Imperial College London

COURSEWORK

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

Rough Paths for Machine Learning

Authors:
Yousif Al-Naimi (CID: 01843790)
Praneesh Kathirvel (CID: 01870253)
Dariyan Khan (CID: 01723886)
Daniel Zhang (CID: 02044064)

Date: 8th April 2024

1 Introduction

The HMS - Harmful Brain Activity Dataset is a dataset containing EEG time series data and spectrograms of critically ill patients during harmful brain activity like seizures. The goal of this task is to accurately classify the data into one of six types of harmful brain activities, as classified by a panel of experts in each case. The quality of our model is assessed by Kullback-Liebler Divergence with the “true” data, being the votes of the experts.

To accomplish this, we have utilised an ensemble model to combine various models using different strategies to result in an optimal learning method. These models include image models trained on the provided spectrograms, image models trained on generated spectrograms, and MLP models trained on log-signatures of the EEG time series data.

With this ensemble model, we were able to obtain a leader-board score of 0.34, higher than any of our models on their own.

2 Data

The Data consists of two parts, being the time series data and the spectrogram data for each ID, as well as its corresponding labels. These labels take the form of votes from a panel of experts on each of the events.

In turn, the time series consists of two parts, the EEG data, which comes from electrodes attached to the patients’ scalps at various points, and also a stream EKG data (relating to the heart rhythm of the patient). The spectrogram can also be generated from this EEG data using the correct transforms. The EEG data is sampled at 200Hz, and so the Nyquist frequency is 100Hz, which means that any frequency up to 100Hz is perfectly reproducible - this can help inform a denoising approach based on a band pass filter.

The time series for each event can be of various lengths of total time, and each event’s data is stored in a separate parquet file. To get the relevant data, there is a central 50 seconds containing the data at the time of the harmful brain event, and within those 50 seconds, there is a ten second central period which has been analysed and labelled by varying numbers of experts, giving us our targets. Therefore to use this data, as a pre-processing step we must extract the labelled central 10 seconds.

Finally, to co-ordinate the information together, there is a DataFrame showing which EEG signal corresponds to which votes, as well as other metadata. This is where we extract our targets from for testing KL divergence.

3 Pre-Processing

To pre-process the data, we must first get the labelled data out of the parquet files. To do this, we run through each and use the lengths and EEG IDs provided in the DataFrame to extract the relevant 50s, and then from that we extract the central labelled 10 second chunk of the time series data.

Then, after applying min-max scaling to standardise the different streams to be between 0 and 1, we time augment each time series, and we take signatures up to level 3 of this data using the `signatory` library [KL21]. We make use of the `logsignature` function to encode the relevant information in fewer parameters, and then we pass this data onto the machine learning methods we will be using that operate on the signature [CK16]. This transforms each time series into a vector of 2470 features.

While testing, we tried various techniques to denoise the time series before taking signatures of them. To do this, our first idea was to use a notch filter at 60Hz to cut out powerline frequency noise, but this did not have a significant impact. We then tried a low pass filter and butter filter combined to cut off anything over 20Hz, but while this resulted in a simpler time series, there was minimal impact to training. Lastly, we tried wavelet transformations, but this seemed to add unnecessary complexity and made predictive power worse, so we stopped using it. In the end, we did not use any denoising in our signature based method.

We also generate a spectrogram image directly from the EEG data, taking transformations that are optimised via training to generate an image that can be used for transfer learning from pre-trained image models.
We also applied "triple" lead lag augmentation, by taking the original time series $(x_0, x_1, ..., x_{1999})$, repeating entries and removing initial and final entries so we have:

$$
\begin{gathered}
(x_0, x_0, x_0, x_1, x_1, x_1, ..., x_{1998}, x_{1998}, x_{1998}, x_{1999}) \\
(x_0, x_0, x_1, x_1, x_1, ..., x_{1998}, x_{1998}, x_{1998}, x_{1999}, x_{1999}) \\
(x_0, x_1, x_1, x_1, ..., x_{1998}, x_{1998}, x_{1998}, x_{1999}, x_{1999}, x_{1999})
\end{gathered}
$$

achieving validation loss of 0.30 with N = 4000 EEG IDs.

4 Machine Learning

The first part of our machine learning model utilises the signatures we have calculated into an MLP model built in TensorFlow. After experimenting with various architectures, we found the following to yield strong results:

```
def create_model():
    inputs = tf.keras.Input(shape=(num_features,))
    x = layers.Dense(800, activation='sigmoid')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(500, activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Dropout(0.2)(x)
    x = layers.Dense(500, activation='relu')(x)
    x = layers.Dropout(0.2)(x)
    x = layers.Dense(350, activation='relu')(x)
    x = layers.Dropout(0.1)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)
    model = Model(inputs, outputs)
    return model
```

To assess the effectiveness of this model, we portioned the initial competition training data into a train set and test set (10%). Then after training the neural network with early-stopping (max patience of 10 iterations), we obtained the following results:

```
Epoch 173 - train loss:  0.2293 - val loss:  0.4407
```

The convergence of the full training cycle can be visualised in Fig. 1. We then applied the model to the test set to assess the generalisation ability of the model:

```
Test Loss:  0.4369
```

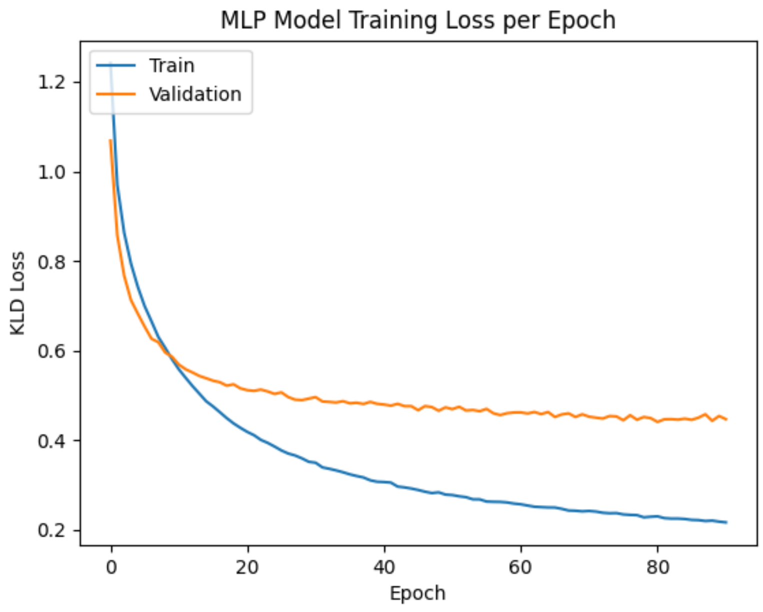

Figure 1: MLP loss convergence plot against training epochs

We also tried to use 1D convolution layers, however, these seemed to not achieve as good validation loss as a standard feed forward neural network.

The spectrogram data is high-dimensional (2000 time steps and 400 points each step). Initially, we tried to apply the signature transform to the spectrogram split across time chunks, then apply Multidimensional Scaling (MDS) as a linear protection to reduce dimensionality, before applying another signature transform to the full signature. Once done, we reshaped the final spectrogram as a 400x401 image, before passing it to the EfficientNet model. After much experimentation, the KL divergence failed to drop below one. Hence, we decided to not use signature based methods for the image data.

We then decided to use an ensemble of three models (all trained EfficientNet models) as our image model for our ensemble. One EfficientNet was trained on the original spectrogram data. The second and third were trained on spectrogram generated from the EEG data created by the community. The second one was trained on these generated spectrograms scaled across all data points, whereas the third one was trained on the spectrogram were each individual spectrogram had its own scaling applied.

Once we have found our best trained models, we took a weighted average of our model trained on the log signature of the EEGs, and the image model trained on the spectrograms in order to try and improve the KL divergence. After experimentation, we found that a weight of 0.15 for the EEG model and 0.85 for the spectogram model produced the best results.

To do prediction for the EEG model, we split the 50 seconds of EEG given to us in each row in the test set into windows from $0-10, 5-15, \dots, 40-50$ seconds, predicted distribution for each, and then averaged. This is because the same EEG appears multiple times in the training set, just with a different part of it labelled. Hence, we wanted to mimic it when forming a prediction.

5 Conclusion

With the above models together, we were able to reach a leader-board score of 0.34 utilising signatures, a fairly strong result.

If we had more compute resources, we would have liked to utilise more sophisticated methods involving the signature, for example the signature kernel, however due to the size of the data, even reducing sampling frequency would not allow the signature kernel to compute the Gram matrix based on our VRAM capacity.

References

[CK16] Ilya Chevyrev and Andrey Kormilitzin. **A Primer on the Signature Method in Machine Learning**. arXiv.org. 11th Mar. 2016. URL: `https://arxiv.org/abs/1603.03788v1` (visited on 08/01/2024).

[KL21] Patrick Kidger and Terry Lyons. **Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU**. 5th Feb. 2021. DOI: `10.48550/arXiv.2001.00706`. arXiv: `2001.00706[cs, stat]`. URL: `http://arxiv.org/abs/2001.00706` (visited on 09/03/2024).