Exploring Simple Neural Network Architectures for Eye Movement Classification

Jonas Goltz
University of Applied Science Munich
Munich, Germany
goltz.jonas@googlemail.com

Michael Grossberg
City University of New York
New York, New York
grossberg@cs.ccny.cuny.edu

Ronak Etemadpour
City University of New York
New York, New York
retemadpour@ccny.cuny.edu

ABSTRACT

Analysis of eye-gaze is a critical tool for studying human-computer interaction and visualization. Yet eye tracking systems only report eye-gaze on the scene by producing large volumes of coordinate time series data. To be able to use this data, we must first extract salient events such as eye fixations, saccades, and post-saccadic oscillations (PSO). Manually extracting these events is time-consuming, labor-intensive and subject to variability. In this paper, we present and evaluate simple and fast automatic solutions for eye-gaze analysis based on supervised learning. Similar to some recent studies, we developed different simple neural networks demonstrating that feature learning produces superior results in identifying events from sequences of gaze coordinates. We do not apply any ad-hoc post-processing, thus creating a fully automated end-to-end algorithms that perform as good as current state-of-the-art architectures. Once trained they are fast enough to be run in a near real time setting.

CCS CONCEPTS

• **General and reference** → **Evaluation**; • **Applied computing** → **Bioinformatics**;

KEYWORDS

Eye movement, Event detection, Machine learning, Deep learning

ACM Reference Format:
Jonas Goltz, Michael Grossberg, and Ronak Etemadpour. 2019. Exploring Simple Neural Network Architectures for Eye Movement Classification. In *2019 Symposium on Eye Tracking Research and Applications (ETRA '19), June 25–28, 2019, Denver , CO, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3314111.3319813

1 INTRODUCTION

Eye movement recordings have been studied in a wide variety of disciplines, as previous research has demonstrated a strong link between attention and eye movements based on the "eye-mind hypothesis" [Rayner 1998]. Since, visual attention can be influenced by many factors, to assess the allocation of visual attention, eye movement patterns recorded by eye tracking systems need to be analyzed, since they produce large quantities of spatio-temporal data. To support the statistical analysis and hypothesis building, we need to detect distinct types of eye movement events, such as fixations, saccades, and post-saccadic oscillations (oscillation after rapid movement) from a raw data stream of an eye-tracker. However, manually extracting the events from data streams is a time consuming process, that takes significantly longer than the data collection itself [Hooge et al. 2018]. Recently, some studies used machine learning algorithms for event detection [Startsev et al. 2018; Zemblys 2016; Zemblys et al. 2018] and some others evaluated the classifications of eye-movement event detection algorithms [Andersson et al. 2017; Zemblys et al. 2015], focusing on the prediction of the most important events mentioned above.

In this paper, we present five computationally efficient, well performing *end-to-end* algorithms. We focus on a simplified architecture and feature extraction, equalizing it with the feature learning ability of neural networks. The suggested models can be run on any small machine, potentially in a near real-time setting (Section 3.3). After discussing related work in Section 2, we discuss our data pre-processing step in Section 3.1.2. We then discuss feature extraction in Section 3.1.1) followed by our model architecture in Section 3.2. We also evaluate the sample-level performance and inference time in Section 3.3.

2 RELATED WORK

Different event detection algorithms have been developed [Bahill and Brockenbrough 1981; Salvucci and Goldberg 2000] since 1970. However, most of the traditional approaches have the disadvantage of using manually chosen thresholds, which have to be set by the user. For example, Zemblys et al. [Zemblys 2016] classify data incorrectly when run outside of their intended range. Therefore, in another study [Zemblys et al. 2017], they developed a random forest based classifier that was able to learn from data and outperformed the traditional state-of-the-art algorithms of that time. Although, their algorithm reached nearly human expert performance, they have stated, that the model had the drawback of the necessity of handwritten post-processing heuristics to produce meaningful events. In their suggested model, the context is only given by handcrafted heuristics, relying on thresholds which have to be set by the user. They claimed that they lose accuracy on the prediction due to those heuristics and the need of manually chosen thresholds.

GazeNet [Zemblys et al. 2018] tackles the post- processing and heuristic problems. They only use the raw spatial coordinates of the tracker without applying any feature extraction, using a combination of convolutional and recurrent layers to learn even low-level features. They suggested a data efficient way to train such a network. This approach of training their network is not restricted to their

ETRA '19, June 25–28, 2019, Denver , CO, USA

ACM ISBN 978-1-4503-6709-7/19/06…$15.00
https://doi.org/10.1145/3314111.3319813

model and could be used with our architecture as well. In contrast to them, we use simple feature extraction with less sophisticated models (Section 3.1.1).

Recently, Startsev et al. [Startsev et al. 2018] used a stack of three 1D CNN-layers in combination with a bidirectional LSTM layer and two time distributed fully connected layers, before and after the bidirectional LSTM layer. They used x and y coordinates relative to the initial gaze of the time-slice and extract velocity and acceleration in radians, relative to the horizontal vector from left to right. However, we used the speed and acceleration in Cartesian coordinates. They also introduced a noise label, conceptual similar to what we suggest as *NULL*-class. GazeNet and the model introduced by Startsev et al. are using a window-based approach in order to capture temporal correlations, which is similar to what we discuss in this paper, which is different compared to the vast majority of other literature approaches to the best of our knowledge.

3 METHODS

We have trained five different models on our data: 1) a simple Recurrent Neural Network (RNN); 2) a Long-Short-Term Memory Network (LSTM); 3) an altered bidirectional RNN; 4) an altered bidirectional LSTM; 5) a standard Neural Network with a time distributed fully connected input layer (TDNN). Please note that we do not use the standard approach for bidirectional networks, but a slightly altered one (explained in Section 3.2). Nevertheless, for simplicity sake, we will refer to them as bidirectional throughout the rest of the paper. Table 1 shows the performance of different models we used in this paper. You can have access to our code [1].

We use the open source library Keras to implement our models. It has very little code overhead and is sufficient for small and simple architectures. The models are evaluated using the open source library sklearn, which was sometimes slightly adapted.

3.1 Data set

We use the same monocular eye movement data that has been used in gazeNet [Zemblys et al. 2018] to develop their model and Startsev et al. [Startsev et al. 2018] to validate their model. The data is published by the Lund University [Nyström 2016]. The data set consist of data collected at 500 Hz by the Hi-speed system from SensoMotoric Instruments (SMI). The data was collected from participants viewing images, videos, and moving dots. The dataset also contains the sample frequency, the distance of the subject from the screen, the screen dimension, the screen resolution, the time stamp, the horizontal and vertical pupil parameters, the horizontal and vertical gaze coordinates as a sequence of datapoints. The data was manually labeled into the following 6 events: fixations, saccades, post-saccadic oscillations (PSO), smooth pursuit, blink and an "undefined" label, if the datapoint did not fit any of the previous categories [Nyström 2016]. We focus on the main categories necessary for image tasks, which are fixations, saccades and PSO. We limit the data set to the data extracted from the images and videos, since the moving dots lack the time stamp. The data set contains 159 847 fixation, 19 485 saccade, 11 525 PSO, 157 862 pursuit and 1 912 blink samples. We discard the pursuit and blink samples and focus on the three remaining events. The very uneven distribution of the labels causes several problems, including a high risk of overfitting for PSOs and especially blinks. On the other hand, algorithms will be biased towards the most common label. Both effects have to be taken into account when training the algorithms.

[1] https://github.com/raharth/eyetracking_classification

3.1.1 Feature extraction. We use a very simple feature extraction, hence, we extract only the velocity as the relative change of the position over time $\vec{v} \approx \frac{\vec{x}_t - \vec{x}_{t-1}}{\delta t}$ and the acceleration as the relative change of the velocity over time $\vec{a} \approx \frac{\vec{v}_t - \vec{v}_{t-1}}{\delta t}$, as well as their norm, in Cartesian coordinates. As a consequence of the idea that eye movement events are temporally and spatially invariant, the absolute coordinates and time are irrelevant. Therefore, we use only features as v, a which are temporarily and spatially invariant. This leads to the feature tuple $(v_x, v_y, \|v\|_2, a_x, a_y, \|a\|_2)$, where subscript x, y indicates the directional component in Cartesian coordinates. We use as few features as possible to decrease the probability of overfitting, simplifying our model and reducing the computational complexity.

3.1.2 Pre-processing. The pre-processing was a crucial step in our approach. In the first step we delete all time frames, where the camera lost track of the eye, splitting the data set accordingly into separate files. In the second step we split the data into three sets: training, validation and test set. To ensure that neither test nor validation data gets polluted, we divide the data based on recordings, separating them from each other before any further processing. This may lead to poorer testing performance, since the test data might be drawn from a different distribution, resulting from different subjects performing the task, but ensures that the evaluation is not biased. The training set contains about 154 500 data points ($\sim$ 81%), while the validation and test set each contain about 16 200 ($\sim$ 8%) and 20 000 ($\sim$ 10%) data points. We also exclude the original gaze coordinates and the time. The events are invariant to translation hence their raw values do not have any meaning, since they are dependent on the origin of the coordinate and the position of the camera. Therefore they can only lead to overfitting, but do not contribute to the generalization of the problem. All features are whitened, based exclusively on the data in the training set to avoid pollution of the test and validation data. Our models learn from time-slices of data, containing several consecutive data points. Hence, the number of consecutive data points, is a hyper-parameter which we chose as 41 empirically as part of our validation search.

Slicing the data into time windows $I = [x_{t-20}, x_{t+20}]$, we have to be careful to not include time jumps in any of the slices or pollute our test and validation data. For our recurrent bidirectional models, we split the interval into two sets $I^- = [x_{t-20}, x_t]$ and $I^+ = [x_{t+20}, x_t]$, with 21 data points each. The data point on which the model has to make a prediction is included in both sets. Please note that in I^+, we started with the most future data point x_{t+20}, going backwards through time. As mentioned before, we split the data into train, validation and test data based on recordings. Therefore it is impossible to have the same data point, present in any of the time-slices of the training data, anywhere in the test or validation data. Though the same data point may appear multiple times within one of the sets.

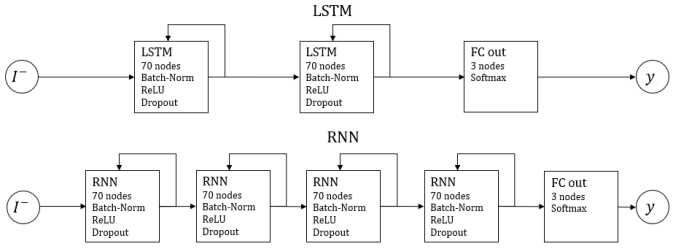

Figure 1: Architecture of the recurrent neural networks. To the left the input I^- processed by a stack of recurrent layers.

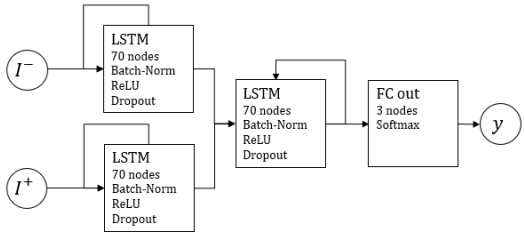

Figure 2: Architecture of the bidirectional recurrent neural networks. To the left the two separate input layers, which are processed to recurrent merging layer followed by a fully connected layer on the right.

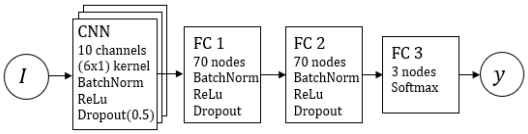

Figure 3: Architecture of the time distributed neural net. With a convolutional input layer, which is followed by 3 consecutive fully connected layers.

3.2 The model

All our models use a fully connected (FC) output layer with three nodes as there are classes to predict. We chose a softmax as the activation function for the output layer because that gives us the probability for a data point to belong to each of the classes. From here on out, we will only talk about the hidden layers of our models assuming that all of them use the mentioned output setting. In the hidden layers, each model is using Batch Normalization [Ioffe and Szegedy 2015] before a rectified linear unit (ReLU) activation function and Dropout [Srivastava et al. 2014] of 50%. This setting is used by every model in this paper.

Our simple recurrent models, the RNN and the LSTM [Hochreiter and Schmidhuber 1997], are both only using I^- as input as shown in Figure 1. For the RNN, we use a stack of four recurrent layers, while our LSTM only uses 2 hidden layers (shown in Figure 1).

Bidirectional neural nets use information from the past and the future on their predictions. As mentioned before, we use a altered slightly reduced bidirectional non-standard setting, in which the model gets past and future data separately. Therefore, the model only processes half of the data in each "time branch", always starting with the most distant data point. Doing so, we reduce the computational complexity and make our model available to small devices, potentially running it in a near real-time setting. The advantage of a bidirectional network over a regular recurrent neural network, which only uses past information, is that a bidirectional network, knows the entire embedding of a data point. Since our model is not required to run in real time, we can afford the time lag generated by this approach. Compared to a simple recurrent network, as our LSTM or RNN, that only learns from the past, it makes the algorithm more robust, hence rarely predicting very short events of the length of sometimes just a single data point.

Besides the type of recurrent layer, our two bidirectional models shown in Figure 2 use the same architecture. Both contain three recurrent layers, with seventy nodes in each layer, hence they are very similar to the latter layers of the gazeNet [Zemblys et al. 2018]. As shown in Figure 2, the inputs I^+ and I^- are separately fed into two recurrent layers, which are then concatenated and processed by another recurrent layer before fed forward to the output layer.

Our TDNN shown in Figure 3 contains a single one-dimensional convolution layer with a kernel reaching over the entire feature space, but only over a single point in time. Even though, we implemented it using a convolutional layer, it is effectively a series of time distributed fully connected layers because the kernel reaches over the entire feature space without performing any strides. The convolution layer is followed by two fully connected layers with 70 nodes, each using Batch Normalization, a ReLU activation function and Dropout of 50%. The input to the TDNN is the entire sequence I of 41 consecutive data points. This architecture is somehow comparable to the latter layers of the architecture developed by Startsev et al. [Startsev et al. 2018] while only using one time-distributed layer, but a stack of fully connected layers.

All of our architectures were optimized performing a hyperparameter search, especially focusing on the number of layers and nodes used.

3.2.1 Training the networks. As mentioned before, we have to carefully balance the distribution of the training data. The data set contains substantially more fixations than any other label, hence models tend to favor them over any other class. Taking into account the distribution, we use bootstrapping techniques to ensure that the models are trained on an equal number of data points from each class. We sample 1000 data points, including their embedding, from each class, which are then used as training data for a single epoch. To reduce the risk of overfitting, each epoch we draw a new bootstrap sample from the training set. We perform the same technique for validation and test sets. For further overfitting prevention, we use early stopping. Each epoch of the performance of the model is evaluated on a fixed, balanced validation set. We also keep track of the performance of the model on that validation set. If the performance on the validation set is not increasing for a fixed number of epochs, called *patience*, learning is terminated and the best performing weights are restored. In our setting, we use a *patience* of 50. No network was trained for more than 500 iterations.

3.3 Evaluation

To evaluate the performance of the classifier, we must use a balanced distribution of labels in the test set, since it would otherwise favor the most frequent labels over all others, given a biased estimation

Table 1: Test accuracy of the different models on each event and as average over all events reported on test data. gazeNet - RA and gazeNet - MN refer to the performance of gazeNet compared to two human specialists, abbreviated by RA and MN. We evaluate our model on the same data, but do not distinguish between the two specialists, neither during training nor evaluation.

model	data set	average	fix	sacc	PSO
TDNN	I	90.89 ± 0.79%	93.22 ± 1.49%	89.74 ± 1.83%	89.76 ± 1.14%
RNN	I^-	87.30 ± 0.99%	88.77 ± 2.32%	86.17 ± 2.78%	87.26 ± 2.02%
LSTM	I^-	88.62 ± 1.20%	89.25 ± 1.61%	86.45 ± 3.17%	90.63 ± 1.28%
bidir. RNN	I^-, I^+	89.96 ± 1.02%	94.06 ± 2.05%	86.83 ± 2.22%	89.40 ± 1.28%
bidir. LSTM	I^-, I^+	90.33 ± 1.06%	94.14 ± 1.55%	86.61 ± 2.87%	90.85 ± 1.33%
gazeNet - RA	-	88.97%	98.36%	92.86%	75.68%
gazeNet - MN	-	88.53%	97.99%	94.97%	72.63%

Table 2: Number of parameters and inference time of each model, run on a i5-4590 CPU with 3.3GHz. The inference time is approximated by 100 runs using 500 data points each.

model	trainable parameters	inference time in s
TDNN	31 123	0.010 ± 0.001
RNN	35 773	0.084 ± 0.003
LSTM	61 533	0.059 ± 0.002
bidir. RNN	26 183	0.054 ± 0.001
bidir. LSTM	102 833	0.084 ± 0.002

towards the most common label. We evaluate the model using the accuracy and the normalized confusion matrix. Since we are interested in the performance of an architecture, but not a particular trained model, we estimate the performance by training 20 random initializations of the architecture on bootstrapped data.

Each model is using a bootstrapped validation set and evaluated on a bootstrapped test set. To get more reliable performance estimate, the model is evaluated on 20 different test data bootstraps. We evaluate each model using the confusion matrix on its test data. We estimate the mean and standard deviation for each architecture. Table 1 shows the results of different models as well as the performance of gazeNet. Unlike Zemblys et al. [Zemblys et al. 2018] who had a comparison between gazeNet's performance and human expert coders (i.e. MN and RA), we do not compare our model to each of them separately, but to a random sample from both of them. Running the same evaluation twice comparing the results, we found that 20 randomized evaluations are sufficient to get a good performance estimation.

We also estimated the inference time of our models by predicting 3000 data points, randomly sampled from the test data, 100 times by each model (Table 2). We excluded the first run, since Keras has some additional overhead when a model is run for the first time. We only report the average and standard deviation for the remaining 99 runs. We ran the model on a standard desktop with a i5-4590 CPU with 3.5GHz and sufficiently memory to store the data. Additionally, we evaluated the model on a GTX970 GPU, but the overhead of the memory swap slowed down the process significantly. The short inference time for our TDNN comes with the fact that we do not use any actual convolution, but basically just fully connected layers.

Moreover, we include a *NULL-class* to the model by defining a probability-threshold for predictions. For each data point, different models predict a probability for each class using a softmax activation function. The predicted probability for all classes below that threshold the data gets labeled as *NULL-class*. Depending on the model, we choose the threshold in the range of 99% to 99.99%. The variance in the threshold is controlled due to the fact that some models tend to have sharper class boundaries than others. This means that the same threshold can lead to a very different uncertainty-prediction rate when used in different models. We set the threshold in a way that about 5% of the data is classified as *NULL-class* by the model. We choose this approach over training the model with *NULL-class* data for two reasons. First, the accuracy of the model dropped significantly to about 66% accuracy on average on the test data when including data from other classes. Second, the only class with a substantial amount of data is the *smooth pursuit*, which makes it hard to construct a *NULL-class*We assume that including a certainty to the model might be helpful to a specialist who interprets the results.

4 CONCLUSION

Our results showed that small neural networks could be sufficient to keep up with state-of-the-art models when provided with the context of an eye tracking data point as we only used simple features such as speed and velocity. Furthermore, we showed that a small time distributed and computational efficient time distributed neural network can achieve high and reliable performance on a sample based evaluation.

Since neither feature extraction nor inference is computationally expensive, we assume especially, our TDNN approach can be run in a near real-time application with just a small temporal delay.

Although, our models could produce reasonable results, it would be interesting to investigate how well our simple models perform when combined with Zemblys et al. suggested data generator in [Zemblys et al. 2018]. As a future direction, we are also interested in including our approach of certainty into the training process in order to optimize the process in its entirety by enabling the model to actively predict the default class as suggested by Startsev et al. [Startsev et al. 2018].

ACKNOWLEDGMENTS

We want to thank Dr. Timothy Ellmore at Psychology Department at CCNY for sharing his eye-tracking data with us.

REFERENCES

Richard Andersson, Linnea Larsson, Kenneth Holmqvist, Martin Stridh, and Marcus Nyström. 2017. One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms. 49, 2 (2017), 616–637.

A Terry Bahill and Allan Brockenbrough. 1981. Variability and development of a normative data base for saccadic eye movements. 21, 1 (1981), 11.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. 9, 8 (1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Ignace T. C. Hooge, Diederick C. Niehorster, Marcus Nyström, Richard Andersson, and Roy S. Hessels. 2018. Is human classification by experienced untrained observers a gold standard in fixation detection? 50, 5 (2018), 1864–1881. https://doi.org/10.3758/s13428-017-0955-x

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. (2015), 11.

Marcus Nyström. 2016. Dataset. (2016). https://www.humlab.lu.se/en/person/MarcusNystrom/

Keith Rayner. 1998. Eye movements in reading and information processing: 20 years of research. 124, 3 (1998), 372.

Dario D. Salvucci and Joseph H. Goldberg. 2000. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the symposium on Eye tracking research & applications - ETRA '00*. ACM Press, 71–78. https://doi.org/10.1145/355017.355028

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from OverĭňĄtting. *Journal of Machine Learning Research* 15 (2014), 30. http://jmlr.org/papers/v15/srivastava14a.html

Mikhail Startsev, Ioannis Agtzidis, and Michael Dorr. 2018. 1D CNN with BLSTM for automated classification of fixations, saccades, and smooth pursuits. (2018). https://doi.org/10.3758/s13428-018-1144-2

Raimondas Zemblys. 2016. Eye-movement event detection meets machine learning. 20, 1 (2016).

R. Zemblys, K. Holmqvist, D. Wang, F.B. Mulvey, J.B. Pelz, and S. Simpson. 2015. Modeling of settings for event detection algorithms based on noise level in eye tracking data. 8 (2015).

Raimondas Zemblys, Diederick C. Niehorster, and Kenneth Holmqvist. 2018. gazeNet: End-to-end eye-movement event detection with deep neural networks. (2018). https://doi.org/10.3758/s13428-018-1133-5

Raimondas Zemblys, Diederick C. Niehorster, Oleg Komogortsev, and Kenneth Holmqvist. 2017. Using machine learning to detect events in eye-tracking data. 50, 1 (2017), 160–181. https://doi.org/10.3758/s13428-017-0860-3