Towards Turnkey Brain-Computer Interfaces

Definition

Project overview

An electroencephalogram[1] (EEG) is a recording of the electrical activity of the brain using electrodes placed on the scalp. Signals from EEGs can be interpreted in the frequency domain (neural oscillations[2]) or in the time domain (event-related potentials[3] or ERPs). These signals are highly subject-specific, depend on experimental conditions and vary between recording sessions. For this reason, most Brain-Computer interfaces[4][5] (BCIs) require a calibration phase, during which data is collected in order to train a subject-specific classifier. This calibration phase is often perceived as an hindrance, and can be difficult to complete for subjects with a limited attention span.

Problem statement

In this study, we focused on the problem of transfer learning in ERP-based BCIs. There are many kinds of ERPs, and we concentrated our efforts on the detection of the P300[6][7] evoked potential, a type of neural response elicited when a subject pays special attention to a stimulus[8].

Specifically, we attempted to build a generic model that works reasonably well across different subjects and attention tasks. The resulting classifier performs on a single-trial input, that is, a matrix of shape (*n_samples*, *n_electrodes*), and produces a binary output (*target stimulus* or *distractor*).

Dataset

The dataset is publicly available[9] and published under the Creative Commons Zero license[10]. The data was acquired to investigate whether neural activity detected by ERP-based BCIs is task specific (Wenzel, Almeida, and Blankertz, 2016)[11].

It was collected from 13 subjects in 3 different experimental conditions. For each participant and condition, 840 trials were recorded. About 20% of the trials refer to the target stimulus.

Metrics

The Area Under the Receiver Operating Characteristic Curve[12] (AUROC) is particularly well suited for unbalanced binary sets. It was used as the final evaluation metric. A score of 1 represents a perfect result while a score of 0.5 is equivalent to chance.

Analysis

Data exploration

The dataset can be imported as a multi-indexed dataframe. It contains 3,173,100 lines and 64 columns. A sample is provided in *Figure 1*.

			Signal	Target	EOG	Fp1	…	O2
Condition	Participant	Epoch	Time					
A	0	0	-00:00:00.100	0	5.731808	4.032938	…	-1.808079
			-00:00:00.090	0	4.015707	2.663035	…	1.452488
			…	…	…	…	…	…
			-00:00:00.010	0	-0.219369	2.072680	…	-3.999745
			00:00:00	0	3.401393	-2.750054	…	1.573483
			00:00:00.010	0	3.674389	-4.733631	…	4.730165
			…	…	…	…	…	…
			00:00:00.880	0	23.840640	3.220363	…	0.096679
			00:00:00.890	0	24.267097	1.859800	…	-1.860631

FIGURE 1: DATA SAMPLE

The 4 indices provide the following information:

- **Condition**: The attention task (A, B, or C).
- **Participant**: The subject ID (0 to 12).
- **Epoch**: The trial number.
- **Time**: The sample time, relative to the stimulus onset. We can see that a trial starts 0.1 second before the stimulus and ends 0.9 second after. A full epoch has a duration of one second, and contains 100 samples. From this, we can determine the sample rate: 100Hz.

The 64 columns can be described as follow:

- **Target**: Distractor (0) or Target (1)
- **EOG**[13]: The electrical potential at a special electrode positioned below the eye. It is used for detecting eye movements, which can cause artifacts in the EEG data.
- **Fp1** to **O2** (62 columns): The electrical potential, in μV, measured at each EEG electrode. The column names refer to the positions of the electrodes, as defined in the 10-20 system[14].

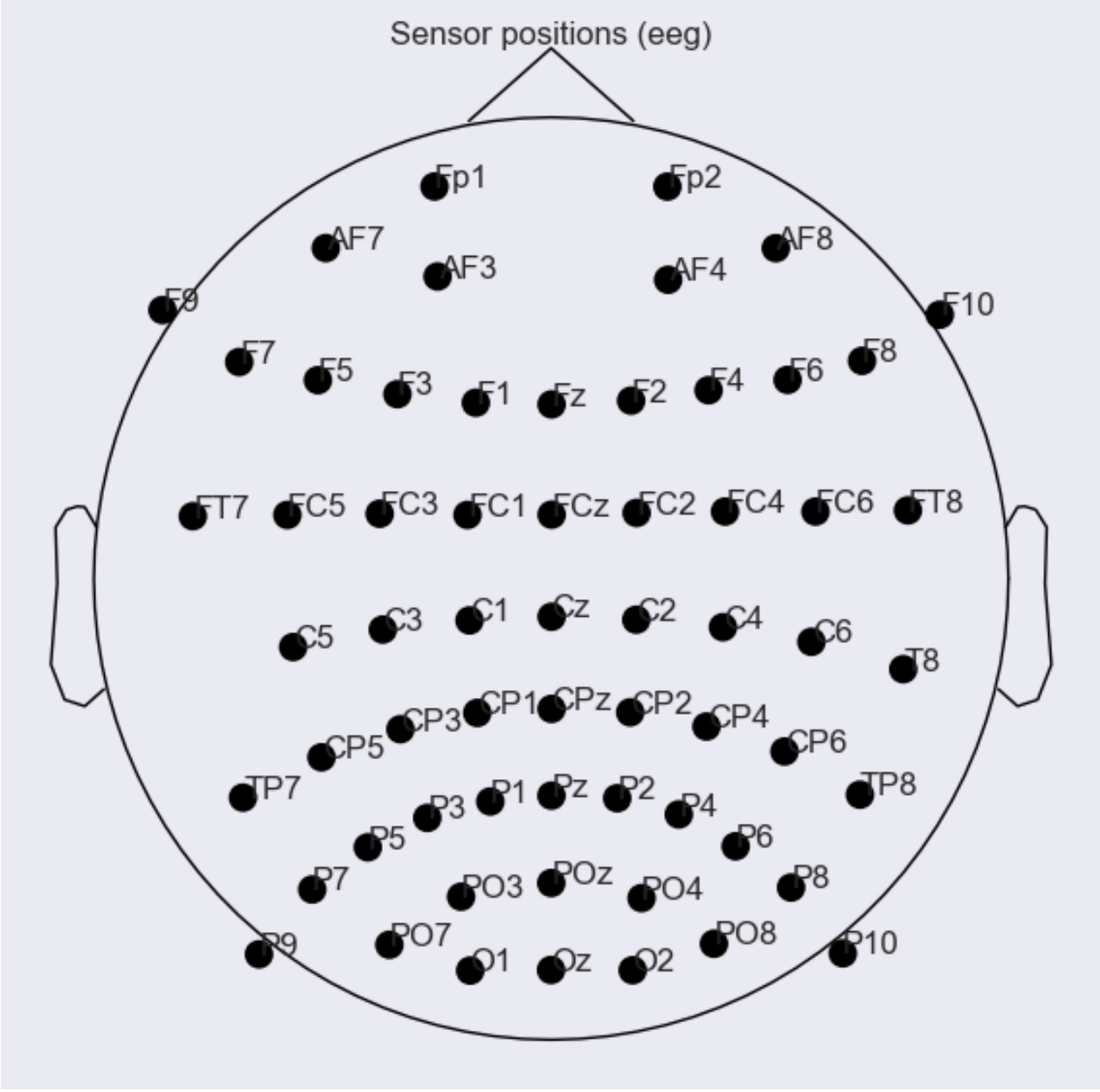


FIGURE 2: MONTAGE

There are 840 trials for each condition and participant. The target stimulus accounts for about 20% of the trials.

Exploratory visualization

Figure 3 shows the raw data for condition C and participant 0. Here, only one third of the electrodes and the first 20 trials are displayed. Distractors are marked as 1 and targets as 2. We can see that the data is noisy, with sudden departures from the mean. At first glance, there is no obvious distinction between targets and distractors.

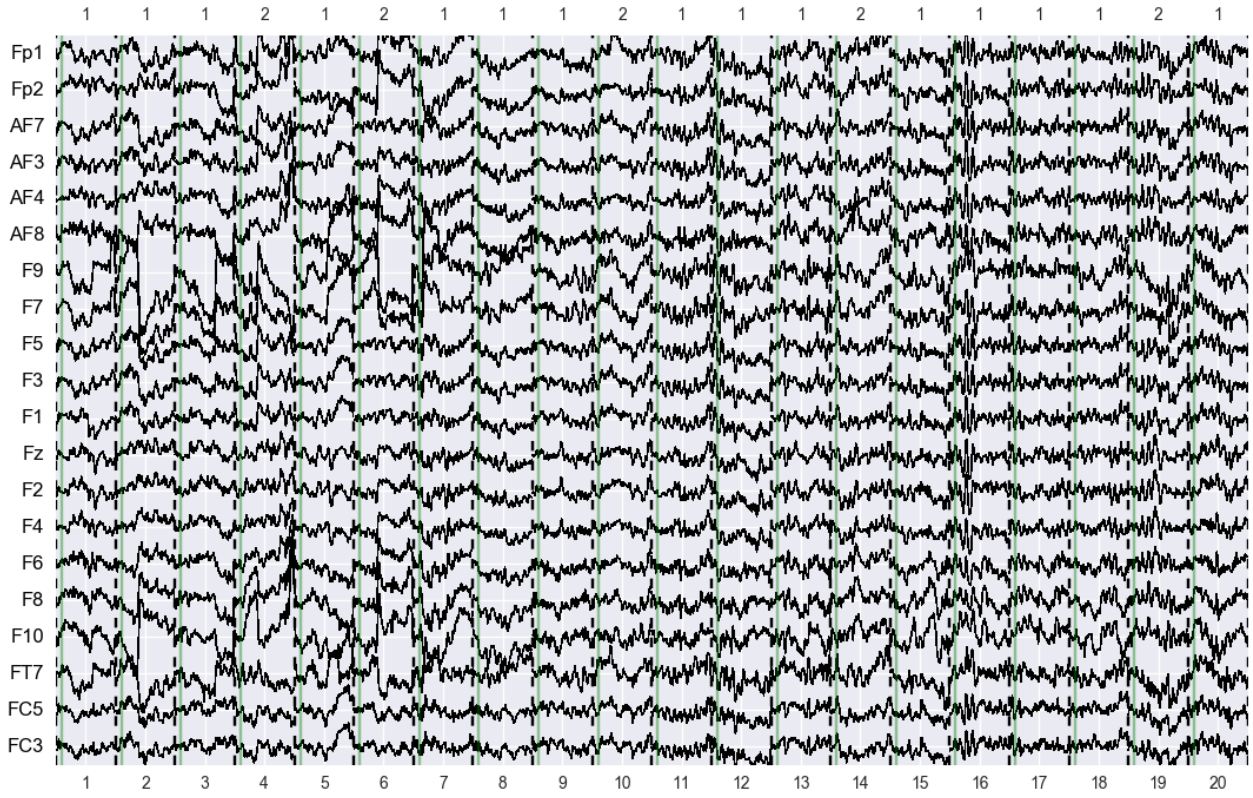


FIGURE 3: RAW DATA VISUALIZATION

However, if we average the signals for each class and electrode, patterns emerge. *Figure 4* shows a clear P300 ERP in the occipital lobe.

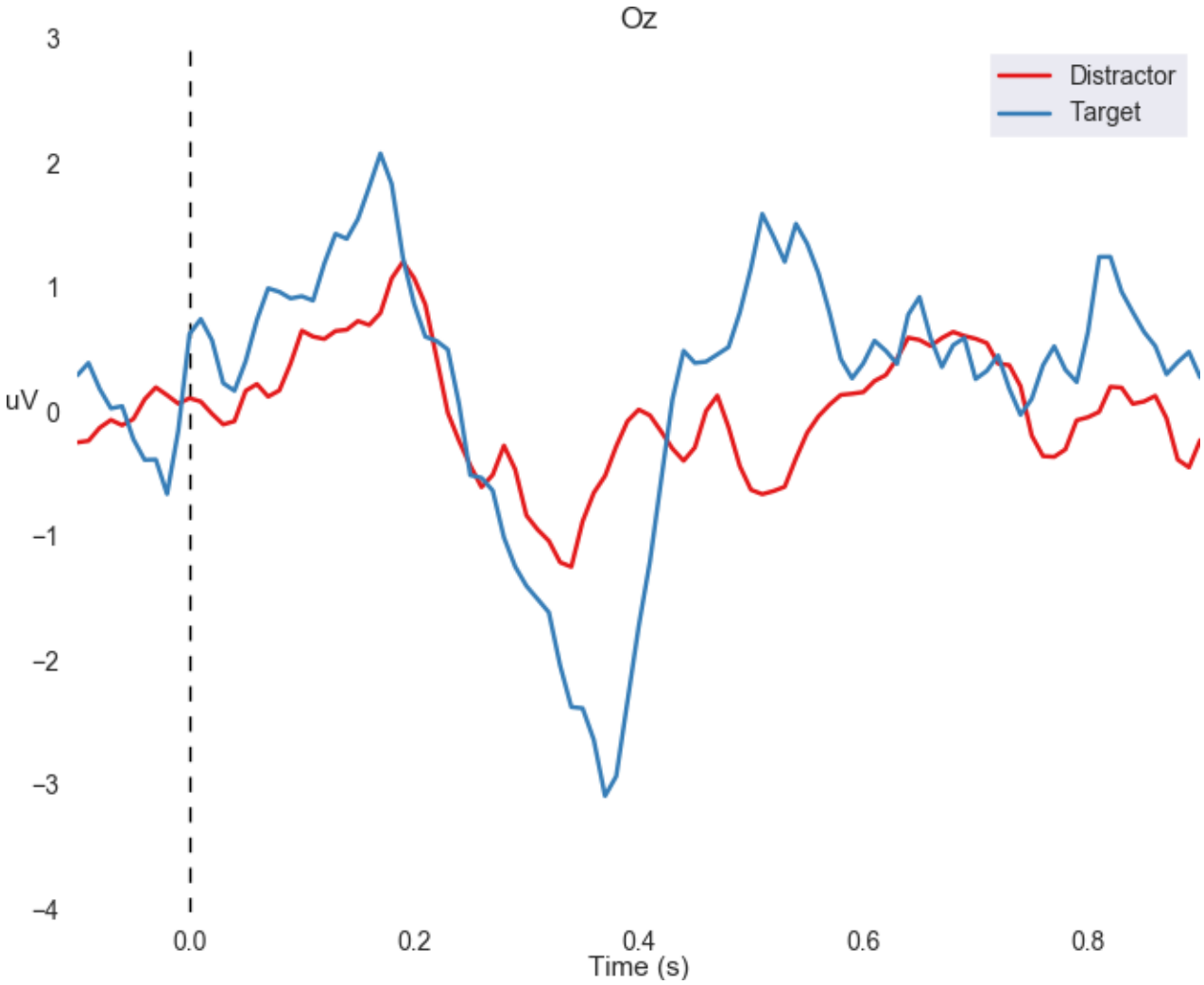


FIGURE 4: EVOKED DATA RECORDED AT ELECTRODE OZ

Algorithms and techniques

Since the EEG data is actually a time series, we decided to use a Long Short-Term Memory[15] (LSTM) deep neural network. This is a sequence classification problem, which can be addressed with a many-to-one prediction model. LSTMs overcome the challenge of vanishing and exploding gradients faced in Recurrent Neural Networks (RNNs), maintain an internal state over time, are resistant to noise, and can learn both linear and non-linear relationships.

In traditional Feed Forward Networks (FFNs), information flows from one layer to the next. They are stateless: they have no memory of previous inputs. RNNs introduce loops in the network. By allowing outputs to feedback as inputs, information can persist. This makes RNNs suitable for time series prediction problems. However, RNNs are hard to train, because back propagated errors can either grow or decay exponentially, and cause the network to stall (very small weight updates) or to overflow (very large updates). LSTMs are a particular implementation of RNNs that try to solve this

issue. In addition to input and output weights, each memory cell (the equivalent of the neuron in FFNs) contains an internal state that is maintained by three sigmoid layers called gates. They determine what information is allowed to enter, to leave or should be discarded. They are the *Input*, *Output*, and *Forget* gates. They stabilize the cell, thus producing a constant error flow.

We evaluated different LSTM architectures by testing several parameters on a subset of the data: number of layers, units per layer, loss function correction with class weight, batch size, dropout and recurrent dropout, early stopping rule, statefulness.

To allow a direct comparison with the results of the original paper, we trained one model per condition pair, excluding the data from one subject that was kept for evaluation. Therefore, for 3 conditions and 13 subjects, a total of 117 models were built. AUROC scores were averaged for each condition pair. This variation of the leave-one-out cross-validation method ensured that the evaluation set was not contaminated by partial sessions that could have been seen during the training stage.

Benchmark

As a baseline, we replicated the results of the original study. We implemented the *de facto* method described in (Blankertz *et al.*, 2011)[16], which is is usually quite effective for the classification of ERP components. It basically involves two steps : downsampling the EEG signal to reduce the dimensionality of the features, and training a regularized Linear Discriminant Analysis (LDA) estimator.

Our results closely match those from the original paper (*Figure 5* and *Figure 6*), with a grand mean AUC score of 0.61.

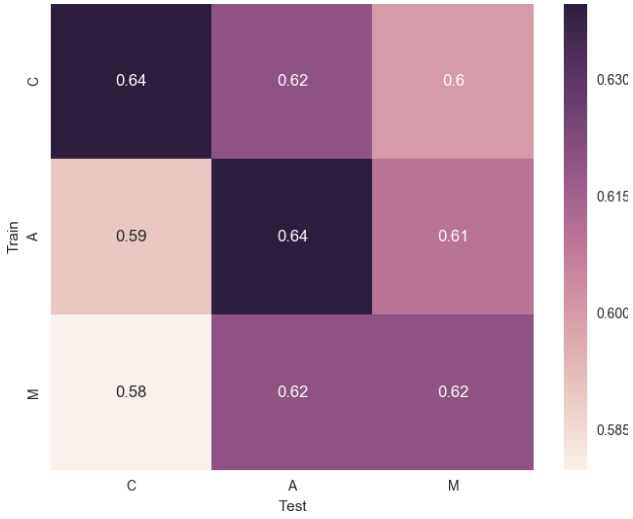


FIGURE 5: AUC FOR EACH CONDITION PAIR (BENCHMARK)

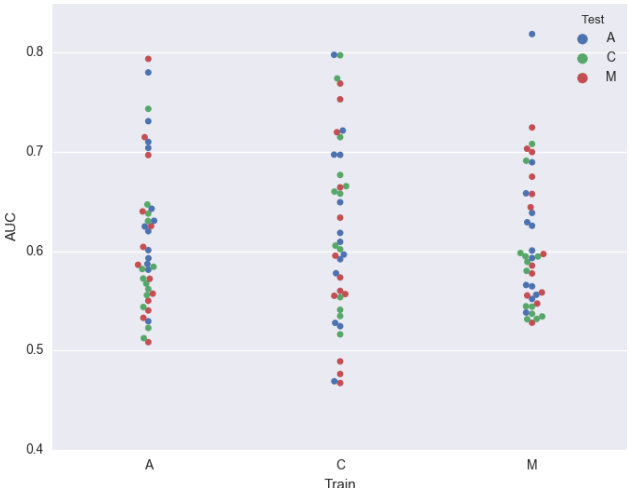


FIGURE 6: AUC FOR EACH PARTICIPANT (BENCHMARK)

Methodology

Toolset

The following tools were used in this study:

- Python 2.7
- MNE[17]: EEG and MEG analysis and visualization
- scikit-learn: machine learning library
- Keras: deep learning library
- TensorFlow: numerical computation using data flow graphs
- Seaborn: statistical data visualization

Data preprocessing

The dataset was modified in several ways:

- Conversion to MNE objects (specifically *Raw*, *Epochs* and *Evoked* objects)
- Application of a series of transformations:
  - Bandpass filtering: to remove the DC offset and the AC noise.
  - Downsampling: to reduce the dimensionality of the features.
  - Removal of outliers: by convention, it is considered that anything above 100 μV is an artifact, but since we want to build a noise-resistant model, outliers were not removed.
  - Baseline correction: mean signal amplitude at each electrode can vary considerably between and during sessions. For this reason the mean of a few samples before the stimulus was subtracted from the main signal for each trial.
  - Cropping: during data recording, the signal is transmitted wirelessly, and in practice there often is some lag. Each trial was cropped to minimize information unrelated to the stimulus.
  - Standardization: data for each electrode was centered and scaled.

Only the EEG channels were kept, the EOG channel was discarded.

Implementation

The network architecture is shown in *Figure 7*. It consists of the input layer, three stacked LSTM layers, one fully-connected layer, and one sigmoid activation layer.

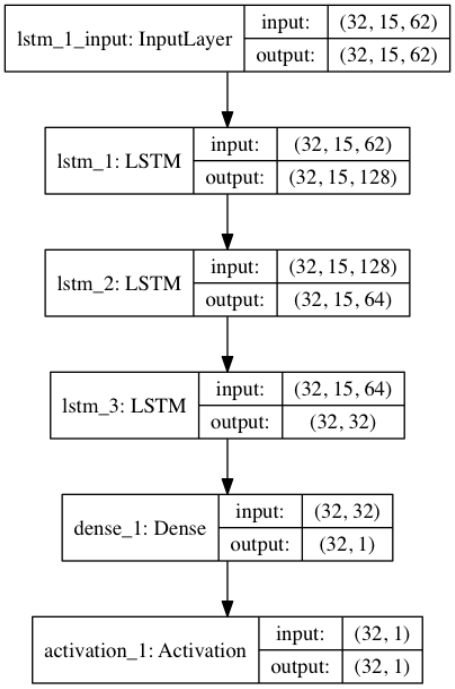


FIGURE 7: NETWORK ARCHITECTURE

To reduce overfitting, a 40% dropout on both inputs and recurrent states was applied after each LSTM layer.

An Adam[18] optimizer was used for the stochastic gradient descent algorithm.

This architecture can be translated in code as follow:

```
>>> model = Sequential()
>>> model.add(LSTM(
>>>   128,
>>>   batch_size=batch_size,
>>>   input_shape=(X_train.shape[1], X_train.shape[2]),
>>>   stateful=True,
>>>   dropout=0.4,
>>>   recurrent_dropout=0.4,
>>>   implementation=implementation,
>>>   return_sequences=True
>>> ))
>>> model.add(LSTM(64, dropout=0.4, recurrent_dropout=0.4,
implementation=implementation, return_sequences=True))
>>> model.add(LSTM(32, dropout=0.4, recurrent_dropout=0.4,
implementation=implementation))
>>> model.add(Dense(1))
>>> model.add(Activation('sigmoid'))
>>> model.compile(optimizer='adam', loss='binary_crossentropy')
```

The AUROC is not differentiable and could therefore not be used as a loss function. Instead, we trained the network with the binary cross-entropy metric, which was rescaled using the Keras *class_weight* parameter.

The class weights was computed as follow:

```
>>> y_len = float(len(y_train))
>>> class_weight = dict((i, (y_len - (y_train == i).sum()) / y_len) for i in
np.unique(y_train))
```

We wanted to use a batch size of 32 trials and retain statefulness until the end of each training epoch. By default, Keras reset states after each batch. To avoid this, the *stateful* flag was set, and states were reset in a callback after each epoch.

The callback is very simple:

```
>>> class ResetState(Callback):
>>>   def on_epoch_end(self, epoch, logs={}):
>>>         self.model.reset_states()
```

We also had to slightly resize the train, validation and test sets in such a way that their sizes were multiple of the batch size. Here is how we extracted a validation set from the train set:

```
>>> batch_size = 32
>>> validation_split = 0.25
>>> valid_size = int(X_train.shape[0] * validation_split / batch_size) *
batch_size
>>> train_size = int(X_train.shape[0] * (1 - validation_split) / batch_size) *
batch_size
>>> X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,
test_size=valid_size, train_size=train_size, stratify=y_train)
```

We specified a large *epochs* value, and implemented the following early stopping policy:

- After each epoch, compute validation loss and validation AUC.
- If the validation AUC has improved, save the model.
- If the validation loss has not improved for 50 epochs, stop training.
- Use the last saved model as the final model.

The AUC computation is done in the *History* callback:

```
>>> class History(Callback):
>>>   def __init__(self, filepath):
>>>         self.filepath = filepath
>>>   def on_train_begin(self, logs={}):
>>>         self.logs = {'loss': [], 'val_loss': [], 'val_auc': []}
>>>   def on_epoch_end(self, epoch, logs={}):
>>>         predictions = self.model.predict(self.validation_data[0])
>>>         auc = roc_auc_score(self.validation_data[1], predictions)
>>>         logs['val_auc'] = auc
>>>         self.logs['loss'].append(logs.get('loss'))
>>>         self.logs['val_loss'].append(logs.get('val_loss'))
>>>         self.logs['val_auc'].append(auc)
```

```
>>>         with open(self.filepath, 'w') as fp: json.dump(self.logs, fp)
>>>         print('val_auc: %f' % auc)
```

We used the standard Keras *EarlyStopping()* and *ModelCheckpoint()* callbacks to monitor the loss function and regularly save the best model.

Refinement

During our first tests with Keras default parameters, our results were just a little better than chance. The network was converging after only one or two epochs.

Because of limited resources, we did not have the opportunity to systematically evaluate all hyper-parameters with a grid search. Instead, we adjusted several parameters on a subset of the data by manual trial and error, with intuition as our starting point.

Overfitting is obviously a major concern in deep neural networks. The following techniques greatly improved the initial performance:

- Randomly dropping inputs during updates (Nitish, Hinton, Krizhevsky, Sutskever, and Salakhutdinov, 2014)[19].
- Randomly dropping recurrent states (Yarin and Zoubin, 2015)[20].
- Keeping a relatively small batch size to allow frequent weight updates while maintaining statefulness for the full training epoch.

The results of a few significant experiments can be found in *Figure 8*. They were all ran on the same reduced dataset (conditions: [C, A, M], participants [0, 1, 2]), with three LSTM layers (output sizes: 128, 64, 32), and a batch size of 32 trials.

Experiment	Dropout	Recurrent Dropout	States reset after	Grand mean AUC
1	None	None	Batch	**0.560**
2	0.4	None	Batch	**0.572**
3	0.4	0.4	Batch	**0.574**
4	0.4	0.4	Epoch	**0.576**

FIGURE 8: EXPERIMENTS SUMMARY

These results should be taken with a grain of salt. Because of the small subset, and of course, the stochastic nature of neural networks, they are not entirely reliable. They can only be seen as a tendency that must be verified at scale.

The performance of our final model is detailed in the *Justification* section.

Results

Model evaluation and validation

K-fold cross validation is computationally expensive in the context of deep learning. Instead, we used three subsets for training and evaluation:

- Train set: the data on which the model was trained.
- Validation set: the data on which the model was evaluated at the end of each epoch.
- Test set: the data on which the final model was evaluated.

As explained in the *Preprocessing* section, in order to increase robustness to noise, artifacts were not removed.

The main parameters are described in the *Implementation* section. The input and output sizes of each layer can be found in *Figure 7*.

To ensure our results were not biased, an exhaustive cross-validation scheme (as described in the *Algorithms and techniques* section) was implemented, resulting in the computation of 117 models. This took about 53 hours on a Tesla K80 (single GPU).

Figure 9 shows the progression of the loss function for the training and validation sets. The data was averaged over the 117 models. The validation loss is significantly higher than the training loss because Keras rescale only the training loss when the *class_weight* parameter is set. Our early stopping rule selects the model at epoch 46 on average (standard deviation: 17.83), which is consistent with *Figure 7*. We can notice that the loss becomes noisy at the end. It is because the early stopping is rarely triggered after 120 epochs.

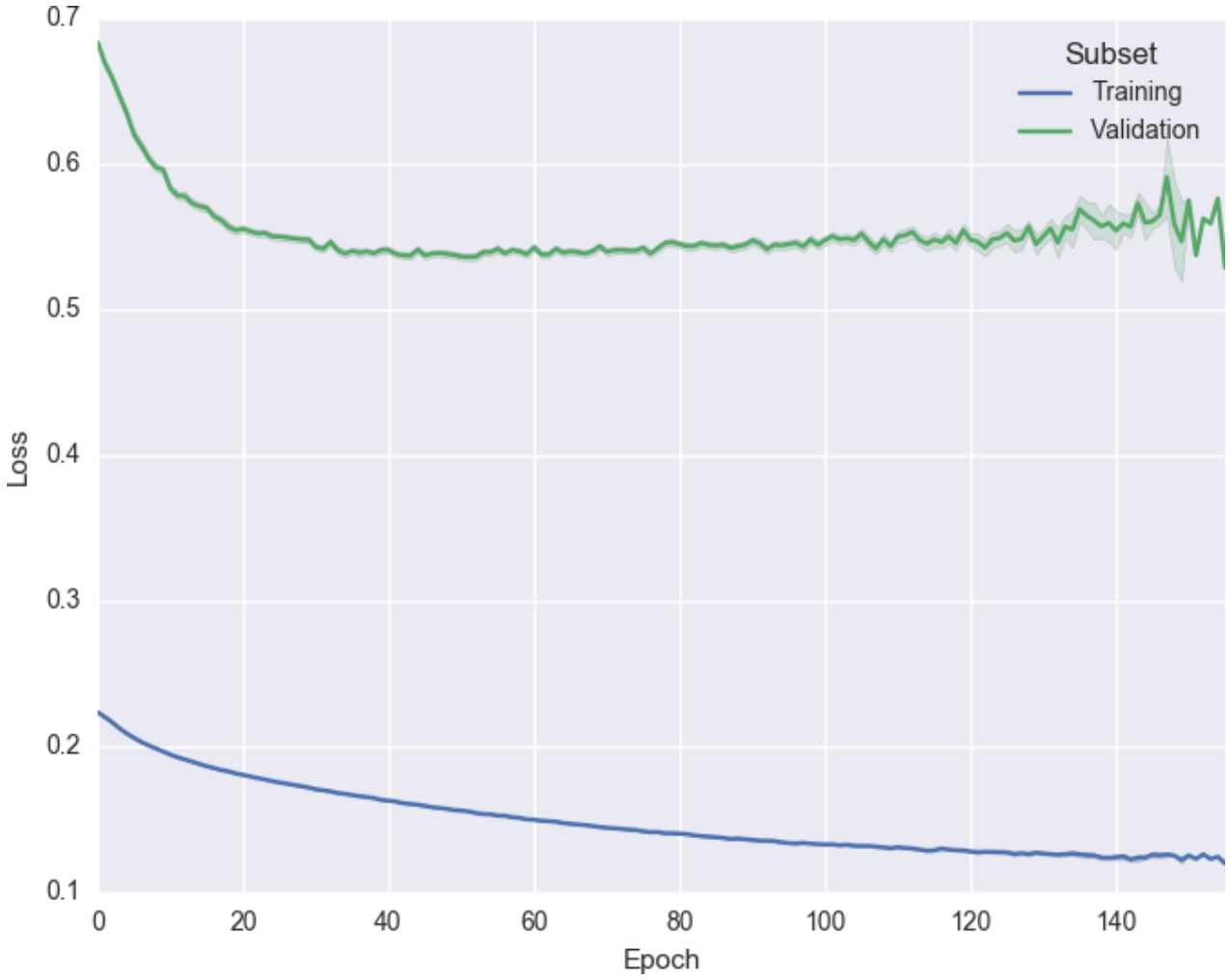
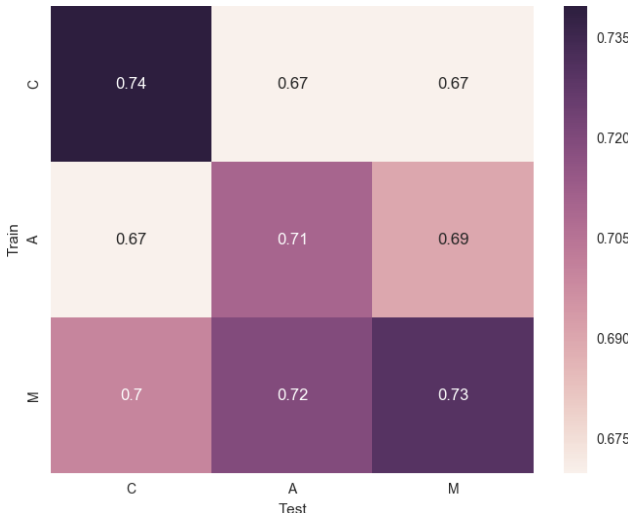


FIGURE 9: TRAINING AND VALIDATION LOSSES

Justification

The final model offers significant improvement over the baseline model, with a mean AUC increase of 0.09 (*Figure 10* and *Figure 11*).



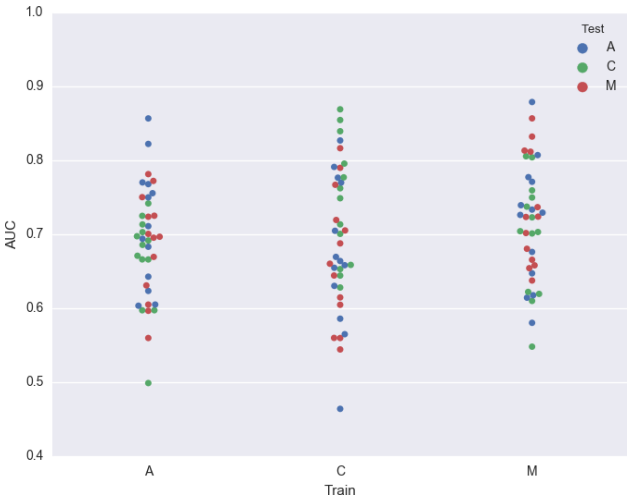


FIGURE 10: AUC FOR EACH CONDITION PAIR (FINAL)

FIGURE 11: AUC FOR EACH PARTICIPANT (FINAL)

The final model has a grand mean AUC score of 0.7, *versus* 0.61 for the benchmark model. According to *Figure 12*[21], it can be considered as a fair model. There is of course room for improvement.

AUC	Grade	Meaning
.90 — 1	A	Excellent
.80 — .90	B	Good
.70 — .80	C	Fair
.60 — .70	D	Poor
.50 — .60	E	Fail

FIGURE 12: MEANING OF AUC SCORE

Conclusion

Spatio-temporal visualization

The scalp topographies for the Target stimulus (*Figure 13*) and the Distractor stimulus (*Figure 14*) can be compared. Data is averaged across all participants and all conditions. We can clearly see differences between the two stimuli.

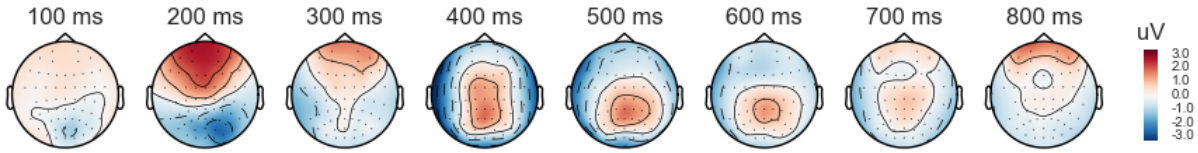


FIGURE 13: SCALP TOPOGRAPHIES FOR THE TARGET STIMULUS

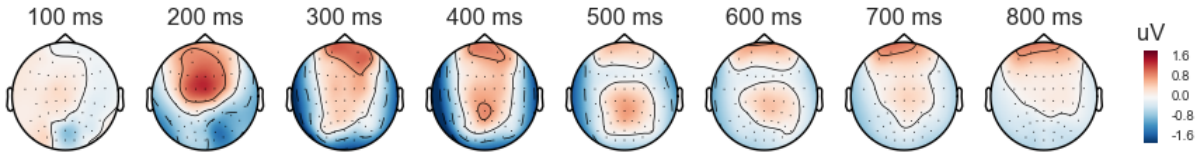


FIGURE 14: SCALP TOPOGRAPHIES FOR THE DISTRACTOR STIMULUS

Reflection

In this study, we have shown that LSTM networks can successfully be applied in the context of single trial classification of ERP signals.

One challenge we encountered was overfitting. This was overcame by maintaining statefulness during each training epoch, and by randomly dropping inputs and recurrent states.

The main obstacle was the computational cost. Because of the relatively long training time, it was difficult to rapidly test new ideas. Small subsets tended to overfit quickly. This was partially avoided by reducing the high dimensionality of features, which was accomplished by downsampling the data.

The final model is promising, but still far from optimal.

Improvement

With more time and resources, we would have done a full hyper-parameter sweep, including the following: optimizer, initial learning rate, learning rate decay, random distribution. We noticed afterwards that we forgot to apply statefulness to the second and third LSTM layers. This should

definitely be corrected in future versions. Other implementations of recurrent networks, such as Gated Recurrent Units[22] (GRU) should also be investigated.

With more data (more trials and more users), we could produce deeper and wider networks. We could also try to reduce or even completely discard the downsampling stage.

A critical step towards general ERP classifiers would be the constitution of a standardized database of signals. It would ideally be compiled from many studies, and include different experimental conditions, attention tasks, stimuli types, electrodes types, and amplifiers configurations. An “ImageNet[23] of ERP signals” would certainly stimulate the research in this field.

To be really useful, a general ERP classifier should not depend on the montage (the number and position of the electrodes). Instead, an estimator should be trained independently for each position (according to the standard 10-20 system). Meta-models or ensembles would then be pre-computed for common configurations or could easily be composed on demand.

References

[1] https://en.wikipedia.org/wiki/Electroencephalography

[2] https://en.wikipedia.org/wiki/Neural_oscillation

[3] https://en.wikipedia.org/wiki/Event-related_potential

[4] Ortiz-Rosario, Alexis, and Adeli, Hojjat. “Brain-Computer Interface Technologies: From Signal to Action.” *Reviews in the Neurosciences* 24, no. 5 (January 1, 2013). doi:10.1515/revneuro-2013-0032.

[5] https://en.wikipedia.org/wiki/Brain-computer_interface

[6] Chapman, Robert M., and Bragdon, Henry R. “Evoked Responses to Numerical and Non-Numerical Visual Stimuli While Problem Solving.” *Nature* 203, no. 4950 (September 1964): 1155–57. doi:10.1038/2031155a0.

[7] https://en.wikipedia.org/wiki/P300_(neuroscience)

[8] https://en.wikipedia.org/wiki/Oddball_paradigm

[9] Wenzel, Markus, Almeida, Inês, and Blankertz, Benjamin. “Is Neural Activity Detected by ERP-Based Brain-Computer Interfaces Task Specific? - Data Set.” Technische Universität Berlin, 2016. doi:10.14279/depositonce-5523.

[10] https://creativecommons.org/publicdomain/zero/1.0/

[11] Wenzel, Markus, Almeida, Inês, and Blankertz, Benjamin. “Is Neural Activity Detected by ERP-Based Brain-Computer Interfaces Task Specific?” Edited by Daniele Marinazzo. *PLOS ONE* 11, no. 10 (October 28, 2016): e0165556. doi:10.1371/journal.pone.0165556.

[12] Fawcett, Tom. “An Introduction to ROC Analysis.” *Pattern Recognition Letters*, ROC Analysis in Pattern Recognition, 27, no. 8 (June 1, 2006): 861–74. doi:10.1016/j.patrec.2005.10.010.

[13] https://en.wikipedia.org/wiki/Electrooculography

[14] https://en.wikipedia.org/wiki/10%E2%80%9320_system_(EEG)

[15] Hochreiter, Sepp, and Schmidhuber, Jürgen. “Long Short-Term Memory.” *Neural Computation* 9, no. 8 (November 1997): 1735–80. doi:10.1162/neco.1997.9.8.1735.

[16] Blankertz, Benjamin, Lemm, Steven, Treder, Matthias, Haufe, Stefan, and Müller, Klaus-Robert. “Single-Trial Analysis and Classification of ERP Components — A Tutorial.” *NeuroImage* 56, no. 2 (May 2011): 814–25. doi:10.1016/j.neuroimage.2010.06.048.

[17] https://martinos.org/mne/

[18] Kingma, Diederik P., and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” *arXiv: 1412.6980 [Cs]*, December 22, 2014. http://arxiv.org/abs/1412.6980.

[19] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex, Sutskever, Ilya and Salakhutdinov, Ruslan. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” *Journal of Machine Learning Research* 15 (2014): 1929–58.

[20] Gal, Yarin, and Ghahramani, Zoubin. “A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.” *arXiv:1512.05287 [Stat]*, December 16, 2015. http://arxiv.org/abs/1512.05287.

[21] http://gim.unmc.edu/dxtests/roc3.htm

[22] Chung, Junyoung, Gulcehre, Caglar, Cho, Kyunghyun, and Bengio, Yoshua. “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches.” *arXiv:1409.1259 [Cs, Stat]*, September 3, 2014. http://arxiv.org/abs/1409.1259.

[23] Deng, Jia, Dong, Wei, Socher, Richard, Li, Li-Jia, Li, Kai, Li, and Li, Fei-Fei. “ImageNet: A Large-Scale Hierarchical Image Database.” In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–55, 2009. doi:10.1109/CVPR.2009.5206848.