Using a Neural Network to Classify EEG Brain Signals

Computational Neuroscience Spring 2019
J.J. Balisanyuka-Smith & Rose Ridder

*You have been added as a collaborator to our Swarthmore Git Repository, so you should be able to access it at: https://github.swarthmore.edu/rridder1/Psych43Project

Abstract

We sought to classify EEG data recordings between focused attentive, unfocused attentive, and inattentive mental states using a 1-D convolutional neural network on time-domain and frequency-domain data. We found that in the two datasets we examined, using the fourier transformed data did not conclusively alter the accuracy of the classifier, but it did improve accuracy in some cases.

Background

Classifying a person's attentiveness using EEG data can be useful in numerous ways, including real-time brain computer interface algorithms, psychological studies in which subjects may not be attending the task at hand, and in studying electrical rhythms within the brain.

Neural Networks

A neural network usually involves a large number of processors operating in parallel and arranged in tiers. It is not a topic that we have covered in class however the ability to transform the data on multiple layers to abstract different features at different levels of depth means that small features can be added in larger and larger quantities to create a larger and larger view of the environment. Neural networks are known for their ability to classify objects which is the type of task we are analysing.

A convolutional neural network allows us to compare features no matter where they appear in a dataset, we are not sure there is a strong theoretical reason to include it for our type of sta in the way we are using it. A Dense Neural Network means that every node in a layer has a connection of some type to the layer below.

Datasets

We identified to possible datasets to use as part of our experiment from the Kaggle data science community. The first dataset was in CSV-format with key statistics (mean, variance, correlation matrices), fft, and time series data. This dataset had 4 people classified into 3 mental states (relaxed, neutral, and concentrating) for 60 seconds each. This data had been used previously in a paper, attaining 87% prediction accuracy with just 44 out of 2100 features. We were interested in replicating these results and comparing how time and frequency domain inputs affected the accuracy.

The second dataset we investigated had MATLab-format data in a timeseries. This data also had 3 mental states (focused, unfocussed, drowsy), but contained 10 minutes of each state over 34 trials. We hoped to use this larger set of data to again compare time domain and frequency domain inputs to a neural network.

Data Parsing

Our first problem with both datasets was parsing the data from its original file format, such that we would be capable of calculating the fourier transform. The CSV file had headings on the first row for about half the data. These headings included key statistics like means, standard deviations, and moments, as well as a series of fft data columns. These metrics were followed by 1274 unlabelled columns, which we deduced were time data columns. Finally, there was a categorization column with integers of 0, 1, 2. In parsing our data, we first had to distinguish the unlabeled columns from each other within the data file by manually labeling these columns.

For the MATLab data, we used the scipy io library to load the data from each of the 34 files and parse it into arrays. The original data was composed of a single numpy data array nested within several list structures, which initially hindered our access to the data. However, after identifying the parsing issues with the data, we were able to separate the MATLab objects that were included in each file. From this, we identified the data objects that might be useful and stored them as numpy arrays.

Data Inputs and Target Categories

The CSV data was easily split into categories because the original authors included a final column filled with only 0's, 1's, and 2's corresponding to the three categories of mental attention state. We extracted this column from the data, and stored it in an array indexed identically to the time and frequency data.

From the MATLab data, we isolated the timestamp and data objects. We originally intended to use the stored "marker" object to separate the mental state of the participants, but after further investigation, we discovered that data was all 0's. The experimenters said they spent 10 minutes each in the focused, unfocussed and drowsy mental states in that order, so we considered using the timestamp dataframe. However, because the data was sampled at a constant rate, we were able to simplify the parsing process and ignore the timestamps, by instead splitting the data by calculating the sample interval indices using time*sample frequency. We ignored the 30 second time intervals on each side of focus transitions in order to exclude slight delays or other inconsistencies in the transition. Some of the data files were markedly longer than 30 minutes, but it was unclear from the documentation what mental state was in this excess data, so we did not include it.

Training and Testing Data Separation

We separated 80% of the data into a training set and 20% into a testing set that is completely excluded from any training. This exclusion allows us to accurately assess the validity of our trained neural network.

From the CSV data, we extracted 2,360 rows of data corresponding to that many input data/target pairs. When separated into testing and training data, it left us with 1,888 training sets and 368 testing sets.

For the MATLab data, in order to increase our training and testing datasets we separated the 9-minute (10 minutes were originally collected, but we excluded a 30-second buffer) segments into 18 30-second segments. We did this for each of the 34 MATLab files leaving us with 1,836 data to target pairs. This was then separated into 1,468 training pairs and 368 testing pairs.

Secondary Data Cleaning

The data in the CSV file was pre-cleaned for us, so we did not do any further filtering. We did however normalize the data so that all data was on a [-1, 1] interval so the neural network weights were trained on trends rather than relative amplitudes of measurements.

For the MATLab data, we did have to perform baseline filtering. We checked the fourier domain spectra to observe any other DC or AC component coming from the power sources, but we did not

observe a 50Hz or 60Hz outlying signal component, so we only performed the baseline filtering. For this data, we also normalized the input to avoid biasing the network by trends in the amplitude scale.

The Neural Network Model

Model Building and Compilation

Because the data was a 1-dimensional time series from the CSV file, we used only a dense neural network with a 3-node output for categorization. The output layer used a softmax activation function in order to get a set of probabilities to then select the most probable categorization.

For the MATLab data which included 13 channels, we started with a 1-dimensional convolutional neural net using 32 features and a window size of 5 channels. We then included a 30% dropout layer to mitigate overfitting by changing 30% of input in each iteration to 0. This was then flattened in preparation for a dense layer that connected to a 3-node output layer using a softmax activation function similar to that used for the CSV. We found that having multiple convolutional layers or dense layers lead to severe overfitting in our model, so we limited it to only one layer each. Pooling also could have helped in this regard, but resulted in too great a dimensionality reduction in the network structure.

Both models were compiled using the Nadam optimizer because it yielded the best results. Our loss and metrics functions were set to categorical cross-entropy and categorical accuracy respectively to ensure that the network predicted one and only one classification for each input. We trained the network over 12 epochs (cycles) which was enough to grow the accuracy without overfitting the training data.

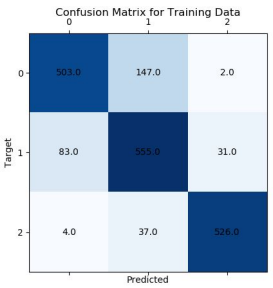
Conclusion

Our greatest limitation was the relatively small amount of documentation associated with our datasets. This was concerning because in the CSV data for example, it was unclear what the unlabelled columns were, and the fft columns appeared to extend beyond the nyquist frequency. This was one of the main reasons we began to pursue the MATLab data more seriously. With the MATLab data, although we encountered similar issues with the documentation, we were more confident in the type of data we were analyzing and our fourier transform calculation of that data.

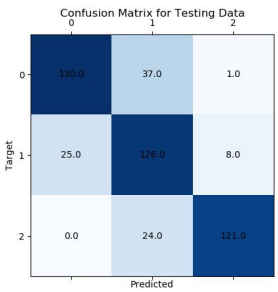
Data Type	Training Accuracy	Testing Accuracy
CSV Time series	83.3	79.87
CSV Frequency	58.90	55.51
MATLab Time series	99.59	36.68
MATLab Frequency	58.04	60.05

Overall, the CSV time series data yielded better accuracy while the MATLab frequency series data was more accurate. The time series data was grossly overfitted, which was the case even with only one epoch, and despite adjusting the model parameters to increase dropout and decrease feature space.

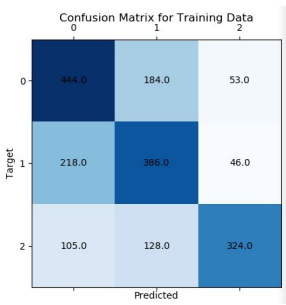
We also produced Confusion Matrices to display the neural network prediction vs. the target classification. As expected, in the data inputs with lower accuracy, there is a less definitive line along the diagonal. Also displayed in these matrices is which data classification the network had the hardest time distinguishing. In the frequency domain, the relaxed and neutral categories of CSV data were most often confused. In the MATLab dataset, the unfocused attentive state was the most confused.



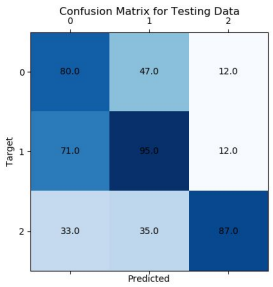
CSV Time Training Data



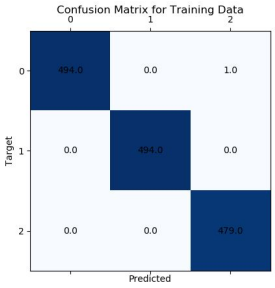
CSV Time Testing Data



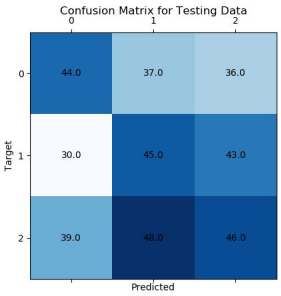
CSV Frequency Training Data



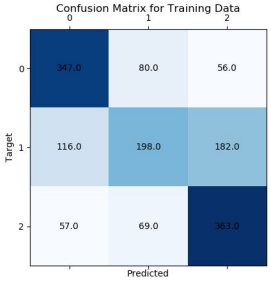
CSV Frequency Testing Data



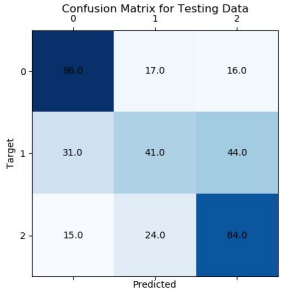
MATLab Time Training Data



MATLab Time Testing Data



MATLab Frequency Training Data



MATLab Frequency Testing Data

This data suggests that in some datasets, using the fourier transform of time data as input may be more helpful and accurate than time series data. However, it is unclear whether this is true in all cases or only few. The confusion matrices help to display similarities in the data which may not be detected simply by examining the waveforms, showing that the frequency representations of the data may not be as distinct for less focused mental states.

Attribution

Datasets:

C Inan, “EEG data for Mental Attention State Detection” on Kaggle.com, 2019.
https://www.kaggle.com/inancigdem/eeg-data-for-mental-attention-state-detection
J. J. Bird, L. J. Manso, E. P. Ribiero, A. Ekart, and D. R. Faria, “A study on mental state classification using eeg-based brain-machine interface,”in 9th International Conference on Intelligent Systems, IEEE, 2018.
J. J. Bird, A. Ekart, C. D. Buckingham, and D. R. Faria, “Mental emotional sentiment classification with an eeg-based brain-machine interface,” in The International Conference on Digital Image and Signal Processing (DISP’19), Springer, 2019.

Author Contributions

Conceptualization

R.R.: Initial neural network conception; fourier analysis comparison
J.B.S: Theory and connections to class material; overall viability assessment

Methodology

R.R.: fourier analysis comparison and neural network proposal
J.B.S: Review code; wrote documentation

Software

R.R.: parse & split data; structure neural network; testing
J.B.S: data format analysis

Data Curation

R.R.: Edit CSV headers, parse data in python
J.B.S: Data acquisition and management

Writing

R.R.: Original Presentation Draft; Review and Edit Report
J.B.S: Original Report Draft; Review and Edit Report

Visualization

R.R.: Confusion Matrix Plots
J.B.S: Formatting