Inverse Problem's Solution Using Deep Learning: An EEG-based Study of Brain Activity. Part 1 - rel. 1.0

Roman Tankelevich
California State University, Long Beach

Inverse Problem's Solution Using Deep Learning: An EEG-based Study of Brain Activity. Part 1

Roman Tankelevich

Computer Engineering and Computer Science Department

California State University Long Beach, USA

Roman.Tankelevich@csulb.edu

***Abstract*—A method of solving large scale 3D inverse problems by using the Deep Learning technique is presented and studied. The method is applied to the cases when the empirical data is absent while some forward-feeding models may exist. These models are used to create an appropriate dataset for training the ANN. Both continuous and discrete (binary value) domains are considered in the ill-posed problem's solution. The proposed technique is based on regularization using the Hamming norm which leads to a solution presented as the set of clusters. The method is applied to Low Resolution Brain Tomography based on the EEG signals. Examples of the TensorFlow implementations are given.**

Keywords—ill-posed inverse problems, deep learning, Artificial Neural Network, Low Resolution Brain Tomography, EEG

I. Introduction

The most common formulation of an inverse problem includes three components: a cause, a model (transfer function) and an effect. Two main possible cases of inverse problems, are, usually, about finding: (1) the cause given both a model and an effect; and (2) the model when given pairs of (cause-effect). Any inverse problem is, most likely, to be *ill-posed* which is to say that it is prone to numerical noise, inaccuracy and instability. (A simple example would be irreversibility of time dependent dynamic processes – reversal of time makes such a process unstable).

Case (1) is about finding an inverse operator of the model and then applying this operator to the effect. In linear models, it is about finding the inverse matrices. The complexity of the task is aggravated by the fact that the number of causes (inputs) can be significantly different (most likely, greater) than the number of effects (model's outputs).

Case (2) requires obtaining many samples of cause-effect pairs to recover parameters of the model's operator – this is known as a regression problem.

In both cases, the *training* procedures can be applied to find either the parameters of the inverse or forward operators. The training techniques using, for example, a supervised learning with Artificial Neural Networks are constructed as mapping of the causes (ANN input) onto the effects (ANN output), or mapping of the effects (ANN input) onto the causes (ANN output).

In this paper, we consider the problem of the EEG mapping onto areas of the brain. The task known as the *EEG based tomography* is about finding the clusters of the brain (more specifically, in the cortex) corresponding to the EEG real time measurements. There are some known approaches to its solutions including the LORETA methods of solving assumed linear models representing the electrode potentials effected by neuron activation in specific cognitive situations [1].

One of the tasks studied in the context of the EEG tomography is the emotions' recognition: objective identification of human reactions to different situations that induce emotions. The collection of real-time EEG signals is expected to enhance the effectiveness of data collection in education, entertainment, commerce, and in other forms of human activities.

Here is the problem considered in this project: *Using real-time recorded EEG, map the electrode potentials onto the clusters of activated cortex neurons.*

Based on numerous studies mostly using the fMRI [1], the activated clusters can be associated with certain emotions and even meanings of thoughts (see Fig. 1 as an example). This can be done with sufficient spatial resolution in identifying the activated clusters but can't be used in real time applications. The EEG cluster mapping, on the contrary, is meant to provide the real time interpretation of the human brain activities including the emotional response.

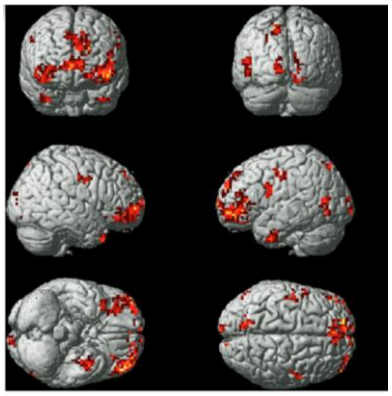


Fig. 1. Images of clusters representing six emotion words [1]

Our goal is to use the *Deep Learning* methods and tools to solve the inverse problem in the EEG-Cortex context. We analyze the representation of the neurons and their clusters activation methods at the Binary vs. Real-value activation level. MFS (Radial functions) approach and real-value activation (Moore-Penrose formulation, example of solution as compared to LORETA) is presented.

The inverse problems are, most commonly, considered in the context of the Moore-Penrose method and regularization technique. In the brain study, it's been implemented as the LORETA techniques [2].

In this paper we consider an alternative approach based on Deep Learning Neural Networks.

We have shown that the binary representation is more efficient than the continuous one. The formulation of Inverse Problem for binary neuron activation and regularization using the Hamming distance (*min H* criterion) is given. It leads to the assumption that the most compact cluster based solution is the one that most appropriate among other possible ones (minimization of H is somewhat equivalent to the tighter clustering of neurons).

The paper's material is organized as follows:

We start with considering the LORETA model which is known as one of the efficient methods of the EEG mapping. The *Low Resolution Tomography* deals with the discrete sources of the continuous values in the 3D cortex area. The purpose of the method is to find a minimum norm solution.

The proposed approach to the solution of the large scale inverse problem is given, next. It implements the forward/backward modeling using the method of machine learning. In the core of the method, the cortex activity is considered as a potential field model using Method of Fundamental Solutions (Radial functions).

We, then, define the parameters of the model emphasizing the significance of the binary representation of the sources, the Hamming norm, and clusterization. The Monte-Carlo method and its variants are used in populating the given domain of the cortex.

The discussion of the binary (integer) programming follows. The proposed model with the binary value discrete sources in the domain can't be solved as the inverse problem with such traditional methods as the Moore-Penrose. The proposed here forward/backward learning method is suggested to provide the solution.

Model's implementation consists of two parts: forward and backward (Deep Learning) models.

Also, we provide some specifics concerning Big data collection and its utilization.

Then we define our model and its parameters including the following aspects:

- Randomized representation of sources
- Forward continuous and integer models
- Clusterization of the brain model
- Integer programming and Hamming distance regularization
- Inverse solution and clustering
- Deep Learning algorithm
- Experiments and results

II. Minimum Norm, Dale and LORETA models

Hämäläinen, M.S., and Ilmoniemi [2] were the first to publish a particular solution to the instantaneous, distributed, discrete, linear EEG inverse problem. Their solution is known as the minimum norm inverse solution. [2]

Dale, et al. [3] proposed a method in which localization inference is based on a standardization of the current density estimates. In particular, they employed the current density estimate given by the minimum norm solution, and they standardized it by using its expected standard deviation, which is hypothesized to be originated exclusively by measurement noise.

The sLORETA method [4] expands the above approaches using the regularization technique. The core model is:

$$\mathbf{F} = \mathbf{K}\,\mathbf{J}.$$

Here $\mathbf{F} \in \mathbb{R}^{N_E \times 1}$ is a vector containing scalp electric potentials measured at N_E cephalic electrodes. The primary current density $\mathbf{J} \in \mathbb{R}^{N_V \times 1}$ where $J_l = \mathbb{R}^{3\times 1}$ represents 3 unknown dipole moments in a voxel l ($l = 1, \dots, N_V$).

The lead field $\mathbf{K} \in \mathbb{R}^{N_E \times (3N_V)}$ is a linear operator that maps strengths of dipoles in voxels onto the electrode potentials.

The LORETA method employs the current density estimate given by the minimum norm solution. The functional to minimize is presented as follows:

$$F = ||\mathbf{F} - \mathbf{K}\,\mathbf{J}||^2 + a||\mathbf{J}||^2$$

with minimum $\hat{\mathbf{J}} = \mathbf{TF}$ where $\mathbf{T} = \mathbf{K}^T[\mathbf{KK}^T + a\mathbf{I}]^+$; here $[]^+$ is the Moore-Penrose pseudoinverse.

The standardized (sLORETA) method takes into account two sources of variation: mainly the variation of the actual sources, and then finally, if any, the variation due to noisy measurements. As reported in [4] this results in better localization of the sources.

III. Forward/backward model

Even with regularization, the inverse problem's solution remains sensitive to localization of sources – it is possible to improve accuracy of finding the strength of sources but the error of spatial identification of significant clusters most likely remains high.

In this paper, we utilize forward – and therefore non-sensitive – modeling before approaching the inverse problem.

We start with generating the sources of activation (*neurons*) at randomly selected locations in the given domain (cortex). Assuming a model of electrical field generated by the sources, the electrical potentials measured by cephalic electrodes are calculated. Thus, the dataset consisting of causes and effects is constructed. Then, this dataset is used for training the Deep Learning Model which takes the electrode potentials as inputs and the *locations* of activated clusters of neurons as outputs (*classes*). The trained model is, then, used for predicting the activated locations (clusters) based on the observed electrode potentials.

Obviously, the inverse character – and, therefore, instability and sensitivity of the problem – remains unchanged but, this time, the task is formulated in *locations of clusters* rather than in the strengths of the sources.

TABLE 1. Elements of the proposed method

1.	Create a neuron activation model and calculate the electrode potentials, `ElPot`, for a given set of fired (activated) neurons, `NeuronSet` – *forward model*.
2.	Run the forward model for random sets of activated neurons (and/or their clusters) – collect the dataset by *mapping activated neurons/clusters onto electrode potentials*: `NeuronSet → ElPot`.
3.	Prediction: Given an observation of electrode potentials, `ElPot_Obs`, find the possible corresponding clusters of the activated cortex neurons, `NeuronSet_Obs`; use the dataset from *step* 2 to search through – *inverse solution*.
4.	Implement the inverse model by using the Deep Neural Network (DNN): training the model with the `ElPot_Obs` as the input and `NeuronSet` as the output classes.
5.	Provide validation and testing of the DNN as above.
6.	Estimate the hypothesis $\mathscr{H}$: *the DNN in Step 4 generalizes beyond the dataset used for the training.*

IV. Model of Cortex Activity

Consider the following model of cortex activity.

1. Given is a set, *S*, of *N* sources of activation inside the cortex area *G*(*x*, *y*, *z*). The sources represent the neurons locations:
$$S = \{(sx_i, sy_i, sz_i): (sx_i, sy_i, sz_i) \in G;\ i \in [1:N]\} \quad (1)$$

2. A set, *E*, of *M* < *N* locations of sensors (electrodes) applied to the boundary surface ∂G:
$$E = \{(ex_j, ey_j, ez_j): (ex_j, ey_j, ez_j) \in \partial G;\ j \in (1:M)\} \quad (2)$$

3. The brain's electrical field *u*(*x*,*y*,*z*) is modeled as elliptical (Laplace) equation:
$$Lu = 0, \qquad (x, y, z) \in G,$$
$$u = g(x, y, z); \qquad (x, y, z) \in \partial G \quad (3)$$
where $g(x, y, z)$ is the boundary conditions – the electro-encephalographic potentials (EEG) on the surface of the scalp.

4. The method of fundamental solutions is used to approximate the solution, *u**(*x*,*y*,*z*), as the linear combination of the basis functions of the elliptical equation (functions of the distance between the source points, (sx_i, sy_i, sz_i), and the points, (x_j, y_j, z_j), on the boundary, where the electrodes are located):
$$u^*(x_j, y_j, z_j) = \sum_{i=1}^{N} \alpha_i \varphi(r_{i,j}) \quad (4)$$
$$r_{i,j} = \left\| (x_j, y_j, z_j) - (sx_i, sy_i, sz_i) \right\|^2 \quad (5)$$
In the 3D case the basis function is defined as a radial function:
$$\varphi(r_{i,j}) = 1/r_{i,j} \quad (6)$$

5. By introducing the vectors representing the geometric locations of sources, **S**, and electrodes, **E**, the strengths (potentials) of the sources, $\mathbf{P}_S$, and the collected electrode potentials, $\mathbf{P}_E$, the following equation is obtained:
$$\mathbf{P}_E = \widetilde{\mathbf{D}}\mathbf{P}_S \quad (7)$$
where $\widetilde{\mathbf{D}}$ is the matrix of reciprocals of the elements of the distance matrix **D**: $d(i, j) = r_{i,j}$.

6. The model (7) is used to find the sources $\mathbf{P}_S$ as the solution of the inverse problem. The solution is based on the training of the corresponding DNN (Table 1, #4). The following steps take place:
   (a) Definition of the model's parameters;
   (b) Forward feeding;
   (c) Inverse solution.

V. Definition of the Model's Parameters

To determine the parameters of the 3D model, the following steps should be performed:

(i) Define the positions of the electrodes as a vector **E**.
(ii) Create a random set of the source locations as a vector **S**.
(iii) Assign strengths as electrical potentials (vector $\mathbf{P}_S$) to the sources.

Two approaches to definition of the strength of the sources were considered: (1) each source is assigned a random *continuous* value, and (2) the sources are simulated as the firing neurons, thus the *binary* values are given to the randomly selected neurons.

One way to diminish the size of the model is to consider the *clusters* of the closely coupled neurons and using their centroids as the sources **S.** The strength of the sources in the centroids can be calculated as the sum of its neuron's activations via equation (7).

The task here is to identify, as the solution of the inverse problems, ***the clusters*** of the activated neurons rather than the neurons themselves.

In this research, we studied mostly the *binary* activation model. Our assumption is that such an approach can come closer to more realistic method of solution. With very large number of neurons, the assumptions can be made that the activation of a neuron propagates to its closest neighboring neurons just forming a cluster of activation.

Here we are looking for a solution of the inverse problem on a given set of large number of sources. The problem of finding a corresponding binary string is underdetermined and requires regularization.

The *Hamming distance* is suggested here as the regularization norm. So, the task is to find the binary string representing the status of neurons (1/0 for activated/

nonactivated neurons) that minimizes the Hamming norm (the number of non-zero elements in the activation string) subject to the constraints given by the intervals of the values of electrode potentials.

It is shown that the solution based on this regularization norm becomes clusterized.

VI. Creating a random set of the source locations

The Monte-Carlo method is used to set up the positions of the neurons.

A prohibitively high number of calculations is needed to scan the parametric space of sources deterministically. Using Monte-Carlo simulation provides an opportunity of sampling the intervals with higher density because the same random sample can contribute to more than one parameter's sampling.

In this work we used and analyzed different Monte-Carlo based randomization methods for generation of the sources' locations: spatial uniform randomization, Latin Hypercube (LHC), orthogonal sampling, the Halton method and others.

The most effective set of randomized location of sources should allow to avoid the situation when random samples become not evenly distributed and when the densely grouped sources are formed in less important parts of the parametric space.

An example of the LHC method is illustrated with the following distribution (Fig. 2):

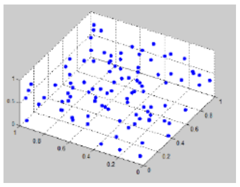


Fig. 2. An example of source distribution generated by using MATLAB's `lhsdesign(n,p)` function

From this illustration, it can be seen that

- The samples are taken randomly within the selected intervals in each dimension.
- The density of the samples does not depend on subintervals.
- The samples for the sources seem to be uniformly distributed over the given multidimensional space.

It is evident, also, that there are no "collapse" points.

The space-filling Latin hypercube was designed specifically for computer experiments [5]. Fig. 3 shows an example of a random set of the source locations in the cortex model.

VII. Binary value sources and Clusterization

A. Binary value model

Consider the sources (neurons) which take only discrete (binary) values: 0/1 – not active / fired.

The neurons are mapped onto the N cortex locations. The solution is to be found as a binary N-vector, X.

As we discuss below, the advantage of this approach is that the activated neurons are expected to be found as a set of tightly coupled clusters.

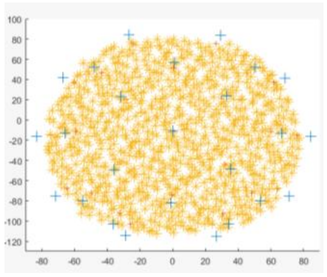


Fig. 3. A random set of sources S generated for cortex model (top view). Blue 'plus' signs: electrode locations; yellow 'star' marks: N neurons (N=10 000) generated by LHC (the brain image is based on Matlab MRI.m model; the positions of electrodes are taken from [4]).

The approach here is based on two stages as follows:

(1) Modeling stage: Produce different combinations of activated neurons, calculate and record the electrode potentials;

(2) Inverse solution: Given a set of observed potentials of the electrodes (EEG potentials) find the closest values of the modeled potentials and the corresponding strengths (1/0) of the neurons.

The task is underdetermined thus having the multiple solutions. Here, we are using a regularization technique allowing to obtain a single, most plausible solution. The Hamming metric is used as the regularization norm H(X) which is equivalent to minimizing the number of activated neurons. It can be interpreted as the task of finding the smallest, tightest clusters.

B. Definitions

1) A source set **S** consists of N random locations, as 3D points, uniformly distributed in the volume V of the 3D domain G (here, the cortex area).

2) Activation vector

A binary vector $\mathbf{X} = \{x_i = [0, 1],\ i \in [1..N]\}$ is defined on the set of sources **S.** The vector **X** is a binary form of the neurons' activation strength – other formulations, such as dipole (see [4], for example), or tensors are possible, as well.

3) Cluster

The volumetric statistical density σ of N points in volume V is defined as N / V and the mean distance between the closest points p_1 and p_2 is given as

$$\hat{d} = d(p_1, p_2) \sim \sigma^{-1/3}.$$

A subset of **S**: $\mathbf{C}_i \subset \mathbf{S}$ is defined as a *cluster* if it is a set of all points used as sources such that for any single source $s^o \in \mathbf{C}_i$ there is at least one other source $s' \in \mathbf{C}_i$ such that the expectation for the cluster distances

$$\mathrm{E}\big(d(s^o, s')\big) \leq \hat{d}. \qquad (8)$$

An *active* cluster $\mathbf{C}_i$ has all its neurons in the fired state such that the strength of the cluster's sources is a unit vector:

$\mathbf{X}\,(\mathbf{C}_i) = \mathbf{1}.$

The cluster $\mathbf{C}_i$ generates the electrode potentials using the radial function operator, $\widetilde{\mathbf{D}}$, in (7).

4) An active set of clusters

This is a set $\mathbf{C}$ of m active clusters $\mathbf{C}_i \subset \mathbf{C}$ (i =1,.., m). Any active cluster can be separated of others or tightly coupled with some others.

C. *The problem – formulation 1* : *Set of sources*

Given a vector of observable potentials $\mathbf{P}_E$, find an activation vector $\mathbf{X}(\mathbf{S})$ defined on the complete set $\mathbf{S}$ of sources such that it

minimizes H($\mathbf{X}$) subject to the following constraints

$\mathbf{P}_E = \widetilde{\mathbf{D}}\mathbf{X}$,

$\boldsymbol{\varepsilon}_{\min} < \mathbf{L}2(\mathbf{P}_E) < \boldsymbol{\varepsilon}_{\max}$

Here

H($\mathbf{X}$): the Hamming norm of $\mathbf{X}$ used as a regularization norm;

$\mathbf{P}_E$: the given observations of electrode potentials;

$\boldsymbol{\varepsilon}_{\min}$, $\boldsymbol{\varepsilon}_{\max}$: the given lower and upper bounds of electrode potentials;

$\mathbf{D}$: the distance matrix used in calculation of electrode potentials as in (7);

$\mathbf{L}$2: Euclidean norm.

D. *The problem – formulation 2* : *Set of clusters*

Find a set $\mathbf{C}$ of m active clusters $\mathbf{C}_i \subset \mathbf{C}$ (i =1,.., m) such that $\mathbf{P}_E$ ($\mathbf{C}$) is within the constraints given with an open interval $\boldsymbol{\varepsilon}$ = ($\boldsymbol{\varepsilon}_{\min}$, $\boldsymbol{\varepsilon}_{\max}$):

$\boldsymbol{\varepsilon}_{\min} < \mathbf{L}2(\mathbf{P}_E(\mathbf{C})) = \mathrm{L}2(\widetilde{\mathbf{D}}\,\mathbf{X}(\mathbf{C})) < \boldsymbol{\varepsilon}_{\max}$

E. *Interpretation*

From formulation 1, it follows that the solution sets should be of minimal number of active sources (*min* value of the Hamming norm). This means that the active sources should be compactly located forming clusters.

It can be proven by induction. Consider a process of construction of solutions as a set of source locations covering the full range of the observed electrode potentials. Assume a step δ used to cover the full range of observations.

For the simplicity of the analysis, consider that only one electrode potential, p, is needed to be observed. Each solution corresponding to a given range δ can be found by using n steps for active sources to be identified.

At (n = 0, p = 0) a source's location s_0 can be found to get the observation $p_0 \in [0,\delta_0]$ at x_0=s_0. Here $\delta_0 = 1/r_0$ where r_0 is the distance from s_0 to the electrode. The next closest observed potential is due to an addition of a new source, s_1, at the minimal distance to s_0 such that the distance expectation $\mathrm{E}\big(d(s_0, s_1)\big) \le \hat{d}$ is due to the definition of the cluster (8). Thus, the 1-cluster $\mathbf{C}_1$ is formed.

The increment of the observed potentials at the electrode for the 1-cluster is given by the

$$\delta_1 = \frac{1}{r_1}$$

where r_1 is the distance between the source s_1 and the electrode. The new source is selected to produce a maximum increase of the electrode potential, thus $r_1 \le r_0$

Consider an inductive step at $n = m$.

Assume that a (m – 1)-cluster, $\mathbf{C}_{m-1}$, was formed.

Identify a source s_k $(0 \le k \le m-1)$ at the closest location, r_k, to the electrode. By adding a new source, s_m, at the closest location to s_k such that $r_m \le r_k$, a new $\mathbf{C}_m$ cluster is obtained.

An increment to the observable bounds of the electrode potentials, thus, is found as:

$$\delta_m = \frac{1}{r_m}.$$

Repeating the process of adding the closest sources to the previous cluster iteration, the sequences of N increments of the observable potentials corresponding to the constructed clusters can be obtained to define the bounding constraints:

$\boldsymbol{\varepsilon}_{\min} < \delta_i < \boldsymbol{\varepsilon}_{\max}$ $(0 < i < N)$.

Thus, the binary clusters can be constructed to produce the observable continuous potentials within the corresponding *bins* given by constraints.

Lemma 1. For any given bin of potentials defined by a cluster $\mathbf{C}_m$, the Hamming norm H($\mathbf{C}_m$) is minimal among non-cluster solutions.

Consider the cluster source s_k at the closest location to the electrode, r_k . It adds, on average, $\frac{1}{r_k+\hat{d}}$ to the electrode potential.

To replace this source with a non-cluster source distanced from the electrode by $R > r_k + \hat{d}$, more than one additional source will be required. Thus, non-clusterized sets of sources have the larger number of elements as compared to the cluster solution.

Lemma 2. Clusters producing larger number of bins at the range of electrode potentials have a smaller Hamming norm.

Larger clusters allow to obtain higher sensitivity in filling up the potential domain since they have more sources at remote positions.

Thus, there should be such clusters that cover the range of observable potentials faster. Those clusters have smaller number of sources.

To cover the given range of electrode potentials, they are located closer to the electrode location. Clearly, there should be such clusters that minimize the Hamming norm.

Theorem. Given a discrete set of binary sources N and a finite number of electrode locations M, the solution of the inverse problem for observable electrode potentials is a set $\mathbf{C}$ of clusters of a minimal size.

The theorem suggests that the binary activation of neurons presented as tightly coupled neurons satisfies the Hamming norm regularized solution of the potential field equation (3). It follows from *Lemma* 1 (the solution has a form of tightly coupled source locations) and *Lemma* 2 (there is a smallest cluster among all others satisfying the regularization condition).

The above discussion concerned a single electrode case. Considering multiple electrodes does not change the outcome of the analysis assuming that there are, possibly, multiple clusters – distinct or tight up to each other – satisfying the condition:

$$\boldsymbol{\varepsilon}_{\min} < \mathbf{L}2(\,\mathbf{P}_{\mathrm{E}}\,(\mathbf{C})) = \mathbf{L}2(\widetilde{\mathbf{D}}\;\mathbf{X}\,(\mathbf{C})) < \;\boldsymbol{\varepsilon}_{\max}$$

The above analysis leads to the conclusion that the forward modeling can be performed on a set of clusters rather than arbitrarily chosen neurons. One of the elements of the proposed method described in Table 1. (n. 2) should be reformulated as follows:

Run the forward model for random sets of activated **clusters** – collect the dataset by *mapping the activated clusters onto electrode potentials*:

```
        NeuronClusterSet → ElPot.
```

Clearly, this approach based on clusters rather than single neurons should facilitate the forward modeling. It is also natural in the context of the brain functioning.

VIII. Use of Clusters in Forward Modeling

A. *Clusters of sources as a solution of a binary programming problem*

In the previous section, it was shown that the clusters exemplify a solution of the binary discrete models regularized by using the Hamming norm. Therefore, the clusters can be used for the forward modeling to gather data in the inverse solution.

The binary programming problem can be formulated in the electrodes-neurons models considered and solved directly for relatively small problems and carefully selected constraints.

We have analyzed this formulation to verify the fact the solution has the form of clusters.

Here is the task:

Find a binary vector $\mathbf{X} = \{x_i = [0, 1],\, i \in [1..N]\}$ that delivers the minimum of $F = \sum_1^N x_i$ under the constraints

$$\boldsymbol{\varepsilon}_{\min} < \mathbf{L}2(\widetilde{\mathbf{D}}\mathbf{X}) < \;\boldsymbol{\varepsilon}_{\max}$$

where, as above, $\mathbf{L}2$ is a quadratic norm, $\widetilde{\mathbf{D}}$ is a radial function operator, and $\boldsymbol{\varepsilon}_{\min}$, $\boldsymbol{\varepsilon}_{\max}$ are the given lower and upper bounds of electrode potentials.

Most commonly, the problems of integer/binary programming as above can be solved using the *Balas* method [7] with so called implicit enumeration with the branch and bound search method. Similar implementation can be obtained with a MATLAB function, `intlinprog`, which uses a Mixed-Integer Linear Programming (MILP) method.

B. *Binary programming exemplifies clusterization*

Consider the following toy example: a 2D model with 100 neurons located at equal distances d = 1 from each other along the x-axis at level y = 1and with two electrodes (at x = 20 and 80) above them at level y = 2. Given the observed potentials on electrodes and their expected bounds, find the activated neurons.

A few solutions are shown on Fig. 4 with the electrode potentials and neuron activations (0/1) along the vertical axis.

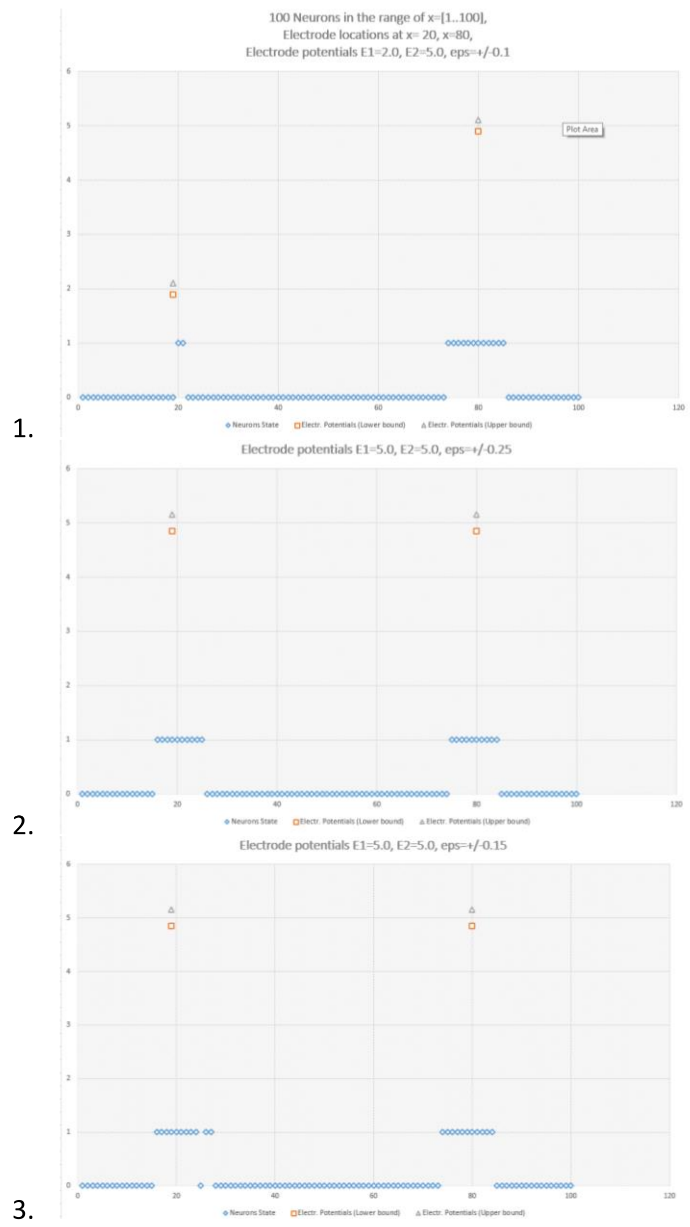


Fig. 4. Binary programming solutions as activated clusters of neurons

The above examples of binary programming problems represent a set of tests ran to illustrate the principal of clusterization as a valid solution of the inverse problem under consideration.

It was also found the importance of the proper selection of the bounds (bins) within which the electrode potentials are to be given. The diagram 3 in Figure 4 shows the effect of *spurious* clusters due to the tighter bounds (eps = 0.15) as compared with diagram 2 (eps = 0.25) where a spurious cluster is not to be found.

Conjecture. Spurious clusters occur at tighter constraints, or low sources density, or the both. Higher accuracy of modeling requires very large number of sources in the given domain which may be prohibitive when using the binary programming method of solution.

C. Clusterization

We consider a few methods of creating clusters for forward modeling: neurologically motivated selection of clusters; a coarse analysis of potential field using the method of fundamental solutions; and an agglomerative clusters based on the dendrite structures.

1) Neurology based selection

It uses the areas of the brain associated with specific emotional responses and other identifiable brain functions. In [1], "240 most stable voxels across six presentations of emotion words were superimposed on one another, with resulting clusters of 25 or more voxels" (see Fig 1). These emotion related clusters have specific level of confidence based on the experiments with human participants.

More examples of neurological clusterization are given by the *Berkley semantic map* research project [6]. (See Fig. 5)

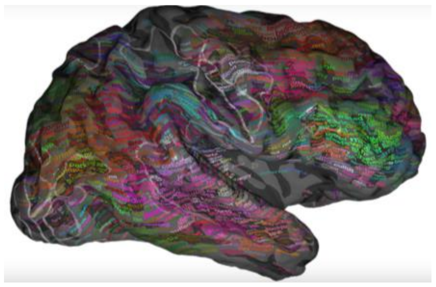


Fig. 5. Semantic map of the cortex [6]

This research shows that the semantic as well as emotional information is represented and distributed in many regions of cortex.

2) MFS based solution

We also consider another approach to identifying the brain areas associated with the neurological activities. It involves the continuous inverse modeling for finding approximate (coarse) locations of the possible clusters.

A specific process of modeling the data available from an EEG research related to the study of emotion effects were used here.

A geometric model of the brain was created first by using the MATLAB prototype for the MRI research (the mri.m module) as shown in Fig. 6.

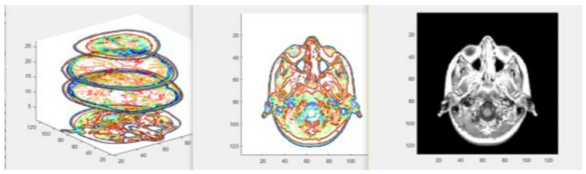


Fig. 6. Views of the brain geometry used in the continuous model

The experimental data presented in [4] were used to position 25 electrodes (vectors `cortex_P)` on the head's surface. Next, the surface of the cortex underlying the electrodes was geometrically defined using a surface fitting procedure such as

```
sf_cortex = fit(cortex_P,'biharmonicinterp');
```

In the referred experiments, the EEG readings were taken to record emotional responses by the human participants to two types of the visual stimuli: a dull (grey) image vs. a bright flower. The values of 25 electrode potentials were measured as ERP P300 (*Event Related Potentials with 300* ms *delay*).

They were applied as the boundary conditions for solving equation (3) by using the MFS [8]. The method requires setting the points of singularities. In the considered case, we used the points in proximity of the electrode positions.

After applying the collocation method, the potential field has been calculated. An example of the field calculated for one of the stimuli above is presented on Fig. 6. Its shape clearly indicates the positions where the clusters can be identified.

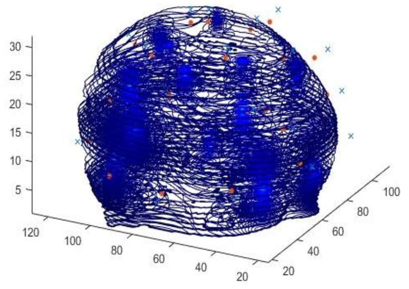


Fig. 7. An example of the collocation solution using the MFS

The collocation equation (7) is underdetermined if the number of chosen singularities exceeds the number of conditions on a relatively small number of electrodes (which is typical). We investigated the use of the two collocation methods to find the values `X` at singularities (here `A`: the $\widetilde{\mathbf{D}}$ and `b`: $\mathbf{P}_S$):

```
% pseudoinverse
X = A\b
X = pinv(A)*b
```

The two implementations of the Moore-Penrose method produce different results: the backslash operator only aims to minimize norm(A*x-b), whereas pinv also minimizes norm(x). The latter method comes close to the Hamming norm approach proposed here.

3) Aglomerative clusters

In this case, we assume that any part of the brain domain can be a valid cluster possibly presented in the solution of the inverse problem. So, the task becomes to construct the complete *voxel* coverage of the cortex domain.

The creation of clusters is based on the dendrite structures found in the brain models (such as in Fig. 8).

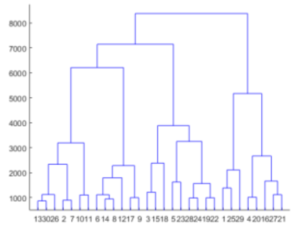


Fig. 8. An example of the dendrite structure of the neurons in the model of cortex

The process of clusterization here is controlled by the description of the external and internal boundary surfaces of the cortex (this surface fitting was mentioned before). The space between the boundary surfaces is populated with clouds of random neuron sources. Figure 9 illustrates the principle of the agglomerative cortex clusterization used here.

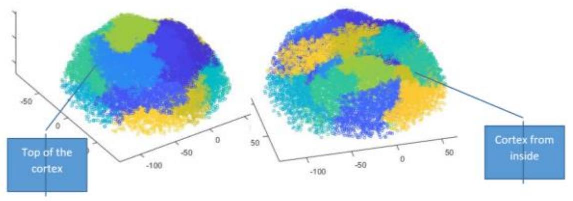


Fig. 9. An example of cortex clusterization

The clusters shown from top and inside the cortex are constructed using the Latin hypercube method to produce 50,000 neurons locations as the `cortexP` structure. The obtained neuron cloud projected on a plane is viewed as in Figure 10.

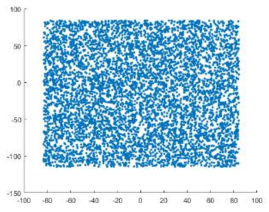


Fig 10. Neuron cloud distribution (2D view)

Then, the following MATLAB function is called:

```
clusterdata(cortexP,'linkage','ward','maxclust',64);
```
to produce 64 clusters (voxels) by geometric agglomeration.

The technique as above allows to construct relatively large neuron population within the cortex geometry and then produce a significant number of agglomerative clusters (see Figure 11 as an example).

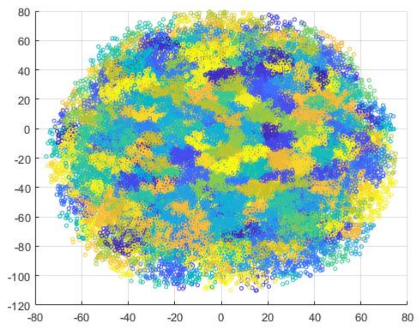


Fig. 11. A clusterized cortex containing 100,000 neurons agglomerated into 1024 clusters (2D view)

The locations of neurons in different combinations of clusters can be extracted to be used in forward modeling.

IX. Deep Learning Model

A. Deep ANN

A deep fully connected Artificial Neural Network has the following architecture:

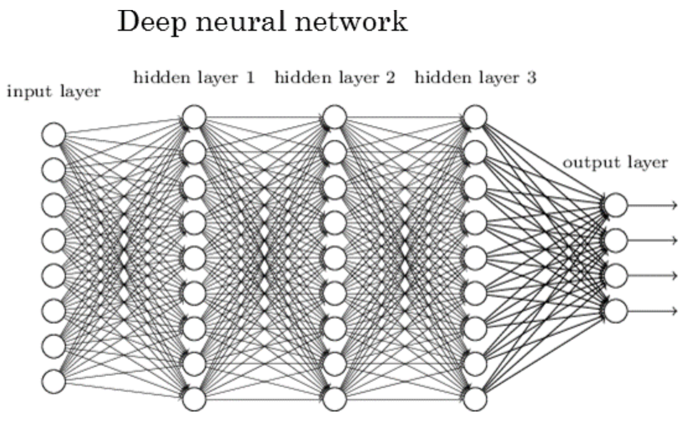


Fig 12. Deep ANN architecture

The input layer accepts the continuous values of the electrode potentials. The number of the layers is subject of variation as well as the number of nodes in each layer.

The output layer of this model represents the status of the clusters. Each output line is assigned to a specific cluster and shows the mean squared error [0..1.0] of the expected presence of the cluster in the group that produces the electrode potentials used as the input.

The value on an output line is interpreted as the probability of the corresponding cluster to be part of the activated group of clusters.

On the diagram above, the nonlinear activation functions and the modules calculating the loss functions used in the output layer are not shown. It is also important to select an appropriate optimization algorithm used in the backpropagation. These architectural features will be found experimentally.

Among many known platforms used for Deep ANN implementation the TensorFlow and Keras became very popular and efficiently used in many applications [11], [12].

B. *A forward model as a set of randomly activated clusters*

First, we study the effect of multiple clusters activities.

The process of using the Deep Learning modeling starts with collection of data reflecting the effect of randomly activated clusters onto the electrode potentials. The dataset is created here based on the *forward modeling* of the assumed model of neuron activity.

The dataset, then, is used for the Deep Learning model (a) design; (b) training, (c) validation and (d) prediction.

The activation of clusters is the significant part of cognition [9]. In interpretation of EEG, the researchers can rely on "the assessment of the number of possible configurations, or microstates, that the system can adopt" [10]. The clusters, or *ensembles of microstates*, interact among themselves, thus creating the effect of *connectomes* which suggests what combinations of clusters relate to specific cognitive or emotional states. That leads to the ability to interpret the observed distribution of the EEG electrode potentials.

Another important aspect of this analysis is the *information-theoretical* argument: the process of clusters activation has a statistical nature and can be characterized by the *entropy* level.

Although this subject, as it comes in the context of the current study, requires a significant and targeted analysis which we leave for the future – there have been a few simple steps performed here to understand how the study can be continued.

First, we looked at the effect of the Bernoulli activation of the clusters to find out how the electrode potentials would look like. As a first example, we consider 50 time steps (number of realizations of random activation of 50,000 sources) for 25 electrodes.

If all activations are uniformly distributed with probability 0.5 or close, the entropy becomes 0: no information processed ("Is the brain dead?"). This situation is shown in Fig 13.

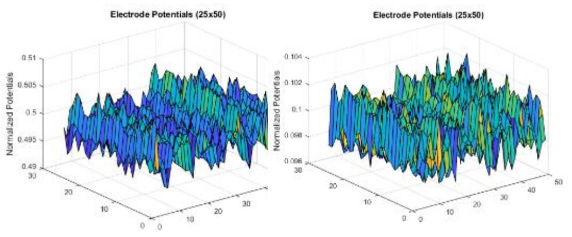


Fig. 13. Two examples of potentials of 25 electrodes induced by random activation of neurons during 50 timesteps

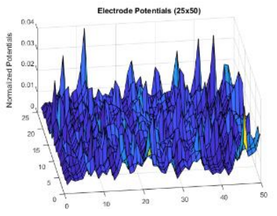
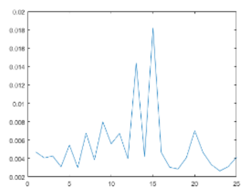


Fig. 14. The brain is active!

However, if, at the different timesteps, only single and different clusters become activated as shown in Fig. 14, the electrode potentials are affected distinctly, and it becomes possible to associate the electrode potentials with specific clusters.

Here is a toy example. Consider a cortex model with 100,000 neurons agglomerated into 100 clusters. One of the clusters (#4) is selected to be activated (the selected cluster has 261 neurons). Calculate the normalized distribution of potentials of 25 electrodes. The result is shown in Fig. 15.



"The inverse problem's conclusion" would be that if the electrodes 13 and 15 are found to have dominant potentials then it is likely that the cluster #4's activation is responsible for such potentials.

Fig. 15. A toy example: a cephalic potential distribution due to the activation of a single cluster (cluster #4)

C. *Experimental Dataset for DNN training*

We consider an experimental dataset to study the Deep ANN for the suggested method of the EEG mapping onto the clusters. The set consists of 256 combinations of the 8 clusters defined on the set of 50,000 neurons.

The calculated effect of each cluster is given in Table 2.

TABLE 2. Electrode potentials generated by the clusters

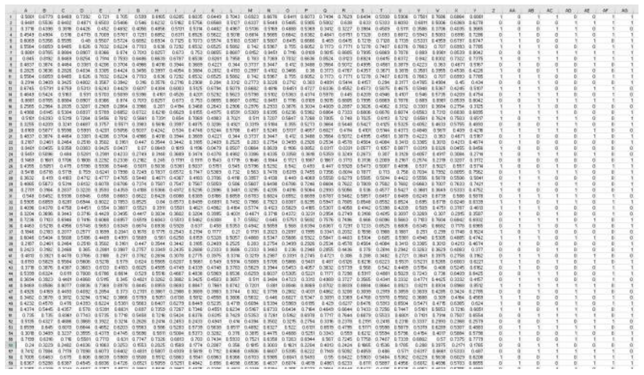
Electrodes	Cluster1	Cluster2	Cluster3	Cluster4	Cluster5	Cluster6	Cluster7	Cluster8	TOTAL Potential
1	0.04013	0.07183	0.14451	0.09547	0.08315	0.26498	0.15308	0.14685	1.00
2	0.04548	0.07805	0.17285	0.15007	0.10279	0.14123	0.14144	0.16808	1.00
3	0.03579	0.0758	0.17194	0.07623	0.0773	0.20002	0.19232	0.17061	1.00
4	0.03834	0.071	0.21935	0.18419	0.10364	0.08621	0.11642	0.18084	1.00
5	0.04247	0.12098	0.12744	0.05882	0.08765	0.07034	0.26307	0.22923	1.00
6	0.04369	0.08233	0.16041	0.16493	0.15839	0.05424	0.10222	0.23379	1.00
7	0.06998	0.22877	0.09784	0.05746	0.12263	0.04964	0.1555	0.2182	1.00
8	0.06488	0.10942	0.11266	0.08989	0.22672	0.04736	0.1041	0.24498	1.00
9	0.16188	0.16508	0.09246	0.0644	0.15618	0.0502	0.13069	0.17911	1.00
10	0.11963	0.1274	0.09974	0.07765	0.21878	0.05034	0.11535	0.19111	1.00
11	0.04793	0.08659	0.12974	0.07896	0.08735	0.18119	0.23394	0.1543	1.00
12	0.05071	0.08061	0.15928	0.19668	0.12355	0.09155	0.12577	0.17187	1.00
13	0.06641	0.11843	0.11271	0.06876	0.09852	0.09072	0.27987	0.16458	1.00
14	0.0639	0.09096	0.1384	0.16188	0.17215	0.06933	0.11899	0.1844	1.00
15	0.11059	0.17785	0.09914	0.06396	0.11892	0.06212	0.19536	0.17206	1.00
16	0.0852	0.1039	0.1143	0.10609	0.23251	0.05624	0.11306	0.18869	1.00
17	0.03172	0.06677	0.3024	0.09887	0.08088	0.10293	0.12977	0.18666	1.00
18	0.03308	0.08647	0.17013	0.0723	0.10122	0.04942	0.11609	0.37128	1.00
19	0.05768	0.14283	0.10209	0.06441	0.16473	0.04307	0.11158	0.3136	1.00
20	0.10569	0.20744	0.09309	0.0605	0.15107	0.0477	0.1336	0.20092	1.00
21	0.08769	0.12453	0.10097	0.07747	0.2383	0.04663	0.10767	0.21674	1.00
22	0.03186	0.0791	0.21909	0.06889	0.07892	0.08587	0.17891	0.25736	1.00
23	0.03257	0.06938	0.26647	0.13099	0.10039	0.06163	0.10664	0.23193	1.00
24	0.04546	0.10061	0.12936	0.08769	0.16716	0.04585	0.10321	0.32066	1.00
25	0.04563	0.15014	0.11673	0.05973	0.11237	0.04936	0.1488	0.31724	1.00

Dataset consists of max 2^8=256 combinations of clusters.

1	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1
1	0	0	1	1	0	1	0
0	1	0	0	1	0	1	1
1	1	0	0	1	1	0	1
1	1	1	0	0	0	1	1
0	0	1	1	1	1	1	1
1	0	1	1	0	1	1	1
0	1	0	0	0	1	0	1
1	0	0	1	0	0	0	1
1	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1
1	1	1	0	0	1	1	0
0	1	1	1	0	0	1	0
0	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

Assuming random activation of the clusters, such as in the 17 realizations shown on the left, and their effect on electrode potentials (Table 2), the dataset can be obtained as partly presented in Table 3.

TABLE 3. The dataset for the Deep ANN training and validation



D. *Training of model: Clusters in the EEG mapping problem*

Here is an example of implementation of the Deep Neural Network using the Keras with the TensorFlow as a backend. In this example, the dataset presented above is used.

The Deep ANN model has the following meta-parameters.

Number of inputs (Electrode potentials):	25
Number of outputs (clusters):	8

Each output is of real value and represents the probability of the corresponding cluster to be activated when the potentials as given.

The dataset (as shown in table 3) was produced by a MATLAB program and split onto two files. One of them is `train_Console.csv` file with 200 sets of 25 electrode potentials and the corresponding 8-bit string representing the activated clusters. The other one, `test_.csv,` has 20 lines of the same structure as above. It is supposed to be used for testing and prediction.

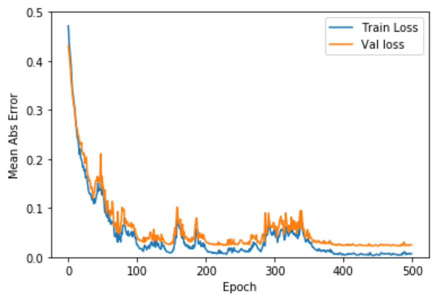
The deep learning model is designed as a *sequential* and *dense* combination of three layers each consisting of 1024 neurons plus the output layer. The non-linear activation function is selected as ReLU (this function provides a better approximation as compared with the “classical” sigma function). The backpropagation is supported with *AdamOptimizer* which is found to be better performing in this case.

The training performed with the `train_Console.csv` file included also the validation of the results (for validation, 20% of the data in this file were used).

Originally, 500 epochs were set to train. At epoch 1590, validation loss stopped decreasing while train loss continued to improve. It means that the model is overtrained (overfit) – it improves the accuracy for the training set but do not *generalize* to work for a new validation set.

The better results produced when stopped training at 100 - 120 epochs. With the trained model by that time, validation shows better generalization.

This effect is considered to be one of the most efficient regularization technique for the Deep ANN training.



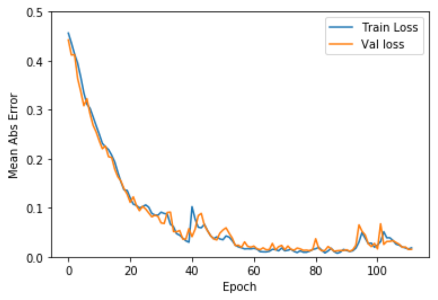


Fig. 16. The train/validation losses as functions of the number of epochs; the graphs show the convergence *before* (top) and *after* regularization

E. *Prediction results*

After completion of training the testing (prediction) was performed by using the file `test_.csv`.

The following example illustrates the obtained results.

Here is a sample of three sets of activated clusters for which the electrode potentials are used as the inputs:

1 0 1 1 1 0 0 0   0 0 1 0 1 0 0 1   1 0 0 0 1 0 1 1

Here are the predicted outcomes for these cases:

[0.73 0.17 1.0 0.9 1.0 0.05 0.09 0.03]

[0.14 0.09 1.00 0.05 1.0 0.02 0.003 0.9]

[0.69 0.1 0.03 -0.02 1.11 0.01 0.99 0.91]

Applying the condition

Predicted activation level = 1 if output > 0.7

and = 0 otherwise

the predicted results are fully compatible with the test expectations.

The full analysis and broader experiments are still needed to make a conclusion about validity and accuracy of this Deep Learning ANN model.

The example above is a part of the broader experimental study concerning solving the inverse problems of linear algebra. Some discussion and the results of the experiments are given in the Appendix 2.

X. CONCLUSION

Here we presented and analyzed a new method of solving some 3D inverse problems by using the Deep Learning technique.

The main subject of this study is the EEG mapping technique that requires to identify, in real time, the areas of the cortex being activated as a result of various stimuli and cognitive processes.

The method is based on obtaining the empirical data by using some forward-feeding models. In this paper, we used a potential field model described by the 3D Laplace equation and analyzed by the Method of Fundamental Solutions.

Unlike some other methods of solving the 3D inverse problems for physical fields, this technique considers the random sets of the source variables (neurons) having only binary activation values. The method most commonly used in such cases is the binary programming which is a NP-complete problem and has some computational limitations.

The proposed technique of solving such problems is based on regularization using the Hamming norm which leads to a solution presented as the set of clusters.

We considered different ways of performing the clusterization of the cortex domain. The examples were used for training a Deep ANN model created and analyzed with the TensorFlow and Keras implementations.

The presented results have mostly illustrative value. We are going to explore some other Deep ANN techniques (Autoencoder, CNN, RBM, etc.) to understand better how to identify clusters with highest likelihood of representation the given data.

One of the important tasks to be addressed in the future research was formulated earlier:

Estimate the hypothesis $\mathscr{H}$: *the Deep ANN as defined here generalizes beyond the dataset used for the training*.

This analysis will be given in the upcoming parts of this project.

REFERENCES

[1] Karim S. Kassam. Identifying Emotions On The Basis Of Neural Activation, PLOS, 2013

[2] Hämäläinen, M.S., and Ilmoniemi, R.J. Interpreting measured magnetic fields of the brain: estimates of current distributions. Tech. Rep. TKK-F-A559, Helsinki University of Technology, Espoo, 1984.

[3] Dale AM, Liu AK, Fischl BR, Buckner RL, Belliveau JW, Lewine JD, Halgren E. Dynamic statistical parametric mapping: combining fMRI and MEG for high resolution imaging of cortical activity. Neuron 2000, 26: 55-67.

[4] R.D. Pascual-Marqui. Standardized low resolution brain electromagnetic tomography (sLORETA): technical details. Methods & Findings in Experimental & Clinical Pharmacology 2002, 24D:5-12.

[5] M. D. McKay, et.al. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics, Vol. 21, 1979.

[6] Alexander G. Huth, at al. Natural speech reveals the semantic maps that tile human cerebral cortex (Nature, 2016)

[7] E. Balas, An Additive Algorithm for Solving Linear Programs with Zero-One Variables, JORSA, 13, (1965) pp 517-546.

[8] A.Karageorghis, G. Fairweather, The method of fundamental solutions for elliptic boundary value problems, Advances in Computational Mathematics. 9 (1998) 69–95.

[9] P. L. Nunez. Toward a quantitative description of large-scale neocortical dynamic function and eeg. Behavioral and Brain Sciences, 23(03):371–398, 2000.

[10] R. Guevara Erra, et.al, Towards a statistical mechanics of consciousness: maximization of number of connections is associated with conscious awareness, arXiv: 1606.00821, Jan 2017.

[11] TensorFlow, https://www.tensorflow.org/

[12] KERAS: THE PYTHON DEEP LEARNING LIBRARY, HTTPS://KERAS.IO/

Appendix 1. The TensorFlow model for Training An EEG model and Prediction of Clusters

```
# -*- coding: utf-8 -*-
"""
@author: roman tankelevich
"""
#
https://www.tensorflow.org/tutorials/keras/basic_reg
ression

import tensorflow as tf
from tensorflow import keras

import numpy as np

print(tf.__version__)

# Number of inputs n: Electrode potentials;
# Number of outputs m: Probability of a cluster;

n = 25
m = 8

# split into input (X) and output (Y) variables
train = np.loadtxt("train_Console.csv",
delimiter=",")
train_data = train[:,0:n]
train_labels = train[:,n:(n+m)]# split into input
(X) and output (Y) variables

test = np.loadtxt("test_Console.csv", delimiter=",")
test_data = test[:,0:n]
test_labels = test[:,n:(n+m)]


print(train_data[0,0:n])  # Display sample features,
notice the different scales
print(test_data[0,0:n])  # Display sample features,
notice the different scales
print(test_labels[0,0:(n+m)])  # Display sample
features, notice the different scales


def build_model():
  model = keras.Sequential([
    keras.layers.Dense(1024, activation=tf.nn.relu,

input_shape=(train_data.shape[1],)),
    keras.layers.Dense(1024, activation=tf.nn.relu,

input_shape=(train_data.shape[1],)),
    keras.layers.Dense(1024, activation=tf.nn.relu),
    keras.layers.Dense(m)
  ])
optimizer = tf.train.AdamOptimizer(0.001)
# Learning rate = 0.001

  model.compile(loss='mean_squared_error',
              optimizer=optimizer,
              metrics=['mae'])
  return model

model = build_model()
model.summary()

# Display training progress by printing a single dot
# for each completed epoch.
class PrintDot(keras.callbacks.Callback):
  def on_epoch_end(self,epoch,logs):
    if epoch % 100 == 0: print('')
    print('.', end='')
EPOCHS = 500

# Store training stats
history = model.fit(train_data, train_labels,
epochs=EPOCHS,validation_split=0.5, verbose=0,
                   callbacks=[PrintDot()]
)

import matplotlib.pyplot as plt


def plot_history(history):
  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error')
  plt.plot(history.epoch,
np.array(history.history['mean_absolute_error']),
           label='Train Loss')
  plt.plot(history.epoch,
np.array(history.history['val_mean_absolute_error'])
,
           label = 'Val loss')
  plt.legend()
  plt.ylim([0,0.5])

plot_history(history)


model = build_model()

# The patience parameter is the amount of epochs to
# check for improvement.
early_stop =
keras.callbacks.EarlyStopping(monitor='val_loss',
patience=20)

history = model.fit(train_data, train_labels,
epochs=EPOCHS,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop,
PrintDot()])

plot_history(history)

[loss, mae] = model.evaluate(test_data, test_labels,
verbose=0)

print("Testing set Mean Abs Error:
${:7.2f}".format(mae))

test_predictions =
model.predict(test_data)#.flatten()
print(test_data[0:4,:])
rounded = [round(x[0]) for x in test_predictions]

print(test_predictions)
print(rounded)
```

APPENDIX 2. DEEP ANN MODEL FOR FINDING AN INVERSE OPERATOR A^{-1} IN EQUATION Y = A X.

Here are some technical details and examples from the study of the ANN model to solve Linear Inverse Problems of various complexity.

Train set.

The dataset was created based on a linear model:

Y = A X, with a given matrix A.

Randomly selected vector X is multiplied by matrix A to produce Y. Y is considered to be an input for the sought model and X it's the output.

Thus, the task is to construct the Deep ANN model of the operator A^{-1}.

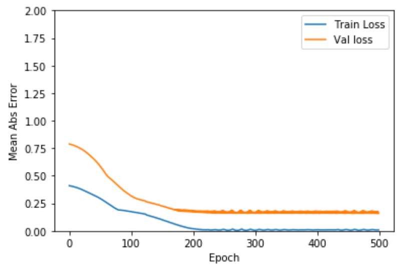
Data was prepared by a MATLAB program.

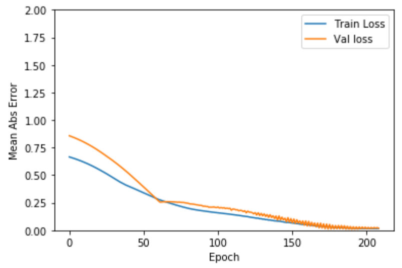
Example 1.

```
K = 5;%100;      % Number of train samples
KK = 100;        % Number of test samples
N = 2;           % Input/output size
X_ = rand(N,K);  % ANN output (inverse)
Y_ = A*X_;       % ANN input (inverse)
X = X_';         % ANN output
Y = Y_';         % ANN input
train_ = [Y X];  % train data
def build_model():
  model = keras.Sequential([
    keras.layers.Dense(32, activation=tf.nn.relu,
        input_shape=(train_data.shape[1],)),
        keras.layers.Dense(32
           activation=tf.nn.relu),
        keras.layers.Dense(2)
  ])

  optimizer = tf.train.RMSPropOptimizer(0.001)
# Learning rate =
  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae'])
  return model
# Store training stats
```

```
history=model.fit(train_data,train_labels,
epochs=EPOCHS,    validation_split=0.5,    verbose=0,
callbacks=[PrintDot()] )
```





Validation split = 0.5 means that 2 of 5 data was used for validation.

Figure 1. Originally, we set 500 epochs to train. At Epoch 200, validation loss stopped decreasing while train loss continued to improve. It means that the model is overtrained (overfit) – it improves the accuracy for the training set but do not *generalize* to work for a new (validation set).

Figure 2. So, it is better to stop training at 200 Epochs. With the trained model by that time, validation shows better generalization.

Prediction. The found model can be used to predict the output for unknown (test) data.

100 test samples used were estimated as 0.02 Mean Abs Error accurate.

Expected output:

```
0.800280468888800    0.141886338627215
0.421761282626275    0.915735525189067
0.792207329559554    0.959492426392903
0.655740699156587    0.0357116785741896
0.849129305868777    0.933993247757551
```

Model's output (predictions):

```
8.038849234580993652e-01   1.323804557323455811e-01
4.420972466468811035e-01   8.988556861877441406e-01
8.168487548828125000e-01   9.403240680694580078e-01
6.459368467330932617e-01   4.603081196546554565e-02
8.744648098945617676e-01   9.154735207557678223e-01
```

MAE=0.02 is appropriate but not the best result. It could be improved by adjusting more meta-parameters.

Model's modification.

(a) Change optimizer to AdamOptimizer (from RMSPropOptimizer).

(b) Change number of neurons in each of two layers to 64 (from 32).

It accelerates the convergence for the better estimate of accuracy (0.01).

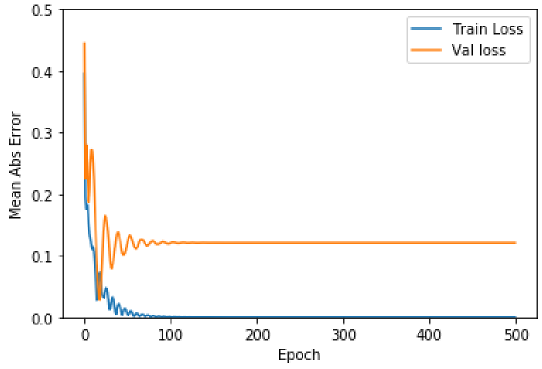
It is achieved at Learning rate = 0.01 vs. 0.001.

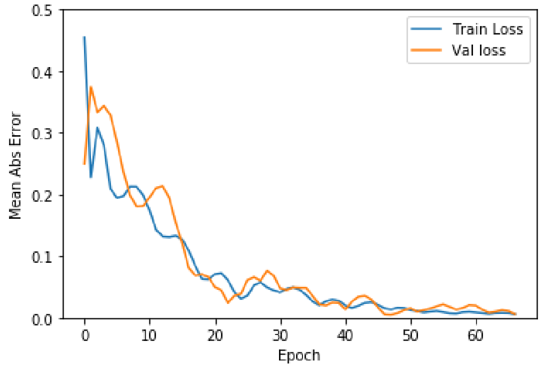
Expected output:

0.800280468888800	0.141886338627215
0.421761282626275	0.915735525189067

Model's output (predictions):

0.80820912	0.13480571
0.4112173	0.92070287





Conjecture. The training dataset size (here 5) is unlikely a significant factor in improving the convergence (the size should be appropriate for the model's assumed number of degrees of freedoms). In other words, 4 tests being linearly independent should be enough to construct the conditions for finding parameters of 2by2 matrices.

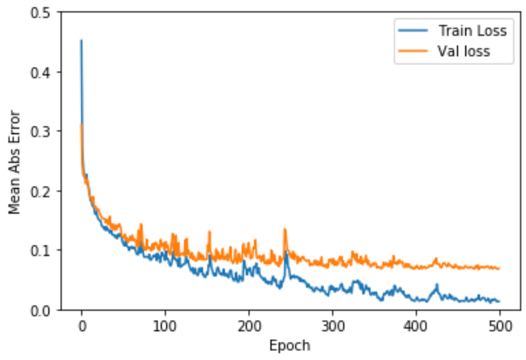
Example 2. Finding inverse operator model for 10by10 matrix

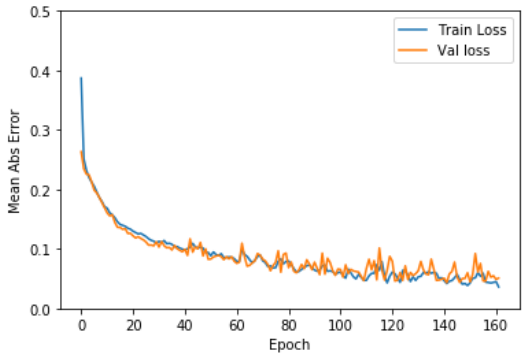
In tf_inverse_problem:

```
model = keras.Sequential([
keras.layers.Dense(1024, activation=tf.nn.relu,
        input_shape=(train_data.shape[1],)),
keras.layers.Dense(1024, activation=tf.nn.relu,

input_shape=(train_data.shape[1],)),
keras.layers.Dense(1024, activation=tf.nn.relu),
       keras.layers.Dense(n)
     ])

optimizer = tf.train.AdamOptimizer(0.001)
```

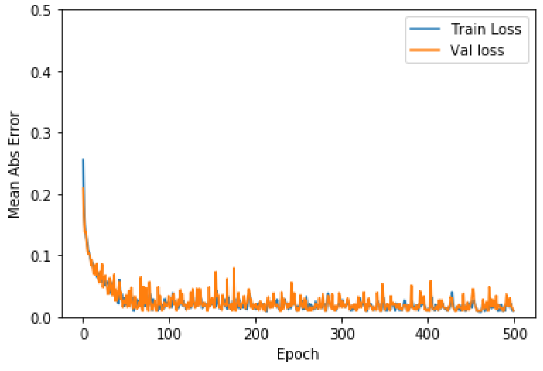


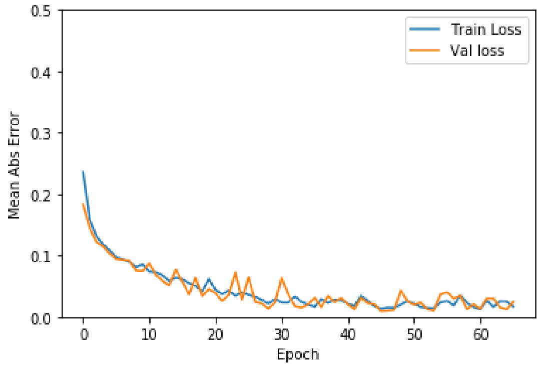


The model has the following meta-parameters:

Layer (type)	Output Shape	Param #
dense_380 (Dense)	(None, 1024)	11264
dense_381 (Dense)	(None, 1024)	1049600
dense_382 (Dense)	(None, 1024)	1049600
dense_383 (Dense)	(None, 10)	10250

Total params: 2,120,714
Trainable params: 2,120,714
Non-trainable params: 0





The process converges fast.

The effect of larger size of the train data 2000 (instead of 200) is more pronounced as it can be seen on the experimental results from the left.