Predicting motion from LFP Project Pre-Proposal

Jackson Cornell, Maxwell Rosenzweig, Sidharth Sunil

November 12, 2022

1 Background and Significance

Brain-Computer Interfaces (BCIs) are an emerging technology for medical applications such as neural rehabilitation for disabled individuals. BCIs allow for the processing of neural signals to either transduce an action or provide a feedback stimulus to the brain. For our project, we will consider the processing of Local Field Potential (LFP) signals to predict hand and arm movement captured by accelerometer data. We can use this predictive model to translate the thought of movement and resultant LFP into a variety of outcomes such as movement of a prosthetic, control of a cursor on a screen, or interaction in a Virtual Reality (VR) space, among other applications of the produced accelerometer data.

These particular outcomes are significant for a variety of applications: A prosthetic could be developed for individuals who have lost a limb; a cursor control system could help someone who has lost control or limited control of their limbs; commercial applications such VR could be developed.

The aim of our project is to develop a BCI which will predict the desired movement (i.e. accelerometer data) given the corresponding LFP data. We will compare two approaches to decoding LFP data, the first using the spectral power of different frequency bands, and the other using the temporal dynamics. These will be decoded using a Convolutional Neural Network (CNN) and Kernel Least Mean Squares (KLMS) filter, respectively.

2 Preliminary Studies

One of the most persistent ongoing debates in computational neuroscience is if neural information is encoded in the frequency domain, the time domain, or a combination of

both. One of the most common processing techniques for neural signals is binning the spike train data into a spike train average (STA), which effectively computes the frequency over windows of time. Other methods use adaptive filters such as the Kalman or Wiener filter to encode the time-domain dynamics of the signal to predict an output. In this paper, we compare the performance of two separate models, the CNN and the KLMS, which will decode the LFP signals from their frequency and time domain representations, respectively.

Convolutional neural networks have gained considerable attention in recent years with many applications ranging from natural language processing to medical image analysis and image classification. Essentially, CNNs are multi-layer neural networks composed of several convolution-pooling layers followed by nonlinear activation units and fully connected layers. These layers are stacked such that the input data is passed through all connected layers to drive the output of the network. [1]

Wiener filters are a widely used algorithm for estimating the dynamics of a system and predicting the outcome given some input. In practice, however, the Wiener filter falls short of expectations. The first reason being its assumption that the system it is modeling is wide sense stationary (WSS), and the second being its matrix inversion having a complexity of $O(n^3)$. For an algorithm expected to run on a resource-constrained BCI, both of these facets make it impractical to deploy. Luckily, 70 years of signal processing research has found many ways to circumvent these issues. [2]

Instead of doing a matrix inversion to find the optimal model parameters, these parameters can instead be found using stochastic gradient descent (SGD). SGD basically computes the MSE between the estimated signal and the actual signal, and uses this error to descend down the learning manifold to the optimal parameter values. Given the same assumptions the Wiener filter makes about the signal being WSS, the SGD approach is guaranteed to find the optimal parameters with $O(n)$ complexity. This approach is known as the Least Means Squares (LMS) filter. [2]

The next problem to solve is the fact that LFP data is not WSS. Kernel methods are known to be able to handle cases such as these. Kernel methods map the input data into a higher-dimension feature space where optimal weights can be found. This allows non-Gaussian and non-linear system's dynamics to be estimated. The Kernel LMS (KLMS) filter is a kernelized version of LMS filter, and can be used to train data on-line. We will use the KLMS to decode the time-domain LFP signal into accelerometer data. [2]

3 Research Design and Methods

3.1 Dataset

Our dataset comes from Oweiss Lab patient data. At the lab, patients are undergoing treatment for tremors using deep brain stimulation (DBS). Patients were instructed to do a variety of tasks, such as draw a spiral or move an object to a specified location. These tasks were repeated over a number of sessions. During each task, custom software called alpine recorded their movement with an accelerometer, an EMG using electrodes, and LFP data using an implant.

For the purposes of our study, we are only concerned with the resulting movement from neural activity. As such, we will be ignoring the splits of tasks in each session and the metadata for sessions (i.e, task type, date, other metadata). However, it is important that we process each session separately, as sensor placement and configuration may vary slightly between sessions.

We will split the dataset into a training and test dataset. Following the standard for constructing machine learning models, this will consist of 80% for training and 20% for testing. The testing set will be further split into several folds for cross-validation of optimal hyperparameters (this process will be repeated for each model).

First, we will split the dataset as described above. 80% of the sessions will make up the training set, and the remaining 20% will make up the testing set. The session identifiers will be stored in tables. We will use the provided ”read_data.m” program to import the data into matlab, and will feed the program the data from our tables to get the session data for further processing.

3.2 Data Pre-Processing

To increase model performance, we will employ low-pass filtering and downsampling of the data. All signals will be low-pass filtered with a bandpass of 0-220 Hz to remove high frequency artifacts. The signals will then be downsampled to 500 Hz to reduce computational complexity. This is shown to be an effective preprocessing strategy for LFP signals. [3]

Studies show that the gamma band of LFP signals have a strong correlation with reach goals [4]. For models using the PSD to decode trajectory, we will compute the PSD of the following frequency bands (delta band: 0 to 4 Hz; theta band: 4 to 8 Hz; alpha band 8 to 13 Hz; beta band: 13 to 30 Hz; low gamma band: 30 to 48 Hz; mid gamma band: 48 to 100 Hz; high gamma band: 100-200 Hz). These will be computed from windows

of 200 ms, applied every 50-100 ms [5]. The power of each band can be simultaneously computed using the short-time Fourier transform (STFT), which produces an image of the power of the frequencies and how they change over time.

For the kernel-based method, the trajectory can be decoded directly from the time domain signal [2]. Thus, no further preprocessing is needed after downsampling the signal.

3.3 Neural Decoding

3.3.1 Kernel Least Mean Squares Filter (KLMS)

There are few hyperparameters needed to run the KLMS. The first is the filter order, which can be derived from where the autocorrelation function (ACF) of the input data crosses zero. The other two are the kernel width and step size, which will each be found by using hyperparameter tuning described above. Below presents the algorithm for computing the KLMS, given these hyperparameters:

Algorithm 1 KLMS

Require: step size μ, filter order M, and kernel size σ initialized

$a^{(0)} \leftarrow 0$

for each vector of size M in x_n **do**

$f_i \leftarrow \sum_{n=1}^{N} a^{(i)} k(x_i, x_n)$

$e_i \leftarrow y_i - f_i$

$a^{(i+1)} \leftarrow a^{(i)} + \mu e_i$

end for

Where $k(x_i, x_j) = exp(-\sigma||x_i - x_j||^2)$

3.4 Convolutional Neural Network (CNN)

For this regression task, we will design a CNN in Python using the deep learning libraries TensorFlow and Keras. The CNN will consist of multiple convolution layers to create output feature maps. [6]

The model layers will consist of a spatial convolution layer, a non-linear activation function, batch normalization and max pooling layers to generate the feature maps. These feature maps will be flattened and passed through fully connected layers, dropout layers and a non-linear activation function to obtain the predicted accelerometer output values. The kernel size, stride, dropout rate, number of hidden layers and more will be initially defined but modified using Hyperparameter training.

3.4.1 Hyperparameter Tuning

CNNs need several hyperparameters to construct its architecture, some of which will be found using hyperparameter tuning, others will be found using knowledge of the data. Specifically, since we are essentially performing regression rather than classification of the data, we will use a linear activation function on the output. For the hidden layers, leaky ReLU is known to perform well for CNNs. Hyperparameter tuning will need to be carried out to determine quantities such as kernel size, dropout rate, number of hidden layers, batch size, learning rate, and more. Luckily, python libraries such as Sklearn and Tensorflow have many methods that allow easy tuning of such parameters.

3.4.2 Training

The tensor dataset obtained after preprocessing the raw LFP data will be of the shape NxCxHxW, where N is the number of Spectrograms, C is the number of channels(i.e., 1 for the spectrogram data), H and W are the height and width of the image. The dataset will be split following a traditional 80/20 split for ML models, where 80% of the data is for training the model and 20% of the data is for validating the model.

The input to the CNN will be an input tensor of the Spectrogram of LFPs in batches, each individual spectrogram can be considered as a single channel image over a window of time. The output of the data generated will be a three-dimensional vector corresponding to the three axes of acceleration (x, y, z).

The weights and biases of the CNN are calculated using a back-propagation algorithm, where the labeled training set is exposed to the network and the error e between the desired output and the predicted output by the network is calculated [7]. The loss function used to calculate the error will be the Mean Squared Error (MSE) function and the Adam optimization algorithm will be used to modify the training weights. The optimal number of epochs and the learning rate for training the network will be defined using hyperparameter tuning.

3.5 Performance Analysis

The predicted accelerometer data will be compared to the actual accelerometer data to produce an error measurement, which will be used to assess their performance as well as calculate the model parameters. Once model training is complete, the models will be evaluated using the test set. Other metrics that will be considered are the computational cost (how long it takes each model to process an input) and memory usage (how much

memory does the model use to process an input). With these metrics, we will be able to compare the performance and accuracy of these different models, and–more specifically–which ones will be best suited for use in a BCI.

We will use a least squares error to compare the predicted accelerometer data to the session data from the test set. Because the predicted data and session data will have different sampling frequencies, we will compare the downsampled session accelerometer data to the predicted accelerometer data. We can then graph this error over time to compare the models, as well as compute the average error and variance of the error for the models to compare accuracy and precision as well.

For computational cost, we will time the models on each test session and compare the average runtime. Similarly, we will measure and compare average memory usage.

4 Milestones, Metrics of Success, and Timeline

We aim to complete the following milestones by the following dates.

Date	Milestone
11/20	Split dataset and get data into Matlab
11/27	Implement KLMS
11/27	Implement tests for models (Accelerometer LMS, Time usage, Memory usage)
12/4	Implement Preprocessing
12/4	Implement CNN
12/11	Finish Paper

To measure how successful our models are, we need a model with low average error and low variance in error. This translates to better movement of a prosthetic, for example. As an estimate, the average error plus three times the standard deviation of the error should not exceed $5cm/s^2$.

Additionally, we will compare the computational cost of the two models. The time it takes the models to evaluate a test point will be recorded, and the resulting averages will be compared to see which model has faster evaluation times. A benchmark of 100 ms will be used for model evaluation time, as this is roughly half of what studies indicates humans notice for delays in movements of prosthetics.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] L. Li, *Kernel based machine learning framework for neural decoding*. University of Florida, 2012.

[3] N. F. Ince, R. Gupta, S. Arica, A. H. Tewfik, J. Ashe, and G. Pellizzer, "High accuracy decoding of movement target direction in non-human primates based on common spatial patterns of local field potentials," *PloS one*, vol. 5, no. 12, p. e14384, 2010.

[4] E. J. Hwang and R. A. Andersen, "Spiking and lfp activity in prr during symbolically instructed reaches," *Journal of neurophysiology*, vol. 107, no. 3, pp. 836–849, 2012.

[5] S. Zhang, P. K. Fahey, M. L. Rynes, S. J. Kerrigan, and J. Ashe, "Using a neural network classifier to predict movement outcome from lfp signals," in *2013 6th International IEEE/EMBS Conference on Neural Engineering (NER)*, pp. 981–984, IEEE, 2013.

[6] M. Angrick, C. Herff, G. Johnson, J. Shih, D. Krusienski, and T. Schultz, "Interpretation of convolutional neural networks for speech spectrogram regression from intracranial recordings," *Neurocomputing*, vol. 342, pp. 145–151, 2019.

[7] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, Elsevier, 1992.