6/13/2014

Smart Weights

California Polytechnic State University

Luke Rafla-Yuan and Austin Fox, advised by Tina Smilkstein
ELECTRICAL ENGINEERING DEPARTMENT

Table of Contents

List of Tables and Figures

Abstract

The goal of this project is to design and implement weights which can record and analyze work out patterns. Motivation for this project stems from the high cost of personal training. The hope is that this device will provide many of the benefits a user receives from personal training at only a fraction of the cost. The Smart Weight is designed with an on-board Inertial Measurement Unit providing acceleration, gyroscope, and magnetometer data. A microcontroller records and analyzes changes in motion, feeding this data into Multiplicative Recurrent Neural Network (MRNN) for exercise classification. A Raspberry Pi was chosen as the microcontroller, along with a Polulu Minimu-9 V2 for the IMU. These were attached to a five pound free-weight, where the motion of an exercise could be accurately recorded. The IMU communicates with the Raspberry Pi via the i2c protocol, and provides roughly 50 data points per second. Code was written to preprocess and feed data from the IMU into the MRNN, where the type of exercise can then be determined. The MRNN was trained on graphics processing units (GPUs) with the help of Ersatz Labs, a company that specializes in training Neural Networks. The prototype Smart Weight is able to classify one exercise (the bicep curl) with an accuracy of over 90%, but many more exercises will be added in the future.

Introduction

The Smart Weight is a device that attaches to a dumbbell, tracks a user's movements, and is able to accurately identify a wide variety of exercises. In order to accomplish this, a nine-axis IMU (Inertial Measurement Unit) will be needed, providing accelerometer, magnetometer, and gyroscope data to an on-board cpu. The data will then be processed by an MRNN (Multiplicative Recurrent Neural Network) in order to identify if a known exercise has been performed.

As a prototype, the device uses the raw x,y,z acceleration values. The possibility of using the magnetometer and normalized euler angles to remove the device's orientation from the equation was explored. However, it was discovered that the ferrous material of the weight interfered with the magnetometer data, and the method was discarded. The raw x,y,z acceleration values are sufficient to identify exercises as long as the Smart Weight device is oriented the same way each time.

Figure 1: Drawn model of the Smart Weight device

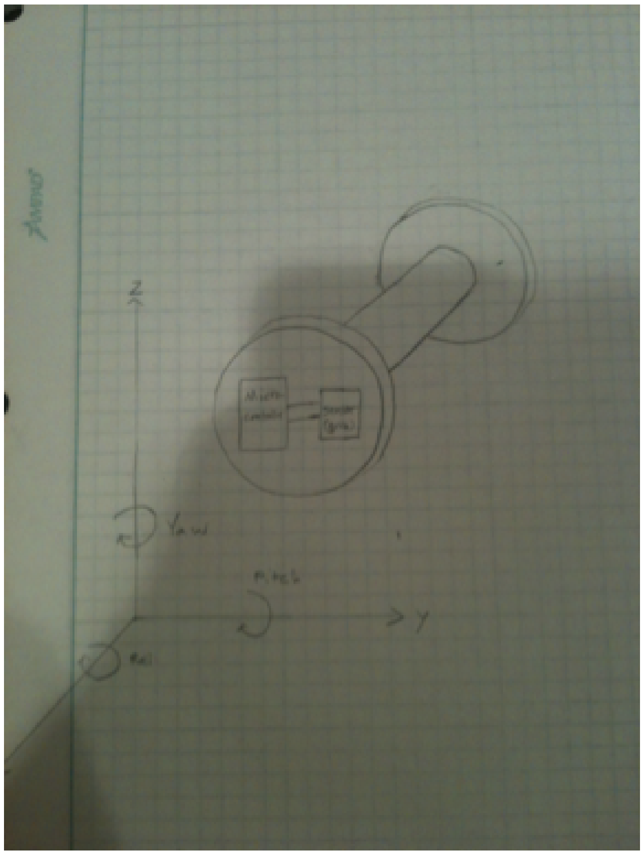


Figure 1 shows a model of the Smart Weight device implemented on a free weight. There is an x,y,z coordinate system drawn for reference with yaw, pitch, and roll euler angles. For the part of project that was done for prototyping purposes, the euler angles are disregarded and only the x,y,z acceleration vectors are used.

<u>Engineering Specifications</u>

At the beginning of the project a list of engineering specifications were created for the device. During the construction of Smart Weights, steps were taken to create the device with these constraints.

- Accurate sensor that can measure within 5% deviation of 3D acceleration vectors.
    - Sensor must be accurate enough to distinguish and measure exercises. It must also allow for correct exercise identification dependant on device orientation.
- Finalized prototype with complete on board microcontroller and IMU.
    - Need a demoing platform to display.
- Neural network algorithm can distinguish between exercises and non-exercises.
    - The Smart Weights can automatically measure and track exercises throughout workouts and pauses in workouts.

Competitor Products

- Moov
    - A beta device including nine-axis motion sensors and CPU in a small portable package
- Atlas Wearables
    - Also a beta device, includes nine-axis IMU and has a screen on a wristband to display progress

Table I: Comparison of Competitor Products

Moov		**Atlas Wearables**		**Smart Weights**	
Good	**Bad**	**Good**	**Bad**	**Good**	**Bad**
Small	Still only a beta project	Small	Still a beta project	Small	Not waterproof
Refined look	No determined cost yet (will be over the pre-order price of $69.95)	Portable	No smartphone app, only has API to integrate with others	Portable	Will not have ability to train new exercises on the device (at first)
Portable	No feedback about performance yet because it is only a beta project	Waterproof	The pre-order price is $179.99 and this could go up once the device is manufactured	Can be attached to any dumbbell	Will not have the ability to correct user's form
Can be attached to any dumbbell		Will be able to monitor multiple exercise	No feedback about performance yet because the device has not been released	Can be purchased by gyms and loaned out to customers	No feedback about performance yet because the device has not been released
Waterproof		New exercises can be trained directly to the device		Focus is on exercises performed in a gym. This allows for a more refined experience, while also tracking a wider range of exercises in the gym	No certified or professional help confirming correct exercise form
Will be able to monitor multiple exercises		Will have a corresponding iPhone app which the Atlas Wearable will automatically connect to and		iPhone app will offer helpful features including a competition feature, where friends nearby can comparte	

		provide real time information		workouts and keep each other accountable	
Will have a corresponding iPhone app which the Moov will automatically connect to and provide real time information		Has a community of shared data to help provide useful feedback on form		A website will be available where users can see all their data across a wide range of devices. This will also offer an API to integrate with other fitness trackers in order to track overall health of an individual	
Is tested and advised by professional athletes and certified coaches					

While there are two large competitors in the field of tracking weight lifting, the Smart Weight will excel by being a cheaper alternative to these two products. A website that allows users to go on and have access to more in-depth analytics of their performance in combination with a community of users providing workout data will give the Smart Weight an edge over the Moov and Atlas.

Design

Figure 2: High-Level block diagram

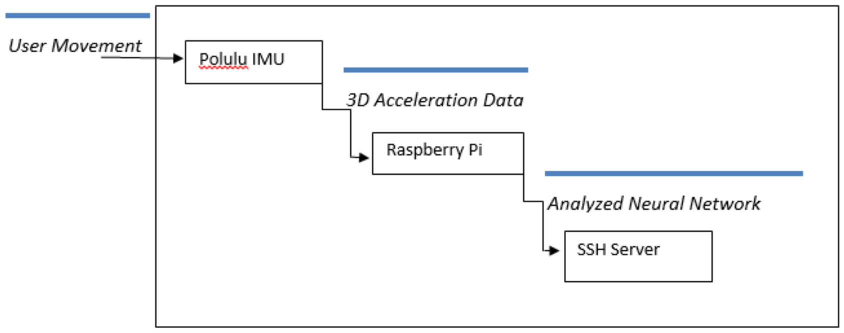


Figure 2 is a block diagram of the overall system. The Polulu MinIMU-9 module outputs X, Y, and Z acceleration data to the Raspberry Pi. The Raspberry Pi receives this data via the i2c connection and analyzes it in real-time with the neural network algorithm. The exercise data is stored on the Raspberry Pi, but can be accessed by a device on the same local network via the SSH protocol.

Table II: Block diagram inputs and outputs

Input/Output	**Name**	**Function**
Input	User Movement	User performing exercise
Input/Output	3D Acceleration Data	Supplies X, Y, Z data via i2c protocol
Input/Output	Analyzed Neural Network	Provides user with info

Table II describes the block diagram outputs and inputs in a table format. The functions of these input/outputs are also described.

Parts

Table III: Cost Estimate

Item	Number	Company	Cost
Raspberry Pi	1	PiBorg	\$50.00
Polulu gyroscope and magnetometer	1	Polulu	\$40.00
Free Weight	1	Unknown	\$20.00
Labor Costs (\$10 per hour)	165		\$1650.00
		Total	\$1760.00

Table III shows cost for parts and labor. The labor cost was derived by applying a wage of \$10 per hour for each hour worked on the project.

Table IV: Cost Approximations

Cost Types	Dollar Values
$Cost_a$	\$760.00
$Cost_b$	\$2760.00
$Cost_m$	\$1760.00

Table IV shows the optimistic, pessimistic, and most likely total costs for the project. These approximated costs are used in the following equation from *Ford & Coulston* to calculate project total cost.

$$\textbf{Cost} = (\textbf{Cost}_a + \textbf{Cost}_b + 4\textbf{Cost}_m)/6$$

$$= \$2760.00$$

Construction

Figure 3: IMU (Inertial Measurement Unit)

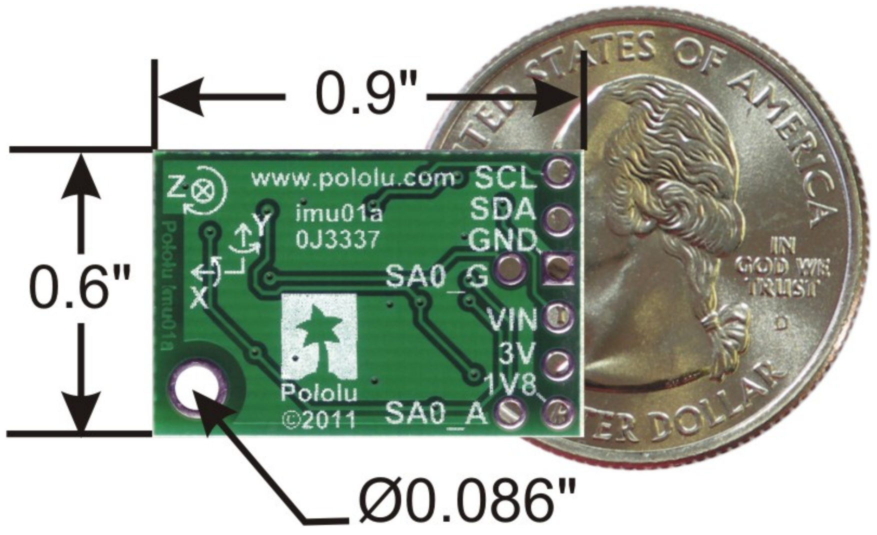


Figure 3 shows the Polulu Minimu-9 V2 that was used in the Smart Weights project. A quarter is also included in the image for size reference. The website for the product is: http://www.pololu.com/product/1265

Table V: Example IMU data

X_Mag	Y_Mag	Z_Mag	X_Accel	Y_Accel	Z_Accel	X_Gyro	Y_Gyro	Z_Gyro
-138	129	-416	112	-8	228	-50	14	9
-138	129	-419	120	-4	232	-49	20	18
-138	129	-419	116	-12	228	-51	15	8
-138	129	-419	116	-12	228	-50	21	17
-137	130	-421	116	-8	232	-51	22	11
-137	130	-421	120	-12	220	-56	20	14

Table V shows the example data provided by the Polulu IMU. This 9-axis IMU provides acceleration, gyroscope, and accelerometer data in the above format.

Using the open-source software:
Name: Minimu9-ahrs
Download: https://github.com/DavidEGrayson/minimu9-ahrs
Wiki: https://github.com/DavidEGrayson/minimu9-ahrs/wiki

This software is able to calibrate the device and fuse the magnetometer and gyroscope data in order to output the orientation of the device in the form of a direction-cosine matrix, quaternion, or euler angles.

The project used a python script to gather real-time data from the c-program. It is possible to remove gravity from the x, y, and z acceleration vectors knowing its orientation, but raw acceleration values were chosen because the orientation of the dumbbell would stay consistent throughout the exercises performed.

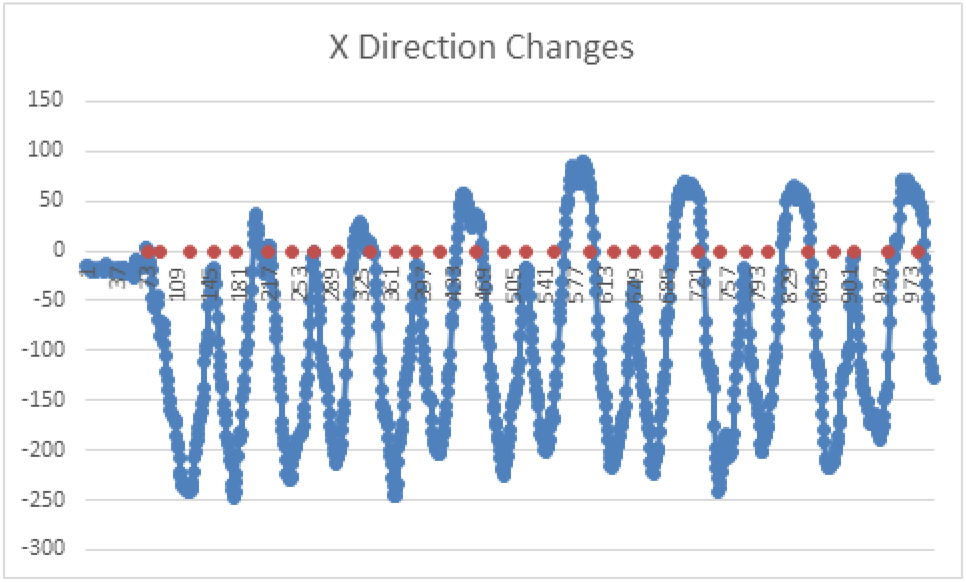
Here is a portion of the code written to detect changes in motion of each separate axis. This was then utilized this to help analyze when exercises are started and stopped to be deciphered by the neural network algorithm.
The code also attempts to filter any noise that could be encountered from the inputs.

```
if x > prev_x:
        x_counter_1 = x_counter_1 + 1
        if x_counter_1 > 5:
                x_max = x
                x_counter_0 = 0
                if x_switch == 1:
                        sheet1.write(i-5, 5, "-500")
                        x_switch = 0
  else:
        x_counter_0 = x_counter_0 + 1
        if x_counter_0 > 8:
                x_min = x
                x_counter_1 = 0
                if x_switch == 0:
                        sheet1.write(i-8, 5, "500")
                        x_switch = 1

  if y > prev_y:
        y_counter_1 = y_counter_1 + 1
        if y_counter_1 > 5:
                y_max = y
                y_counter_0 = 0
                if y_switch == 1:
                        sheet1.write(i-5, 6, "-500")
                        y_switch = 0
  else:
        y_counter_0 = y_counter_0 + 1
        if y_counter_0 > 8:
                y_min = y
                y_counter_1 = 0
                if y_switch == 0:
                        sheet1.write(i-8, 6, "500")
                        y_switch = 1

  if z > prev_z:
```

```
                z_counter_1 = z_counter_1 + 1
                if z_counter_1 > 5:
                        z_max = z
                        z_counter_0 = 0
                        if z_switch == 1:
                                sheet1.write(i-5, 7, "-500")
                                z_switch = 0
        else:
                z_counter_0 = z_counter_0 + 1
                if z_counter_0 > 8:
                        z_min = z
                        z_counter_1 = 0
                        if z_switch == 0:
                                sheet1.write(i-8, 7, "500")
                                z_switch = 1
```

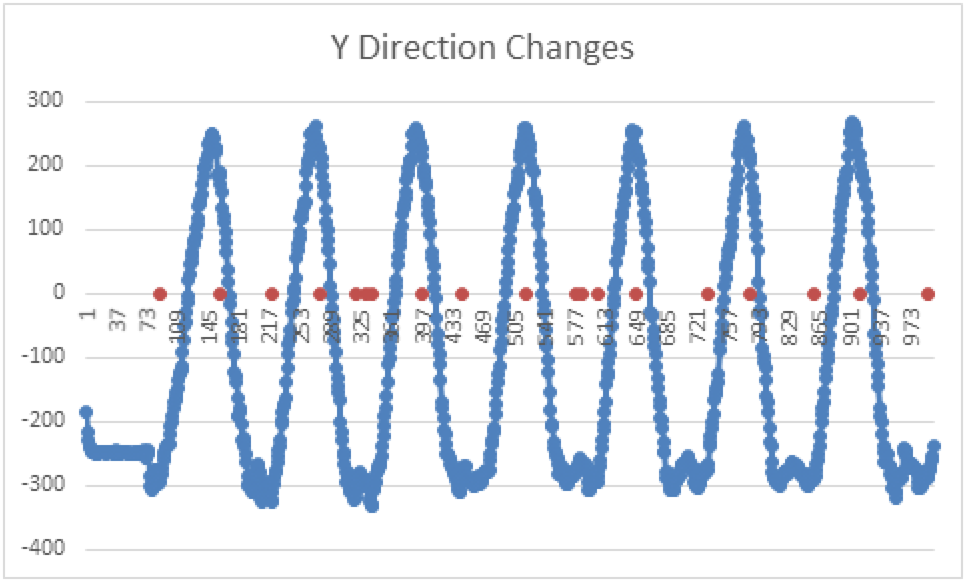
Test exercises were performed, and graphs were created with the resulting exercise and direction change data. The data is shown below in the following three graphs.

Graph 1: IMU X direction changes



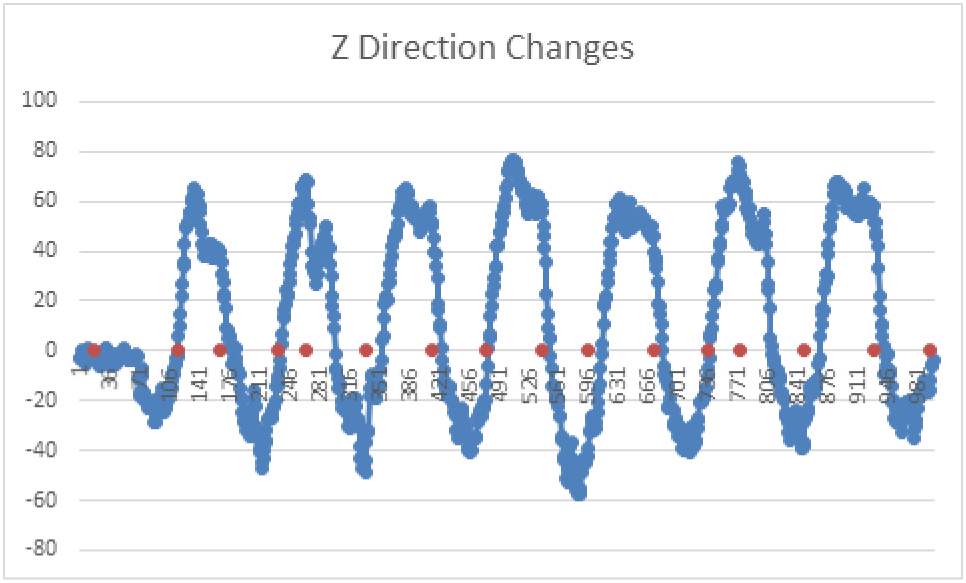
Graph 1 shows x acceleration data of repeated bicep curls. As shown in the data, there is is an identifiable pattern that can be used to recognize future exercises. The red dots show locations where the IMU changes direction (switches sign) in the x direction. This was identified automatically by a python script written for the project that helps analyze the exercise data.

Graph 2: IMU Y direction changes



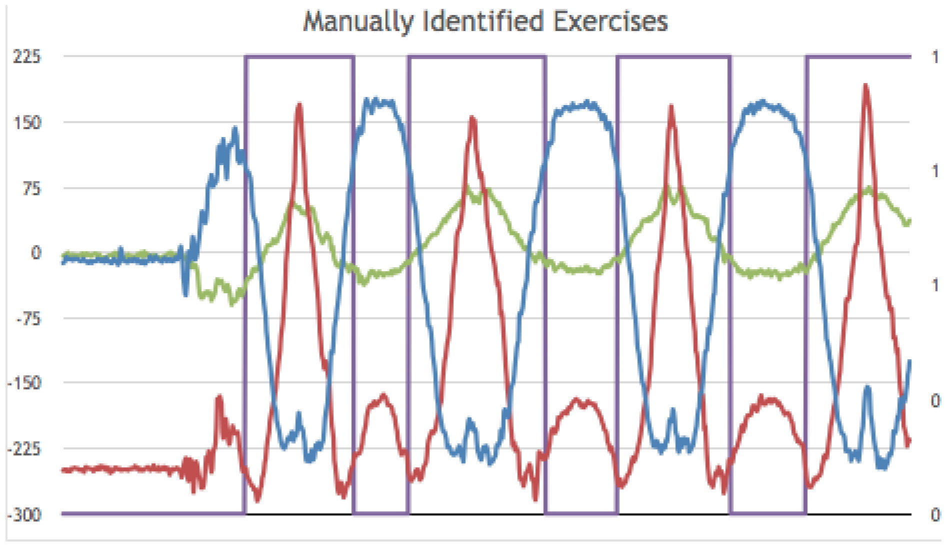
Graph 2 shows y acceleration data of repeated bicep curls. The analysis of this data is the same as described for the x acceleration data.

Graph 3: IMU Z direction changes



Graph 3 shows z acceleration data of repeated bicep curls. The analysis of this data is the same as described for the x acceleration data.

Graph 4: Manually identified exercises using x,y,z acceleration vectors



Graph 4 shows the acceleration data for the x,y, z vectors with the exercises manually identified. The purple line is the network constant that goes high when a exercise is being performed. The patterns for the exercises were identified by performing single exercises and overlaying them onto repeated exercises.

<u>Artificial Neural Networks:</u>

Artificial Neural Networks (ANN) are digital models of a biological brain and the neurons that it consists of. The ANN is part of a family of machine learning techniques with the goal of feature recognition in large sets of data. Each digital neuron is given one or more inputs, each with their respective weights, and a threshold that the sum of these inputs must equal in order for the neuron to ‘fire’. In modern neural networks, a continuous function is used to model this firing, such as the sigmoid function. This function takes any any value as an input and normalizes it on a scale from 0 to 1. This allows for a much closer approximation when trying to train the network with sample input and output data.

Figure 4: Neural Network with Sigmoid Neurons

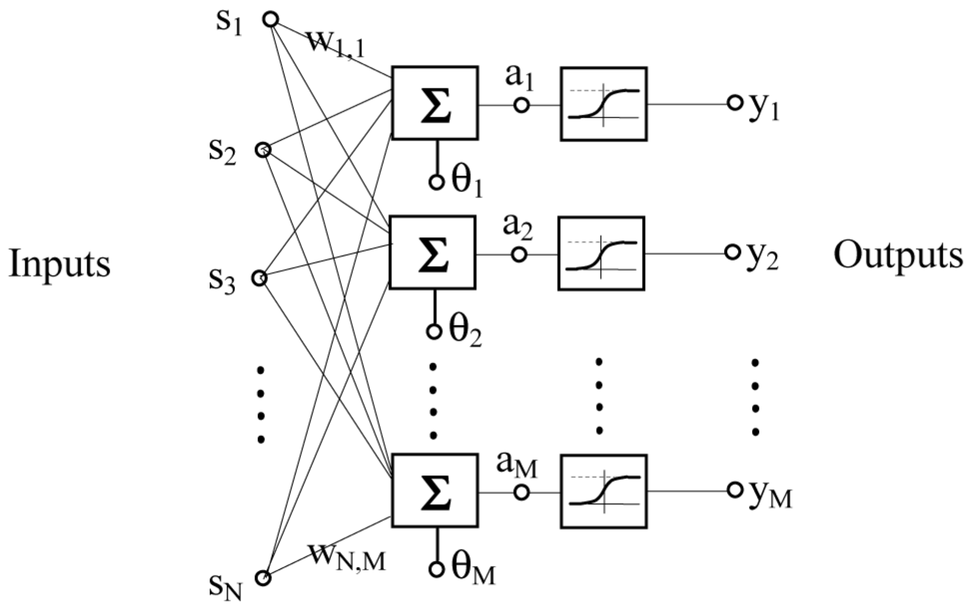


Figure 4 shows an example of an Artificial Neural Network with sigmoid neurons. Only one layer of the network is depicted, with the inputs being fed into a summing junction. The output of these neurons ranges from 0 to 1, determined by the sigmoid function.
Image credit: “Pattern recognition with ultrasonic sensor: a neural network evaluation” by J.R. Llata, E.G. Sarabia, and J.P. Oria.

The drawback of the traditional feedforward neural network is that all the data required for the network to make a decision must be present at the same time. While this is useful in many situations, it does not work well with time-series pattern recognition. In order to account for this using the traditional feedforward neural network, a moving window consisting of numerous timesteps would need to be fed into the network in order for it to have a chance of recognizing an exercise. This is highly inefficient and could easily prove ineffective. Neural networks with enough inputs for all the data required to recognize an exercise would need to be quite large and would be difficult to train correctly. Another issue lies in the fact that the exact length of the exercise is not known, and thus a specific window size to be used is not known.

The remedy for these issues is to use a variant of the traditional neural network known as a Recurrent Neural Network (RNN). The difference with an RNN is in the use of feedback. The output of the network is fed back in as an input, allowing for memory to be stored in the device. Previous inputs to the network can affect the current output, and time-series patterns can thus be recognized. RNNs can be thought of as extremely deep neural networks with weights shared across timesteps.

A known issue of RNNs is that, when training, the learning gradient can either vanish or explode quickly. In order to address this problem, a modified RNN known as an Multiplicative Recurrent Neural Network (MRNN) has been chosen for this project. The difference between an MRNN and an RNN is that MRNNs allow the current inputs to determine the weight matrix from one hidden state to the next. This small adaptation solves the issue of convergence and divergence, and has breathed new life into the traditional RNN once thought too hard to train properly.

Figure 5: Multiplicative Recurrent Neural Network (MRNN)

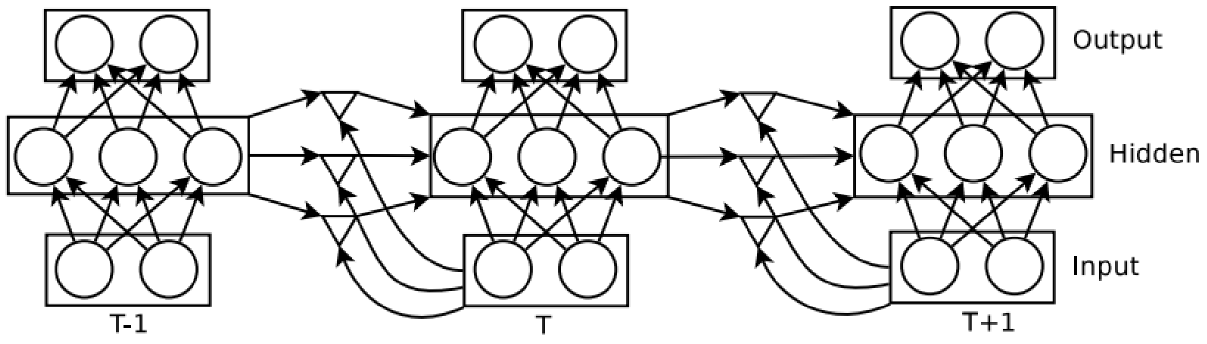


Figure 5 shows a diagram of Multiplicative Recurrent Neural Network structure. Triangles represent the control of the current input on the weight matrix for the previous state. This is an important adaptation in MRNNs, as it eliminates the issue of convergent and divergent learning gradients seen in RNNs.

Image credit: “Generating Text with Recurrent Neural Networks” by Ilya Sutskever, James Martens, and Geoffrey Hinton

MRNN Algorithm:

Definitions:

h_t = hidden state

X_t = input vector

W_{hx} = hidden-to-input weight matrix

$W_{hh}^{(X_t)}$ = hidden-to-hidden weight matrix

W_{oh} = hidden-to-output weight matrix

b_h = hidden bias vector

b_o = output bias vector

O_t = output vector

In order to train a Multiplicative Recurrent Neural Network, the hidden-to-hidden weight matrix, hidden state, and output vector (exercise recognition layer) must be updated for every timestep (input vector). The following equations can be used to update the matrices and vectors for each input vector:

for t = 1 to T:

(1) $W_{hh}^{(X_t)} = \sum_{m=1}^{M} X_t^{(m)} W_{hh}^{(m)}$

(2) $O_t = W_{oh} h_t + b_0$

(3) $h_t = tanh(W_{hx} X_t + W_{hh}^{(X_t)} h_{t-1} + b_h)$

The storage of the tensor $W_{hh}^{(X_t)}$ becomes too impractical for an MRNN with a large amount of hidden units and inputs, so it is factored and the follow equations are derived:

(4) $W_{hh}^{(X_t)} = W_{hf} \cdot diag(W_{fx} X_t) \cdot W_{fh}$

(5) $f_t = diag(W_{fx} X_t) \cdot W_{fh} h_{t-1}$

(6) $h_t = tanh(W_{hf} f_t + W_{hx} X_t)$

(7) $O_t = W_{oh} h_t + b_0$

Note: The standard method of gradient descent for training the MRNN is not sufficient and will not lead to a good result. A 2nd order learning method such as Hessian-Free Optimization is a much better approach.

Training and Testing

The first step in training the MRNN was to decide what the inputs and outputs of the network would be. For the Smart Weight prototype, one exercise - the bicep curl - was chosen to use for training. This allows for only two outputs for the MRNN, with a neuron representing a ‘1’, meaning a bicep curl has been performed, and a ‘0’ meaning that one has not been performed. The inputs to the network were the raw acceleration values obtained from the IMU - raw x, y, and z acceleration. The values were first normalized on a scale from 0 to 1, with 0 being the minimum value obtained from the training data and 1 being the maximum. Every value in between was mapped linearly to a fraction of 1. A dataset of 30 bicep curls was recorded, resulting in approximately 6500 timesteps of data. The IMU records roughly 50 timesteps per second, and each exercise lasted an average of 140 timesteps, or 2.8 seconds. An equal number of timesteps corresponding to no exercise were also recorded. These movements ranged from staying still, random movements, and other exercises that would be trained later. The goal was to make sure the network would only output a ‘1’ if the exercise was actually a bicep curl, and a ‘0’ for any other movement.

The data was grouped into sets of 140 timesteps and trained using help from Ersatz labs. Ersatz specializes in training Neural Networks using graphics processing units (GPUs). This method drastically reduces the time required to train an MRNN, meaning more iterations can performed to increase accuracy. The CEO of Ersatz, Dave Sullivan, personally overlooked the training of the Smart Weight MRNN and was crucial in helping to adjust the hyperparameters to ensure optimal network training.

Results:

Figure 6: Change in % accuracy of MRNN predictions vs. iterations

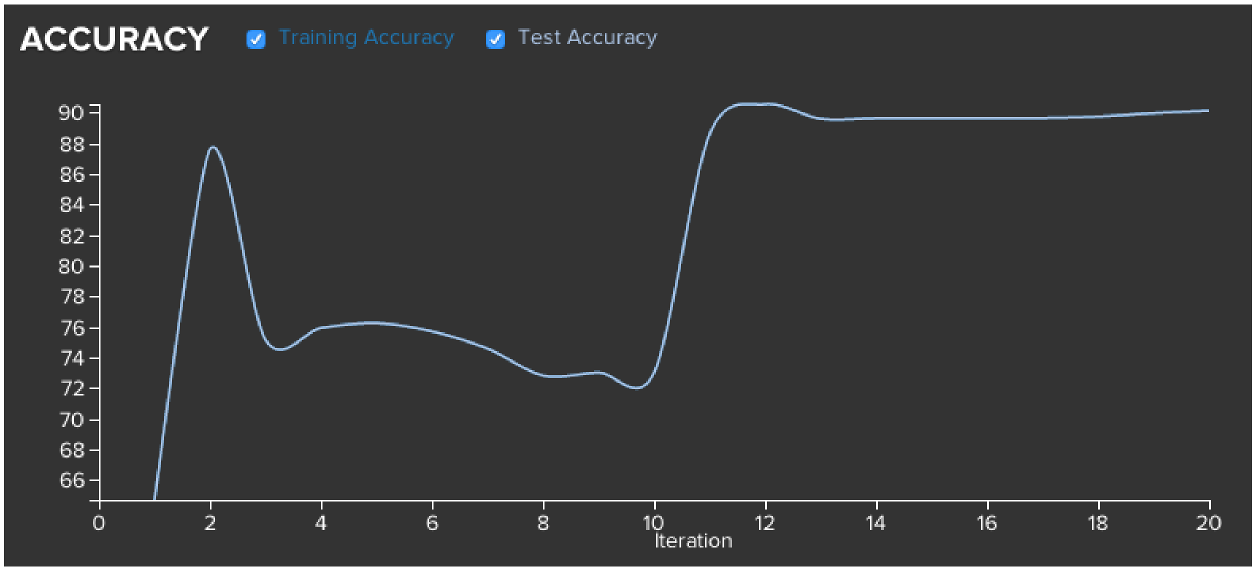
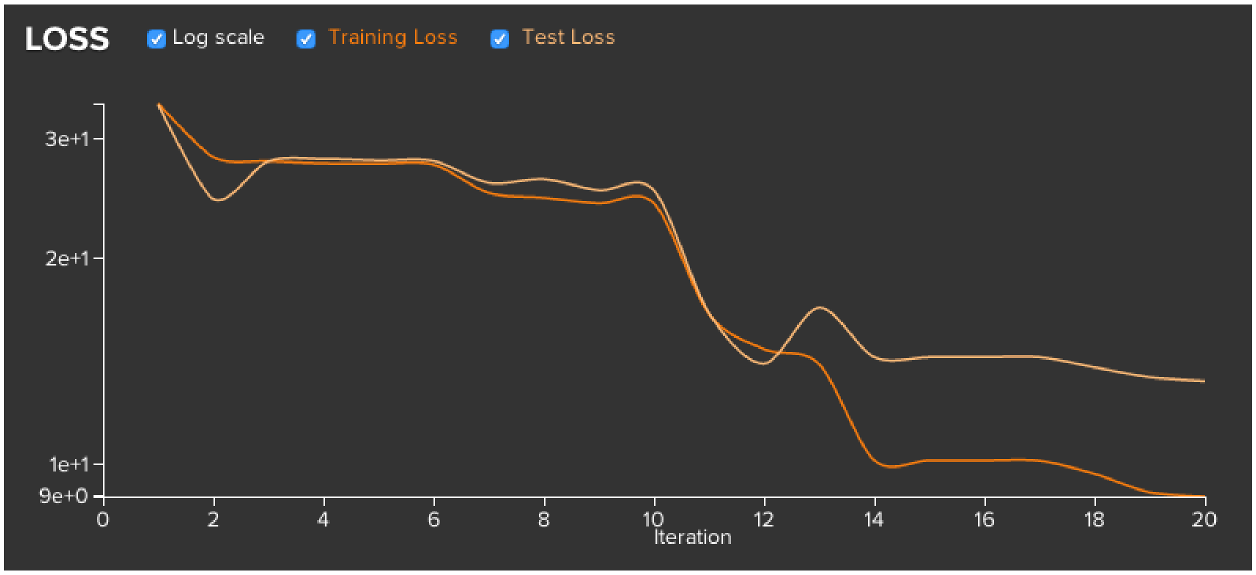
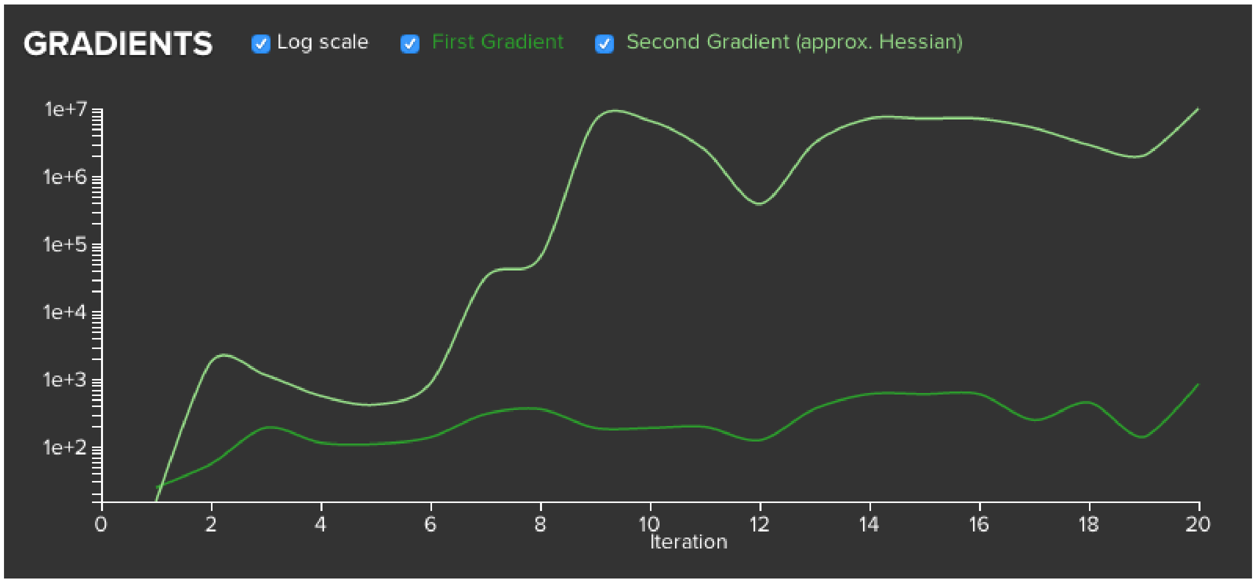


Figure 7: Change in % error of MRNN predictions vs. iterations



Figures 6 and 7 show how accurate the MRNN is at classifying an exercise as either a bicep curl or some other movement. After each iteration, the error of this classification decreases, showing that the network is truly learning how to classify the vairous movements used for training. By the second iteration, the network was already close to its peak accuracy, but it proceeded to to drop back down for a few iterations. Once the network weights were close to optimal levels given the dataset used, the accuracy started to flatten out around 90%, meaning it is able to correctly recognize if a bicep curl is being performed 9 out of 10 times.

Figure 8: Change in 1st and 2nd order derivatives for learning vs. iterations



The change of the first and second derivative (Hessian-Free Optimization) is shown in figure 8. The graph illuminates why the second gradient is necessary; the first derivative stays relatively flat over each iteration, while the second gradient increasing rapidly allowing for the network to learn more effectively.

Figure 9: Change in damping values vs. iterations

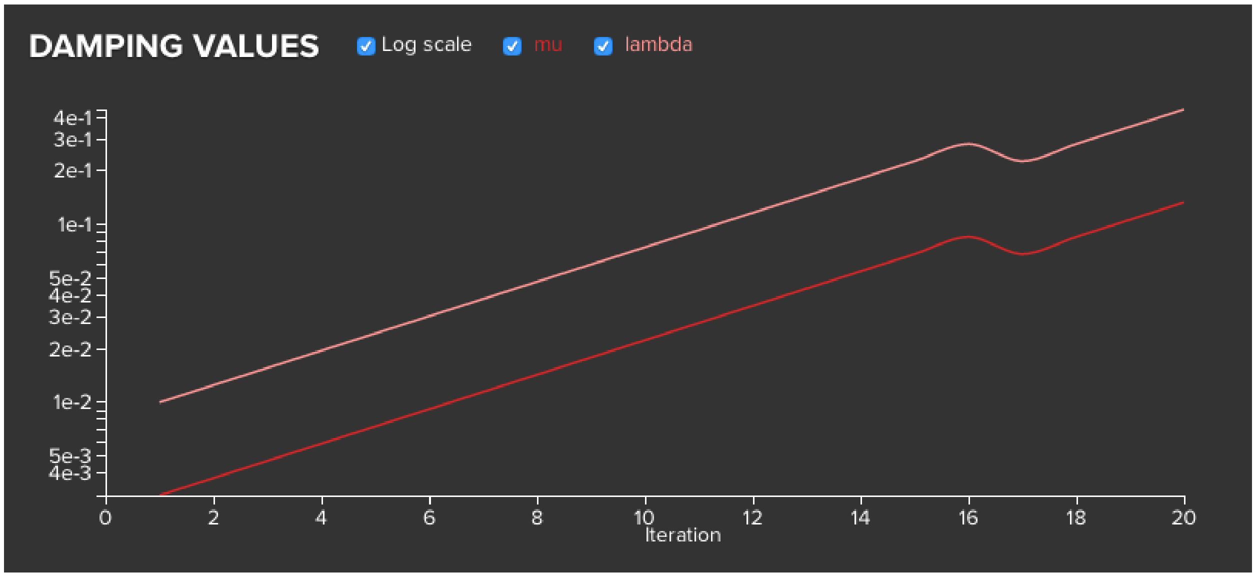


Figure 10: Change Conjugate Gradient vs. iterations

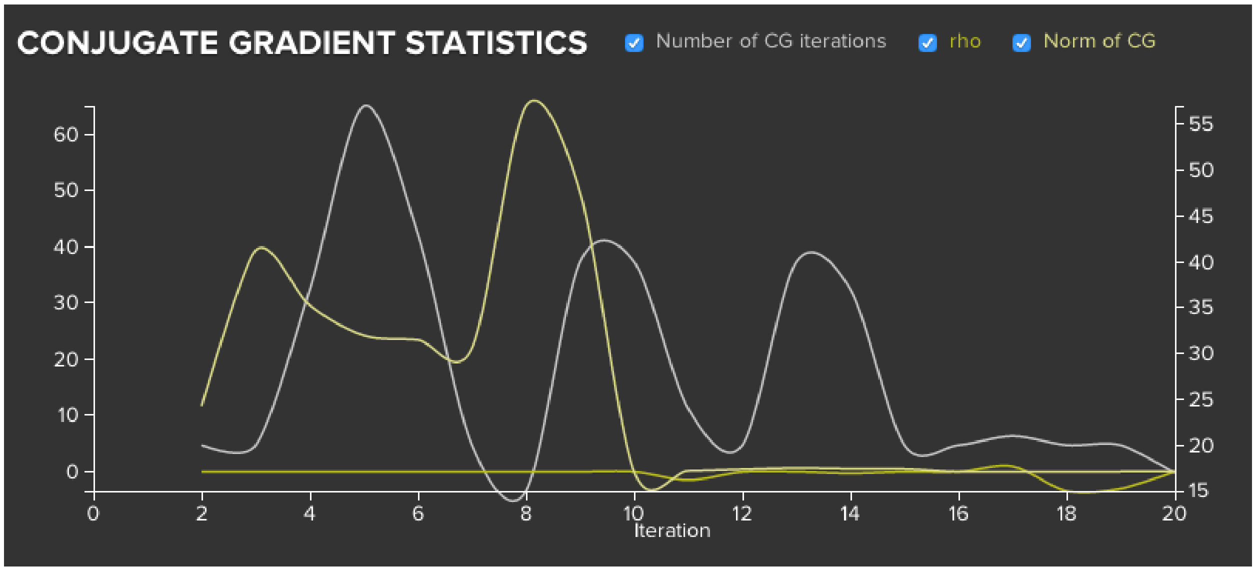
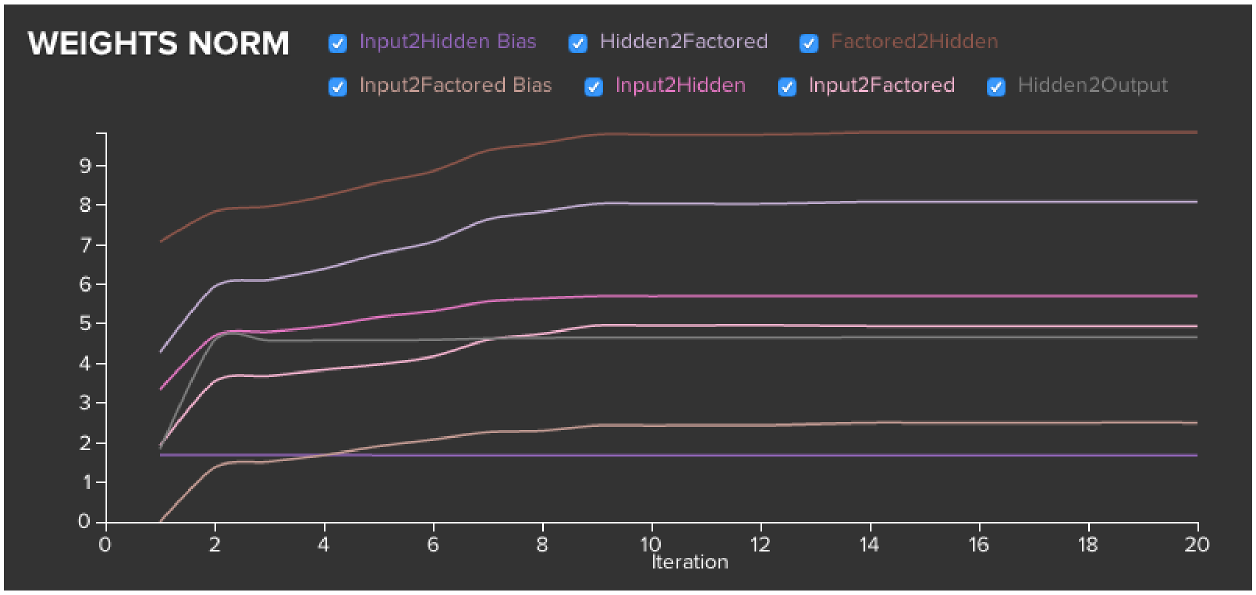
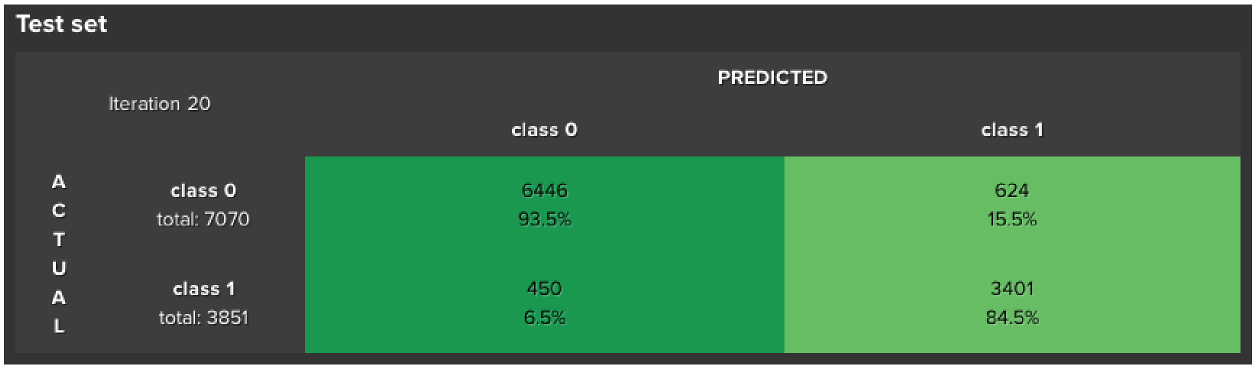


Figure 11: Change in normalized weights vs. iterations



Figures 9, 10, and 11 illustrate the change in hyperparameters of the network over time. As the iterations increased, these values became closer to their optimal values given the amount of data provided for training. After 20 iterations, anymore training would not be likely to increase the accuracy much more. Instead, the network could actually be overtrained and lose accuracy.

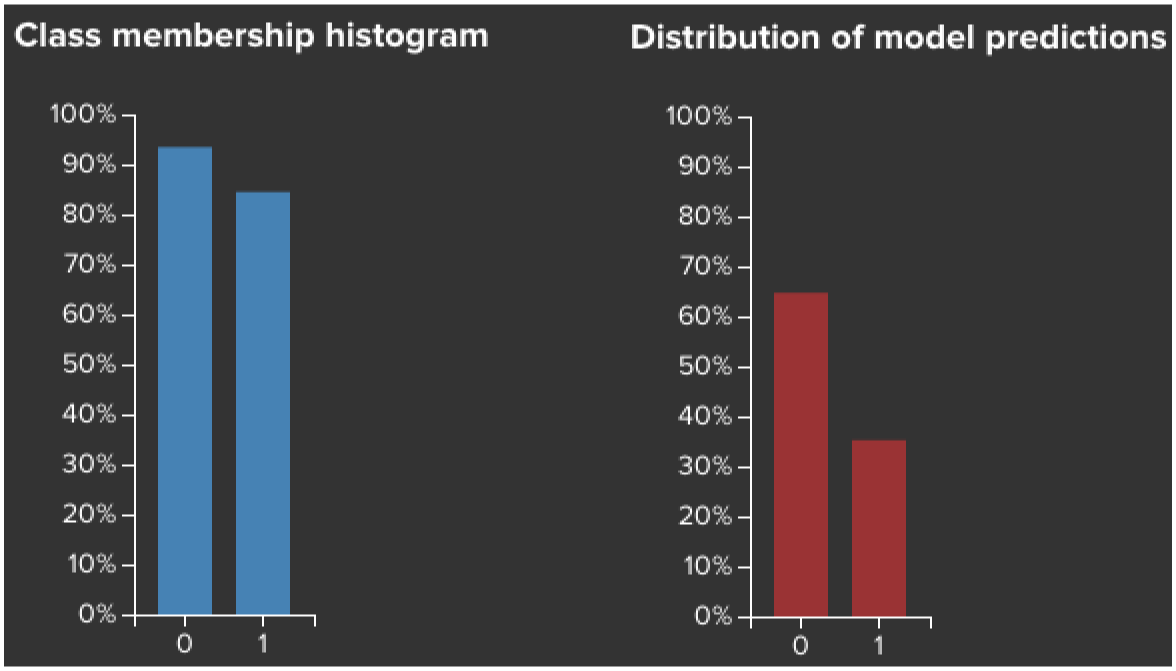
Figure 12: Confusion matrix of MRNN after 20 iterations



Test set

Iteration 20		PREDICTED class 0	PREDICTED class 1
ACTUAL	class 0 total: 7070	6446 93.5%	624 15.5%
	class 1 total: 3851	450 6.5%	3401 84.5%

Figure 13: Predictions of test set (‘1’ represents bicep curl, ‘0’ represents anything else)



Figures 12 and 13 show the performance of the network with regards to each class. The MRNN is more accurate at discerning if an exercise is not performed than if one is. If an exercise isn’t performed (Class ‘0’) it will know approximately 94% of the time. If an exercise is performed, it will know approximately 85% of the time. This accuracy can further be increase in the future by training the MRNN over a much larger dataset, as well as increasing the number of exercises performed. As more exercises are added, there will be more outputs with which the network can classify a movement pattern.

Conclusion

The Smart Weights project proved to be a complex and engaging process. There were hiccups initially when attempting to receive data from the Polulu IMU on the Raspberry Pi. One of the first big hurdles involved an issue with getting the stdout of the minimu9-ahrs program to feed into the python script correctly. They minimu9-ahrs program was providing timesteps at a rate of 50Hz, and it initially appeared to be too fast for the python script to handle. All the preliminary solutions tested caused buffer overflows that limited the data received and caused much of a motion’s data to be lost. After overcoming these issues, the main part of the project was developing a method to analyze the type of exercises were being performed. In order to create datasets that could be used to train the MRNN, a python script was created that recognized and recorded direction changes. This helped the manual process of preparing the datasets for training by indicating the start and stop of exercises performed. This data was then trained with the help of Ersatz Labs to create an algorithm that could automatically identify various exercises. The results were promising, showing that the MRNN was indeed a good algorithm for discerning time-series patterns. Classification accuracies of over 90% were achieved and adding more exercises in the future, along with a larger dataset, will undoubtedly offer an even higher level of precision.

Future Goals

Dynamic Time Warping

Dynamic time warping is a method used to identify patterns that are done at different rates. This is frequently used in machine learning codes and is applicable to our project as the exercises are performed at different speeds. For a complete project, a method such as dynamic time warping will be implemented in order to more accurately identify exercises that are performed at different rates. Python has a machine learning library that includes a dynamic time warping function in it: the .mlpy library.

Remove Orientation Dependancy

The Smart Weight prototype currently needs to be oriented correctly when performing a workout in order for it to classify exercises correctly. Removing this dependence on orientation is extremely important, and will make the device much more accessible to wide range of exercises and movements.

<u>More Exercises</u>

As a prototype, the Smart Weight was only trained with one exercise, the bicep curl. In order for this to be a viable product that will be useful to users, a wide variety of exercises will need to be implemented. This will involve working with fitness professionals to get large datasets for a wide variety of exercises.

<u>Form Correction</u>

A useful feature of the Smart Weight would be the ability to notify user's if they are not performing the exercise properly. This can be displayed either on a user's smartphone or announced through various personal assistants on newer smartphones (Apple's Siri, or Google Now).

<u>Website and iPhone app</u>

In order for the project to be more user friendly a website and iPhone application will be developed. The Smart Weight will be able to connect to a user's smartphone and sync the results of a workout. This data can then be uploaded to a website through an open api, allowing this data to be accessed from any device with internet access. More in-depth analysis of user workouts can also be performed server-side.

<u>Finalizing Design and Packaging</u>

The Smart Weight will need a custom PCB that integrates a power supply, a microprocessor, a wireless chip, and the inertial measurement unit. This finalized design should be power efficient (enough to last one day) and small enough to be convenient for users.

<u>Manufacturing</u>

With the finalized design, a manufacturer must be identified in order to bring the Smart Weight to market. Adjustments to the design may need to be made in order to reduce the cost of the device.

Bibliography

[1] Ahlgren, W. L. (2014, May). Associate Professor. (A. f. Rafla-Yuan, Interviewer)

[2] Atlas Wearables. (2014). *Measured Succes. Track Your Workout. Train Your Body*. Retrieved from Atlas: http://atlaswearables.com/

[3] Corder, S. (n.d.). *Neural Networks in Plain English*. Retrieved from AI Junkie: http://www.ai-junkie.com/ann/evolved/nnt1.html

[4] Coulston, F. &. (2007). *Design for Electrical and Computer Engineers.* McGraw-Hill.

[5] Grayson, D. (2014). *minimu-9 ahrs*. Retrieved from GitHub: https://github.com/DavidEGrayson/minimu9-ahrs

[6] J.R. Llata, E.G. Sarabia, J.P. Oria, (2001) "Pattern recognition with ultrasonic sensors: a neural networks evaluation", Sensor Review, Vol. 21 Iss: 1, pp.45 - 51

[7] JDS. (2013, September 10). *Algorithm advice for motion tracking with accelerometer*. Retrieved from Stackoverflow: http://stackoverflow.com/questions/18434934/algorithm-advice-for-motion-tracking-with-accelerometer

[8] Moov. (2014). *Meet Your Personal Coach*. Retrieved from Moov: http://preorder.moov.cc/

[9] Rong Zhu; Zhaoying Zhou, "A real-time articulated human motion tracking using tri-axis inertial/magnetic sensors package," *Neural Systems and Rehabilitation Engineering, IEEE Transactions on* , vol.12, no.2, pp.295,302, June 2004

[10] Sam Naghshineh, G. A. (2003). *Human Motion Capture using Tri-Axial accelerometers.*

[11] Sullivan, D. (2014). CEO of Ersatz Labs. (A. Fox, Interviewer)

[12] Sutskever, I. (2011). *Generating Text with Recurrent Neural Networks.*

Appendices

Analysis of Senior Project Design
Smart Weights

Luke Rafla-Yuan
Austin Fox

Advised by Tina Smilkstein

• 1. Summary of Functional Requirements

i) The Smart Weights project uses gyroscope and magnetometer data to track workout patterns. A Raspberry Pi microcontroller is hooked up to a Polulu Minimu-9 and communicates via i2c. After collecting data with the Raspberry Pi, the exercises are analyzed by a neural network provided by Ersatz Labs. The resulting algorithm is placed in a program stored on the Raspberry Pi in order to identify future exercises.

• 2. Primary Constraints

i) Efficient neural network that can effectively recognize work out patterns. This presented difficulties because of the complex pattern analysis required to measure specific 3D motion.
ii) Clearly defined data and analysis to be used for future development. In order to clearly present the project and set it up for the next stage in development, there was code written to collect, label, store, and graph the exercise data.
iii) Stand-alone demo platform used for presenting. The Raspberry Pi and IMU were attached to the weight in a specific orientation, and an SSH network was set to remotely program and collect data from the Raspberry Pi.

• 3. Economic

i) <u>Human Capital:</u> The Smart Weights project took 165 man hours to develop. The project is intended aid workout routines, and expected to expedite workouts while making them easier. This will allow user's to be more efficient during workouts and get better results with less effort and time.

ii) Financial Capital: The financial capital necessary for project development was used for the prototype system including the Raspberry Pi and the Polulu MinIMU-9 sensor.
iii) Manufactured Capital: The project had a monetary development cost of $1760.00. Because of the low cost of development materials, most of the cost is accredited to labor costs.
iv) Natural Capital: Smart Weights will require a small amount of natural capital in order to develop the circuitry and housing. After being manufactured, the product would have no further effect on natural capital.

• 4. If manufactured on a commercial basis:
i) The predicted number of devices to be sold per year will initially be 2,000. With a predicted manufacturing cost of $50.00 dollars per device and an estimated selling price of $70.00, a net profit of $20.00 is calculated per device sold, netting a total of $40,000.00 per year. With a cost of $70.00 and an expected lifetime of two years, the device will cost the user $35.00 per year.

• 5. Environmental
i) Because of the diminutive size of the Smart Weight it will use minutia natural resources. The project will require a small amount of non-renewable resources because of the plastics and silicon used. The weight is only used for the prototyping so it is not included in the environmental impact. There is no predicted environmental impact of the product after it is manufactured.

• 6. Manufacturability
i) The project was done with a prototyping platform consisting of a Raspberry Pi microcontroller, Polulu IMU, and a demo five pound hand weight. To convert the project from the prototyping to manufacturing stage, the Raspberry Pi microcontroller will be replaced with a much smaller and cheaper CPU. Also the platform would be placed in a small, custom manufactured package.

• 7. Sustainability
i) The Smart Weight will be designed to last for as long as it is taken care of and maintained in a reasonable fashion. However, it will not be able to improve upon the sustainability of classical weights because of the additional non-renewable resources required to make it.
ii) In order to make the product more sustainable, research must be done to use the sustainable components during manufacturing. Replacing the microcontroller

with a pre-programmed CPU would get rid of the many unneeded peripherals on the Raspberry Pi. Also, choosing and a sustainably constructed IMU and housing for the final product would be beneficial.
iii) In order to use a CPU in the product, all the testing and development would need to be finished before ordering the part. Since the project only covers a prototype system, using a CPU wouldn't be feasible. Similarly, choosing sustainable housing and IMU would be done at a later stage in development.

• 8. Ethical
i) The Smart Weight project is designed to be used to improve the user's health. In and of itself, this is an ethical goal based on utilitarian standards. However, the designers, must make sure to not overstate or make false claims when promoting the application of Smart Weights.
ii) Applying the IEEE code of ethics requires the projections made from the data to be realistic and as accurate as possible. This constraint was applied when predicting the future use and possibilities of the project.

• 9. Health and Safety
i) The Smart Weights project has no foreseeable health dangers associated with it, as long as it is used correctly. Before manufacturing the device, the exercises would be verified by health professionals and certified coaches to ensure they are performed correctly.
Smart Weights will be used to promote a healthy lifestyle. One of the major goals of the project is to allow users to exercise more efficiently. Smart Weights will push individuals to work out consistently, and to their peak potential. Therefore, Smart Weights is predicted to have a positive impact the health of its users.

• 10. Social and Political
i) The Smart Weights project has social implications because it can theoretically be used to improve the health lifestyle of a community. As the project grows and other similar projects are developed, social media will be a huge benefactor for fitness training.
There are also political implications involved in the possible repercussions of a faulty device. If the device is not calibrated correctly and guides users through incorrect exercises, there is a possibility of incurring injury. There will be proactive steps taken against this possibility by thoroughly testing the device with health professionals.

- 11. Development

  i) Over the course of the project, the developers learned about creating a neural network and the associated difficulties. The project team also dealt with the difficulties of data processing. Significant effort went into trying to zero out gravitational forces using Euler angles and initialized data. Using the current orientation of the device at each timestep, a change of basis matrix was used to convert the 1g of gravity (normally all in the z direction) into the corresponding local x, y, and z acceleration values. These values were then subtracted off of the acceleration values for each timestep in order to get the acceleration of the device due user movement rather than movement and gravity combined. Ultimately, this idea was not implemented in the final design because it dramatically increases the number of calculations performed for each timestep and the developers realized this will have no increase in accuracy for MRNN. The ultimate goal was to prevent the orientation of the device from affecting the output of the network, but no reliable method of doing so was achieved. There also was a learning curve involved with integrating the hardware. The Pololu needed to be connected to the Raspberry Pi and code needed to be implemented to set up the i2c protocol. Also, an SSH network needed to be created so that the Raspberry Pi could be remotely programmed and the data could be easily accessed. There was significant effort put into setting up the i2c communication between the Polulu IMU and the Raspberry Pi, as well as setting up the SSH network for the Raspberry Pi.

Bill of Materials

-Raspberry Pi: $40 (http://www.element14.com/community/community/raspberry-pi)
-Polulu Minimui-9 v2: $39.95 (http://www.pololu.com/product/1268)
-Hand Weight: $0.00 (Provided)
-Total Material Cost: $79.95

Code

```
import pexpect
import xlwt
import timeit
cmd = 'minimu9-ahrs -b /dev/i2c-1 --mode=raw'
rawData = pexpect.spawn(cmd)
i=1
start=0

#Open xls worksheet and name rows
book = xlwt.Workbook()
sheet1 = book.add_sheet("Exercises")

sheet1.write(0, 0, "X")
sheet1.write(0, 1, "Y")
sheet1.write(0, 2, "Z")
sheet1.write(0, 3, "Time")
sheet1.write(0, 4, "Delta Time")

#Initialize timer
start_time = timeit.default_timer()
time2 = start_time

x_counter_0=0
x_counter_1=0
y_counter_0=0
y_counter_1=0
z_counter_0=0
z_counter_1=0

prev_x_direction=0
prev_y_direction=0
prev_z_direction=0

x_switch=0
y_switch=0
z_switch=0
```

```
for line in rawData:

    #Get XYZ acceleration values and convert them into floating point numbers
    values = line.split()
    accel = "x=" + values[3] + " y=" + values[4] + " z=" + values[5]
    x = float(values[3])
    y = float(values[4])
    z = float(values[5])



    if i==1:
        initial_x = float(values[3])
        initial_y = float(values[4])
        initial_z = float(values[5])
        prev_x=x
        prev_y=y
        prev_z=z

    print accel

    if x > prev_x:
        x_counter_1 = x_counter_1 + 1
        if x_counter_1 > 5:
            x_max = x
            x_counter_0 = 0
            if x_switch == 1:
                sheet1.write(i-5, 5, "-500")
                x_switch = 0
    else:
        x_counter_0 = x_counter_0 + 1
        if x_counter_0 > 8:
            x_min = x
            x_counter_1 = 0
            if x_switch == 0:
                sheet1.write(i-8, 5, "500")
                x_switch = 1

    if y > prev_y:
        y_counter_1 = y_counter_1 + 1
        if y_counter_1 > 5:
            y_max = y
            y_counter_0 = 0
            if y_switch == 1:
                sheet1.write(i-5, 6, "-500")
                y_switch = 0
    else:
        y_counter_0 = y_counter_0 + 1
        if y_counter_0 > 8:
            y_min = y
            y_counter_1 = 0
            if y_switch == 0:
                sheet1.write(i-8, 6, "500")
                y_switch = 1

if z > prev_z:
        z_counter_1 = z_counter_1 + 1
        if z_counter_1 > 5:
            z_max = z
            z_counter_0 = 0
            if z_switch == 1:
                sheet1.write(i-5, 7, "-500")
                z_switch = 0
else:
        z_counter_0 = z_counter_0 + 1
        if z_counter_0 > 8:
            z_min = z
            z_counter_1 = 0
            if z_switch == 0:
                sheet1.write(i-8, 7, "500")
                z_switch = 1


prev_x=x
prev_y=y
prev_z=z

if (abs(initial_x-x)>20) and (abs(initial_y-y)>20) and (abs(initial_z-z)>20) and start<1:
        # sheet1.write(i, 0, "Approx_start")
        # sheet1.write(i, 1, "Approx_start")
        # sheet1.write(i, 2, "Approx_start")
        # sheet1.write(i, 3, "Approx_start")
        # sheet1.write(i, 4, "Approx_start")
        sheet1.write(i, 5, "1")

        i=i+1
        start=2

#Calculate time and delta time values
time1 = timeit.default_timer()
delta_time = time1-time2
time2 = timeit.default_timer()
print ("i = %f" % (i))
print ("time = %f" % (time1-start_time))
```

```
        #Write values to xls sheet and increment row number (i)
        sheet1.write(i, 0, x)
        sheet1.write(i, 1, y)
        sheet1.write(i, 2, z)
        sheet1.write(i, 3, time1-start_time)
        sheet1.write(i, 4, delta_time)

        i=i+1

        #After i data points, save xls sheet and close program
        if i>1000:
                book.save("Random_Data.xls")
                exit()


rawData.close()
```