Design of a Smart American Sign Language Translator

# Introduction

Physical disabilities can prevent people from immersing themselves in everything society has to offer. Deafness, in particular, has a significant impact on a person’s life in that it prevents one from communicating through speech. Studies show that deaf persons are less likely to make use of medical services and are more prone to psychological disorders – disorders stemming from the difficulty of forming meaningful connections with hearing counterparts [1]. There have also been recorded instances of discrimination of the disabled in employment due to their physical disabilities, including deafness [2], [3]. In the case of deafness, communication with others is the overarching issue, bringing with it a surge of inevitable side-effects. This paper seeks to develop a translation tool that relies on the training of a neural network to recognize different ASL (or American Sign Language) gestures.

The following project, tentatively called the “Sign AI,” seeks to design an ASL (American Sign Language) translation tool having deaf persons as the target demographic. The translation tool will take the form of a Windows-based GUI program with which users can interact for various purposes. A deep learning model will be trained to recognize ASL through pre-processed EMG data and orientation data in the form of quaternions calculated, all acquired from the Myo armband developed by Thalmic Labs. The final product is a gesture-based translator, providing translation for each gesture performed for a total of 100 gestures. Also, the final product will have gesture-to-speech (text-to-speech) and speech-to-text capabilities.

Although design constraints are few due to the laissez-faire or “hands-off” nature of an AI project, undoubtedly, they do exist. One constraint lies in the ASL language itself, while others in the software used to develop the GUI. For one, some gestures are not arm-based or hand-based, which means that they cannot be reliably measured by the EMG sensors on the Myo armband. This is due to the linguistic nuances of ASL. For example, in ASL, pronouns are expressed through pointing a finger. Even with a sensing device, there is no way to determine if a gesture meant for a pronoun is “I,” “you,” “he,” “she,” or “they.” It is up to the signers to determine this for themselves. Since the neural network will be hosted locally, it must run on a device that possesses sufficient storage or RAM as well as processing power. In addition, there are no pre-made datasets involving the Myo armband that can be used to train the neural network. Therefore, the dataset must be produced by me. Finally, as a consequence of software dependencies, a wireless internet connection will be necessary to access Google’s text-to-speech and speech-to-text APIs.

# Project Approach

## Data Visualization & Acquisition

Naturally, the start of any deep learning project involves determining a means of acquiring reliable data. While the meaning of “reliable data” is different from project to project, its function remains the same for every project: to capture qualities of *something*. The job of data is to capture desired qualities while the job of the neural network is to make sense of them.

In the case of Sign AI, the training data originates from the Myo armband, which, is a glorified sensing device. The Myo armband acquires EMG readings from 8 different sensor modules and transmits it to the host computer via Bluetooth. The sensor modules are also equipped with an IMU (inertial measurement unit), which the Myo armband accounts for when calculating quaternions. The Myo armband outputs EMG readings for gauging muscle activation and quaternions for representing orientation in 3D space. EMG readings recorded by the Myo armband are pre-processed in that they are normalized to a range of values between -128 and 127. Since EMG can vary from person to person, the normalization of the raw EMG signals will help to center all readings to a common scale.

There are 2 methods through which a user can view data (EMG, orientation) from the Myo. The recommended method is through the Myo Python library created by Nikolas Rosenstein. The Myo Python library acquires information from the Myo armband through the hub instance. The hub instance establishes Bluetooth communication with the Myo armband until the program calling it created is terminated.

The key to formatting the sensor data for use in a neural network lies in stacking readings element-wise and storing them in a master file where the neural network can process them all at once. The NumPy library was used to perform the stacking. At every interval of time, the Myo armband acquires many different readings. To represent them properly, all readings captured at a specific point in time must be altogether contained. Since 50 data points from each sensor module were collected from the Myo armband, the data points must correspond chronologically. Also, a label was appended to the stacked arrays once all readings are processed. This label represents the gesture performed when the now-stacked readings were taken. All stacked arrays will be stored in a pickle file for ease of access. Storage in a pickle file is beneficial in that it maintains a content’s integrity and simplifies access.

## The LSTM Model

The specialty of the LSTM model is memorizing long-term dependencies or patterns in data. There are mechanisms within an LSTM layer that allows it to keep track of dependencies. In an LSTM layer, two elements are responsible for memorization: the cell state and the gates. For reference, the terms “cell state” and “hidden state” can be used interchangeably. The cell state is an additional output produced by an LSTM layer. It is where changes in the input are documented. The cell state and gates go hand-in-hand in that the gates are responsible for updating the cell state. In short, the “gates” are activation functions that have a binary range, meaning their outputs lie between 0 and 1. The activation functions determine if the change to the cell state is significant.

What makes the LSTM model suitable for gesture classification? The answer lies in its ability to memorize long-term patterns and dependencies. Recall that the LSTM model contains layers that update themselves based on the changes that the input data undergoes in the network. These “updates” are representative of the patterns and dependencies that the network has extracted after processing the input data. The input data will come in the form of readings from EMG sensors and a 9-axis IMU. These readings vary with time and correspond to a gesture. The job of the neural network is to determine the performed ASL gesture based on the aforementioned readings. To be successful, it will need to keep track of changes in the sensor readings. In other words, it will need to memorize patterns and dependencies. This need is fully met by the LSTM model, due in large part to its design and intended function.

Hermans and Schrauwen delve into the design of an RNN, particularly the logic behind it: “For RNNs, the primary function of the layers is to introduce memory, not hierarchical processing” [4]. Hierarchical processing in neural networks is the idea that new knowledge is extracted at each layer. Hermans and Schrauwen believe that hierarchical processing is not the aim of an RNN. They also believe that shallow hierarchical processing can limit an RNN’s performance, even listing several performance-affecting limitations. Hermans and Schrauwen make the following assertions: that the RNN does not excel in hierarchical processing, which can affect its understanding of the input, and that the RNN is not by default capable of analyzing data piece-wise with respect to time [4].

In turn, Hermans and Schrauwen proposed a fusion of the DNN and RNN. Dense layers will be used to increase hierarchical processing, while RNNs will be stacked to give temporal context to the input data at different instances of time. Due to the benefits of increased hierarchical processing and temporal context, the paper has adopted the Hermans and Schrauwen architecture.

## Regularization Methods

### Dropout

Deep neural networks are beneficial in that they can learn complicated relationships. This is corroborated by Hinton et al., who state that having multiple layers help the network recognize underlying connections between inputs and outputs [5]. In addition to its architecture, a neural network also relies on training data, which is dissected for hidden information. Ideal circumstances mean that training data is abundant. However, this is not always the case. A limited dataset can cause a neural network to overfit without proper regularization, which is a means of increasing a network’s ability to “generalize.” If a network is capable of “generalizing,” then it can handle data that it has not been trained with. A network not capable of generalizing is considered to be “overfit.” Training a neural network involves micro-managing weights and biases, both of which are parameters that dictate the network’s response to inputs. At every epoch or training cycle, weights and biases are constantly overwritten in an attempt to find the most ideal values. There are many reasons behind an overfit model. One obvious culprit is the number of epochs. If the neural network is exposed to the training data for a long time, it will start to develop weights that respond well to the training data but fail with other data.

An often-used form of regularization, the dropout method is a method that randomly disables nodes in a layer (or layers) of a neural network. In the Keras API, one can specify the percentage of nodes to be disabled by passing it as an argument to the dropout layer. As discussed in Hinton et al.’s paper, there is a biological basis behind the dropout method, one rooted in the topic of genetics and the passing of genes.

Hinton et al. explain the role of dropout in a neural network through the concept of natural selection. The keywords are “reduction” and “co-adaptations.” In biology, evolution forces individuals to develop new genes. These new genes are supposed to help humans develop advantageous characteristics. The end goal is to develop new *and* advantageous genes. This is done by minimizing interplay between genes, known as biological co-adaptations. Co-adaptation is a phenomenon in which a person’s characteristics are determined by a gene pair. Each gene relies on its counterpart to remain functional. These dependencies between genes are not desirable as they create multiple points of failure. Co-adaptation also exists in neural networks in the form of dependencies between layers. Co-adaptations between layers in a neural network can be minimized through dropout. Each layer feeds information into future layers, which can create unwanted or unintended dependencies between layers and cause the neural network to overfit [5]. One way to reduce these dependencies is by preventing small chunks of information from reaching future layers, which the dropout method accomplishes through shutting down nodes. The logic behind the dropout layer is to force layers to be more self-reliant and infer the missing information themselves [5].

### Batch Normalization

Ilyas et al. state that the reason why batch normalization is so beneficial is because of its reparameterization of “the underlying optimization problem” being solved by the neural network during training [6]. One might ask: what is the optimization problem that the neural network is solving? The answer lies in the gradient of the cost function. Recall that the goal of training a neural network is to reduce errors in predictions, which means developing weights that navigate the cost function towards its global minima. How does the neural network determine that the error has been reduced? It does so by looking at the gradient, which is an expression for the rate of change of something. After all, a gradient is simply a partial derivative. The gradient tells the neural network which direction it is traveling the cost function. Since Ilyas et al. have stated that batch normalization makes the gradient “more predictive and well-behaved,” the network will display more stable performance during training as it knows with more certainty where it is going [6]. “Well-behaved” in this case reflects the gradient’s Lipschitzness, as mentioned by Ilyas et al [6]. The mathematical concept of Lipschitzness deals with gauging a function’s continuity, the criteria used by Ilyas et al. to measure “smoothness.” More continuous functions tend to have fewer regions where the change from one value to another value is too significant and abrupt. A continuous function looks more connected, requires less interpolation between points, and is therefore more “smooth.” In short, a smoother gradient lead to a more continuous cost function, which in turn promotes gradient predictability.

## GUI Development

When developing a graphical user interface (or GUI, for short), both form and function go hand-in-hand. In terms of form, a GUI should be easy-to-navigate; in terms of function, it should require as little heavy lifting or accommodation from the user as possible. Firstly, the design of a GUI should not be littered with features. There is such a thing as sensory overload, after all. Having more functions usually translates to having more UI elements. This can end up cluttering the user’s screen and make going through the GUI troublesome.

It was decided that the GUI for Sign AI will be simple and fulfill 3 basic functions: acquiring sensor data from the Myo armband, sending sensor data to the neural network for processing, and acquiring speech data to be converted to text (and vice-versa) by Google APIs. Since the GUI will be written using the Python language, the Tkinter library will be used to develop it. The Tkinter library compiles and adapts the Tk GIU toolkit written in its language to Python. One main benefit of using Tkinter for GUI development is that it is a cross-platform library, meaning that GUIs spawned from it can be run on Microsoft, Linux, and Apple operating systems. It is also equipped with all the traditional UI elements found in most GUIs such as buttons, drop-downs, and scrolled lists.

In Python, making GUIs that call long-running processes through button clicks is not as straightforward as it seems. This is because Tkinter GUIs are not capable of performing asynchronous tasks without outside help. The Tkinter GUI updates itself as well as executes all other user-defined functions or methods in a single thread. The GUI updating itself is last in the pecking order. To make matters worse, if the GUI does not receive the request to update itself within a short period of time, it will become unresponsive, which will prompt the operating system to shut it down and diagnose it.

To allow my GUI to call a long-running process, I decided to use the multiprocessing library in conjunction with implementing a queue where outputs from long-running processes are polled periodically. The multiprocessing library will allow my GUI to bypass Python’s global interpreter lock (GIL) and spawn concurrent processes. Recall that the GIL prevents Python from running multiple threads at once. This is effectively dealt with by spawning sub-processes instead. On button click, the multiprocessing library spawns the desired subprocesses. The output of the subprocess is appended to the queue, where it can be fetched. The reason behind the success of this implementation lies in the fact that the GUI is not running the subprocess inside the loop where it is supposed to update UI elements. In this implementation, the only thing that the GUI is responsible for is fetching the output of the subprocesses from the queue. This fetching process takes but a fraction of a second, which means it does not disturb the flow of the main loop and will not render the GUI unresponsive.

# Results and Discussion

Table 1 splits up all 100 gestures in 3 categories: first 3 attempts, between 5 to 10 attempts, and beyond 10 attempts. Each category describes the number of attempts that were needed to get a correct translation of a gesture. Gestures belonging in the first category were gestures which the neural network was easily able to recognize. Most if not all the gestures in the category were recognized on the first attempt. Meanwhile, gestures belonging in the second category were gestures which the neural network was capable of recognizing, but with a little more difficulty. Detection of these gestures relied on accurately performing the versions of the gestures that the neural network was trained with. Putting personal variations on these gestures – no matter how slight – can cause occasional mispredictions. On the other hand, gestures belonging in the final category were gestures which the neural network consistently mispredicted. Many of these gestures overlap with other gestures in terms of the types of motion performed, which make them tricky for the neural network to recognize.

Table 1. Assessment of Performance through Measuring Number of Translation Attempts

|  |  |  |
| --- | --- | --- |
| **First 3 Attempts** | **Between 5 to 10 Attempts** | **Beyond 10 Attempts** |
| Sister, brother, aunt, uncle, separate, day, hot, cold, hamburger, egg, fork, hungry, socks, shoes, coat, hurt, bathroom, sleep, sad, sorry, bad, excuse, help, who, where, big, tall, full, more, blue, yellow, red, brown, orange, cost, bird, horse, sheep, pig, bug | Gold, silver, drink, spoon, angry, single, home, work, school, store, church, come, go, in, out, with, night, week, year, today, finish, hotdog, apple, cheese, cup, cereal, water, candy, cookie, cat | Mom, dad, marriage, grandma, grandpa, baby, pants, underwear, brush, nice, cry, like, good, love, please, thank you, what, when, why, how, stop, green, dollars, dog, cow, children, future, here, milk |

The accuracy of Sign AI in determining a gesture on its first try was also tested. The test was done a total of 5 times, with each gesture performed just once. The average number of gestures that Sign AI could guess correctly was 49. The results for each trial are recorded in Table 2. As expected, while performing the test, most of the gestures that were guessed correctly on the first try were the gestures that belonged in the first category of Table 1. In most of the trials, these gestures were guessed correctly by the neural network. On the other hand, there were times when the neural network guessed gestures in categories 2 and 3 (albeit, with lesser frequency, especially for gestures in category 3). These “lucky” guesses can explain why the range of correctly-guessed gestures between all trials is moderately high.

Table 2: Evaluating the Accuracy of Gesture Detection in First Attempt

|  |  |
| --- | --- |
| Trial Number | Correctly-Guessed Gestures (Out of 100) |
| 1 | 46 |
| 2 | 55 |
| 3 | 44 |
| 4 | 53 |
| 5 | 48 |
| Average | **49** |

There are potential reasons why the real-time performance of the translator pales in comparison to the performance projected in Keras. One main reason lies in the inevitable variations in EMG readings. In theory, EMG readings can be used to measure the activation of muscle from an electrical domain. However, certain factors throw off EMG readings, especially non-invasive EMG like surface EMG. Reaz, Hussain, and Mohd-Yasin state that a person’s physical makeup can very well influence their electrical state: “The amount of the tissue between contracting muscles and electrodes, along with their thickness, affects the amplitude of the EMG signal” [7]. Also, muscular diseases can lead to abnormal EMG readings, as can slight changes in skin conductance (which can be caused even by changes in one’s mood) [8], [9]. The point is that there is no surefire catch-all when it comes to EMG. Such is likely the reason why there is a performance gap between the performance projected by Keras and real-time performance.

# References

[1] J. Fileccia, “Sensitive care for the deaf: a cultural challenge,” *Creat. Nurs.*, vol. 17, no. 4, pp. 174–179, 2011.

[2] F. G. Bowe, B. T. McMahon, T. Chang, and I. Louvi, “Workplace discrimination, deafness and hearing impairment: The national EEOC ADA research project,” *Work*, vol. 25, no. 1, pp. 19–25, 2005.

[3] C. Lou Garberoglio, S. Cawthon, and M. Bond, “Deaf People and Employment in the United States,” *Washington, DC US Dep. Educ. Off. Spec. Educ. Programs, Natl. Deaf Cent. Postsecond. Outcomes*, 2016.

[4] M. Hermans and B. Schrauwen, “Training and Analysing Deep Recurrent Neural Networks,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 190–198.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[6] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, “How does batch normalization help optimization?,” in *Advances in Neural Information Processing Systems*, 2018, pp. 2483–2493.

[7] R. H. Chowdhury, M. B. I. Reaz, M. A. B. M. Ali, A. A. A. Bakar, K. Chellappan, and T. G. Chang, “Surface electromyography signal processing and classification techniques,” *Sensors (Basel).*, vol. 13, no. 9, pp. 12431–12466, Sep. 2013, doi: 10.3390/s130912431.

[8] C. Larivière, D. Gagnon, and P. Loisel, “The comparison of trunk muscles EMG activation between subjects with and without chronic low back pain during flexion–extension and lateral bending tasks,” *J. Electromyogr. Kinesiol.*, vol. 10, no. 2, pp. 79–91, 2000.

[9] E. Gatti, E. Calzolari, E. Maggioni, and M. Obrist, “Emotional ratings and skin conductance response to visual, auditory and haptic stimuli,” *Sci. data*, vol. 5, p. 180120, 2018.