Smart alarm clock powered by Deep Learning

Artem Bachynskyi, Maksym Sukhorukov
for course Neural Networks (LTAT.02.001)

Abstract

This project deals with the problem of detecting the best sleep phase for the person to wake up, using a wrist-worn accelerometer. In order to solve the problem, various deep learning models, such as LSTM, RNN, CNN were used. As the result, a decent performance was achieved for both sleep action detection, and sleep phase detection.

1 Intro

The main goal of this project is to create a truly working alarm-clock for wrist-worn devices with help of deep learning. Initially, the idea is to to make a model(s) that will be capable to detect sleep stages and turn alarm-clock on when a user experiences the lightest dreams (first sleep stage) and can wake up with the least struggling, also, we will be able to count the length of the deep sleep to evaluate a quality of sleep.
As a source of training data, we will use this dataset from Siegen University which contains 42 labeled records of sleep from three-axis accelerometer. Considering methods, we are going to try CNNs (fixed horizon) or LSTMs (infinite horizon) and bi-directional LSTMs. Since the data set is unbalanced (check histograms below), we also will try to cope with this issue by applying some techniques.

As a result, we are expecting to create a system that will be detected suitable waking up stage. Testing of final results will be made by using self-recorded samples by wrist-worn ECG tracker.

2 Background

The problem of detecting human sleep phases is belonging to the class of human activity recognition field and is always challenging, because of the need for high accuracy and real-time performance. Human activity understanding encompasses activity recognition and activity pattern discovery [1]. As a typical human activity pattern recognition system, our system can be divided into several modules, sensing, segmentation, feature extraction, classification and post preprocessing [2]. For the sensing and segmentation, we should know, what are sleep stages of the human, and how do they impact on the human sleep. Over the course of a period of sleep, there are two main stages: NREM and REM (non-rapid and rapid eye movement respectively), which alternate cyclically. The function of alternations between these two types of sleep is not yet understood [3]. The NREM cycle contains 4 different stages: stage 1 (or light sleep), which is considered as the best to wake up,

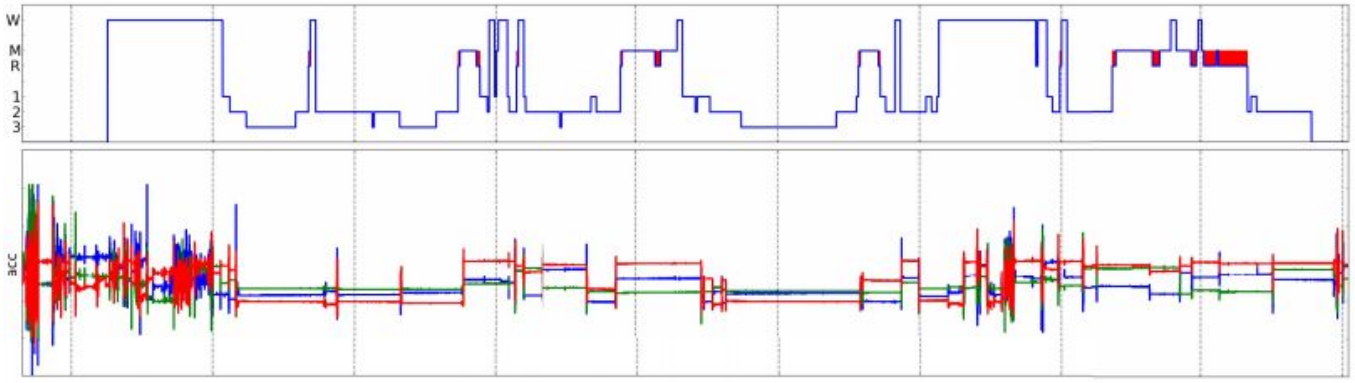

Figure 1 Example of human sleep stages (above) and corresponding accelerometer data (below)

and through our main goal to detect it correctly, stage 2 as a moderate stage or offset of sleep, and stage 3 and 4, which often referred as deep sleep - they are the hardest for a human to wake up in [4]. Using various filters and preprocessing techniques we can learn a good classifier, which would try to detect the first stage of NREM stages as accurate as possible, probably focusing less on
other stages.

3 Methods

The current project was split into two tasks. First, track a laydown event and start sleep detection after law activity for some time. The second task was aimed to detect exact sleep stage.
So, for the first task, it is urgent to firstly automatically detect when a user falls asleep, so that no additional actions from the user should be required and another system, which would track sleep phases would come to work.
During the daily time, we usually perform a lot of different actions, and it is pretty hard to distinguish between some of them, thus we decided to look for a dataset with the biggest amount of tracked actions in order to design a better system. For the task, we used publicly available dataset, collected by Laboratory for Ambient Intelligence and Mobile Robotics of University of Genova [5].
They have used recordings of 14 simple activities(brush_teeth, climb_stairs, comb_hair, descend_stairs, drink_glass, eat_meat, eat_soup, getup_bed, liedown_bed, pour_water, sitdown_chair, standup_chair, use_telephone, walk) of daily living by 16 different volunteers. The data was recording using a wrist-worn accelerometer, attached to the right wrist of the user across 3 axes. The output data rate of the described dataset is 32 Hz.
While it certainly possible to have a decent classifier without using a deep learning techniques, by applying a lot of feature extraction, pre-filtering and preprocessing the data and using classical machine learning algorithms, such as SVM, logistic regressions, trees and their assemblies, or even more complex, such as gradient boosting, state of the art for most problems nowadays are neural
networks, which in addition to better accuracy does not require a lot of feature generation and preprocessing of the data.
For solving this problem we decided to create an LSTM neural network, since we are dealing with a sequential output. For the classical Neural Networks, as well as convolutional neural networks problem is that their API allows accepting only a fixed-sized vector as input (e.g. an image) and outputs a fixed-sized vector (e.g. probabilities of different classes). Besides that, these models perform this mapping using a fixed amount of computational steps (e.g. the number of layers in the model). The core reason that recurrent nets are more suitable for a lot of tasks is that they allow us to operate over sequences of vectors: Sequences in the input, or in the output. We use “many-to-one” architecture, as described in an article by A. Karpathy [6], so that we would accept time series of feature vectors (3-axial signal) and output a single probability vector for each action based in this sequence of vectors. We use TensorFlow library in order to create a model, since we are both familiar with it on a decent level.
The architecture of model used is following: it is two stacked LSTM cells with 32 hidden layers each, with linear activation function. We use a batch size of 1500, as well as softmax loss with L2 regularization in order to prevent overfitting and Adam optimizer with learning_rate of 0.0025. We iterate our network 300 times on the training data length. We also initialize weights and biases randomly from normal distribution.
We also had to resample data to fit our wrist-worn accelerometer data rate (32 to 26 Hz).
This architecture of model allowed us quite fast computations (up to ten minutes of training) and gives decent results: accuracy about 88%, which can be considered as really good within 14 classes for this particular problem.
For the second part of the project, training and test data were sampled from records

using sliding windows (of size 60 seconds or 30 seconds) with stride 10 seconds. So, 3250, 3250, 3756, 2864 train samples were obtained for 1-st, 2-nd, 3-rd and REM stages correspondingly. Moreover, another 750 elements of each class considered as test data. To balance train set imblearn package was used. So, we trained our model on 15024 samples and validated on 3000 samples.
Because of quite a big sequence size (6000 or 3000 points, because of 100 Hz sampling rate), so it was impossible to feed this data through LSTM or GRU. However, topologies with Pooling layers were applied in the beginning of models to reduce dimensionality, afterward, LSTM, CNN, GRU, Bidirectional LSTM, CNN (3 inception layers described on Figure 2) + LSTM. All models were implemented using Keras framework[7].

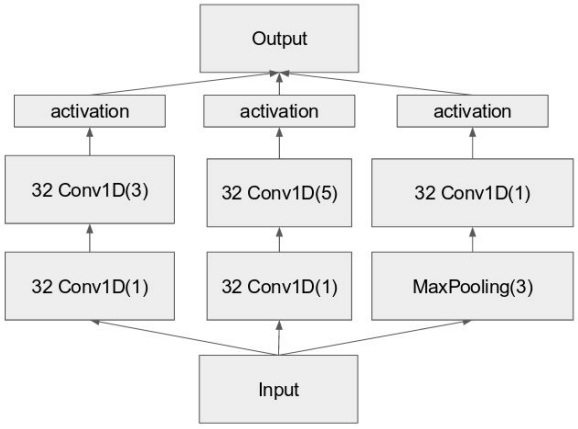

Figure 2 Inception unit structure

4 Results

During the project various models were tried, their topologies and performance after fine-tuning are shown in Table 1:

Table 1 Performances of the considered models

	Model	Validation accuracy
1	Average Pooling 5 3x(Inception + Max Pooling 2) Dense 256 Batch Normalization Leaky ReLu Dense 64 Batch Normalization Leaky ReLu Softmax 4	46.63%
2	Average Pooling 10 GRU 300 Dense 256 Batch Normalization Leaky ReLu Dense 64 Batch Normalization Leaky ReLu Softmax 4	42.17%
3	Average Pooling 3 3x(Inception + Max Pooling 3) LSTM 300 Dropout 0.2 Dense 400 Batch Normalization Leaky ReLu Dropout 0.2 Dense 250 Batch Normalization Leaky ReLu Dropout 0.2 Softmax 4	50.17%
4	Max Pooling 2 Bidirectional LSTM 300 Dropout 0.2 Dense 300 Batch Normalization Leaky ReLu Dropout 0.2 Softmax 4	39.03%
5	Max Pooling 2	37.21%

	LSTM 300 Dropout 0.2 Dense 300 Batch Normalization Leaky ReLu Dropout 0.2 Softmax 4	

For the best model (3rd in Table 1) confusion matrix shown in Figure 3 was obtained. According to this matrix, second sleep stage is indistinguishable and dramatically mixed up with the third stage.

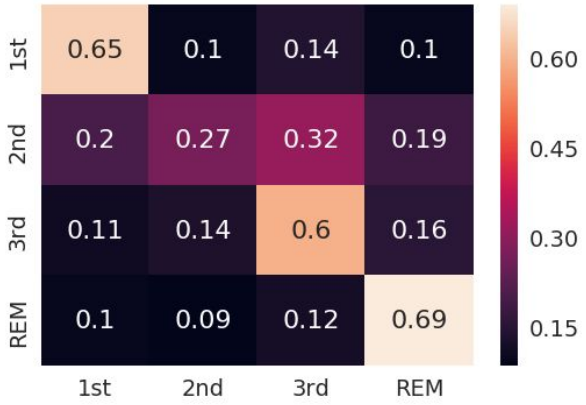

Figure 3 Confusion matrix of best (third) model

Thus, we decided to remove 2nd class and train the same using three classes. In this case accuracy (obviously) increased to 65.3%, confusion matrix has a structure shown in Figure 4.

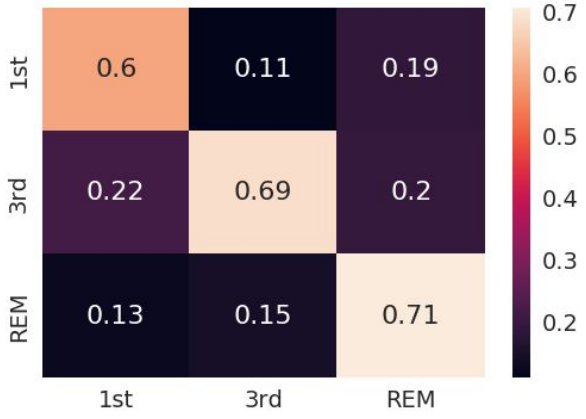

Figure 4 Confusion matrix of best (third) model learned on three classes

So this is the best model we were capable to obtain using declared methods.

5 Discussion

After a presentation, I started to study this topic more deeply and decided to mimic an approach to existing wristband. It's quite straightforward, we consider a segment of deep sleep when there is almost no activity on it and light sleep in another case. To detect it, standard deviation should be calculated for some window. As shown in Figure 5 and Figure 6 results from my wristband and Chinese product, respectively, I obtained quite close results (higher columns stands for light sleep and lower, for deep). However, this approach totally failed on the initial dataset. I have several assumptions, why it happened. Firstly, a "first-night syndrome" is a big problem for sleeping labs, subject experience discomfort because of new sleeping place and surrounding of various sensors, so the data might be corrupted. Secondly, given dataset contains people with sleep disorders, so their behavior during sleep (actigraphy) might be distorted.

Figure 5 Sleep stages based on industry device

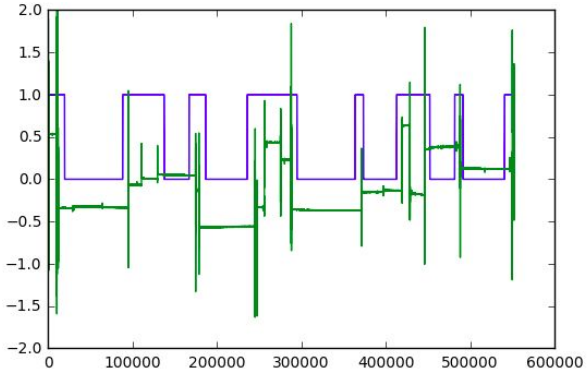

Figure 6 Sleep stages based on alternative naive approach

6 Contribution

Since a project has two logical parts (“go to bed” event, sleep stage detection), they were shared between authors. Maksym Sukhorukov was responsible for the first part, whereas Artem Bachynskyi worked on sleep stage monitoring. Also, we collaboratively created project documentation, performed paper search and discussed methods.

7 Conclusions

In this project, we aimed to create a complete system of alarm-clock, on the base of accelerometer signal input based on deep learning models and we have achieved a satisfying performance of the developed algorithms.

Helping human with such an important problem, which all of us deal with on the daily basis, as sleeping and waking up in the best mood without any struggle was a great project to work on. A perfection of the system could largely simplify our lives and lead to more discoveries in sleeping phase detection, as well as human activity recognition.

For both parts of the project: an action of falling asleep detection and sleep phase detection the best model turned out to be a LSTM, which is logical, considered sequential nature of the input. We have achieved decent accuracies of 88% and 65% respectively.

The results could be improved by some margin, using better equipment, collecting more data for more participants to fight with overfitting while modeling as well as “the first-night syndrome” while collecting data. We can also try more complex and deep LSTM models if we would have bigger datasets.

References

[1]Kim E, Helal S, Cook D. Human activity recognition and pattern discovery http://ieeexplore.ieee.org/document/5370804/

[2]Yuqing Chen, Yang Xue A Deep Learning Approach to Human Activity Recognition Based on Single Accelerometer http://ieeexplore.ieee.org/document/7379395/

[3] Sleep Disorders and Sleep Deprivation: An Unmet Public Health Problem https://www.ncbi.nlm.nih.gov/pubmed/20669438

[4] National Sleep Foundation https://sleepfoundation.org/how-sleep-works/what-happens-when-you-sleep

[5] https://archive.ics.uci.edu/ml/datasets/Dataset+for+ADL+Recognition+with+Wrist-worn+Accelerometer

[6] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[7] https://keras.io/