Analysis of Machine Learning Methods for Decoding the Direction of Movement Using Neural Data from the Motor Cortex of a Non-Human Primate

Saketh Ayyagari, Omar Tawakol

Abstract

Parkinsons, arthritis, and tremors are motor complications many people face as they become older. While advanced prosthetics can substantially mitigate these issues, many require both complex surgeries and a lot of time to get used to. To facilitate the implementation of prosthetics, we present several computational methods for performing neural decoding from a population of neurons in the motor cortex of a non-human primate. We first analyzed the accuracy of traditional methods, specifically the population vector and maximum likelihood estimation (MLE); then we developed and compared different machine-learning models, such as feedforward neural networks, random forest classifiers, and support vector machines (SVMs), and compared their accuracies to those of the more traditional methods. These findings shall help accurately decode the direction of movement in a mammal, which could be used in developing brain-computer interfaces and prosthetics.

Keywords--Neural decoding, Decoding movement, Neural network, Random forest, Maximum likelihood, Population vector, SVM, PCA

Introduction

Each year, more adults develop motor problems that impact their lives. Out of the 25% of adults that have some disability, 12.2% of them have difficulty with motor tasks including walking and climbing stairs [1]. These issues can make their way into other disabilities: for example, mobility issues make it difficult for adults to live independently or care for themselves. While many prosthetics and brain-computer interfaces (BCIs) are effective in helping adults regain mobility, the use of software can greatly assist this process as the time it takes to get accustomed to the prosthetic can be greatly shortened. This software uses a process known as neural decoding to decipher brain signals to determine a potential auditory, visual, or motor response [2].

In the brain, the motor cortex is responsible for direct movement. Each neuron in this region is specifically tuned to handle a specific task; for example, some neurons are tuned for direct licking in different directions, while others are designed for planning this direct licking in those directions.

Several decoding algorithms have been explored for the purpose of neural decoding, such as the population vector and maximum likelihood estimation methods. The MLE algorithm chooses a point estimate that can best describe a population distribution of values [5]. Using knowledge of how a random sample is distributed, both a population mean and standard deviation can be determined to best fit the random sample of data. In neural decoding, this involves identifying the most probable direction of movement based on activity patterns from a given sample of

neurons, providing a probabilistic framework that works effectively when the distribution of neural responses is well understood.

The population vector algorithm relies on the concept that each neuron can be considered a vector pointing in its preferred direction, with the magnitude of the vector proportional to the neuron's firing rate. The algorithm computes a resultant vector that represents the intended movement direction by summing these individual vectors across a population of neurons along with their respective weights (how much of an impact they may have on the direction) [12].

This study aims to compare the accuracies of different algorithms to determine which ones can decode movement direction just as well--if not better--than both the MLE and population vector algorithms.

Methods

We utilized a dataset containing data from the motor cortex of a monkey controlling a joystick in 1 of 8 directions. As the monkey sees a direction in the form of an arrow on the screen, it moves a joystick in that same direction. Using a manipulandum, the number of action potentials (or "spikes") and the firing rate from a sample of 143 neurons was collected. For later use, each firing rate of each neuron during each trial is stored in a 158-row by 144-column Pandas DataFrame, where each row represents a trial, the first 143 columns represent each neuron with the final column containing the direction the monkey moved in during that trial.

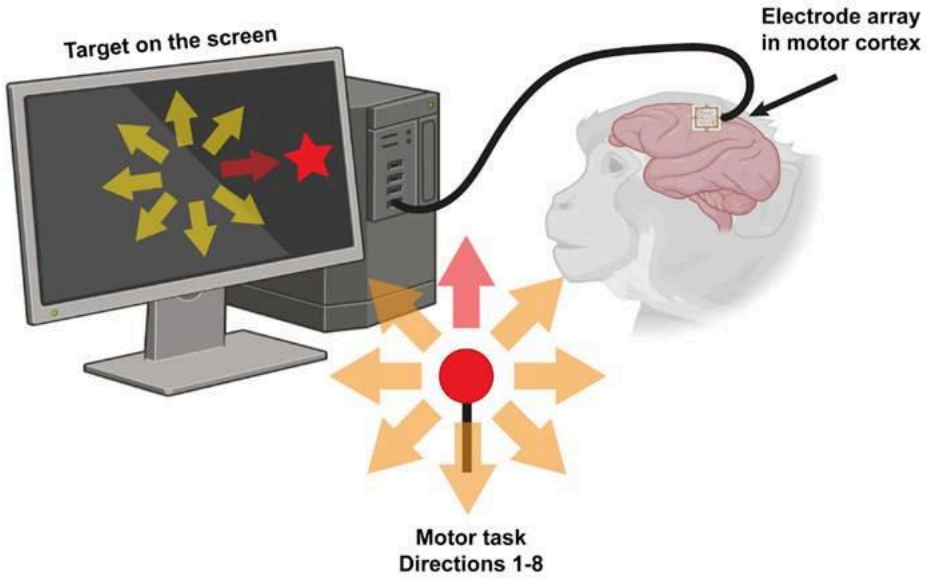

Figure 1: Visual diagram of how data was collected during the experiment.

Both the maximum likelihood estimation (MLE) and population vector algorithms are used as baselines we compare other models to, with both methods achieving accuracies of 99% and 100% respectively. We trained a support vector machine classifier (SVC), random forest

classifier, and a neural network with a dataset containing firing rates and spike counts of 143 neurons during a recording period for 158 trials.

Support vector machines work by finding an optimal line that separates data points of different classes in a high-dimensional space [6]. This line maximizes the margin, or the distance, between two data points of opposite classes, where each distance between the line and a nearby data point can be represented as a vector, and each class represents a unique direction. However, SVMs are designed to find the boundary between two classes [8]. Therefore, for all 8 possible directions, 8 optimal lines are determined, where each line classifies whether a set of firing rates from each of the 143 neurons will result in a monkey moving in a specific direction or not.

Random forest classifiers build multiple decision trees during training, each trained on a random subset of the training data and a subset of features. In the context of neural decoding, multiple decision trees can consider different features or characteristics that split the neuron data into clusters, making it easier to classify the direction of the monkey's movement. When making predictions, each tree votes for a predicted direction, and the majority vote determines the final prediction for the direction. This approach reduces overfitting, which is common in single decision trees, and enhances generalization to new data. [7].

Neural networks are inspired by the structure and function of the human brain and consist of layers of interconnected nodes, or "neurons." Each neuron receives inputs, applies weights and biases, processes the information through an activation function, and passes the result to the next layer [10]. The input layer takes in each neuron's firing rate and spike count, while the hidden layers apply specific weights and biases to each value. After the data passes through these hidden layers, the result is fit through an activation function, which is a modified softmax function that outputs a value between 1 and 8, where each number represents a specific direction. Then, each result is sent to the output layer, which consists of 1 neuron outputting an integer value between 1 and 8 (where each integer represents a separate direction). Eventually, each value is then rounded down using the *numpy.floor()* method as the directions are only represented as integers and not continuous floating points. Before training our neural network, the data was preprocessed using Principal Component Analysis (PCA) into two principal components to highlight features that can maximize the variance of our data, which could make decoding the movement easier [9]. Given the data we collected, the two principal components were known to be firing rate and spike count. Each component was created using the *scikit_learn.pca()* function.

We generated code for each model using Python in Google Colab. Our dataset was split into 80% training and 20% testing data to ensure our models would receive enough data and to prevent them from overfitting. The *scikit-learn* Python package was used to create the support vector machine and random forest classifier, as well as the *TensorFlow* package to create the neural network. We also used the Numpy package to create the modified softmax activation function. For each model, an 8x8 confusion matrix was generated using *scikit-learn.metrics*, which contains the frequencies of the actual direction the monkey was moving in compared to each model's predicted direction.

Results

Overall, our first neural network yielded an accuracy of 9% after training the model with the transformed data, with a 13% final accuracy after rounding our results down (as the direction must be represented as an integer). However, this resulted in a worse accuracy of 9% after flooring each predicted direction using *tensorflow.floor()*. The Random Forest, which was composed of 75 decision trees, and Support Vector Machine models, however, yielded accuracies of 78.125% and 53.125% respectively.

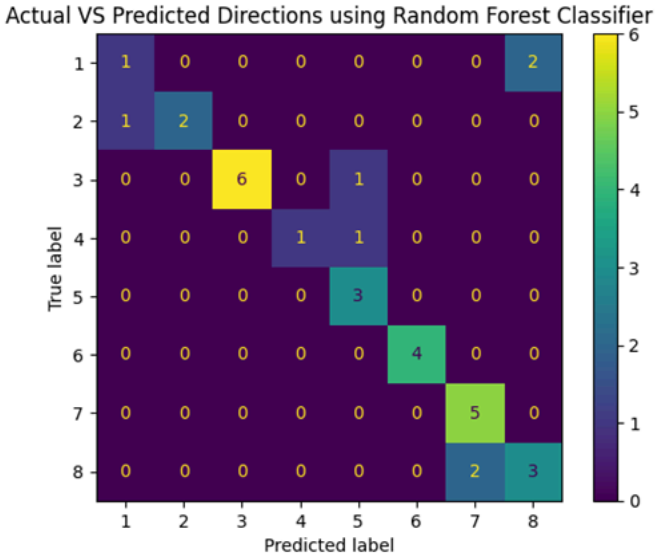

Figure 2: Confusion matrix comparing actual and predicted directions from random forest model.

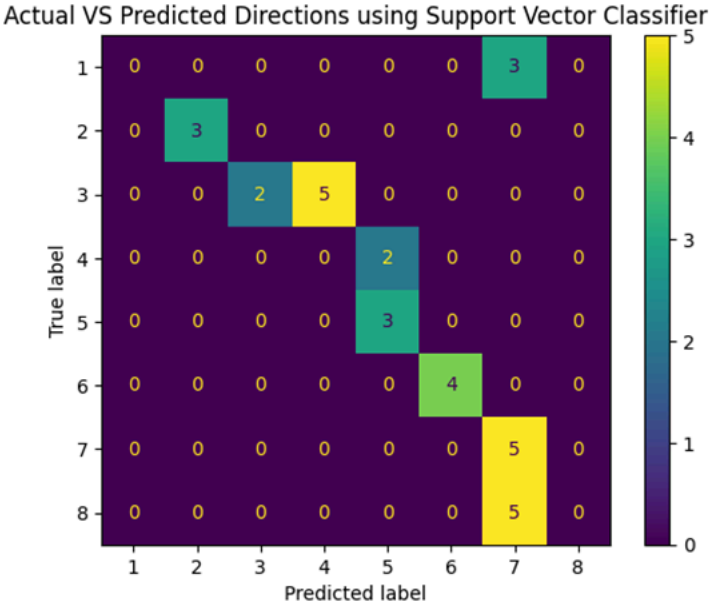

Figure 3: Confusion Matrix comparing actual and predicted directions from SVM.

Discussion

After transforming our data (which contains both the firing rate and spike count of each neuron) using PCA, we obtained covariance values for each principal component which explain the strength to which a principal component contributes to the decision. The ratio of covariance values of both principal components was 1:0, signifying the first principal component is the primary contributor to the direction. In other words, a neuron's firing rate contributes to the predicted direction, while the number of spikes it produces has no effect on the predicted direction. This result corroborates existing literature as neuron spike count is not considered when decoding neural patterns. After confirming that spike count has no effect on a brain signal, we reformatted our data to contain the firing rate values for all 143 neurons during all 158 trials.

When implementing the neural network, a primary issue encountered was creating the new activation function, which is a modified softmax function that outputs a value between 1 and 8, with each number representing a unique direction. Because each direction must be an integer, continuous floating points skew the accuracy to be lower as they do not match with actual directions. Because of this, we modified the softmax function into a step function, where each result is either rounded or floored using the *tensorflow.floor()* and *tensorflow.round()* methods. However, neural networks implement the use of gradient descent to minimize a cost, or loss, function, where a larger value represents a greater model inaccuracy [11]. Because our modified activation function only returns integer values, there are discontinuities at each actual direction value, meaning the gradient of the activation function at each point does not exist.

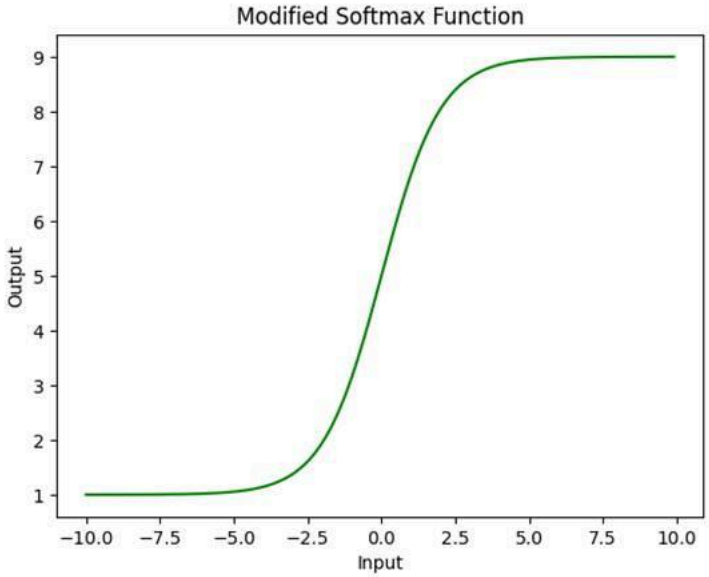

Figure 4: Softmax function used in the creation of the neural network

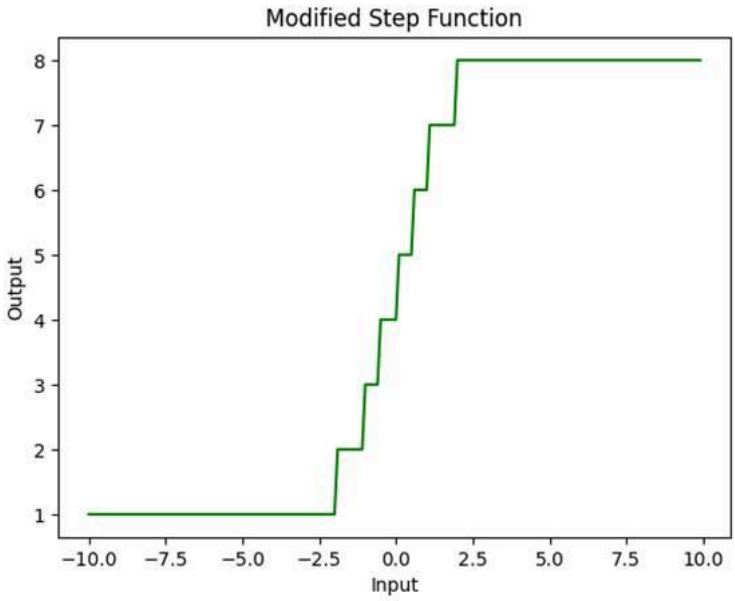

Figure 5: Custom step function for initial neural network design.

Additionally, the random forest and support vector machines are designed to directly classify a specific output while neural networks reach their final result after separate weights and biases are applied to each firing rate. As a result, the result becomes a continuous, floating point rather than an integer specifying the direction, leaving room for further error in the results.

One change that can be tested is to modify the structure of the neural network by modifying the output layer. Rather than having our result be outputted to one neuron and modified afterward, we could instead have the output layer consisting of 8 neurons, with each neuron outputting the probability that the firing rate could indicate a specific direction of movement. From there, the neuron with the highest probability can be indicated as our predicted direction. This method would also not require the use of a modified activation function.

Conclusion

We developed and tested three different machine learning models against the population vector and maximum likelihood estimation algorithms, two traditional methods used in neural decoding. While some machine learning models such as random forest and support vector machines do yield above-average accuracies when decoding brain patterns, more traditional methods like the population vector and maximum likelihood estimation algorithms prove to be superior to these other models. However, like with other models, these ones can potentially yield greater accuracies with further tuning. After further tuning, these models can be implemented into brain-computer interfaces and prosthetics, both of which can be controlled by people given their brain signals.

Acknowledgments

The author would like to thank Omar Tawakol for his creation of Figure 1, continuous guidance on improving the experiment, as well as anonymous reviewers for their feedback on this paper.

References

[1] Centers for Disease Control and Prevention (CDC), n.d. *Disability impacts all of us*. [online] Available at: https://www.cdc.gov/ncbddd/disabilityandhealth/infographic-disability-impacts-all.html [Accessed 20 Jan. 2025].

[2] ScienceDirect, n.d. *Neural decoding*. [online] Available at: https://www.sciencedirect.com/topics/veterinary-science-and-veterinary-medicine/neural-decoding [Accessed 5 Oct. 2024].

[3] Collaborative Research in Computational Neuroscience (CRCNS), n.d. *About ALM-4 data set*. [online] Available at: https://crcns.org/data-sets/motor-cortex/alm-4/about-alm-4 [Accessed 20 Jan. 2025].

[4] PubMed, 2015. *PMID: 25731172*. [online] Available at: https://pubmed.ncbi.nlm.nih.gov/25731172/ [Accessed 20 Jan. 2025].

[5] The Pennsylvania State University, n.d. *Introduction to Probability Theory*. [online] Available at: https://online.stat.psu.edu/stat415/lesson/1/1.2 [Accessed 20 Jan. 2025].

[6] IBM, n.d. *Support vector machine (SVM)*. [online] Available at: https://www.ibm.com/topics/support-vector-machine [Accessed 20 Jan. 2025].

[7] IBM, n.d. *Random forest*. [online] Available at: https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems. [Accessed 20 Jan. 2025].

[8] Roth, D., n.d. *Multiclass classification*. [online] Available at: https://www.cis.upenn.edu/~danroth/Teaching/CS446-17/LectureNotesNew/multiclass/main.pdf [Accessed 20 Jan. 2025].

[9] IBM, n.d. *Principal component analysis (PCA)*. [online] Available at: https://www.ibm.com/think/topics/principal-component-analysis [Accessed 20 Jan. 2025].

[10] IBM, n.d. *Neural networks*. [online] Available at: https://www.ibm.com/think/topics/neural-networks [Accessed 20 Jan. 2025].

[11] Towards Data Science, 2018. *Step-by-step: The math behind neural networks*. [online] Available at: https://towardsdatascience.com/step-by-step-the-math-behind-neural-networks-490dc1f3cfd9 [Accessed 20 Jan. 2025].

[12] Frye, C., n.d. *Foundational Neuroscience - Chapter 49*. [online] Available at: https://charlesfrye.github.io/FoundationalNeuroscience//49/ [Accessed 12 Feb. 2025]