Sleep Stage Classification Using Deep Neural Networks

Marouen CHAFROUDA Marouen.chafrouda@student.ecp.fr
Student at CentraleSupélec
Research Project for Dreem

Editor:

Abstract

This report describes the approach that was adopted for the classification of sleep stages and the intuition behind the models that will be presented. The goal is to be able to figure out the sleep stage of and individual among one of the five sleep stages (W, N1, N2, N3 and REM) by analyzing data extracted from Dreem's Headband (EEG, Accelerometer and Pulsometer data). Although, at first a more usual approach of using hand-engineered features that require deep knowledge of the type of signals we're working with (especially EEG data) was adapted the final choice was to try a more ambitious approach of using the power of Convolutional Neural Networks to Try and automatically extract relevant features from raw data and by also using Recurrent Neural Networks to use the Succession of these extracted features over several epochs to finally be able to get the sleeping stage. The code for this project can be found on the following github repository Dreem-Deep-Sleep-Net

1. Introduction

sleep stage scoring is a time-consuming task for sleep experts and in this project we propose an automatic sleep stage annotation method using the data recovered by Dreem's Headband. Our Model is composed of deep convolutional neural networks (CNNs) to extract time-invariant features, frequency information, and a type of Recurrent Neural Networks called LSTM (Long-Short-Term-Memory) to pick up the transitions dynamics between the different sleeping stages. In addition, to reduce the effect of the class imbalance especially for stage N1 sleep which doesn't occur quite as much and doesn't last for long, we applied Synthetic Minority Over-sampling Technique [2] and prevent the model from being biased towards classes that are more recurrent in dataset.

2. Data

2.1 The Sensors

The data consists of a set of signals from the Different Sensors available in the headband: 7 electroencephalogram (EEG) channels, an accelerometer for the 3 directions x,y and z, and a Pulsometer. The EEG data is sampled at a frequency of 50 Hz, the rest of the data at a frequency of 10 Hz.The signals are segmented into 30-s epochs, which are then be classified into different sleep stages by the experts.

2.2 Pre-processing

Usually the data that needs pre-processing is the EEG signal. As this can be extremely noisy. After examining the literature on how experts use the EEG data for sleep scoring [1] we decided to not perform any filtering on the data since usually any filtering below the 50 Hz level would have a high risk of loosing valuable Frequency domain information that is relevant to the task in hand, so at a sampling frequency of 50Hz no filtering can be done without loosing relevant data. Overall the pre-processing pipeline ended up in simply reformatting the data so that it can be fed to our Neural Network later on. We also standardized the data so that each epoch would have a mean of 0 and standard deviation of 1.

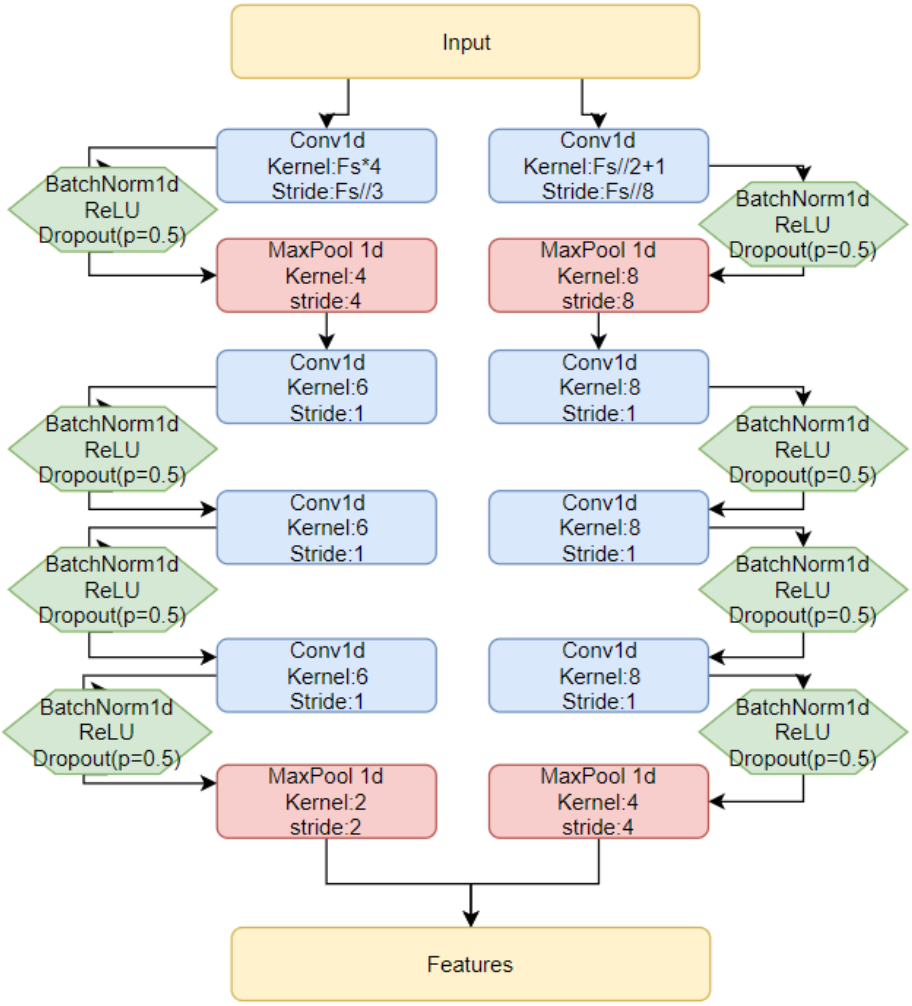
3. The Model

The architecture of the adopted Model is highly inspired from the DeepSleepNet[6] and sleepNet [5] . It consists of two main parts the Feature Extractor,which consists mainly of CNNs and used to learn relevant time-invariant features. The second part is the Sequential Model which consists of LSTM Networks and fully connected layers and is used to learn stage transition rules from a sequence of extracted features from several epochs of data (in our case we used a sequence of length 3 after testing shorter and longer lengths and comparing the performance of each one).

3.1 Feature Extraction

Before explaining more in detail how the feature extractor works it should be mentioned that the data is not sampled at the same frequency(50Hz for EEG channels and 10Hz for the rest) and the architecture of the model actually depends on these frequencies so two separate feature extraction models were actually used but they both have the same global architecture as shown in Figure 1 . The architecture is designed such that we can extract features that represent both the temporal aspect of the data and the frequency components. The small CNN is better equipped to capture temporal information and patterns in data, while the larger filter is better used to capture frequency information. The model was able to quickly overfit the data especially considering that the available data wasn't enough to fully train it so we used a lot regularization techniques to avoid overfitting and we also used early stopping, i.e we stop the training of the model before it fully converges but just before it starts to overfit (which is when the training error keeps going down but the validation error starts to go up). each CNN layer is followed by BatchNormalization, Rectified Linear Unit (ReLU) activation ($Relu(x) = max(x, 0)$) and a Dropout Layer with a dropout probability of $p = 0.5$. we also down-sample two times with the Maxpooling layer. Finally the extracted features (temporal and frequency components) are then concatenated and in a fist stage linked to a fully connected layer to output the class. This is used only in the phase of training the feature extractor and the fully connected layers will then be removed as the output is connected to the sequential model.

Figure 1: Model For Feature Extraction, Fs is sampling Frequency the fully connected Layers at the end are not represented as these are only used temporarily before getting rid of them

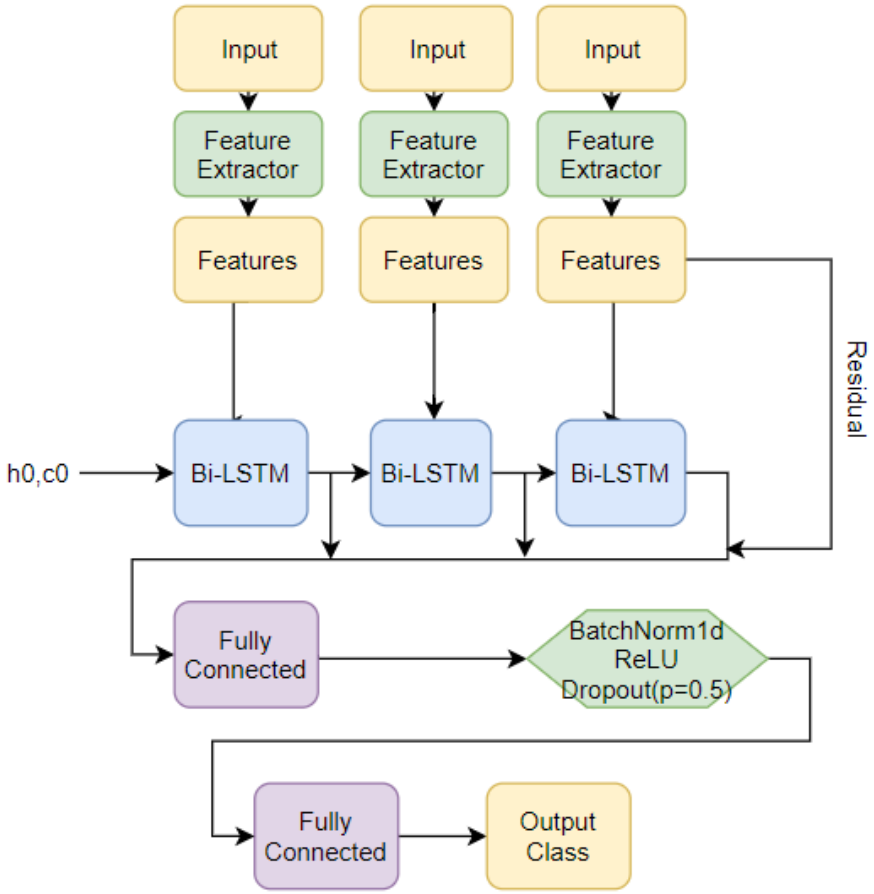


3.2 Sequential Model

Once the Feature extraction model is fully trained the data is reformatted to for sequences of several 30s epochs of data from which the features are extracted such that we have a sequence of features which is then passed to a Bidirectional LSTM. We also use Residual Learning here as we can see the features extracted from the last element of the sequence (which is the one we're trying to assign a sleeping stage to) are added back to the output of the Recurrent networks. This was First introduced in ImageNet architecture [3] and we use it here to make sure we don't lose any of the features extracted. Recurrent Dropout with probability p=0.5 is also used to avoid overfitting. The Full Model is presented in Figure 2

and as it can be seen we further use Regularization techniques as we introduce intermediate layers (BatchNorm,ReLU,Dropout) before outputting the final five classes.

Figure 2: Full Model: Feature Extraction followed by Bidirectional LSTMS and fully connected layers.



4. Model Training

The Model Was coded using Pytorch Library which is very handy to easily create Neural Networks while being able to manipulate how the are used. The model was trained Using a Google Colab GPU (Tesla K80 GPU) and takes around 1.5 hours overall to train. The Optimizer used is Adam Optimizer [4], and the Loss is CrossEntropyLoss. Now as described earlier the model training was done in an unconventional manner as we don't have one single model per say but 3 separate Models : (feature extraction for EEG data sampled at 50Hz, feature extraction for Acceleration and Pulse data sampled at 10Hz, Sequential Model). What we did is first we trained separately the feature extraction modules on an oversampled dataset using the SMOTE(Synthetic Minority Over-sampling Technique)

approach [2] in order to avoid the model being more biased towards the more occurring sleep stages. the features were then simply linked to a fully connected layer and we try to assign a sleeping stage. Once this is done we move on to training the sequential Module. This is done on two steps: First the Feature extractors weights are frozen and we only change the rest of the weights. This is done over several epochs and the best model weights are kept. Second Step is Fine tuning the Overall Weights: all the weights are changed but with a much lower learning rate and as always only the best model is kept(i.e we use early stopping when training the model).

5. Results

The model is evaluated on the Validation set using several metrics: Precision, Recall, F1-score the results are reported on Table 1. The Model seems to have poor performance recognize the N1 sleep Stage which coincides with the class with the lowest available data as we mentioned earlier so even though we tried to oversample the data this doesn't seem to fully solve this problem although the performance is better compared to training without oversampling. We also tried to change the Loss function used in a way that we penalize more the wrong guesses for the least occurring stages and that got us a better performance on those classes (class 1 especially) but the overall performance was worse.

Class	Precision	recall	F1-score
0	0.75	0.67	0.71
1	0.41	0.12	0.18
2	0.78	0.74	0.76
3	0.81	0.89	0.85
4	0.73	0.86	0.79
accuracy			0.77
macro Average	0.70	0.66	0.66
Weighted Average	0.75	0.77	0.75

Table 1: Results Over The Validation set with Features extracted from all sensors

As can be seen in Table 2, the performance doesn't increase is pretty much the same if we only use data from EEG sensors but the major difference is in classes 0 and 1, for which we have a much better performance when we add the accelerometer and the pulsometer data.

Class	Precision	recall	F1-score
0	0.74	0.62	0.67
1	0.37	0.06	0.10
2	0.74	0.83	0.78
3	0.88	0.90	0.89
4	0.76	0.76	0.76
accuracy			0.77
macro Average	0.70	0.63	0.64
Weighted Average	0.76	0.77	0.76

Table 2: Results Over The Validation set with Features extracted only from EEG data

6. Future work, Possible Improvements

This first improvement that we can think of is that we use a lot more data especially for the feature extracting modules and use less regularization as it seems to make our model somewhat more unstable and we use too many layers (Dropouts and BatchNorm) simply to avoid overfitting the data and we saw that if we use less data we'd need to do more regularization so using more data would definitely yield a much better performance. Due to Hardware Limitations, Hyperparameters choice wasn't done in a very thorough way and this can be optimized by doing some cross validations over these parameters although the necessary hardware would need to be used (as mentioned earlier the model as is takes about 1.5 hours to train ...) On Last thing that might need more in depth study is the feature extraction from the Accelerometer and the Pulsometer sensors for which we simply used the same architecture as for EEG data but with an adapted size of Network and this is probably not the best approach for this especially that as stated earlier these sensors seem to be able to provide some information about the sleeping states that our model struggles to recognize (classes 0 and 1) due to lack of data and more focus on these sensors could probably yield better results for these two classes.

References

[1] Irfan Al-Hussaini, Cao Xiao, M. Brandon Westover, and Jimeng Sun. Sleeper: interpretable sleep staging via prototypes from expert rules, 2019.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. 2011.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[5] Sajad Mousavi, Fatemeh Afghah, and U. Rajendra Acharya. Sleepeegnet: Automated sleep stage scoring with sequence to sequence deep learning approach. 2019.

[6] Akara Supratak, Hao Dong, Chao Wu, and Yike Guo. Deepsleepnet: a model for automatic sleep stage scoring based on raw single-channel eeg. 2017.