System based Machine Learning for Analyzing EEG Waves

By: Lior Solomon

Supervisor: Raid Saabni

Project submitted in partial fulfillment of the requirement for an MSc

degree in Computer Science

The Academic College of Tel-Aviv Yaffo

School of Computer Science

December 2016

Contents

I. Abstract

Electroencephalography (EEG) is a noninvasive measurement for modeling brain activity as waves within the exterior layers of the brain. Many subfields of Psychology and brain analysis uses these models to analyses emotions, diseases or even as an alternative way of communication, such as the case of Brain Computer Interface (BCI). In this research, we use machine learning techniques in general and deep learning with Artificial Neural Network in specific to analyze the EEG brain activity in order to enable BCI and emotion diagnosis. Due to the easy access to cheap kits of nearly professional tools for recording the electrical activity of the brain like Emotiv or OpenBCI toolkits, it is easier to research and develop tools for BCI and diagnosis based on these affordable kits.

In this research, we use a well-defined EEG dataset to train and test out models. A preprocessing phase is used to extend, organize and manipulate the structure of available data sets to our needs for better training and testing results. We use several techniques to improve system accuracy. Data augmentation, system tuning and different system initializations strategies were used for that purpose. We also make different scenarios of simulations in order to get the best results, such as using smaller groups of EEG channels to get same results as larger groups.

Classification of EEG patterns may enable to diagnose the human brain action and reaction in certain cases like move limbs, driving, sleeping, listening to music, study, trying to concentrate or even when the human body or brain is sick or having mental disease.

II. Introduction

The study of the human brain is very diverse and have intersections with many fields of science including computer science. One aspect of such intersection is the machine learning approach involving Artificial Neuron Network (ANN) to analyze and learn from EEG channels about human activities, which is the main subject of the proposed research.

In 1875, Richard Caton presented first findings about electrical activity of animal subjects, rabbits and monkeys. Following the findings, Hans Berger discovered the alpha wave activity in the human brain. Alpha waves presents resting state for the brain, its dominant just before and after a sleep. In 1924, Hans Berger record the first human EEG.

Electroencephalography (EEG) is a monitoring method which can help to record the electrical activity of the brain. This electrical activity can lead us to better understand the human brain and how it functioning. Earlier observation were shown that the EEG spectrum contains common characteristic that fall within 5 known brain waves types: gamma (> 27 Hz), beta (12 Hz – 27Hz), alpha (8Hz – 12 Hz), theta (3Hz – 8 Hz) and delta (0.2 Hz – 3 Hz). The range of the Hz in each type can be a little different from one research to another.
In order to sample electrical brain activity – EEG, electrodes consisting of small metal discs with wires, are pasted onto the scalp. The electrodes detect electrical charges that result from the activity of the brain cells (neurons). This electrical activity are amplified and translated as a graph on the computer display. Exists a variety of brainwave reading headset in the market.
With this ability, systems based Brain Computer Interface (BCI) enable to send commands to an electronic device only with brain activity which can affect people with motor disabilities. BCI relies on a classification algorithm in order to identify the brain activity pattern.
There are a wide spectrum of classification techniques used in machine learning with ANN being one of them. Recently sophisticated variants of ANN with deep architecture and unique learning algorithms are giving the state of art results in many learning tasks. There are two main types of ANN based on machine learning, supervised where output values are known beforehand, using variants of back propagation algorithm and unsupervised where output values are not known in advanced, historically used for clustering, and recently are one of the reasons of the real breakthrough of the last years advance in the ANN field.

Generating data sets for training and testing in such filed is tedious and take much time and effort. Therefore, in this research, we use standard benchmarks with the 5 known brain waves types: gamma, beta, alpha, theta, and delta; we plan to analyze a well-defined EEG dataset. In order to be able to work with the dataset on our ANN system. For the proposed data set, we have changed its structure and change it accordingly for our needs keeping the main purpose and data. We used several techniques to improve the accuracy, such as augmentation methods, system tuning, deeper network and use given values to initiate the system, details in section VI. There are different ways to augment the dataset, such as elastic deformation, white Gaussian noise, smoothing and more ways to achieve that, we will explore some.

In this research we will answer the question of the possibility classification of a motor imagery task of a subject based on learning patterns of recorded EEG signals, and what compact set of channels would be most effective.

III. Comprehensive Literature Review

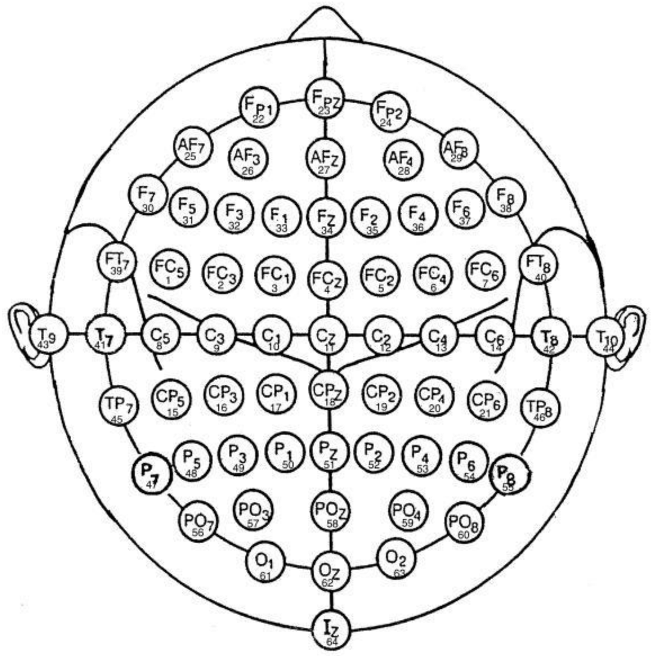
1. Electroencephalogram, EEG

EEG measures electric brain activity and can evaluate electrical activity in the brain. Electrical currents created during synaptic excitations of the dendrites in the neurons.

In order to record EEG, the systems use electrodes, amplifiers, analog to digital converter and a recording device. There are several ways to record the activity and this one is very popular due to the easily to do so. The activity can be detect using flat metal discs (electrodes) attached to the scalp. It does not provide the best quality signals because it need to go through several layers and can be affected by background noise. The waves that came out from the EEG signals is measured over time.

Hans Berger invent the EEG and was the first to record the human Electroencephalogram in 1924.

Usage of EEG is broad, detection problems in the electrical activity of the brain, these problems can be associated with brain disorders – sleep disorders, stroke, brain tumor, head injury, other brain diseases, determine the level of the brain activity when someone is in a coma, etc.; EEG can be used in Brain Computer Interface (BCI), control devices through the brain;



[1]Demonstrates a mapping of 64 electrodes that give 64 channels that can record EEG.

There are sort amount of channels that can be used, it can be 14, 16, 64 or other number of channels. Different numbers of channels can give different results. It can connect to the human head with a cap or separate electrodes.

EEG comprises a set of signals – brain waves, which may be classified according to their frequency. Well-known frequency ranges have been defined. Some researchers define the ranges a little bit different from others.

[1] http://www.bbci.de/competition/ii/albany_desc/albany_desc_i.html

EEG has been used mainly to evaluate neurological disorders, since Hans Berger's original paper, in the clinic and to investigate brain function in the laboratory. Over this time EEG thought to be used to decipher thoughts, or intent, so that a person could communicate with others or control devices directly by means of brain activity. EEG attract little serious scientific attention until recently. EEG reflects brain activity, person's intent could be detected in it; EEG based communication requires the capacity to analyze the EEG in real-time, now the technology exist and it's not so expensive.

2. Brain waves

The brain is made up of billions of brain cells called neurons. Neurons use electricity to communicate with each other. The electricity can be detected using sensitive medical equipment for EEG. EEG signals are measured in microvolts (one millionth volt). The brain waves can changed according on what the person is doing, feeling and on which mental state is in. The brain waves types' frequency is diverse for each type between Information Sources. There are 5 known brain waves types as follow.

Gamma 27 Hz and up - High frequency brainwaves. Gamma brainwaves fades during deep sleep. Too much Gamma brainwaves can be from stress and anxiety. When the person is study, formation an idea the Gamma brainwaves appears.

Beta 12hz - 27hz - This is the normal awake state of consciousness during the day, people found in mental state where they are awake and in this state. It is seen usually when we are in focus, engaged in a problem solving and decision making.

Alpha 8hz - 12hz - Alpha brainwaves presents resting state for the brain. Alpha brainwaves dominant close to sleep, before and after. Too much alpha waves can cause daydreaming and lose focus.

Theta 3hz - 8hz - Most often occur in light slip or deep meditation and relaxing. Theta represent a mind in low sense and deeply relaxed person. Too much can point on depression, impulsivity and inattentiveness. A small amount of Theta waves is detected in a normal awake adult.

Delta 0.2hz - 3hz - Delta waves generated in deepest meditation and dreamless sleep, Deep sleep. In this state, the body is healing and "resetting" itself. In this state the human is completely unconscious. Sleep and depressants increase delta waves. It will be unusual and abnormal to detect Delta waves in adults when in an awake state.

3. Neurons

Neurons are cells that processes information through **electrical** and chemical **signals**. Neurons specialized to carry "messages" through an electrochemical process. Neurons come in many different shapes and sizes. The central nervous system is composed from neurons. The average human brain has about 100 billion neurons (or nerve cells). Each neuron may be connected to up to 10,000 other neurons, passing signals to each other via as many as 1,000 trillion synaptic connections.

Information transmission within the brain, takes place during the processes of memory, is achieved using a combination of electrochemical - chemicals and electricity. A typical neuron possesses a soma dendrites and a single axon.

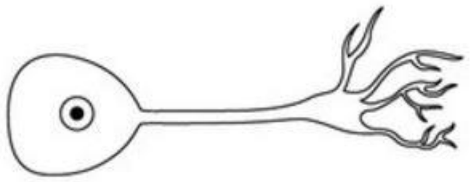
Every neuron maintains a voltage **gradient** across its membrane, due to metabolically-driven differences in ions of sodium, potassium, chloride and calcium within the cell, each of which has a different charge. If the voltage changes significantly, an electrochemical pulse called nerve impulse is generated. **This electrical activity can be measured** and **displayed as a wave form called brain wave**.

This pulse travels rapidly along the cell's axon, and is transferred across a specialized connection known as a synapse to a neighbor neuron, which receives it through its feathery dendrites. A synapse is a complex membrane junction or gap used to transmit signals between cells, and this transfer is therefore known as a synaptic connection. Although axon-dendrite synaptic connections are the norm, other variations are also possible. A typical neuron **fires** 5 - 50 times every second.

Each individual neuron can form **thousands of links** with other neurons. Functionally related neurons connect to each other to form **neural networks** also known as neural nets or assemblies. The connections between neurons are not static, though, they change over time. **The more signals sent between two neurons, the stronger the connection grows**, with each new experience and each remembered event or fact, the brain slightly re-wires its physical structure.

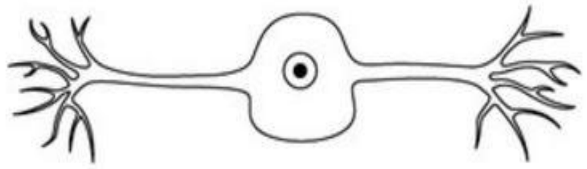
There are many specific types of neurons in the human brain, but there are three general types: Bipolar, Unipolar, Multipolar

Unipolar:



Unipolar neurons only have one process emerging from the cell. Humans do not have true unipolar cells. These neurons are not found in vertebrates, but are found in insects where they stimulate muscles or glands. But instead have pseudo-unipolar cells. They are called pseudounipolar cells because although one process extends from the cell body, this single process then divides into two, with one end extending from the sensory receptors and the other end extending to the central nervous system.

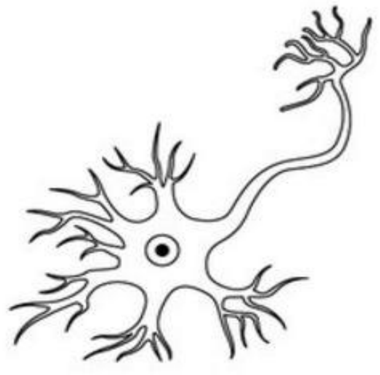
Bipolar:



A bipolar neuron is a nerve cell located in humans and is defined by two processes that connect to the cell body: one dendrite and one axon.
A neuron that has two processes arising from opposite poles of the cell body.

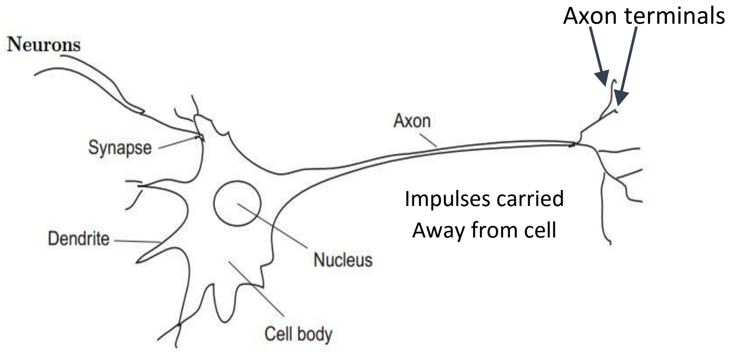
Multipolar**:**

Multipolar neurons, are the most common type of neuron. They are located in the central nervous system. Have more than two processes emanating from the neuron cell body. These types of neurons are constructed in such a way as to allow immense connectivity and networking with other cells.



4. Artificial Neural Network

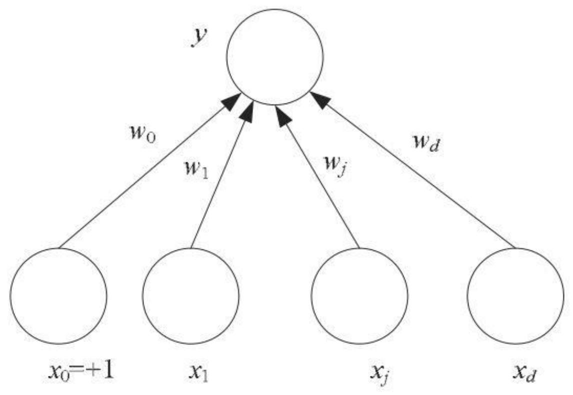
Artificial neural network (ANN) models, take their inspiration from the brain. Brain's neurons receives signals through synapses located on the dendrites or membrane of the neuron. When the signals received are strong enough (surpass a certain threshold), the neuron is activated and emits a signal though the axon. This signal might be sent to another synapse, and might activate other neurons.



ANN, is a classifier that simulating working principle of the human brain. ANN basically consist of inputs (like synapses), which are multiplied by weights (strength of the respective signals), and afterwards computed by a mathematical function which determines the activation of the neuron. Another function (which may be the identity) computes the output of the artificial neuron. The higher a weight of an artificial neuron is, the stronger the input that is multiplied by it will be. Weights can be negative; we can say that the signal is inhibited by the negative weight. Depending on the weights, the computation of the neuron will be different. By adjusting the weights of an artificial neuron we can obtain the output we want for specific inputs. When we have an ANN of thousands of neurons, it would be complicated to find all the weights. But we can find algorithms which can adjust the weights of the ANN in order to obtain the desired output from the network. The process of adjusting the weights is called learning or training. One of those algorithms is called Backpropagation, backward propagation of errors.
Generally, a computer has one processor. Conversely, the brain is composed from a large number of processing units (neurons), $\sim 10^{11}$ operating in parallel. The neurons are believed to be much simpler and slower than a processor in a computer. The brain is believed to provide its computational power from the large connectivity. Neurons in the brain have connections, called synapses, to $\sim 10^{4}$ other neurons, all operating in parallel. In a computer, the processor is active and the memory is separate and passive, it is believed that in the brain, both the processing and memory are distributed together over the network. Processing is done by the neurons, and the memory is in the synapses between the neurons. There are two important types of artificial neuron: the perceptron and the sigmoid neuron.

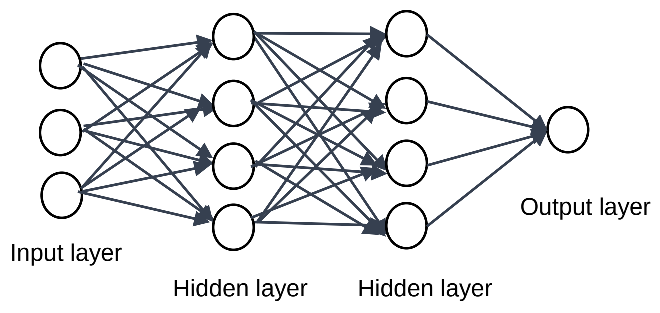
5. Perceptron

A type of artificial neuron. Developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts.
Perceptron is the basic processing element. It has inputs which can be from the environment or may be the outputs of other perceptrons.



Perceptron. x_j, j = 1,...,d are the input units. x_0 is the bias unit that always has the value 1. y is the output unit. w_j is the weight of the directed connection from input x_j to the output.

It's easy to see that, $y = \sum_{j=1}^{d} w_j x_j + w_0$. w_0 – intercept value to make the model more general.



Presents parallel perceptrons with hidden layers. The left most layer is called the input layer, and the neurons within the layer are called input neurons. The right most or output layer contains the output neurons, or, as in this case, a single output neuron. The middle layer is called a hidden layer.

Let's say, there are K > 2 outputs, there are K perceptrons, each of which has a weight vector w_i.

How to train a Perceptron?
The perceptron defines a hyperplane, and the neural network perceptron is just a way of implementing the hyperplane. Given a data sample, the weight values can be calculated offline, the perceptron can be used to calculate the output values. In training neural networks, we generally use online learning where we are not given the whole sample, but we are given instances one by one and would like the network to update its parameters after each instance, adapting itself slowly in time.

6. Sigmoid Neuron

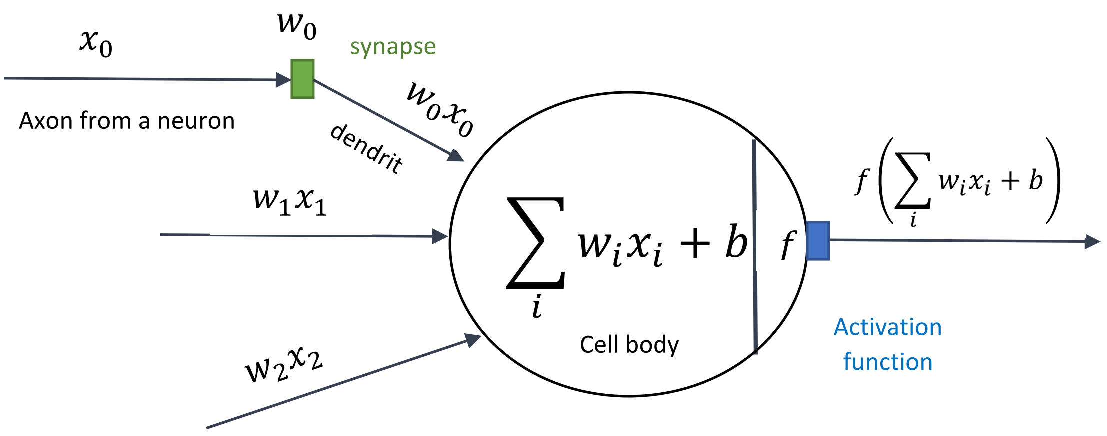
A type of artificial neuron. Sigmoid neurons are similar to perceptrons, but modified so that small changes in their weights and bias cause only a small change in their output.

Just like a perceptron, the sigmoid neuron has inputs, x_1, x_2, x_3, … instead of being 0 or 1, inputs can take on any values between 0 and 1.
So, for instance, 0.752… is a valid input for a sigmoid neuron. Like a perceptron, the sigmoid neuron has weights for each input, w_1, w_2, w_3, … and an overall bias, b. The output is not 0 or 1. Instead, it's $\sigma(w \cdot x+b)$ where σ is called the sigmoid function:

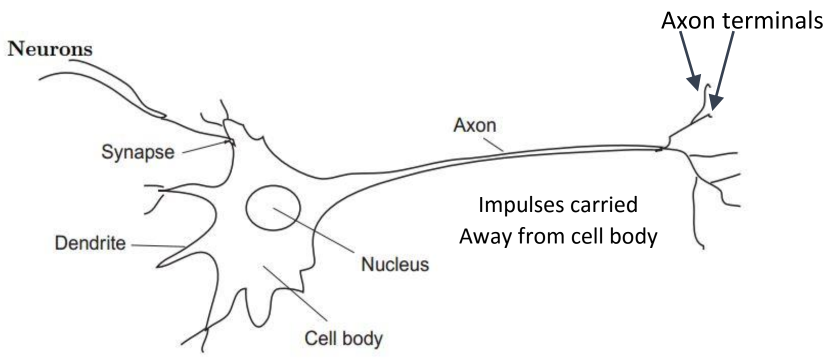
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

7. Activation function

The output of a neuron (y) is a function of the weighted sum $y = f(v)$
Activation function or activation in short for a neural network is defined as the mapping of the input to the output via nonlinear function (transform function), like sigmoid or tanh function at each node in the net.

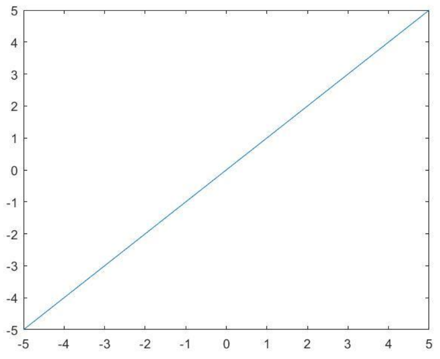


Resemblance to the neuron:



There are several well-known activation function that have different properties, like: range, derivative, if it's a monotonic or not and more.
When creating the neural network for the case problem, a "right" activation function should be choose.

8. Linear function

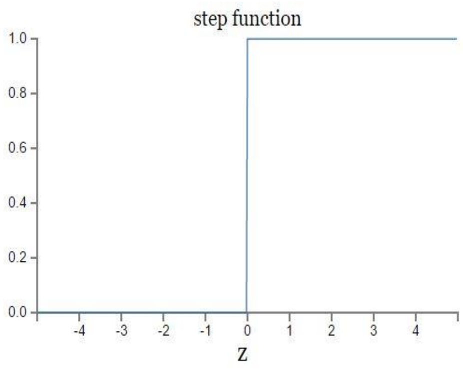


$$f(v) = b + v = b + \sum w_i x_i$$

The parameter "b" stands for the bias. A neuron becomes a linear model with bias b being the intercept and the weights, $w_1, \ldots, w_m$ being the slopes.

Usually, nonlinear activation function is been used in nontrivial problems via neural networks.

9. Step Function



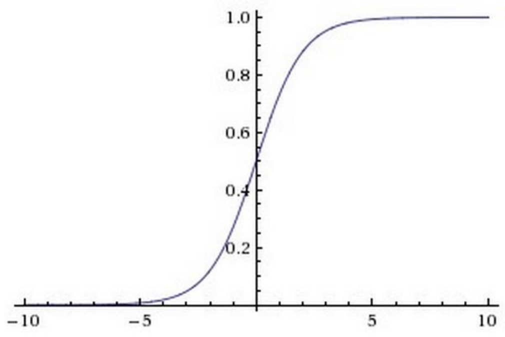
$$f(v) = b + v = b + \sum w_i x_i$$

$$f(v) = \begin{cases} 1 \;\; if \;\; v \geq b \\ 0 \;\; otherwise \end{cases}$$

b Called the threshold

The step function is very similar to the sigmoid function as describe below. As we can see, the sigmoid function is actually a smoothed differential version of the step function.

10. Sigmoid Function



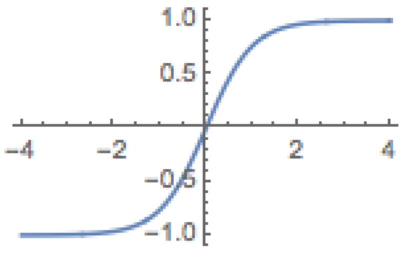
Defined by: $\sigma(z) = \frac{1}{1+e^{-z}}$

Get real-valued number and "squashes" it into range [0,1]. In general, large negative numbers become 0 and large positive numbers become 1. Interpretation to firing rate of a neuron: not firing at all – 0, Fully-saturated firing at an assumed maximum frequency – 1.

The question of using this function as part of the training process has major draw backs such as Gradients drawback (See gradient section III.14). When the neuron's activation saturates at 0 or 1, the gradient at these regions is almost zero. If the local gradient is very small, it will effectively "kill" the gradient and almost no signal will flow through the neuron to its weights and recursively to its data. If the initial weights of sigmoid neurons are too large then most neurons would become saturated and the network will barely learn.

If σ had been a step function then the sigmoid neuron would be a perceptron – the output would be 0 or 1.

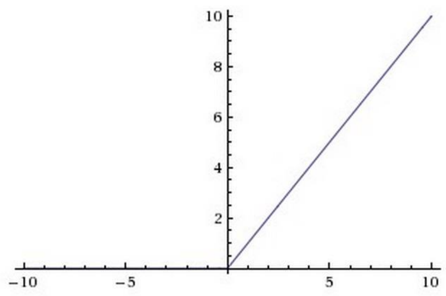
11. Tanh function



$\tanh(x)$ non-linearity function can be used as activation function. It gives the hyperbolic tangent of x.

$\tanh(x)$ function give a real valued number to the range [-1,1]. Like the sigmoid neuron, its activations saturate, but unlike the sigmoid neuron its output is zero-centered. In practice the $\tanh(x)$ non-linearity is always preferred to the sigmoid nonlinearity.

12. Rectified Linear Unit, ReLU



$f(x) = \max(0, x)$ threshold at zero.

x – input to a neuron.

The ReLU can be implemented by thresholding a matrix of activations at zero – not suffer from saturating.

The ReLU function are less expensive from other activation functions.

The ReLU units can cause a problem during training. Large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again – this can cause the gradient flowing through the unit will forever be zero from that point on. This can lead to a network with un-useful neurons that never activate across the entire training dataset.

There are several variants of the RLU function: leaky ReLU, Noisy ReLU, parametric ReLU, randomized ReLU and more.

13. Cost function

What we'd like is an algorithm which lets us find weights and biases so that the output from the network approximates y(x) for all training inputs x. To quantify how well we're achieving this goal we define a cost function

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$ An example of cost function.

w , denotes the collection of all weights in the network, b all the biases, n is the total number of training inputs, a is the vector of outputs from the network when x is input, and the sum is over all training inputs,x.

When applying the backpropagation algorithm, usually, two main assumptions are taken place about the form of the cost function:

1. The cost function can be written as an average: $C = \frac{1}{n}\sum_x C_x$. Where x is individual training examples. As we can see at the equation above, the quadratic cost function has this form.

2. The cost function can be written as a function of the outputs from the neural network. $cost\, C = C(a^L)$ where L denotes the number of layers in the network. As we can see at the equation above, the quadratic cost function has this form. Since the quadratic cost for a single training example x may be written as $C = \frac{1}{2n}\sum_j (y_j - a_j^L)^2$. Where y is the corresponding desired output; j – index on the neuron.

The learning algorithm find weights and biases for the network, for the network to output classify approximation for each input. To quantify how well the learning algorithm achieving that goal the cost function take place.
The aim of the training algorithm is to minimize the cost function as a function of weights and biases. The algorithm will find the weights and biases which minimize the cost function, the algorithm is called **gradient descent**.

Usually we have limited number of training set. The learning algorithm should generalize the problem and give set of weights and biases such that the network would give the best approximation for a given input, making a small change in the set of the weights and biases would not change much the number of training images classified correctly. But the goal is to make changes in the weights and biases in order to get improved in the performance of the network. Here comes the cost function, like the quadratic cost function at the equation above, give the ability to figure out how to make small changes in the weights and biases in order to get improvement in the performance.

14. Gradient, Gradient descent, Stochastic gradient descent

The gradient descent adjust the weights according to the error they caused. The gradient represents how two variables relate to each other, the relationship between the network's error and a single weight; how does the error vary as the weight is adjusted. The **right weight** will produce the least error; the right weight will correctly represents the signal contained in the input data and translates them to a correct classification; the right weight will classify a "wheel" in an input image to a "car" label and not to a "ship" label.
During the learning of the neural network, it slowly adjusts weights so that they can map signal to meaning correctly. The relationship between network *Error* and each of those *weights* is a derivative $\frac{dE}{dw}$ - measures the degree to which a slight change in a weight causes a slight change in the error.
The signal of the weight passes through activations and sums over several layers, we use the chain rule of calculus: $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$ to march back through the networks

activations and outputs and finally arrive at the weight in question, and its relationship to overall error.
With that equation we can calculate how a change in weight affects a change in Error by first calculate how change in activation affects a change in Error and how a change in weight affect a change in activation.
Feedforward network with single weight, chain rule:

$$\frac{dError}{dweight} = \frac{dError}{dactivation} \cdot \frac{dactivation}{dweight}$$

Adjusting a model's weights in response to the error it produces, until the algorithm cannot reduce the error any more.

It's a learning algorithm for neural networks.
The most common learning algorithm for neural networks is the gradient descent algorithm although other and potentially better optimization algorithms can be used.

15. Learning rate

One of the parameter that can and should be set in order to make gradient descent to work correctly. Before the run of the algorithm, we can choose the learning rate parameter, it need to be small enough that we will get good approximation but at the same time we don't want it to be too small because the gradient decent algorithm will work very slowly – affect the speed of the algorithm to get the equation solution.
It's a training parameter that controls the values and size of weight and bias changes in learning of the training algorithm.

16. MSE - Mean squared error

Measures the averages of the squares of the errors. The difference between the estimators to the estimated. Used to evaluate the performance of an estimator. The mean squared error is also useful to relay the concepts of bias, precision, and accuracy in statistical estimation.

MSE = $\frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2$
n – Number of predictions.
$\hat{Y}$ – Vector of n predictions.
Y – Vector of observed values (vector of the real values).
$\left(\hat{Y}_i - Y_i\right)^2$ – Square of errors.

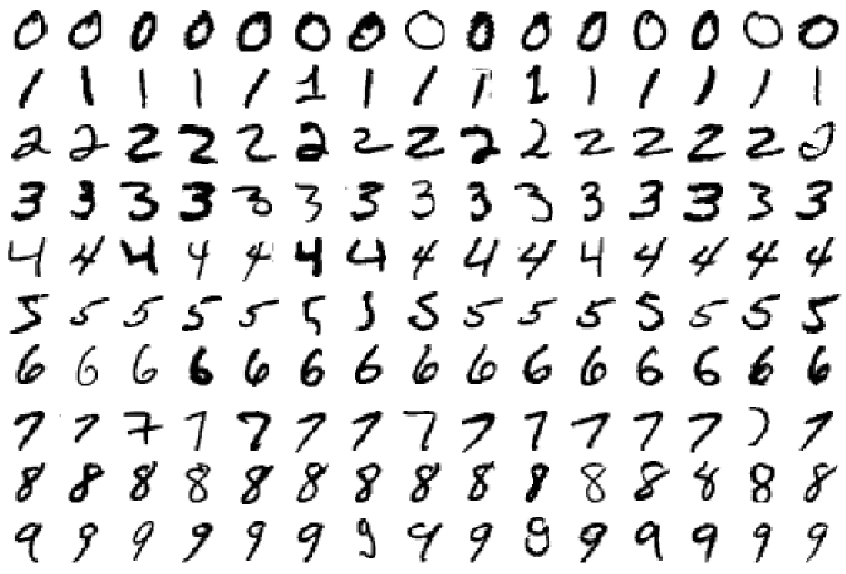
17. Epoch

An epoch is a complete pass through a given dataset. By the end of one epoch, the neural network will have been exposed to every record to example within the dataset once, differs from an iteration – many iterations can occur before an epoch is over. An epoch is part of the learning algorithm in neural network. In order to calculate the weights and errors the learning algorithm can pick out a randomly chosen mini-batch of training inputs and train them, then pick out another randomly chosen mini-batch and train with those, and so on, until the learning algorithm exhausted the training inputs, which is said to complete an epoch of training. And then – start over with a new training epoch.
Mini-batch: The learning algorithm pick a small number m of chosen training inputs $x_1, \ldots, x_m$ and refer to them as a mini-batch.

It determines when training will stop once the number of iterations exceeds epochs.

18. MNIST data [28]

MNIST is the "hello world" of neural networks and deep learning datasets. Many projects uses MNIST to test their neural networks in order to see if the net is actually works. The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. With their correct classification. The training data images are scanned handwriting samples from 250 people, half of whom were US Census Bureau employees, and half of whom were high school students. The images are greyscale and 28 by 28 pixels in size. The test set of the MNIST data also are 28 by 28 greyscale images. It's a good database for people who want to try learning techniques on real-world data. The data is stored in a very simple file format designed for storing vectors and multidimensional matrices.



Images from[2] MNIST
This database is freely available.

[2] http://yann.lecun.com/exdb/mnist/

19. Feedforward

Model of artificial neural network.
A feedforward neural network is a biologically inspired classification algorithm. It consist of a number of simple neuron-like processing units, organized in layers. Every unit in a layer is connected with all the units in the previous layer. The weights on these connections encode the knowledge of a network.
Data enters at the inputs and passes through the network, layer by layer, until it arrives at the outputs. During normal operation, there is no feedback between layers. This is why they are called feedforward neural networks.

Feed-forward networks have the following characteristics:

1. Perceptrons are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers.
2. Each perceptron in one layer is connected to every perceptron on the next layer. Hence information is constantly "fed forward" from one layer to the next, and this explains why these networks are called feed-forward networks.
3. There is no connection among perceptrons in the same layer.

In our research, we have treated all the tasks and subjects mentioned up as free parameters needed to be tested adapted and tuned for best results. Therefore, for example a pre-trained ANN on the MNIST data set have been used to initialize out ANN weights and several step function were considered. A detailed description of this process is given in section VI.

20. Recurrent neural network

Model of artificial neural network
Artificial neural networks in which feedback loops are possible.
These models is to have neurons which fire for some limited duration of time, before becoming quiescent. That firing can stimulate other neurons, which may fire a little while later, also for a limited duration. That causes still more neurons to fire, and so over time we get a cascade of neurons firing. Loops don't cause problems in such a model, since a neuron's output only affects its input at some later time, not instantaneously.

21. Backpropagation

Backward Propagation of Errors. A specific technique for implementing weight for a multilayered perceptron, MLP. The idea is to efficiently compute partial derivatives of an approximating function realized by the artificial neural network (ANN) with respect to all the processing neuron of the adjustable weight vector for a given value of input vector. It's a popular common algorithm for training ANN. Designing and training an ANN using back propagation requires many arbitrary choices such as the number and types of nodes, layers and learning rates. This choices can be critical.
The backpropagation algorithm is used in layered feed-forward ANNs. The artificial neurons are organized in layers, and send their signals "forward", and then the errors are propagated backwards. The network receives inputs by neurons in the input layer, and the output of the network is given by the neurons on an output layer. There may be one or more intermediate hidden layers. The backpropagation algorithm uses supervised learning, which means that we provide the algorithm with examples of the inputs and outputs we want the network to compute, and then the error (difference between actual and expected results) is calculated. The idea of the backpropagation algorithm is to reduce this error, until the ANN learns the training data. The training begins with random weights (or customized weights of other training), and the error should be minimal when the algorithm is done.
It's an algorithm for computing the gradient of the cost function and to understand how changing the weights and biases in a network changes the cost function. This algorithm was introduced in the 1970s and it's one of the most important concepts of learning in neural networks.
One of the expressions that we get when computing the gradient of the cost function C is the partial derivative $\frac{\partial C}{\partial W}$ and $\frac{\partial C}{\partial b}$, where w is any weight and b is any bias in the network. The expressions express how quickly the cost changes when there is a change in the weights and biases. With this knowledge we can understand how changing the weights and biases effect on the behavior of the network.

In order to compute the partial derivatives $\frac{\partial C}{\partial W}$ and $\frac{\partial C}{\partial b}$ the backpropagation algorithm need to compute the errors for each j neuron in each layer l - δ_j^l.

Recap, this technique comes to calculate the gradient – relate weights to error. Repeated method of chain rule of calculus for partial derivatives.
A unique form of backpropagation - backpropagation through time, BPTT, specifically useful for recurrent neural networks, RNN, analyzing text and time series. Each time step of the RNN is the equivalent of a layer in a feed-forward network.

22. Vanishing Gradient Problem

Intuitively we can expect that the more hidden layers we have in the network the more powerful the network is but this is not necessarily true. Suppose we have a neural network that support image recognition with 4 hidden layers, each layer of neurons responsible on the following, First layer: recognize edges; Second layer: recognize Shapes, like Circle; Third layer: face; Forth layer: smile. These multiple layers of abstraction seem to give deep networks a compelling advantage in learning to solve complex image recognition problems. Moreover, there are theoretical results that deep networks are more powerful than shallow networks. It’s a hard task to train such deep network. When trying one with stochastic gradient descent by backpropagation we can run into a problem, the deep network can perform the same as shallow network or even worse. It’s easy to think that if we have a neural network and we add to it a hidden layer it ought to make the network able to learn more complex problems and can classify better. But this is not always the case, it can even get worse.

In deep networks, different layers could learn at different speeds. Could be a situation where later layers are learning well and earlier layers often get stuck during training and can learn almost nothing. This reason of learning slowdown connected to gradient-based learning techniques. The opposite phenomenon can also occur, where earlier layers are learning well and later layers often get stuck during training. Studies have shown in some deep neural networks that the early hidden layers learn much more slowly than later hidden layers - gradient tends to get smaller as we move backward through the hidden layers. The gradient quantity controlled how rapidly the bias changes and how rapidly the weights input to the neuron change during learning. In conclusion it gives that neurons in the earlier layers learn much more slowly than neurons in later layers. This phenomenon - **vanishing gradient problem**.

The problem can be avoided, although the alternative is not very attractive. The opposite phenomenon as mentioned before is called **exploding gradient problem**.
Researches show that the gradient in deep neural networks is unstable - one of the phenomenon can occur. These phenomenons occur due to training with gradient-based learning in deep neural networks.
We need to pay attention that these phenomenons may be avoided or act differently if we initialized the weight in the network in a different way. Neural networks can response differently between different initialization - randomly or take the weights from completely different problem.

Why these phenomenons occurs (vanishing gradient problem and exploding gradient problem)?
The problems lies here:
When we take a look on how the gradient $\frac{\partial C}{\partial b}$ is been calculated and how the expression is look like we can see that the gradient expression has a unique structure: product of terms of the weight w_j multiplied by $\sigma'(z_j)$ - which σ is the sigmoid activation

function and $z_j = w_j a_{j-1} + b_j$ which is the weighted input to the neuron - $w_j \sigma'(z_j)$ (disregard of the layer index which give little different expression - for a simplicity, but it's the same idea). It depends how we initialized the weights in the network - suppose we initialized randomly. We can see that the more terms we have in the gradient expression the product will tend to exponentially decrease. The more terms, the smaller the product will be. As we get closer to the later layers there is less terms in the product and if we have less terms in the product the result will be bigger.

Recall, later layers ➡ less terms ➡ bigger result of the gradient ➡ faster learn ➡ later layers learn better from earlier layers. This is the essential origin of the vanishing gradient problem. This is not a proof but it's a good reason of the problem. We get the exploding gradient problem exactly the opposite case when the gradient grow exponentially as we move backward through the layers. Gradient based learning techniques makes different layers in the network tend to learn at wildly different speeds. When using sigmoid neurons the gradient will usually vanish.

23. Regularization and Overfitting [12]

During the process of training the neural network, we can get into a problem so called overfitting. One way to control and prevent overfitting is by regularization.
When we want to make our neural network better at generalization we should use a regularization methods which known as the cross-entropy cost function.
When we have small set of training and using not many training examples we usually train with larger number of epochs. What could happen is that the classification accuracy on the training dataset can be very encouraging. On the other hand the accuracy on the test dataset could be less encouraging, the learning on the test dataset with small examples, can gradually slow down, and stop improving before get even close to the final epochs, this phenomenon is called overfitting.
When we can probably assume that the learning algorithm is overfitting?
When the algorithm is more accurate in fitting known data, like training dataset, and less accurate in making a prediction on data which the algorithm not seen before, like testing dataset. Hence, the model's accuracy has much better results on the training dataset than on the testing dataset. If this is the case we say that our model is not generalized.
The problem with the overfitting is that the model memorize the training dataset and not learning to generalize the data.
Overfitting can occur when the model have too many parameters comparatively to the number of observations the model got and it can be a result of too complex model and to less training dataset.
If the model is overfitting we should reduce the model flexibility by better features selection and better regularization techniques.

In contrast, poor classification accuracy on the training dataset could occur if the model is too simple and a bad features selection which not describe the target well, the classification accuracy can improved by increasing model flexibility.

A good technique to use in order to prevent overfitting is to use validation data – early stopping. We train the model until the accuracy is saturated.
Other technique is to change the network architecture.
Another technique which can reduce overfitting called regularization techniques such as weight decay or L2. The idea of regularization techniques is to change the cost function and to add a term to it – regularization term. The regularization usually improved network generalization, increase classification accuracies and reducing the effect of overfitting.
The best technique is to augment the size of the training dataset, it can be done by artificial augmentation of the existing data.
In modern network we can find the overfitting phenomenon and it's a real problem in neural networks, we need to have techniques to reduce this effect.

24. Deep Neural Networks

Deep learning is under the domain of Machine Learning. With deep learning we can achieve multiple levels of representation and abstraction that can help to understand how the data is organized within an image, text, sound etc.

There are well known deep learning algorithms with 2 main families.
Supervised learning algorithms:

1. Convolutional Neural Network,
2. Logistic Regression.
3. Multilayer Perceptron.

Semi supervised / unsupervised learning algorithms:

1. Restricted Boltzman Machines, RBM.
2. Deep Belief Networks, DBN.
3. Auto Encoders.

Until 2006, we fail to train deep architectures. Deep networks who trained with backpropagation perform badly than shallow networks. The networks were limited to two layers max.

a. Applications

1. Classifications
    a. It falls under supervised learning category. The algorithm classifies a sample to one of the, given in advanced, categories based on some similarity measurement. Hence, it depends on the labeled datasets, which usually achieved in advanced by human. Some samples of are Text classification.
    b. Face classification.
    c. Object in an image classification.
    d. Voice detection.
    e. And more.

2. Clustering
   Clustering is a way of grouping similar entities to groups. It falls under unsupervised learning. Hence, labels to detect similarities is not required. Most of the problems have data, which are not labeled.
   With clustering we can achieve:
    1. Anomaly detection.
    2. Image segmentation.
    3. Searching. Grouping items with in an image / sound / video.
    4. And more.

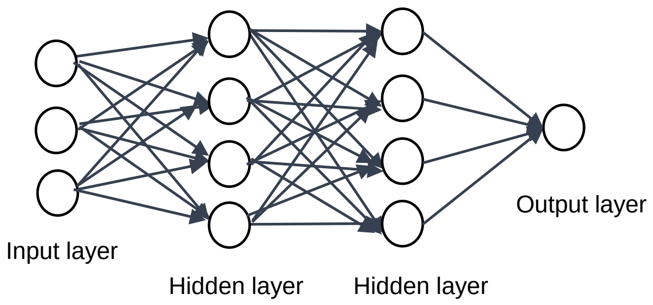
3. Predictive Analytics
   Statistical techniques in order to make predictions on future events. To do so, the system need to be exposed to enough data and to establish correlations between the present and the future.
   With predictive analytics we can achieve:
    1. Fraud detection
    2. Risk management
    3. CRM analytics
    4. Human healthcare
    5. And more.

25. Convolutional Neural Networks

Convolutional Neural Networks, CNN or ConvNet in short, are similar to Neural Networks as we seen above and use the same ideas. Ideas such as backpropagation, gradient descent, regularization, non-linear activation functions. Build up from neurons that have learnable weights and biases. Each neuron receives inputs and performs a dot product. Convolutional Neural Network took their inspiration from living organisms. The visual cortex of an animal contains a complex arrangement of cells. These cells act as local filters over the input space. The work of convolutional neural networks start at 1968. But the modern subject of convolutional networks was published at 1998.

a. Convolutional Neural Networks Architecture

The CNN use a different architecture than artificial neural network, using this architecture makes convolutional networks fast to train. With this architecture we can train deep, many-layer networks. The regular neural networks architecture don't scale well for many use cases, this is because there is a wasteful and huge number of parameters that can be reduced; the full connectivity between the layers can lead to over fitting. One of the advantages of the CNN architecture is, when using it, it makes the algorithm fast to train and it helps to train many layers network (deep learning).



Artificial neural network with fully connected layers – every neuron in the network is connected to every neuron in adjacent layers. CNN use a different architecture than this.

In general, a simple CNN is a sequence of layers, each layer is transforms a volume of activations to another through a differentiable function. There is four main types of layers which the CNN is built from: Convolutional Layer, RELU, Pooling Layer, and Fully-Connected Layer. CNN for image dataset classification could have the following architecture which built from the following layers:

$$\text{Input} \longrightarrow \text{Convolutional} \longrightarrow \text{RELU} \longrightarrow \text{Pooling} \longrightarrow \text{Fully Connected}$$

(In general, the architecture can be built from several layers of convolutional – pooling. Before the fully connected layer, there can be layers like softmax).

The input layer will contain the pixel values of the image (if it's an image context problem).

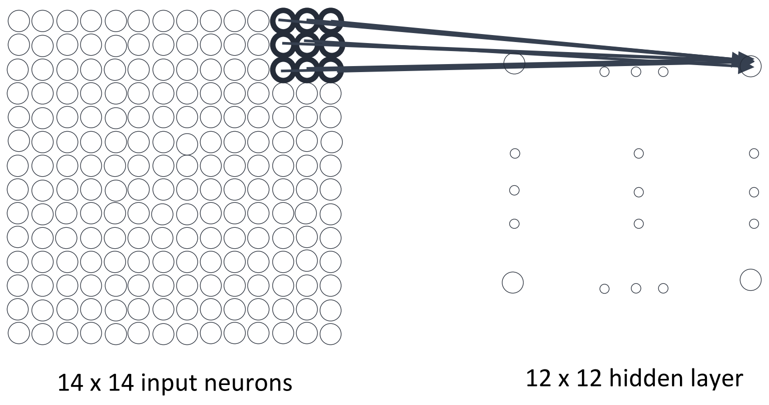
The Convolutional layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.

The RELU layer, will apply an elementwise activation function, such as the $f(x) = \max(0, x)$ thresholding at zero. RELU stands for Rectified Linear Units. This layer contains neurons that applies the activation function. It's not affecting the receptive fields of the convolution layer and increases the nonlinear properties of the decision function and the overall network. To increase nonlinearity, other functions can applies as well, like the sigmoid function $\sigma(x) = (1 + e^{-x})^{-1}$.

The Pooling layer perform a down sampling, progressively reduce the amount of parameters and computation in the network, and control overfitting. The Pooling Layer operates independently on every feature map and resizes it.
The most common function that do so is max pooling – output the maximum activation in a specified region. For example, pooling layer can take a region of 2 x 2 neurons from the feature map layer (the previous hidden layer) and summarize it to get the maximum activation from this region. The needed of the pooling layer is to perform a down sampling and to seek if the given feature is found in the region. There is a possibility to use other technique instead of outputting the maximum activation, like average pooling or L2 pooling which means to output the square root of the sum of the squares of the activations in the region.

The Fully Connected layer will compute the class scores. Neurons in a fully connected layer have full connections to all activations in the previous layer.

The input neurons in convolutional net connected to a layer of hidden neurons such that each neuron in the first hidden layer will be connected to a small region of the input neurons:



This small region, marked in bold, is called local receptive field. As we can see, the input neurons size 14 x 14 is connected to a neuron in the hidden layer. The size of the hidden layer is 12 x 12 because of the size of the local receptive field, 3 x 3. All this is true if we moved the local receptive field by 1 neuron. This parameter (of 1 neuron move) is called stride length. Each hidden neuron has a bias and the same weights (shared weights) and bias (shared bias) is been used for all the 12 x 12 neurons in the hidden layer. The meaning is that all the neurons in the hidden layer detect the same feature, a feature can be an edge or other type of shape in an image (The definition of the shared bias and weights are defining the kernel or filter).

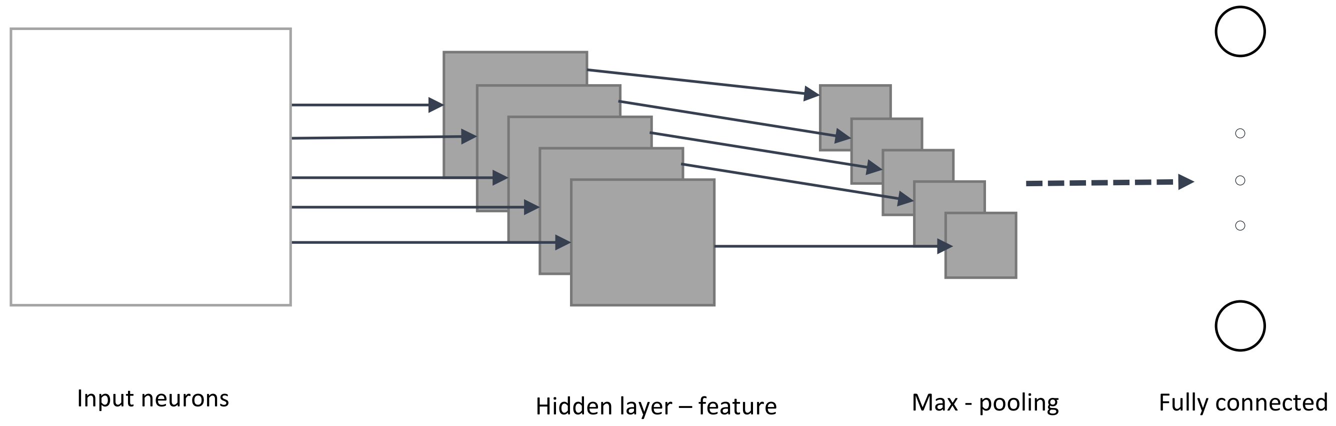
Usually CNNs use more than one feature map and detect several different kinds of features, each feature is being detectable across the entire image.
As mentioned before, convolutional neuron network architecture use shared weights and biases and this give a first advantage on the regular fully connected neural network because the reduction in the number of the parameters been used.
In order to calculate the output of the j,k hidden neuron we should calculate the following equation:

$$\sigma\left(b + \sum_{l=0}^{n} \sum_{m=0}^{n} w_{l,m}\, a_{j+l,k+m}\right)$$

σ – Activation function.
b – Shared bias.
n – Square root of the numbers of neurons in the local receptive fields.
$w_{l,m}$ – Shared weights.
$a_{j+l,k+m}$ – Input activation at (j+l), (k+m) position.

It’s easy to think that the name convolutional neural network came from this equation because of the fact that the left side of the equation (The multiplication) is called convolution operation in mathematics, but the name is by incidentally.

Pooling layer is connected to each feature map, if we have 5 features than there will be 5 pooling in the pooling layer



An example of convolutional neural network architecture. The Final layer is a fully connected layer – every neuron from the pooling layer is connected to every one of the neurons in the last layer.

The convolutional neural network as demonstrated can be trained using stochastic gradient descent and backpropagation. The using of backpropagation in convolutional neural network is different from the backpropagation in the regular fully connected neural network. The root of the differences is in the derivation.

CNN are deep neural network that is currently the state of the art in image processing. They used widely every year on contests like ImageNet to increased accuracy on accepted benchmark.

26. BCI – Brain Computer Interface

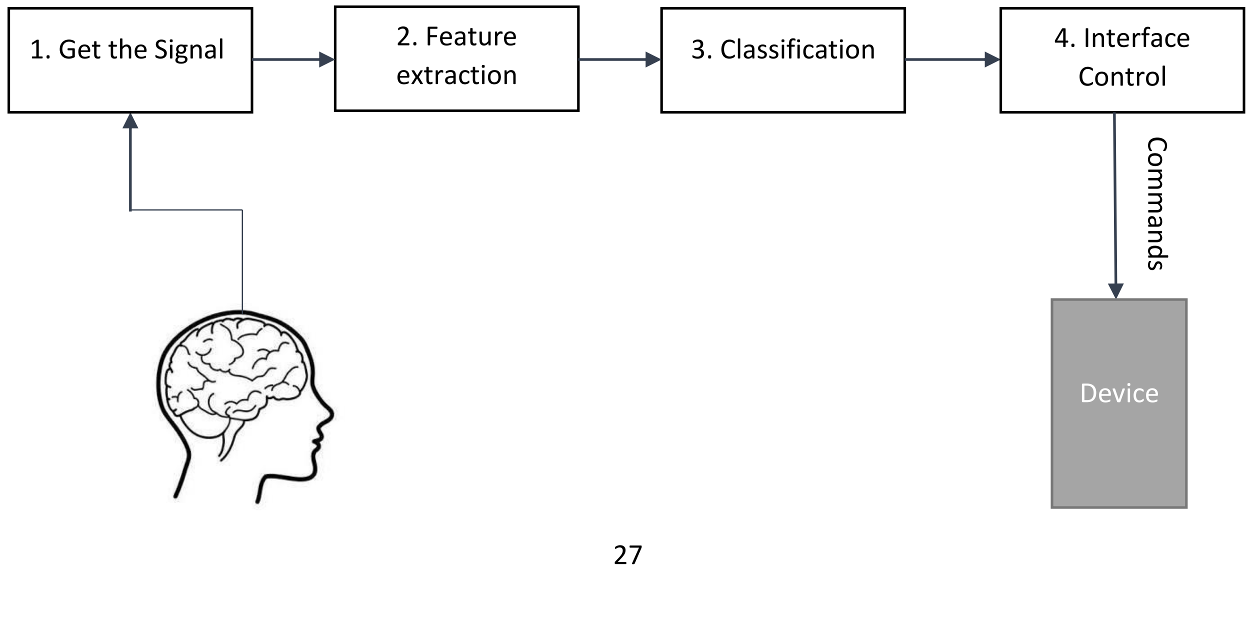
Brain Computer Interface combine hardware and software communication system that permits cerebral activity alone to control computers and other devices. BCI enables to interact with the surroundings, without the involvement of peripheral nerves and muscles, by using control signals generated from electroencephalographic activity. BCI use brain signals to gather information on user intention which record and measures brain activity and translate that information into electrical signals. BCI creates an interactions between humans to external devices, creates an interface to control external devices with the use of the brain of the human only. This kind of interface could provide to humans that suffering from motor disabilities an improvement on their quality life.

BCI can recognize a set of patterns in brain signals. There are several stages to do so:

a. Get the Signals. Capture the brain signals and make noise reduction. Preprocessing the signals in order to be able to process it in more a convenient way;
b. Feature Extraction. Identifies discriminative information in the brain signals that have been recorded. This can be a challenging job, because of the many mixed signals with large number of sets activity in the brain that overlap in time and space, and we don't want to loss information;
c. Classification, classify the signals to achieve pattern recognition in order to decipher the user's intentions;
d. Interface Control, translate the classified signals into the user desired commands for any kind of device such as a computer.

BCI System

As explained above, this figure presents a design of a BCI system. Electrodes on the head acquire the signals from the brain.



The BCI monitor the brain activities to get the information from the brain in order to translate and use it. There are two types of brain activities:

a. Electrophysiological:
   Related to the electrical activity that occur in the brain.
   Between the neurons there is a process of exchanging information, this activity creates the Electrophysiological and it generated by electro-chemical transmitters. The neurons activities generate ionic currents. There is a large variety of current pathways between the neurons. Electroencephalography, Electrocorticography, Magnetoencephalography, and electrical signal acquisition in single neurons measure electrophysiological activity.
   In order to use this type it consist of a set of sensors that acquire electroencephalography signals from different brain areas. The quality of the information is affected by background noise and others.

b. Hemodynamic:
   Related to the blood and glucose in the brain that active neurons at a greater rate than in the area of inactive neurons. The blood stream in the brain is pass in the glucose and oxygen results in a surplus of oxyhemoglobin in the veins of the active area and in a distinguishable change of the local ratio of oxyhemoglobin to deoxyhemoglobin. The changes of the blood stream can get an information regarding the activity in the brain.

   The Hemodynamic type is different from the Electrophysiological, its measure the hemodynamic response and can be categorized as indirect approach and it's not directly related to neural activity.
   The most common type usage is to get the information from the brain through Electrophysiological.

Methods to get neuroimaging

#	Method	Activity measured	Measurement	Risk
1	EEG	Electrical	Direct	Non-invasive
2	MEG	Magnetic	Direct	Non-invasive
3	ECoG	Electrical	Direct	Invasive
4	fMRI	Metabolic	Indirect	Non-invasive
5	NIRS	Metabolic	Indirect	Non-invasive

Our focus will be only on EEG.

27. Techniques to improve accuracy & performance

Enhancing a machine learning model can be a challenging task. There are several techniques and strategies that we will explore. To enhancing our model in “only” 1% can be considered as a success. A predictive machine learning model can be built on different ways. But there are proven ways to improve the accuracy and the performance of the model.
In order to improve accuracy we can look at the data set and at the model itself.
It is highly recommended to understand the given problem, the domain of the problem and know the data and its structure, this can help to improve a model’s accuracy.

a. Augment the training data.
   Adding more data to the training data set can give to the model better understanding about the data. The more data the training model see in the training process it will give more accurate prediction on the testing dataset.
   If we don’t have the option to add more data to the training set, there are techniques to manipulate the existing data and to augment it [24].

b. ANN, Ensemble of networks: create several different neural networks. Get the result from each one and determine the best classification. Even though the networks would all have similar accuracies, they might make different errors, due to the different initializations.

c. Tune the algorithm
   Each machine learning algorithm has a parameters that can be tuned. These change can impact the accuracy outcome. In order to find the optimum value for each parameter to improve the accuracy of the model can be a hard task and in order to do so we should run simulations process to test it.
   For example, regarding ANN there are parameters such as learning rate, deltas, momentum number of epochs. It is even considered to change the architecture – add or reduce layers (CNN – add or reduce pooling or fully connected layer).
   Change the number of training epochs and regularization, can help to reduce the effects of overfitting.
   Using the right cost function can also influence the accuracy.

d. Cross validation
   With cross validations we can find the right answer to the question. We save data set samples a side which we didn’t train the model on, and test the model on those group of samples before finalizing the model.
   It helps to achieve more generalized relationships.

e. Weight initializations
   ANN, when done to design the neural network we need to initial weights and biases. There can be a big difference in the accuracy of the system if we choose

one initialization approach to the other. The intuitive approach is to give random initialize of the weights and biases, there is a better way to initialize the system to have the ability to learn faster and give more accurate results. Researchers’ shows that if we take well learned weights and biases on a completely different topic from our system, and initial our system with those values we can improve the results substantially.

f. Multiple algorithms
For different domain problems there are different machine learning algorithms. For each problem we can use several algorithms that can fit to the problem. It is not easy to pick the right machine learning algorithm for a given problem, the intuition comes with experience and incessant practice. Some algorithms are better to a particular type of data sets than others. We can apply several different algorithms to test who will give the best accuracy. We might need to adjust the data structure from one algorithm to another.

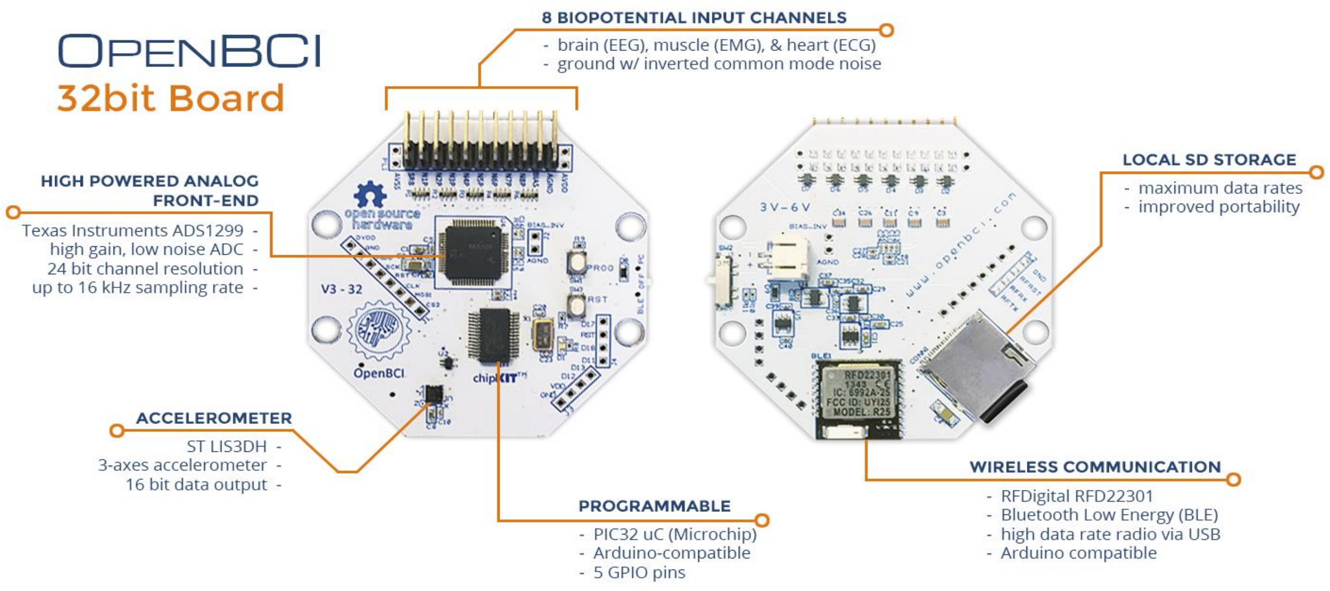
g. Overfitting – see III.23.

28. Tools

As part of the project, we explore several tools that can be used in order to sample, record and give an interface of the electrical activity of the brain, EEG. There are several tools on the market. There are “light” equipment that can be used to sample the EEG waves, which can be used for home or research based needs. We examine the following tools:

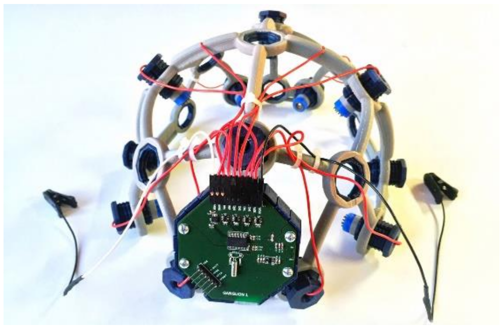
a. openBCI – Open source Brain Computer Interface [2]

OpenBCI is a tool used to measure and record electrical activity produced by the brain (EEG), muscles (EMG), and heart (EKG). The OpenBCI boards can be integrated with open-source EEG signal processing tools like OpenBCI GUI, or others.
The hardware is programmable; it has an integrated microcontroller with digital and analog pins, similar to an Arduino and can be programmed through the Arduino IDE.


OPENBCI
32bit Board
8 BIOPOTENTIAL INPUT CHANNELS
- brain (EEG), muscle (EMG), & heart (ECG)
- ground w/ inverted common mode noise
HIGH POWERED ANALOG FRONT-END
Texas Instruments ADS1299 -
high gain, low noise ADC -
24 bit channel resolution -
up to 16 kHz sampling rate -
ACCELEROMETER
ST LIS3DH -
3-axes accelerometer -
16 bit data output -
PROGRAMMABLE
- PIC32 uC (Microchip)
- Arduino-compatible
- 5 GPIO pins
LOCAL SD STORAGE
- maximum data rates
- improved portability
WIRELESS COMMUNICATION
- RFDigital RFD22301
- Bluetooth Low Energy (BLE)
- high data rate radio via USB
- Arduino compatible

OpenBCI Board.

The figure presents the 32bit OpenBCI Board is an Arduino-compatible, there are 8-channel neural interface with a 32-bit processor.
The OpenBCI 32bit Board can be used to sample brain activity (EEG), muscle activity (EMG), and heart activity (EKG). The board can communicates wirelessly to a computer via the OpenBCI programmable USB dongle. It can also communicate wirelessly to any mobile device or tablet compatible with Bluetooth Low Energy (BLE).


3D-printed headset that records brainwaves and converts them into signals.

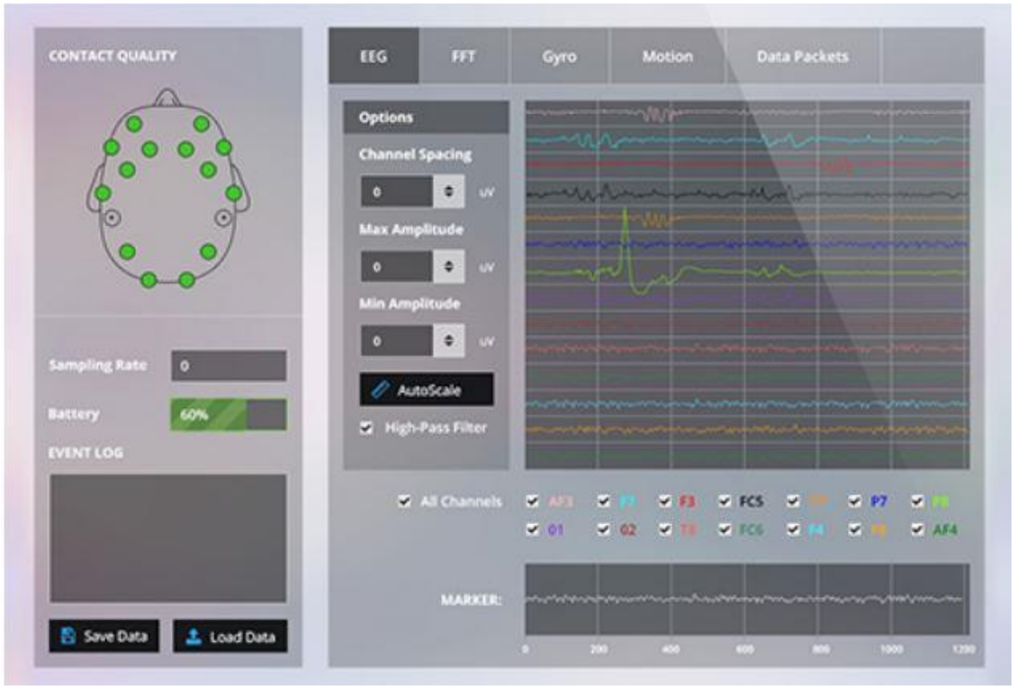
The figure presents the openBCI headset, it capable of allowing the wearer's brainwaves to control software and hardware.
OpenBCI accessible to researchers and developers with no hardware experience. OpenBCI boards have a growing list of data output formats, making them compatible with an expanding collection of existing biofeedback applications and tools.

b. Emotiv EPOC / EPOC+ [1]

Brain Computer Interface and Scientific Contextual EEG. 14 EEG channels + 2 references. A wireless EEG system. Provides access to high quality, raw EEG data using SDK.

emotive EPOC / EPOC+ head set.

The headset come with software application. Real-time display of the Emotiv headset data stream. Record and replay files in binary EEGLAB format. File converter included to produce *.csv format.



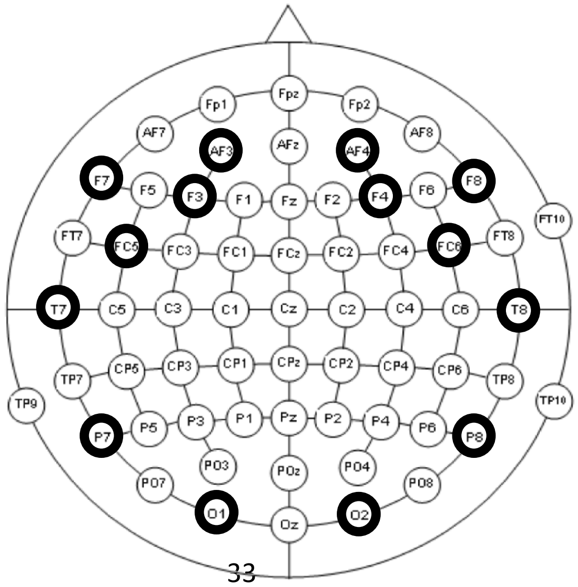
emotiv EEG display

EMOTIV Epoch+ is a 14 channel wireless EEG, designed for contextualized research and advanced brain computer interface (BCI) applications.

Features:

- Easy to fit and flexible design
- Wireless and rechargeable
- Lithium battery providing up to 12 hours of continuous use
- Provides access to Raw EEG data with software subscription

Technical Specifications:
Signals: 14 channels: AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4



Signal resolution

- Sampling method: Sequential sampling. Single ADC
- Sampling rate: 128 SPS or 256 SPS* (2048 Hz internal)
- Resolution: 14 bits 1 LSB = 0.51μV (16 bit ADC, 2 bits instrumental noise floor discarded), or 16 bits
- Bandwidth: 0.2 – 43Hz, digital notch filters at 50Hz and 60Hz
- Filtering: Built in digital 5th order filter
- Dynamic range (input referred): 8400μV(pp)
- Coupling mode: AC coupled

Connectivity

- Wireless: Bluetooth® Smart
- Proprietary wireless: 2.4GHz band

Power

- Battery: Internal Lithium Polymer battery 640mAh
- Battery life: up to 12 hours using proprietary wireless, up to 6 hours using
- Bluetooth Smart

Compatibility

The EMOTIV Epoc+ neuroheadset connects wirelessly to PCs, tablets, and smartphones running Windows, Linux, MAC OSX, Android or iOS.

IV. The Problem and its Sources

1. Channels reduction

Professional EEG recording tools are expensive. Affordable EEG equipment are mostly enables limited scale recording. The more channels the tool can record the more expensive it is. Generally, more channels enables better learning and analyzing. One of the main goals in such case is to gain from 128-channles recorder equivalent knowledge as the equipment's with less channels. What is the differences between them in terms of classification accuracy? In other words, what are the channels which are more discriminative than others?
EEG equipment with vast amount of channels are not accessible to everyone. This unmet need for a less expensive EEG has enticed a number of smaller companies to offer products that supposedly fill this market demand.

2. Improve classification accuracy

One of the problems is to find a well-defined dataset with enough data such that the machine-learning algorithm will be able to learn and generalize well enough to give sufficient results on unseen testing samples. Finding such EEG dataset is important because to creating one can be real and long project. One of the rules of thumb is the more data an algorithm can train on, the more accurate it will be. In real world classification problems, it can be expensive to acquire a big training set – traditional learning algorithms often perform poorly. The goal of the learning is to succeed to get the best classification and the best accuracy. The classification accuracy can be evaluate by the number of correct predictions made divided by the total number of predictions made.

V. The System

1. System description

In any classification system, feature selection and extraction is main and important phase toward successful classification system. In our case it's hard to think directly about which features and which classifiers to use in order to get the best results. In such case Neural Network are general and good option due to their ability to get the data as is, calculate the feature and learn by itself.
The specification of our Neural Network is as follow:
We have used two hidden layers ANN trained by the backpropagation algorithm. The System can be configured with several different functionalities and be set with values upon parameters as initial state. In such case few free parameters could be selected and tuned.

Given weights:

Generally ANN start the learning process with random weight. Many researches in the field shows that starting with weights learned for other tasks improve the learning process.
Run the network with given weights and biases, those weights were trained in advanced on other different topic. (Details in next section)

In order to enable such initialization, the current network and the pre-trained network, should contain the same number of neurons in the first layer.

Augment training data:

Run the network with an extended version of the training data. There were implemented several techniques in order to expand the data. With this approach we can get better results.

Hidden layers:

Number of hidden layers.
Number of neurons in each hidden layer.

Unipolar / Bipolar:

Select Unipolar or Bipolar neuron

Unipolar: $\frac{1}{1+e^{-x}}$

Bipolar: $\frac{-1}{2+e^{-x}}$

Learning rate:

To make gradient descent work correctly, we need to choose the “right” learning rate.

Weight decay rate:

Set weight decay rate value.

Number of Epochs:

Choose the number of epochs.

Resilient gradient descent:

Enable / Disable resilient gradient descent

Before start to train the network:

- The samples were shuffled (randomize shuffle).
- The data were normalized.

Input layer (first layer):

We have two-dimensional array, size: 314 x 22.
There are 22 channels – 22 columns.
Each trail duration length is 314 – 314 rows.
(We can look at it as an image with 314 by 22 pixels)

We get: 314 x 22 = 6,908, a one-dimensional array.
⇨ 6,908 neurons in the input layer (first layer).

(See EEG Datasets [18])

Output layer (last layer):
Contain 4 neurons, indicate the 4 categories of the trails. (See EEG Datasets [18])

2. EEG Datasets

Several datasets were considered, were the main goal is a will defined ground trothed data set with many samples as possible. The purpose was to find a well-defined EEG dataset, which outline a human motor movements. It is a difficult task to find one we could use. Several datasets were examined in this project and the following two were picked to be used in our project:

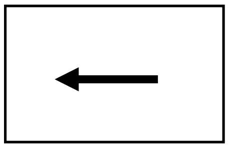
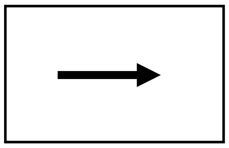
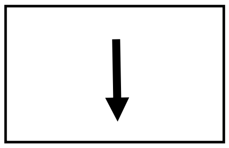
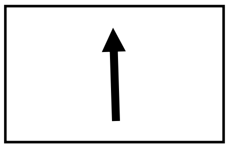
a. An efficient P300-based brain-computer interface for disabled subjects [26]

We start to analyze and learn this dataset, but we considered to neglect it because of the poor definition, and to move on.

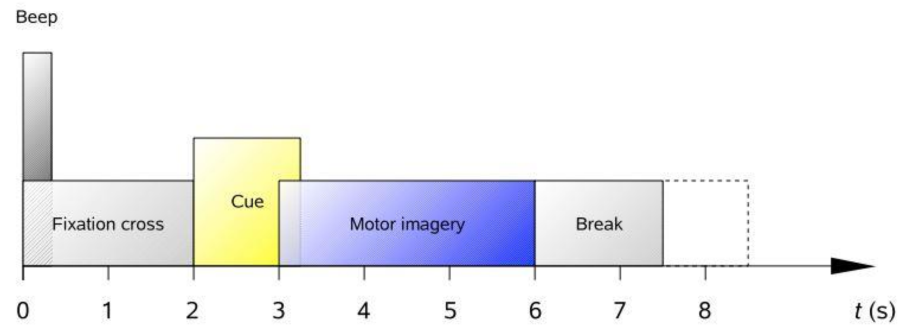
b. BCI Competition 2008 – Graz data set A [18]

This is the EEG dataset that we choose to work with. This data set consists of EEG data from 9 subjects. The data is divided to 4 classes that presents 4 motor imagery tasks, 4 different movement Imagination.

Experimental flow:
The subjects were sitting in a comfortable armchair in front of a computer screen. At the beginning of a trial (t = 0s), a fixation cross-appeared on the black screen. In addition, a short acoustic warning tone was presented. After two seconds (t = 2s), a cue in the form of an arrow pointing either to the left, right, down or up corresponding to one of the four classes, as describe in the table below and in the article.

Class 1 - Left Hand 	**Class 2 - Right Hand** 
Class 3 - Both Feet 	**Class 4 – Tongue** 

The arrow was appeared and stayed on the screen for 1.25s, in this time the subject perform the desired motor imagery task. No feedback was provided. The subjects were ask to carry out the motor imagery task until the fixation cross disappeared from the screen at t = 6s. A short break followed where the screen was black again. The paradigm time flow:



The data sets were stored in 9 files that represent 9 subjects.

Each file contain:

- 288 trials: 12_{trials} x 4_{classes} = 48_{trials} x 6_{runs} = 288_{trials}

The files were stored in General Data Format (GDF). In order to load the data sets via MATLAB we used BioSig toolbox[3]. And used the command:

```
[S, H] = sload('A01T.gdf');
```

S parameter – Present a matrix $S_{m\times n}$ which contains the signals. Number of columns n = 25 and m changing from file to file.

$$\begin{array}{c} \begin{array}{ccccc} & \text{EEG} & & \text{EOG} & \end{array} \\ \begin{pmatrix} S_{11} & \cdots & S_{1(n-3)} & \cdots & S_{1n} \\ \vdots & \ddots & & \vdots & \\ S_{m1} & \cdots & S_{m(n-3)} & \cdots & S_{mn} \end{pmatrix} \end{array}$$

The first 22 columns are EEG (each column different channel) and the last three are EOG signals.

The –H- parameter - present header (Meta data) about the signals with well-defined structure. The header structure contains event information that describes the structure of the data over time. The fields type and event types description are mentioned in the article.

The recording description and the experimental paradigm can be found in the article as well.

[3] http://biosig.sourceforge.net/

In order to use the function ‘sload’ on MATLAB we need to run one time the following code:

```
BIOSIG runs on Matlab and Octave.
This is a script installing all components in an automatically.

1) extract the files and
2) save the BIOSIG files in <your_directory>
3) start matlab
       cd <your_directory>
       biosig_installer

4) For a permanent installation, save the default path with
     PATH2RC or
     PATHTOOL and click on the "SAVE" button.
5) For removing the toolbox
    remove the path to
       HOME/tsa
       HOME/NaN
       HOME/BIOSIG/ and all its subdirectories
```

c. Data sets retrieval Process

The data sets were collected from a well-defined structure.

As a start, we pull out the data from the given files and prepare the following data structure:

2-dimentional array:	2,592 x 6,908
Data description:	$2592 = 288_{trials} \times 9_{subject}$
	$6908 = 314_{duration} \times 22_{channels}$

Each row in the 2-dimentional array is a vector of one trial with all the relevant data on the trial (all the channels combined).

VI. Our Approach

1. The Process

a. Create EEG data set

Creating EEG data set was considered. Few tools were examinee to record EEG brain waves, Emotiv Epoch [1] and openBCI [2] are the leading candidates due to their availability and price

Because of the hard work needed to record such experiments and the lack of students for such experiments we have decided to neglect it and to find well-defined EEG dataset.

b. Searching for EEG data set

EEG well-defined data set is highly precious commodity. It is a difficult task to find one. A few data set were examinee and tested. There are 2 data sets that were the last candidates and eventually one was elected [18].

c. Machine learning developers tools

Learning developers' tools for machine learning against ANN, CNN: Learning MATLAB, learning MatConvNet.

d. Pool out the data set

After selecting *Graz data set A* [18] as our EEG dataset we needed to pool it out and create data structure for our needs. Data file description and the metadata are explained in the document [18].

All data sets are stored in the General Data Format.

The process:

1. Save data and metadata to Excel file.
   MATLAB code, See IX.A2.

   Do this process to all 9 GDF files.

2. Create new *.csv file from the given excel file, s_t_1.csv. Take only the EEG data from the s_t_1.csv file by the given index type and position.
   Create excel files with Java code, See IX.A3.

   Output: 9 csv files which contain the EEG dataset – a table with 25 columns (22 EEG + 3 EOG) and 90432 rows (314_{duration} x 288_{trials})

3. Save the 9 csv files in *.mat format.
   MATLAB code, See IX.A4.
4. Adjust the dataset.
   Create one file, MATLAB code, See IX.A5.

5. Remove EOG data (3 columns).
   MATLAB code, See IX.A6.

6. Create array with the right structure,
   $2592_{288\text{(number of trials) x 9 (subjects)}}$ rows, $6908_{314\text{ (duration) x 22 (eeg channels)}}$.
   Each row is a full 1 trial with 314 duration of 22 eeg channels.
   MATLAB code, See IX.A7.

Do the same process 1 – 6, also to the test dataset.

7. Save train labels
   MATLAB code, See IX.A8.

   The test labels were given.

e. Adjust the algorithm

Take working neural network system and adjust it to our needs, in order to work with the prepared in advanced EEG dataset.

f. Simulations

Start making simulations and observe the results.

g. Improvements and analysis

Improve classification accuracy, by known techniques and methods. In addition, continue to make simulations accordingly.
A technique that was taken into account to improve accuracy was to use different machine learning algorithms - CNN, neglected system have been developed and adapted to our need. Poor results have been reached, which could be improved, but discarded to lack of computation power.

2. Improvements and analysis

As we saw earlier at the “Techniques to improve accuracy & performance” section III.27, there are several techniques that we could use in order to enhance our model, we applied some of them.
We looked at our data and our model, in order to improve classification accuracy we realized that we should act as follows:

Weight initialization – initiate the system

As we can see at the simulations type B, D & E when we choose one initialization approach to the other the classification accuracy is changed.
We show that if we take pre-learned weights on a completely different topic from our system such as MNIST problem, and initial our system with those values we can improve the results substantially.
See section VI.2.a and simulations type B, D & E at VI.4.

Extending the data set

Machine learning literature proves the fact that larger training data sets leads to more generalization and avoid over fitting. The more data the training model see in the training process it will give more accurate prediction on the testing dataset. We do not have an option to add a native data to the training set, therefore we have considered techniques to manipulate the existing data and to augment it. See Data Augmentation section VI.2.b.

During the process of training our neural network, we got into a problem so called overfitting, as we can see in the simulations section VI.4. We have noticed that the model memorize the training dataset and not learning to generalize the data. We didn’t have the ability to make a better features selection. In order to solve overfitting problem we used regularization technique such as weight decay. We have also augment the size of the training dataset, which was done by artificial augmentation of the existing data as mentioned earlier.

Simulations with data augmentation can be found under simulations type C & D.

Tune the algorithm

Our machine learning algorithm have a set of free parameters that can be tuned as mentioned at the system description section V.1. As we can see at the simulations section (from VI.4) – all types, we tune the system parameters in order to observe if the classification accuracy is improved.

a. Initiate the system - weight initialization

As we have already seen at "Techniques to improve accuracy & performance" (III.27) section, one of the techniques to improve model accuracy is to initialize the Artificial Neural Network with pre learned weights and biases based on a completely different topic training.
We can choose to initiate the weights and biases with random variables, but there are better ways to initiate weights and biases and help the model to get better classification accuracy and to learn faster.

The other topic that we choose is the MNIST problem [28]. The original problem has 784 input neurons at the ANN, the images are greyscale and 28 by 28 pixels in size (28x28 = 784).
Our original EEG wave size is $6908_{314\text{(duration) x 22(EEG channels).}}$
In order to be able transfer weights from one ANN to another, both problems must have the same sizes and dimensions. This similarity can be easily done with hidden layers, but have to be carefully achieved by a manipulation on the examples sizes.
Different sizes can give different accuracy. We have tried several sizes as we can see in the simulations section (from VI.4):

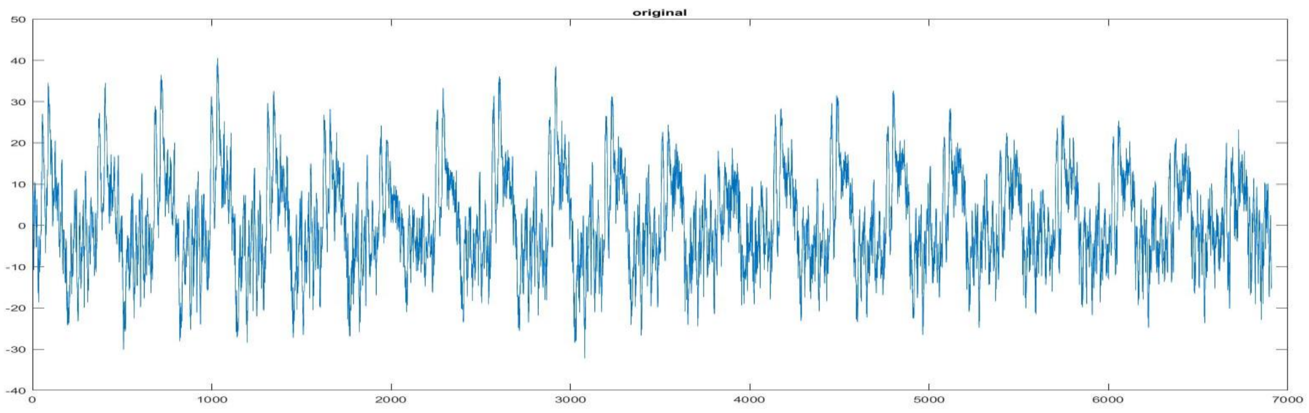
EEG size	**MNIST size**	**size**
22 x 314	---------------------------------------	6908
24 x 288	72 x 96	6912
24 x 294	84 x 84	7056
25 x 256	80 x 80	6400
21 x 210	63 x 70	4410
4 x 121	22 x 22	484
22 x 315	77 x 90	6930

b. Data Augmentation

In order to improve the results on the test data, one technique is to expand the training data. In the following section we give description of some of the techniques we have applied for this purpose.

Original

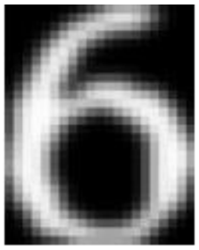
EEG signal, 22_{channels} x 314_{duration} = 6908_{values}

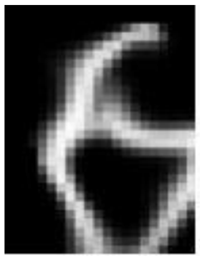


Elastic deformation

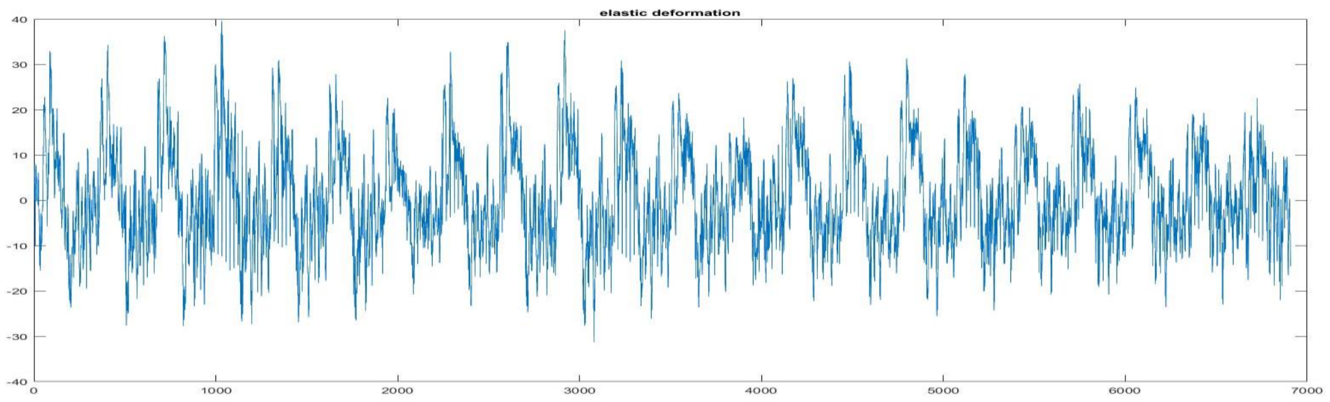
Augmentation method - elastic deformation, this technique described in this paper [23].

The process:
We wanted to augment the EEG dataset. In order to do so, we needed to create for each image an **Elastic Deformations** image respectively as mentioned in the paper. For MNIST dataset (size: 28 x 28 px)

Before: 

After: 

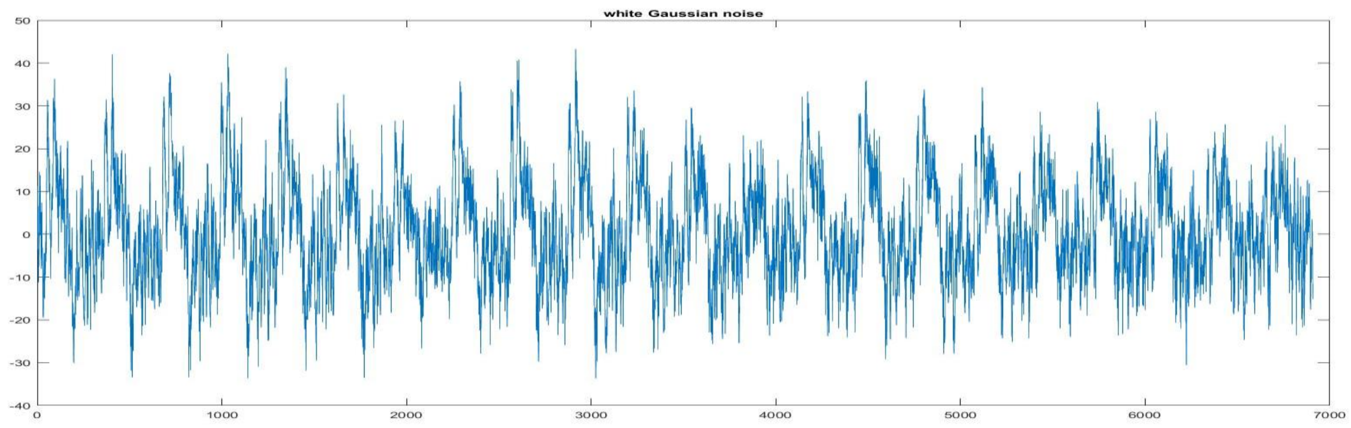
EEG dataset (1 x 6908 – 22 x 314), after elastic deformation:



To achieve elastic deformation we used MATLAB, code can be found at IX.A1.

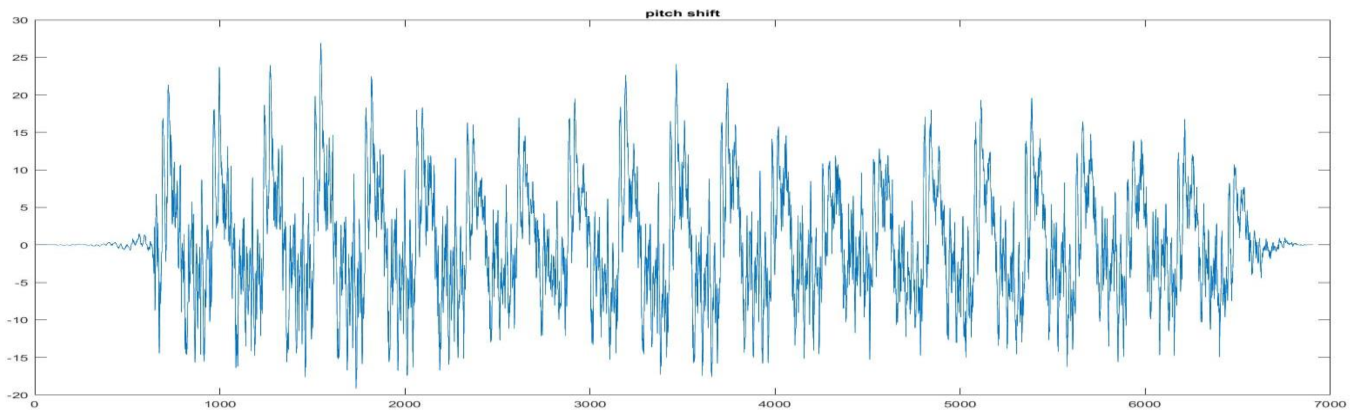
White Gaussian noise

Add white Gaussian noise to signal.
EEG dataset (1 x 6908 – 22 x 314), after adding white Gaussian noise:



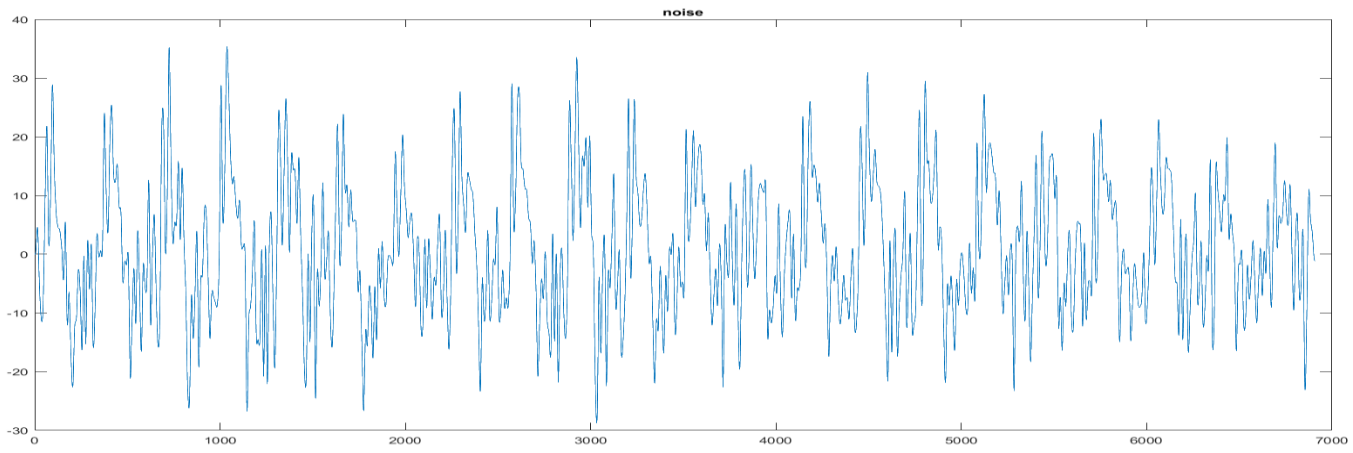
Pitch shift [24]

Pitch Shifting is a way to change the pitch of a signal without changing its length.



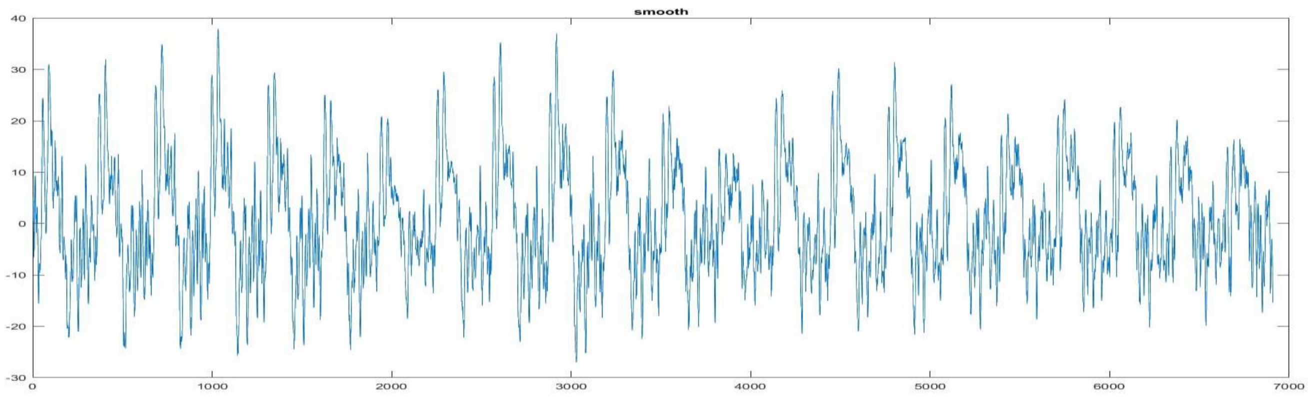
Lowpass filter

Filter that allows signals below a cutoff frequency and attenuates signals above the cutoff frequency.



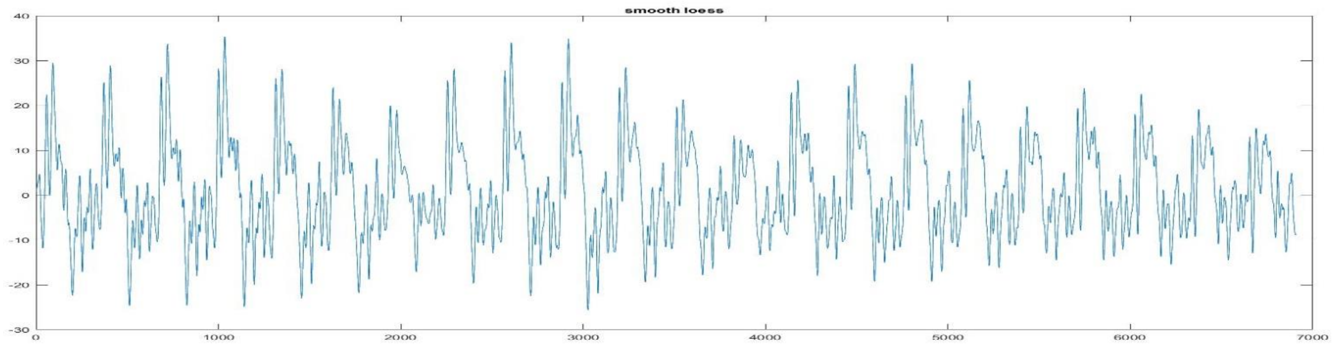
Smooth moving

Moving average (default). A lowpass filter with filter coefficients equal to the reciprocal of the span.



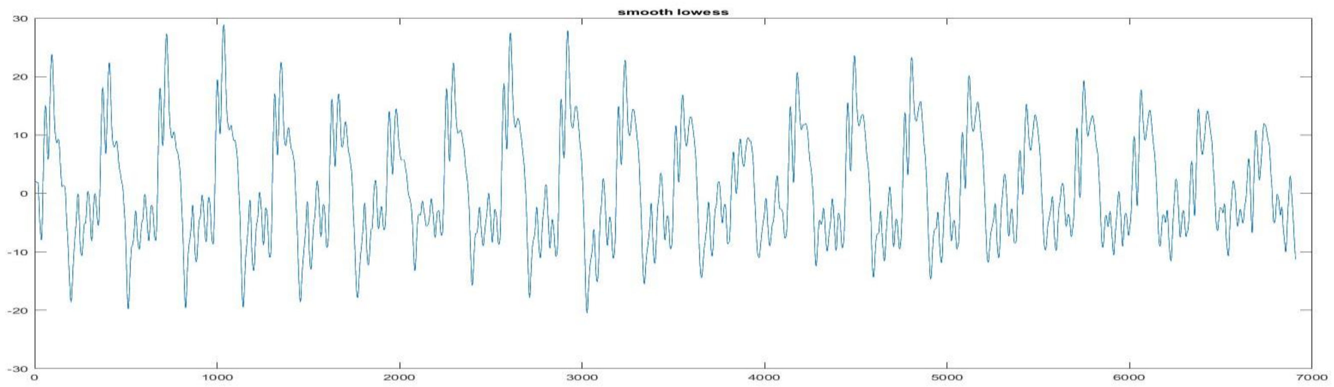
Smooth loess 0.005

Local regression using weighted linear least squares and a 2nd degree polynomial model



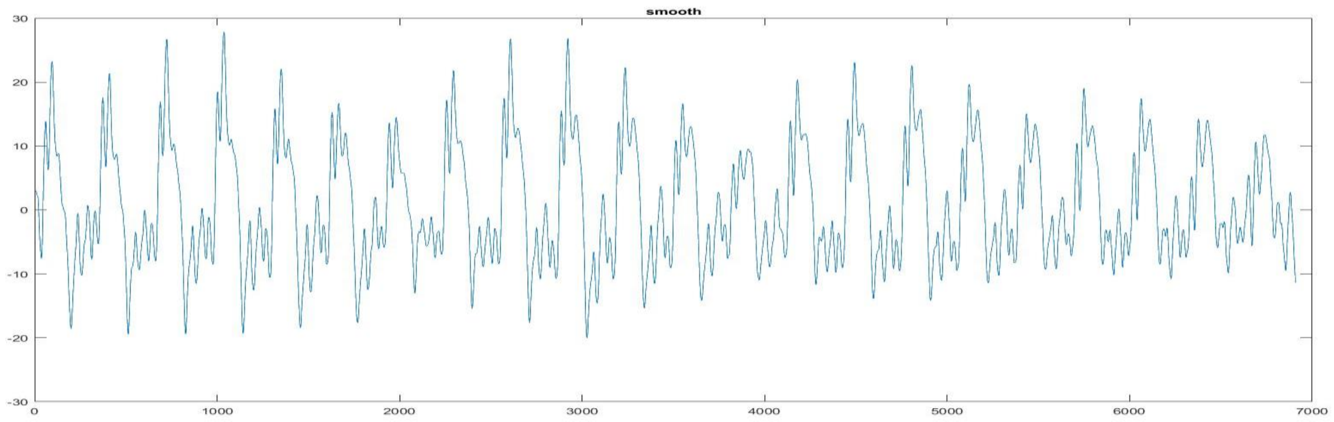
Smooth lowess 0.005

Local regression using weighted linear least squares and a 2nd degree polynomial model



Smooth rlowess 0.005

A robust version of `'lowess'` *that assigns lower weight to outliers in the regression. The method assigns zero weight to data outside six mean absolute deviations.*



3. Types

a. Subject dependent vs Subject independent

In the case of subject dependent we test the system with unseen samples from the same subjects we have used to train it while in subject independent the system is tested using unseen samples from new subject that was introduced to the system in the training process. It is clear that Subject independent is more difficult than subject dependent, therefore anticipated result for MSE should be less.

The given datasets were taken from 9 subject. We have divided the datasets to train dataset and test dataset. The article ‘BCI Competition 2008 – Graz data set A’ [18] not mentioned if it’s the same subjects, which the test and the train datasets were recorded from. It means that we cannot know if the simulations is for subject depended or subject independent.

b. Channel group size / channel reduction

We have used a dataset with 22 electrodes, which were used to record the EEG signals. 22 electrodes are translated to 22 EEG channels. One question we tried to answer in these experiments is the effect of the size of the group of channels on the system’s accuracy. We can see at the next section that there are channels which are more discriminative than others and the impact of less channels in the group in terms of accuracy VI.4.b.

4. Simulations

Type A. 22 channels.

Simulations without augmentation methods nor given weights. As we can see, at almost all simulations the results on the training set are much better than the results on the testing set.

#	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
1	2592	2592	none	none	50 50	Bipolar	0.03	80	enable	Train: 0.91667 Test: 0.30054
2	2592	2592	none	none	60	Bipolar	0.03	80	enable	Train: 0.75424 Test: 0.37269
3	2592	2592	none	none	50	Unipolar	0.03	80	enable	Train: 0.49846 Test: 0.33102
4	2592	2592	none	none	50	Bipolar	0.03	80	disable	Train: 0.83102 Test: 0.375
5	2592	2592	none	none	40	Bipolar	0.03	80	enable	Train: 0.78009 Test: 0.37076
6	2592	2592	none	none	80	Bipolar	0.03	80	enable	Train: 0.6983 Test: 0.35455
7	2592	2592	none	none	50	Bipolar	0.04	80	enable	Train: 0.88426 Test: 0.36188
8	**2592**	**2592**	**none**	**none**	**50**	**Bipolar**	**0.03**	**80**	**enable**	**Train: 0.80015 Test: 0.37963**
9	2592	2592	none	none	100 50	Bipolar	0.03	80	enable	Train: 0.84722 Test: 0.32215
10	2592	2592	none	none	100 100	Bipolar	0.03	80	enable	Train: 0.64159 Test: 0.3125
11	2592	2592	none	none	100	Bipolar	0.03	100	enable	Train: 0.64738 Test: 0.36343
12	2592	2592	none	none	100	Bipolar	0.05	100	enable	Train: 0.60494 Test: 0.33912
13	2592	2592	none	none	100	Bipolar	0.06	100	enable	Train: 0.56289 Test: 0.32832
14	2592	2592	none	none	100	Bipolar	0.01	100	enable	Train: 0.35841 Test: 0.29707
15	2592	2592	none	none	100	Bipolar	0.04	100	enable	Train: 0.67631 Test: 0.36188

Type B. Given weights, 22 channels.

Simulations with given weights – initiate the system with given values. As we can see, the test result were highly improved. But still at almost all simulations the results on the training set are much better than the results on the testing set.

#	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
1	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.01	80	enable	Train: 0.87886 Test: 0.48534
2	**2592**	**2592**	**none**	**EEG: 24 x 288 Given: 72 x 96**	**100**	**Bipolar**	**0.008**	**100**	**enable**	**Train: 0.84375 Test: 0.48958**
3	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.005	100	enable	Train: 0.77932 Test: 0.48573
4	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.007	100	enable	Train: 0.78897 Test: 0.47261
5	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100 50	Bipolar	0.008	100	enable	Train: 0.90548 Test: 0.46142
6	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100 100	Bipolar	0.008	100	enable	Train: 0.9267 Test: 0.48032
7	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.008	100	disable	Train: 0.85224 Test: 0.48032
8	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100	Unipolar	0.008	100	enable	Train: 0.45255 Test: 0.39815
9	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.03	100	enable	Train: 0.9865 Test: 0.46412
10	2592	2592	none	EEG: 24 x 294 Given: 84 x 84	100	Bipolar	0.008	100	enable	Train: 0.85378 Test: 0.4865
11	2592	2592	none	EEG: 24 x 294 Given: 84 x 84	100	Bipolar	0.009	100	enable	Train: 0.87269 Test: 0.48148
12	2592	2592	none	EEG: 25 x 256 Given: 80 x 80	100	Bipolar	0.008	100	enable	Train: 0.82523 Test: 0.47377
13	2592	2592	none	EEG: 25 x 256 Given: 80 x 80	100	Bipolar	0.005	100	enable	Train: 0.73071 Test: 0.47299
14	2592	2592	none	EEG: 21 x 210 Given: 63 x 70	100	Bipolar	0.008	100	enable	Train: 0.75116 Test: 0.47647
15	2592	2592	none	EEG: 4 x 121 Given: 22 x 22	100	Bipolar	0.008	100	enable	Train: 0.43056 Test: 0.39082

Type C. Augmentation methods, 22 channels.

Simulations with data augmentation. As we can see, the test result were improved from **Type A**. But still at almost all simulations the results on the training set are much better than the results on the testing set.

#	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
1	**5184**	**2592**	**elastic deformation**	**none**	**50**	**Bipolar**	**0.03**	**80**	**enable**	**Train: 0.92091 Test: 0.39468**
2	5184	2592	White Gaussian noise	none	50	Bipolar	0.03	80	enable	Train: 0.91281 Test: 0.3777
3	5184	2592	pitch shift	none	50	Bipolar	0.03	80	enable	Train: 0.75154 Test: 0.37346
4	5184	2592	lowpass filter	none	50	Bipolar	0.03	80	enable	Train: 0.8206 Test: 0.39005

Type D. Given weights, augmentation methods, 22 channels.

Simulations with data augmentation and given weights. As we can see, the test result were improved. But still at almost all simulations the results on the training set are much better than the results on the testing set.

#	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
1	5184	2592	elastic deformation	EEG: 4 x 121 Given: 22 x 22	100	Bipolar	0.008	100	enable	Train: 0.46393 Test: 0.41705
2	5184	2592	elastic deformation	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.008	100	enable	Train: 0.86941 Test: 0.48418
3	5184	2592	white Gaussian noise	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.008	120	enable	Train: 0.90548 Test: 0.48765
4	5184	2592	white Gaussian noise	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.007	120	enable	Train: 0.85552 Test: 0.49074
5	5184	2592	white Gaussian noise	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.006	120	enable	Train: 0.84008 Test: 0.47955
6	7776	2592	white Gaussian noise & elastic deformation	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.008	50	enable	Train: 0.86523 Test: 0.49653
7	5184	2592	smoothing	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.008	65	enable	Train: 0.78665 Test: 0.50424
8	**7776**	**2592**	**smoothing -** `smooth(x, 'moving')` & **smoothing -** `smooth(x, 0.005,'lowess')`	**EEG: 22 x 315** **Given: 77 x 90**	**100**	**Bipolar**	**0.008**	**50**	**enable**	**Train: 0.7293** **Test: 0.51543**

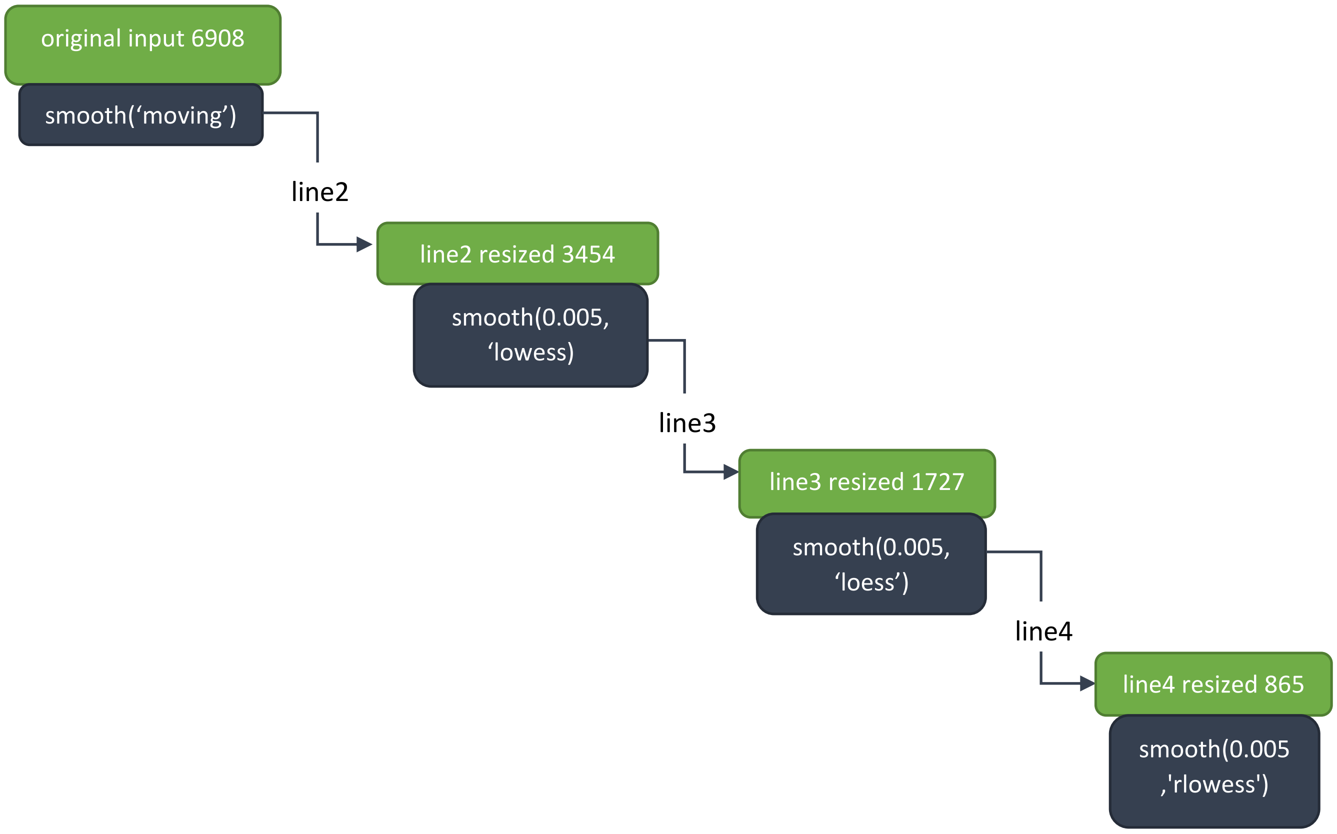
Type E: Run concatenate, 22 channels.

Simulations with given weights and instead of using data augmentation we used concatenation technique as described below. As we can see, the test result were improved. But still the results on the training set are much better than the results on the testing set.

#	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
1	2592 x 12954	2592 x 12954	none	Given: 102 x 127	100	Bipolar	0.008	100	enable	Train: 0.80556 Test: 0.53241
2	**2592 x 12954**	**2592 x 12954**	**none**	**Given: 102 x 127**	**100**	**Bipolar**	**0.009**	**80**	**enable**	**Train: 0.83796 Test: 0.54321**

How the samples were created?

The original data contain 2592 samples which each sample is size 6908. Each sample underwent the following transformation:



When we concatenate all the transformation results we get a sample with size of: 12954 = 6908 + 3454 + 1727 + 865

The neural network first layer, input layer, will contain 12954 neurons.

a. Result summery

Type	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
A	2592	2592	none	none	50	Bipolar	0.03	80	enable	Train: 0.80015 Test: 0.37963
B	2592	2592	none	EEG: 24 x 288 Given: 72 x 96	100	Bipolar	0.008	100	enable	Train: 0.84375 Test: 0.48958
C	5184	2592	elastic deformation	none	50	Bipolar	0.03	80	enable	Train: 0.92091 Test: 0.39468
D	7776	2592	smoothing - `smooth(x,'moving')` & smoothing - `smooth(x,0.005,'lowess')`	EEG: 22 x 315 Given: 77 x 90	100	Bipolar	0.008		enable	Train: 0.7293 Test: 0.51543
E	**2592 x 12954**	**2592 x 12954**	**none**	**Given: 102 x 127**	**100**	**Bipolar**	**0.009**	**80**	**enable**	**Train: 0.83796 Test: 0.54321**

b. Group of channels

Run by channel

Each channels separately.

Channel	Train MSE	Test MSE
1	0.42593	0.3125
2	0.43364	0.29784
3	0.45602	0.30324
4	0.43711	0.31019
5	0.4456	0.31713
6	0.42323	0.2963
7	0.40046	0.28202
8	0.43056	0.28395
9	0.4213	0.34144
10	0.45216	0.33526
11	0.4213	0.31983
12	0.42323	0.29282
13	0.42284	0.27662
14	0.4159	0.31327
15	0.44252	0.32215
16	0.45293	0.33565
17	0.43133	0.32369
18	0.4429	0.30787
19	0.45988	0.32793
20	0.46103	0.33025
21	0.4429	0.33565
22	0.49614	0.34954

best 14 channels

Channel	Train MSE	Test MSE
1	0.42593	0.3125
4	0.43711	0.31019
5	0.4456	0.31713
9	0.4213	0.34144
10	0.45216	0.33526
11	0.4213	0.31983
14	0.4159	0.31327
15	0.44252	0.32215
16	0.45293	0.33565
17	0.43133	0.32369
19	0.45988	0.32793
20	0.46103	0.33025
21	0.4429	0.33565
22	0.49614	0.34954

Best 7 channels

Channel	Train MSE	Test MSE
9	0.4213	0.34144
10	0.45216	0.33526
16	0.45293	0.33565
19	0.45988	0.32793
20	0.46103	0.33025
21	0.4429	0.33565
22	0.49614	0.34954

c. Run with channel reduction

14 group channel

314$_{\text{(duration)}}$ x 14$_{\text{(number of channels)}}$ = 4396

#	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
1	2592	2592	none	none	30	Bipolar	0.03	80	Enable	Train: 0.74807 Test: 0.37423
2	**2592**	**2592**	**none**	**EEG: 16 x 275 Given: 55 x 88**	**100**	**Bipolar**	**0.008**	**100**	**Enable**	**Train: 0.76929 Test: 0.47454**

7 group channel

314$_{\text{(duration)}}$ x 7$_{\text{(number of channels)}}$ = 2198

#	Train size	Test size	Augment methods	Given weights	Hidden Layers	Bipolar / Unipolar	Learning Rate	Epochs	Resilient	Results MSE
1	2592	2592	none	none	30	Bipolar	0.03	80	Enable	Train: 0.65162 Test: 0.36767
2	**2592**	**2592**	**none**	**EEG: 11 x 200 Given: 40 x 55**	**100**	**Bipolar**	**0.008**	**100**	**Enable**	**Train: 0.63773 Test: 0.45023**

VII. Work Contribution

1. Improve results

The data sets were taken from BCI Competition IV [18]. The result of the competition can be found on the reference.

Their performance measure is kappa value. Our result presented in MSE.
One of the goals of this work is to get better results and to find different techniques to do so. Our result can be found in the result summery section VIII.
Processing of EEG signals are been very popular in these days, it used in a wide variety of applications such as BCI applications, motor imagery classification, mental tasks, drug effects, sleep disorders. Improve accuracy results improves the applications effects.

2. Reduce channels

As discussed earlier in this paper, we can see that we can achieve similar classification accuracy when we have less EEG channels. In our case, using different experiments we have found the important channels, which are more discriminant than others are. This may affect the recording process in order to focus on these channels and eliminate others. Large number of EEG channels acquired can affect algorithm performance, efficient channel selection is needed. The needs of reducing the computational complexity of any processing task performed on EEG signals by selecting the relevant channels and extract the features is highly important. Reducing the amount of overfitting that may arise by the irrelevant channels is highly important as well.

VIII. Summery

Randomly selecting one group of four, to predict which imaginary move occur will stamp probability of 0.25. The first experiments with Artificial Neural Network trained by backpropagation yield accuracy of 0.29707_{MSE} on the test dataset which is not much better than guessing. As long as we tried to tune the algorithm, the best result we got is 0.37963_{MSE} and 0.80015_{MSE}, train and test datasets respectively. These results were the best we could achieve using the proposed technique. To improve the results we tried to find new techniques. A known technique is to initial the system with values which were taken from other pre trained systems such as the ANN trained on the MNIST data set. By this, we get the benefit of a very large training set. The Neural Network architecture of the two problems need to have the same number of input neurons, hence the dataset should be prepared in advanced. When run the algorithm with the given set of values we got $0.39082_{MSE(test)}$ and $0.43056_{MSE(train)}$. After tuning the algorithm as described in the simulations section (from VI.4) we got $0.48958_{MSE(test)}$ and $0.84375_{MSE(train)}$. Says that, as long as the training dataset is bigger and the algorithm learn more data the results can improve dramatically. Our EEG dataset is considered as a small one, therefore several techniques that were applied to augment the dataset: elastic deformation; White Gaussian noise; Pitch shift and several Smoothing techniques (We tried more augmentation techniques that are not mentioned). Each of which can get different results. With the run of the augmentation technique with the given values we got $0.51543_{MSE(test)}$ and $0.7293_{MSE(train)}$. As we can see clearly, those techniques improve the results. Instead of augment the dataset "vertically" we can augment it "horizontally" – take the originally dataset and add to each wave a tail, concatenate the waves which created with the original one, this technique yields results of **$0.54321_{MSE(test)}$ and $0.83796_{MSE(train)}$.**

We can see in the simulations section that when trying to reduce the number of channel, in order to test if we can use tools that have less channels, the result not changed much, and there is still advantage to more channels. For 22 channels with given weights only, the best we got is $0.48958_{MSE(test)}$ and $0.84375_{MSE(train)}$; for 14 channels with given weights only, the best we got is $0.47454_{MSE(test)}$ and $0.76929_{MSE(train)}$; for 7 channels with given weights only, the best we got is $0.45023_{MSE(test)}$ and $0.63773_{MSE(train)}$;

Research in the field of EEG brain waves with the respect of machine learning can achieve and be useful for broad aspects in the medicine world. Get better understanding of diseases and predictions, help disabled with brain computer interface. Making a science contribution for EEG research.

IX. Appendix - A

A1 – Transform an elastic deformation on a dataset, MATLAB.

```
load('TrainArr.mat');
load('TrainLables.mat');

TrainArray_ed = [];


for i=1:size(Arr,1) %for each sample
    img = reshape(Arr(i,:), [22 314]);  % get one EEG with 22 channels

    dx = -1+2*rand(size(img));
    dy = -1+2*rand(size(img));

    sig=4;
    alpha=60;
    H=fspecial('gauss',[7 7], sig);
    fdx=imfilter(dx,H);
    fdy=imfilter(dy,H);
    n=sum((fdx(:).^2+fdy(:).^2));
    fdx=alpha*fdx./n;
    fdy=alpha*fdy./n;

    [y x]=ndgrid(1:size(img,1),1:size(img,2));

    ed = griddata(x-fdx,y-fdy,double(img),x,y);
    ed(isnan(ed))=0;

    ed = reshape(ed, [1 6908]);
    TrainArray_ed = [TrainArray_ed; ed];
    display([i]);
end
```

A2 - Save data and metadata to Excel file

```
%need to run biosig_installer first. see document
biosig_installer

fileName = ['A01T.gdf'];
[s_t, h_t] = sload(fileName, 0, 'OVERFLOWDETECTION:OFF');
dlmwrite(['C:\Desktop\excel\s_t_1.csv'], s_t, 'delimiter', ',', 'precision', 15);
h_pos = h_t.EVENT.POS;
h_typ = h_t.EVENT.TYP;
h_dur = h_t.EVENT.DUR;
dlmwrite(['C:\Desktop\excel\h_pos_t_1.csv'], h_pos, 'delimiter', ',', 'precision', 15);
dlmwrite(['C:\Desktop\excel\h_typ_t_1.csv'], h_typ, 'delimiter', ',', 'precision', 15);
dlmwrite(['C:\Desktop\excel\h_dur_t_1.csv'], h_dur, 'delimiter', ',', 'precision', 15);
```

A3 - Create excel files with Java code

```
package eeg.data.excel;

import java.io.FileReader;
import java.io.FileWriter;
import java.util.ArrayList;
import java.util.List;
import au.com.bytecode.opencsv.CSVReader;
import au.com.bytecode.opencsv.CSVWriter;

/**
 * @author LioR SoLoMoN
 *
 */
public class Excel {
      static final String type = "t"; //e
      static final int NUM_OF_FILES = 9;
      static final int NUM_OF_DURATION = 313;
      static final int NUM_OF_TRIALS = 288;

      public static void main(String[] args) {
            System.out.println("START Program");

            for (int i = 1; i <= NUM_OF_FILES; i++) {
                  createFile(String.valueOf(i));
            }

            System.out.println("END Program");
      }
	public static void createFile(String fileNumber) {
		System.out.println("START file: " + fileNumber);

		String userName = System.getProperty("user.name");
		String csvFileRead = "C:\\Users\\" + userName + "\\Desktop\\files\\s_" +
			type +"_" + fileNumber + ".csv";
		String csvFileReadIndex = "C:\\Users\\" + userName +
			"\\Desktop\\files\\indexes_" + fileNumber + ".csv";
		String csvFileWrite = "C:\\Users\\" + userName + "\\Desktop\\files\\" +
			type + "\\file_" + fileNumber + ".csv";
		CSVReader reader = null;
		CSVReader readerIndexes = null;
		CSVWriter writer= null;

		try {
			reader = new CSVReader(new FileReader(csvFileRead));
			readerIndexes = new CSVReader(new FileReader(csvFileReadIndex));

			int[] indexes = new int[NUM_OF_TRIALS];
			String[] line;
			int k = 0;
			while((line = readerIndexes.readNext()) != null) {
				indexes[k] = Integer.valueOf(line[0]);
				k++;
			}
			System.out.println("1");
			List<String[]> readAll = reader.readAll();
			List<String[]> newData = new ArrayList<String[]>();

			for (int i = 0; i < NUM_OF_TRIALS; i++) {
				System.out.println("i: " + i);
				try {
					for(int ind = indexes[i]; ind <= (NUM_OF_DURATION +
						indexes[i]); ind++) {
						System.out.println("ind: " + ind);
						newData.add(readAll.get(ind-1));
					}
				} catch (Exception e) {
					System.out.println("2");
					e.printStackTrace();
					System.exit(0);
				}
			}
			writer = new CSVWriter(new FileWriter(csvFileWrite));
			for(String[] row : newData) {
				writer.writeNext(row);
			}
			writer.close();
		} catch (Exception e) {
			System.out.println("3");
			e.printStackTrace();
			System.exit(0);
		}
		System.out.println("END file: " + fileNumber);
	}
}
```

A4 - Save the 9 csv files in *.mat format

```
for i=1:9
    fileName = ['file_' num2str(i) '.csv'];
    TrainArr = csvread(fileName)
    save(['TrainArr\TrainArr_' num2str(i) '.mat'], 'TrainArr');
end
```

A5 - Adjust the dataset

```
Arr = [];
for i=1:9
    fileName = ['TrainArr_' num2str(i) '.mat'];
    load(fileName);
    Arr = [ Arr; TrainArr ]
end
save('TrainArrWithEOG.mat','Arr');
```

A6 - Remove EOG data

```
load('TrainArrWithEOG.mat');
TrainArr = Arr(:,1:end-3);
save('TrainArr.mat','TrainArr');
```

A7 - Create array with the right structure

```
Arr = [];
load('TrainArr.mat');

for i=1:2592
    from_line = 1 + 314*(i-1);
    to_line = 314 * i;
    line = reshape(TrainArr(from_line:to_line,:),[1,6908]);
    Arr = [ Arr; line ];
end
save('TrainArr_final.mat','Arr');
```

A8 – Save Train Labels

```
%install the plugin on start
biosig_installer
%Train labels
for i=1:9
    fileName = ['A0' num2str(i) 'T.gdf'];
    [s_t, h_t] = sload(fileName);
    TrainLables = h_t.Classlabel;
    saveFileName = ['TrainLables\TrainLables_0' num2str(i) 'T.mat'];
    save(saveFileName,'TrainLables');
    fprintf(['done file ' num2str(i)]);
end
```

X. Bibliography

[1] Emotiv - Brain Computer Interface: http://www.emotiv.com

[2] Openbci - Brain Computer Interface http://www.openbci.com/

[3] NeuroSky - Brainwave Sensors: http://www.neurosky.com

[4] E. Alpaydin, 2010, Introduction to Machine Learning. Second Edition. MIT Press, pp. 233-278

[5] An, X., Guo, X., He, L., Kuang, D., & Zhao, Y. (2014). A Deep Learning Method for Classification of EEG Data Based on Motor Imagery. *ICIC*.

[6] D. Kriesel. A Brief Introduction to Neural Networks, pp. 3-127. 2005

[7] K.R Muller, M. Tangermann, G. Dornhege, M. Krauledat. G. Curio, B. Blankertz. Machine Learning for Real-Time Single-Trial EEG Analysis: From Brain-Computer Interfacing to Mental State Monitoring.

[8] S.Barua, S.Begum, A Review on machine Learning Algorithms in Handling EEG Artifacts. May 2014.

[9] Podgorelec, V. (2012). Analyzing EEG Signals with Machine Learning for Diagnosing Alzheimer's Disease. *Elektronika Ir Elektrotechnika, 18*(8), 61-64.

[10] Subasi, A.; Kiymik, M.K.; Alkan, A.; Koklukaya, E. Neural Network Classification of EEG Signals by Using AR with MLE Preprocessing for Epileptic Seizure Detection. *Math. Comput. Appl.* 2005, *10*, 57-70.

[11] Deep Learning Summer School, http://videolectures.net/deeplearning2015_montreal/

[12] Michael A.Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015.

[13] S. Baillet, J. C. Mosher and R. M. Leahy, "Electromagnetic brain mapping," in *IEEE Signal Processing Magazine*, vol. 18, no. 6, pp. 14-30, Nov 2001.

[14] Brain-computer interfaces for communication and control Wolpaw J.R., Birbaumer N., McFarland D.J., Pfurtscheller G., Vaughan T.M. (2002) Clinical Neurophysiology, 113 (6) , pp. 767-791.

[15] Gomez-Gil, J.; San-Jose-Gonzalez, I.; Nicolas-Alonso, L.F.; Alonso-Garcia, S. Steering a Tractor by Means of an EMG-Based Human-Machine Interface. *Sensors* 2011, *11*, 7110-7126.

[16] Michael Bensch, Ahmed A. Karim, Jürgen Mellinger, et al., “Nessi: An EEG-Controlled Web Browser for Severely Paralyzed Patients,” Computational Intelligence and Neuroscience, vol. 2007, Article ID 71863, 5 pages, 2007. doi:10.1155/2007/71863.

[17] Ulrich Hoffmann, Jean-Marc Vesin, Karin Diserens, and Touradj Ebrahimi. An efficient P300-based brain-compuer interface for disabled subjects. Journal of Neuroscience Methods, 2007. submitted.

[18] EEG dataset - C. Brunner1 , R. Leeb1 , G. R. M¨uller-Putz1 , A. Schl¨ogl2 , and G. Pfurtscheller. BCI Competition 2008 – Graz data set A.
http://www.bbci.de/competition/iv/desc_2a.pdf

[19] EEG dataset - http://www.bbci.de/competition/iv/#dataset2a

[20] Neurons - http://www.human-memory.net/brain_neurons.html

[21] Fabien Lotte. A Tutorial on EEG Signal Processing Techniques for Mental State Recognition in Brain-Computer Interfaces. Eduardo Reck Miranda; Julien Castet. Guide to Brain-Computer Music Interfacing, Springer, 2014.

[22] Boundless. “Neurons.” *Boundless Biology*. Boundless, 26 May. 2016. Retrieved 25 Dec. 2016 from https://www.boundless.com/biology/textbooks/boundless-biology-textbook/the-nervous-system-35/neurons-and-glial-cells-199/neurons-759-11992/

[23] P. Y. Simard, D. Steinkraus and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003, pp. 958-963.

[24] Grill, T., & Schlüter, J. (2015). Exploring Data Augmentation for Improved Singing Voice Detection with Neural Networks. ISMIR.

[25] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines, pp. 807-814. 2010.

[26] EEG dataset - An efficient P300-based brain-computer interface for disabled subjects http://mmspg.epfl.ch/BCI_datasets

[27] Hertz, J.A., & Krogh, A. (1991). A Simple Weight Decay Can Improve Generalization. *NIPS*.

[28] The MNIST Database: http://yann.lecun.com/exdb/mnist/