OAP: An efficient online principal component analysis algorithm for streaming EEG data

Abdulghani Ismail Zubeir
Abstract

**OAP: An efficient online principal component analysis algorithm for streaming EEG data**

*Abdulghani Ismail Zubeir*

Data processing on streaming data poses computational as well as statistical challenges. Streaming data requires that data processing algorithms are able to process a new data point within micro-seconds. This is especially challenging on dimension reduction, where traditional methods as Principal Component Analysis (PCA) require eigenvectors decomposition of a matrix based on the complete dataset. So a proper online version of PCA should avoid this computational involved step in favor for a more efficient update rule. This is implemented by an algorithm named Online Angle Preservation (OAP), which is able to handle large dimensions in the required time limitations.

This project presents an application of OAP in the case of Electroencephalography (EEG). For this, an interface was coded from an openBCI EEG device, through a Java API to a streaming environment called Stream Analyzer (sa.engine). The performance of this solution was compared to a standard Windowised PCA solution, indicating its competitive performance. This report details this setup and details the results.
Acknowledgment

I would like to thank my thesis supervisor Kristiaan Pelckmans, co-supervisor Tore Risch and reviewer Lars Oestreicher for their great help, support and passionate participation in providing thesis tools and ideas. The space was always open whenever I ran into a trouble spot or had a question about my research, implementation or writing.

Special thanks to Stream Analyze company in Sweden for their assistance in the implementation of the project work.
Abbreviations

• PCA - Principal Component Analysis
• OPCA - Online Principal Component Analysis
• EEG - Electroencephalography
• IPCA - Incremental Principal Component Analysis
• SA - Stream Analyze
• VA - Visual Analyzer
• DPCA - Dynamic Principal Component Analysis
• RPCA - Recursive Principal Component Analysis
• MWPCA - Moving Window Principal Component Analysis
• EVD - EigenVector Decomposition
• SVD - Singular Value Decomposition
• O-SVD - Online Singular Value Decomposition
• OAP - Online Angle Preservation
7 Discussion ........................................................................................................... 37
  7.1 Performance ................................................................................................. 37
  7.2 Convergence ............................................................................................... 38
8 Conclusion ......................................................................................................... 39
9 Future work ..................................................................................................... 40
References .......................................................................................................... 42
1. Introduction

1.1 Problem Description

Principal Component Analysis (PCA) is a well-known and widely used data analysis in scientific computing. It is commonly used to measure the direction where there is the most variance, i.e the direction where the data is most spread out. This can simply be referred as a process of dimensional reduction of the data. Traditionally, PCA has been done successfully in a batch data environment. However, in regard to the cutting edge of technology, data explosion with high velocity and dimension, a data streaming technology is becoming prevalent. This consequence, demands some online techniques for data processing including PCA. The online PCA (OPCA) technique, which do not require storing of all data in memory (memory limited) is a problem that is to be addressed in this report. Despite the availabilities of different theoretical algorithms that can efficiently update the PCA when new data are observed, the results have as yet not trickled down to a practical implementation.

The online PCA problem can be formulated in many different ways, but one is fundamental for context of online algorithm. Given n high-dimensional vectors $x_1, ..., x_n \in \mathbb{R}^d$ and a target dimension $D < d$, produce $n$ other low-dimensional vectors $y_1, \ldots, y_n \in \mathbb{R}^D$ such that the reconstruction error is minimized. [1]

The source of streaming data for this problem is an electroencephalogram (EEG). This is electro-physiological monitoring technique that detects electrical activities in the brain of a subject using small flat electrodes attached to the scalp. The brain cells communicate via electrical impulses which are active around the clock. This produces a continuous flow of data that can be live streamed and continuously processed such as online PCA. An EEG produce voltage fluctuation of brain neurons and maps the voltage to raw data as continuous signal. Each channel of an EEG represents a discrete column of the continuous vectors, where 16 channels, produce vectors of 16 elements (dimensions) continuously.

1.2 Project Purpose and Goal

The main objective of this thesis project is to investigate, implement and test a suitable version of Online Principal Component Analysis (OPCA) in the
context of streaming multi-channel EEG recordings. Specifically, aiming to perform a compact computation on actual live EEG readings that varies from different brain locations with different basic cerebral activities (such as body movements) mapping to a low visual dimension. This is studied under the name of Information Flow connectivity.

A complete workflow therefore will introduce a complete setup for experimental environment and be able to examine the results. In order to have a deep insight of the OPCA subject and experimental results, a research part involve the designing the algorithm and evaluating the result from project setting.

1.3 Why EEG

The experimental choice of EEG as a source of streaming data is precisely meaningful not as luck would have it. The streaming data could have made possible from various sources such as laptop microphone. However, the setting of principal component analysis on EEG source is becoming interesting in studying anatomical connectivity of statistical dependencies between brain regions that have different functional properties. Every part of the brain is defined with a set of particular actions on a resting state and task-state. This scenario if well examined, produces a cross-correlations among different brain regions.

The study of PCA on EEG signal, can be very well involved with functional connectivity to evaluate the results. For instance, a resting person closing eyes and thinking very hard could result in receiving high EEG values from the frontal part of the brain. This can lead to high variance in the direction of frontal lobe of the brain. The most crucial part of PCA on EEG signal, can deducted as the ability of abnormal detections from brain impulses by a field of pattern recognition. Brain disease such as epilepsy, coma, sleep disorders can be studied and researched further with detection through abnormal variance of the PCA. In contrast to a traditional way of observing the abnormalities from raw signals, which underlies inaccuracy.

1.4 Delimitation

In this thesis project the focus is to study, design, implement and evaluate the results out from the prototype. Promising to provide at least good solution representation. An optimal result and design is not necessary prerequisite for this thesis.
The thesis took into account current OPCA challenges but does limit to solve these challenges. The design implementation is not on developing a commercial solution. The goal is as yet not to investigate the neurological aspects of the resulting visualization.
2. Background

2.1 An EEG and the OpenBCI

The EEG headset *Ultracortex "Mark IV" EEG Headset* from OpenBCI, is an open source 3D headset for live viewing and recording brain activity (EEG), muscle activity (EMG), and heart activity (ECG). The headset is capable of recording up to 16 channels that represent 16 different coverage of brain regions. The OpenBCI headset provides an open source software for live viewing of EEG signals from the brain called OpenBCI_GUI.

The headset configuration shown in Figure (2.1) is a streaming data from EEG signals where each signal represent electrode placed on a particular region of the brain.

*Figure 2.1.* The left figures is OpenBCI helmet, middle figure is EEG signals, and right figure EEG electrodes on brain

Different industries have been using much of these tools in entertainment, medical application and for researching purpose. A study of Effective Connectivity, for measuring interactions among multiple brain regions using Independent Component Analysis (ICA) is one of revolutionary research, that draws a meaningful result to analyze brain impulses. The study helps further in researching of brain medical disease. Not mentioning in entertainment, these results typically helped in interactive mind-controlled video game platform which allows users to control, influence, and interact with video games.

2.2 Batch setting PCA

Traditionally, PCA is solved in a batch setting, called batch PCA or offline PCA, where one assumes that relevant data is recorded before computation
starts. Mathematically, this type of PCA is implemented by calculating the covariance matrix of the batch data, followed by eigenvalue decomposition (EVD) of the covariance matrix or the singular value decomposition (SVD) of the centered data \[2\].

The problem can be formulated from given high dimensional vectors \(x_1, x_2, x_3 \ldots x_n \in \mathbb{R}^d\), where \(d\) is sufficiently large. The batch setting intends to produce a promising representation of the data (via projection of eigenvectors) in a low dimensional space \(D\), without losing potential features of the data. If the setting requires a much smaller \(D\) than provided input data \(n\), and dimension \(d\) (which is a common approach in dimensional reduction), the solution can be simply achieved with adequate accuracy. However the case, as \(D\) approaches zero, this can lead to loss of important features from the data \[2\].

The performance of the subspace representation \(D\) (from \(d\)), is measured by mean squared error (MSE) of the distance between the centered (mean) vectors and their projections in the subspace representation. This is an optimization problem, with a goal of finding a projection matrix \(P_D\) that minimize a loss function, \(R_n(P_D)\) as given in equation 2.1

\[
R_n(P_D) = \frac{1}{n} \sum_{i=1}^{n} ||(x_i - \mu_n) - P_D(x_i - \mu_n)||^2 \quad (2.1)
\]

From the given d-dimension data, it’s covariance matrix is formulated in equation 2.2

\[
Cov_n = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_n)(x_i - \mu_n)^T \quad (2.2)
\]

Where, \(v_{1:n}, \ldots, v_{d:n}\), are orthonormal vector of \(Cov_n\) and \(\lambda_{1:n} \geq \ldots \geq \lambda_{d:n} \geq 0\) it’s corresponding eigenvalues. The minimum p x p projection matrix \(P_D = V_D V_D^T\), maps each data point \(x_i\) onto its rank-D. The orthogonal projection of \(x_i\) onto the subspace spanned by the orthogonal columns of vector \(V_D\) and the reconstruction is be made by \(P_Dx_i\) \[3\]. These eigenvector in matrix \(V_D\) are called Principal Components, and it can be shown that minimum reconstruction error, \(R_n(P_D) = \sum_{j=D+1}^{D} \lambda_{j:n}\).

From the above theoretical presentation of the problem, it can be clearly noted that, the batch PCA setting has complexity of \(O(d^2)\) from Covariance matrix \((d \times d)\) computation and complexity of \(O(nd)\) from memory usage (of \(n \times d\)-data), hence full EVD computation has complexity of \(O(nd^3)\). Because of these performance challenges, that are mainly defined by time and space, batch PCA setting is impractical when data is;

- Large with very high dimension (i.e \(n\) and \(d\) are extremely large)
• At high velocity, (i.e data that change rapidly with time)

Batch PCA requires stationary data by processing all at once, which is not always the case. Under these limitations, there is a need of much faster algorithm to compute both large and moving data.

2.3 Online PCA

As clearly pointed out in the previous section, the traditional setting for computing eigenvectors and eigenvalues via EVD uses $O(nd^3)$ time-space complexity in $\mathbb{R}^d$. This has been proved impractical for large dimension and high velocity data sets. On this account, it is crucial to take into consideration a PCA setting that keeps incrementing and updating their decomposition matrix from time to time, this is known as online PCA also Incremental PCA (IPCA). Therefore, an incremental setting has to consider each data point and updating their estimates of the eigenvectors. Computing such eigenvectors timely with corresponding new data point, has $O(d)$ complexity space of varying data[4].

A major drawback of the batch PCA is that the model, once built from a particular data, is time-invariant. The real time-varying characteristics of data, i.e changes in the mean, variance, dimensions and correlation structure make the batch PCA infeasible[5]. A significant amount of research has been done on this problem to come up with different promising algorithms that prevail efficiency on large data set and real time series data. Intuitively, over so often, algorithms have emerged, starting from static PCA, then jumped to Dynamic Principal Component Analysis (DPCA), which was first introduce in [6]. DPCA is a version of PCA predominantly on a fixed moving or sliding window of the data. DPCA typically works in the same way as a batch PCA, but rather by adding lagged rows of original variables, aiming to update autocorrelation to cross-correlation[7]. Despite the case, DPCA addresses the autoregressive and moving average structures in the data, it still faces the same challenges as static PCA in time and space complexity [8].

![Figure 2.2. The left two figures showing the RPCA and the right two figure showing MWPCA [9]](image)
Furthermore, other intuitive algorithms that undertook DPCA, by addressing the rapid change of data are Recursive Principal Component Analysis (RPCA) and Moving Window Principal Component Analysis (MWPCA). Both of the two method try to reduce the autocorrelation influence of the previous observations on computing mean and covariance. MWPCA will keep updating every time, however strict estimations of older observations by a specified window period. Assume a window size $H$, a data matrix at time $t$ will be given as $X_t = x_{t-H+1}, x_{t-H+2},...x_t$. On the other hand, RPCA will keep updating every time, but includes estimations of entire older observations. However, down weighting the of older observations in calculating the mean and covariance by a forgetting factor $\eta < 1$ [9].

Determinately, many more incremental PCA model and algorithms have been developed from earlier definition of online PCA. For examples, A random projections online PCA a work of Sarlos [10]. This type of PCA computes an output $y_t = S^T x_t$, where $S \in \mathbb{R}^d$ is a projection matrix generated randomly and independently from the data. A Stochastic method by [11] and Regret minimization researched by[12]. However the case, fundamental challenges still exist such as the optimal convergence rate and estimating the top D eigenvectors, for $D > 1$ as described by [4]
3. Related Work

The PCA problem has a long history from its extensively usage in dimensionality reduction. The traditional batch setting normally uses a method of eigenvalue decomposition which is quite prohibitive in real time series data as discussed in previous chapter. In the field of online learning and preprocessing real time data, it is of great interest to equip an online version of PCA. An extensive line of research and algorithms discussing online PCA from the statistical perspective have been published. However, not so many implementations have reflected the efficiency of these theoretical works.

Many related publications exist in the space of solving top eigenvector problem. For a better understanding of this thesis solution approach, it is therefore crucial to review some similar works of online PCA. In this chapter, three different algorithms that uses statistical estimators will be analyzed. First, one of the popular related works is a method that uses a power iteration to update the projection matrix and stretch the initialized vector iteratively. Second, a foundational work in the field of top eigenvector problem was done by two popular old classical methods of Krasulina (1969) and Oja (1983). The basic idea of their work is to maintain an estimation of the top eigenvector and incrementally update repeatedly with new observed data set [4]. Their results have led to the evolving of many works that uses a stochastic and incremental algorithm such as Arora et al., 2012 [11], and De Sa et al., 2015 [13]. Last, a scheme that adopt the stochastic and power iteration methods at the same time is Accelerated Stochastic Power Iteration will be analyzed in the next section of this chapter.

3.1 Power Iterations

The power method is an iterative algorithm for computing the top eigenvalue by absolute value[14]. Given a dataset, initialize a random vector $P_0$ and observe new data by stacking them up in $X_t$. That is, at each iteration, and the algorithm computes the matrix $M = X_t X_t^T$. The bottomline of the algorithm is that, the top principal components reflect the direction in which the random initialized vector $P_0$ stretches by the effect of multiplication with the matrix M over and over. The concise formulation of this approach is shown in algorithm 1 below [15].
Algorithm 1 Online PCA using Power Iteration method

Require: Let $D = 2$. Given matrix $M = X_t X_t^T$, Select a random $P_0 \in \mathbb{R}^{dxD}$

for $t = 1, \ldots$ do

1. Observe $D$ new data points $x_t^{(1)}, \ldots, x_t^{(D)} \in \mathbb{R}^d$ and stack them up in $X_t \in \mathbb{R}^{dxD}$.

2. Update as,

\[ P_t = M' P_0, \]

\[ \text{if} \ P_t / \|P_t\| \approx P_{t-1} / \|P_{t-1}\| \text{ the return } P_t / \|P_t\|. \]

end for

\[ \xrightarrow{\text{End of loop}} \]

The runtime iteration for the power iteration method is $O(nd)$, which is not as good enough as stochastic methods described below. Despite the case, compared to stochastic/incremental method, the power iteration method doesn’t use or/and depends on the learning rate and the multiplier constant. Therefore these two multipliers have no effect on the convergence of the algorithm. On the other side, the drawback of the power iteration method is that, $M$ must have a dominant real eigenvalue $\lambda_1$, such that, $\lambda_1 / \lambda_2$ is large. Otherwise, If $\lambda_1 \approx \lambda_2$, then the algorithm might not converge. This implies that the data should be extremely linear [14].

3.2 Stochastic/Incremental

Among the study of online PCA using top eigenvector is by Balsubramani et al., 2013 adopting Krasulina (1969) and Oja (1982) methods [4]. The two methods are similar and close enough but the update function differs. The idea is, at time $t = 1$, observe data points and stack them into $X_t$. The projection matrix $P_0$ is initialized by random data points, then the update of projection matrix $P_t$ is performed as shown in algorithm 2 using Krauslina and algorithm 3 using Oja [4].
Krauslina (1969) scheme:

**Algorithm 2** Online PCA using Krauslina method algorithm

**Require:** Let $D = 2$ and fix a suitable $\varepsilon > 0$. Set $P_0 \in \mathbb{R}^{dxD}$. Let $\sum_t \pi_t = \infty$ be a sequence such that $\sum_t \pi_t^2 < \infty$.

**for** $t = 1, \ldots$ **do**

1. Observe $D$ new data points $x_t^{(1)}, \ldots, x_t^{(D)} \in \mathbb{R}^d$ and stack them up in $X_t \in \mathbb{R}^{d \times D}$.

2. Update as,

$$P_t = P_{t-1} + \pi_t (X_t X_t^T - \frac{P_{t-1}^T X_t X_t^T P_{t-1}}{|P_{t-1}|^2}) P_{t-1},$$

(1)

where $\pi_t$ is a learning rate chosen such that $\pi_t = c/t$ for all $c = 1, \ldots, D$.

**end for**

=end of loop

Oja (1982) scheme:

**Algorithm 3** Online PCA using Oja method algorithm

**Require:** Let $D = 2$ and fix a suitable $\varepsilon > 0$. Set $P_0 \in \mathbb{R}^{dxD}$. Let $\sum_t \pi_t = \infty$ be a sequence such that $\sum_t \pi_t^2 < \infty$.

**for** $t = 1, \ldots$ **do**

1. Observe $D$ new data points $x_t^{(1)}, \ldots, x_t^{(D)} \in \mathbb{R}^d$ and stack them up in $X_t \in \mathbb{R}^{d \times D}$.

2. Update as,

$$P_t = P_{t-1} + \pi_t (X_t X_t^T - \frac{P_{t-1}^T X_t X_t^T P_{t-1}}{|P_{t-1}|^2}) P_{t-1},$$

(1)

where $\pi_t$ is a learning rate chosen such that $\pi_t = c/t$ for all $c = 1, \ldots, D$.

**end for**

=end of loop

The fact that these methods are close enough, most of their practical setups fall together. In evaluating their performance, Balsubramani et al., 2013 investigate the two schemes with the high dimension and a large/growing dataset. Both methods were practically surpass the eigenvalue decomposition problem in terms of time and space complexity. The runtime iterations for the two scheme are $O(d)$ and $O(nd)$ for a memory usage. Despite the efficiency benefits, these methods still have left an open problem in accuracy of the estimation. Especially the effect of the learning rate $\pi_t$ and the multiplier $c$ on the rate of convergence [4].
3.3 Accelerated Stochastic Power Iteration

On previous section, a brief description of the stochastic/incremental and power iteration have slightly indicated the limitations of the two methods in the rate of convergence. The work by De Sa et al., 2015 [16], has claimed the prohibition of these two methods, and can only be relied on small and medium problems. For this and other reasons, De Sa et al., 2015 came up with algorithms that combine the two methods. Their work uses the power iteration methods but rather operate in stochastic setting. This approach aim to have a sample complexity with an asymptotically optimal dependence on the eigen-gap and an iteration complexity with an asymptotically optimal dependence on the eigen-gap [16]. The goal of this scheme is to increase momentum in power iteration method, applying it in stochastic setting and achieve the accelerated convergence rate [16].
4. Methodology

The overall purpose of this thesis project is to research, implement and test a suitable version of an online PCA algorithm in the context of streaming data from EEG recordings. These three mentioned purposes provide the fundamental blocks of the research methodology for this thesis. The main objective of this chapter is to describe an overview of the scientific methods used in this thesis. This means only the research strategies and general research methods are described. The practical implementation, experiment and results will be discussed in the next chapters separately.

In this chapter, main steps of the thesis are reflected and described in three sections. In each of the three sections, a detailed description is given on information gathered and thesis procedures. The first section represents the study of literature review with the aim of understanding different theoretical promising algorithm of On-line PCA. The second section represents the experimental phase and the reasons for selecting a chosen PCA algorithm, EEG data and tools. The last section is devoted to the experimental results, evaluation and limitations of the research methods.

4.1 Literature study

One among the basic component of this master thesis is researching a state of art of an online PCA. The traditional PCA algorithm has no doubt produces excellent results and has left almost tolerable challenges. Many theoretical works and algorithms have been published on online PCA with a target based on the promising solutions, solution limitations and open problems in their works. However, most of these algorithms performances have not reflected on practical implementations.

In their study, Cardot et al. (2015) [2], reviewed the prevalent online PCA algorithms with interest of finding the most efficient performing and updating the PCA on time-varying data. Their work investigates different online PCA algorithms such as candid covariance-free incremental PCA, stochastic approximation, perturbation approximation, incremental PCA and many others based on computational cost and memory. In concluding their remarks, most of the algorithms reviewed in their study, depends on eigenvalue decomposition method. All these methods seemed to struggle to perform a trade-off
between attaining accuracy and speed. The stumbling block of recomputing the eigenvalue decomposition iterative hinders those methods from efficiency in practical implementations.

Therefore, these findings have driven this thesis project in choosing an online PCA method that is independent on eigenvalue decomposition. For this and other reasons, the thesis project, has derived its exclusive online PCA algorithm for Online Angle Preservation (OAP). This algorithm plays a significant role in this research and a choice for analyzing the results.

4.2 Experimenting: OPCA using Online Angle Preservation (OAP)

The literature review done in this thesis has given the fundamental approach in solving the research question. The study has explored some popular algorithms from the different articles and papers. As stated on a previous section, an online PCA algorithm for Online Angle Preservation (OAP) was chosen. The reasoning is basically from its method of avoiding eigenvalue decomposition re-computation, making it possible to process high-dimensional signals in a real-time fashion. A comprehensive detail on OAP implementation in conjunction with other stated methods has been documented in Chapter 4 and in particular section 4.6.

The OAP algorithm aims to find a projection matrix $P: \mathbb{R}^d \rightarrow \mathbb{R}^D$ such that $x_i^T x_i$ and $P(x_i)^T P(x_i')$, are not too different, where $x_i^T$ are observed data and $x_i'$ are projected data. The projection matrix $P$ is constantly updated according to the accuracy factor $\varepsilon$. This illustration simply means preserving the pairwise angles of the projected data to those of original in the high dimension. The concise illustration of the OAP algorithm is shown in algorithm 4.
Algorithm 4 Online PCA for Online Angle Preservation

**Require:** Let $D = 2$ and fix a suitable $\varepsilon > 0$. Set $P_0 \in \mathbb{R}^{d \times D}$. Let $\{\gamma_t > 0\}_t$ be a sequence such that $\sum \gamma_t^2 < \infty$.

for $t = 1, \ldots$ do

(1) Observe $D$ new data points $x_t^{(1)}, \ldots, x_t^{(D)} \in \mathbb{R}^d$ and stack them up in $X_t \in \mathbb{R}^{d \times D}$.

(2) Calculate loss:
$$l(X_t; P_{t-1}) = \| X_t^T X_t - X_t^T P_{t-1}^T P_{t-1} X_t \|_\infty,$$
\[\triangleright (1)\]
and
$$\sum_t = \text{sign} ( X_t^T X_t - X_t^T P_{t-1}^T P_{t-1} X_t ) \in \{-1,0,1\}^{D \times D} \triangleright (2)$$

(3) If $l(X_t; P_{t-1}) > \varepsilon$, update as
$$P_t = \pi_t ( P_{t-1} - (X_t X_t^T) P_{t-1} \sum_t ) ,$$
\[\triangleright (3)\]
where the normalization $\pi_t$ is chosen such that $\| P_t e_i \|_2 = 1$ for all $i = 1, \ldots, D$.

end for \[\triangleright \text{End of loop}\]

**Table 4.1. Table definition of symbols used in algorithms 4**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>Number of data point observed at each iteration</td>
</tr>
<tr>
<td>$d$</td>
<td>Dimension of the data</td>
</tr>
<tr>
<td>$x_t^{(1)}, \ldots, x_t^{(D)}$</td>
<td>$D$ observed data at time $t$.</td>
</tr>
<tr>
<td>$X_t$</td>
<td>A matrix at time $t$ with two observed data set</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>The accuracy factor, between 0 and 1</td>
</tr>
<tr>
<td>$t$</td>
<td>Iteration at time $t$, such that $t = 1, \ldots, \infty$</td>
</tr>
<tr>
<td>$P_{t-1}$</td>
<td>A projection matrix at time, $t-1$</td>
</tr>
<tr>
<td>$P_t$</td>
<td>A projection matrix at time, $t$.</td>
</tr>
<tr>
<td>$l(X_t; P_{t-1})$</td>
<td>A loss function at time, $t$.</td>
</tr>
<tr>
<td>$\pi_t$</td>
<td>A normalization factor chosen at time, $t$ (learning rate).</td>
</tr>
<tr>
<td>$\sum_t$</td>
<td>A sign of a loss function at time, $t$.</td>
</tr>
</tbody>
</table>

In the practical set and implementation of this thesis project, $x$ is data set of a very high dimensional vectors from EEG streaming headset, of at least eight dimensions, 16 to 47. These dimensions represent an electrolytes position on the brain. The vectors have normalized numerical values that represent the brain electrical impulses. The EEG offers both high dimension and velocity data set for experimental purpose of OAP. Since the brain regions show an interesting aspect of Information Flow Connectivity (IFC), a meaningful pattern can be extracted with PCA. Therefore, the reasons for particular choice of an online PCA application on EEG data are some meaningful and interesting subjects in a field of medicine, security and entertainment. This application can be combined with a field of pattern recognition aiding anomaly detection of EEG such as brain medical conditions, detecting deceptions and video games.
controller. However, the case, the purpose of this thesis is not to investigate the neurological aspects of the resulting visualization.

4.3 Testing: Evaluation of OAP

The successful implementation of OAP algorithm coupled with defined methods of the thesis provides an evaluation environment of the implementation. The objective of this section is to test the results using at least two different approaches. The first approach is evaluating the OAP with an online PCA algorithm that uses an eigenvalue decomposition (O-SVD). This method will prove the hypothesis of time complexity and memory usage and shows how efficiently the OAP algorithm computes its projection matrix. The second approach is testing and evaluating the accuracy of OAP algorithm using different learning rates and most importantly different accuracy factors. This setup of the OAP coupled to the visualization tool will be evaluated in a biological relevant setting, relating the visualization to insights from relevant functional neuro-sciences such as movements, sleeping, dancing and eye closing.
5. Implementation

This chapter describes the implementation approach used and accomplished based on the choice of methods argument mentioned in the previous chapter. It focuses on describing how the methods were put into implementation to bring a prototype solution. This implementation gave a grounding in setting up the experimental simulation, which together with results will be discussed in the next chapters. The crucial blocks of code made to develop the prototype can be found in Appendix A.

The system prototype is made up of different modules, programming languages and tools. The challenge was to bring all these modules into compatibility of working together and communicating. Figure (5.1) illustrates a concise architecture of the system architecture.

The rest of next sections in this chapter will provide a detailed description of the implementation and breaking down of the system components shown in system architecture Figure (5.1). The core of the system is implementation of online PCA version algorithm known as Online Angle Preservation (OAP).

### 5.1 OpenBCI Headset

OpenBCI is an open-source brain computer interface (BCI) company that provides tools for sampling electrical activity of the body. In the context of this project, a 3D-printable EEG headsets with 16 sensors, named *Ultracortex "Mark IV" EEG Headset* used to get a live research-grade EEG recordings [17]. The OpenBCI headset supports two different integrated chip technologies namely Cyton board and Ganglion board. The cyton board technology
allows up to EEG 16 channels and have fewer noisy frequencies compared to ganglion board which can support at maximum eight channels. Specifically, this project setup used cyton board to retrieve 16-channels serving the purpose of a very high dimension. Below is a brief enumeration of steps to connect the device with a Computer:

1. To get started, it is recommended having the listed drivers in the Open-BCI page [18].

2. By plug in an OpenBCI USB Dongle, and turn the USB Dongle switched to GPIO 6 and not RESET.

3. Power on the cyton board to PC (not OFF or BLE). Launch an OpenBCI_GUI software to test if the connection is on.

To demonstrate the designated setup of the project, the next sections in this chapter reflect the consecutive steps. A more detailed description of the headset and establishing the connection can be found from the OpenBCI getting start page [18].

5.2 LSL Library

LSL is a unified system with collection of libraries handling networking and time synchronization [19]. As shown in Figure (5.1), these are low level libraries handling communication with hardware device, where wrappers can be developed on top of it. The LSL Library contain the foundation classes supporting different programming languages, whereas in this thesis project, Java was used. To build the LSL environment for the purpose of streaming from the OpenBCI helmet (headset), the following are procedures:

Figure 5.2. The left figure shows LSL tree directory and the right figure shows LSL built libraries tree[9]
• LSL directory files: These are prerequisites open source files that contain basic classes and can be downloaded from LSL [19]. A hierarchal tree of the files before building the lsl library are shown in Figure 5.2.

• Building the LSL Libraries (liblsl): The libraries are configured with CMake inside LSL working directory. The simple approach is done by the following consecutive commands on a terminal cmake-gui . . ., then Configure (selecting compiler and click finish), and last Generate to generate the libraries. Once built, the libraries and binary files will be created as shown in Figure 5.2. At this point, the environment has been set to develop applications.

5.3 Java-API (wrapper)

Java being one of the language supported by LSL library, the streaming API (wrapper) was hence written in Java language per reasons stated before. Basically, the wrapper is handling the communication between the LSL libraries and the module on top of it. Categorically, the API task can be grouped into the following functions;

• Sending streams (Push): The first simple task of API is to draw the streams from EGG and send the streams to another co-module of the API. A sending task is typically accumulated in one as a Java class. The Sending class communicates with main classes in the libraries by defining some attributes of the stream such as nature of streams, data type of the streams, number of streaming channels and name of the EEG device. At this point the streams are fed into the next module.

• Receiving streams (Pull): This can be viewed as an important functionality of the API. The module receive the streams from subordinate module of the API. In abstract, this function is mapped as a Java class as well. When receiving the streams, this class pre-process the data before feeding the designated result to another component of the system. A simple form of preprocessing could be specifying streaming window size, normalization and even much more complex computation of the streams.

The Java API can be replicated into a class diagram as shown in Figure ???. At this point, it is sufficient to replicate the result and get the streams in to the terminal. The Java codes for sending and receiving classes can be found in Appendix A for more reference.
5.4 SA-Engine

The *sa.engine* is a system toolbox from (*Stream Analyze company, Sweden*) for developing application that handles and process real-time streams of data. The *sa.engine* system supports local edge analytics of data streams in a real-time directly on the edge devices. In particular, it is cable of processing high-speed data by filtering the streams for analysis such as OPCA. The *sa.engine* supports a query language called OSQL (Object Stream Query Language) for compact streams processing such as machine learning computational techniques and database queries. The *sa.engine* supports plenty of built-in and any procedural functions that can be easily executed with a single call. The following sub-sections will cover the steps for connecting the previous section of Java-API with *sa.engine*, implementation of OAP and then to the next visualization section (a visual analyzer).

5.4.1 API-*sa.engine* Connection

To replicate the implementation, the *sa.engine* system for Windows, OSX and Linux is available free of charge for non-commercial use[20]. Among other tools, the *sa.engine* is built-in with some Java classes that can be implemented in any Java application. This enabled the connection between the Java streaming application and the *sa.engine* in this thesis project. To accomplish the connection approach, this sub-section presents steps for obtaining the objectives described in the section:

1. The first step is to interface the previous described Java-API with *sa.engine*. The Java-API is responsible to interface the OpenBCI device by extracting the streams. In the Java-API, the *sa.engine* Java packages are com.sa.callin.* and com.sa.callout.*. The Java-API is defined with the `ReadF` (*CallContext ctx, Tuple tpl*) main method (*sa.engine* method) which pulls the streams from the SendData class and pushes them to *sa.engine* system.

2. In the *sa.engine*, it follows the implementation of an OSQL function (supported by the function signature) that defines the types of the arguments and output of this function. The function signature reads from the Java-API as a foreign function and receive the streams as strings. Finally, a function signature produces a bag of vectors and feed the bag to the defined function called `bci_stream()`. To successively run this set-up by getting the live streams, the following commands can be executed after compilation of Java classes.

- Pulling EEG Data(Calling Java program) in *cmd*:
  ```java
  java -Djna.nosys=true -cp "jna-4.2.2.jar:src" examples.SendData
  ```
• Pushing EEG data (Calling OSQL function as java foreign function) in sa.engine: bci_stream()

At this point, the bci_stream() function converts bag of vector of number to a stream of vector of number that will be printed in a REPL command line. The next step is the computational approach of OAP algorithms using OSQL.

5.4.2 OAP implementation

The OAP implementation step is used to compute the OAP algorithm 4 described in section 4. After successively receiving the streams from Java-API into sa.engine command line tool, it follows the development of the OSQL query implementing the OAP. The concise implementation of the algorithm using procedural function in sa.engine is given as follow,

1. The implementation starts by generating a random 100-sample (of 16-dimensions) data points \((X_T)\) that will be projected by the projection matrix computed at each iteration.

2. A random initial projection matrix, \(P_0\) of (of 16-dimensions by 2).

3. The OAP implementation function takes two argument as a stream of vectors of 16-Numbers from OpenBCI devices (supported by function bci_stream()), and the accuracy factor \(\epsilon\). The function output is a Vector of vector representing 100-samples of 2 principle component at each iteration.

• create function _oap_pca(Stream of Vector of Number s, Number \(\epsilon\)) -> Vector of vector

4. Iteratively, striding the stream into a window of two vectors, \(X_t = [x^1, x^2]_t\).

5. For each \(X_t\), calculate a loss function, \(l(X_t; P_{t-1})\), from given set of formula.

• \(l(X_t; P_{t-1}) = \| X_t^T X_t - X_t^T P_{t-1}^T P_{t-1} X_t \|_\infty\)

6. Together with a sign of loss function, \(\Sigma_t = \text{sign}(l(X_t; P_{t-1}))\).

7. The implementation will update the projection matrix if the loss function is greater than the accuracy factor \(\epsilon\), \(l(X_t; P_{t-1}) \geq \epsilon\). The update is given by:

• \(P_t = \pi_t \left( P_{t-1} - (X_t X_t^T) P_{t-1} \Sigma_t \right)\),

• such that \(\pi_t\) is a normalization factor (learning rate) given that \(\| P_t e_i \|_2 = 1\) for all \(i = 1, ..., D\).
8. At each iteration of the stride, the function returns updated projection matrix $P_t$ if the loss if greater than the accuracy factor, otherwise nothing is update. Hence the image (projected data points), $X'$ is given as;

$$X'^T = P_t^* X_T$$

9. Furthermore, the function can be called from sa.engine command line tool and will produce a vector of vector of 100 samples of 2-principles components. The function call with appropriate arguments is given as

- `_oap_pca(bci_stream(), 0.1);`

where `bci_stream()` is a function that inject streams to the function and 0.1 is the accuracy factor $\varepsilon$.

This function implementation will print the output in the sa.engine REPL command line tool for as long as the input stream argument is active. Each vector $X'^T$ has 100 data point of 2-principal (x,y) component. For the purpose of visualization, sa.engine comes hand in hand with a Client-API tool called Visual Analyzer (VA).

### 5.5 Visual Analyzer: Client API

The sa.engine core system has a REPL (Read Eval Print Loop) module called Visual Analyzer (VA). In this project implementation, the VA is used as a basic client API. This is a StreamAnalyze web application that handles the development of streams in a very fast and flexible fashion. This client API provides the environment where the OSQL expressions are interactively queried, executed in a real-time and their results are immediately printed. Especially, the Visual Analyzer supports different methods of graphical visualization such as text, line plot, scatter plot, bar plot and many more. Additionally, the client API allows users to access and query the sa.engine database. The sa.engine database stores the streams, built-in functions and procedural functions (osql defined functions) [20].

After the installation of sa.engine from StreamAnalyze [21], the VA can be simply started by command `va();` in the command prompt of sa.engine or directly by starting the sa.engine application. A web browser will start by showing a user graphical interface (GUI) where the expressions are specified and results are visualized. Figure 5.3 and 5.4 shows different sections of a GUI of VA. The blank icon in the GUI provides an empty screen for executing the expressions. Some basic commands and functions used on VA to query the database and visualization of the streams in this implementation:

- `va();`: A built-in function to initialize the VA from sa.engine command prompt.
- **pcascore()**: A built-in command to project input streams onto the eigenvectors corresponding to the greatest eigenvalues. The implementation procedure is filtering the stream into designated ones, for visualization purpose.

- **winagg()**: A built-in function to form the size of streams, and striding the window into sliding/tumbling.

- **_oap_pca(stream s, \( \varepsilon \))**: A procedural function, that evaluate the OAP PCA version where \( \varepsilon \) is the accuracy factor.

![Figure 5.3. GUI of VA showing home page icons on a browser window:](image)

**Figure 5.3.** GUI of VA showing home page icons on a browser window:

![Figure 5.4. The left figure shows a VA visualization of scatter plot in x-y plane and the right figure shows a 1-channel stream of EEG](image)

**Figure 5.4.** The left figure shows a VA visualization of scatter plot in x-y plane and the right figure shows a 1-channel stream of EEG

The VA has given approach that enables local edge analytics of EEG data streams. Streams can be analyzed remotely from the OpenBCI headset registered with sa.engine stream servers on the Internet [22]. This makes the implementation interesting for IoT performance, especially when the Java-Wrapper for sending and receiving the streams able to communicate over the network. On the other hand, the VA is currently under the concrete update to a more operational version. One of the limitation is inability to perform multi tasking. In particular missing the instance for visualizing multi-channel streams in one window.
5.6 Unexpected Issues

In the implementation of the system setup, quite number of issues were unexpected or never considered prior to research work.

- Since the sa.engine goal is to have platform independent approach. It does not provides support to python as programming language. For this reason, it was impossible to extend the interface to allow a state-of-the-art python toolbox with various tools of machine learning such as scikit-learn to work directly with the EEG recording device. This however, made the goal possible to achieve with OSQL implementation.

- The sa.engine has fewer built-in linear algebra functions. For instance, most of the mathematical functions such as matrix multiplication, transpose and absolute value are not available. These kind of functions required independent implementation which made the work a bit intense than expected or even might affect the performance.
6. Results and Evaluation

This chapter describes the result findings obtained from the research question. The experimental setting was replicated from system implementation as illustrated in the previous chapter. The aim of this investigation is basically to evaluate the online PCA for OAP algorithm based on statistical performance, time and space complexity. In particular, the evaluation study involves a comparison between the OAP algorithm under study and the online PCA adopting a traditional setting of eigenvalue decomposition (SVD).

6.1 Simulation

Experimental study was carried out to compare the numerical performance of the online PCA for OAP algorithm under study with an online incremental SVD. The data source used for this experimentation was the random EEG streams from Ultracortex "Mark IV" EEG Headset. Each stream is a vector of 16 dimension, at a sampling rate of 100 Hz. In this simulation a constant random initialized 100 sample data ($X$) with the similar features of EEG stream was used in the projection using the top eigenvectors of the streaming data in each iteration. The second random initialization is a projection matrix with the similar features of the streaming data and of two vector length window ($P_0$). The number of principal components D of the estimated eigenvectors two, four and seven were tried in the simulation. The experiment conducted under this simulation took place on MacBook Pro machine of operating system Sierra version 10.13.13 (64bit), Intel(R) Core i5 CPU 2.90GHz with 8GB RAM. For clear intuition, Figure 6.4 shows the graphical representation of PCA for OAP results.

In evaluating the statistical accuracy of the OAP algorithm, the method used for measuring performance is a mean squared error (mse) of the sample $X$ from the image sample $X'$ projected by an inverse of top eigenvector $P_{t'}$ called Reconstruction error. The goal is to reproduce the original sample from the sample image, using the estimated top eigenvectors, by minimizing the error such that, the results are not too different. Numerically, the accuracy measure of the eigenvectors estimation is given by

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (X_i - X'_i * P_{i-1'})^2$$  \hspace{1cm} (6.1)
6.2 Comparison to IPCA (SVD)

6.2.1 Statistical accuracy

From equation (6.1), evaluation results suggest that the convergence of the OPCA for OAP algorithm by far most depends on the accuracy factor $\epsilon$. Following many experimental simulation, the best $\epsilon$ setup found is 0.0001.

![Figure 6.1. Eigenspace MSE estimation (reconstruction error) for the first $D = 2$ eigenvectors, and $n = 100$, $\epsilon = 0.0001$](image)

As seen in Figure (6.1), these results show that the IPCA (SVD) attain high accuracy in the computation of eigenspace compared to it’s competitor OAP under study. Overall, through out different $\epsilon$ of OAP, IPCA (SVD) slightly out performs its competitor even far much converges faster.

6.2.2 Time and space complexity

The two investigations made in this case were time precision and space complexity with principal components $D = 2$ and $4$. The aim of these investigations is a performance comparison between the OAP under study and the traditionally batch PCA under online fashion. Figure (6.2) shows the time and space complexity of the two algorithms [2]. The OAP algorithm has a computational cost that grows linearly with the data and hence can be used fairly with large datasets compared to IPCA (SVD) which its computation depends on both size and dimension of the data.
### Figure 6.2. Computational cost.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Required Memory</th>
<th>Computation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA (SVD)</td>
<td>$O(nd)$</td>
<td>$O(nDd)$</td>
</tr>
<tr>
<td>OPCA (OAP)</td>
<td>$O(Dd)$</td>
<td>$O(nd)$</td>
</tr>
</tbody>
</table>

Figure (6.3) shows experimental results for operational comparison of the time and space complexity presented in Figure (6.2)

### Figure 6.3. Computation time of online PCA, ($n = 100$, $d = 16$)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$D = 2$</th>
<th>$D = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA (SVD)</td>
<td>0.5</td>
<td>0.9</td>
</tr>
<tr>
<td>OPCA (OAP)</td>
<td>0.2</td>
<td>0.3</td>
</tr>
</tbody>
</table>

From those results, one can say that, when the principal components $D$ of the data is small, both methods perform quite well. Yet, still the OPCA (OAP) out performs the speed of the IPCA (SVD). As $D$ increases, the computation time for IPCA (SVD) doubles, whereas of OPCA (OAP) slightly increase. In other case, trying to increase the size of sample $n$, will even make the IPCA (SVD) worse. This shows how significant the OAP algorithm can be in high dimension, speed and large data.

### 6.3 Overall Performance

The overall performance of OAP and the system in general were evaluated using the real EEG data representing a biological relevant setting. The data were recorded from a person using Two-finger gameplay with deliberately failing controller presented in 47 channels at sampling rate of 512 Hz. The data were selected from an open source database of BNCIHORIZON2020 [23]. The evaluation present the converged PCA plot ($D=2$) for IPCA (SVD) and OPCA(OAP) in Figure (6.4). Both figures show high correlation of the data in x-direction, relating this visualisation to insights from relevant functional neuro-sciences. Such correlation could be deduced by high active set of an EEG from one region of the brain compared to the rest by the presented activity in the data set.

Figure (6.6) evaluate the overall approximate computation time, memory usage and accuracy of the two online PCA algorithms. As can be observed, the IPCA (SVD) requires much more memory than the OAP algorithm since it has higher memory cost as seen in previous section. During the simulation, the size of the data is large enough to cause high CPU computation task, hence slacken the computation or even stuck and failing the process for IPCA (SVD)
case. No doubt some of the latency were caused by independent components of the system such as sa.engine and streamanalyze, which was ignored since are out of scope in the evaluation study. Furthermore, the OAP algorithm out performs its competitor with time and memory, it is more faster and requires much less memory. However the case, the OAP algorithm expected even more faster than the evaluated, but since the implementation used here requires a Gram-Schmidt orthogonalization in high dimension at each iteration slows the OAP a bit less[2]. Overall, the OAP implementation could not defeat the IPCA (SVD) accuracy and converging rate. The IPCA (SVD) seemed to compensate time and memory complexity in attaining high accuracy.
Figure 6.6. Computation time, Memory and MSE performance of Two-finger gameplay dataset (n = 200, t = 10000, D = 2 and d = 47)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (Mils)</th>
<th>Memory (Mb)</th>
<th>Error (MSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPCA (SVD)</td>
<td>3</td>
<td>15</td>
<td>3.54-E4</td>
</tr>
<tr>
<td>OPCA (OAP)</td>
<td>1</td>
<td>1</td>
<td>4.79-E4</td>
</tr>
</tbody>
</table>

Figure (6.5) evaluate reconstruction error using different accuracy factor, $\varepsilon$. The experiment results show that, using the highest value of $\varepsilon$ (which is 1) the reconstruction error remain constant with a value of 0 as seen in the graph ($\varepsilon = 1$, yellow curve shows nothing). This phenomenon suggest that, highest value of $\varepsilon$ causes the algorithm to stuck and fails to converge at all. Reducing the values of $\varepsilon$ improves the accuracy, the best setting observed is of value $\varepsilon = 0.0001$ (blue curve). Reducing further, to $\varepsilon = 0.00001$, will only start to increase the reconstruction error of the OAP algorithm.
7. Discussion

This thesis project was an attempt to investigate an online PCA by implementing a coherent system that compute OAP algorithm. The objective was to solve the stumbling block in the computation of an online PCA, hence the evaluation of the results with existing method such as IPCA (SVD) was vital. The results obtained were put through computational and statistical performance analysis for evaluation purpose. The findings in the evaluation show that the OAP can be reliable enough for efficiency computation in terms of speed and memory in compensation of accuracy and rate of convergence. The following section discuss further these findings.

7.1 Performance

The performance of OAP algorithm to the high dimensional and velocity data has proved to be efficiently enough especially on account of computation time and storage complexity. The OAP performance has remain practical viable even in quadratic time and memory complexity. In the two algorithms under study, the OAP algorithm provide the highest computation speed with least computational memory compared to the its competitor IPCA (SVD). However, it is very sensitive to the choice of \( \epsilon \), learning rate \( \pi_t \) and converge more slowly than the IPCA (SVD). The major dependence on these tuning parameters cost the OAP and poorly compromise between statistical accuracy and computational speed. The fact that the OAP algorithm is time and computational resource friendly, the result still suggest that OAP is significant enough to be used in the relation to applications that have a long run.

On the other hand, the IPCA (SVD) has compensate computation time and storage to attain high accuracy. However, in most cases due to high cpu computational task, IPCA (SVD) frequently enough may cause application to crush or unresponsive. Overall, the IPCA (SVD) can be well utilized under some favorable factors such as low dimension and small space. Uttermost, for a very high dimension and high frequency such as EEG studied, the OPCA (OAP) algorithm cut the point in the speed performance and least memory requirement.
7.2 Convergence

As the results show in the previous chapter, the rate of convergence for the OAP algorithm is still scant. It has been observed very clearly that the convergence rate is proportional to the number of iteration of the algorithm, and much more on the $\varepsilon$ which plays a significant role in the convergence rate. In comparison with IPCA (SVD) which tends to converge very fast, yet still with a bit higher accuracy than the OPCA (OAP). It is important to note that the high $\varepsilon$ makes the algorithm slow to converge, while small $\varepsilon$ makes the rate of convergence a bit faster but slower as already expected. Both of these results are still not optimal compared with IPCA (SVD) performance.

Another important aspect is the learning rate $\pi_t$, in the update of the algorithm. In this thesis, learning rate $\pi_t$, was dynamical chosen such that $||P_t \cdot e_i||_2 = 1$ for all $i = 1, ..., n$. Larger values of $\pi_t$ can produce better convergence rates, however this can lead OAP algorithm to get stuck in local minimum and fail to converge. This scenario can give results that are far from the true eigenvectors (i.e Global optimum).

Finally, the analysis suggest small $\varepsilon$ are best for fast convergence of the algorithm under the best choice of constant learning rate $\pi_t$ to the update function of the algorithm. A good relation between $\varepsilon$ and learning rate $\pi_t$, yet still requires much experimentation for optimal solution.
This thesis project has documented substantial development of a compact system for an online processing of live EEG streams. The objectives of the system is to investigate the researched online PCA algorithm for OAP which can provide a quick update of the projection matrix with a considerably sufficient accuracy and downdate capacity. The main focus on the solution of a kind of an online PCA on a real time series data, in comparison with IPCA (SVD). The results from the experiment show that the proposed method has high calculation speed, friendly memory usage and acceptable efficiency.

The research has still left open question. The results have proved to attain reliable accuracy in the first 10,000 iterations. In contrast to the IPCA(SVD), which tends to converge very fast, yet still with slight higher accuracy compared with OPCA(OAP). Preliminary results, suggest the accuracy factor and the learning rate affect the OPCA (OAP) convergence rate and attaining of sufficient accuracy. What optimal combination of learning rate and accuracy factor would produce the fast convergence of the OAP?

The end solution of the project has produced a compact system implementing the algorithm with considerably high achievements. The solution can easily be replicated to reproduce the results or/and for future work and similar projects. The prototype has been successfully enough and in thought to be opt by streamAnalyze company, Sweden.
9. Future work

The research solution presented in this thesis seems to have left open problems from what has been answered. There are several lines of open questions arising from this work seeking answers. Similarly, numerous different experimental setups, improvements, and expandable ideas have been left out for the future work as the consequence of limited resource availability and lack of time. Future work require deeper evaluation of the adopted algorithm against existing ones, proposal in extending the algorithm, or/and adoption of decaying multipliers such as \( \varepsilon \). This thesis has been mainly focused on the use of OAP for PCA visualization on EEG data. The most of the OAP evaluation used to find the best result where obtained from a sliding window PCA, leaving concrete evaluations with other existing algorithms outside the scope. The following ideas could be adopted:

1. **System Improvement:** The experiment setup do not seem to be satisfactory enough, especially the LSL communication library and the performance of the algorithm under study. The LSL-library is an open-source which seemed to be abandoned. Not much support and updates of the system is given, this makes it very unreliable especially with most recent OS platforms. Much more effort is needed to develop a compact system that is platform independent and easy to replicate the results.

2. **OAP Improvement:** To guarantee the results from Chapter 6, further evaluation study is still required in order to understand the behavior of convergence of the OAP algorithm. Specifically, battling the OAP algorithm against related algorithms such as Accelerate Stochastic Power Iteration discussed in chapter 3. However, since the preliminary results suggest the convergence rate goes hand in hand with \( \varepsilon \), it will be interesting enough to introduce a decaying \( \varepsilon \) and a constant learning rate. It shows a trivial approach of adopting a constant \( \varepsilon \) and linear decaying learning rate for an infinity streaming data.

3. **Interpreting the OAP plot:** An important direction for future work is to visualise the outcome of OAP on the stream in the original head-space. That is, we want to express the mixture of the found principal components in relation to the actual locations of the EEG-electrodes as attached to the subjects’ head. This will enable us to pinpoint the major sources of the recorded EEG activity in a physiological meaningful way.
Regarding the application of OAP algorithm and the system at large, an extension for the near future is the use of more powerful supervised algorithm such as Deep learning (DL) to learn the data representation for the choice of $\varepsilon$. An additional task is to adopt DL to learn the converged patterns of OAP algorithm over different EEG streams under certain study. This will avoid recomputation of OAP over and over again which can be CPU intensive work for very high dimensions.
References


[12] Manfred K. Warmuth and Dima Kuzmin. Randomized online pca algorithms with regret bounds that are logarithmic in the dimension. 2007.


[19] SCCN. lab streaming layer. [Date accessed: 2018-04-20].


