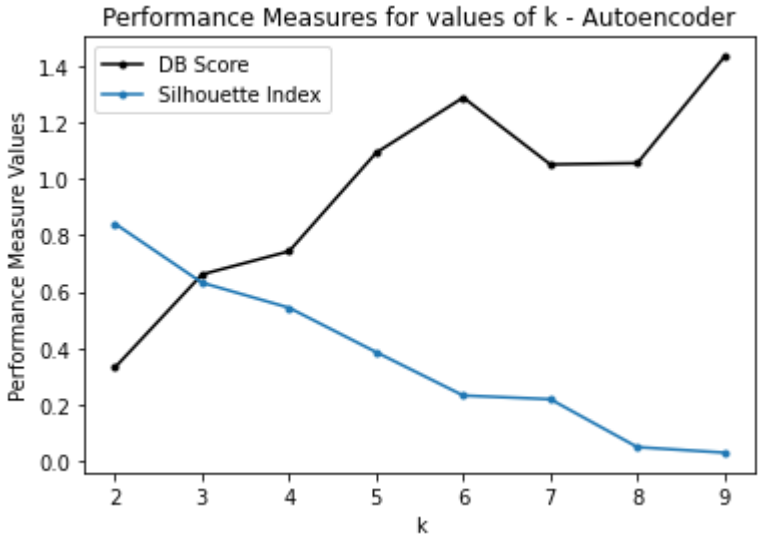
Using spectral clustering, we tested values of k (numbers of clusters) from 2 to 9 and plotted the values as shown in Figure 4.8. In addition, Figure 4.9 shows the same data in a way that is easy to compare between filtering methods.


Performance Measures for values of k - No Filtering
Performance Measure Values
k
DB Score
Silhouette Index

(a)

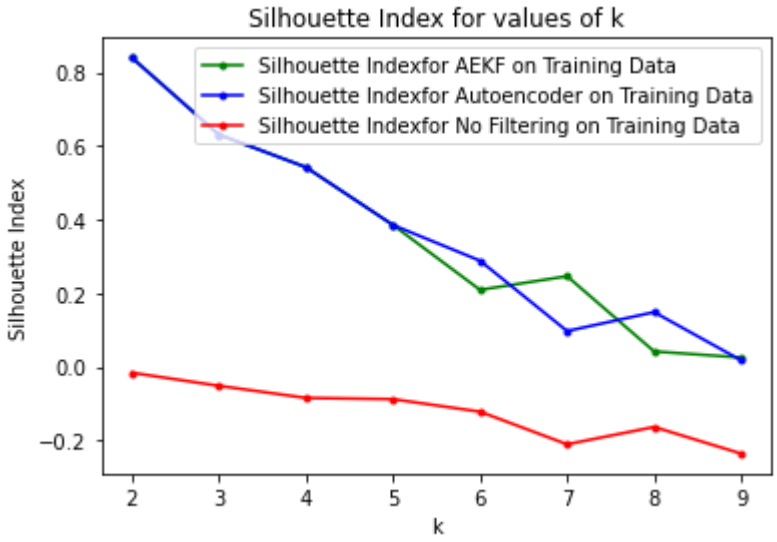

Performance Measures for values of k - AEKF
Performance Measure Values
k
DB Score
Silhouette Index

(b)

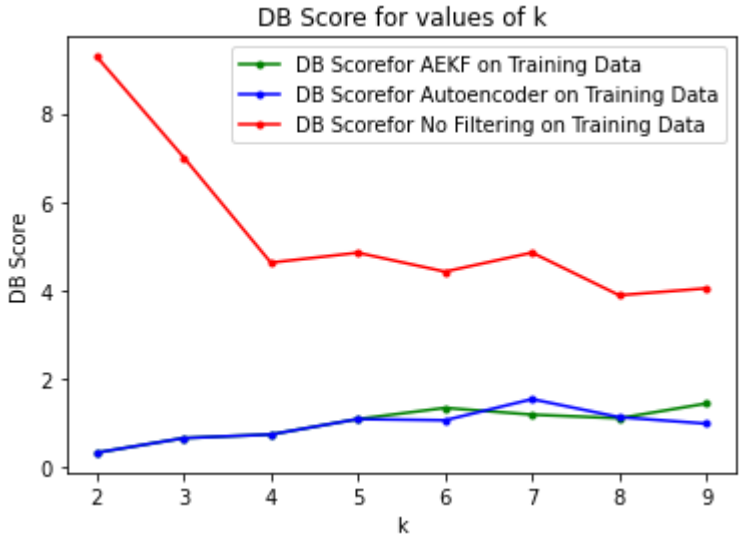

Performance Measures for values of k - Autoencoder
Performance Measure Values
k
DB Score
Silhouette Index

(c)

Figure 4.8: Performance Measure Scores for Each Filtering Method Across Values of k



(a)



(b)

Figure 4.9: Silhouette Indices (a) and DB Scores (b) for Each Filtering Method Across Values of k

The unfiltered data performed significantly worse in both DB scores and silhouette indices than both types of filtered data. The filtered data types performed very similarly in both performance measures.

Chapter 5: Conclusions

In this section, we will summarize the analysis of our results and draw conclusions, as well as answering our research question of which filtering method for action potentials is most effective as preprocessing before clustering.

5.1 Clustering Performance Measures

In Table 5.1, we summarize which filtering methods had the best performance for each clustering algorithm.

Table 5.1: The Best Performance Measures Values by Filtering Method for Each Clustering Algorithm

Clustering Algorithm	**Filtering Method with Best Performance Overall**			
	DB Score - Training	**DB Score - Testing**	**Silhouette Index - Training**	**Silhouette Index - Testing**
k-Means Clustering	Autoencoder	Autoencoder	Autoencoder	Autoencoder
Mean Shift Clustering	Autoencoder	No Filtering	Autoencoder	Autoencoder
Agglomerative Hierarchical Clustering	Autoencoder	---	Autoencoder	---
OPTICS Clustering	Autoencoder	---	No Filtering	---
Spectral Clustering	Autoencoder and AEKF	---	Autoencoder and AEKF	---

Overall, we can conclude from this table that the autoencoder method by Ciecierski (2020) performed best across multiple clustering algorithms. In addition, when evaluating these

results overall, we can place more importance on agglomerative hierarchical clustering than the other clustering methods because it is the most highly recommended clustering method for clustering action potentials (Seif, 2020). We can also place less emphasis on the OPTICS clustering because all of the filtering methods performed poorly with OPTICS clustering.

One factor that might give Ciecierski's autoencoder method an advantage over other filtering methods is the custom loss function that is designed for action potential data that is 48 data points wide. The custom loss function gives more emphasis to the most important data points of the action potentials, which are also the points that vary the most since they are centered around the center of the spike. This could cause this method to perform better when used as preprocessing before the various clustering algorithms.

5.2 Limitations

One limitation of our research is that we did not use multiple datasets for our research. We also did not implement the domain randomization for training used in (Weiss et al. 2019). This means that our training and testing datasets were different, randomly selected subsets of the same dataset. It also means that it is possible that our results do not generalize to more datasets. For example, the autoencoder method could simply be more suited to this particular dataset, but not better overall when considering other datasets.

A limitation of our comparison of the methods used by Ciecierski (2020) and Weiss et al. (2019) is that we implemented our own versions of the methods described in their papers. Our implementations are slightly different from theirs, although they implement the same noise filtering methods.

5.3 Recommendations for Future Research

One additional task that we were not able to complete over the course of this project was to combine the unlabeled data we used for clustering with some labelled data to use as a check on the clustering. This would allow the researchers to better evaluate and compare the performance of the clustering methods, as well as indicating whether the clusters of action potentials correspond to groups of individuals who share particular characteristics such as diseases that cause demyelination of neurons and thereby slow down the action potentials.

Another continuation of this study would be to conduct the study on different action potential datasets. For example, future researchers could use datasets with action potentials from different individuals and from different parts of the brain. This could reveal whether there is any variation in the performance of these methods based on the action potential data itself, or whether the different filtering methods are more specialized to particular types of action potential data.

Another potential continuation of this study would be to use more clustering algorithms in addition to more data. Since different clustering algorithms are better suited to different datasets based on the types of clusters they contain, different algorithms might perform better or worse on the different datasets. There are also many clustering algorithms that we did not use in this study because we chose to use only a subset of the clustering algorithms that exist, so using more clustering algorithms could reveal some more insights.

We also recommend for future research to build upon this project by running the code from each of these papers directly on the same dataset, if this is possible. In this project, we compared the methods based on our implementations of them. However, our implementations

might be slightly different from the ones used in the papers. One main reason for this is that we did not implement domain randomization, so the models were trained and tested on the data itself rather than on domain randomized data. This might have an effect on the performance of each filtering method.

Bibliography

[1] Alashwal, Hany, Mohamed El Halaby, Jacob J. Crouse, Areeg Abdalla, and Ahmed A. Moustafa. *The Application of Unsupervised Clustering Methods to Alzheimer's Disease.* Frontiers in Computational Neuroscience (2019). DOI: 10.3389/fncom.2019.00031

[2] Burns, Alexis, Hojjat Adeli, and John A. Buford. *Brain–Computer Interface after Nervous System Injury* (2014). DOI: 10.1177/1073858414549015

[3] Ciecierski, Konrad. *Neural Spike Sorting Using Unsupervised Adversarial Learning* (2020). DOI: 10.1007/978-3-030-59491-6_18

[4] Coggan, Jay, Stefan Bittner, Klaus M. Stiefel, et al. *Physiological Dynamics in Demyelinating Diseases: Unraveling Complex Relationships through Computer Modeling*. International Journal of Molecular Sciences (2015). DOI: 10.3390/ijms160921215

[5] Fränti, P., Sieranoja, S. *K-means properties on six clustering benchmark datasets.* Applied Intelligence 48, 4743–4759 (2018). DOI: 10.1007/s10489-018-1238-7

[6] Gouwens, Nathan W., et al. *Toward an integrated classification of neuronal cell types: morphoelectric and transcriptomic characterization of individual GABAergic cortical neurons.* Allen Institute for Brain Science (2020). https://www.biorxiv.org/content/10.1101/2020.02.03.932244v1.full.pdf

[7] Gupta, Alind. "ML | OPTICS Clustering Explanation." Geeks for Geeks. https://www.geeksforgeeks.org/ml-optics-clustering-explanation/

[8] Hazan, L., & Zugaro, M. *Free Data Visualization and Processing Tools for Neurophysiologists.* (2020). http://neurosuite.sourceforge.net/

[9] Keras. *The Functional API.* Keras (2020). https://keras.io/guides/functional_api/

[10] Khairullah, Enas, Murat Aricanb, and Kemal Polatc. *Brain-computer interface speller system design from electroencephalogram signals with channel selection algorithms.* Medical Hypotheses (2020). DOI: 10.1016/2020.109690

[11] Maulik, U., Bandyopadhyay, S. *Performance evaluation of some clustering algorithms and validity indices*. Institute of Electrical and Electronics Engineers (IEEE) (2002). https://ieeexplore.ieee.org/abstract/document/1114856

[12] Masaad, Sarah, Safiya Jassim, Layla Mahdi, and Zouhir Bahri. *Versatile Brain-Computer-Interface for Severely-Disabled People.* International Journal of Computing and Digital Systems (2020). http://journal.uob.edu.bh/bitstream/handle/123456789/4043/1570641925.pdf?sequence=4&isAllowed=y

[13] Meier, S., A U Bräuer, B Heimrich, R Nitsch, N E Savaskan. *Myelination in the hippocampus during development and following lesion.* Cell Mol Life Sci (2004). DOI: 10.1007/s00018-004-3469-5

[14] Neuralink. *Neuralink Applications.* Neuralink (2020). https://neuralink.com/applications/

[15] Rossant, C., Kadir, S., Goodman, D. et al. *Spike sorting for large, dense electrode arrays.* Nat Neurosci 19, 634–641 (2016). DOI: 10.1038/nn.4268

[16] SciPy Cookbook. *Kalman filtering.* SciPy Cookbook (2018). https://scipy-cookbook.readthedocs.io/items/KalmanFiltering.html

[17] Seif, George. *The 5 Clustering Algorithms Data Scientists Need to Know.* Towards Data Science (2020). https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68#:~:text=Clustering is a Machine Learning,point into a specific group.

[18] Shah, Nishal P., Nora Brackbill, Colleen Rhoades, Alexandra Kling, et al. *Inference of nonlinear receptive field subunits with spike-triggered clustering*. eLife Sciences (2020). DOI: 10.7554/eLife.45743

[19] Susuki, K. (2010) Myelin: A Specialized Membrane for Cell Communication. Nature Education 3(9):59 https://www.nature.com/scitable/topicpage/myelin-a-specialized-membrane-for-cell-communication-14367205/

[20] Swanson, Jerry W.. *Demyelinating disease: What can you do about it?* Mayo Clinic (2020). https://www.mayoclinic.org/diseases-conditions/multiple-sclerosis/expert-answers/demyelinating-disease/faq-20058521#:~:text=Multiple%20sclerosis%20(MS)%20is%20the,nerve%20fibers%20that%20it%20surrounds.

[21] Weiss, Matthew L, Paffenroth, Randy C., and Uzarski, Joshua R. *The Autoencoder-Kalman Filter: Theory and Practice.* IEEE (2020). DOI: 10.1109/IEEECONF44664.2019.9048687

[22] Weiss, Matthew L, Paffenroth, Randy C., Whitehill, Jacob R., and Uzarski, Joshua R. *Deep Learning with Domain Randomization for Optimal Filtering.* IEEE (2019). DOI: 10.1109/ICMLA.2019.00288

[23] Open Data Science (ODSC). *Assessment Metrics for Clustering Algorithms*. Open Data Science (2020). https://medium.com/@ODSC/assessment-metrics-for-clustering-algorithms-4a902e00d92d