FINAL REPORT FOR INDEPENDENT STUDY IN P300 SPELLERS: UNDERSTANDING THE SUBJECT PROGRESS AND INVESTIGATING SOME REPRESENTATIVE SOLUTIONS

SUPERVISOR: PROF. LESLIE COLLINS, PROF. BOYLA MAINSAH

Zion Sheng (NetID: zs144)
Department of ECE, Duke University
Durham, NC 27708
zion.sheng@duke.edu

ABSTRACT

Brain-computer interface (BCI) is a broad collection of machines that can facilitate the interaction between human brains and external devices. Generally, each type of BCI is designed to work on a specific task, where the P300 speller can help users in spelling. By classifying whether received electroencephalogram (EEG) signals contain a special event-related potentials (ERP) called P300, it can infer the user's desired character. Among all the traditional classification algorithms, stepwise linear discriminant analysis (SWLDA) reaches the state-of-the-art performance, and it can be further enhanced by some language models. Nevertheless, with the emergence of many powerful deep learning techniques, recent studies have turned to exploring the applications of deep neural networks on P300 spellers. This report tries to deliver a concise overview of the freshly-finished independent study. It will start from the problem description and subject progress, and then proceed to elaborate on the two competitive solutions: one is SWLDA combined with an 2-gram language model, and the other is EEGNet, a convolutional neural network tailored for P300 BCIs. The two methods are implemented and tested on the real data from 17 participants. Results demonstrate that EEGNet has better generalizability that is unprecedented in SWLDA, but the latter still achieves a higher spelling accuracy on average.

Keywords EEG signal processing, brain-computer interface, P300 speller, machine learning

1 Introduction

1.1 Background

Recently, Brain-computer interfaces (BCIs) have gained increasing attention as an alternative human-machine communication tool. Many BCIs are reported to have the potential to augment normal people's perception ability and extend physical limits. But more importantly, many BCIs can

help to restore control/communication ability and mobility to disabled people. Among these types of BCIs, the P300 speller is targeted at individuals with severe neuromuscular injuries who can not communicate effectively by speaking, writing, or typing in a regular manner. Typical users of P300 spellers are Amyotrophic lateral sclerosis (ALS) patients, especially in the late stage of the disease (commonly known as the "locked-in" stage) as the nervous system has lost most muscle control Moghimi et al. [2013]. These patients cannot move any part of their body, including vocal cords, fingers, and eyes, to complete even a simple spelling task in any language. However, as long as the brain and cognitive ability are in normal operation, there is still hope. The first P300 speller was created by Farwell and Donchin [1988], which enables the user to "type" on a virtual keyboard on the screen by utilizing a special EEG signal called the P300 waves. Based on this pioneering work, later scientists proposed many new designs by applying better hardware, improving the inner algorithms, and adapting the display settings, but the general framework of the P300 spellers didn't change much.

Before starting to elaborate on the general framework, it is necessary to give a brief explanation of what is the P300 wave and why it can be used on these machine-assisting spelling tasks. The fundamental biological principle behind all BCI is that the brain (or generally, the neural system) generates electrical signals/waves along with all kinds of neural activities, where different stimuli (inputs) and intended reactions (outputs) will lead to various responses, i.e., different shapes of the signals/waves. These signals/waves depending on the preceding events are called event-related potentials (ERPs), and the P300 waves also fall in this category. The P300 waves occur due to a ubiquitous neural activity mechanism called the oddball paradigm, which happens when the subject detects the appearance of a rare stimulus (target) out of some common stimulus (non-targets) presented more frequently. Such a mechanism will lead to two different brainwaves depending on the type of the received stimuli. Figure 1 illustrates a representative waveform of the P300 wave (red line) VS non-P300 wave using the data from one of the participants in the experiment. More details about the data will be given in the later sections, but the visualization here is already sufficient to pass the intuition that the P300 wave is distinguishable from non-P300 wave as there is a positive deflection in voltage occurs around 200 ms - 400 ms after the target stimulus is presented (when `Time = 0s`), while the non-P300 waves remain regular after the non-target stimulus onset.

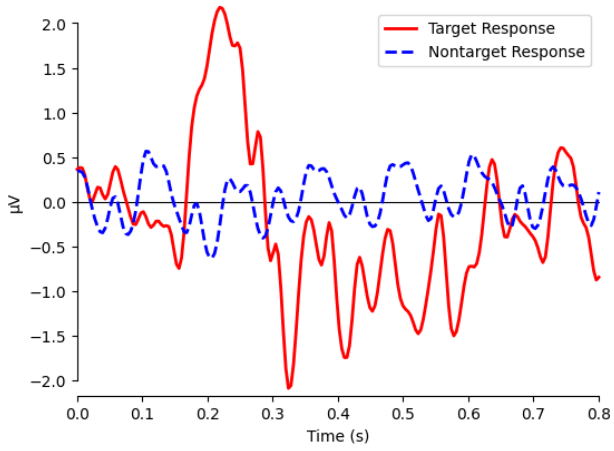


Figure 1: Waveforms of P300 waves (elicited by target stimuli) and non-P300 waves

This useful mechanism can be applied to the settings of the visual display of a keyboard, commonly used as the interface of the P300 speller. Subjects are asked to focus on their desired character

on the keyboard (only one at a time) when a random subset of these characters gets flashed every time. Suppose there are N characters in total and the subset size is k, then the probability of the target being flashed is around $\binom{N}{k-1}/\binom{N}{k} = \frac{k}{N-k+1}$. Therefore, such a rare event can elicit a P300 wave every time it happens. Theoretically, the desired character should be the one eliciting the most number of P300 waves, so the P300 speller can infer it after a certain amount of flashes.

1.2 General Framework of P300 spellers

There are three key components in the general framework that apply to every P300 speller: a screen to show the keyboard, a headset to collect EEG signals, and a computer to serve the communication between subjects and the keyboard. The connection of these components is shown in Figure 2.

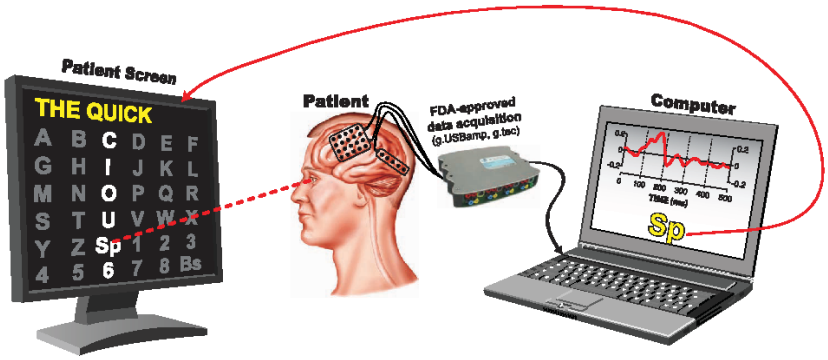


Figure 2: The general framework of P300 spellers (credit: Brunner et al. [2011])

The general process is that, the computer first acts as the scheduler of the character display by picking a subset of characters to flash every round with a fixed time interval. There are several popular flashing paradigms, and in our experiment, we choose the "RC" (row-column) paradigm, which flashes a whole row or column in each round. Users are asked to focus on the desired character and count how many times it gets flashed. By doing so, the P300 wave will be generated once the targeted character is in the flashed row or column. Users also need to wear a headset to monitor the EEG signals from different areas on the brain surface. These sensors are not invasive, and we can only collect signals from certain areas where the signals are strong, although the noise in data still needs further processing. Next, the data will be transmitted to the computer where a pre-trained user-specific classifier is deployed to decide whether a P300 wave occurs. The computer will keep track of the decisions and accumulate them as a score for each character. Once it is confident to infer the desired character, it will spell it on the screen, marking the end of a cycle. By repeating the above steps, users can spell a complete word or sentence. The inference tends to be more accurate with more flashes (or called "trials"), but the side-effect is that users will gradually get bored and distracted. Clearly, there is a trade-off between the inference accuracy and efficiency. Another concern is that every row and column should be flashed by the same amount of times to avoid any unfairness, so the total flashes to spell one character are usually measured as s sequences, where one sequence is flashing all rows and columns once.

1.3 Our work

Based on this basic framework, there are many flexibilities in the actual design. Past studies have extended and upgraded the P300 spellers from the very first one (Farwell and Donchin [1988])

to a huge family where each member has some valuable features. Elshout [2009] and Fang et al. [2021] respectively provide a comprehensive review on the progress of P300 spellers. By learning this history, we can see the evolutional path of the improvements in hardware, in displaying settings, and especially in classification and inference algorithms since the emergence of deep learning-based techniques in the last decade. Serving as the final report for this independent study, we will specifically focus on two representative P300 spellers in history. They are different in the choice of signal-classification and character-inference algorithms, but both are reported to achieve a competitive performance. One of them uses step-wise linear discriminant analysis (SWLDA) combined with Bayesian inference, whereas another one is powered by a convolutional neural network (called EEGNet) with simple counting. The first one is the generative probabilistic model as we need to estimate the conditional distribution of the target class and non-target class. On the contrary, the second one is essentially a discriminative neural network since it only outputs the decision results. In addition, it is also worth mentioning that the first one can be further enhanced using a 2-gram language model to utilize the previously spelled character. We will elucidate all these technical details in the following sections.

In this report, we first go through the related papers to figure out the details of the two methods, which will be covered in Section 2. Then, we implemented the algorithmic part of the two methods in Python and tested them on the dataset we borrowed from Mainsah et al. [2014] (we received the permission of the original authors). The specifications of the data and experiment setup are clearly explained in Section 3. Results are provided in Section 4, where we also included the supplementary analysis by comparing the two methods and their variations qualitatively and quantitatively. Finally, 5 gives a brief summary of this independent study accompanied by some thoughts for future work.

2 Methods

First, let's take a quick look at the workflow of the algorithmic part of P300 spellers as shown in Figure 3. The gray boxes (steps) are shared commonly in both methods, whereas the red boxes indicate the places to make different design choices.

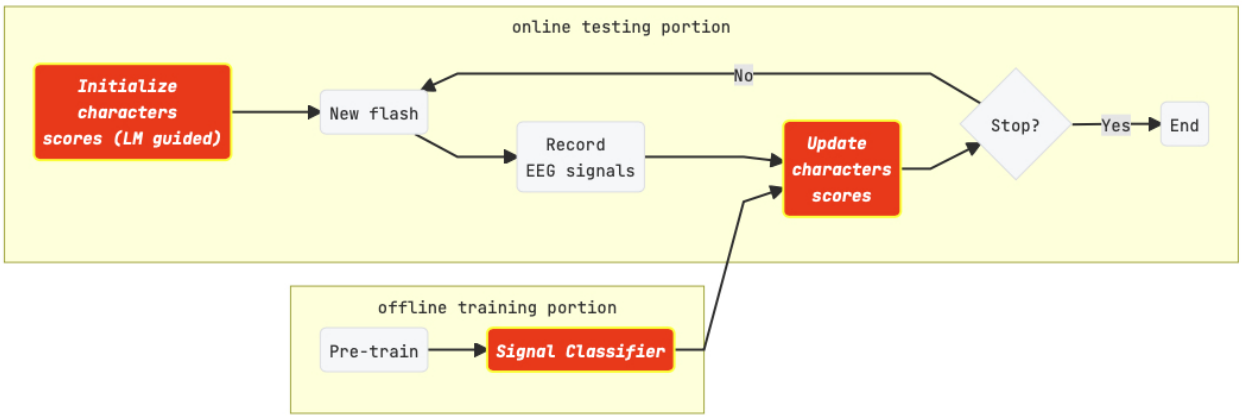


Figure 3: The general workflow of the algorithmic part of P300 spellers

We start from the offline training portion in the lower part of Figure 3. Here, we need to pre-train a classifier so that it can detect the presentation of the P300 wave given a standardized chunk of EEG

signals which usually span from the beginning of the stimulus till 800 ms later. Due to the high variance of the reactions from different subjects to the same stimuli, traditionally it is recommended to train a signal classifier for each user based on his/her training data. However, recent DL-based methods suggest a robust generalizability can be achieved. Therefore, we will stick to the traditional approach when training the SWLDA classifier, and for EEGNet, we will try to build a general classifier without being user-specific. Now, we are ready to move on to the online experiment portion, where "online" refers to the real-time application of the P300 speller. To spell each letter, the computer first needs to initialize the character scores. The initial scores are the same, but as we enter the iterations of the flash and update, the scores will change and polarize. In each iteration, we will randomly choose a row and column to flash. The subject is required to focus on the desired character and count the times it gets flashed. The observation of the EEG signal in a fixed time interval (800 ms) is sent to the classifier in the computer and the output is returned to update the scores. The general rule is that characters in any row or column that elicits a P300 wave should have a higher score increment. Theoretically, only the row and the column where the desired character resides can generate a P300 signal, so the score of that character should be the highest after some trials.

However, in practice, the EEG signals we collect are very noisy, making the classifier prone to make a wrong classification. This can severely disturb the inference of the target. Thus, we need to either find a better classification algorithm that can survive under a low signal-to-noise (SNR) ratio, OR we can try to avoid the erroneous tendency by some special processing, such as adding the **language model** when initializing the scores (as we highlighted in the top-left box of Figure 3). Even though the subject can only spell a single character each time, those previously typed ones can still provide useful information.

2.1 Methods 1: SWLDA + Bayesain Inference (enhanced by the Language Model)

The first approach uses SWLDA as the signal-classification algorithm (the red box in the bottom of Figure 3), Bayesian inference as the score-updating algorithm (the red box in the middle-left of Figure 3), and 2-gram LM to aid the score initialization (the red box in the top-left of Figure 3). We have learned LDA in class. Recall that LDA aims to find a linear combination of features that can maximize the separation of different classes while maintaining a low variance within each class. SWLDA is essentially LDA but trained in a stepwise manner. In each iteration, SWLDA will try different combinations by adding or dropping some features based on certain criteria until it finally finds the optimal one. To implement SWLDA, we refer to the existing `stepwisefit` function in `MATLAB` and LDA implementation in `Scikit-learn`, the code is enclosed in `swlda.py` submitted with this report.

Now suppose the classifier is well-trained and correctly returns the classification score for every EEG input data, now we need to think about how to update the score of each character. Bayesian inference uses the Bayes theorem to update the probability of each character being the target given the information accumulated along the trails. To do this, we need to estimate the probability density function of the classifier scores (y) of non-P300 waves (non-target) and P300 waves (target), defined as $P(y|H_0)$ and $P(y|H_1)$ respectively. Let C_m denote any character, and C^* denote the target character. From trail 0 to trail t, suppose $\boldsymbol{F_t} = [\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_t]$ is a series of stimulus event presentations, and $\mathbf{Y_t} = [y_1, y_2, ..., y_t]$ is classifier score responses. After each stimulus is presented, the classifier score, y_t, will be used to update the character probabilities:

$$P(C_m = C^*|\mathbf{Y_t}, \mathcal{F}_t) = \frac{P(C_m = C^*|\mathbf{Y_{t-1}}, \mathcal{F}_{t-1})p(y_t|C_m = C^*, \mathcal{F}_t)}{\sum_{j=1}^{M} P(C_j = C^*|\mathbf{Y_{t-1}}, \mathcal{F}_{t-1})p(y_t|C_j = C^*, \mathcal{F}_t)} \tag{1}$$

The likelihood, $p(y_t|C_m = C^*, \mathcal{F}_t)$, for each character C_m is updated depending on whether or not the character is in the flashed subset:

$$p(y_t|C_m = C^*, \mathcal{F}_t) = \begin{cases} p(y_t|H_0) & (C_m \notin \mathcal{F}_t) \\ p(y_t|H_1) & (C_m \in \mathcal{F}_t) \end{cases} \tag{2}$$

The final thing that needs to be specified is the initial probability score of each character, denoted by $P(C_m = C^*|\mathbf{Y_0}, \mathcal{F}_0)$. An intuitive way is to set them all equally to $1/N$ where N is the number of characters. This works well for the first letter to spell, but for the later letters, it's not the most effective way since we waste the information from the previously typed letters. Typically, subjects are required to type a meaningful English word (usually contains 5-7 letters). Given the prior knowledge of the first few letters, the probability distribution of letters in the current location is not uniform. Now suppose any English word can be decomposed into a series of characters $[c_1, c_2, ..., c_k]$. Here, we are going to apply a 2-gram model to correct the initial probability scores for the second and following letters. Firstly, we approximate the conditional probabilities $P(c_i|c_{i-1})$ by maximum likelihood estimation using a relative frequency count $f(c_{i-1}c_i)$ and $f(c_{i-1})$:

$$P(c_i|c_{i-1}) = \frac{f(c_{i-1}c_i)}{f(c_{i-1})} \tag{3}$$

As Mainsah et al. [2014] suggests, we choose to use the Carnegie Mellon University online dictionary as the training corpus to derive the conditional probabilities. The result can be visualized by the paired probability matrix shown in Figure 4 below:

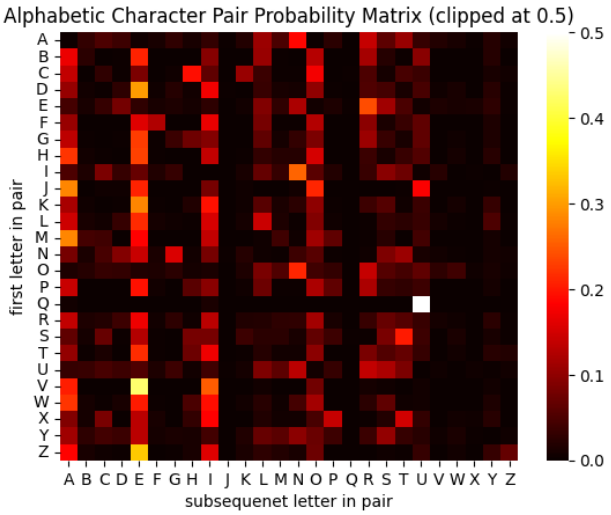


Figure 4: Probability matrix of paired English character

Before we integrate the **language model** with the rest of the algorithmic part, we need to address the concern that the previously typed letter is incorrect. To avoid the possible negative effect,

we can neutralize the LM-guided probability distribution with the uniform distribution we used before. Another detail that needs to be taken care of is that not all characters on the keyboard are English letters, so we need to rescale the probability. The final form of the equation to initialize the probability scores of the n-th character can be expressed by:

$$P(C_i) = \alpha P(C_i|A_{i-1})(1 - \frac{m}{N}) + (1 - \alpha)\frac{1}{N} \tag{4}$$

where $P(C_i|A_{i-1})$ denotes the initial probability of each character being the target given that the previous letter is letter A_i, which can be found in the probability matrix in Figure 4. Finally, α is set to 0.9 in this study as proved to work well on simulations in Mainsah et al. [2014].

2.2 Methods 2: EEGNet + Simple Counting

As the competitor of the above approach, this one utilizes a neural network called EEGNet introduced by Lawhern et al. [2018]. This network has been verified to work well on many EEG-related problems, including the P300 speller (Lee et al. [2020]). Figure 5 shows the general architecture of the EEGNet. Different from SWLDA, EEGNet consumes the data of all users as the training input. The training data is concatenated together with each column representing a timestamp (spanning from 0 to 800 ms) and each row recording the EEG signal of a certain subject at a certain trail. The input first is processed by a 2D convolutional layer, which extracts the temporal features of the signals. Then, the output feature map is passed to a 2D depthwise convolutional layer, where the spatial features in each feature map are learned. Next, the third layer is a combination of multiple depthwise convolutional layers, which serve to summarize the temporal information in each feature map separately. Finally, the output will be processed by a pointwise convolutional layer to further combine the feature information and make the final classification decision.

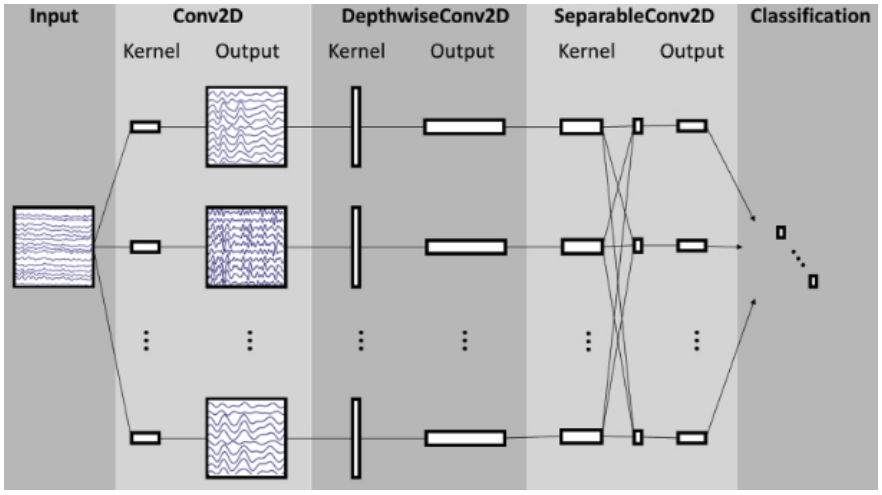


Figure 5: General architecture of EEGNet (Credit: Lawhern et al. [2018])

Depending on the input size, the size of kernels in EEGNet should be adapted correspondingly. Following the instruction in Lawhern et al. [2018] and the input data size (see Section 3.1), we implemented our EEGNet by `PyTorch`. The code is submitted with the report in the `Python` script called `eegnet_utils.py`, and the detailed information about the model structure is summarized in

Figure 6, where each "Sequential" from top to the bottom corresponds to the convolutional layer from left to right. The final Linear layer is the classification layer at the rightmost of Figure 5.

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
EEGNet                                   [4284, 2]                 --
+-Sequential: 1-1                        [4284, 64, 32, 207]       --
|    |-Conv2d: 2-1                       [4284, 64, 32, 207]       4,096
|    |-BatchNorm2d: 2-2                  [4284, 64, 32, 207]       128
+-Sequential: 1-2                        [4284, 256, 1, 51]        --
|    |-Conv2d: 2-3                       [4284, 256, 1, 207]       8,192
|    |-BatchNorm2d: 2-4                  [4284, 256, 1, 207]       512
|    |-ELU: 2-5                          [4284, 256, 1, 207]       --
|    |-AvgPool2d: 2-6                    [4284, 256, 1, 51]        --
|    |-Dropout: 2-7                      [4284, 256, 1, 51]        --
+-Sequential: 1-3                        [4284, 256, 1, 6]         --
|    |-Conv2d: 2-8                       [4284, 256, 1, 52]        4,096
|    |-Conv2d: 2-9                       [4284, 256, 1, 52]        65,536
|    |-BatchNorm2d: 2-10                 [4284, 256, 1, 52]        512
|    |-ELU: 2-11                         [4284, 256, 1, 52]        --
|    |-AvgPool2d: 2-12                   [4284, 256, 1, 6]         --
|    |-Dropout: 2-13                     [4284, 256, 1, 6]         --
+-Linear: 1-4                            [4284, 2]                 3,074
==========================================================================================
Total params: 86,146
Trainable params: 86,146
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 139.03
==========================================================================================
Input size (MB): 112.96
Forward/backward pass size (MB): 34059.31
Params size (MB): 0.34
Estimated Total Size (MB): 34172.61
==========================================================================================
```

Figure 6: The information summary of each layer in our EEGNet

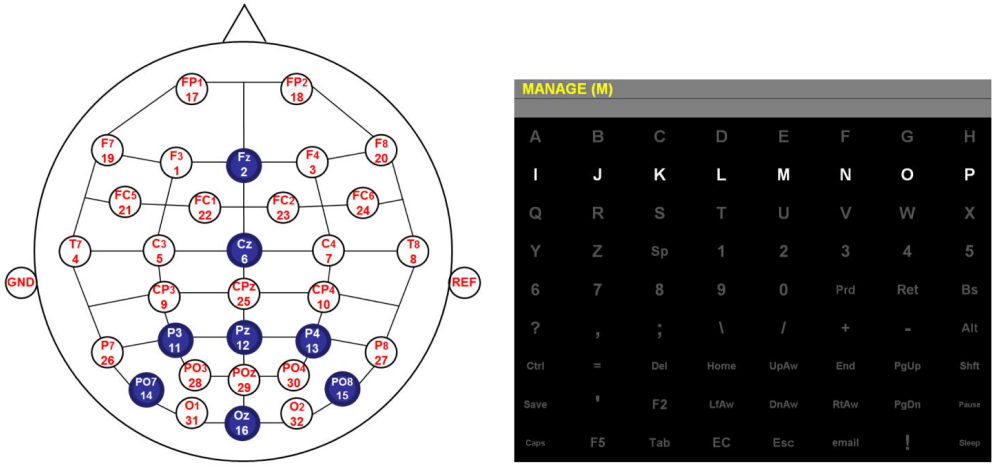
Since this EEGNet classifier is essentially a discriminative model, it can not work with the Bayesian inference algorithm we used before. Instead, we choose to directly count the number of times each character is classified as the target. In other words, if the character is in the row or column which is classified as eliciting a P300 wave, it will be incremented by 1. Otherwise, the score remains unchanged. In this way, there is no probabilistic method included here. It is still possible to adapt the **language model** to carry on the information of prior letters, but it is unknown what value we should assign at the beginning. Therefore, we decide to abandon the **language model** we used before and initialize all the scores to 0.

3 Experiment

3.1 Data

The data we used is borrowed from Mainsah et al. [2014] as we are granted permission by the original authors Prof. Mainsah and Prof. Collins at Duke University. The original studies were approved and financed by Duke University and East Tennessee State University. For the hardware side, the study used 32-channel electrode caps to collect the EEG data, and the open-sourced BCI2000 software suite to process the signals. The distribution of the electrode is visualized in

Figure 7 (a). Also, the keyboard (see Figure 7 (b)) used by the study is a standard 8×9 keyword adopted in many P300 speller research.



(a) Brainmap of the headset electrodes (b) Keyboard

Figure 7: Visual display of some hardware used in the experiment

In our study, we chose to use the data from the 13 participants since there is some missing data in the other 6 participants. Each one is required to complete one training section and one testing section. In every section, participants are required to type 5 "words" (some words contain non-alphabetic characters), with a total number of 36 characters. To spell each character, participants need to go through 7 sequences, i.e., $7 \times (8 + 9) = 119$ trails. Thus, each one will generate $119 \times 36 = 4289$ trials for a training section or a testing section. Now let's look closely at each trial. There are 32 electrodes working to collect EEG signals at different locations simultaneously. The frequency is 256 Hz, so there are $256 \times 0.8 \approx 207$ observations in each time window. This explains why the number of one participant's input data is 4284 in Figure 6.

For EEGNet, the only data preprocessing it needs is to concatenate every participant's data. As a comparison, the preprocessing procedure for SWLDA is much more complicated. Firstly, we need to keep in mind that the SWLDA classifier needs to be user-specific, so we will train the classifier for every participant. Secondly, we want the model to be light-weighted as possible, so only the signals from the 8 core electrodes highlighted by blue in Figure 7 (a) will be used (Krusienski et al. [2006]). Moreover, the 207 observations are also too much for the model. We divide the time window into 15 equal parts with each one including 13 timestamps, and then we average each part. We repeat this process for all the 8 electrodes, and vertically concatenate these averaged data together, ending up with $15 \times 8 = 120$ features for one trial.

3.2 Setup

The training of the SWLDA classifier is conducted on a 2021 Macbook Pro with an Intel i5 CPU. There are 13 participants, so we trained 13 classifiers individually with their own training data. The easy data preprocessing is offset by the complicated setup of the training configuration. We use the cross-entropy loss as the loss function (criterion), and Adam as the optimization algorithm (optimizer) with zero-initialized parameters and a fixed learning rate at 0.001. We choose to train 100 epochs, where the data is divided into 32 batches for each iteration. The training of the EEGNet

is no longer available on a normal CPU, so we turned to leveraging the T100 GPU powered by Google Colab. One thing we need to be very careful about is the unbalance of the data. The nature of the P300 waves requires the target signals to appear very rarely, leading to a relatively small portion of the target samples. As a result, we must assign a bigger weight to balance the effect from both classes when creating the loss function.

Once the classifiers are well trained, we deploy them to test their performance on the given test data. We first check the signal-classification accuracy. Then, we integrate the classifiers with the rest of the algorithmic part to test the overall spelling accuracy. The results and analysis are presented in the next section.

4 Results

First, let's take a look at the training procedure of EEGNet in Figure 8. Surprisingly, after the first epoch, the accuracy has already achieved over 80%, but the problem is that neither the train nor test accuracy appears to stabilize throughout the training procedure as they are fluctuating between 70% and 85%. A similar trend happens again in the losses ("mean loss" refers to the average loss for each observation/trial) where training loss and test loss also fluctuate. However, training loss becomes even lower while the testing loss basically remains at the same level. This warns that the model may start to overfit the training data.

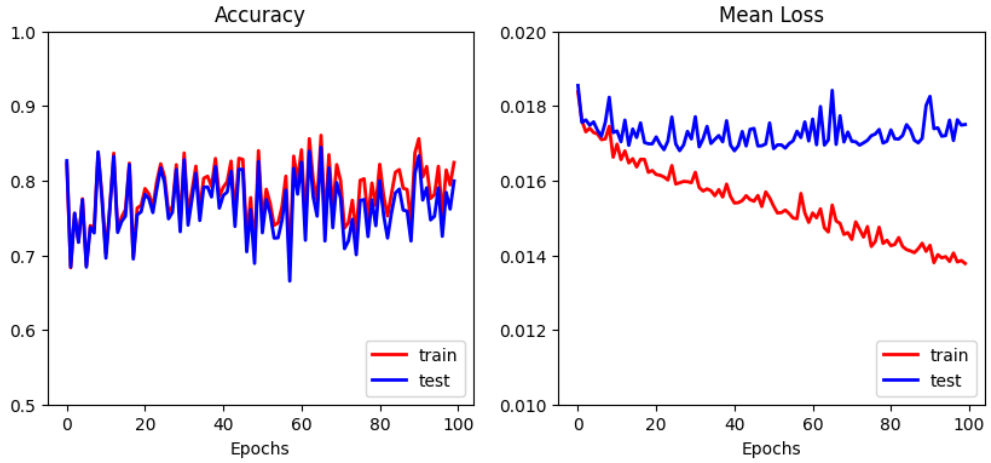
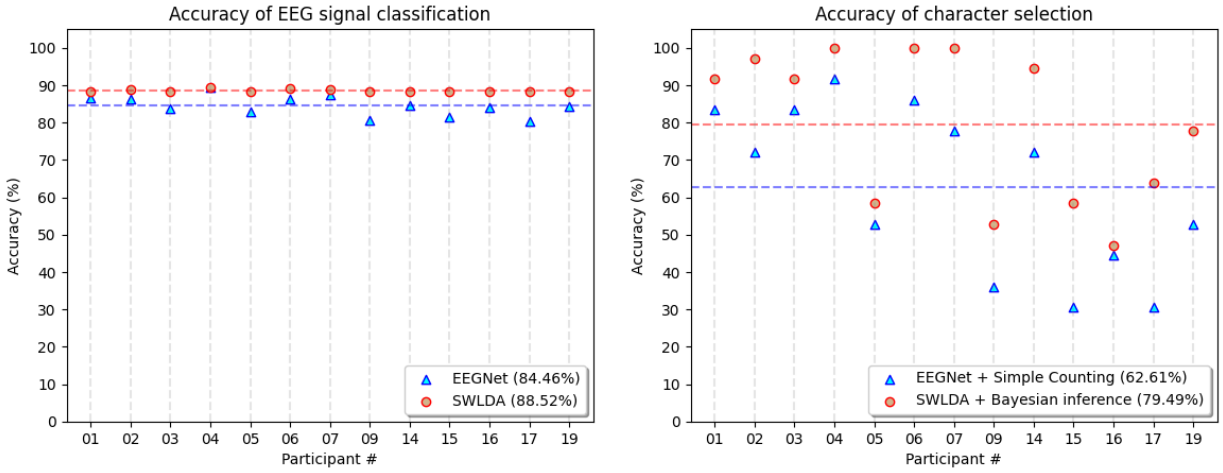


Figure 8: Loss and accuracy at each epoch during the training of EEGNet

As we mentioned before, it is quite necessary to balance the effect of the two classes since the target class is the absolute minority in the original dataset. Before we fix this problem, the model is trained to assign every input to the non-target class. This is intuitive since there are around 82% data in the training set belonging to non-targets. Always returning the decision as the non-target class can automatically get an accuracy of 82%. However, the classification accuracy does not reflect the whole story. Failing to detect any target signals makes the EEGNet useless for the overall algorithmic part. After we fixed this problem, we derived a working EEGNet with the training procedure shown in Figure 8. We saved the model with the best test (validation) accuracy as the checkpoint. Although the checkpoint model is sufficient to use, we must rethink why the accuracy has such a significant fluctuation. Again, maybe only measuring the accuracy is not enough, we

need to investigate other measurements, such as recall and precision.

Now it is time to compare how well the two methods are performed on the test data. Figure 9 compares the signal classification accuracy. The red dotted line and blue dotted line mark the averaged accuracy for SWLDA and EEGNet correspondingly. SWLDA classifier wins by a narrow margin of 4%. In each method, the variation among all 13 participants is very small, so comparing the average level is enough. For the comparison of the spelling accuracy, the gap becomes more obvious, the gap not only exists between the methods but also exists among the participants. This time, the average spelling accuracy of the SWLDA-based algorithm (method 1) is around 17% ahead of the EEG-based algorithm (method 2) although both are not satisfactory as both are lower than 80%. However, if we look at each participant's performance, we can notice that some of them have a way higher spelling accuracy than others. This observation is very important for us to interpret the results as this verifies the fact that reactions to the stimulus are very different among participants. As a result, there is no wonder that the SWLDA-based algorithm (user-specific) can perform better than the EEGNet-based algorithm (general).



(a) Signal classification accuracy

(b) Spelling accuracy

Figure 9: Performances of the two models on test data

Another interesting question is: why the small gap (4%) in signal classification accuracy is enlarged to 17% in spelling accuracy? We believe this can be attributed to the use of the 2-gram **language model** and the essence of the generative probabilistic model. Firstly, when we probe into the wrong prediction made by EEGNet, we notice that a lot of them are nearby characters around the desired one. This is a very common mistake due to the noise and the inevitable pattern that participants are prone to be distracted by the flash of the nearby characters. The 2-gram language model can eliminate the problem to some extent since it utilizes the information of the previously typed letter, which can give some initial advantage to a letter with a higher probability of being the next desired character.

Even though the SWLDA-based algorithm seems to be the better one (after all, it remains as the state-of-the-art method for a long time till now), we should not ignore the upsides and bright potentials of the EEGNet. The biggest advantage is that it does show the generalizability across

different participants. It is very likely a higher accuracy can be achieved by feeding with more data and equipping with better training setups. In fact, we should be aware that the pre-training process is very tedious for the users. If we can bypass it with a given general classifier, it will make the P300 speller more user-friendly and easy to use. Even if the original EEGNet is not good work, we can leverage transfer learning to adapt the original one to be more user-specific, which is still better than training a SWLDA classifier for each user from scratch.

5 Conclusion

In this study, we successfully implemented two representative but completely different algorithms in P300 spellers. Method 1 (SWLDA classifier and Bayesian inference) represents the generative probabilistic model, whereas method 2 (EEGNet classifier and simple counting) represents the discriminative neural network. By comparing their performance on real data, the SWLDA-based method defends its title as the state-of-the-art solution, but we should not forget that the **language model** we added to it also enhances its performance. On the other hand, the result that the EEG-based method fell behind doesn't hide its value and potential. We have to admit that we didn't maximize the power of this deep neural network as the training setup is not optimized, but this also reflects how promising if it can be improved with more data and better training configuration since it already shows a convincing sign of generalizability.

We learned a lot from this final project. The implementation of the two methods undoubtedly enhances our coding skills, but more importantly, consolidates my comprehension of the two methods. If reading papers can gain half of the knowledge, then implementing the algorithms and conducting the experiments are the necessary conditions to cover the remaining half. Moreover, we benefit a lot from the cycle of "test run-analysis-debug" since it leads us to a deeper understanding of the phenomenon we observed whereas sometimes it ends up with some fixed bugs.

The only pity here is that we didn't find an optimal joint between the P300 speller and natural language processing in this project. I started my research in P300 speller and ECE 684 at the same time, and previously I thought there should be some amazing topics between the two where I could experiment with some advanced NLP techniques such as RNN, attention, and transformer on the application of P300 spellers, but later I was demoralized by the depressing fact that current P300 spellers only works by spelling one character at a time. This makes it impossible to apply any word-based language models. In the end, the only available language model available to use is the n-gram, which we learned at the beginning of this course. Although it is rather simple, it turns out that it works perfectly well on this problem. In addition, designing and training EEGNet serves well as an alternative practice for me to build a deep neural network. It is not a pure language model, but it shares the common designing and training process with those DL-based language models.

Speaking of future work, we are going to further improve the performance of the EEGNet-based algorithm. It is also interesting to try to enhance it with some language models as we did in the SWLDA-based algorithm. The key goal here and for the future is to keep updating the EEGNet until it is good enough to challenge the title of "state-of-the-art". Another area that could be the next step to explore is to fundamentally update the P300 speller from spelling individual letters to typing continuous sentences. Only in this way can we open great possibilities to apply some powerful language models and further improve the capability of the speller.

References

Saba Moghimi, Azadeh Kushki, Anne Marie Guerguerian, and Tom Chau. A review of eeg-based brain-computer interfaces as access pathways for individuals with severe disabilities. *Assistive Technology*, 25 (2):99–110, 2013.

Lawrence Ashley Farwell and Emanuel Donchin. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. *Electroencephalography and clinical Neurophysiology*, 70(6): 510–523, 1988.

Peter Brunner, Anthony L Ritaccio, Joseph F Emrich, Horst Bischof, and Gerwin Schalk. Rapid communication with a "p300" matrix speller using electrocorticographic signals (ecog). *Frontiers in neuroscience*, 5:5, 2011.

JA Elshout. Review of brain-computer interfaces based on the p300 evoked potential. Master's thesis, 2009.

Tao Fang, Zuoting Song, Lan Niu, Song Le, Yuan Zhang, Xueze Zhang, Gege Zhan, Shouyan Wang, Hui Li, Yifang Lin, et al. Recent advances of p300 speller paradigms and algorithms. In *2021 9th International Winter Conference on Brain-Computer Interface (BCI)*, pages 1–6. IEEE, 2021.

Boyla O Mainsah, Kenneth A Colwell, Leslie M Collins, and Chandra S Throckmorton. Utilizing a language model to improve online dynamic data collection in p300 spellers. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(4):837–846, 2014.

Vernon J Lawhern, Amelia J Solon, Nicholas R Waytowich, Stephen M Gordon, Chou P Hung, and Brent J Lance. Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of neural engineering*, 15(5):056013, 2018.

Jongmin Lee, Kyungho Won, Moonyoung Kwon, Sung Chan Jun, and Minkyu Ahn. Cnn with large data achieves true zero-training in online p300 brain-computer interface. *IEEE Access*, 8:74385–74400, 2020.

Dean J Krusienski, Eric W Sellers, François Cabestaing, Sabri Bayoudh, Dennis J McFarland, Theresa M Vaughan, and Jonathan R Wolpaw. A comparison of classification techniques for the p300 speller. *Journal of neural engineering*, 3(4):299, 2006.