Coherent Averaging Estimation Autoencoders applied to Evoked Potentials Processing

Iván E. Gareis[a,c], Leandro D. Vignolo[a], Rubén D. Spies[b], Hugo L. Rufiner[a,c]

[a] *Instituto de Investigación en Señales, Sistemas e Inteligencia Computacional, sinc(i), Facultad de Ingeniería y Ciencias Hídricas, Universidad Nacional del Litoral - CONICET*
[b] *Instituto de Matemática Aplicada del Litoral, IMAL, Universidad Nacional del Litoral - CONICET*
[c] *Laboratorio de Cibernética, Facultad de Ingeniería, Universidad Nacional de Entre Ríos*

Abstract

The success of machine learning algorithms strongly depends on the feature extraction and data representation stages. Classification and estimation of small repetitive signals masked by relatively large noise usually requires recording and processing several different realizations of the signal of interest. This is one of the main signal processing problems to solve when estimating or classifying P300 evoked potentials in brain-computer interfaces. To cope with this issue we propose a novel autoencoder variation, called Coherent Averaging Estimation Autoencoder with a new multiobjective cost function. We illustrate its use and analyze its performance in the problem of event related potentials processing. Experimental results showing the advantages of the proposed approach are finally presented.

Keywords: Coherent Averaging, Artificial Neural Networks, Event Related Potentials, Brain Computer Interfaces, Autoencoders

1. Introduction

Coherent averaging (CA) is a widely used technique to recover a repetitive response masked by uncorrelated noise. It dates back to the early 50's [1] and it

*Corresponding author
Email addresses: ldvignolo@sinc.unl.edu.ar (Leandro D. Vignolo), rspies@santafe-conicet.gov.ar (Rubén D. Spies), lrufiner@sinc.unl.edu.ar (Hugo L. Rufiner)

has become a classic processing strategy in several fields. It consists on averaging several responses to a repetitive stimuli, synchronizing their phases according to the stimulus time.

One of the main assumptions of CA is that the signal of interest $x : [0, T] \to \mathbb{R}^s$ (where s is the number of sensors) can be modeled as $x(t) = r(t) + a(t)$, where $r(t)$ is the response to the stimuli and $a(t)$ is the additive noise. If $r(t)$ is time invariant and $a(t)$ is uncorrelated with the response, stationary and with zero mean, CA improves the signal-to-noise ratio (SNR) of $x(t)$. The variance of an estimation of $r(t)$ obtained using CA will be reduced by a factor $\sqrt{K}$, where K is the number of samples used for averaging [2]. However, adding realizations is time consuming since the process under study must be repeated to record each realization. In general it is highly desirable to obtain estimations in the least possible time, and when K is large, the cost can become prohibitive.

The CA technique has been successfully used in many applications. For instance, it was used for noise reduction in mechanical signals for failure detection [3], in radar applications to improve the SNR [4], etc.

In the context of biomedical signals, particularly for electrocardiographic signals, CA has been used for detection of ventricular late potentials, fetal electrocardiogram estimation, prediction of ventricular arrhythmias, monitoring during surgery, and other direct applications [5], as well as for estimation of template waveforms [6].

Since its beginning CA has been applied for estimating the morphology of evoked potentials (EP) [7]. This is a task CA is well suited for, since EPs are fairly repetitive, they can be elicited, so there is a way to know precisely when they will happen, and they are masked by noise which is uncorrelated with the stimuli with a usually very low SNR.

A brain computer interface (BCI) is a device that provides a direct link between the brain of its user and a computer [8]. A BCI can be based on different physiological phenomena, like the somatosensory event related potential (ERP) called P300. When a P300-based BCI uses electroencephalographic (EEG) signals, CA is commonly used to improve the SNR. The translation algorithms,

which convert the brain signals into commands, occupy a central role in the BCI and make use of machine learning algorithms.

An artificial neural network (ANN) is a machine learning algorithm consisting of simple interconnected parts called neurons. ANNs where more than one layer of neurons are stacked, date back to the 60's and were popularized with the backpropagation algorithm (BP) [9]. A particular type of ANN to consider is the so-called autoencoder (AE), which tries to replicate the input in the output with at least one constraining hidden layer to prevent the network from learning the identity function. The basic AEs can be modified by introducing variations either in the network architecture and/or in the training strategy. The simplest form of AE is that of a feedforward ANN with one hidden layer with less units than the input (this is the simplest constraint) trained using BP.

Even though feedforward ANNs with at least one hidden layer have long been known to be universal approximators [10] they were outperformed and lost popularity to other machine learning techniques. Hardware advances, the availability of large amounts of data and deep learning techniques [11], have allowed the resurgence of ANNs, giving them back a protagonical role in the last 10 years. Moreover, deep ANNs captured the attention of the whole scientific community after improving well known benchmarks reached by traditional techniques and wining several international application competitions. An historical survey on ANNs and particularly on deep learning can be found in [11].

The recent resurgence of ANNs has been reflected in the BCI community in some ideas that were adapted to solve EEG pattern recognition problems. In [12] convolutional neural networks (CNN) make use of the spatial and temporal correlations in EEG signals to learn spatial and temporal filters for optimizing recognition rates. In [13] deep belief networks were used as the weak classifiers of an AdaBoost ensemble to improve the recognition rate of motor imagery data.

To cope with missing data in EEG signals (which can be the result of extreme artifacts when a whole segment of signal is discarded), an AE variant called denoising autoencoder (DAE) was used by Li et al. in [14]. In this work the authors use simulated data to show that the spectral power can be satisfactorily

estimated from incomplete data, and real motor imagery data, to show that the results obtained with the AEs are comparable with those obtained using support vector machines (SVMs).

Based on the classic CA and AE methods, in this article we propose a new algorithm to improve the classification and estimation of small repetitive signals masked by large noise. The cost function of this new ANN, which throughout this work we shall refer to as Coherent Averaging Estimation Autoencoder (CAEA), includes a reconstruction, a discrimination and a sparsity terms, making its training a complex multiobjective optimization problem. A BCI problem is proposed as application of this new method. Here the CAEA is used to classify and process both real and artificially generated data. A search for suitable hyperparameters is also performed, and guidelines for their tuning are provided.

In Section 2 an introduction to BCIs is given and, within this context, the importance of the processing of small signals masked by large noise is explained. Some basic background is introduced regarding the building blocks of the CAEAs in Section 3. In Section 4 the proposed architecture is presented and its relevant features are explained. The data and criteria with which the CAEA was tested are presented in Sections 5 and 6 respectively. In Section 7 the strategy followed to adjust the ANN hyperparameters is depicted. The usage of the CAEA is exemplified and the corresponding results presented in Section 8. Finally some closing remarks and conclusions are presented in Section 9.

2. Brain Computer Interfaces

A BCI can be roughly defined as a system that translates brain signals into new kinds of outputs. Roughly speaking a BCI can be divided into four blocks [15]. The first block (I) is the brain signal acquisition system. The second block (II) consists of the feature extraction system, whose purpose is to enhance the discriminative information while discarding the useless information contained in the brain signals. The third block (III) is the translation system which classifies the features provided by the previous block into the prediction

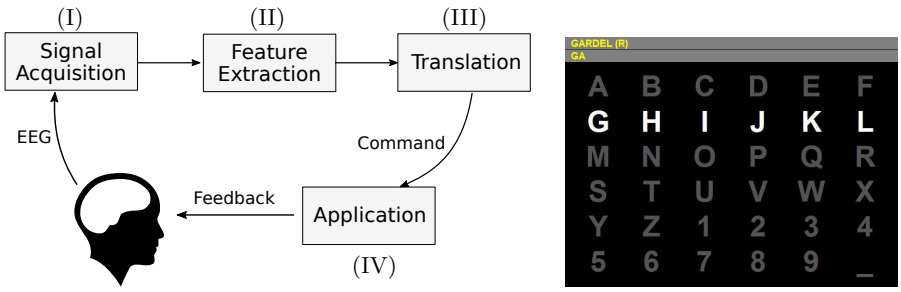
of the user's intent. The second and third blocks are often combined since several classification algorithms include their own feature extraction methods. The fourth block (IV) is the system which executes the predicted command or intention. A block diagram of a general BCI is shown in Figure 1a. This article focuses mainly on the second block, which is essential to improve the classification and the overall performance of the system.

Although the use of EEG signals has several advantages over other brain-generated signals (such as electrocorticographic signals) it presents several drawbacks such as their low SNR, they can be contaminated by artifacts coming from eye movements or from electromyographic activity, and they have low spatial resolution [16]. These drawbacks hinder the implementation of an efficient electroencephalography based BCI and promote the development of suitable methods to extract useful information from the data.

The performance of a BCI is highly dependent on the feature extraction technique and on the classification method used to predict the user's intention. Different approaches have been followed for optimizing these processes. Comprehensive reviews up to 2007 on classification and feature extraction methods proposed for BCIs based on electroencephalography can be found in [17] and [18], while a more recent review can be found in [19]. Much further work has been done in the area. Worth mentioning are the articles on sensor selection methods and spatial filtering [20, 21, 22, 23] and the feature extraction and dimensionality reduction methods [24, 25, 26]. Lately Bayesian methods have also been successfully applied to EEG signal processing [27, 28], particularly in the context of BCIs [29, 30, 31, 32].

2.1. P300-Speller and Coherent Averaging

When an infrequent or particularly significant auditory, visual or somatosensory stimulus is mixed with frequent or routine stimuli, an ERP is typically evoked over the parietal cortex. This phenomenon can be used to implement a BCI called P300 speller, in which the user is prompted to select symbols from a matrix in a computer screen. The P300 speller, was introduced for the first time



1a 1b

Figure 1: (a) General BCI block diagram. (b) P300-Speller.

by Farwell and Donchin [33]. Here the subjects are presented with a 6×6 character matrix as illustrated in Figure 1b. Several variations of the P300 speller have been introduced, such as the single character, the checkerboard, the region based and the mismatch presentation paradigms [34, 35].

To determine the chosen symbol, ERP based BCIs must be able to determine if a signal collected after an stimuli (named a post-stimulus signal) contains an ERP or not. Although a perfect post-stimulus signal classification rate would result in a perfect symbol prediction rate, no classification method is close to achieving this goal. To improve prediction rate several blocks of flashes can be presented and all the post-stimulus signals corresponding to the same flashed row/column coherently averaged [36]. Although this process attenuates uncorrelated noise and facilitates classification, it is usually highly time consuming. If a good accuracy rate could be achieved for single trial ERPs classification (i.e. with just one block of flashes), the information transfer rate of the BCI could be significantly increased. One way of achieving this goal is by approximating the output of an hypothetical coherent averaging method by means of an ANN with just one post-stimulus signal as input. More details on this idea are given in Section 4.

3. Relevant Algorithms

3.1. Basic Units

The basic artificial neurons used in this article implement the well known weighted averages with an activation function (AF) σ given by:

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1}$$

where $w = (w_1, ..., w_n)^T \in \mathbb{R}^n$ is the weights vector, $b \in \mathbb{R}$ is the bias and $x = (x_1, ..., x_n)^T$ is the input. The identity and sigmoid units were used as AFs.

3.2. Softmax Units

The softmax function (SF) is a generalization of the logistic function from one to several dimensions [37]. As the logistic function, the SF can be associated to probabilities and can be implemented using an ANN design. A softmax artificial neural network (SNN) tipically asociates each class to an output unit. Each output can be interpreted as the relative probability that the input pattern corresponds to a given class. The output of the ℓ^{th} softmax unit in a SNN with M outputs is the function $h_\ell : \mathbb{R}^n \times \mathbb{R}^{M\times n} \to \mathbb{R}$ defined by:

$$h_\ell(x; \Gamma) \doteq \frac{e^{\Gamma_{\ell,:}\, x}}{\sum_{j=1}^{M} e^{\Gamma_{j,:}\, x}}, \tag{2}$$

where $\Gamma \in \mathbb{R}^{M\times n}$ is a matrix whose components are the softmax weights and $\Gamma_{j,:}$ is its j^{th} row.

Let $X \doteq (x^1, ..., x^N)$ and $y \doteq (y_1, ...y_N)^T$ where $x^i = (x_1^i, ..., x_n^i)^T$ and y_i are the i^{th} sample and its corresponding label, respectively. Let also $\mathbf{I}_j(y_i) = 1$ if $y_i = j$ and 0 otherwise. The cost associated with this network is then the function $J_S : \mathbb{R}^{n\times N} \times \mathbb{R}^N \times \mathbb{R}^{M\times n} \to \mathbb{R}$ defined as:

$$J_S(X, y; \Gamma) \doteq -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} \mathbf{I}_j(y^i) \log\big(h_j(x^i; \Gamma)\big), \tag{3}$$

where N is the number of samples and M is the number of output units of the SNN or the number of categories in the problem.

3.3. Basic Autoencoder

As mentioned before, basic AEs are ANNs that try to replicate the input in the output, with at least one constraining hidden layer to prevent the network from learning the identity function. Let $\Theta \in \mathbb{R}^{(n+1)\times m}$ and $\Phi \in \mathbb{R}^{(m+1)\times n}$ be defined as $\Theta = [W_\Theta^T, b_\Theta]^T$ and $\Phi = [W_\Phi^T, b_\Phi]^T$ where $W_\Theta \in \mathbb{R}^{n\times m}$, $b_\Theta \in \mathbb{R}^m$, $W_\Phi \in \mathbb{R}^{n\times m}$ and $b_\Phi \in \mathbb{R}^n$ are the weight matrices and bias vectors for the encoder and the decoder, respectively. Let $f(\cdot;\Theta) : \mathbb{R}^n \to \mathbb{R}^m$ be the encoder function mapping the inputs to the code space, $g(\cdot;\Phi) : \mathbb{R}^m \to \mathbb{R}^n$ the decoder function mapping the code back to the input space, and $L : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ the loss function. With this notation, and using the quadratic error as loss function (i.e. $L(x, y) = \|x - y\|^2$), the cost function $J_A : \mathbb{R}^{n\times N} \times \mathbb{R}^{(n+1)\times m} \times \mathbb{R}^{(m+1)\times n} \to \mathbb{R}$ is defined as:

$$J_A(X;\Theta,\Phi) \doteq \frac{1}{N}\sum_{i=1}^{N} \left\| g\big(f(x^i;\Theta);\Phi\big) - x^i \right\|^2 \tag{4}$$

where X and N are as before.

3.4. Sparse Autoencoder

The simplest way of constraining the hidden representation of an autoencoder is by reducing the number of features of the code (i.e. by making $m < n$). Another (non-exclusive) constraint consists of enforcing sparsity over the code. Although there are several ways of promoting sparsity, a widely used approach consists of using the Kullback-Leibler divergence (KLD) to define the following sparsity promoting penalization term:

$$P_S(X,\Theta;\rho) \doteq \sum_{j=1}^{m} KL(\rho\|\hat{\rho}_j) = \sum_{j=1}^{m} \left(\rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \right) \tag{5}$$

where ρ is the desired average activation and $\hat{\rho}_j = \hat{\rho}_j(X;\Theta) = \frac{1}{N}\sum_{i=1}^{N} f_j(x^i;\Theta)$ is the average activation of the j^{th} hidden unit over the training set. To ensure that (5) is well defined, it is convenient to exclude 0 and 1 as possible values of f_j, $\forall j = 1, ..., m$.

3.5. Denoising Autoencoder

DAEs are trained to reconstruct the clean signals $x^i \in \mathbb{R}^n$ from their corresponding artificially corrupted versions $\tilde{x}^i \in \mathbb{R}^n$ [38]. In particular, choosing the quadratic square error as loss function, yields the following cost function:

$$J_{DAE}(X, \tilde{X}; \Theta, \Phi) \doteq \frac{1}{N} \sum_{i=1}^{N} \left\| g\big(f(\tilde{x}^i; \Theta); \Phi\big) - x^i \right\|^2, \tag{6}$$

where X, Θ and Φ are as previously defined and $\tilde{X} \doteq (\tilde{x}^1, ..., \tilde{x}^N)$.

DAEs are suitable to find robust features for clean patterns. However in cases where the original signals are already significantly corrupted by noise, further corruption does not necessarily improve the representation.

4. Coherent Averaging Estimation Autoencoder

A simple model of ERPs assumes that every stimulus generates the same waveform with the same latency, masked with uncorrelated noise which is also uncorrelated with the stimulus. The CA method consists of averaging several realizations of the same process, synchronizing them in relation to the stimuli, what attenuates the amplitude of the noise, so enhancing in-phase repetitive patterns. Although it is common to use CA in order to improve SNR of ERPs, the acquisition of each sample is significantly time consuming which, in the context of BCI, reduces the attainable bit rate. Usually, for BCI applications, between five and fifteen samples are averaged. However, in this work we intend to use only one trial to perform the classification. For this, we propose a method to estimate coherently averaged signals using only one signal.

We define a CAEA as an ANN with one hidden layer in which the number of outputs and inputs coincide. In this work we propose to train CAEAs as feature detectors. Unlike the AEs, in which the input is set as the desired output, a coherent average of inputs of the same class is presented. Following this idea, the network cost term due to reconstruction is defined as:

$$J_R(f(X; \Theta); \Phi) \doteq \frac{1}{N} \sum_{i=1}^{N} \left\| g\big(f(x^i; \Theta); \Phi\big) - \frac{x^i + \sum_{j \epsilon R_i^K} x^j}{K+1} \right\|^2, \tag{7}$$

where R_i^K denotes a random subset of K indices corresponding to K different patterns (columns of X) of the same class as x^i, excluding x^i itself. This cost function is similar to the one corresponding to the DAEs (6), except that instead of corrupting the input, we estimate a denoised output and propose it as target.

In CAEAs, we use as encoding function $f(x;\Theta) = \sigma(W_\Theta^T x + b_\Theta)$ (here, the AF σ applied to a vector is meant to be its componentwise action), where $W_\Theta \in \mathbb{R}^{n\times m}$ is the matrix of weights connecting the input and the hidden layer and b_Θ is the bias vector (as defined in Section 3.3). The components of $\sigma(W_\Theta^T x + b_\Theta)$ are called hidden features. Analogously, the decoder function, defined as $g(f;\Phi) = \phi(W_\Phi^T f(x;\Theta) + b_\Phi)$, where ϕ is another AF, transforms the feature representation back to the input space. Here, one should pay special attention to the fact that the range of the AF ϕ must contain all possible values of the input x. After training, the decoding layer may be discarded.

In order to ensure that the CAEAs maintain discriminative information, their latent representations are taken as inputs for softmax outputs. The ℓ^{th} softmax unit output (see (2)) is fed with the CAEAs hidden layers activations, that is:

$$h_\ell(f(x;\Theta);\Gamma) = \frac{e^{\Gamma_{\ell,:}\, f(x;\Theta)}}{\sum_{j=1}^{M} e^{\Gamma_{j,:}\, f(x;\Theta)}}, \tag{8}$$

where, as before, Γ is the softmax weight matrix. The cost term associated to the discrimination power is then defined as:

$$J_D(f(X;\Theta), y; \Gamma) \doteq -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} \mathbf{I}_j(y_i) \log\big(h_j(f(x^i;\Theta);\Gamma)\big), \tag{9}$$

where M, Θ, Γ, y and $\mathbf{I}_j(\cdot)$ are all as previously defined.

During training the softmax error signal is backpropagated through its connections to the encoding weights, steering them towards a discriminative representation. As for the decoding layer, the softmax layer may be discarded after the CAEA is trained.

The basic structure of a CAEA is presented in Figure 2a. The inputs are taken from the data and transformed through $f(\cdot;\Theta)$ to get the feature representation. The feature representation is then transformed through $g(\cdot;\Phi)$ which

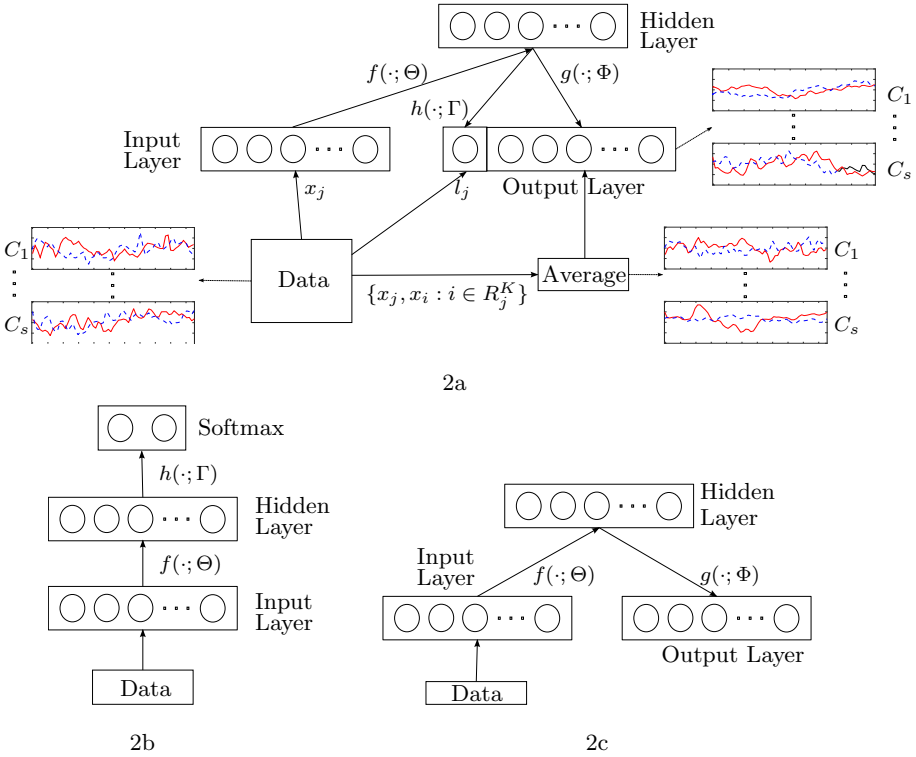


Figure 2: (a) CAEA during training. Target samples are represented with solid blue lines and no target samples with red dashed lines. (b) CAEA during operation used for feature extraction for classification. (c) CAEA during operation used for estimation.

attempts to get a denoised version of the presented input. The desired denoised version is estimated through CA. The feature representation is also used to feed the softmax layer $h(\cdot;\Gamma)$, which tries to predict the corresponding label. Samples of target and non-target input and output signals as well as their corresponding desired outputs obtained through CA are depicted in the aforementioned Figure. Figures 2b and 2c depict CAEA during operation, for classification and estimation, respectively.

A regularization penalty is applied to all the weights in the network and a sparsity penalty is applied to the hidden representation layer. With all the

restrictions the final network cost function ends up being:

$$J(X, y; \Theta, \Phi, \Gamma) \doteq J_R(f(X; \Theta); \Phi) + \gamma J_D(f(X; \Theta), y; \Gamma) + \beta P_S(f(X; \Theta); \rho) + \lambda_\Theta \|\Theta\|^2 + \lambda_\Phi \|\Phi\|^2 + \lambda_\Gamma \|\Gamma\|^2, \tag{10}$$

where γ, β, λ_Θ, λ_Φ and λ_Γ are all positive hyperparameters adjusting the relative weight of each term and P_S is the sparsity promoting penalizer defined in (5). The training process of the CAEAs, which is the typical for AEs, is depicted in Algorithm 1, below. In general, the stopping condition is met if, for a given number of subsequent iterations, the cost computed using the validation data does not improve.

Depending on the number of hidden units and the size of the input patterns the number of parameters in the network could end up being quite high. In fact, there are in principle $m(M + 2n + 1) + n$ free parameters in the model, corresponding to the components of encoding, decoding and softmax weight matrices and vectors, namely W_Θ, b_Θ, W_Φ, b_Φ and Γ. Hence, any reasonable assumption leading to a reduction in the number of free parameters is highly desirable since, on one hand this would in turn, reduce the computational cost of the optimization processes for finding the optimal parameter values, and on the other hand it would diminish the chance of overfitting. One way of reducing the number of free parameters is by enforcing symmetry in the encoding and decoding matrices, i.e. by imposing $W_\Theta = W_\Phi^T$. This assumption is reasonable because, since both the original patterns and their averages have similar waveforms and belong to the same space (i.e. $\mathbb{R}^n$), the decoder action could be thought of as the inverse of the encoder action.

5. Databases

To adjust the parameters and test the performance of the proposed network we used two data sets, which are described below.

Algorithm 1 CAEA training

Initialize $\Theta^0, \Phi^0, \Gamma^0$

stop $\leftarrow$ False

$i \leftarrow 0$

while *stop* = False **do**

Pick N samples from the training set and construct inputs minibatch X^i

Pick the corresponding N labels y^i

Coherently average X^i to construct desired outputs minibatch Y^i

Compute the cost J^i (10) and its gradient $\nabla_{(\Theta,\Phi,\Gamma)} J^i$

Compute optimization steps Δ^i_Θ, Δ^i_Φ and Δ^i_Γ through J^i and $\nabla_{(\Theta,\Phi,\Gamma)} J^i$.

$\Theta^{i+1} \leftarrow \Theta^i + \Delta^i_\Theta$

$\Phi^{i+1} \leftarrow \Phi^i + \Delta^i_\Phi$

$\Gamma^{i+1} \leftarrow \Gamma^i + \Delta^i_\Gamma$

$i = i + 1$.

if stopping criterion is met **then**

stop $\leftarrow$ True

return Θ^i, Φ^i and Γ^i

5.1. Competition Database

With the objective of popularizing BCIs and improving the signal processing and classification pipelines between 2001 and 2008 the Berlin Brain-Computer Interface project team organized four competitions centered around classification for EEG based BCIs. For the third edition, a data set with EEG signals recorded with the BCI2000 system [39] using the P300 speller paradigm was released [40]. This data set consists of EEG data sampled at 240 Hz, recorded from 64 channels, of which only ten channels were used (Fz, C3, Cz, C4, P3, Pz, P4, PO7, PO8 and Oz). Each one of two individuals (A and B) spelled a total of 85 training characters and 100 test characters, with 15 blocks of flashing per character. This procedure yields a total of 15300 training post stimulus signals and 18000 testing post stimulus signals per individual, with a target to

non-target ratio of 1:5. Further details about the data acquisition setup as well as the data set itself are available at the competitions' webpage[1].

5.2. Synthetic Database

To adjust the hyperparameters of the network, we initially used an artificial data set. This was decided in order to avoid complexities such as imbalanced data sets, artifacts, limitations in the number of training samples, extremely noisy and/or mislabeled samples, etc. on the first approach to the problem. This way we were able to reduce the search space for the network configuration and hyperparameter setting for the real data.

To generate this artificial data we fed the linear prediction algorithm with the competitions' data and estimated the parameters of several linear autoregressive (AR) systems to model the background EEG for each channel [41]. The optimal orders of the systems were individually set using the Akaike information criterion. The AR models were fed with white noise and the outputs were used as the simulated background EEG. This process was performed only for ten channels (Fz, C3, Cz, C4, P3, Pz, P4, PO7, PO8 and Oz) and for individual A. Since the white noise signals fed to each channel filter are independent, the background noise signals corresponding to each channel are spatially uncorrelated, as opposed to the signals in the real registers where this correlation is present. Figure 3 depicts the frequency responses of the model filters together with the estimated power spectral density of the original signals (the shown order corresponds to the order of each filter and the y-axis has log. scale).

Each synthetic ERP sample was generated by averaging five hundred randomly chosen post-stimulus target signals. Note that the ERP samples have a longer duration than the response and begin before their corresponding stimulus. In order to avoid introducing border artifacts each sample was windowed with an appropriately chosen gaussian window. Every second, a stimulus with a 0.5 probability of being target, was simulated. This resulted in a balanced data

[1] www.bbci.de/competition accessed June 1, 2016

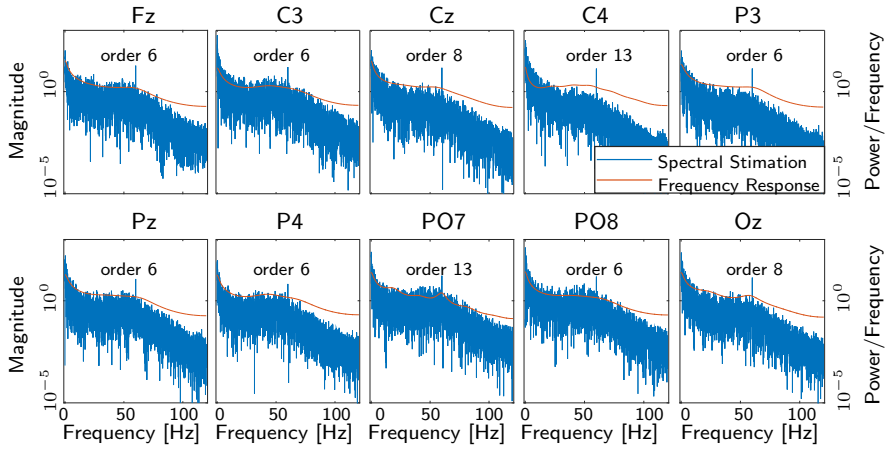


Figure 3: Estimated spectral power of original signals and frequency response of the linear filters found using LPC.

set, as opposed to the real data set where the target to non-target ratio is 0.2. Finally, for each one of the target samples a randomly chosen ERP synthetic sample was added to the simulated background EEG.

Care was taken to maintain the average SNR of the artificial dataset close to that of the original dataset. The SNR was estimated independently for each channel and the average of the ten channels was -19 dB. Since each synthetic sample was randomly generated, this artificial dataset is clearly more realistic and harder to classify than any dataset with a similar SNR, obtained by adding always the same ERP template. The training signals in the simulated dataset were generated using template signals from the training samples in the competitions data, while the testing signals were generated using those from the corresponding testing samples. Figure 4 shows four synthetic ERP samples obtained from the training set for the Pz channel.

5.3. Pre-processing

Both the real and the synthetic data sets were pre-processed in the same way. The pre-processing step started by filtering the data with an order 20 finite impulse response (FIR) filter with band-pass cut-off frequencies at 0.5 and 12 Hz (band-pass filtering is commonly used in EEG based P300 signal

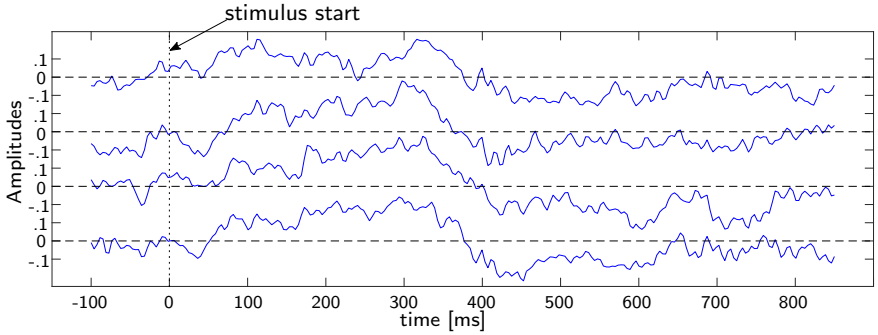


Figure 4: Synthetic ERP samples used to create the training set (Pz channel only).

processing). After filtering the data, single trials (both target and no target) were segmented. Each segment begins with each stimulus and ends 750 ms later. Since the data was sampled at 240 Hz and the highest frequencies present in the P300 ERPs are significantly lower, there is much room for downsampling. In fact for the hyperparameter search and the classification sections we decided to downsample the artificial data to 24 Hz, reducing the number of features. Each post-stimulus segment was then comprised by either 1800 or 180 samples, depending on whether they were downsampled or not. Finally the segments from every channel were concatenated into a single feature vector for each sample.

6. Evaluation of classification performance

To measure the goodness of the feature representations given by the CAEA we tested the classification performance for three different classifiers, namely the SNN, linear SVMs and the n-Nearest-Neighbors classifier (KNNC). The SNN was a natural choice given the structure of the CAEA while the SVM and KNNC were chosen based on popularity, simplicity and the fact that both have few hyperparameters. The choice of the linear SVM allows us to examine whether the fact of having softmax outputs in the CAEA gives the SNN any particular advantage against other classifier with a separation surface of the same complexity (linear). The choice of the KNNC let us observe if there are differences between the usage of linear and non-linear classifiers before and after

the feature extraction by the CAEAs.

The SNNs were trained using the limited memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm implementation included in the optimization package minFunc [42]. All the optimizer parameters were left to default with the maximun number off iterations set to 400.

We used the SVM classifier implementation provided with LIBLINEAR [43] with a MatLab interface. The kernel used was linear as all the configurations available in the mentioned toolbox. All the parameters were left to default and a bias term was added.

The KNNC implementation used is the one provided with PRTools [44]. The euclidean distance was used to define the neighbors. The number of neighbors was set to nine after evaluating the results obtained setting it to three, five, seven and nine. In spite of the simplicity of the algorithms behind KNNC, it can define highly complex separation hypersurfaces.

The performances of the proposed P300 binary classifiers was measured using, the Specificity (Spe), Sensitivity (Sen), Precision (Prec) and Accuracy (Acc), defined as:

$$\text{Spe} = \frac{TN}{FP + TN}, \quad \text{Sen} = \frac{TP}{FN + TP}, \quad \text{Prec} = \frac{TP}{FP + TP}, \quad \text{Acc} = \frac{TP + TN}{NT}. \tag{11}$$

where TP, TN, FP and FN are the number of true positives, true negatives, false positives and false negatives respectively and $NT = FP + TP + FN + TN$ is the total number of samples. Since in the case of the P300-Speller the available data is imbalanced the balanced accuracy (BalAcc), was also computed. This metric is designed to be insensitive to unbalanced data and can facilitate the analysis of the results. It is defined as:

$$\text{BalAcc} = \frac{\text{Spe} + \text{Sens}}{2}. \tag{12}$$

7. Network configuration and hyperparameters setting

It has been shown that the performance of some ANN architectures can vary significantly with the choice of the hyperparameters [45]. As most ANNs, a

CAEA has many hyperparameters and therefore, finding their "optimal" values is often a complex and daunting task. Although some articles address this issue giving practical recommendations and design guidelines [46, 47], each architecture and each dataset has different "optimal" hyperparameters, and appropriate procedures for their estimation are always needed. In this work we propose an ad-hoc five-step approach for setting all hyperparameters. We briefly describe those five steps below. First the main architecture design choices are made, i.e. we choose the AFs and the value of the sparsity target ρ and we decide whether the weights of the encoder and the decoder are tied ($W_\Theta = W_\Phi^T$) or not. Secondly, an objective analysis supported by preliminary experiments leads us to the choice of the number of averages K. Thirdly, we propose the weights adjusting the relative importance of the discrimination and sparsity terms of the cost function (10), i.e. γ and β. Fourthly the regularization hyperparameters of the penalization terms in the cost function, i.e. λ_Θ, λ_Γ and λ_Φ, are chosen via the L-curve procedure [48]. Finally a grid search is performed to find the "optimal" values for the remaining discrete hyperparameters, namely the number of hidden units m, the minibatch size N and L the maximum number of iterations for each run of the optimizer. We now proceed to describe in detail the implementation of each one of the five steps described above and the results obtained.

Many AFs have been proposed for the units of a feed-forward ANN [47]. One of the most known and widely used is the sigmoid, which is a smooth function with bounded derivative in $\mathbb{R}$. This AF provides numerical stability for the training process since its derivative is always between zero and one. For ANN with many layers this type of AF can lead to practical vanishing gradients, and therefore other alternatives such as the ReLU are generally used. Since the vanishing gradients are not a big problem for ANNs with few layers, sigmoid units were used for the experiments described in this article.

The value of the sparsity target ρ was set to 0.1, following guidelines provided in [46]. As mentioned in Section 4, tying the weights of the encoder and the decoder significantly reduces the degrees of freedom of the CAEA, and it

can speed up the training process, at the expense of reducing the expressive capabilities of the model. In this work we adjusted the hyperparameters for the general (untied) network and then we appropriately modified them for the tied case.

The hyperparameter γ associated with the discrimination term of the cost function, was adjusted so that the first and second terms in the RHS of (10) have similar magnitudes. Also the value of the hyperparameter β was set so that the sparsity term (third term in the RHS of (10)) is approximately one order of magnitude smaller than the reconstruction and discrimination terms. This choice is justified by the fact that reconstruction and discrimination are preferred upon sparsity. To estimate the magnitude of each term the cost function was evaluated using the training data.

The number of averages taken for the output can have a big effect on the learned representations. A first approach could be to set the number K of trials to be averaged to a large value to get an output as clean as possible. However, as K increases, the similarity between the input and the coherently averaged desired output diminishes, which leads to a decrease in the CAEA performance since no relation between the output and the input can be established during learning. Experimental data has shown a markedly decrease in performance for values of K larger than 3, reason for which throughout this work we took $K = 2$.

After defining the architecture of the network a fine tuning of the regularization parameters in the cost function can be performed. For this, a three-step process by means of the L-curve method was implemented. First the values of β and γ were fixed as previously described. Then, Θ was randomly initialized to Θ_0. In step 1, in accordance with the L-curve theory [48], λ_Γ was fixed to a value λ_Γ^0, taken approximately equal to that corresponding to the point of maximal curvature of the curve representing $||\Gamma||^2$ as a function of $J_{D}(f(X;\Theta_0), y; \Gamma)$ parameterized by λ_Γ (see Figure 5, step 1). Then we computed:

$$\Gamma_0 \doteq \underset{\Gamma \in \mathbb{R}^{2\times m}}{\operatorname{argmin}} \big(\gamma J_{D}(f(X;\Theta_0), y; \Gamma) + \lambda_\Gamma^0 ||\Gamma||^2\big). \tag{13}$$

In step 2 a similar procedure was used for estimating λ_Φ with λ_Φ^0, a value associated to the point of maximal curvature of the $J_R(f(X;\Theta_0);\Phi)$ - $\|\Phi\|^2$ curve parameterized by λ_Φ (see Figure 5, step 2). Then we computed:

$$\Phi \doteq \underset{\Phi\in\mathbb{R}^{(n+1)\times m}}{\operatorname{argmin}} \left(J_R(f(X;\Theta_0);\Phi) + \lambda_\Phi^0 ||\Phi||^2\right). \tag{14}$$

In step 3 the optimal value λ_Θ^0 of λ_Θ was estimated by the L-curve method as the value corresponding to the point of maximal curvature in the curve representing $\|\Theta\|^2$ as a function of $J_R(f(X;\Theta);\Phi) + \gamma J_D(f(X;\Theta), y; \Gamma_0) + \beta P_S(f(X;\Theta);\rho)$, where Γ_0 and Φ are as in (13) and (14), respectively, computed in steps 1 and 2 above (see Figure 5, step 3). For these three steps the training process was appropriately modified as we describe next. First, instead of splitting the data into minibatches, all the training data was used and the training process was stopped at the first iteration leading to a negligible cost improvement. The cost function was modified as well, since in each step only one of the parameter matrices (i.e. Γ, Φ or Θ) was adjusted. Note that none of the hyperparameters m, N or l was set at this point. Although the values of N and l are not used with this modified training process, it is clearly unavoidable to set the value of m. We performed the three step L-curve process for several values of m (some larger, some equal and some smaller than the size of the input patterns) and found that the optimal values of the penalization hyperparameters (i.e. λ_Γ, λ_Φ and λ_Θ) are essentially independent of m. The L-curve method is extensively used in the inverse problem community but, to the best of our knowledge, no other work extended its use to the context ANNs hyperparameters search.

Finally, a grid search was performed to set the remaining three hyperparameters (i.e. m, N and l). The search for the optimal value of m was performed between 18 and 360 hidden units (corresponding to one tenth and twice the number of input features, respectively). The search for N was performed for minibatches between 10 to 5000 samples as well as using all the training data. The search for l was performed between the values of 2 and 1000. In these experiments the CAEAs were trained using the conjugate gradient method (CG) provided with the minFunc toolbox [42]. The training data was split into train-

sinc(*i*) Research Center for Signals, Systems and Computational Intelligence (fich.unl.edu.ar/sinc)
I. E. Gareis, L. D. Vignolo, R. Spies & H. L. Rufiner; "Coherent averaging estimation autoencoders applied to evoked potentials processing"
Neurocomputing, Vol. 240, pp. 47 - 58, Mar, 2017.

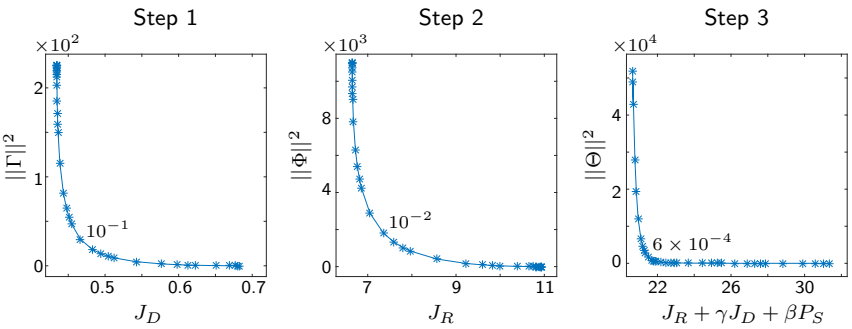


Figure 5: L-curves used to obtain the regularization hyperparameters.

ing and validation subsets, containing 90% and 10% of the data, respectively. These training and validation subsets were subsequently used for all models. In order to reduce uncertainty, all the models with the same value of m where trained with the same initial weights. To assess the performance of the different CAEAs, we used the hidden feature representations that each one produced. Using the transformed versions of the training and validation data, a SNN was trained and evaluated for each CAEA. The goodness of each CAEA was then measured in terms of the classification performance of its corresponding SNN.

The parameters for the "best" (in terms of the performance over the validation data) models are presented in Table 1. Regarding these results, it is important to point out that, although (as it can be seen in Table 1) the classification performance using batches containing all the training data was the best, the improvement with respect to the performances using minibatches was never greater than 2 %. Hence, in some cases were the number of training samples is large, available computing resources and efficiency issues may yield more appropriate the use of minibatches.

It should be noted that the differences between the obtained parameters using real and simulated data for all the experiments were small. This suggests that CAEA's hyperparameters are robust and stable under different data sets. Moreover the strategy of performing a first search using simulated data was valuable. In fact only the hyperparameters obtained with the simulated data were used for the informed results.

Table 1: List of architecture definitions and hyperparameters chosen for the CAEA.

	Simulated Data	Real Data	
		Subj. A	Subj. B
AF for hidden units	Sigmoid	Sigmoid	Sigmoid
Tied weights?	False	False	False
ρ	0.1	0.1	0.1
γ	20	20	20
β	100	100	100
K (# of averages at output)	2	2	2
λ_Θ	6×10^{-4}	10^{-2}	4×10^{-5}
λ_Φ	10^{-2}	2×10^{-2}	10^{-1}
λ_Γ	10^{-1}	4×10^{-2}	2×10^{-1}
m (# hidden units)	90	45	90
N (Minibatch size)	All Data	800	All Data
l (limit of optimizer iterations)	2	250	50

8. Experiments and Results

8.1. P300 classification

Since CAEA are primarily designed to serve as feature representation blocks for posterior classification, we tested their capabilities in this regard. Following the procedure presented in Section 7 the CAEA were trained using the setting given in the second column of Table 1. Note that for the real data set the same experiments were conducted using the optimal hyperparameters found as well (columns 3 and 4, Table 1), however the results obtained were similar in both cases. Then, the CAEAs latent variables were used to train and test SNNs, SVMs and KNNCs.

To generate results for comparison purposes two set-ups were used: one using the raw data and another one using the data coherently averaged twice (each

sample was averaged with a sample belonging to the same class). These signals were also used to train and test SNNs, SVMs and KNNCs. It is reasonable to compare the results obtained using the raw data with those obtained by the CAEA, while the averaged data provides a "ceiling" for the performance since it has a better SNR than the signals fed to the CAEA.

Both data sets are originally separated into the corresponding training and test sets, so hold-out validation was used. All the results presented correspond to the performances over those test sets, which were not used for adjusting hyperparameters nor for model selection. Each training data set was randomly split into two subsets: one (consisting of 90% of the data) to be used for proper training and the other (consisting of the remaining 10% of the data) for validation. The process of splitting the data and training was repeated 100 times for each configuration. The best models for each set-up were then chosen, based upon the performances obtained for the validation sets, and they were subsequently evaluated using the test sets. All available training and testing samples were used in the experiments, i.e. the simulated data set consisted of 10000 training samples and 10000 testing samples and the real data set consisted of 15300 training samples and 18000 testing samples for both subjects (A and B). It should be noted however that, while the simulated data set is balanced, the real data set is not. For the experiments described in this section the downsampled data was used, so the dimension of the input patterns n is 180.

The results over the test set for the best models using the simulated dataset are summarized in Table 2. As it can be seen, the performance obtained using KNNCs is poorer for all configurations. However, it is important to remark that when using the CAEAs latent variables with KNNCs, the performance is improved even in comparison to a KNNCs trained using averages of two trials. Moreover, the latent representations computed through the CAEAs allowed us to improve the classification accuracy, specificity and precision obtained with the SNN and the SVM using single trials. Furthermore, specificity and precision are improved by CAEA+SNN (using single trial) in comparison with the results obtained for SNNs using two trials. These results suggest that the proposed ap-

Table 2: CAEA results for P300/no P300 classification using simulated data. 1T, 2T indicate single-trial and two-trials average respectively using the simulated data.

Method	Spe%	Sen%	Prec%	Acc%
SNN (2T)	72	97	77	84
CAEA + SNN (1T)	**85**	81	**85**	**83**
SNN (1T)	65	**91**	72	78
SVM (2T)	91	89	91	90
CAEA + SVM (1T)	**84**	**84**	**84**	**84**
SVM (1T)	82	81	82	82
KNNC (2T)	79	77	78	78
CAEA + KNNC (1T)	**67**	**95**	**74**	**81**
KNNC (1T)	**67**	65	66	66

proach is able to enhance the relevant information, yielding better classification results.

Although most of the articles featuring classification methods for data recorded using the P300-Speller report the error rate on character classification, we believe that directly measuring the performance of the binary (target/no-target) classification system is easier to analyze in the context of the proposed system (given that our objective is to process and classify ERPs). The results over the testing set for the best models using the real data set are summarized in Table 3. Here we present the results obtained for both subjects as well as the averages using the CAEA softmax outputs and two state of the art methods. The first of the comparison methods is described in [26], where the authors focus on a feature extraction method using Local Discriminant Bases (LDB) using wavelet packet features, to subsequently classify the samples using a linear discriminant analysis classifier. This technique obtains the best results among several other approaches tested by the authors. We replicated the method and tested it using the same data we used for CAEA.

Table 3: Results for P300/no P300 classification using the real data.

Method	Spe%			Sen%			Prec%			BalAcc%		
	A	B	avg.	A	B	avg.	A	B	avg.	A	B	avg.
LDB	65	65	65.0	63	75	69.0	26	30	28.0	64	70	67.0
CNN	69	77	73.0	61	64	62.5	29	35	32.0	65	70	67.5
CAEA	68	73	70.5	62	70	66.0	28	34	31.0	65	71	68.0

The second strategy used for comparison is presented in [12] and, as mentioned in the introduction, consists on a CNN that performs the feature extraction and classification in a single end-to-end system. Although several different networks are proposed in [12], only one (called CNN-2a in the original article) makes use of limited predefined number of the 64 original sensors provided in the database. Since the same approach was taken for this work, we chose precisely that network for comparison. The numbers of TP, TN, FP and FN are informed in [12] and we used those results to compute the other performance estimators. As it can be seen in Table 3 the three methods yield similar performances in terms of the final balanced accuracies, while higher variations can be observed in the specificities and sensitivities.

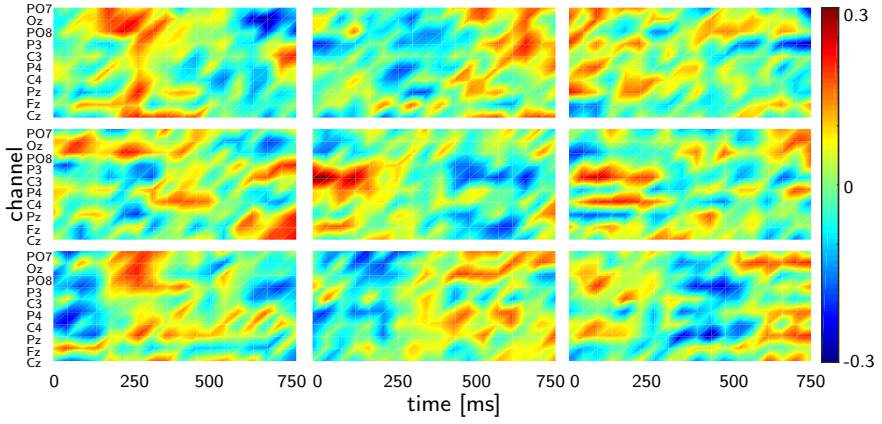


Figure 6: Spatio-temporal representations of the weights of nine hidden units of a trained CAEA.

Figure 6 shows the weights of nine neurons of the encoding layer of a CAEA trained with the real data from subject A. The weights were rearranged from vectors to matrices in order to accommodate different channels in different rows, and different time samples in different columns. The values of the weights are given by the colormap, where green represents zero and blue and red represent extreme negative and positive values, respectively. As expected, the neurons seem to detect characteristic spatio-temporal patterns related with the different classes. For example, some neurons emphasize the temporal area near the P300, while others do the opposite. Some horizontal stripes also indicate the relative importance of the information provided by different channels.

For the models proposed here using the real data set the average training time was around 30 seconds on an Intel Core i7-2600 Quad-Core Processor 3.4 GHz. This training time depends on the parameters of the network and on the initialization of weights. The model was implemented in MatLab without any special hardware optimization. The CAEA was also trained and tested with the hyperparameters given in Table 1 but tying the weights of the encoder and the decoder. With this configuration the classification performance was slightly worse but the training time was reduced.

To evaluate the dependence of the CAEA on the number of training samples we trained the model with reduced subsets of the simulated database. The total number of samples ranged from 20000 (all samples available) to 2700 samples. The performance was degraded by less than 10 %, suggesting that the model would perform relatively well in small sample-size scenarios. This is most likely due to to the regularizations and constraints applied [30].

8.2. Event related potentials estimation

To get a first glance about the network reconstruction capabilities we trained and evaluated the outputs of the CAEA, using the synthetic data set. Since the waveforms are difficult to be visually evaluated if only 18 temporal samples per channel are used, we performed the process using non subsampled data with 180 temporal samples per channel, yielding a total of 1800 features in the input

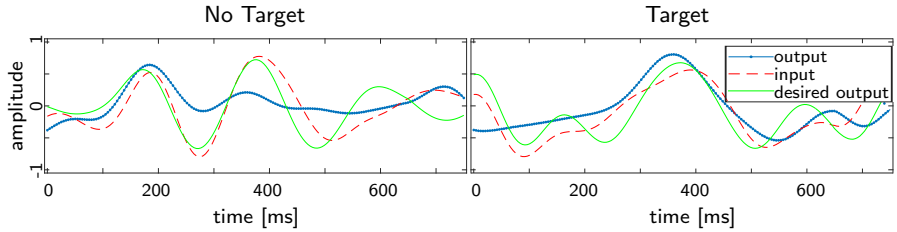


Figure 7: Outputs with their corresponding inputs and desired outputs for a trained CAEA for channel Pz.

and output layers. The CAEA used to obtain the outputs shown in Figure 7 had the same hyperparameters used in Section 8.1 (see Table 1), except for the number of hidden units, which was set to half of the dimension of the input patterns.

Figure 7 shows the features corresponding to channel Pz for the outputs obtained with the CAEA, their corresponding inputs and their desired outputs for a target and a non-target sample. In the case of this CAEA the desired output was computed by averaging three post-stimulus signals. Due to the presence of the P300 wave, target samples commonly have larger amplitudes than their non-target counterparts. Figure 7 shows how this difference, which could be useful for classification purposes, is enhanced by the CAEA.

In Figure 7 we can also observe that the network ignores fast input changes, behaving in this sense like a low pass filter. To further examine this phenomenon, we fed the trained CAEA with Gaussian white noise of an amplitude similar to that of the original signals and computed the power spectral density (PSD) of the outputs. For each channel the averages taken over the PSD of 500 different outputs are depicted in Figure 8. Even though the transformations produced by the CAEA are not linear, and therefore these spectral estimates do not necessarily capture their whole frequency response, they do provide evidence of the ANN behaving as low pass filters. The energy of the PSD of the outputs below 10 Hz is negligible for all channels (the cut-off frequencies are about 9 ± 1 Hz), which is reasonable since the desired outputs of the network were coherent averages of signals filtered with a cutoff frequency of 12 Hz, and the averaging

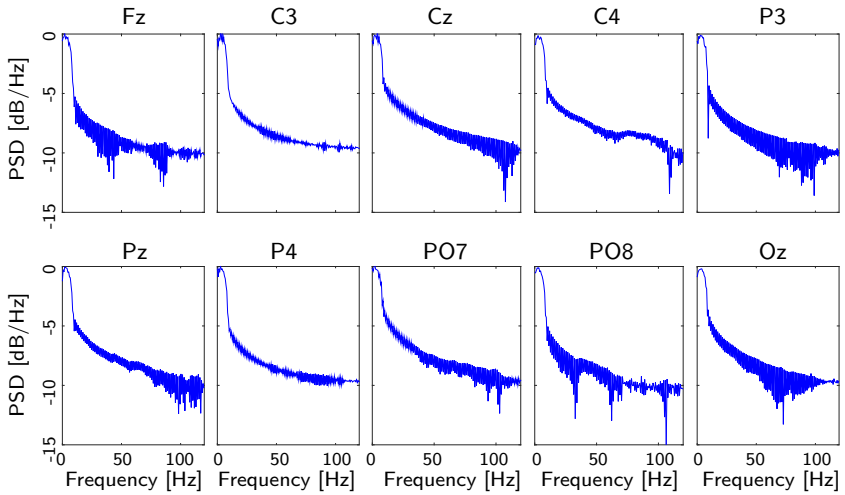


Figure 8: Single sided Power Spectral Density estimation of the CAEA outputs for each channel when fed with white noise.

process attenuates higher frequencies.

9. Discussion and Conclusions

Inspired on classical AEs, a novel network architecture was proposed for feature extraction of signals where small patterns are hidden behind large uncorrelated noise. The network was tested using artificially generated data and real EEG data. The obtained results are promising since classification performance was improved using the hidden features extracted with the CAEAs for different types of classifiers.

The CAEA was tested using a binary classification problem of our interest, however it can be used for multiclass problems as well. It should be noted that off-the-shelf methods were used to train the networks and the results could probably be improved using appropriate ad-hoc optimizers.

Regarding the computational cost of the algorithm, many operations are required during training since the gradients of the full network have to be computed. The CPU time required for training can vary significantly depending

on the number of hidden units and the optimization method used to adjust the weights. If the hyperparameters have to be adjusted, several networks may need to be trained, thus increasing the computational cost of the process. However, application of the resulting encoding function in real time is computationally inexpensive. In fact, it only entails a matrix product and the evaluation of a simple vector function.

The idea of improving the SNR of the output, as opposed to the idea behind DAE of corrupting the input, could be applied in other ways. An ANN analogous to the CAEA could be used in any problem where there is a technique to find a better version of the original pattern that can not be employed in operation time.

Although the number of hyperparameters endow the algorithm with much flexibility, it can yield the network difficult to train, thus preventing inexperienced users from obtaining good results. For that reason a systematic systematic method was used to adjust the network architecture and set the hyperparameters and some guidelines to adjust them to new data were provided. In this regard, to facilitate the use of the CAEA, a reasonably good predefined value should be set for each hyperparameter to ensure good results without the need of time or expertise to adjust them.

As future work we propose to use the CAEA as blocks for progressive (layer-by-layer) feature extraction, similar to the algorithm used to train stacked autoencoders. However its use could not be as direct, since CA over the latent representations would not necessarily improve the SNR due to the non-linear nature of the encoding transformation. Tests on real data would also be carried out, to estimate the performance of the architecture in the character recognition task.

References

[1] G. D. Dawson, A summation technique for detecting small signals in a large irregular background, The Journal of Physiology 115 (1) (1951) 2.

[2] O. Rompelman, H. H. Ros, Coherent averaging technique: A tutorial review Part 1: Noise reduction and the equivalent filter, Journal of Biomedical Engineering 8 (1) (1986) 24–29.

[3] M. Molodova, Z. Li, A. Nunez, R. Dollevoet, Automatic Detection of Squats in Railway Infrastructure, IEEE Transactions on Intelligent Transportation Systems 15 (5) (2014) 1980–1990.

[4] C. T. Allen, S. N. Mozaffar, T. L. Akins, Suppressing coherent noise in radar applications with long dwell times, Geoscience and Remote Sensing Letters, IEEE 2 (3) (2005) 284–286.

[5] J. R. Jarrett, N. C. Flowers, Signal-averaged electrocardiography: history, techniques, and clinical applications, Clinical Cardiology 14 (12) (1991) 984–994.

[6] F. Andreotti, M. Riedl, T. Himmelsbach, D. Wedekind, N. Wessel, H. Stepan, C. Schmieder, A. Jank, H. Malberg, S. Zaunseder, Robust fetal ECG extraction and detection from abdominal leads, Physiological Measurement 35 (8) (2014) 1551.

[7] G. D. Dawson, A summation technique for the detection of small evoked potentials, Electroencephalography and Clinical Neurophysiology 6 (1954) 65–84.

[8] J. N. Mak, J. R. Wolpaw, Clinical Applications of Brain-Computer Interfaces: Current State and Future Prospects, IEEE Reviews in Biomedical Engineering 2 (2009) 187–199.

[9] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536.

[10] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (5) (1989) 359–366.

[11] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117.

[12] H. Cecotti, A. Graser, Convolutional Neural Networks for P300 Detection with Application to Brain-Computer Interfaces, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (3) (2011) 433–445.

[13] X. An, D. Kuang, X. Guo, Y. Zhao, L. He, A Deep Learning Method for Classification of EEG Data Based on Motor Imagery, in: D.-S. Huang, K. Han, M. Gromiha (Eds.), Intelligent Computing in Bioinformatics, no. 8590 in Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 203–210.

[14] J. Li, Z. Struzik, L. Zhang, A. Cichocki, Feature Learning from Incomplete EEG with Denoising Autoencoder, Neurocomputing 165 (C) (2015) 23–31.

[15] J. R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, T. M. Vaughan, Brain-computer interfaces for communication and control, Clinical Neurophysiology: Official Journal of the International Federation of Clinical Neurophysiology 113 (6) (2002) 767–791.

[16] J. R. Wolpaw, BCI meeting 2005-workshop on signals and recording methods, IEEE Transactions on Neural Systems and Rehabilitation Engineering 14 (2) (2006) 138–141.

[17] A. Bashashati, M. Fatourechi, R. K. Ward, G. E. Birch, A survey of signal processing algorithms in brain-computer interfaces based on electrical brain signals, Journal of Neural Engineering 4 (2) (2007) R32–57.

[18] F. Lotte, M. Congedo, A. Lcuyer, F. Lamarche, B. Arnaldi, A review of classification algorithms for EEG-based brain-computer interfaces, Journal of Neural Engineering 4 (2) (2007) R1–R13.

[19] J. N. Mak, Y. Arbel, J. W. Minett, L. M. McCane, B. Yuksel, D. Ryan, D. Thompson, L. Bianchi, D. Erdogmus, Optimizing the P300-based brain-

computer interface: current status, limitations and future directions, Journal of Neural Engineering 8 (2) (2011) 025003.

[20] Z. Qiu, J. Jin, H.-K. Lam, Y. Zhang, X. Wang, A. Cichocki, Improved {SFFS} method for channel selection in motor imagery based {BCI}, Neurocomputing 207 (2016) 519 – 527.

[21] K. Colwell, D. Ryan, C. Throckmorton, E. Sellers, L. Collins, Channel selection methods for the p300 speller, Journal of neuroscience methods 232 (2014) 6–15.

[22] W. Wu, Z. Chen, X. Gao, Y. Li, E. N. Brown, S. Gao, Probabilistic common spatial patterns for multichannel EEG analysis, IEEE transactions on pattern analysis and machine intelligence 37 (3) (2015) 639–653.

[23] Y. Zhang, G. Zhou, J. Jin, X. Wang, A. Cichocki, Frequency recognition in SSVEP-based BCI using multiset canonical correlation analysis, International journal of neural systems 24 (04) (2014) 1450013.

[24] H. Wang, Y. Zhang, N. R. Waytowich, D. J. Krusienski, G. Zhou, J. Jin, X. Wang, A. Cichocki, Discriminative feature extraction via multivariate linear regression for SSVEP-based BCI, IEEE Transactions on Neural Systems and Rehabilitation Engineering 24 (5) (2016) 532–541.

[25] J. Kevric, A. Subasi, Comparison of signal decomposition methods in classification of EEG signals for motor-imagery BCI system, Biomedical Signal Processing and Control 31 (2017) 398 – 406.

[26] V. Peterson, R. C. Acevedo, H. L. Rufiner, R. Spies, Local Discriminant Wavelet Packet Basis for Signal Classification in Brain Computer Interface, in: Anales del VI Congreso Latinoamericano de Ingeniería Biomédica (CLAIB 2014), 2014, p. 279.

[27] W. Wu, S. Nagarajan, Z. Chen, Bayesian machine learning: EEG/MEG signal processing measurements, IEEE Signal Processing Magazine 33 (1) (2016) 14–36.

[28] W. Wu, C. Wu, S. Gao, B. Liu, Y. Li, X. Gao, Bayesian estimation of ERP components from multicondition and multichannel EEG, NeuroImage 88 (2014) 319–339.

[29] Y. Zhang, Y. Wang, J. Jin, X. Wang, Sparse bayesian learning for obtaining sparsity of eeg frequency bands based feature vectors in motor imagery classification, International Journal of Neural Systems (2016) 1650032.

[30] Y. Zhang, G. Zhou, J. Jin, Q. Zhao, X. Wang, A. Cichocki, Sparse bayesian classification of EEG for brain-computer interface, IEEE Transactions on Neural Networks and Learning Systems 27 (11) (2016) 2256–2267.

[31] H. Bashashati, R. K. Ward, A. Bashashati, Bayesian optimization of bci parameters, in: Electrical and Computer Engineering (CCECE), 2016 IEEE Canadian Conference on, IEEE, 2016, pp. 1–5.

[32] L. He, D. Hu, M. Wan, Y. Wen, K. M. von Deneen, M. Zhou, Common bayesian network for classification of EEG-based multiclass motor imagery BCI, IEEE Transactions on Systems, Man, and Cybernetics: Systems 46 (6) (2016) 843–854.

[33] L. A. Farwell, E. Donchin, Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials, Electroencephalography and Clinical Neurophysiology 70 (6) (1988) 510–523.

[34] R. Fazel-Rezai, W. Ahmad, P300-based brain-computer interface paradigm design, in: J. Fagerberg, D. C. Mowery, R. R. Nelson (Eds.), Recent Advances in Brain-Computer Interface Systems, InTech, 2011, Ch. 4, pp. 266–290.

[35] J. Jin, E. W. Sellers, S. Zhou, Y. Zhang, X. Wang, A. Cichocki, A P300 brain-computer interface based on a modification of the mismatch negativity paradigm, International journal of neural systems 25 (03) (2015) 1550011.

[36] D. Ming, X. An, Y. Xi, Y. Hu, B. Wan, H. Qi, L. Cheng, Z. Xue, Time-locked and phase-locked features of P300 event-related potentials (ERPs) for brain–computer interface speller, Biomedical Signal Processing and Control 5 (4) (2010) 243–251.

[37] J. S. Bridle, Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition, in: Neurocomputing, Springer, 1990, pp. 227–236.

[38] P. Vincent, H. Larochelle, Y. Bengio, P. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th international conference on Machine Learning, ACM, 2008, pp. 1096–1103.

[39] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, J. R. Wolpaw, BCI2000: a general-purpose brain-computer interface (BCI) system, IEEE Transactions on Biomedical Engineering 51 (6) (2004) 1034–1043.

[40] B. Blankertz, K.-R. Müller, D. J. Krusienski, G. Schalk, J. R. Wolpaw, A. Schlögl, G. Pfurtscheller, J. d. R. Millán, M. Schröder, N. Birbaumer, The BCI competition. III: Validating alternative approaches to actual BCI problems, IEEE Transactions on Neural Systems and Rehabilitation Engineering 14 (2) (2006) 153–159.

[41] J. Makhoul, Linear prediction: A tutorial review, Proceedings of the IEEE 63 (4) (1975) 561–580.

[42] M. Schmidt, minfunc, `http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html` (2005).

[43] R. E. Fan, K. W. Chang, C. J. Hsieh, X. R. Wang, C. J. Lin, LIBLINEAR: A library for large linear classification, The Journal of Machine Learning Research 9 (2008) 1871–1874.

[44] F. Van Der Heijden, R. Duin, D. De Ridder, D. M. Tax, Classification, parameter estimation and state estimation: an engineering approach using MATLAB, John Wiley & Sons, 2005.

[45] N. Pinto, D. Doukhan, J. J. DiCarlo, D. D. Cox, A high-throughput screening approach to discovering good forms of biologically inspired visual representation, PLoS Comput Biol 5 (11) (2009) e1000579.

[46] G. E. Hinton, A practical guide to training restricted Boltzmann machines, tech. Rep. UTML TR 2010-003 (2010).

[47] Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 437–478.

[48] H. W. Engl, M. Hanke, A. Neubauer, Regularization of inverse problems, Vol. 375, Springer Science & Business Media, 1996.