STEP-BY-STEP: TRAINING IMU-BASED GESTURES WITH LIVE FEEDBACK

Michael Schnebly [1]

ABSTRACT

Recognizing user-defined gestures in inertial measurement unit (IMU) data unlocks new forms of creativity and accessibility in human-computer interaction. However, training gesture recognition models is a difficult task that requires a deep understanding of machine learning. We present Step-by-Step, a software tool that allows users to train gesture recognition models with live audiovisual feedback. Step-by-Step uses a simple neural network to learn to recognize and distinguish multiple gestures in IMU timeseries data. Users can train the model by performing gestures and receiving live feedback on the model's performance. Step-by-Step is designed to be accessible to users with no machine learning experience, while providing a powerful codebase for advanced users.

1 INTRODUCTION

For the past 40 years, the keyboard and mouse have been the primary means of human-computer interaction. The reliability and precision of these devices is undisputed, but it comes at a cost. The limited degrees of freedom available in keyboard and mouse inputs dictate which human body movements can be used to control computers, limiting both accessibility and creativity.

The ideal human-computer interface can adapt to a user's needs or preferences and can be used by anyone, regardless of their physical abilities. The challenge in developing such a device is that it must be able to learn, recognize, and distinguish the user's movements. This is a difficult task that requires a deep understanding of machine learning.

In this paper, we present Step-by-Step, a software tool that allows users to train gesture recognition models with live audiovisual feedback. Step-by-Step uses a simple neural network to learn to recognize and distinguish multiple gestures in IMU timeseries data. Users can train the model by performing gestures and receiving live feedback on the model's performance. Step-by-Step is designed to be accessible to users with no machine learning experience, while providing a powerful codebase for advanced users.

[1]School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA. Correspondence to: Michael Schnebly <michael_schnebly@g.harvard.edu>.

Final project of TinyML at Harvard University (CS249R - Fall 2023), Cambridge, MA, USA. Code available at www.github.com/michael-schnebly/step-by-step.

2 BACKGROUND

2.1 Gesture Recognition

Gesture recognition is the process of recognizing and distinguishing gestures in data. Gestures are body movement patterns that convey information. In a way, keyboards and mice perform gesture recognition by translating body movements into computer inputs. However, these devices are limited to a small set of predefined gestures. The goal of this work is to allow users to define their own gestures and use them to control computers.

2.2 Inertial Measurement Units

Inertial measurement units (IMUs) are one of many types types of sensors that can be used to measure body movement. IMUs typically include an accelerometer, gyroscope, and magnetometer, each of which measures in three axes: x, y, and z. IMUs are commonly used in wearable devices to track body movements over time. In this work, we use IMUs to measure body movement and recognize gestures.

2.3 Neural Networks

Neural networks are a class of machine learning models that can be used to analyze IMU timeseries and identify patterns such as gestures. Neural networks are composed of layers of "neurons", each of which takes in a set of inputs, performs a simple function, and produces an output. The outputs of one layer are used as inputs to the next. The final layer represents the model's prediction.

Neural networks are trained by providing them with examples of inputs and corresponding correct outputs. The model adjusts its parameters to minimize the difference between its predictions and the groundtruth outputs. This process is

called gradient descent and is performed iteratively until the model converges on a solution. Once trained, the model can be used to make predictions on new inputs.

IMUs and neural networks are well-established and have been used together in a wide range of applications for characterizing body movements. However, a machine learning system that can be easily trained to recognize user-defined gestures with performance rivaling that of the mouse and keyboard remains elusive. The goal of this work is to lay the foundation for such a system.

3 METHOD

The system consists of hardware and software components. The hardware measures body movement and streams the data to a computer. The software processes the data, trains a neural network, and provides live feedback to the user. The system is designed to be accessible to users with no machine learning experience while providing a powerful codebase for advanced users.

3.1 Hardware

For measuring body movement, we used the Bosch BNO-055, a 9-axis IMU that includes an accelerometer, gyroscope, and magnetometer. The IMU uses I2C communication protocol to stream data to a microcontroller (Espressif ESP8266) which passes that data on to a laptop (Macbook Pro 2017) via USB for further processing.

Note that the particular hardware components used in this project are not essential to the system. They could be replaced with other options so long as the sensor data contains information necessary to recognize and distinguish the gestures of interest and the computer (or microcontroller if using purely embedded approach) is capable of running the software at a rate matching that of data collection.

3.2 Software

The software is written in Python and structured into three main components: data processing, neural network, and user interface. The data processing component handles data collection, labelling, and storage. The neural network component handles model structure, training, and inference. The user interface component handles user interaction and feedback.

To facilitate realtime training and inference, the software is structured as a pipeline with the following processes running in parallel: data processing, model training, and model inference. Importantly all processes refer to the same global variables, allowing them to share updates to the model and training data in realtime.

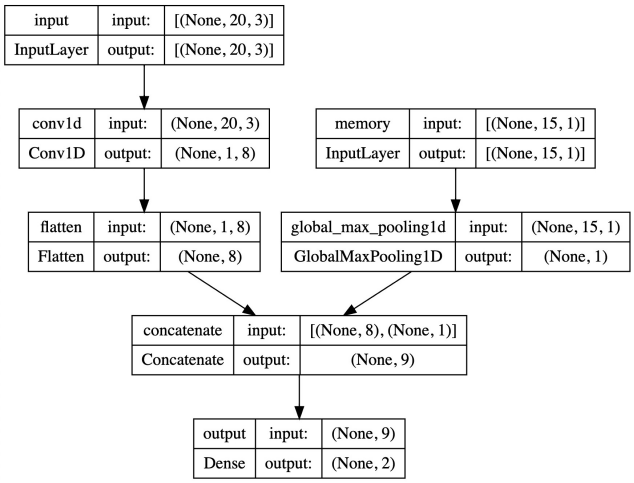

Figure 1. Neural Network Architecture. The sensor branch (left) receives 20 frames (200 ms at 100 Hz) of 3-channel (x, y, z) IMU data. The sensor branch feeds into a 1D-convolutional layer with 8 filters, each also 20 frames in width. The memory branch (right) receives 15 frames of 1-hot encoded prediction data. The two branches are concatenated and fed into a fully connected output layer to predict the currently active gesture.

3.3 Data Processing

3.3.1 Collection

The IMU data is collected as a timeseries of frames. Each frame is a 3D vector representing linear acceleration in each of the three axes: x, y, and z. The data is collected at a rate of 100 Hz and can be streamed directly from an IMU, recorded to a file, or loaded from a file.

3.3.2 Labelling

The data is labelled by associating each frame with a gesture. A gesture is a sequence of frames that represents a single movement of the body. Here, we focus on percussive gestures and assume that a gesture is complete at the local peak in linear acceleration's magnitude. This constraint allow us to achieve realtime, automated labelling of gestures using a simple peak-detection algorithm. When a local peak meet's user-defined criteria (e.g. acceleration is above a certain threshold), the peak frame is labelled as containing a gesture. All other frames are labelled as containing no gesture.

3.4 Neural Network

3.4.1 Structure

The neural network is a shallow, two-branched model. One branch represents a recent history of sensor values while the

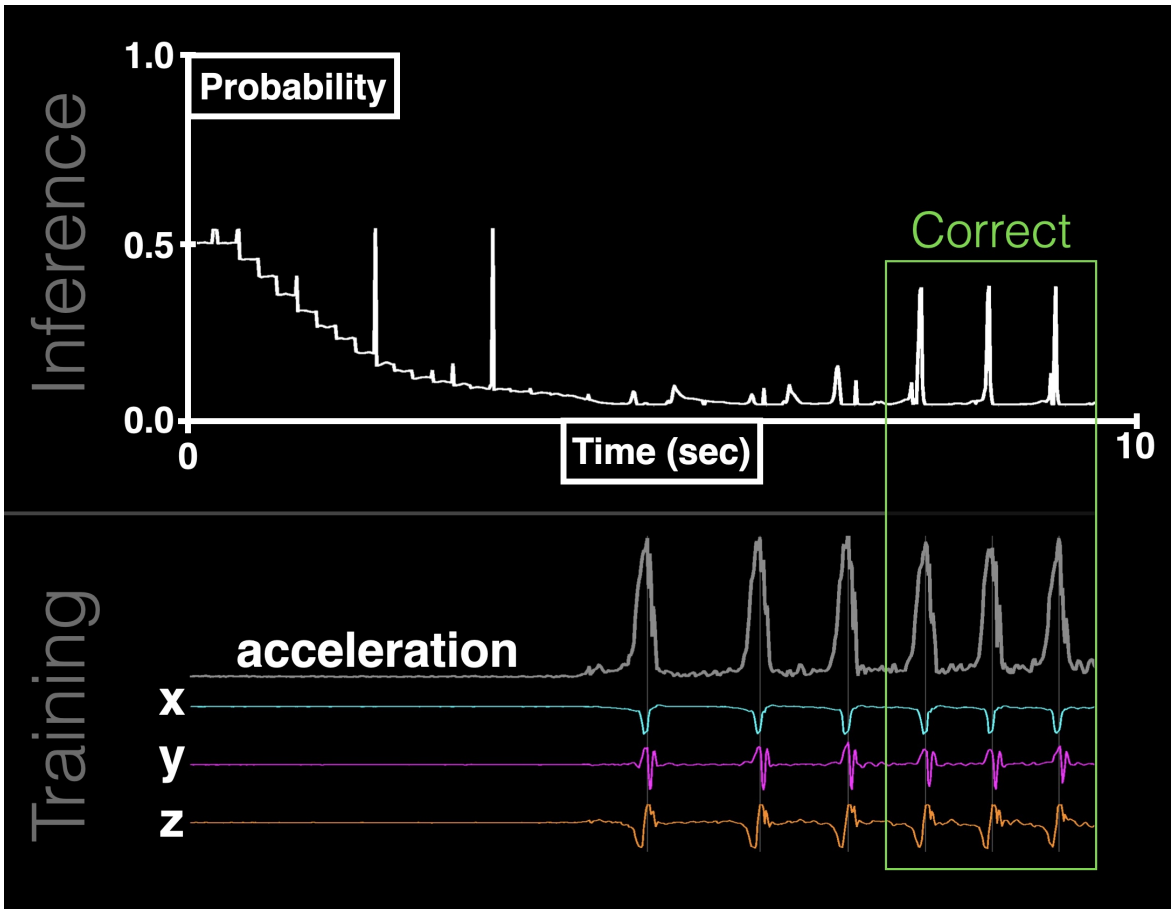

Figure 2. Live Learning Feedback: The user performs a gesture and receives live feedback on the model's performance. Bottom Panel: 10 seconds of training data. Acceleration components (x, y, z) are shown in color; magnitude is shown in grey. Each magnitude peak is marked by a thin vertical line, indicating it has been labelled as a gesture in the training data. Top Panel: 10 seconds of inference data. At t = 0 seconds the neural network is completely untrained. Naive to what is and is not a gesture, it predicts gesture and non-gesture with equal probability (0.5). As the model trains on non-gesture date, it begins to predict non-gesture with higher probability. At t = 5 seconds, the model has converged on a solution that reliably distinguishes non-gesture. At t = 10 seconds, the model has converged on a solution that reliably distinguishes gestures. Given more training data and time to learn, the model's performance would continue to improve with gessture and non-gesture probabilities approaching 1 and 0, respectively. See supplementary video for live demo in the github repo.

other represents a recent history of model outputs. Combining the information from the these two sources, the model is trained to maximize the probability of the correct gesture and minimize the probabilities of all other gestures.

The sensor branch consists of an input layer (short timeseries of recent sensor values) followed by a a 1D convolutional layer (recognizes patterns in the timeseries). The memory branch consists of an input layer (short timeseries of recent model outputs) followed by a max pooling layer (summarizes the timeseries of recent model outputs based on its highest values). These branches are then concatenated and followed by a fully connected output layer.

3.4.2 Realtime Training

Training is performed using Tensorflow's default gradient descent optimizer. The training data consists of a timeseries of frames, each of which is labelled with a gesture. The model is trained on a sliding window of frames where window size is a user-defined parameter.

The window slides forward one frame at a time. If the most recent frame in a window is labelled with a gesture, the window is labelled with that gesture. Otherwise, the window is labelled with no gesture. As new windows are collected, the model is trained on the windows using gradient descent. Training is iterative and the model's training data is updated in realtime, allowing the user to receive live feedback on the model's performance. This process is illustated in Figure 2.

In principle this training could be done stochastically, training on each frame's window as it is collected. However, this would would require a more sophisticated training algorithm that Tensorflow's built-in gradient descent optimizer (at least given the computational performance of the author's machine). For this reason, this proof-of-concept codebase defaults to training on batches of recent windows. When training completes, the model is updated, the training data is updated with recent frames, and training begins again.

3.4.3 Realtime Inference

As the model is updated, its performance is tested on the most recent frames using Tensorflow. Ideally, this would be done on a frame-by-frame basis, but computational constraints required the authors to batch the frames and perform inference on batches.

Along with the groundtruth labels, the model's predictions are used to generate live feedback for the user using both auditory and visual cues. The auditory cue is a simple synthesizer that plays a tone corresponding to the model's prediction. The visual cue is a simple GUI that displays the IMU timeseries, model's predictions, and groundtruth labels in realtime. The visual elements are illustated in Figure 2.

3.5 User Interface and Control

All rendering is done using OpenGL. With keyboard inputs, the user can control various aspects of the application. The user can start and stop data collection, data labelling, model training, and model inference.

4 Results

After just a few examples of a gesture, the model converges on solution that reliably distinguishes gesture from non-gesture. Performance improves as more examples are provided. Here, the authors only test the model's ability to recognize a single gesture, but based on prior experience using this model in non-realtime training scenarios, the authors expect the model to be able to distinguish multiple gestures with high accuracy. A representative example of the model learning to recognize a single gesture in just 10 seconds is shown in Figure 2.

5 Next Steps

There are many ways in which the system could be improved in terms of hardware, software, and user experience.

The current hardware system uses a microcontroller to stream IMU data to a computer for processing. However, given the small size of the model (default settings: 2kb with 500 params) and recent advances in embedded machine learning, it should be possible to run the entire system in realtime on a wireless, wearable microcontroller, including labelling, training, and inference. This would allow the system to be used in a wider range of applications, including those where a computer is not available, desirable, or practical. To this end, the authors are currently exploring the possiblity of using Espressifs ESP32-S3 chip, which includes a neural network accelerator and is capable of running Tensorflow Lite Micro.

The current software is written in Python and uses Tensorflow for training and inference. This solution is relatively simple, but is not ideal for realtime applications. The authors are currently exploring the possibility of rewriting the software in C++ and using Tensorflow Lite Micro for training and inference. This would allow the system to be run on a microcontroller, as described above.

The current user interface is a simple OpenGL-based GUI that displays the IMU timeseries, model's predictions, and groundtruth labels in realtime. This solution is relatively barebones and should work on most operating systems, but it is dependent on streaming the data to a personal computer. The authors are considering the minimal user-interface necessary to run the system on a microcontroller, including a simple LCD display and a few buttons for control.

6 Closing Remarks

Step-by-Step serves as a proof-of-concept for realtime learning of user-defined gestures in IMU data. It is worth noting that the strategies employed here are not limited to IMU data and could be applied to data from other sensors. The authors hope that this work will inspire others to explore the possibilities of realtime machine learning in human-computer interaction.

7 Acknowledgements

The authors would like the thank Harvard's TinyML course staff for their guidance and support throughout the semester, including: Jason Jabbour, Ike Uchendu, Jessica Quaye, Matthew Stewart, and Vijay Janapa Reddi.