Grasp and Lift EEG Detection Project Documentation

Vlastimil Martinek, David Čechák

1. Introduction

This project compares various machine-learning models in terms of viability for EEG lift-and-grasp kaggle competition, which took place in 2015. This competition challenged its participants to identify, when a hand is grasping, lifting or replacing an object using EEG data taken from subjects performing those activities.

Data consists of 32 EEG channels and there are 6 events to identify. Data has been gathered from 12 subjects.

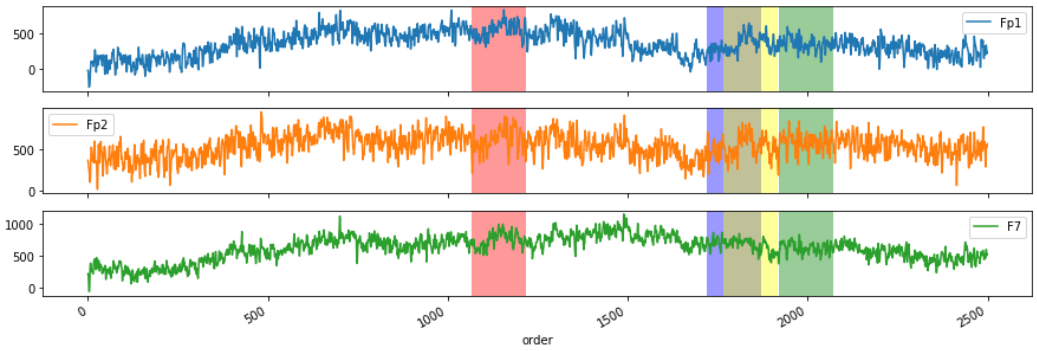

Figure 1: EEG signals with highlighted events

2. Existing implementations

Many people submitted their solution to the competition. Various models were used, including logistic regression, convolutional neural networks and ensemble models.

1st place - 98.1 % success rate

Best submitted solution used 3 levels of models. First level consisted of cca 50 different, event-specific models, including cnn, rnn and logistic regression. Second level models were trained on predictions from first level models. They were trained on all subjects combined. Used algorithms include rnn, mlp and xgboost. Third level consisted of weighted means of second level predictions. Full documentation can be found here.

2nd place - 98 % success rate

Second-best solution used recurrent convolutional neural networks. Full documentation can be found here.

3. Implementation and success rates

Implementation was done in a python notebook. Tested models were Logistic regression, cnn and rnn.

Signal preprocessing

Input signals have various ranges, therefore features are standartized by removing the mean and scaling to unit variance.

Success rate measuring

Since we are predicting 6 different events, internal success rate is measured for each event separately and then averaged. For each event, success is, when predicted value (eg. 0.24542) rounds to expected value (1 or 0).

Kaggle success rate is measured by submitting the predictions to kaggle server which evaluates them.

Models

Logistic regression

First model used was logistic regression. Scikit framework was used for LR model implementation. This model was trained on each subject and each label separately.

Since input data are sparse (there are a lot more non-events than events), next LR model was trained in balanced mode. This mode uses the values of labels to automatically adjust weights inversely proportional to class frequencies in the input data.

Basic LR - total/events	**Balanced LR - total/events**
95.5% / 0.9%	72.1% / 69.4%

Table 1: LR Internal success rates

Basic LR	**Balanced LR**
73.3%	73.2%

Table 2: LR Kaggle success rates

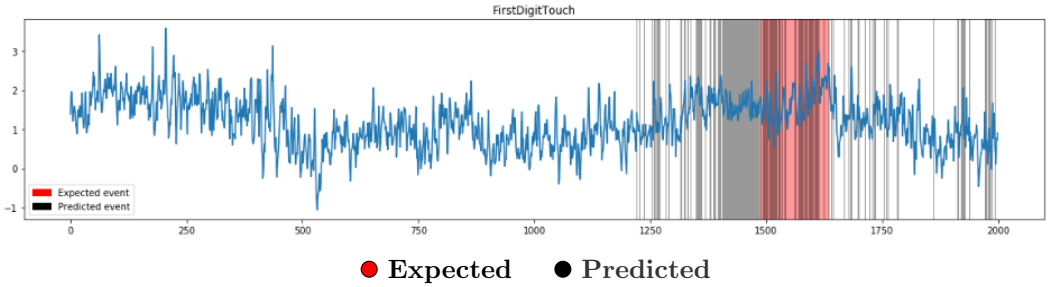

Figure 2: Basic LR event predictions

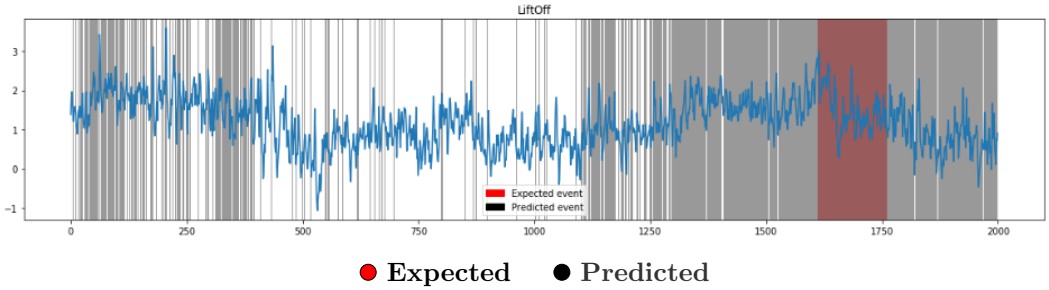

Figure 3: Balance LR event predictions

Recurrent neural network

Second model used was recurrent neural network. Keras framework was used for implementation.

Two RNN models were used - basic and stacked.

Basic model layers

$$Input \rightarrow LSTM \rightarrow Activation$$

Stacked model layers

$$Input \rightarrow LSTM \rightarrow LSTM(stacked) \rightarrow Activation$$

Models were trained and tested on multiple variations of data. Training and testing was done *on each subject separately* **or** *on all subjects at once*. Since there are multiple events, models were also trained *on each label separately* **or** *on all labels at once*.

	Single labels - total/events	**All labels - total/events**
Single subject	96% / 28.9%	96% / 31.2%
All subjects	94.7%/16.2%	95.3%/13.5%

Table 3: Internal Basic RNN model success rates

	Single labels - total/events	**All labels - total/events**
Single subject	96%/28.3%	96.1%/31.8%
All subjects	95.8%/17.7%	94.8%/14%

Table 4: Internal Stacked RNN model success rates

Basic model	**Stacked model**
89.2%	88.7%

Table 5: Kaggle RNN all-labels models success rates

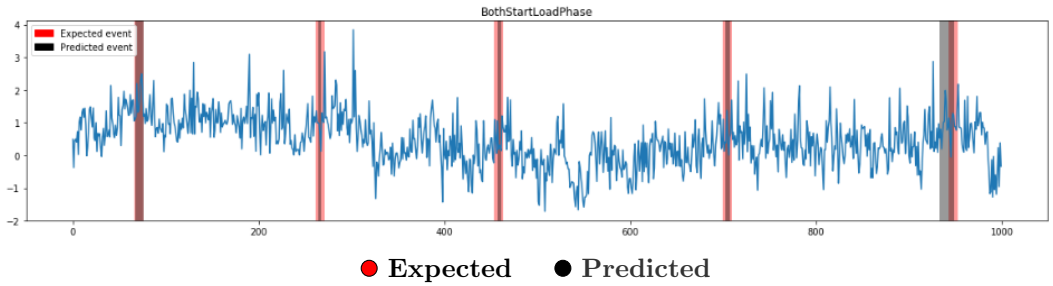

Figure 4: Basic RNN model event predictions

One of the basic RNN models is the most successful among models from this project, reaching 89.2% official kaggle success rate and would place 128th/379 on the competition leaderboards (competition is already closed).

The basic all-labels RNN model was further used for testing a hypothesis, that training the model on one person and testing on other in significantly less successful in predicting events than testing on the trained person.

vertical = trained subject, horizontal = tested subject

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	**55%**	19%	2%	28%	3%	9%	9%	8%	15%	31%	12%	4%
2	6%	**29%**	1%	6%	1%	6%	9%	5%	7%	12%	10%	2%
3	5%	3%	**18%**	7%	5%	5%	3%	6%	4%	5%	1%	0%
4	19%	15%	1%	**36%**	2%	8%	10%	10%	23%	33%	12%	5%
5	3%	3%	1%	2%	**5%**	2%	6%	3%	2%	10%	1%	2%
6	13%	17%	3%	17%	3%	**27%**	22%	6%	24%	29%	15%	5%
7	12%	19%	2%	11%	0%	13%	**40%**	4%	15%	18%	14%	4%
8	6%	5%	2%	11%	1%	4%	6%	**21%**	7%	11%	6%	1%
9	11%	24%	1%	10%	2%	12%	20%	6%	**36%**	22%	20%	3%
10	9%	20%	1%	14%	2%	9%	17%	3%	10%	**39%**	9%	5%
11	9%	18%	0%	7%	1%	6%	11%	4%	17%	11%	**28%**	5%
12	6%	12%	3%	5%	2%	4%	12%	5%	16%	9%	18%	**13%**

Table 6: Internal relative success rates for event prediction

On average, testing on trained person resulted in **28.9%** success rate for event prediction, while testing on different person resulted in **8.7 %** success rate.

Convolutional neural network

Third model was a convolutional neural network. It was developed using Keras library on top of TensorFlow.

Different more architectures were tested. Simple CNN single-convolution-layer model with following parameters was performing the best.

There are 8 series of experiments for a subject. The subject is performing the same activities in each experiment. The number of epochs mentioned in *parameters* is repeated for each series out of 7 (1 series for internal testing). Therefore, f.e. 5 training epochs on each out of 7 series results in 35 total training epochs. For training on all subjects, is the number of epochs repeated for each 7 series on each out of 12 subjects.

CNN single-convolution-layer

Input (lookback, EEG channels) therefore (10, 32)
↓
2D Convolution layer, 64 filters, (3,1) kernel
↓
(2,1) Max pooling
↓
Softmax activation layer

training epochs	**5**
batch size	**128**
downsampling	20
look back [1]	10
hidden layers activation	relu (single-label) / sigmoid (6 labels)
last layer activation	softsign (single-label) / softmax (6 labels)
loss function	mean squared error

Table 7: CNN single-convolution-layer model parameters

The best performing parameters out of different models with structure 'CNN single-convolution-layer + dense layer' is:

CNN single-convolution-layer + dense layer

Input (lookback, EEG channels) therefore (10, 32)
↓
2D Convolution layer, 192 filters, (3,1) kernel
↓
(2,1) Max pooling
↓
Dense layer, 128 neurons
↓
Softmax activation layer

Other models with multiple layers did not manage to capture any event. In other words, they predicted zeroes for all test inputs.

Models were trained and tested on multiple variations of data. Training and testing was done *on each subject separately* **or** *on all subjects at once.*

training epochs	**5**
batch size	**128**
downsampling	20
look back [2]	10
hidden layers activation	relu (single-label) / sigmoid (6 labels)
last layer activation	softsign (single-label) / softmax (6 labels)
loss function	mean squared error

Table 8: CNN single-convolution-layer + dense layer

	Single labels - total/events	**All labels - total/events**
Single subject	79.7% / 22.0%	75.1% / 34.1%
All subjects	78.5% / 4.5%	77.6% / 18.0%

Table 9: Internal CNN single-convolution-layer model success rates

	Single labels - total/events	**All labels - total/events**
Single subject	78.7% / 36.3%	81.8% / 33.6%
All subjects	78.5% / 4.5%	76.4% / 9.1%

Table 10: CNN single-convolution-layer + dense layer

CNN single-convolution-layer + dense layer model
64.7%

Table 11: Kaggle CNN models success rates

The best performing CNN model reached 64.7% official kaggle success rate.

The best performing (out of CNNs) all-labels CNN model was also used for experiment of training the model on one person and testing on others. This resulted in significantly less success rate in predicting events than testing on the trained person.

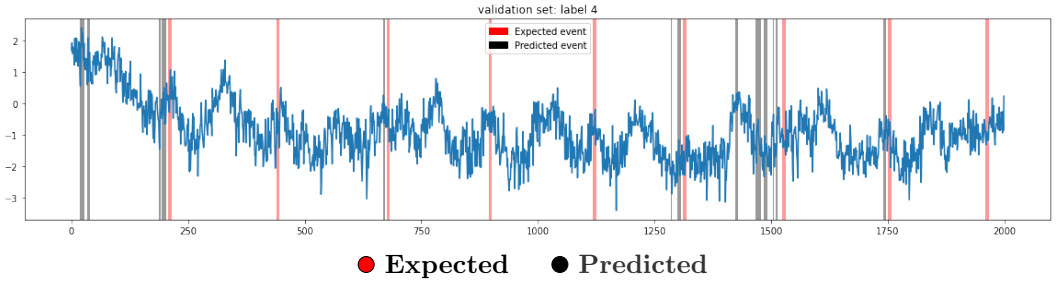

Figure 5: CNN single-convolution-layer + dense layer

On average, testing on trained person resulted in **34.7%** success rate for event prediction, while testing on different person resulted in **17.4%** success rate. This is our success rate for event prediction - it counts only the ratio of "events predicted correctly / total events". It does not take into account false positives or the total accuracy. The total accuracy of a model is showed in Table 9 and Table 10. In conclusion, the CNN has a better result in true positives but also a higher count of false positives. Therefore, its success rate is lower than the one of the RNN.

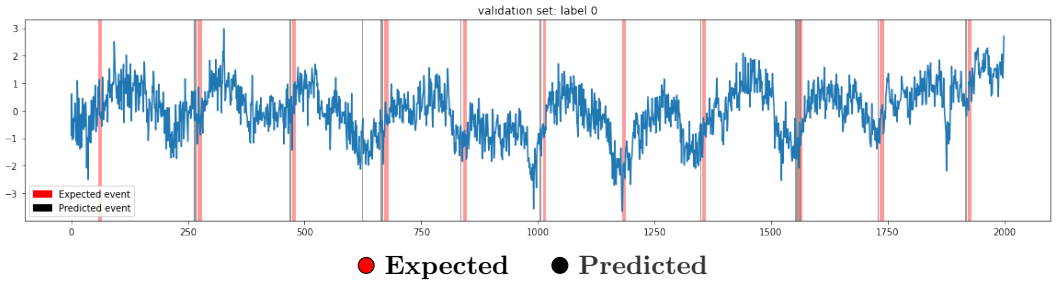

Figure 6: CNN single-convolution-layer + dense layer

vertical = trained subject, horizontal = tested subject

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1	**45%**	16%	18%	23%	12%	21%	19%	20%	23%	13%	24%	18%
2	23%	**39%**	16%	22%	12%	30%	28%	15%	28%	23%	26%	23%
3	18%	12%	**26%**	9%	13%	13%	16%	16%	15%	13%	14%	12%
4	19%	18%	17%	**37%**	13%	22%	19%	17%	21%	13%	25%	14%
5	15%	13%	18%	11%	**21%**	15%	16%	16%	17%	14%	14%	11%
6	16%	13%	12%	12%	9%	**42%**	24%	17%	19%	11%	16%	15%
7	16%	16%	17%	16%	15%	26%	**42%**	18%	25%	15%	20%	21%
8	19%	17%	18%	12%	13%	17%	14%	**34%**	23%	15%	13%	15%
9	13%	14%	10%	22%	11%	26%	21%	16%	**36%**	14%	24%	17%
10	22%	26%	18%	31%	19%	30%	24%	15%	30%	**39%**	27%	22%
11	20%	15%	13%	21%	7%	24%	19%	18%	23%	16%	**30%**	19%
12	12%	15%	14%	12%	12%	19%	13%	15%	22%	11%	17%	**25%**

Table 12: Internal relative success rates for event prediction

Conclusion The RNN has a success rate of 96% with 28.9% of correctly predicted events and the CNN has the success rate of 81.8% with 33.6% of correctly predicted events. One of the reasons for RNN being better might be that CNN processes predefined shape of data and searches in it for specific patterns based on predefined kernels. On the other hand, RNN process arbitrary sequences of inputs and are better for stream analysis. Since this dataset consists of streams, CNN has a disadvantage. When we try to cover longer sequence ("history") by increasing the size of the CNN input using the parameter "look back", than the computational complexity grows. EGG channels have each its own separate meaning and their composition (the way they are placed next to each other to form a two-dimensional matrix) might not correspond well to the way in which they behave during events in this experiment. A CNN filter which considers a point and its neighborhood is not suitable for detection of relationships of channels distant from each other. An interesting experiment would be to find out if the CNN would perform better if the inputs would be transformed into spectrogram.

[2]length of history of one record, size of the time dimension

4. Running the code

The documented code is available as a python notebook.

Logistic regression and RNN (Vlastimil Martinek)

CNN - *UI_Project_CNN_colab.ipynb* (David Čechák)