ELL319

EEG Source Localization using Deep Learning

Group no. 13

Somanshu Singla (2018EE10314)
Shubham Mittal (2018EE10957)
Sarthak Gupta (2018EE30561)
Jayant Choudhary (2018EE30547)
Dhruv Anjaria (2018EE30561)

Table of contents

What is EEG?

Electroencephalography (EEG) is a technique for investigating human brain activity.

Our brain cells communicate with each other using electrical signals, EEG tries to measure this. These electrical signals are typically currents within neurons which cause voltage fluctuations which we measure as our recording.

It is typically measured by placing electrodes on our scalp.

EEG Source Localization

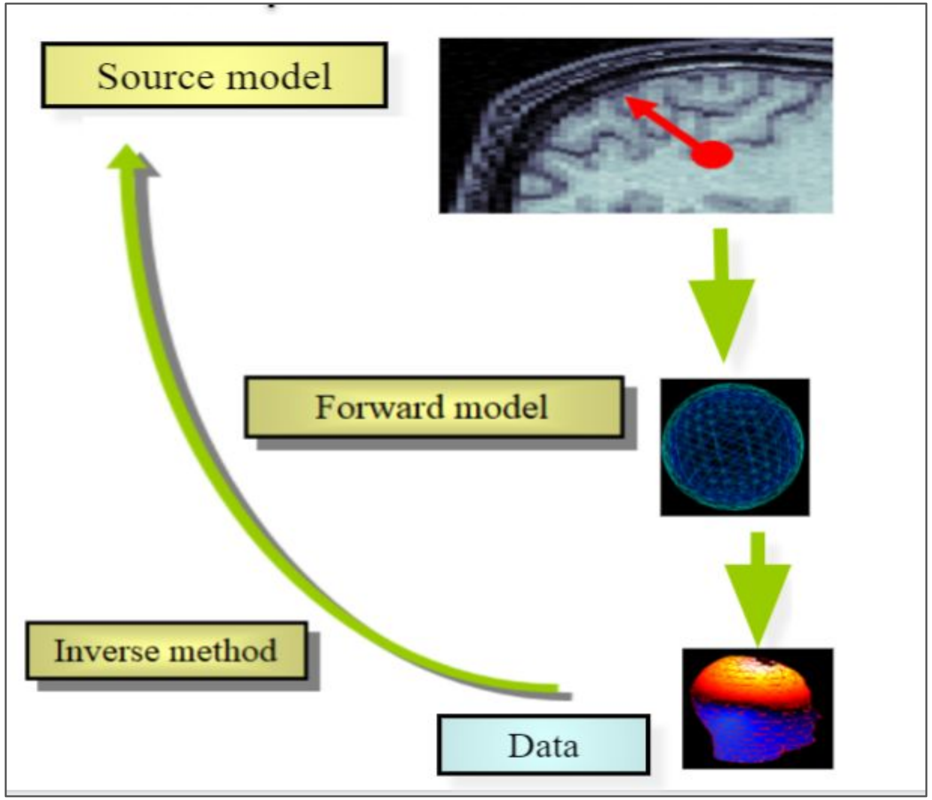
Source localization has been one of the primary goals of solving the inverse problem in EEG - identifying where in the brain a particular type of activity originates based on the surface EEG recording

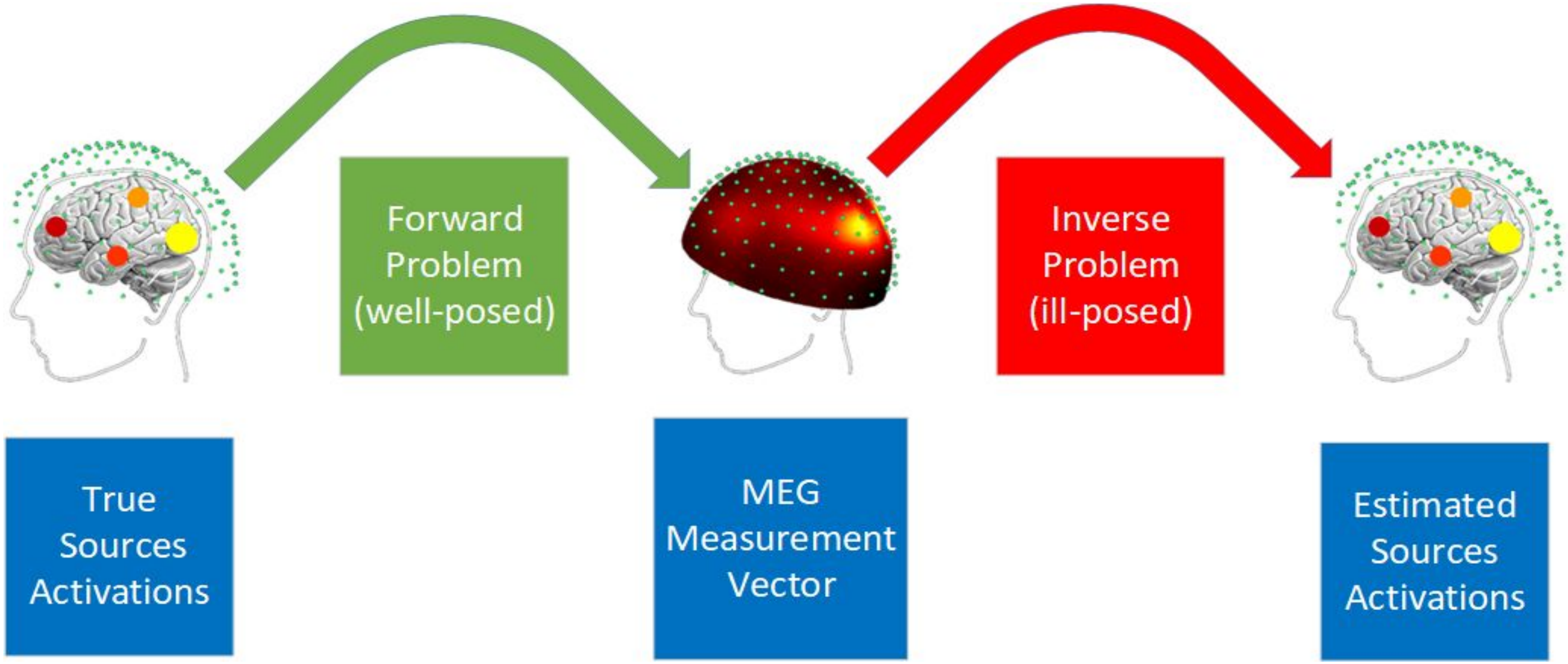
Identifying the source of an EEG signal requires : **source model** - tells 3D position (x,y,z) of dipoles on the cortical surface.

Inverse problem: model parameters (the location of the activity) estimated from measured data

Forward problem: model parameters (e.g. source location) are known and the data (e.g. the field at a given distance) is to be estimated.

Components of the source reconstruction process




Forward Problem (well-posed)
Inverse Problem (ill-posed)
True Sources Activations
MEG Measurement Vector
Estimated Sources Activations

Mathematical Formulation

$$Y = A(P)S + N$$

Y - $(M \times N_s)$ matrix of received signals

where,

$$Y = [y(t_1) \ldots y(t_{Ns})];$$

$$y(t) = \sum_{q=1}^{Q} l(p_q) s_q(t) + n(t)$$

$$A(P) = [l(p_1) \ldots l(p_Q)];$$

$$S = [s(t_1) \ldots s(t_{Ns})];$$

A(P) - (M x Q) mixing matrix of the topography vectors at the Q locations

S - (Q x N_s) matrix of sources

$$N = [n(t_1) \ldots n(t_{Ns})];$$

N is the (M x N_s) matrix of noise.

Note: M is the no. of sensors, Q is the finite no. of equivalent current dipole,

N_s is the no. of samples

WHY DEEP LEARNING

1) Very high localization accuracy of multiple sources from noisy EEG measurements

2) Very fast computation time (fraction of a millisecond) for inference of the source locations.

3) Less explored in the domain of EEG source localisation.

4) The stated paper shows that the Deep Learning architectures are robust to errors in forward matrix.

Deep Learning Techniques

The paper broadly uses two variants deep neural network architectures :

- Multi Layer Perceptron (MLP)
- Convolutional Neural Network (CNN)

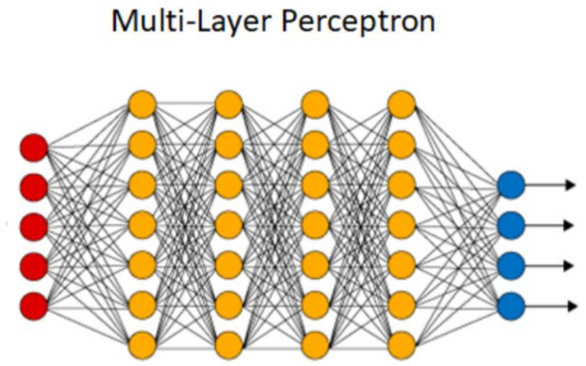
The Multilayer perceptron

The perceptron

“Perceptrons”. A perceptron is a repeating unit in a MLP model whose output is defined as :

$$Y = f(W^T X + b)$$

- Y : Output
- W: Weight Matrix
- X: Input vector
- b: Bias weight
- f: Activation function (non-linear)



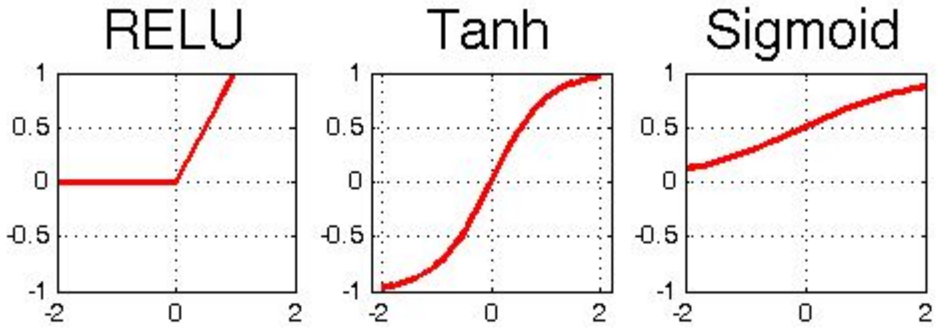
The Multilayer perceptron

More on the activation function

- Why Non-Linear?
  - Linear function is just another matrix, gets absorbed. No added utility.
  - Enhances expressive power (to an extent that universal approximation theoretically feasible)
- Example, the ReLU (Rectified Linear Unit)

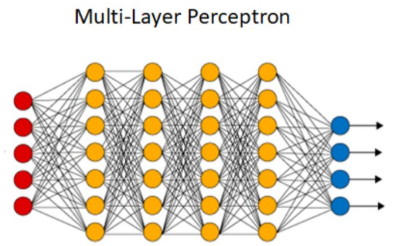
$$f(z) = \begin{cases} z & \text{for } z > 0 \\ 0 & \text{for } z \leq 0 \end{cases},$$

- Many other possibilities, choice may vary as per application.



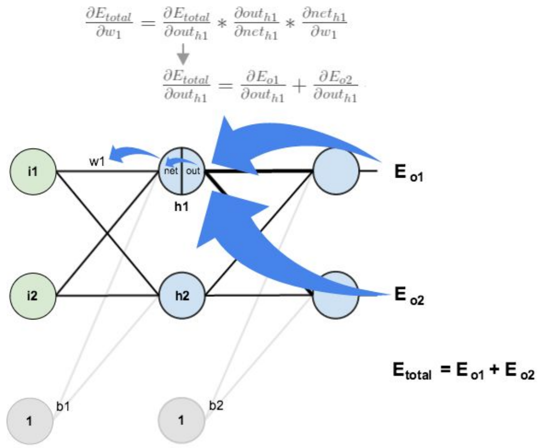
The Multilayer perceptron

Functioning



- The Forward pass.
  Output of each perceptron layer fed to the next layer. Hence the name.

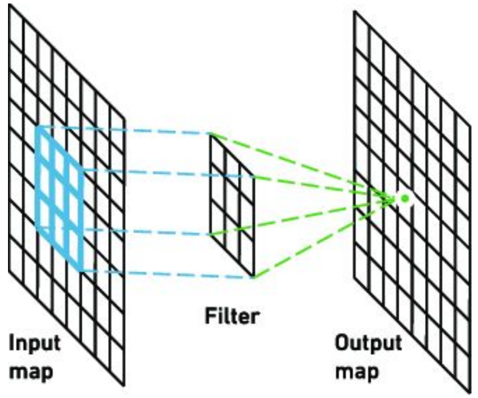
- The Backward pass.
  - Weights need to be decided?
  - Gradient-based methods used based which use available data.
    - Loss function depends on the final output.
    - Chain rule employed to compute gradients.
    - Error is the “propagated” backwards.
  - Called Backpropagation.

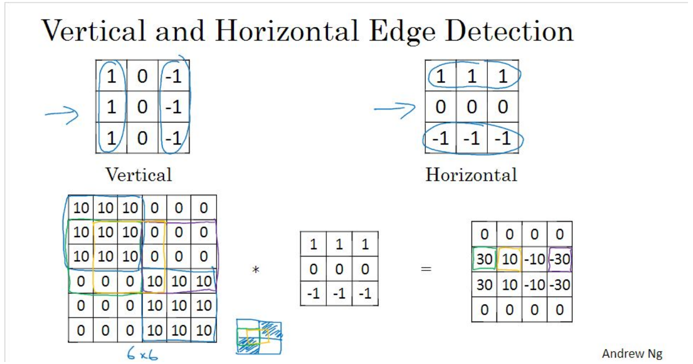

$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$
$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$
i1
w1
net
out
h1
E_{o1}
i2
h2
E_{o2}
$E_{total} = E_{o1} + E_{o2}$
1
b1
1
b2

CNN Architecture

Convolution

- Kernel/ Filter of fixed size.
- Number of parameters lesser than perceptrons.
- Additional methods like pooling/ padding used.
- Captures spatial components of input.
- Known to work extremely well for visual data.
    - Eg. Edge-detection



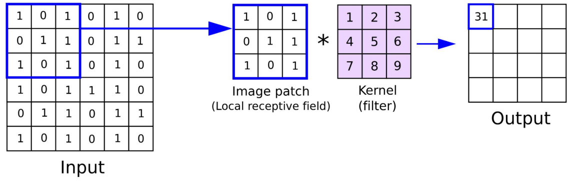

Vertical and Horizontal Edge Detection

$$\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$$
Vertical

$$\begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix}$$
Horizontal

$$\underbrace{\begin{bmatrix}10&10&10&0&0&0\\10&10&10&0&0&0\\10&10&10&0&0&0\\0&0&0&10&10&10\\0&0&0&10&10&10\\0&0&0&10&10&10\end{bmatrix}}_{6\times6} * \begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix} = \begin{bmatrix}0&0&0&0\\30&10&-10&-30\\30&10&-10&-30\\0&0&0&0\end{bmatrix}$$

Andrew Ng



Training process involves gradient-based backpropagation, chain rule.

EEG Dataset

EEG Dataset

We generated two types of datasets,

1. Signals coming from the whole volume of brain
2. Signals coming from the outer cortex of brain only

   The Gain matrix and orientation matrix were provided, we generated the sine samples at some frequency and used that as our intensity vector.

   To produce the data, we multiplied the Gain and orientation matrix and later with intensity matrix.

   For 1 source dataset, we kept one row of intensity matrix to be non-zero (sine samples) and others to be zero, whereas for 2 source dataset, we kept two rows to be non-zero (sine samples) and others to be zero.

For MLP model we created single snapshot datapoints -

$$X \in R^{(M \times 1)} \qquad Y \in R^{((Q*3) \times 1)}$$

For CNN model we created multiple snapshot datapoints -

$$X \in R^{(M \times 16)} \qquad Y \in R^{((Q*3)\times 1)}$$

Where M is number of sensors, Q is number of sources activated.
We created datasets for Q=1,2.
In CNN model we used 16 time samples(as per paper)for creating one image of CNN data.
In our dataset M was 97.

For MLP Model:
Q=1 := 1486881 data samples were created.
Q=2 := 625000 data samples were created.

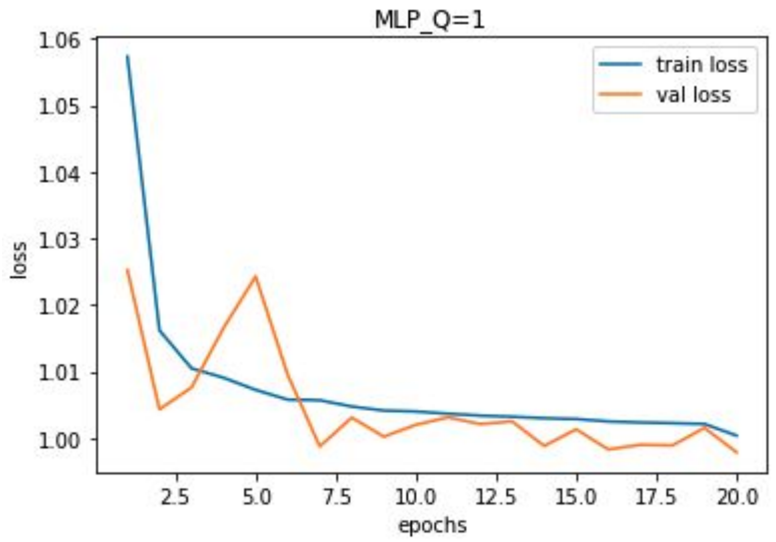
For CNN Model:
Q=1 := 215271 data samples were created.
Q=2 := 175000 data samples were created.

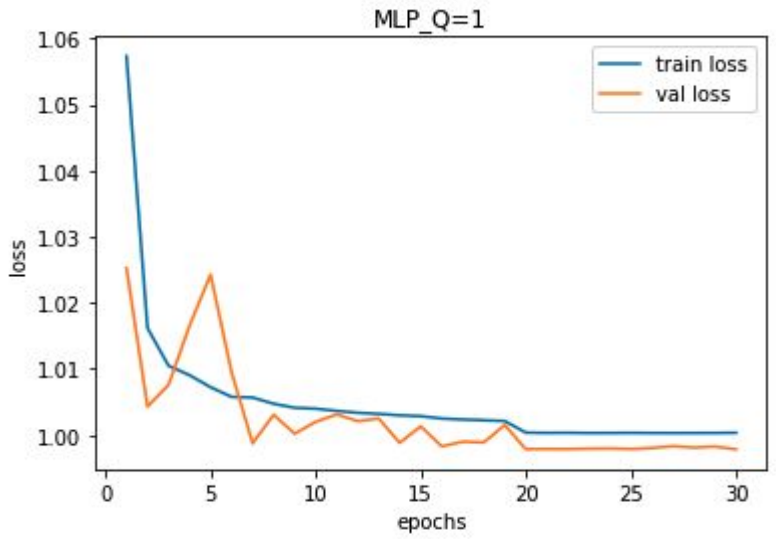
Experimental Results

MLP Architecture:

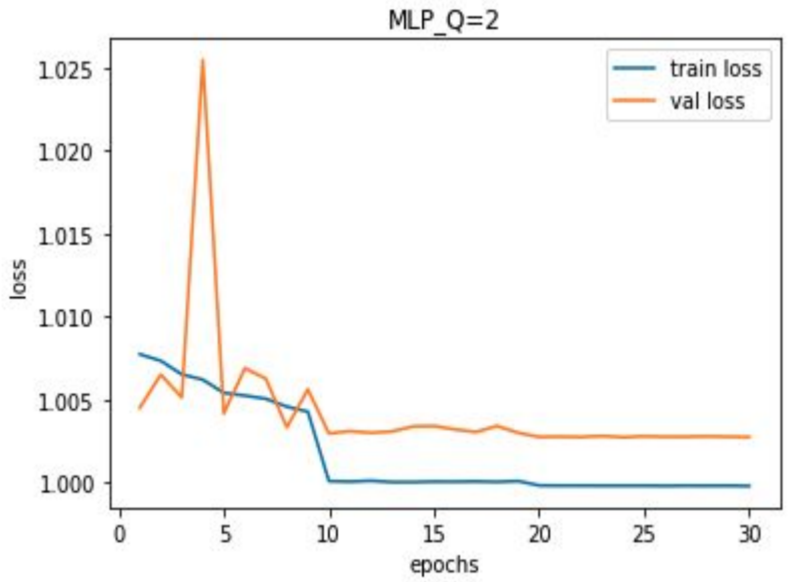
```
DeepEEG_MLP(
  (fc1): Linear(in_features=97, out_features=3000, bias=True)
  (fc2): Linear(in_features=3000, out_features=2500, bias=True)
  (fc3): Linear(in_features=2500, out_features=1200, bias=True)
  (fc4): Linear(in_features=1200, out_features=3, bias=True)
)
```

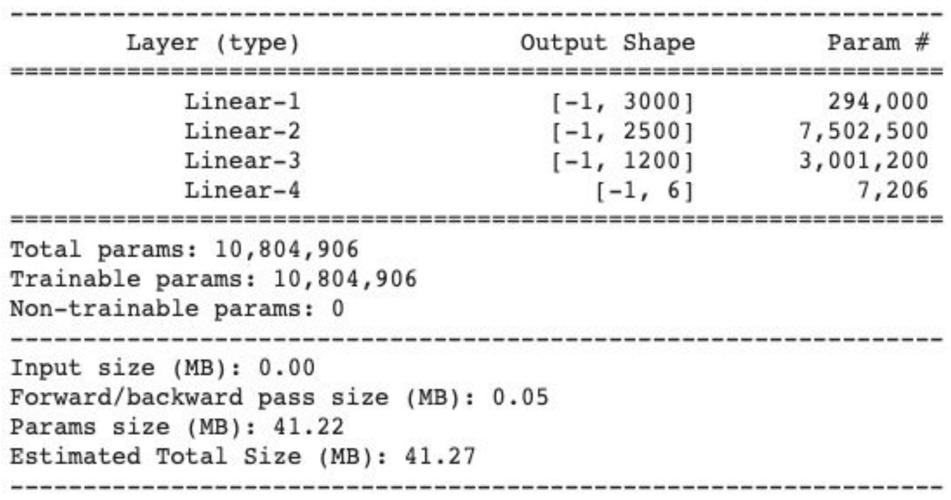
Training and Validation loss vs number of epochs:





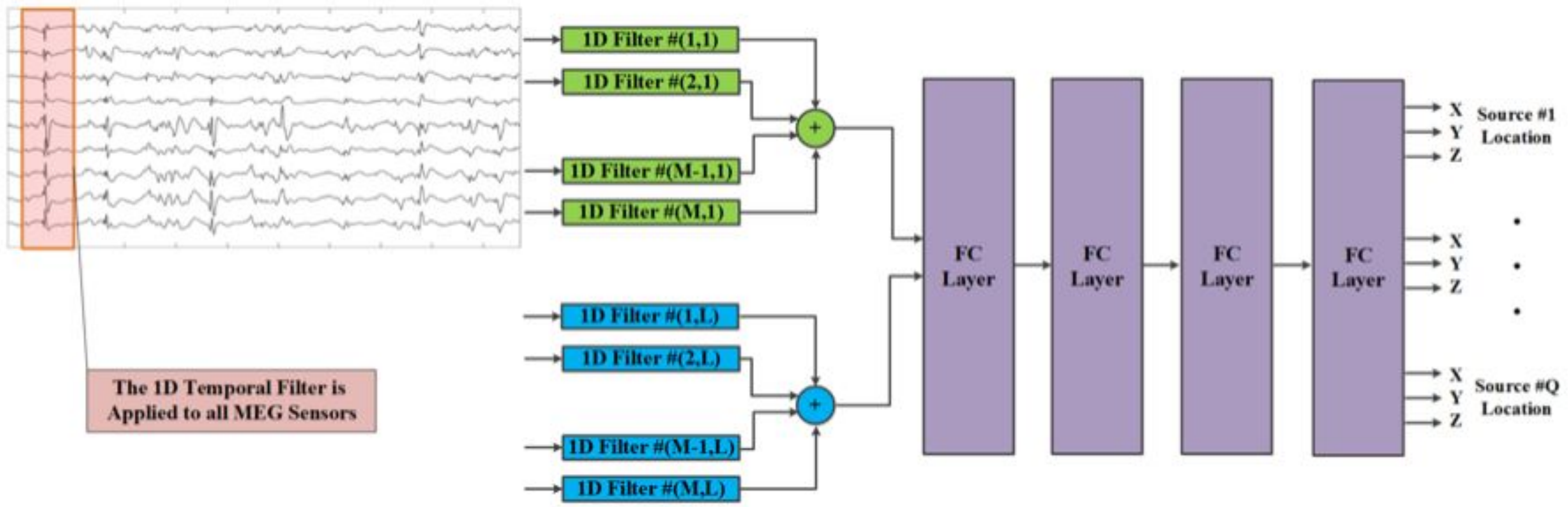
MLP architecture and training results for Q=2:



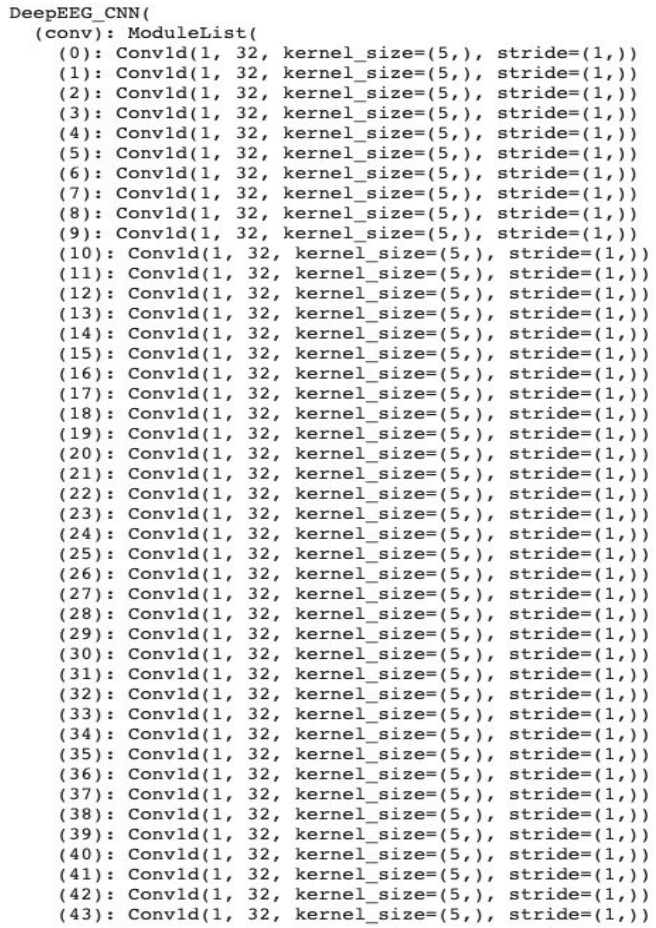


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 3000]         294,000
            Linear-2                [-1, 2500]       7,502,500
            Linear-3                [-1, 1200]       3,001,200
            Linear-4                   [-1, 6]           7,206
================================================================
Total params: 10,804,906
Trainable params: 10,804,906
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.05
Params size (MB): 41.22
Estimated Total Size (MB): 41.27
----------------------------------------------------------------
```

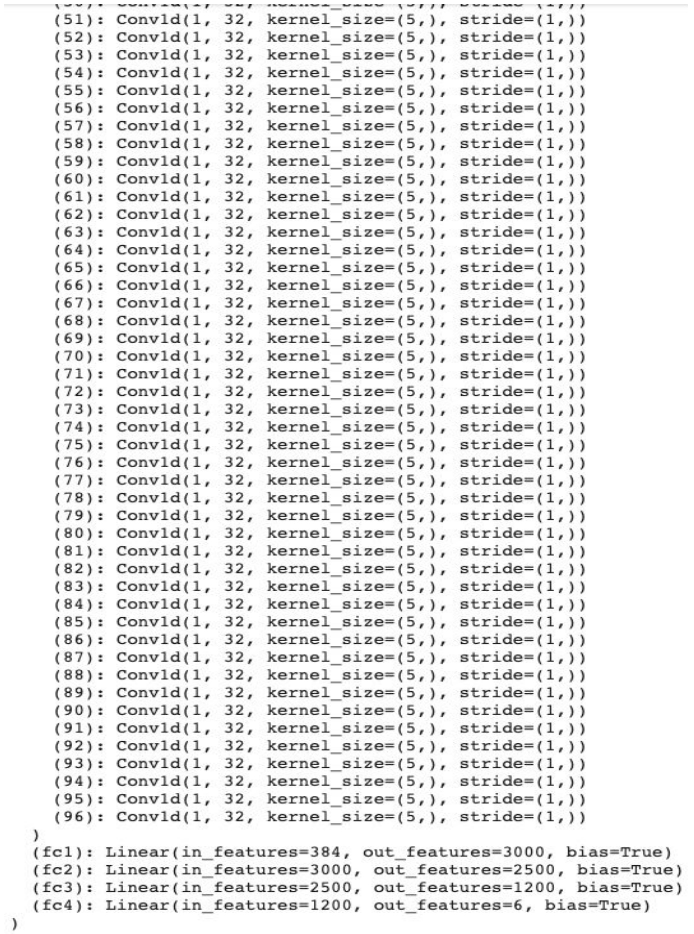
CNN Architecture (1D convolution):



CNN Architecture (1D convolution):

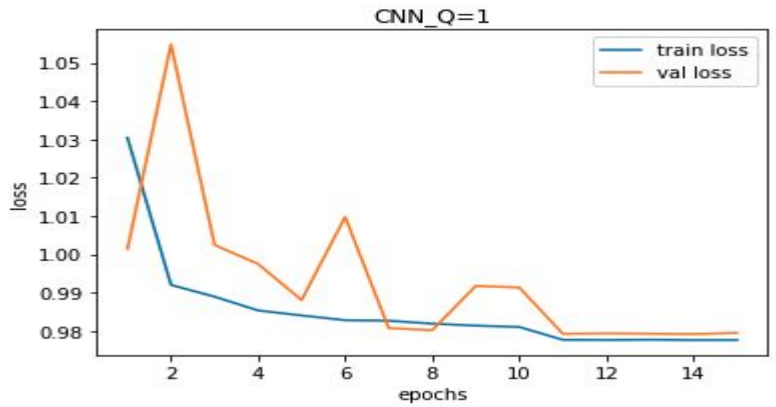


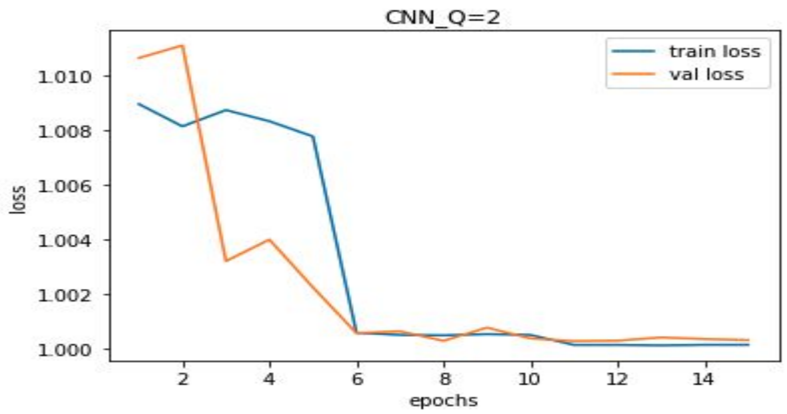
```
DeepEEG_CNN(
  (conv): ModuleList(
    (0): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (1): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (2): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (3): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (4): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (5): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (6): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (7): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (8): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (9): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (10): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (11): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (12): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (13): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (14): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (15): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (16): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (17): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (18): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (19): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (20): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (21): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (22): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (23): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (24): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (25): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (26): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (27): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (28): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (29): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (30): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (31): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (32): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (33): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (34): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (35): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (36): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (37): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (38): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (39): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (40): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (41): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (42): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (43): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
```

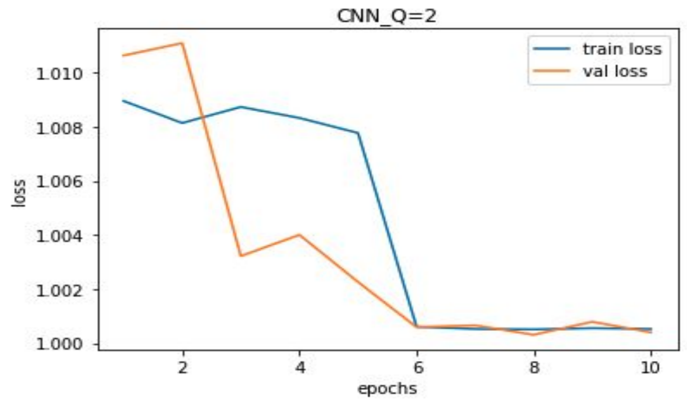


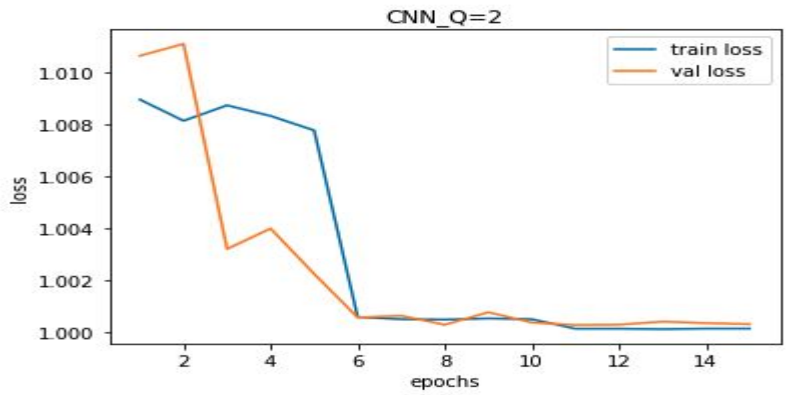
```
    (51): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (52): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (53): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (54): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (55): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (56): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (57): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (58): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (59): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (60): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (61): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (62): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (63): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (64): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (65): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (66): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (67): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (68): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (69): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (70): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (71): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (72): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (73): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (74): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (75): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (76): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (77): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (78): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (79): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (80): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (81): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (82): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (83): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (84): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (85): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (86): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (87): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (88): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (89): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (90): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (91): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (92): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (93): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (94): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (95): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
    (96): Conv1d(1, 32, kernel_size=(5,), stride=(1,))
  )
  (fc1): Linear(in_features=384, out_features=3000, bias=True)
  (fc2): Linear(in_features=3000, out_features=2500, bias=True)
  (fc3): Linear(in_features=2500, out_features=1200, bias=True)
  (fc4): Linear(in_features=1200, out_features=6, bias=True)
)
```

Training and Validation loss vs number of epochs CNN architectures (1D convolution):

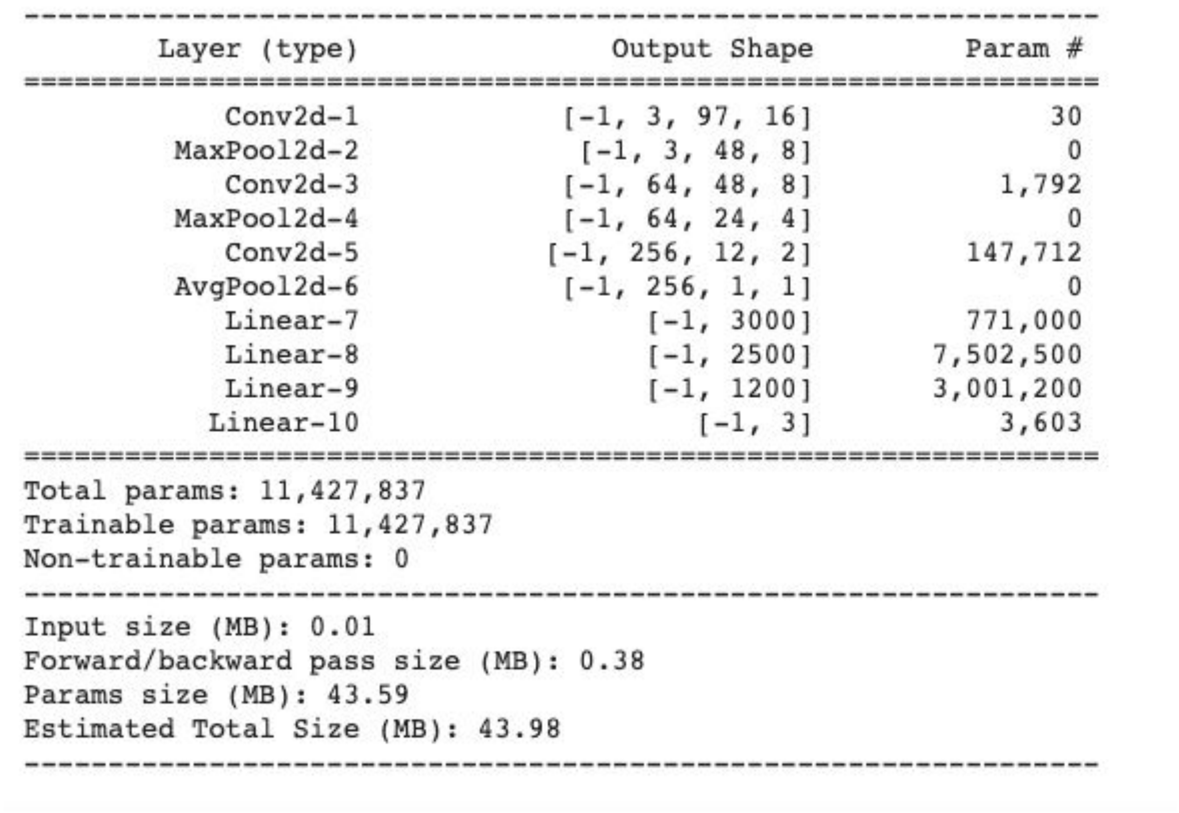






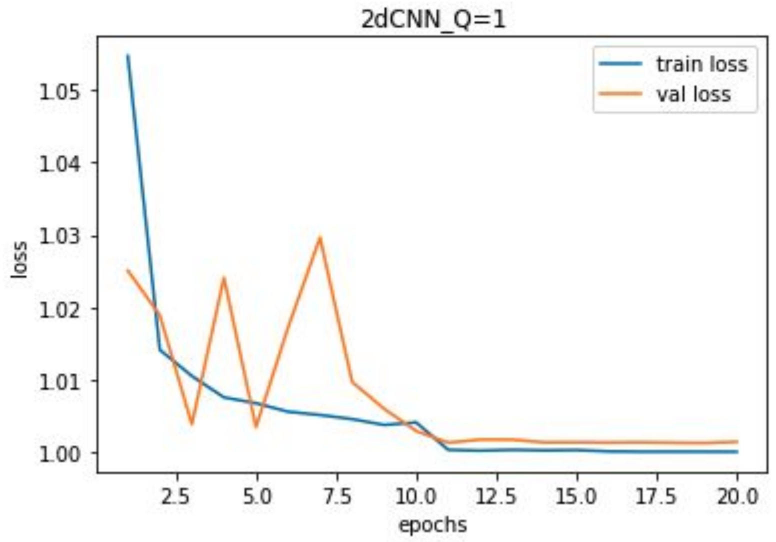


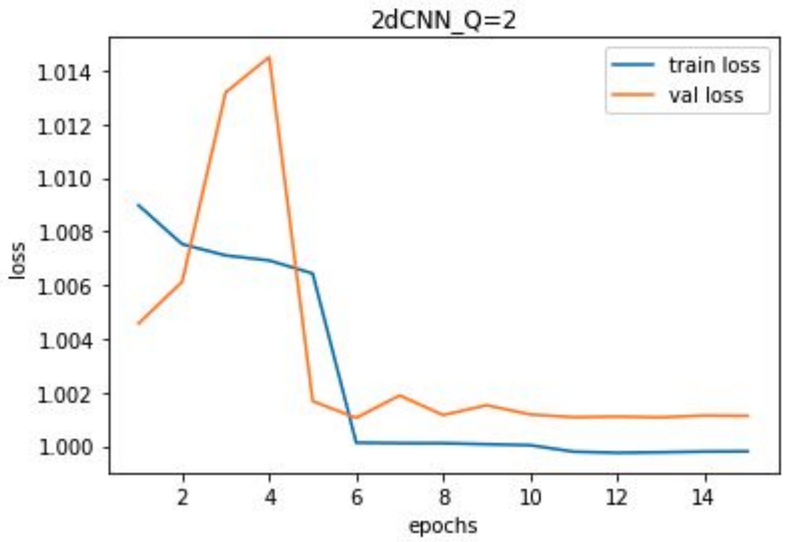
CNN Architecture (2D convolution):



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1            [-1, 3, 97, 16]              30
         MaxPool2d-2             [-1, 3, 48, 8]               0
            Conv2d-3            [-1, 64, 48, 8]           1,792
         MaxPool2d-4            [-1, 64, 24, 4]               0
            Conv2d-5           [-1, 256, 12, 2]         147,712
         AvgPool2d-6            [-1, 256, 1, 1]               0
            Linear-7                 [-1, 3000]         771,000
            Linear-8                 [-1, 2500]       7,502,500
            Linear-9                 [-1, 1200]       3,001,200
           Linear-10                    [-1, 3]           3,603
================================================================
Total params: 11,427,837
Trainable params: 11,427,837
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.38
Params size (MB): 43.59
Estimated Total Size (MB): 43.98
----------------------------------------------------------------
```

Training and Validation loss vs number of epochs CNN architectures (2D convolution):





Euclidean distances for various architectures (Q=1):

Architecture	*Minimum error (mm)*	*Average error (mm)*
MLP	11.85	58.72
CNN (1D Conv)	5.46	58.78
CNN (2D Conv)	**3.65**	59.26

Euclidean distances for various architectures (Q=2):

Architecture	*Minimum error(mm) (source-1)*	*Average error (mm) (source-1)*	*Minimum error(mm) (source-2)*	*Average error (mm) (source-2)*
MLP	8.37	54.07	16.91	55.58
CNN (1D Conv)	**8.10**	54.41	**13.21**	53.13
CNN (2D Conv)	9.57	54.37	13.80	53.12

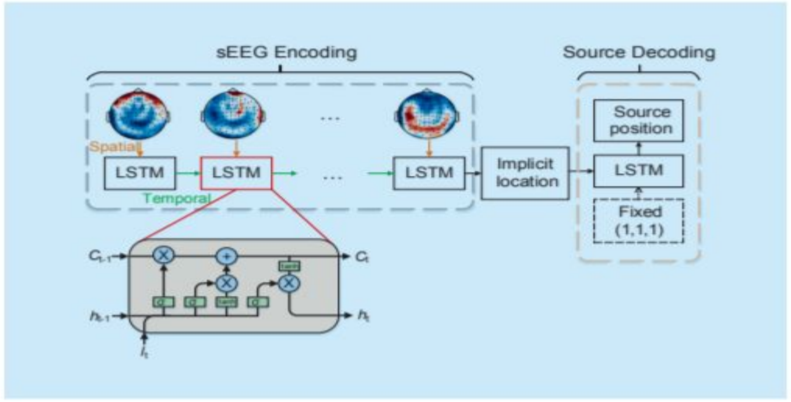
Future Work

EEG dataset or signals have temporal information (in time).

MLP is not designed to capture spatial or temporal information

CNN is used to capture the spatial information, but not temporal info.

Thus, use Recurrent Neural Networks which are made to extract temporal info.



References

1. Source_Localisation_1.
2. Forward_Inverse_Problem
3. A systematic review of EEG source localization techniques
4. Source localization for EEG and MEG
5. MEG Source Localization via Deep Learning
6. Head Harmonics Based EEG Dipole Source Localization
7. EEG source localization using spatio-temporal neural network

The code used in the project is available here.

The dataset and saved neural models are accessible in google drive

Id: g13ell319@gmail.com

Password: G13ell319EE