Fast Direction Decoder for Brain Computer Interfaces

Sean Metzger (seanmetz@stanford.edu) and Jinhie Skarda (jskarda@stanford.edu)

1. Project Goal and Motivation

1 in 50 Americans is paralyzed. Brain computer interfaces use neural signals to control devices like prosthetics or cursors on computer screens, thus offering enhanced mobility and quality of life for this significant proportion of the population that is unable to move. The goal of this project is to improve the accuracy of fast classification algorithms for brain computer interfaces that quickly predict the direction someone is reaching based on real-time electrode data from their primary motor cortex. In the context of a brain computer interface that uses neural signals to control a computer screen cursor, we would want our algorithm to quickly and accurately predict the direction the user wants to move the cursor and move the cursor to that point instantly for faster, easier brain computer interface use. This technology can be translated to help paralyzed people since when they think about reaching somewhere, even if spinal cord lesions block signals from reaching their arm, the motor cortex signals are the same as if they were actually reaching. In this project, we attempt to beat the accuracies of previous direction prediction approaches by using two softmax-based models that take the time-history electrode data as inputs and output the intended reach direction. The two models are Softmax Regression and a layered Neural Network with a Softmax output layer. Our primary motivation for implementing the Softmax Regression model is to take advantage of the simplicity to gain insight into the performance of Softmax-based models for this classification task, while our primary motivation for implementing the layered Neural Network is to take advantage of the increased complexity to achieve as high a test set accuracy as possible.

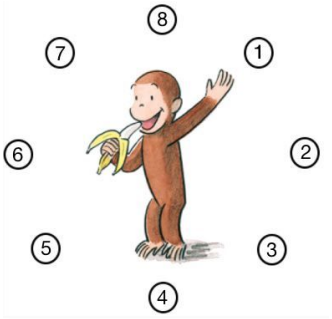
2. Related Work

Brain computer interfaces are an exciting field with interesting and ongoing research. There has been success implementing typing interfaces for people with paralysis [1]. However, typing with these interfaces can be a slow process. Patients can type with a speed of 24.4 correct characters per minute, compared to the 190-200 characters per minute an average person can type [2]. Most current research on brain machine interfaces focuses on controlling cursor position [3]. However, we sought to improve the accuracy of final cursor position prediction given a short range of neural time-history data to speed up neural typing mechanisms. Previous methods for discrete decoding of signals from primary motor cortex during a 'delay period' during a center-reach task included factor analysis and MAP probability estimates which described spike trains as gaussian or Poisson-based processes [4][5]. Both of these methods used time-series data covering 150 ms to 500 ms in the planning period before a reaching task was performed to predict the direction of the reach. Factor analysis methods are the current state of the art and predicted the planned reach direction with an accuracy of 94.5%. Poisson based MAP estimates predicted the planned reach direction with an accuracy of 91.7% [4]. Being able to more accurately predict the direction of an intended reach using only data from within the first 500 ms of the planning period would help speed up neural typing mechanisms [5]. Since the problem of predicting one of eight reach directions seems well-suited to softmax classifiers, we wanted to see if we could achieve higher accuracies by applying softmax regression and deep learning with the softmax function to time-series data covering 150 ms to 500 ms in the planning period of a reach. From our comprehensive literature review, softmax-based classification approaches have not been previously studied in this context of predicting a planned reach direction based on real-time motor cortex data.

3. Dataset and Features

We are using a dataset generated by Krishna Shenoy's group that contains motor cortex electrode data from a monkey during the time period between which a target in one of eight possible directions (Fig. 0) is illuminated and the monkey begins to reach towards it [3]. Thus, the dataset contains time-series information about the planning period during which the monkey is thinking about performing the reaching task. This time-series dataset consists of data from 98 electrodes sampled at 1 millisecond resolution over a period of 700 ms. For each neuron, each data

point in the raw time series data is a 1 if the given neuron fired over the given 1 ms interval and a 0 otherwise. We don't need resolution smaller than 1 ms due to the refractory period of neurons.



The dataset contains 125 trial sets, each containing 1 trial for each of the 8 reach directions. Thus, it is a balanced dataset with 1000 total individual trials. Each trial consists of 98 time-series from each of the 98 electrodes. Then, given that neural data is most often encoded in frequency of firing rate (since action potentials cannot change amplitude), we binned the time-history data for each probe by summing the data in each bin width so the data is counts of how many spikes a given probe registered within the binning interval. When performing the binning, we only use the section of the raw time-history data from 125 ms to 500 ms so we can compare our results with the previous results from [3] that perform factor analysis on similar neural planning data using only the time-history data in this range.

***Fig. 0**. Indexing of reach directions*

In order to encode both the dimensionality of the neurons and of the time-series, we consider all of the data from the 98 probes from each trial as a separate feature and stack the binned data from each probe to create a long vector with length 98*bin_num for each trial. Thus, our matrix of features is size (98*bin_num) X (num_trials). This bin width, along with the learning rate, hidden layer dimensionality, and regularization parameter, is a hyperparameter for our models that we tune using hold-out cross validation. We split the 125 trial sets using 60% for the training set, 20% for the dev set, and 20% for the test set. This gives us 600 individual reach direction trials for the training set and 200 individual reach direction trials for the dev and test sets.

4. Methods

Since our goal is to build a classifier in which the response variable is one of the 8 allowed directions, we chose to use Softmax classification. The most simple model we tried was Softmax Regression. Softmax Regression is a generalization of logistic regression to multinomial data distributed according to a multinomial distribution. The softmax function, which gives the probability that the given example is of the i^{th} class, is given by:

$$P(y = i|x;\theta) = \frac{e^{\theta_i^T x+b}}{\sum_{j=1}^{k} e^{\theta_j^T x+b}} \qquad \textit{Equation 1}$$

The parameters of the model are the weights (*theta*) and the bias (*b*). Given these parameters, we assess the performance of the model using the cross-entropy loss computed by testing the model on the dev set. The cross-entropy loss is given by Equation 2. We computed updates for the weights and bias using stochastic gradient descent on the cross-entropy loss function.

$$J(\theta, b) = -1/m \sum_{i=1}^{m} \sum_{k=1}^{k} y_k^{(i)} log(\widehat{y^{(i)}}_k) \qquad \textit{Equation 2}$$

Then, to push our test set accuracy higher, we increased the complexity of our model and built a neural network with a Softmax output layer. Since the Softmax Regression can be thought of as a one layer neural network, we simply added another hidden layer. We used a sigmoid activation function for the hidden layer of our neural network. Again, we used the average cross-entropy loss as given in Equation 3, which we again minimized using stochastic gradient descent.

$$J(W^{[1]}, W^{[2]}, b^{[1]}, b^{[2]}) = -1/m \sum_{i=1}^{m} \sum_{k=1}^{k} y_k^{(i)} log(\widehat{y^{(i)}}_k) \qquad \textit{Equation 3}$$

We tuned our hyperparameters using hold-out cross-validation. We systematically swept across possible values for the learning rate, bin width, regularization parameter, and hidden layer dimension for the neural network model, and compared the obtained dev set accuracies. In order to have control over tuning our hyperparameters and analyzing the performance of our models, we implemented both the Softmax classifier and the neural network ourselves in Matlab rather than using the built-in functions. Our final hyperparameters are given in the table below. We swept over values of the learning rate with a difference of one order of magnitude, regularization constants by half an order of magnitude, binwidth by checking binwidths of size 25-125 ms by a step size of 25 ms, and for the neural network model, hidden dimension size by checking between 1 and the number of input layers via a step size of 50.

Model	**Learning rate**	**Regularization constant**	**Bin width**	**Hidden dimension size**
Softmax Reg.	0.01	0.5	25 ms	n.a.
Neural Network	1.0	.0001	50 ms	350

Finally, we used Principal Component Analysis (PCA) in order to visualize our high-dimensional data so we could better understand our dataset and model performance. PCA finds the set of components that maximizes the variance when the data is projected onto this subspace. It is thus a nice way of reducing the dimensionality of a dataset to more easily see trends and separability. The components that maximize the variance, the Principal Components, maximize the following equation:

$$1/m \sum_{i=1}^{m} (x^{(i)T} u)^2 = u^T (1/m \sum_{i=1}^{m} x^{(i)} x^{(i)T}) u \quad \textit{Equation 4}$$

5. Model Performance:

First, we investigated the accuracies we were able to achieve with our Softmax classifier and neural network model and compared these accuracies to the results obtained with factor analysis in [3]. For both the Softmax regression classifier and the neural network model, we tune the hyperparameters using the dev set and then retrain the model on both the train and dev sets before testing it on the test set. We first tried training both models without regularization and we found that we achieved very high training set accuracies but lower dev set accuracies, indicating that we were overfitting the training set. We then trained both models with regularization implemented, adding the regularization parameter to the set of hyperparameters tuned on the dev set. We verified that using large regularization parameters led to extreme underfitting to ensure the regularization was affecting the results but found that regularization using the tuned regularization parameter only slightly helped with the overfitting problem. We also found that for the test set we were using, we got better accuracy than for our dev set.

Model	**Train Accuracy**	**Dev Accuracy**	**Test Accuracy**
Softmax Regression without regularization	1.0	0.940	0.910
Softmax Regression with regularization	1.0	0.950	0.915
Neural Network without regularization	1.0	0.940	0.975
Neural Network with regularization	1.0	0.950	0.975

Despite the overfitting issue, we were overall quite pleased with the test set accuracies our models achieved. Comparing to the results of the models in [3], we find that our simple Softmax Regression model obtains about the same accuracy as the 91.7% accuracy of their Poisson-based method but worse accuracy than the 94.7% accuracy of

their factor analysis model. Our Neural Network model, however, does much better on the test set than our simple Softmax Regression classifier. Its 97.5% test accuracy is higher than both the factor analysis and the Poisson-based methods from [3].

6. Analysis of Softmax-based Approach:

Since both of our models are built on Softmax, we then studied the simple Softmax classifier model in more detail in order to gain more insight into how these Softmax-based models could be improved.

A. Investigation of Softmax Regression Model Weights

We began by studying the weights learned in our Softmax regression classifier. We found that the time bins around the beginning, middle, and end of the 150-500 ms planning interval were most heavily weighted, suggesting they are more important than the other time intervals in predicting the planned reach direction. This suggests that shrinking the amount of planning interval time used for these Softmax-based models could degrade the performance. We also found that neurons 32-98 were weighted more heavily weighted than neurons 1-31, suggesting they contribute more significantly to the direction prediction. Since this means different neurons contribute differently to the reaching task, sampling more neurons could allow for the construction of an optimal neuron set that achieves far higher accuracies.

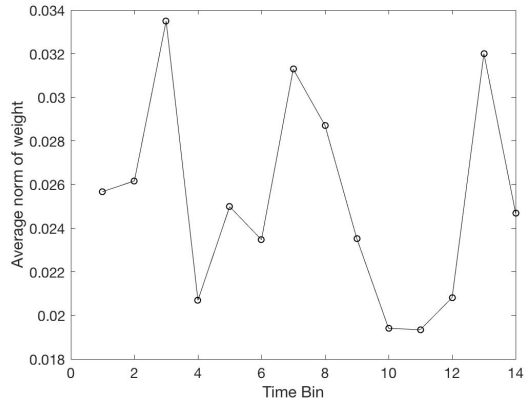


Figure 1. *Average norm of weights learned in our Softmax classifier with bin width = 25ms*

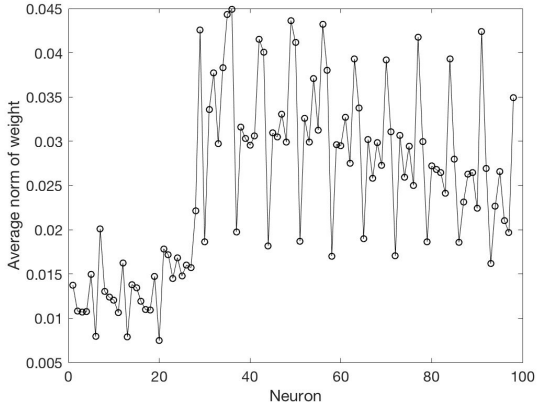
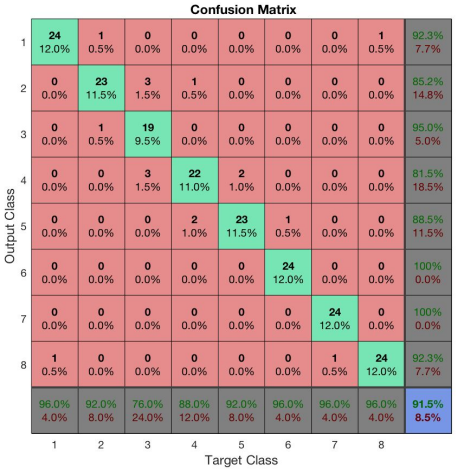


Figure 2. *Average norm of weights vs. neuron for our softmax classifier.*



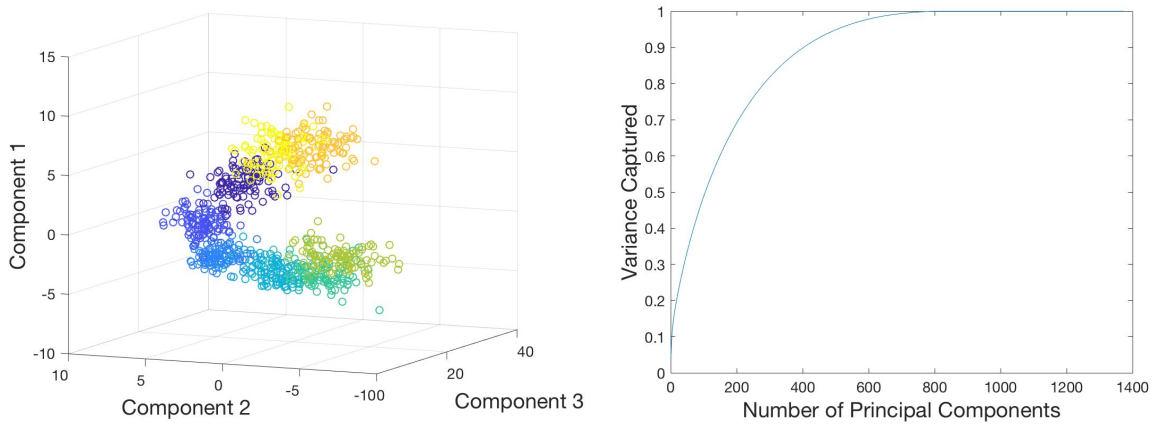
B. Error Analysis for Softmax Regression Model:

We then used the confusion matrix obtained by running our Softmax Regression classifier on the test set to determine which directions are hardest for the model to classify. We found that directions 2, 3, and 4 had the highest error rates. Looking at Fig. 0, we can see that these three directions are all on the same side of the monkey's body. Thus, the model may have a harder time classifying the neural signals for these directions because they are easier reaches to perform than reaches across the monkey's body and thus have less differentiating structure than more complex reach directions.

Figure 3 (left): confusion matrix with softmax regression classifier.

C. Visualizing Data with PCA

To further investigate differences between the neural signals for each of the 8 reach directions, we visualized our whole training set (train and dev) projected onto the first three Principal Components and colored each point by the planned reach direction. We found that, in this space, the points for each reach direction clustered together generally next to their neighboring directions. The exceptions were directions 6 and 7, whose point clusters were the furthest apart. Out of the eight reach directions studied, these two reaches are the farthest and most complex reaches the monkey, using its right hand, must plan. This further supports the idea that more complex reaches have significantly different neural signals than less complex reaches.



(a) (b)

(a) Projection of data onto first three principal components with bin_width = 25 ms. Legend: Purple = direction 1 , blue = 2, light blue = 3, aqua = 4, green = 5, brown-green = 6, orange = 7, yellow = 8. (b) Variance vs. Number of principal components.

7. Conclusions and Future Work:

Our Neural Network model with a Softmax layer achieved a higher test set accuracy than both the previously studied Poisson MAP and Factor Analysis approaches when trained on the same time segment of motor cortex data during the planning period before a reaching task. Our simpler Softmax Regression model had lower test set accuracy than the Factor Analysis approach and comparable accuracy to the Poisson MAP approach. By studying the weights and errors from this model, we were able to gain insights, such as the fact that certain neurons are more important to the direction prediction and that cross-body reaches are harder to predict, that could help improve the accuracy of Softmax-based approaches. Visualizing our high-dimensional dataset by projecting it onto the first three Principal Components provided further evidence that cross-body reach directions have a different neural signature than simpler same-side reaches.

In the future, it would be very interesting to further investigate the differences between the neural signals associated with simpler and more complex reach directions in order to help increase the accuracy of the direction classifier. Continuing analysis of which neurons contribute most to decoding which directions in order to create a detailed map of which parts of the motor cortex are responsible for which reach directions could also help improve accuracies. A direction decoder that is accurate even given only short time segments of the neural signal from the planning period could then eventually be extended to help make a fast brain computer interface keyboard. The eight directions could be increased to twenty-six and the keyboard would then be represented as a circle with letters distributed evenly around it. The algorithm would quickly determine which letter the user intended to type by using the time-history electrode signal to predict the direction they were picturing reaching and snap to it quickly, allowing for fast, brain controlled typing.

Contributions:

Jinhie implemented and tuned the Softmax Regression model in Matlab, and did the analysis of the model weights and errors. Sean implemented and tuned the Neural Network model in Matlab, and implemented PCA for visualizing the dataset. We worked together on preprocessing the data (binning and assembling the matrices) and the interpretation of the results.

References:

[1] Pandarinath C, Nuyujukian P, Blabe CH, Sorice B, Saab J, Willett F, Hochberg LR, Shenoy KV, Henderson JM (2017) High performance communication by people with paralysis using an intracortical brain-computer interface. *eLife*. 6:e18554.

[2] Arif AS, Stuerzlinger W. (2009) Analysis of Text Entry Performance Metrics. *Proceedings of the IEEE Toronto International Conference–Science and Technology for Humanity* (TIC-STH '09).

[3] Gilja V, Nuyujukian P, Chestek CA, Cunningham JP, Yu BM, Fan JM, Churchland MM, Kaufman MT, Kao JC, Ryu SI, Shenoy KV (2012) A high-performance neural prosthesis enabled by control algorithm design. *Nature Neuroscience*. 15:1752-1757.

[4] Santhanam G, Yu BM, Gilja V, Afshar A, Ryu SI, Sahani M, Shenoy KV (2009) Factor-analysis methods for higher-performance neural prostheses. *Journal of Neurophysiology*. 102:1315-1330.

[5] Santhanam G, Ryu SI, Yu BM, Afshar A, Shenoy KV (2006) A high-performance brain-computer interface. *Nature*. 442:195-198