DeepEye: Eye Tracking with Deep Learning

Semester Project
Distributed Computing Group
ETH Zurich

by

Damjan Kostovic

Supervisors:	Prof. Dr. Roger Wattenhofer
	Ard Kastrati
	Martyna Plomecka
Student ID:	18-717-371
Address:	Ruopigenhöhe 34
	6015, Lucerne
E-Mail:	damjan.kostovic@hotmail.com
Date:	July 03, 2022

Abstract

In this thesis, we aimed to combine existing results of Kastrati et al., 2021 to create a multi-task learning architecture which learns simultaneously on multiple tasks with the goal of building an EEG based eye tracker. We used this architecture as a pretraining pipeline for two of the original tasks Kastrati et al., 2021 implemented, namely two tasks which consisted of predicting the correct position and amplitude of a subject's gaze using EEG data. While we achieved that the unified model learned multiple tasks to some extent, the results do not bring us closer to the aforementioned goal because the pretraining pipeline does not lead to better results than published in the original paper.

Contents

1 Introduction

This thesis builds on previous work from Kastrati et al., 2021. We will briefly outline some results that were achieved and how the data is structured.
The EEG data was recorded from 365 healthy adults between 18 and 80 years of age. 190 of the 365 participants were females. All participants were rewarded a monetary compensation for their participation. The EEG data itself was recorded at a sampling rate of 500 Hz using a 128-channel EEG Geodesic Hydrocel system. The data was preprocessed minimally and maximally using the toolbox from Pedroni et al., 2019. In this thesis, we only use the minimally preprocessed data as it showed better results in previous experiments of Kastrati et al., 2021. In the same paper, more details regarding the data and the exact results can be found.
In the following, we will explain some of the conducted benchmarks in more detail because they are needed in the main part of this thesis. Kastrati et al., 2021 propose multiple benchmarks which consist of four different tasks with different difficulties. Multiple classical machine learning models which operate on extracted features rather than the raw data and deep learning models were used in the benchmarks. We can summarize the benchmark tasks into left-right task, angle/amplitude task, and absolute position task. The left-right task is the easiest task and consists of determining the horizontal direction of a participant's gaze. For example, the best model, Xception, predicts the horizontal direction correctly with an accuracy of 98.8% with a standard deviation of 0.1%. The second task, the angle/amplitude task consists of predicting the angle and amplitude of a saccade. At times, we refer to these two tasks as direction task since they use the same dataset. Simple deep-learning models, such as CNN or Pyramidal CNN, perform best in these. The last task consists of predicting the absolute position of a participant's gaze on the screen, described by XY-coordinates. This is the most difficult task and is best predicted by the CNN model. In the next paragraph, the process which lead to the main thesis topic is explained.
Initially, we planned to use the existing models developed by Kastrati et al., 2021 and apply them to different time series datasets. Interestingly, a similar paper was published at roughly the same time, and we decided to go a different path. The project started with the question how good we can decode EEG data. The goal was to maximize the information we get from EEG data, i.e.,

to maximize the throughput. As an example, left-right gaze prediction with EEG data is highly accurate but has only limited information per fixed time unit as the transformation flow is very slow. In the following weeks, we worked on some simple methods which can generate more throughput while being almost as accurate as left-right prediction. For text data, one could think of letters in a circular arrangement, for example 5 letters. Similar models can be found in the literature and will be briefly discussed in Section 2. The ultimate goal was to develop path eye-tracking using EEG data. In other words, instead of just predicting, for example, the fixed position of a subject's gaze, we wanted to predict the path a person follows on the screen. By achieving this, an EEG based eye tracker could potentially replace traditional eye trackers in situations where extremely high precision is not necessary. Next, we derived a few statistical results and conducted some small simulations to gain a bit more intuition how useful a 98.8% accuracy is in the left-right task using the Xception model. In the following weeks, we dived deeper into the literature on expressing language using eye trackers. The goal was to combine these methods and use them on our EEG eye tracker. This posed some challenges, as traditional eye trackers are much more precise. Section 2 goes into more detail on this first phase of roughly 6 weeks.. For the rest of the project, we decided to focus more on a deep learning engineering task. We aimed to build a unified architecture which could help achieve better performance in the simple tasks and, therefore, a better chance of predicting the path of a subject's gaze. As a first task, we reproduced some benchmark runs of Kastrati et al., 2021 using classical machine learning methods as well as using more sophisticated deep learning methods. In the following, we aimed to combine the simple task predictions into one pipeline that outputs predictions for all tasks and learns on all tasks simultaneously. The same architecture was then used as a pretraining pipeline for the single tasks. The exact procedure is explained in more detail in Section 3. Section 4 describes the results of a small simulation we conducted with the unified architecture. Section 5 concludes.

2 Eye Based Text Entry Applications

This section discusses selected parts of the literature on gaze-based speech. Most papers use traditional eye trackers. Hence, this chapter serves to show what types of text entry applications using eye trackers exist. Further, we will briefly outline the limitations of using EEG based eye trackers on such methods. We will mention some specific methods, but also remind the reader that many more exist in the literature. In the following, we may refer to text entry applications as speech applications.

2.1 Categories of Text Entry Applications

There exists a plethora of methods to let a person speak with its eyes. The first thing that may come to mind is a screen with all letters displayed on the screen. A person then may choose letter after letter by looking at it with some mechanism to confirm that the chosen letter is correct. While this approach may work with precise eye trackers, it will be slow compared to more sophisticated methods. In this section, we will mention different types of eye trackers for speech.

According to Chen et al., 2018, the literature often divides text entry applications into two major categories: Dwell-time selection and gaze gesture selection. Dwell-time selection consists of fixating, for example, a point on a screen for a certain amount of time. Most importantly, methods of this kind need high accuracy to correctly identify where the subject's gaze is fixed on. Further, Chen et al., 2018 explain that gaze gestures methods consist of applications that require that the subject generates gestures using their eyes that resemble letters or follow some specific path. This approach is followed by Kane et al., 2008 and their keyboard called EyeWrite. Another possibility of gaze gesture methods require the subject to steer its gaze towards the intended letter as with the implementation by Ward et al., 2000, called Dasher. The final gaze gesture application Chen et al., 2018 mention is the so called smooth-pursuit eye-movement as implemented by Lutz et al., 2015 in their keyboard called SMOOVS.

We may point out that there exist many more methods of gaze spellers which, for example, are based on direct gaze pointing or dynamic context switching. As they need even more accurate gaze positions, they are not

suitable for EEG based eye trackers and are not discussed in more detail here. Further, the aforementioned papers do not mention exact requirements the eye trackers need for the keyboard, but they rely on traditional eye trackers. Hence, we can assume an implicit need for some precision for the keyboards to accurately work.

2.2 Gaze Gesture Approaches

In this section, we will briefly discuss two approaches in more detail. They are chosen as both utilize a gaze gesture approach. In these two methods, participants follow some path to write a letter. In later sections, we will link them to our EEG based methods. The SMOOVS keyboard implemented by Lutz et al., 2015 consists of a hexagonal layout with hexagonal tiles. In more detail, they implemented 6 clusters which consist of letters. As soon as a person's gaze is pointed towards a cluster, the cluster is selected. Immediately following, the subject then selects the correct letter in a similar cluster as before. They argue that such a hexagonal layout, i.e., choosing the correct hexagonal tile, is a good trade-off between accuracy and speed. As they worked with traditional eye trackers, they could rely on high accuracy which is not guaranteed using EEG based eye trackers. Nonetheless, we believe that such a selection is feasible with an eye tracker based on EEG data.
EyeWrite, the implementation of Kane et al., 2008, presents the user with a square window with 4 corners. The subject then writes letters by following some path with the eyes. For instance, the letter "t" is written by starting at the top left corner and then going to the top right corner and then to the lower right corner. Again, they use a precise eye tracker for this task. Similarly, we think that this model is suitable for an EEG based eye tracker.

3 Unified Architecture

This chapter describes the unified architecture we developed. It also outlines some problems and difficulties we faced or potentially could face.

3.1 Challenges

There are different ways to build a unified architecture given continuous EEG data and given all the already implemented models in Kastrati et al., 2021. As a first step, we aimed to implement a unified architecture for the left-right, angle, amplitude, and position task. The final unified architecture also included these four tasks. There are a few things we needed to pay attention to:

- Since there already exist pipelines to predict the specific tasks (left-right, angle, amplitude, path), we should use these. For example, we could pretrain data on a simple task, such as the left-right task, and then use these weights as initialization weights for another task. We later discarded this idea in favor of pretraining on multiple tasks simultaneously
- The end goal is to combine multiple single task prediction pipelines into a single architecture which predicts eye position most correctly. We can use this general architecture to pretrain weights, and later use these pretrained weights on a single task. In this thesis, we pursued this idea.
- If we want to train on all tasks simultaneously, we need to create a merged dataset. Depending on the implementation, we may need to change the original dataset and include an additional column which indicates from which dataset each datapoint originally stems. This is just one possible solution.
- Currently, each of the pipelines has some different output. For example, the left-right prediction pipeline outputs either a 0 or a 1 indicating a left or right gaze. For the general architecture, we need multiple outputs. Similarly, the tasks need different loss functions. Hence, we need to implement a general loss function which applies different losses depending on the origin of the datapoint. Further, a more general output is needed.

3.2 Implementation

As mentioned in Section 3.1, we had multiple ideas such as pretraining on some single task and then use these pretrained weights as initialization weights for another task. This chapter outlines the two approaches we started implementing and the way we implemented them. In the end, we opted for the implementation where we trained simultaneously on all the data and all the models.

We decided to implement a simultaneous architecture for the following 4 tasks; the left-right task, the angle task, the amplitude task, and the position task. The angle and amplitude task use the same dataset, to which we refer to as direction dataset. In the following, we may use direction task synonymously for angle and amplitude task. The datasets corresponding to the left-right task and the position task may be referred to as left-right dataset and position dataset, respectively.

To implement the simultaneous architecture, some main changes in the original code were necessary. The first approach was to change the data loader such that the model trains, during each epoch, with batches from all the different datasets. For example, if epoch 1 consists of 64 batches, 20 batches could be from the left-right task dataset, 20 batches from the position task dataset and so on. By doing so, it will eventually train on all datasets but every batch will contain data from only 1 sample corresponding to only 1 task (apart from the direction dataset which corresponds to the angle and the amplitude task). First testing worked well, but together we opted for a different approach. Instead of training each batch on a different dataset, we included all the datasets within each batch to have an even more "simultaneous training" architecture. For this to work, we included an additional column in each dataset which specifies from which dataset each datapoint comes from. Then, all 3 datasets were concatenated and shuffled such that the data occur in a random fashion. As a side note, both methods are viable and there is no indisputably better architecture.

As discussed in Section 3.1, the 4 tasks have outputs of different sizes. In the unified architecture, we needed to change the network such that it can output predictions for all the tasks independent of where the datapoint stems from. Hence, the output layer was transformed in the BaseNetTorch file. This file contains the basic implementation of a neural network and acts as a parent class to most other deep learning models Kastrati et al., 2021 implemented.

Instead of giving just the needed output, the network will output 5 values corresponding to 4 tasks. The left-right task needs a 1 value output while the direction (angle and amplitude) and position tasks need 2 valued outputs. In more detail, the left-right task outputs whether the subject's gaze is pointed toward the left side of the screen or the right side of the screen. The direction task outputs the angle and amplitude of a subject's gaze on the screen, and the position task output outputs the XY-coordinates of a participant's gaze on the monitor. For the loss calculation, only the needed output is utilized and the rest is discarded. Small implementation details are left out at this point as they do not contribute to understanding the architecture but are necessary for the multiple architecture to work. Nonetheless, they are documented in more detail in the corresponding Google doc.

Further, we needed to create a custom loss function. For example, the performance of the position task was measured with the mean squared error while the left-right task used the binary cross entropy criterion. At first, the task seemed trivial but it was not clear where we should ideally implement this differentiation. This was the main reason for adding a new column to each dataset and, hence, to the merged dataset. Then, every data point also contains the information for which task it is needed. This means that the same losses were used as in the original paper.

Finally, we want to point out that we only trained on the two deep learning models, namely CNN and Xception. This is due to them showing the best results as explained by Kastrati et al., 2021.

3.3 Simultaneous Training and Tuning

This section describes how we arrived at the tuning parameters and what influence they have on the result. The unified architecture model was trained without major hyperparameter tuning. We did some testing which is not displayed here as it was of rudimentary nature. We wanted to get a feeling how some parameters, such as the weights of the losses in the simultaneous training, the number of epochs, and the batch size, have on the final result. We may also underline that no extensive parameter tuning was conducted due to time constraints on this thesis.

With this rudimentary hyperparameter testing we conducted, we did not arrive at hugely different results. For instance, we implemented weight tuning as shown by Lin et al., 2021. We took this choice because the loss of the

position task and the amplitude task were much larger in absolute values than for the angle task and the left-right task. Therefore, the model tries to decrease the error of the position and amplitude task more, i.e., the model focuses more on these tasks. The weight tuning did help with decreasing the test loss for all the 4 tasks more uniformly but, consequently, the errors of the position and amplitude task decreased less compared to the unweighted training procedure. We do not desire this effect as the amplitude task and the position task are of most interest and also the hardest to learn. Even though we aimed to achieve good generalization performance of the unified model, we did not want to neglect these two single tasks. The number of epochs and the batch size barely changed the results, which is another reason why we quickly stopped tuning them in more detail.

4 Results

In the following two sections we will discuss the results of the unified architecture. In Section 4.1, we discuss how well the unified architecture can learn on the single tasks simultaneously. Then, in Section 4.2, we discuss the results of using the unified architecture as a weight tuning procedure to obtain initialization weights for the single tasks using the original architecture.

4.1 Initial Results

The first results, which were conducted using the CNN and Xception models, showed some sort of training. For example, the validation loss for the amplitude task, a subtask of the direction task as well as the position task decreased by roughly 50%. Absolutely speaking, these results are worse than the documented results of Kastrati et al., 2021 when training on the single tasks. Nonetheless, this is not a bad result per se because we aimed to use the unified model as a pretraining procedure and not as a predictor for all the tasks simultaneously. In the following, we will discuss the results of using the unified architecture as a pretraining pipeline for the single tasks.

4.2 Unified Model as a Pretrainer

We decided to do a small simulation to gauge how well the unified architecture acts as a pretrainer for the single tasks. The idea is that training the model on all the tasks and then reusing these weights can help with generalization in the single tasks and consequently decrease error on validation/test sets. This small simulation consists of 10 pretrained models on the unified architecture using the full dataset. As previously mentioned, we opted for the CNN and Xception models. The weights of the 35 epoch training procedure were then saved and used to initialize weights on single task models. In more detail, each of the 10 weights was used for 2 tasks, the amplitude task (part of the direction task and direction dataset) and the position task. We took this choice as these two were the most difficult tasks but also the two tasks of most practical interest. In comparison, we ran the same number of tests on the same single task models without weight initialization. Further, the models (pretrained and non-pretrained) on the single tasks were trained with early stopping, i.e.,

if the validation didn't decrease for 10 consecutive epochs, training was finished early.

The results of this mini simulation can be found in Table 1. We can see that the average best validation loss, at the time of convergence, is independent of the fact that we used pretrained weights to initialize the models or not. This indicates that the initialization itself did not change the final result in terms of absolute validation loss. It is no surprise that the results of the models using pretrained weights are at least as good as the results of the models with random initialization since the network can always "forget" the old weights if changing the weights provides an error decrease. This result does not differ between the models CNN and Xception. Overall, the models using the pretrained initialization weights performed better in 3 out of 4 cases in the first few epochs. This indicates that the unified architecture captured some structure which turns into better performance than random initialization in the early stages of training. As training progresses, this effect diminished and pretraining had no positive effect anymore. To add, convergence happened faster twice for the pretrained model and twice for the non-pretrained models. Hence, we cannot make a statement about convergence speed.

Table 1: This table presents the results of the conducted simulation. The first three columns contain the averaged training losses at the corresponding epochs. The fourth column contains the best averaged validation loss.

	Epoch 1	Epoch 2	Epoch 30*	Best Val-Loss	Converged after
M1	112350	105288	31037	28098	39.5 Epochs
M2	123483	121399	31423	26261	47.5 Epochs
M3	122565	119683	31099	27834	50.8 Epochs
M4	119309	109095	26454	26009	38.75 Epochs
M5	139539	124308	36588	33198	39.25 Epochs
M6	81953	49376	35506	32548	33.2 Epochs
M7	149752	133178	35757	31347	48.4 Epochs
M8	113046	100473	34495	31347	30 Epochs

Note:
M1: CNN model on amplitude task with pretrained weights
M2: Xception model on amplitude task with pretrained weights
M3: CNN model on amplitude task with random initialization
M4: Xception model on amplitude task with random initialization
M5: CNN model on position task with pretrained weights
M6: Xception model on position task with pretrained weights
M7: CNN model on position task with random initialization
M8: Xception model on position task with random initialization
* if the model converged earlier, the loss at the last trained epoch was used

4.3 Discussion

We have seen that our unified architecture does not improve performance in such a way that an EEG based eye tracker built on top of our models could replace a traditional eye tracker. While the unified architecture does capture signal of all the tasks, it does not produce a better end results. Further, in previous works of Kastrati et al., 2021, they have not implemented a task in which subjects follow a path. Consequently, we cannot say with certainty if the methods we saw in Section 2 can be used with our EEG based eye tracker. Nonetheless, they do not need extremely high precision. This question is best answered by conducting additional studies. Similar to the discussed speech applications, most other eye tracking based text entry applications use traditional eye trackers.

On the other hand, further studies may extend the unified architecture by new models. For example, Wolf et al., 2022 have developed new pipelines which could be used for extending the unified architecture and that could make EEG based eye tracking more precise.

Finally, we point out that we cannot conclude whether a unified architecture brings us closer replacing traditional eye trackers with EEG based systems for speech.

5 Conclusion

The unified architecture showed some interesting results. As expected, it was possible to train and decrease the error metrics even when training simultaneously on multiple tasks, but training was not as fruitful as we desired. This can possibly be improved upon by including the segmentation pipeline, implemented by Wolf et al., 2022, as well. Using the unified architecture as a pretraining pipeline showed mixed results. As we did not conduct extensive parameter tuning or testing, we propose that further studies implement simultaneous architectures and try to achieve better performance. Further, we suggest that using different structures may also show different results. Different combination of the tasks, i.e., using the segmentation and the position task in a unified architecture, may show different results as well. Finally, we did not include all the deep learning models. Hence, other models, either for pretraining or final prediction on single tasks, could possibly show better results. We encourage further studies to analyze and develop these and other distinct strategies with the goal of a better single task performance. Nonetheless, we believe that the models discussed in Section 2 can be used using the architecture implemented by Kastrati et al., 2021. Further studies may conduct experiments which analyze these models using EEG data. Finally, we remark that it is uncertain whether achieving traditional eye tracking performance using EEG data is feasible using the models Kastrati et al., 2021 implemented. It is possible that a completely different approach is needed for this goal.

Bibliography

Chen, Y., Li, T., Zhang, R., Zhang, Y., & Hedgpeth, T. (2018). Eyetell: Video-assisted touchscreen keystroke inference from eye movements. *2018 IEEE Symposium on Security and Privacy (SP)*, 144–160.

Kane, S. K., Bigham, J. P., & Wobbrock, J. O. (2008). Slide rule: Making mobile touch screens accessible to blind people using multi-touch interaction techniques. *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*, 73–80.

Kastrati, A., Plomecka, M. B., Pascual, D., Wolf, L., Gillioz, V., Wattenhofer, R., & Langer, N. (2021). EEGEyeNet: a Simultaneous Electroencephalography and Eye-tracking Dataset and Benchmark for Eye Movement Prediction. *35th Conference on Neural Information Processing Systems (NeurIPS), Online*.

Lin, B., Ye, F., & Zhang, Y. (2021). A closer look at loss weighting in multi-task learning. *arXiv preprint arXiv:2111.10603*.

Lutz, O. H.-M., Venjakob, A. C., & Ruff, S. (2015). Smoovs: Towards calibration-free text entry by gaze using smooth pursuit movements. *Journal of Eye Movement Research*, *8*(1).

Pedroni, A., Bahreini, A., & Langer, N. (2019). Automagic: Standardized preprocessing of big eeg data. *NeuroImage*, *200*, 460–473.

Ward, D. J., Blackwell, A. F., & MacKay, D. J. (2000). Dasher—a data entry interface using continuous gestures and language models. *Proceedings of the 13th annual ACM symposium on User interface software and technology*, 129–137.

Wolf, L., Kastrati, A., Plomecka, M., Veicht, A., Klebe, D., Li, J.-M., Wattenhofer, R., & Langer, N. (2022). A Deep Learning Approach for the Segmentation of Electroencephalography Data in Eye Tracking Applications. *39th International Conference on Machine Learning (ICML), Baltimore, Maryland, USA*.