L'apprentissage face à la malédiction de la grande dimension, MVA, 2017/2018

Challenge Data ENS
Sleep stage classification: Dreem

Antoine Guillot, Joël Seytre

{antoine.guillot, joel.seytre} @ student.ecp.fr

This report describes our work related to the data challenge: `https://challengedata.ens.fr/en/challenge/37/learning_sleep_stages_from_physiological_signals_on_dreem_headband.html`.
Our results are entirely reproducible from the see github: `https://github.com/AntoineGuillot2/Challenge_data_ENS`.

1 Challenge description

1.1 Data description

The goal of the challenge is to classify the sleep stage of an individual given some of his vital signals. The classification should be done accordingly to the AASM standards in five different classes: Wake, N1 (Light Sleep), N2, N3 (Deep Sleep), REM (Paradoxal sleep). The following input signals were available for the classification:

- Electroencephalogram (EEG) which measures the brain electric activities through time. The frequency of the EEG is correlated with a given type of brain activity (for instance Delta Wave are associated with stage 3 [wikipedia]). Hence, this variable is expected to be an important predictor of the sleep stage.
  Each observation has four channels of EEG sampled at 125 Hz during 30 seconds (15000 variables).

- Pulse Oxymeter: The POx measures the saturation in oxygen of the blood, according to Ryan et al [Oxygen Consumption During Sleep: Influence of Sleep Stage and Time of Night], the POx decreases as the sleep depth and duration increase. Hence, POx can be use to detect the deeper stages of sleep. Each observation has two channels of Pox sampled at 50 Hz during 30 seconds (3000 variables).

- Accelerometer: the accelerometer measures the acceleration along the three axis x, y and z. Each observation has three channels of accelerometer data sampled at 50 Hz during 30 seconds (4500 variables).

Overall, the dataset contains 22500 explanatory variables. The training dataset is composed of 43830 samples and the test sets of 20592 samples. A few remarks can be made:
- The different signals are sampled at different frequencies, hence, they cannot be merged and dealt with directly. Downsampling of the EEG signals may be necessary before merging it with the POx and Accelerometer signals.
- Given the number of training variable and the number of training examples, the risk of overfitting is important. Hence, our models will require heavy regularizations and should be parametrized by few parameters.

1.2 Sleep stages description

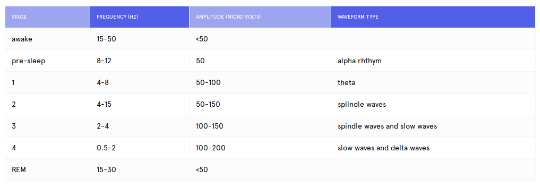
(i) awake
(ii) pre-sleep (N1): Respiration is slowing down, muscles are relaxing and you feel "half-asleep"
(iii) Slow light sleep (N2) : Eyes and muscles activities are going down, you can easily be awaken by a light or a noise. This stage represents 50% of the sleeping time.
(iv) Slow Deep Sleep (N3) : Eyes and muscles movements almost completely stop, brain waves slows and sometimes show burst of activity
(v) Slow Deep Sleep (N4) : This is the transition between N3 and deep REM sleep, Delta waves and faster waves are produced in the brain. This is typically when you dream, have night terror, ...
(vi) REM: The brain activity is similar to the one in the awaken state, eyes are closed but moving quickly. Expressions can appear on one's face. This stage represents 25% of the sleeping time.

From a machine learning standpoint, it is to be noticed that:
- without outside stimulation, the evolution through the stage sleep is sequential, one has to go through all the sleep stages before reaching REM.
- Sleep stages are unbalanced, N2 and N3 should be less frequent than other stages.

1.3 Sleep stages characterization

As said previously, the EEG frequency provides a good priors on the sleep stage, the typical EEG frequency associated with each sleep stage is shown in Figure 1. This will be of primary use for feature engineering.

Figure 1: Evolution of EEG with sleep stages



STAGE	FREQUENCY (HZ)	AMPLITUDE (MICRO VOLTS)	WAVEFORM TYPE
awake	15-50	<50	
pre-sleep	8-12	50	alpha rhthym
1	4-8	50-100	theta
2	4-15	50-150	splindle waves
3	2-4	100-150	spindle waves and slow waves
4	0.5-2	100-200	slow waves and delta waves
REM	15-30	<50	

1.4 General description of the pipeline

Due to the sequential nature of the data and the size of the dataset, we relied on CNN, RNN and CRNN to build a good representation of the data. Two class of model were built on these representations, end-to-end neural network and classifiers trained on the representation learned by the last layer of the neural network.

The neural networks were feed the 9 time series (EEG, POx, accelerometer) representing each observation and had to output the correct label.

2 Preprocessing and features engineering

No feature engineering was done, we let the work to the neural network and try to build an architecture that would be able to extract good representation of the data.

On the other hand, since we used neural network, the scaling of the data is of primary importance (see *Efficient Backprop*, Lecun et al.), the mean of the variables should be close to 0 and their variance to 1. Hence, two standardization scheme can be implemented:

- column-wise standardization: for a given variable x_t^i of the $i-th$ line observed at time t. The standardized variable is $x_t'^i = \frac{x_t^i - \overline{x_t}}{std(x_t)}$. This standardization actually removes the continuity between time-steps for a given variable and is not robust to extreme values. For instance, if one large anomaly happens in the measurement of the EEG, the variance will be driven to a large value and for this time-steps, all the $x_t'^i$ will be close to 0. This happened in the dataset since the EEG measurement are noisy and extreme values frequent.
- Row-wise standardization: for a given variable x_t^i of the $i-th$ line observed at time t. The standardized variable is $x_t'^i = \frac{x_t^i - \overline{x^i}}{std(x^i)}$. This standardization keeps the continuity within a time-series and just rescale it around 0 with unit variance. The variance and mean of each time-series was saved to be used as an input in the neural network.

3 Main Model

In this section, we describe the different iterations of the main model we tried and their respective performances on the training set. The models were evaluated using the Cohen Kappa on a validation set and implemented in Keras and Scikit-Learn.

3.1 Training parameters

The networks were trained on 15 epochs, the first 10 epochs are done using an RMSProp optimizer with the defaut Keras parameters and the last 5 epochs are done using a SGD (lr = 0.01) to stabilize the learning. Each batch contains 128 observations.The loss function is the categorical cross entropy and only the model with the best validation loss is used to do the estimation on the test set.
Due to the size of the dataset and our hardware, the batch are not completely random and some batches showed to be significantly harder than other ones, running the code with more

RAM would solve the problem and probably improve overall performance.
The training takes from 20 minutes to 1h30 for the more complex models and was done on a GTX1060 with 6Go of memory with 8Go of RAM (this was our bottleneck) on an i7-7700HQ.

3.2 Convolutionnal Neural Network

CNN (aka convolutionnal neural networks) were initially designed to analyze and classify pictures. For instance, given the raw picture of the number,how to determine which number is represented on the picture? The CNN use convolutionnal filters which are trained all along with the network to extract specific features from the image. These filters glide on the picture to extract patterns. The succession of layers and non-linearities make it possible to extract more and more complex patterns.
For our specific problem, we will use CNN on time series (EEG, POx and Accelerometer). Thus, only 1D filters are required. Each 1D filter will glide along the time axis and return a combinations of the different values. For instance, if we use a 1D filter of size 2 of value $[-1, 1]$ and with a stride of one, the convolution of the filter with the time serie will return the first difference of the time series. In the CNN case, the filter are trained all along with the network so that they can specialize for the given task.
The POx and the accelerometer channel are fed to the same input while the EEG is fed to an other input. Both inputs are processed through a three Convolutionnal layers (of size 5 with resp 16, 24 and 32 channels) and through a Max-pooling of different pooling size (resp. 2 and 5). With this two pooling size, both signals now have the same time resolution, hence, both signals can be merged merged and processed through 3 convolutionnal layers (of size 8 with 20 channels) with Elu activations and one pooling layer (of size 4). Finally the features are fed to two fully connected layers (resp with 30 and 50 units) and separated with a dropout (Dropout Rate: 0.5).

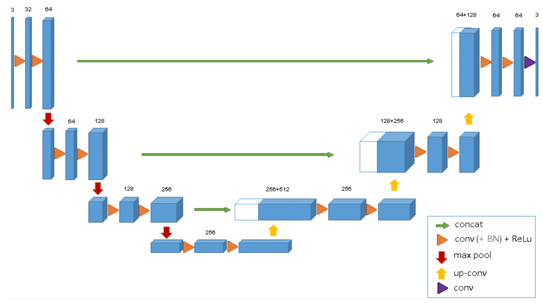
Results:

- Validation accuracy: 0.725
- Training accuracy: 0.78
- Validation τ: 0.625
- Training τ: 0.67

3.3 Multi-Scale CNN

Recently, segmentation techniques have been developed to deal with images and extract informations from different scales. For instance the U-net (Ronneberger, 2015),classicaly applies a sequence of conv-pool to the image but also feed the output of each pooling layer to the fully connected layers (Figure 2). The underlying idea is that the last layers will be able to combine informations from different scales.

Figure 2: U-net architecture



We applied a similar idea to our time series, several ConvPool Layers with different resolution are applied to the concatenation (channel-wise) of the time-series and their output is concatenated and fed a fully connected net.Hence, the FCN can work on signals of different scales and as expected this yielded an important improvement of the accuracy and τ To be more specific: The POx and the accelerometer channel are fed to the same input **A** while the EEG is fed to an other input **B**. Both inputs are processed through a three Convolutionnal layers (of size 5 with resp 16, 24 and 32 channels). Then, three differents pooling strategies are applied to their output:

- **Branch 1**: max pooling of size 2 on Input **A** and max pooling of size 5 on Input **B** $\rightarrow$ concatenation $\rightarrow$ three ELU convolution (20 channels, size 8) $\rightarrow$ max pooling of size 20.
- **Branch 2**: max pooling of size 4 on Input **A** and max pooling of size 10 on Input **B** $\rightarrow$ concatenation $\rightarrow$ three ELU convolution (20 channels, size 10) $\rightarrow$ max pooling of size 10.
- **Branch 3**: max pooling of size 20 on Input **A** and max pooling of size 8 on Input **B** $\rightarrow$ concatenation $\rightarrow$ three ELU convolution (20 channels, size 5) $\rightarrow$ max pooling of size 5.

The three branches are flattened and concatenated, they are then fed to two fully connected layers (resp with 30 and 50 units) and separated with a dropout (Dropout Rate: 0.5).

Results:

- Validation accuracy: 0.76
- Training accuracy: 0.81
- Validation τ: 0.67
- Training τ: 0.70
- Public Leaderboard τ: 0.61

3.4 Recurrent Neural Network

Recurrent neural networks are designed to deal with recurrent inputs such as time series, text or speech. Hence, RNN are likely to work well on our dataset which contains four time series.

We will quickly detail the functioning of a classical RNN cell. While a standard neural network layer only receive the current input, a RNN cell also has some memory of the previous states. It will keep some latent state in memory. We denote x_t the input and s_t the latent (memory) state at time t. f is the activation function of the layer, A_x, A_s and A_y the weights matrices and b the bias. Then, if we assume to have T time periods, and only one output y_T

$$s_{t+1} = f(A_x x_{t+1} + A_s s_t + b) \; and \; y_T = A_y s_T \tag{1}$$

In our architecture, we used GRU cells instead of vanilla RNN cell.In a vanilla RNN cell, there are as many matrix multiplication as the number of step, hence the norm of the gradient can easily go to 0 (gradient vanishing) or diverge. GRU cells are used to avoid gradient vanishing by having a direct gradient flow (Bengio 2014) and are require given the length of our series (2000 time steps for EEG).
However, the computational complexity was too high and and we were not able to run the full model on our computers.

Results: Not applicable, this motivates the development of the multi-scale CRNN.

3.5 Multi-Scale CRNN

Since we did not manage to run the vanilla RNN on our computers, we developed a way to mix the multi-scale-CNN with a RNN. The CNN is used to downsample the time-series and to lower their length to 50 to 200 timesteps. Then, a Bidirectional GRU take the output of the CNN as input. The architecture can be summarized as : $Input \rightarrow ConvPoolLayer \rightarrow downsampled\ input \rightarrow Bidirectionnal\ GRU \rightarrow Fully\ connected\ net \rightarrow Output$
The extension to a multi-scale CRNN is easy, a Bidirectionnal GRU is added at the end of each branch, the output of the 3 GRU are then concatenated and pull us back in the previous situation.
More formally the architecture is the following. The POx and the accelerometer channel are fed to the same input **A** while the EEG is fed to an other input **B**. Both inputs are processed through a three Convolutionnal layers (of size 5 with resp 16, 24 and 32 channels). Then, three differents pooling strategies are applied to their output:

- **Branch 1**: max pooling of size 2 on Input **A** and max pooling of size 5 on Input **B** → concatenation → three ELU convolution (20 channels, size 8) → max pooling of size 4.
- **Branch 2**: max pooling of size 4 on Input **A** and max pooling of size 10 on Input **B** → concatenation → three ELU convolution (20 channels, size 10) → max pooling of size 2 → concatenation with **Branch 1** → Bidirectionnal GRU (30 units)

- **Branch 3**: max pooling of size 20 on Input **A** and max pooling of size 8 on Input **B** $\rightarrow$ concatenation $\rightarrow$ three ELU convolution (20 channels, size 5) $\rightarrow$ Bidirectionnal GRU (30 units)

The three branches are concatenated, they are then fed to two fully connected layers (resp with 30 and 50 units) and separated with a dropout (Dropout Rate: 0.5).

Results:

- Validation accuracy: 0.78
- Training accuracy: 0.83
- Validation τ: 0.707
- Training τ: 0.74
- Public Leaderboard τ: 0.63

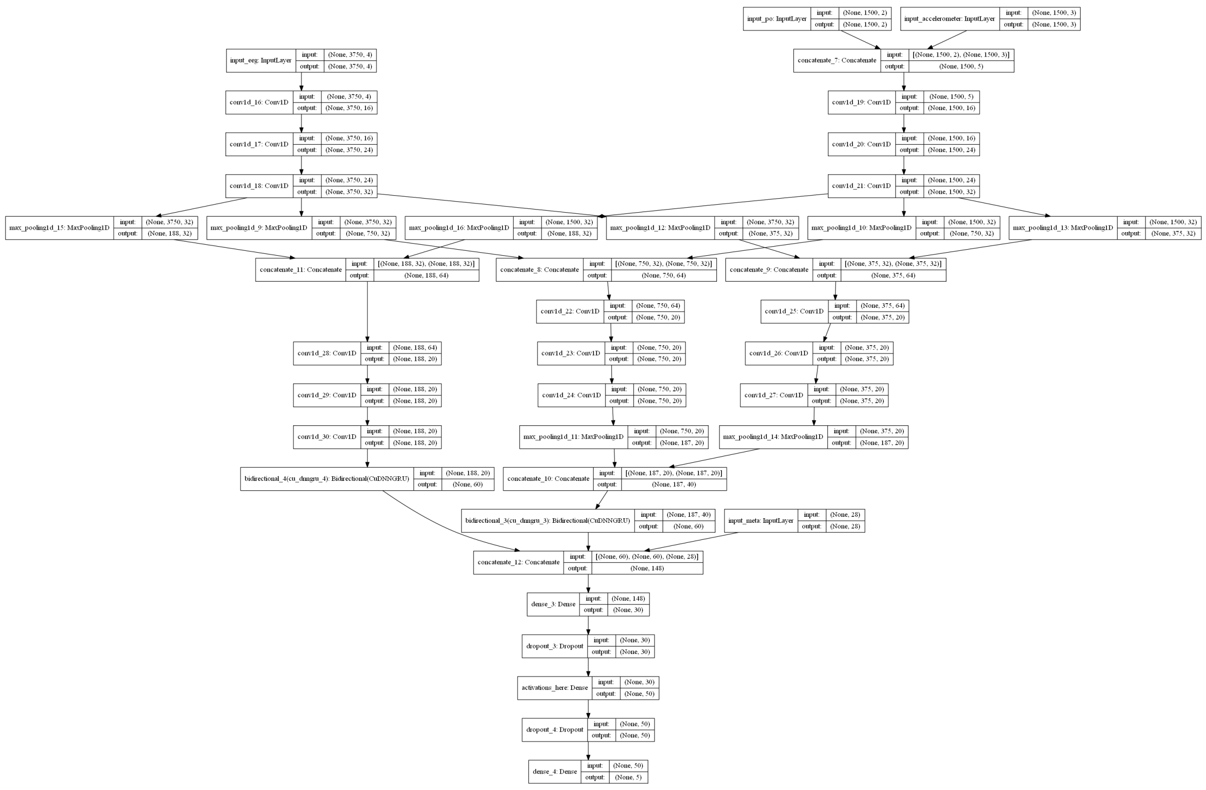
3.6 Multi-Scale CRNN with prior

The observations in the dataset are ordered, observations which are close in to another index-wise are likely to have the same class. From a biological points of view, this makes sense since the progression through sleep stages is sequential.One does not go directly from being awake to deep sleep. Hence, using the sleep stage of the previous and next observation is a strong predictor.
A logistic regression with the previous and next sleep stage as input had an accuracy of 89 %. However, in the test set, the real sleep stage of an observation is not available. hence we used the probability distribution estimated by the neural network to have an estimation of the sleep stage of the previous and next observation. To lower the impact of noise, these estimates were exponentially smoothed (half-life: 10).The estimates are then added to the network as an input.

Results:

- Validation accuracy: 0.81
- Training accuracy: 0.845
- Validation τ: 0.725
- Training τ: 0.75
- Public Leaderboard τ: 0.647578

Figure 3: Final neural network architecture.


input_po: InputLayer | input: (None, 1500, 2) | output: (None, 1500, 2)
input_accelerometer: InputLayer | input: (None, 1500, 3) | output: (None, 1500, 3)
input_eeg: InputLayer | input: (None, 3750, 4) | output: (None, 3750, 4)
concatenate_7: Concatenate | input: [(None, 1500, 2), (None, 1500, 3)] | output: (None, 1500, 5)
conv1d_16: Conv1D | input: (None, 3750, 4) | output: (None, 3750, 16)
conv1d_19: Conv1D | input: (None, 1500, 5) | output: (None, 1500, 16)
conv1d_17: Conv1D | input: (None, 3750, 16) | output: (None, 3750, 24)
conv1d_20: Conv1D | input: (None, 1500, 16) | output: (None, 1500, 24)
conv1d_18: Conv1D | input: (None, 3750, 24) | output: (None, 3750, 32)
conv1d_21: Conv1D | input: (None, 1500, 24) | output: (None, 1500, 32)
max_pooling1d_15: MaxPooling1D | input: (None, 3750, 32) | output: (None, 188, 32)
max_pooling1d_9: MaxPooling1D | input: (None, 3750, 32) | output: (None, 750, 32)
max_pooling1d_16: MaxPooling1D | input: (None, 1500, 32) | output: (None, 188, 32)
max_pooling1d_12: MaxPooling1D | input: (None, 3750, 32) | output: (None, 375, 32)
max_pooling1d_10: MaxPooling1D | input: (None, 1500, 32) | output: (None, 750, 32)
max_pooling1d_13: MaxPooling1D | input: (None, 1500, 32) | output: (None, 375, 32)
concatenate_11: Concatenate | input: [(None, 188, 32), (None, 188, 32)] | output: (None, 188, 64)
concatenate_8: Concatenate | input: [(None, 750, 32), (None, 750, 32)] | output: (None, 750, 64)
concatenate_9: Concatenate | input: [(None, 375, 32), (None, 375, 32)] | output: (None, 375, 64)
conv1d_22: Conv1D | input: (None, 750, 64) | output: (None, 750, 20)
conv1d_25: Conv1D | input: (None, 375, 64) | output: (None, 375, 20)
conv1d_28: Conv1D | input: (None, 188, 64) | output: (None, 188, 20)
conv1d_23: Conv1D | input: (None, 750, 20) | output: (None, 750, 20)
conv1d_26: Conv1D | input: (None, 375, 20) | output: (None, 375, 20)
conv1d_29: Conv1D | input: (None, 188, 20) | output: (None, 188, 20)
conv1d_24: Conv1D | input: (None, 750, 20) | output: (None, 750, 20)
conv1d_27: Conv1D | input: (None, 375, 20) | output: (None, 375, 20)
conv1d_30: Conv1D | input: (None, 188, 20) | output: (None, 188, 20)
max_pooling1d_11: MaxPooling1D | input: (None, 750, 20) | output: (None, 187, 20)
max_pooling1d_14: MaxPooling1D | input: (None, 375, 20) | output: (None, 187, 20)
bidirectional_4(cu_dnngru_4): Bidirectional(CuDNNGRU) | input: (None, 188, 20) | output: (None, 60)
concatenate_10: Concatenate | input: [(None, 187, 20), (None, 187, 20)] | output: (None, 187, 40)
bidirectional_3(cu_dnngru_3): Bidirectional(CuDNNGRU) | input: (None, 187, 40) | output: (None, 60)
input_meta: InputLayer | input: (None, 28) | output: (None, 28)
concatenate_12: Concatenate | input: [(None, 60), (None, 60), (None, 28)] | output: (None, 148)
dense_3: Dense | input: (None, 148) | output: (None, 30)
dropout_3: Dropout | input: (None, 30) | output: (None, 30)
activations_here: Dense | input: (None, 30) | output: (None, 50)
dropout_4: Dropout | input: (None, 50) | output: (None, 50)
dense_4: Dense | input: (None, 50) | output: (None, 5)

4 Final results

As of April 5 2018, we are ranked 2nd with a τ of 0.647578 obtained with the Multi-Scale CRNN with prior.