Brain2Word: Decoding Brain Activity for Language Generation (Language)

Lokesh Paidi, Varun Chowdary, Sashank S

Project Statement

- The current problem involves introducing a deep learning model that decodes brain activities in the form of fMRI scans into text.
- Instead of using the decoder to predict words, we build a decoder that works for sentences. This can be done using sentence embeddings.
- Decoding the natural language that represents semantic/narrative content from the neural activation patterns.

Goal : Decode natural language information from fMRI data

Our Understanding of Problem statement

- We will use the Paranoia dataset of Finn et al. (2018). This dataset consists of 22 participants' fMRI data, as they listened to a 22-minute audio-narrated story. Each participant's fMRI data contains three runs. The goal is moment-to-moment decoding of the natural language information from the corresponding fMRI responses.
- Build a model that predicts the sentence representations from the fMRI data and evaluate the Top-10 classification accuracy for the predictions.

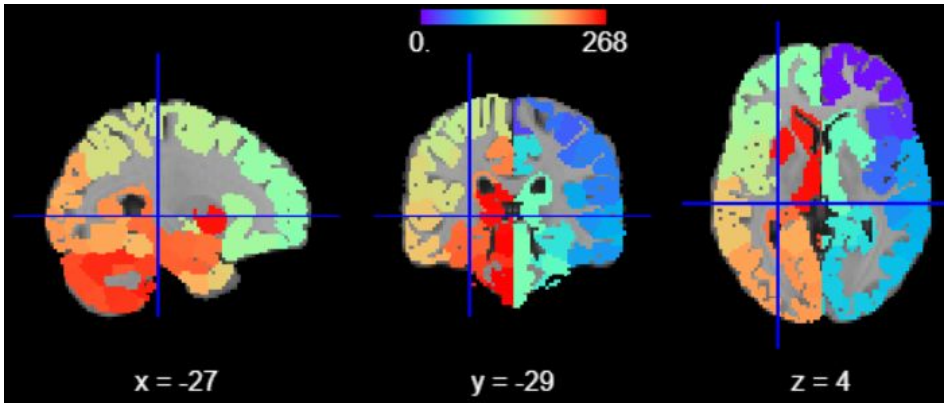
Methodology

Steps

- Preprocess fMRI data - ROI extraction
- Generate word embeddings using Glove
- Simple Linear Regression Model
- Simple Neural Network
- Neural Network with Self Attention
- Simple Auto Encoder
- Auto Encoder with Self Attention
- Transformer Model

Functional MRI data - ROI Extraction

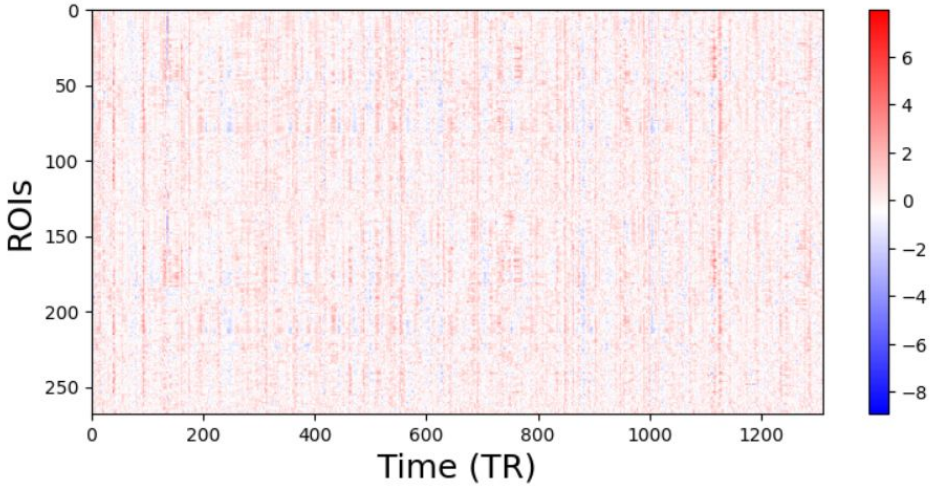
To reduce the fMRI data's dimensionality, we extract the mean time course of the regions of interest (ROIs), which is defined using the whole-brain functional parcellation called the Shen atlas (Shen et al., 2013), which has 268 parcels.



Functional MRI data

Next, we will extract the mean time course from all voxels in each ROI.

The original fMRI data is 4D (91, 109, 91, 1310), which corresponds to (x, y, z, time). The Shen Atlas has 268 ROIs, so the dimensions of the reduced data should be (1310, 268).

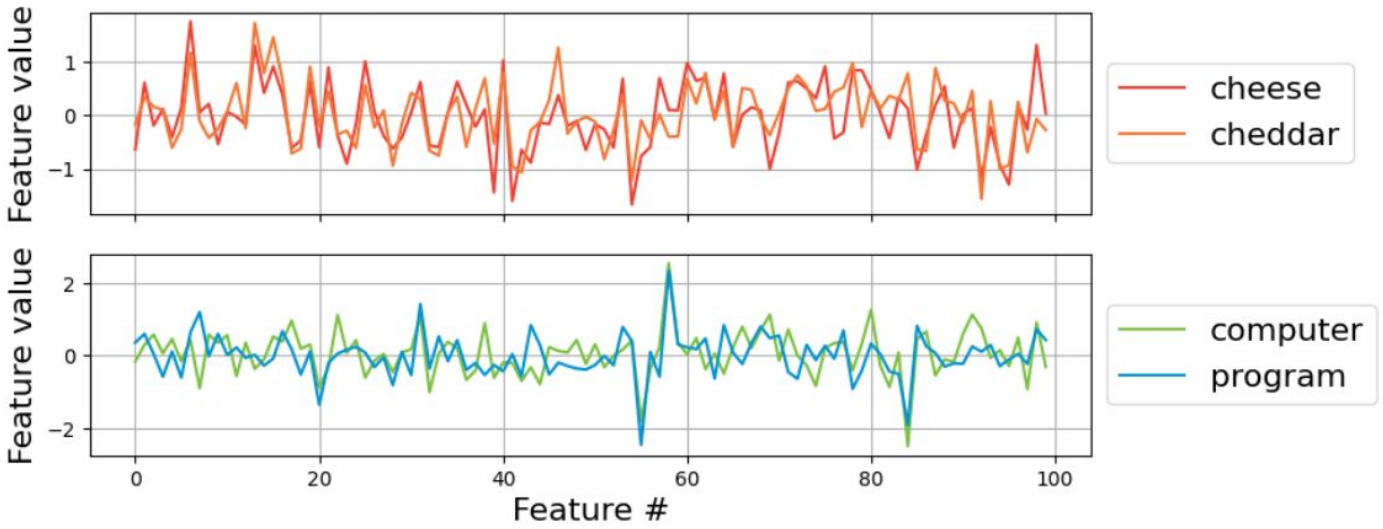


Word embedding

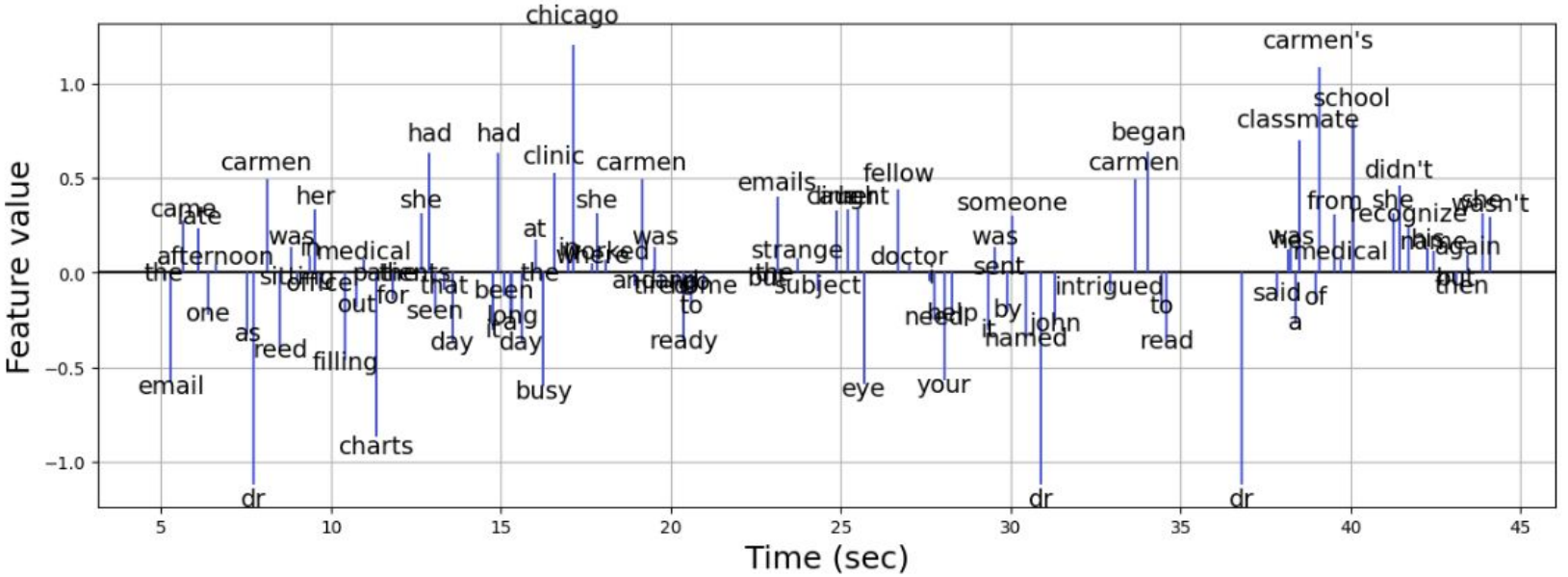
Next, we transform natural language information from the Paranoia transcript into vector representations using natural language processing (NLP) to represent the semantic meanings of the story quantitatively.

To represent each word in the Paranoia stimuli, we use a pretrained word embedding model. The purpose is to project each word into a point in a multi-dimensional embedding space, so that it retains its semantic relationship with all other words.

GloVe: Pretrained word embedding



The word vectors



Preprocessing the Paranoia transcript

To map embeddings to the fMRI data, the data matrix should be (1310, 100), which corresponds to (number of TRs, number of features from Glove).

Simply take the average of the embedding vectors of all words in each sentence, then assign this average vector for the duration of the corresponding sentence (from onset to offset).

Model 1 - Regression Encoder (Simple ML)

Finally, we have the fMRI data of (1310, 268), and the natural language data of (1310, 100).

We use a linear model to learn the relationship between the fMRI responses and word embeddings computed from the Paranoia transcript.In other words, we train a linear model with all time points for all subjects and test the model on the held-out participants' fMRI time course.

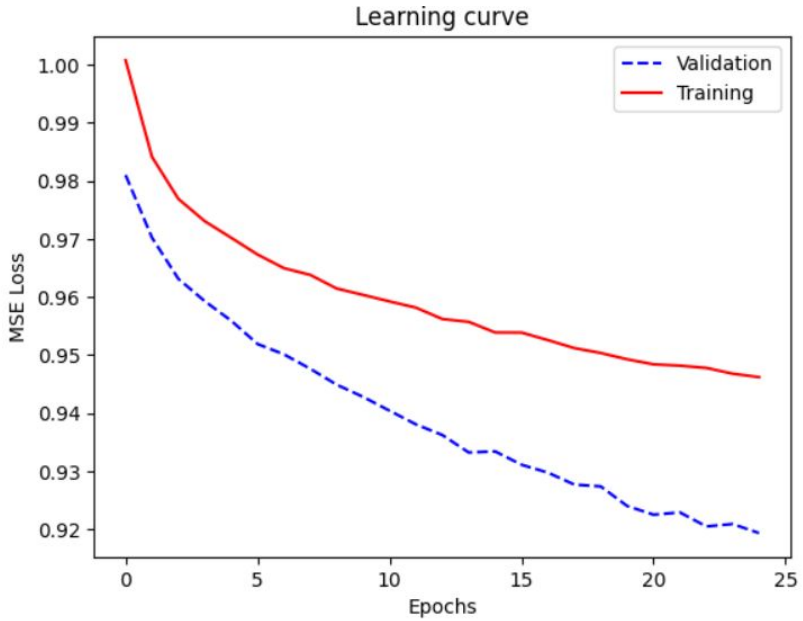
Metric - Top10 Similarity Accuracy

We predict the word embedding vector of each time point and measure its similarity (i.e., cosine similarity) with the observed vector. The decoding is successful if the similarity between the predicted and corresponding observed word vectors is ranked within the top-N over the similarities between the predicted and observed word vectors at all other time steps. Here, we use top-10, meaning the decoding is successful if the similarity between the predicted and observed vectors is within a rank of 10 over 1310 other possible observed vectors. Thus, the chance level is 10 / 1310 * 100 = 0.763%.

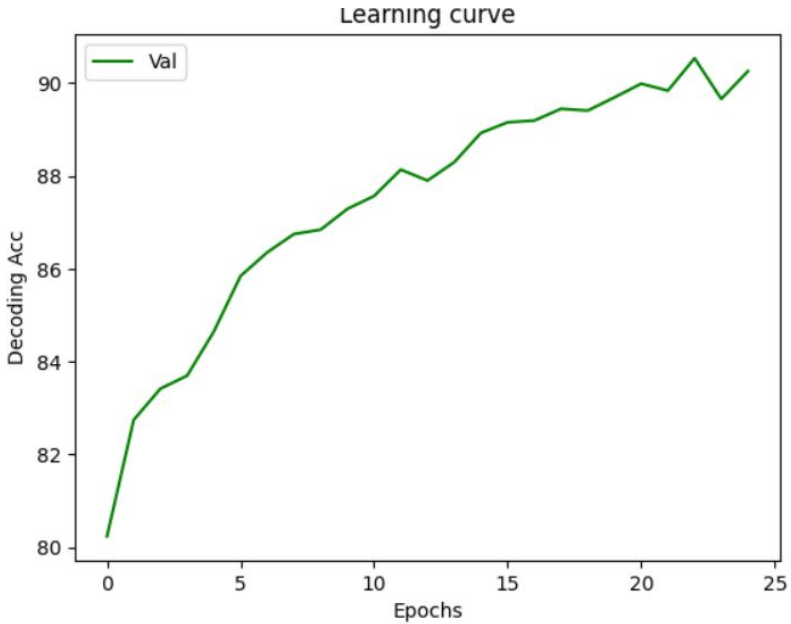
Result : Test decoding acc.: 0.957%

Simple Neural Network

```
class ROIBaseline(nn.Module):
    def __init__(self):
        super().__init__()

        self.encoder = nn.Sequential(
                    nn.Linear(268, 200),
                    nn.BatchNorm1d(200),
                    nn.LeakyReLU(0.3),
                )

        self.regressor = nn.Sequential(
            nn.Linear(200, 150), nn.BatchNorm1d(150), nn.LeakyReLU(0.3),
            nn.Linear(150, 100),
        )

        self.dropout = nn.Dropout(0.4)


    def forward(self, x):

        output = self.encoder(x)

        reg_out = self.regressor(self.dropout(output))

        return reg_out
```

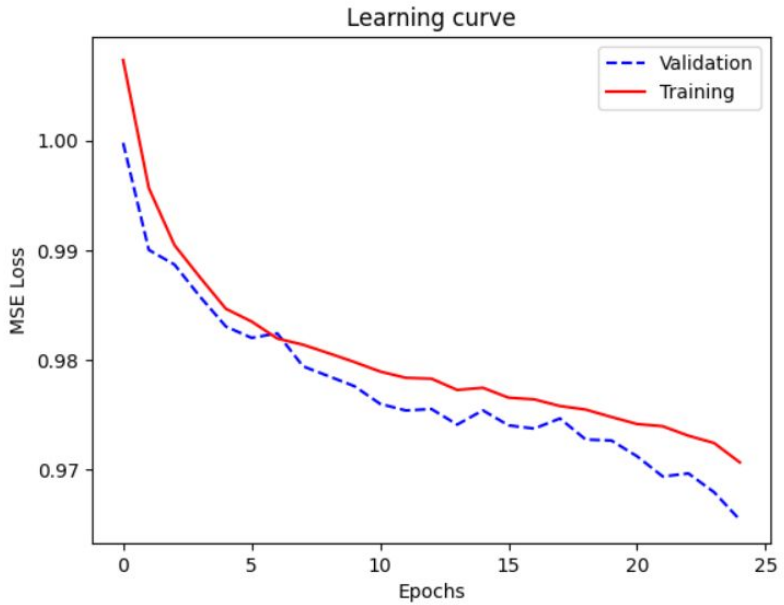


Decoding Accuracy

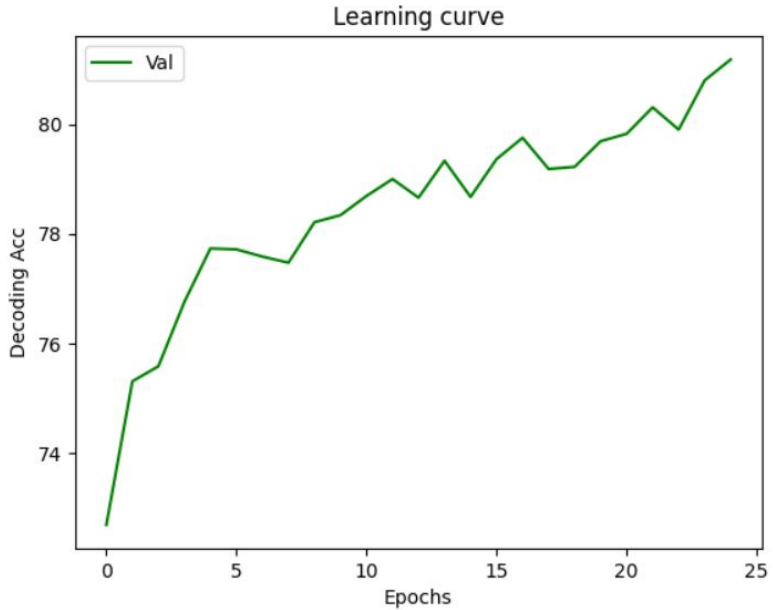


Attention Neural Network

```python
class SelfAttnROI(nn.Module):
    def __init__(self):
        super().__init__()

        self.encoder = nn.Sequential(
                    nn.Linear(268, 200),
                    nn.BatchNorm1d(200),
                    nn.LeakyReLU(0.3),
                )

        self.multi_head_attn = nn.MultiheadAttention(200, 8)

        self.regressor = nn.Sequential(
            nn.Linear(200,  150), nn.BatchNorm1d(150), nn.LeakyReLU(0.3),
            nn.Linear(150, 100),
        )

        self.dropout = nn.Dropout(0.2)

    def forward(self, x):

        latent = self.encoder(x)

        attn_output, _ = self.multi_head_attn(latent, latent, latent)

        reg_out = self.regressor(self.dropout(attn_output))

        return reg_out
```

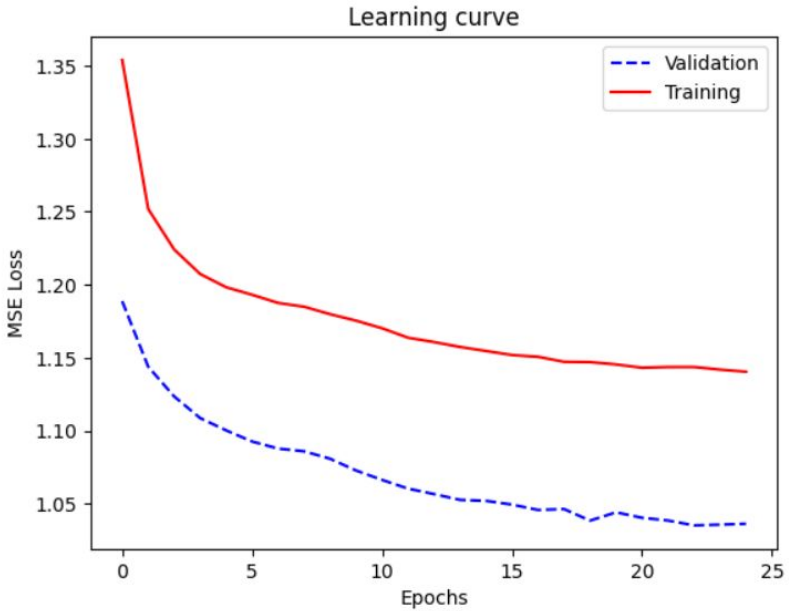


Decoding Accuracy

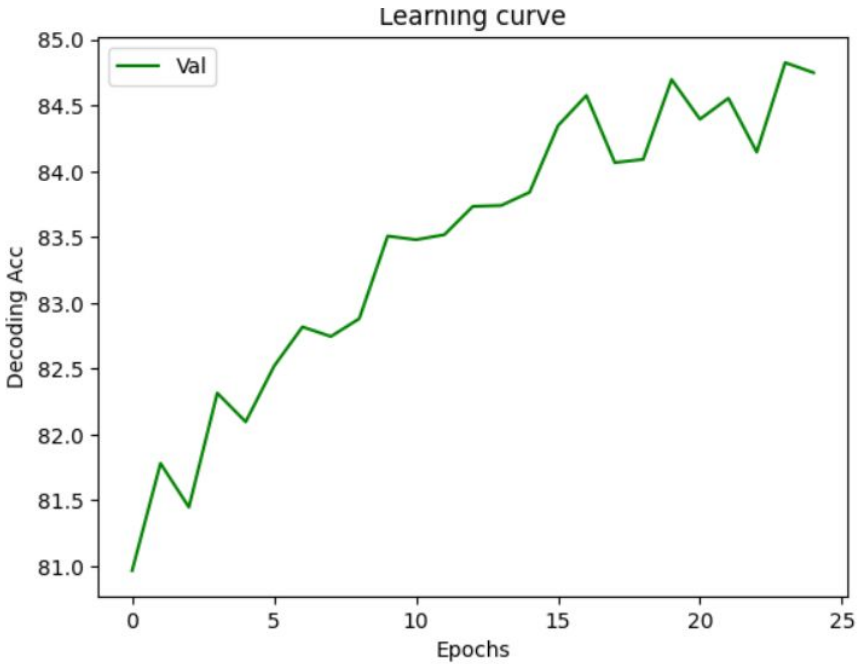


Autoencoder Network

```
class ROIAutoencoder(nn.Module):
    def __init__(self):
        super().__init__()

        self.encoder = nn.Sequential(
                    nn.Linear(268, 200),
                    nn.BatchNorm1d(200),
                    nn.LeakyReLU(0.3),
                )

        self.decoder = nn.Sequential(
            nn.Linear(200, 250),
            nn.BatchNorm1d(250),
            nn.LeakyReLU(0.3),

            nn.Linear(250, 268)
        )

        self.regressor = nn.Sequential(
            nn.Linear(200, 150), nn.BatchNorm1d(150), nn.LeakyReLU(0.3),
            nn.Linear(150, 100),
        )

        self.dropout = nn.Dropout(0.4)


    def forward(self, x):

        latent = self.encoder(x)

        reg_out = self.regressor(self.dropout(latent))

        recon_out = self.decoder(latent)

        return recon_out, reg_out
```

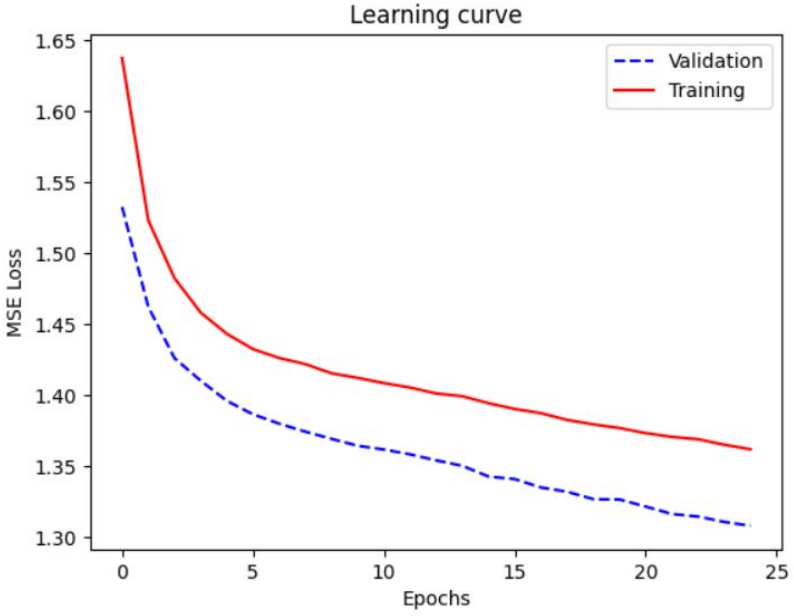


Decoding Accuracy

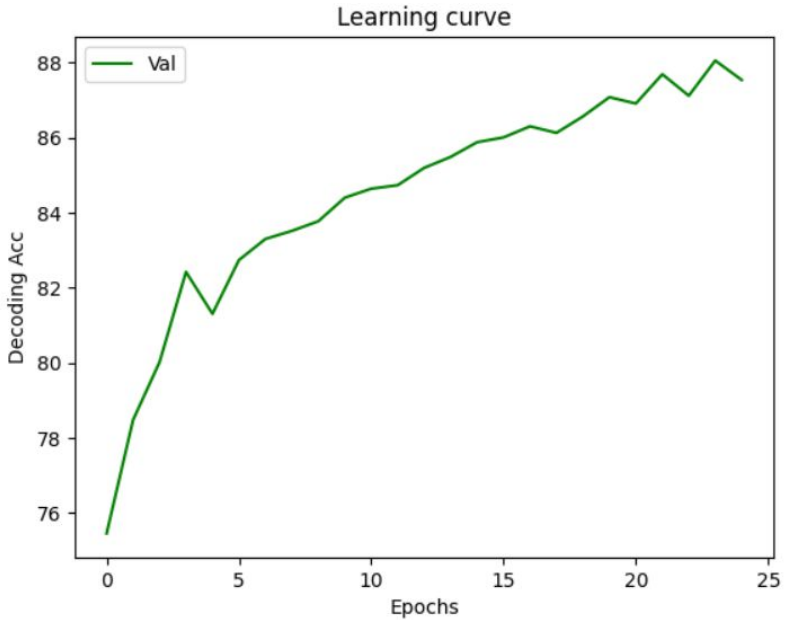


Attention Auto Encoder

```
class SelfAttnROIAutoencoder(nn.Module):
    def __init__(self):
        super().__init__()

        self.encoder = nn.Sequential(
                    nn.Linear(268, 200),
                    nn.BatchNorm1d(200),
                    nn.LeakyReLU(0.3),
                )

        self.decoder = nn.Sequential(
            nn.Linear(200, 250),
            nn.BatchNorm1d(250),
            nn.LeakyReLU(0.3),

            nn.Linear(250, 268)
        )

        self.regressor = nn.Sequential(
            nn.Linear(200, 150), nn.BatchNorm1d(150), nn.LeakyReLU(0.3),
            nn.Linear(150, 100),
        )

        self.multi_head_attn = nn.MultiheadAttention(200, 8)

        self.dropout = nn.Dropout(0.4)


    def forward(self, x):

        latent = self.encoder(x)

        attn_output, _ = self.multi_head_attn(latent, latent, latent)

        reg_out = self.regressor(self.dropout(attn_output))

        recon_out = self.decoder(attn_output)

        return recon_out, reg_out
```



Decoding Accuracy



Outcomes

Model	Decoding Accuracy
Simple ML	0.957
Simple Neural Network	90.53
Attention Neural Network	81.17
Autoencoder	84.82
Attention Autoencoder	88.05

Timeline

- Train Existing Models upto Convergence - 15 secs per step on average, Total of 1310 steps per epoch
- Build the Transformer Model (Architecture Done - Need to train) - 2 days approx

Plan for potential publication

Stage 1 - fMRI Processing : Other dimensionality reduction methods (e.g., PCA, ICA, SRM, etc.) can be chosen.

Stage 2 - We have taken each time stamp as a separate input. Use position embeddings for each time stamp for each subject instead.

Stage 3 - Sentence Embeddings - Use other embedding models such as BERT, Use, etc. Sent-BERT on entire sentence

Thank you