Extended Essay - Mathematics

Monocular on-Screen Gaze Coordinate Prediction with a Multi-Layer Perceptron and Pre-Trained Face Landmark and Gaze Predictors

Word Count: 3850

Table of Contents

I. INTRODUCTION

Artificial intelligence (AI) and machine learning (ML) algorithms have seen a tremendous advancement in the past decade. Programming modules have gained popularity for enabling the creation of artificial neural networks (ANNs) with ease and efficiency. For the past year I have been learning the mathematics behind ANNs and have gained experience constructing them from the ground-up. It is my belief that ML is the technology of the future, as it could help humans in many areas, such as those with disabilities. Before starting this project, I had the intention of developing software that would make people's lives easier. I eventually came up with the idea of enabling people to use computers in the simplest way possible, with eye movement. With software that could track eye movement, many people would gain the ability to control a cursor without the need of using a mouse.

Figure 1: Stephen Hawking uses an Intel device that helps him artificially communicate.

Stephen Hawking had amyotrophic lateral sclerosis (ALS) which is a disease that limits the ability to activate muscles over time. This meant that his ability to communicate would eventually diminish. To enable Hawking to regain his ability to communicate, Intel developed a device that tracked the muscle movement in his cheek. By activating the muscle, he was able to select words from the monitor and an artificial voice would pronounce them. However, many ALS patients do not lose the ability to control their eyes. By creating a software that analyzes eye movement, disabled individuals could be gain the ability to communicate by controlling a device.

Previously, there has been work pertaining gaze tracking that uses a binocular setup[1], but little work has been conducted on gaze tracking via a monocular setup[2]. Most of monocular gaze tracking methods require some sort of setup that involves the utilization of other objects for fixating the camera. In this paper, I present an on-screen gaze tracking technique, using an ANN, that can be used with any device with a monocular camera. My technique involves the usage of pre-trained face landmark and gaze predictors, which provide the 2D coordinates of specific facial landmarks (e.g., the eyelids, points that make up the facial outline), information about pupil coordinates, and not-to-scale values that represent gaze direction from an image of a face (e.g., how far to the left or how far up the person is looking, represented by a scalar value). These predictors are used to gather a dataset and train an ANN to correlate this data with actual on-screen coordinates that indicate where on the device monitor the user is looking at. The technique is not generalizable, meaning that it can only be tailored for a single user. Calibration is done by collecting data of the user looking at pre-defined points on the screen and using it to train the ANN, as per the explanation in *Section IV.* To the best of my knowledge, it is the first time such a technique has been used.

ML is an advanced topic, generally taught as a master's course, and I have tried my best to explain it as understandably as possible. I have explained the mathematics involved;

[1] Hennessey, Craig, and Peter Lawrence. "Noncontact Binocular Eye-Gaze Tracking for Point-of-Gaze Estimation in Three Dimensions." *IEEE Transactions on Biomedical Engineering*, vol. 56, no. 3, Mar. 2009, pp. 790–99. *DOI.org (Crossref)*, https://doi.org/10.1109/TBME.2008.2005943.

[2] Feng, Yu, et al. "Real-Time Gaze Tracking with Event-Driven Eye Segmentation." *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, IEEE, 2022, pp. 399–408. *DOI.org (Crossref)*, https://doi.org/10.1109/VR51125.2022.00059.

however, this topic requires a lot of thinking to grasp, thus it may be reasonable to go over the preliminaries for a while before moving onto the main content. I have categorized this project as mathematics because it heavily relies on calculus, and computers are just used to automate the mathematical calculations involved.

II. PRELIMINARIES

II.A. Matrices

A matrix is an array of numbers arranged in a rectangular pattern. It is made up of rows and columns, where a row is a horizontal list of numbers and a column a vertical list of numbers. For example, in the following matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

the rows are $[a, b, c]$, $[d, e, f]$ and $[g, h, i]$; and the columns are $[a, d, g]$, $[b, e, h]$ and $[c, f, i]$. The size of a matrix is indicated through the notation, $M \in \mathbb{R}^{r \times c}$, where r is the number of rows, c is the number of columns and $\mathbb{R}$ indicates that the numbers are real. The following operations can be done with matrices:

<u>Matrix addition</u>

$$M_1 \in \mathbb{R}^{r \times c}, \qquad M_2 \in \mathbb{R}^{r \times c}$$

$$(M_1 + M_2) \in \mathbb{R}^{r \times c}$$

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \\ g_1 & h_1 & i_1 \end{bmatrix} + \begin{bmatrix} a_2 & b_2 & c_2 \\ d_2 & e_2 & f_2 \\ g_2 & h_2 & i_2 \end{bmatrix} = \begin{bmatrix} a_1 + a_2 & b_1 + b_2 & c_1 + c_2 \\ d_1 + d_2 & e_1 + e_2 & f_1 + f_2 \\ g_1 + g_2 & h_1 + h_2 & i_1 + i_2 \end{bmatrix}$$

Matrix multiplication

$$M_1 \in \mathbb{R}^{r_1 \times c_1}, \qquad M_2 \in \mathbb{R}^{c_1 \times c_2}$$

$$(M_1 M_2) \in \mathbb{R}^{r_1 \times c_2}$$

$$\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} = \begin{bmatrix} a_1 a_2 + b_1 c_2 & a_1 b_2 + b_1 d_2 \\ c_1 a_2 + d_1 c_2 & c_1 b_2 + d_1 d_2 \end{bmatrix}$$

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ d_1 & e_1 & f_1 \end{bmatrix} \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix} = \begin{bmatrix} a_1 a_2 + b_1 b_2 + c_1 c_2 \\ d_1 a_2 + e_1 b_2 + f_1 c_2 \end{bmatrix}$$

Matrix transpose

$$M \in \mathbb{R}^{r \times c}$$

$$M^T \in \mathbb{R}^{c \times r}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^T = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

Matrix scalar operations

$$M \in \mathbb{R}^{r \times c}, \qquad u \in \mathbb{R}$$

$$(M + u), (M \cdot u) \in \mathbb{R}^{r \times c}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + u = u + \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a + u & b + u \\ c + u & d + u \end{bmatrix}$$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot u = u \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a \cdot u & b \cdot u \\ c \cdot u & d \cdot u \end{bmatrix}$$

II.B. Back Propagation

A function can be defined as $f(x) = \omega x + \beta$ where ω is the 'weight' and β is the 'bias'. The weight is initialized randomly in the range of $[-1,1]$ and the bias is initialized as 0. There can be different methods of initialization depending on preference, but in general the weight is initialized randomly. The objective is to have this function output specific values, given specific inputs. This is where back propagation is employed. Back propagation assigns values to the weight and bias such that the function fits a given dataset. A dataset is a collection of pre-defined input and output values with which we are trying to find a function $f(x)$ that best represents this dataset. In other words, we are producing a best-fit line by optimizing a function. This is done via partial derivatives and a cost (loss) function.

Partial derivatives are used to take the derivative of a specific variable in a multivariable function. During this process, all other variables are treated as constants. For example, if I were to take the partial derivative of $f(x)$ with respect to ω, I would treat x and β as constants. This means that the value of x has an impact on the outcome this partial derivative.

A loss function evaluates how far off a prediction is from what it is supposed to be. For example, if $f(0) = 1$, when the output is supposed to be 3, it would result in a bigger loss than if $f(0) = 2$ which is closer to 3. A popular loss function is *L2 Loss*, given by $\mathcal{L}_2(\hat{y}, y) = ||\hat{y} - y||_2^2$ and returns the squared distance of the prediction of the function $\hat{y} = f(x)$ from the ground-truth (what we want the output to be), y. Loss functions must always output a positive value. Furthermore, the derivative of the loss function with respect to $\hat{y}$ essentially indicates the direction (negative, positive or 0) in which the value of $\hat{y}$ should move **to increase the output of the loss function**. By **negating** this derivative, the direction which $\hat{y}$ must move to **reduce the loss** is obtained. The ultimate objective is to have the loss function output a value as small as possible. The smaller the loss, the closer the function is to what we want it to be. Since $\hat{y} = f(x)$, the chain-rule of derivatives can be applied with respect to the weight and bias.

$$\nabla_{\omega}\mathcal{L}(f(x), y) = \frac{\delta\mathcal{L}(f(x), y)}{\delta\omega} = \frac{\delta\mathcal{L}(f(x), y)}{\delta f(x)} \cdot \frac{\delta f(x)}{\delta\omega} \qquad \text{Eq. (1)}$$

$$\nabla_{\beta}\mathcal{L}(f(x), y) = \frac{\delta\mathcal{L}(f(x), y)}{\delta\beta} = \frac{\delta\mathcal{L}(f(x), y)}{\delta f(x)} \cdot \frac{\delta f(x)}{\delta\beta} \qquad \text{Eq. (2)}$$

where $\mathcal{L}(f(x), y)$ is the loss function, ∇_{ω} and ∇_{β} mean "partial derivative with respect to ω and β" respectively, and δ is the equivalent of 'd' used for normal derivatives. If we negate these derivatives, we end up with values that indicate the direction in which ω and β must move to reduce the output of the loss function. These values can then be used to update ω and β.

$$\omega' = \omega - \eta\nabla_{\omega}\mathcal{L}(f(x), y) \qquad \text{Eq. (3)}$$

$$\beta' = \beta - \eta\nabla_{\beta}\mathcal{L}(f(x), y) \qquad \text{Eq. (4)}$$

where η is the 'learning rate', generally initialized as $1 \gg \eta > 0$ and sometimes decreased over time. ω' and β' are the updated values of ω and β. Note that back propagation does not instantly update the values to the most optimal, back propagation is done several times and over time the function becomes more and more optimized. The algorithm which manipulates the learning rate is called a 'scheduler'. The decreasing of the learning rate helps with the preciseness of the optimization as the loss function reaches its 'minima'. A minimum is the tip of a convex curve where the output of the loss function is minimal. There may be *local minima* and a *global minimum*. The global minimum is the lowest point of the loss function, and the local minima are the minima which are not the global minimum. As the output of the loss function gets closer to a minimum, the amount that we change the values of ω and β must decrease (by decreasing the value of η) in order to ensure that we can be as precise as possible. In reality, we do not know where these minima are located, but we know that over time, back propagation will bring the function closer to a minimum, so for every time we use back propagation, we can decrease the value of η.

This example was the simplest form of optimization via back propagation. Depending on the use-case, different ‘optimizers’ with complicated algorithms may be used to update the weight and bias. Back propagation for MLPs is explained in more detail in *Section II.C*.

II.C. Multi-Layer Perceptron

An MLP (multi-layer perceptron) is a function that can take multiple inputs and produce multiple outputs. It is built using two types of functions: “layers” and “activation functions”. A sequence of these functions is chained together to produce one complex function called an MLP. While layers are linear functions, activation functions are generally non-linear. Given as an example in the previous section, layers are similar to $f(x) = \omega x + \beta$, but instead of x being a scalar value, it is in vector form, $x \in \mathbb{R}^a$. A layer must be defined in such a way that the output does not have to be the same sized vector as x, and this is where matrices come into play. In a layer, ω is a matrix $\omega \in \mathbb{R}^{a\times b}$, where a is the size of the input vector, and b is the size of the output vector. Consequently, the size of β must match the size of the output, hence $\beta \in \mathbb{R}^b$. This is formalized into:

$$l(X) = \mathcal{W}^T X + \mathcal{B} \qquad \text{Eq. (5)}$$

where $\mathcal{W}^T$ is the transpose of $\mathcal{W} \in \mathbb{R}^{a\times b}$; $\mathcal{B} \in \mathbb{R}^b$ and $X \in \mathbb{R}^a$; where a and b are the sizes of the input and output vectors respectively. A layer is usually denoted as $(a \times b)$, indicating the vector size it takes as input and the output size it returns.

Although layers must always be in the format of Eq. (5), activation functions can be of any nature. For example, the *sigmoid* activation function is as follows:

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \text{Eq. (6)}$$

This activation function has a range of $(0,1)$. Such a function guarantees that the output always lies between 0 and 1. This is useful in many aspects, such as if you wanted an MLP to output a value that indicated whether something is TRUE or FALSE; you could put a sigmoid activation function at the end of the MLP and round the output to the nearest 0 or 1, where 0 means FALSE and 1 means TRUE. Now, set the output vector size to 64, and the MLP will return a 64-bit binary value.

I have mentioned before that an MLP is a chain of layers and activation functions. What I mean by that is this:

$$\mathcal{M}(X) = \mathcal{A}_N\left(l_N\left(\mathcal{A}_{N-1}\left(l_{N-1}\left(\dots\mathcal{A}_1\big(l_1(X)\big)\right)\right)\right)\right) \qquad \text{Eq. (7)}$$

$\mathcal{M}(X)$ is an MLP where l_n is the n^{th} layer, $\mathcal{A}_n$ is the activation function of l_n, and N is the total number of layers. Note that activation functions are not a requirement in the construction of MLPs and can be placed in any sequence.

Back propagation works the same way with MLPs as described in *Section II.B*. Eq. (8) gives an explicit example of how the chain-rule is used in MLPs.

$$\nabla_{\mathcal{W}_n}\mathcal{L}(\mathcal{M}(X_1),y) = \frac{\delta\mathcal{L}(\mathcal{M}(X_1),y)}{\delta\mathcal{A}_N\big(l_N(X_N)\big)}\cdot\frac{\delta\mathcal{A}_N\big(l_N(X_N)\big)}{\delta l_N(X_N)}\cdot\frac{\delta l_N(X_N)}{\delta X_N}\cdot\frac{\delta X_N}{\delta l_{N-1}(X_{N-1})}\cdots\frac{\delta X_{n+1}}{\delta l_n(X_n)}\cdot\frac{\delta l_n(X_n)}{\delta\mathcal{W}_n} \qquad \text{Eq. (8)}$$

$\mathcal{W}_n$ is the weight matrix of the n^{th} layer; $X_n = \mathcal{A}_{n-1}\big(l_{n-1}(X_{n-1})\big)$ and X_1 is the MLP input vector. The derivative of the loss function with respect to the bias vector of the n^{th} layer is calculated the same way, the only difference being $\mathcal{W}_n$ replaced with $\mathcal{B}_n$ in the notation.

II.D. Mean Squared Error

MSE loss (Mean Squared Error) is the mean of the square of the difference between the prediction and ground-truth of each index of the vectors:

$$MSE(\hat{Y}, Y) = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \qquad \text{Eq. (9)}$$

where n is the size of the MLP output vector. The partial derivative of MSE with respect to the prediction is:

$$\nabla_{\hat{Y}} MSE(\hat{Y}, Y) = \frac{-2}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i) \qquad \text{Eq. (10)}$$

An aspect that makes MSE useful is that its output gets exponentially smaller as the prediction gets closer to the ground-truth.

II.E. Rectified Linear Unit

ReLU (Rectified Linear Unit) is an activation function that has a range of $[0, \infty)$:

$$ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad \text{Eq. (11)}$$

The derivative of ReLU yields:

$$ReLU'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \qquad \text{Eq. (12)}$$

ReLU serves as something like a criterion, “the value is required to be at a certain standard in order to pass”.

II.F. Exponential Learning Rate Scheduler

The exponential LR scheduler exponentially decays the learning rate by γ after every epoch (an epoch is each time we optimize the MLP). Formally, it is defined as:

$$\eta_t = \eta_0 \gamma^t \qquad \text{Eq. (13)}$$

where η_t is the learning rate after t number of epochs, $t \in \mathbb{Z} \geq 0$ is the number of epochs that has elapsed, $0 < \gamma < 1$ is the decay rate, and η_0 is the initial learning rate. The exponential LR scheduler allows finer adjustments in weights and biases as training proceeds and the MLP gets closer to convergence (fully optimized). The smaller the γ value, the faster the learning rate will shrink over time.

II.G. Facial Landmark Prediction

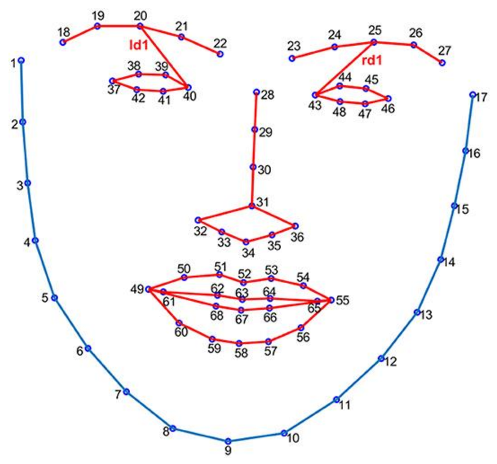

Figure 2: The 68 landmarks of the "68 face landmarks" format[3].

Facial landmarks are pre-defined points on a face that correspond to specific features. The process of predicting the position of these landmarks in an image, through an algorithm or function, is called "facial landmark prediction". It is commonly used in computer vision for purposes such as emotion detection[4]. A set of landmarks in an image of a face is denoted as $\mathcal{F} \in \mathbb{R}^{N\times 2}$ where N is the number of landmarks. Each element of $\mathcal{F}$ corresponds to a single landmark with two coordinates indicating its position in a 2D image. A widely used format of facial landmarks is the "68 face landmarks", as its name suggests it consists of labels for 68 different landmarks on the face (see *Figure 2*).

[3] Shen, Xunbing, et al. "Catching a Liar Through Facial Expression of Fear." *Frontiers in Psychology*, vol. 12, 2021. *Frontiers*, https://www.frontiersin.org/articles/10.3389/fpsyg.2021.675097.

[4] Khabarlak, Kostiantyn, and Larysa Koriashkina. "Fast Facial Landmark Detection and Applications: A Survey." *Journal of Computer Science and Technology*, vol. 22, no. 1, Apr. 2022, p. e02. *arXiv.org*, https://doi.org/10.24215/16666038.22.e02.

III. NETWORK ARCHITECTURE

The network consists of two modules: the "Prior Module" and "MLP Module". The prior module is a pre-trained face landmark and gaze predictor that returns 68 face landmarks and gaze information from an image of a face[5]. Gaze information includes the following: left pupil and right pupil coordinates (each $\mathbb{R}^2$), horizontal ratio of gaze $(-1, 1)$, vertical ratio of gaze $(-1, 1)$, eyes being closed or open (0 or 1); whether the user is looking to the left, right, or center (each 0 or 1). Of the information provided by the prior module, only a selected portion is used as input to the MLP. This includes the following: 37th to 48th indices inclusive of the 68 face landmarks (the landmarks associated with eyes), both pupil coordinates, horizontal and vertical ratio of gaze. That adds up to an input vector of size 30 (coordinates are stacked individually rather than as pairs).

The MLP architecture is as follows:

1. Layer (30x128)
2. ReLU Activation
3. Layer (128x128)
4. ReLU Activation
5. Layer (128x2)
6. No Activation

The MLP has a total of 3 layers. There is ReLU activation after every layer, except the last. The two-dimensional output vector is clipped in the range of $[0,1]$ for each value and scaled by monitor resolution to give 2D pixel coordinates that correspond to a location on the user's screen (this calculation is considered during back propagation).

[5] Gaze Tracking, antoinelame. GitHub Repository: https://github.com/antoinelame/GazeTracking

IV. DATA & TRAINING

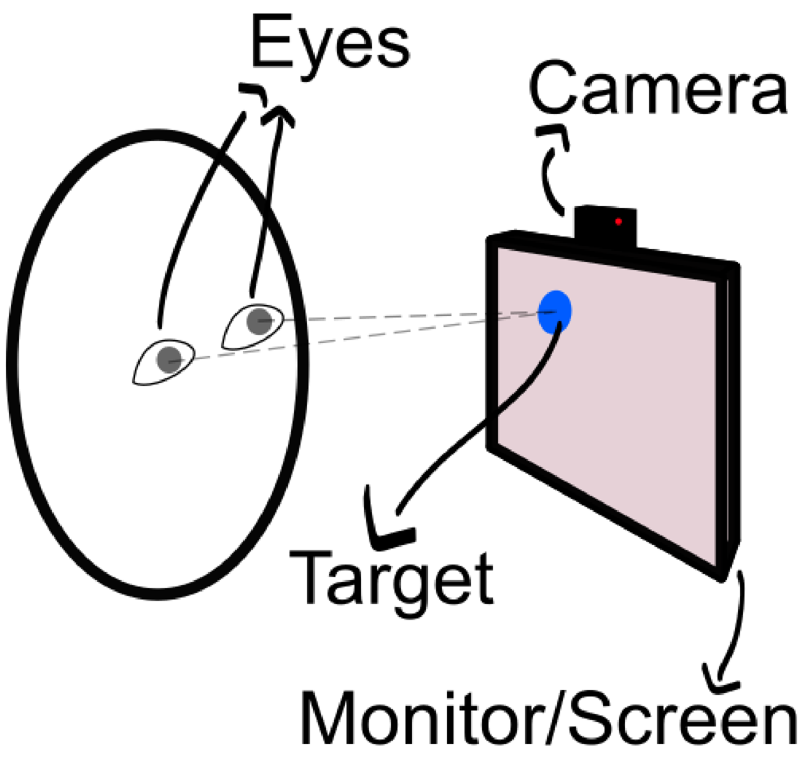

Figure 3: Data collection setup.

Data required for training is obtained via a computer application. When the user starts the software, a target on a blank background appears on the screen (see *Figure 3*). As the user looks at the center of the target while moving their head to different positions, the software takes multiple images with the camera. Looking at the same target from different positions introduces flexibility to the data that will allow the network to track the user's gaze from various positions. Once an image is taken, it is passed onto the prior module to obtain face landmark and gaze data. If the user's eyes are closed or the tracker is unable to extract data, a new image is taken. Data from the prior module is combined with the coordinates of the target to create a data pair $\mathcal{P}_i = \{X \in [0,1]^{30}, Y \in [0,1]^2\}$ where X is the data output from the prior module and Y is the target coordinates. Once 100 data pairs have been obtained, the target moves to a new point on the screen to repeat the process. The screen is divided into 9 points

(3×3) that helps to enable tracking from anywhere on the screen. After data from all points has been obtained, it is combined into a dataset with 900 examples $\mathcal{D} = \{\mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_{900}\}$.

The MLP is trained using the dataset and MSE loss. For every epoch, a random portion of the data is selected (called a 'batch'). For every data pair ($\mathcal{P}_i$) in the batch, X is input into the MLP. It then returns a prediction, $\hat{Y}$. We want this prediction to be as close to the ground-truth (Y) as possible. Therefore, we use the MSE loss function, $MSE(\hat{Y}, Y)$, and back propagate to optimize the MLP. After the MLP has been optimized for every data pair in the batch, the learning rate, η, is decreased using the exponential learning rate scheduler. See *Section V* for details on the exact parameters used.

When back propagating, instead of using Eq. (3, 4), I use a specialized optimizer called '*Adam*'[6]. It is an algorithm that optimizes the MLP in a way that is more efficient. It still uses back propagation, but the equation for updating the weight and bias is a little more complicated than what I presented.

V. EXPERIMENTAL RESULTS AND EVALUATION

Table 1: Training Parameters

Parameter	Value
η_0	0.008
γ	0.9999
β_1	0.9
β_2	0.999
ϵ	10^{-8}
T	25000
B	128

[6] Kingma, Diederik P., and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv, 29 Jan. 2017. *arXiv.org*, http://arxiv.org/abs/1412.6980.

Parameter Definitions for Table 1	
η_0	initial learning rate
γ	exponential learning rate decay rate
β_1	(Adam) exponential decay rate 1 for moment estimate
β_2	(Adam) exponential decay rate 2 for moment estimate
ϵ	(Adam) term added to the denominator to improve numerical stability
T	number of epochs
B	batch size

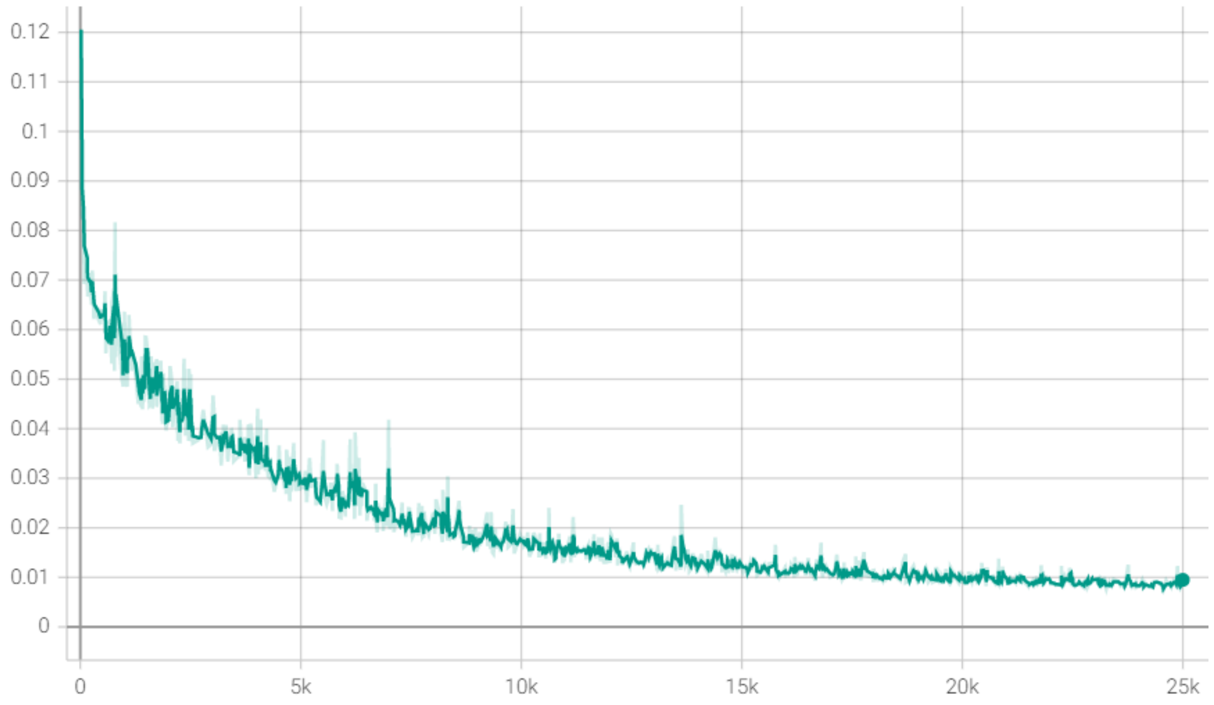

Figure 4: Trial 1 results of training the MLP with parameters from *Table 1*. Loss (vertical axis) versus epoch (horizontal axis).

The loss encountered during trial 1 training is graphed in *Figure 4.* At first, the MLP learns quickly, and over time slows down. The graph is not exactly smooth, as there are some perturbations, which are most likely the result of the existence of multiple minima. During training, the MLP sometimes overshoots a minimum and enters a different one. By transitioning from one minimum to the other, the loss increases in between. These increases correspond to some of the sudden increases seen in the graph. The results indicate that the MLP converges after about 25000 epochs, ~1 hour on an Nvidia GTX 1050 graphics card. The final average loss of the MLP is $\sim 8.7 \cdot 10^{-3}$. The parameters I used for trial 1 are given in *Table 1*.

After the MLP was trained, I tested its performance. The evaluation of the test is mostly qualitative. The MLP was able to predict the general location of where the user was looking at on the screen. However, it could not accurately pinpoint the exact location where the user was looking. This could be due to numerous reasons, including:

- Insufficient data
- Lack of camera quality
- Not enough training
- Training parameters that are not perfect

The following may be done to improve accuracy:

- Collect more data by taking more photographs and/or increasing the number of target positions (e.g., from (3x3) to (5x5)).
- Utilize a higher definition camera (although this may not be viable for individuals who do not have access to one)
- Increase the number of epochs to train the MLP and see how that affects accuracy.
- Adjust training parameters in various ways to see if there are more suitable ones.

VI. CONCLUSION

VI.A Drawbacks and Applications

The technique I have developed for on-screen gaze tracking has potential to be used by many individuals who are in need. This technique, if developed upon, could remove the necessity for excess gadgets for on-screen eye tracking, as the only device that is required is a single webcam and a computer. Such a program can be provided to people online as standalone software, making it extremely accessible. The applications include but are not limited to: “mouse-free” control of a computer to help disabled individuals to use a computer, and the control of games and software. As of now, the technique does not provide pinpoint accuracy, only a general position of the user’s gaze. However, accuracy may be increased by improving the method of data collection and perhaps the network architecture itself. Nonetheless, the technique offers usability and a new opportunity for those who need it.

VI.B Thoughts

Throughout the development of this project, I have encountered many obstacles. I had to learn to create a user interface from scratch and find ways to make data collection and neural network training computationally efficient. I spent hours and weeks training different NN architectures, comparing and contrasting to see which one was more efficient. This was an exhaustive process because training can only progress as fast as the computer can handle. Knowing this led to me attempting to find an architecture and training parameters that sufficed both accuracy and efficiency to show that my technique is viable. If I had the proper resources, I may have perhaps been able to train a more performant network.

In the future I would recommend the trial of different neural network types and the utilization of a camera with good resolution. Additionally, the data collection method may be improved by animating the target instead of using stationary targets. Currently the biggest limitation my technique faces is lack of preciseness. This is likely able to be improved by making the aforementioned improvements, prolonging training time and collecting more data. However, the training time could be a limitation in and of itself, as the user must wait for training to finish in order to use the program. Furthermore, this technique does not generalize to all users, it can only be tailored for a single individual. Therefore, another improvement may be the development of a technique that involves training the network once, such that it does not need to be re-trained in order to be used by a new individual. If such a generalizable technique is developed, it would be significantly more accessible due to the user not needing to re-train the network, hence a faster set-up time.