CS 74 Final Project Report: Predicting Stress from Android Sensor Data

Nishant Kumar, Minsoo Kim

March 15, 2015

I. Project Goal

Stress is an indicator for a wide variety of both physical and mental health disorders. Although stress is relatively easily self assessed and addressed, it often goes unnoticed as a natural part of everyday life. Unaddressed in this way, especially over a prolonged period of time, stress may lead to more serious mental health disorders such as depression or increase the likelihood of physical disorders such as heart attack.

An automatic stress level prediction system that can infer a user's level of stress from sensory input will be helpful in addressing the issues caused by stress. Because sensory data is gathered continuously, such a system can monitor a user's stress level with little interruption and alert/remind the user of his or her current stress level or make recommendations based on it.

We propose an application that can predict a user's stress levels by analyzing the raw sensor data from his/her android device. This application then in theory can sit on a centralized web server or on the user's Android device itself.

II.Scope

The current scope of the project is to demonstrate that a machine learning algorithm provided enough raw sensor data from a smartphone can learn a model to predict a user's stress levels.

III. Data

The data used to learn this model is from the Dartmouth StudentLife study. This is available on the CS department's website. This data was collected from student volunteers from the CS-65 Smartphone programming class offered last year. Student volunteers were given android phones which had an in built app which recorded raw sensor data such as:

Audio inputs
Activity inferences
Conversation logs
GPS location
Bluetooth
GPS location
WiFi
WiFi location,
Light sensor

Phone lock
Phone Charge
Education data - Deadlines etc.

Data collection monitoring
Students were sent reminder emails if there were any gaps in the data collection or the data was not getting uploaded at all.

Incentive
The student volunteers were offered incentives such as free T-Shirts, Google Nexus smartphones etc.

Privacy
The student volunteers' identities were hidden by using random ids. The call logs and sms logs were hashed.

Survey Data
The dataset also contains self reported stress level data ranging between 1 and 5. The original mapping used in the dataset is as follows:

Stress Level Id	**Description**
1	A little stressed
2	Definitely stressed
3	Stressed out
4	Feeling good
5	Feeling great

We have used a new mapping in order to show a monotonically increasing values of stress. Its shows as follows:

Stress Level Id	**Description**
1	Feeling great
2	Feeling good

3	A little stressed
4	Definitely stressed
5	Stressed out

The data collected is stored in csv and JSON formats.

IV.Data Processing

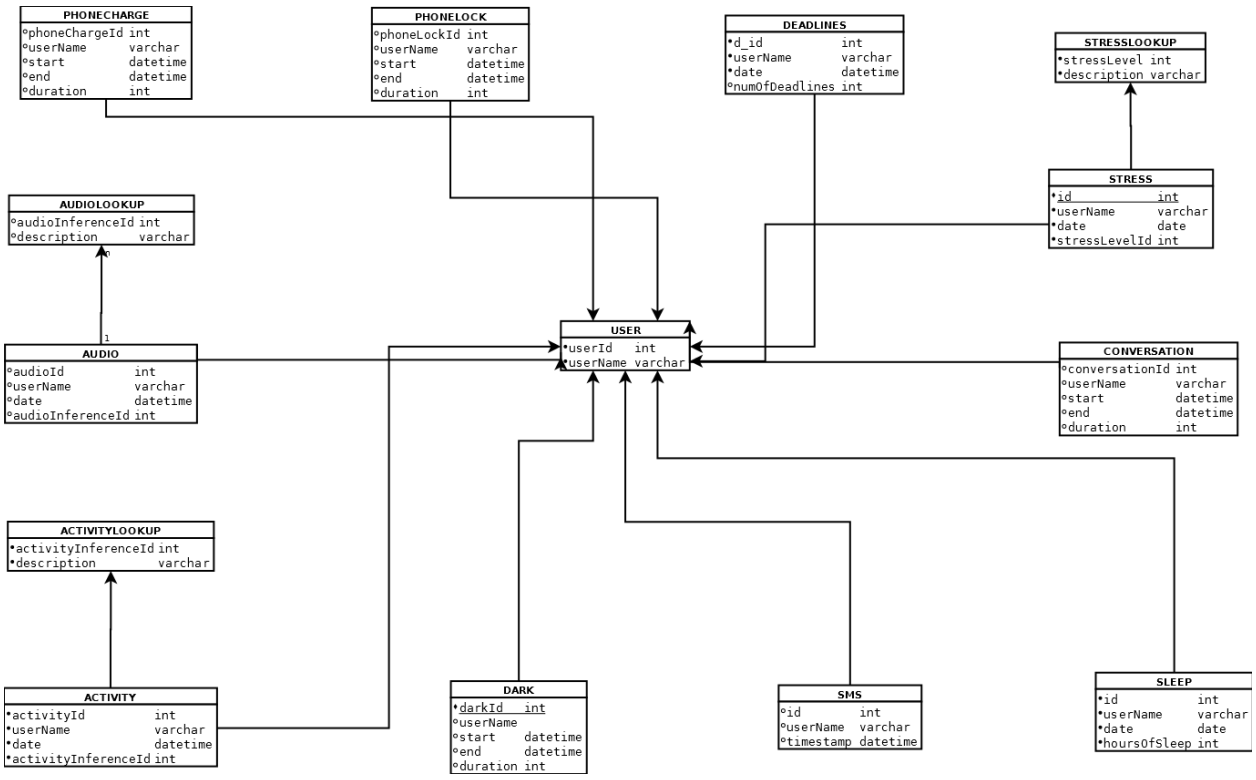
The project requires certain amount of Software Engineering effort. Its described as follows:

a. We have written several parsers in Java to read the data that is in csv and JSON formats.

b. The respective parsers then further pass on the data to data adaptors that store all this sensor data in a database. The data adaptors are implemented in Java and Hibernate which provide the Data Access Object(DAO) layer over the database.

c. The database is implemented in MySQL database server.

The database schema diagram is as follows:



Every individual sensor data is stored in its own table. The tables further should have a foreign key relationship with the USER table, indicating that they cannot store data about any user that is not present in the USER table.

This database is then queried to generate input for the Neural Network to train on. For the milestone, the sensor data we used for the features is as follows:

Conversation logs
Light data
Phone lock
Phone Charge
Sleep

Stress data acts as our y - label.

If the sensor data is recorded multiple times in a given day for a given user, we add up all the data. Eg: the conversation data for a given day and given user is added up to indicate the number of hours a user has conversation logs on that day.

The stress data on the other hand for a given day and user is averaged up.

Feature expansion

In order to generate discriminative feature vectors, we expanded our original set of 5 features.

We expanded Conversation based on histogram by total calls made under certain call length. Eg how many calls were made by a given user on a given day that were of the length 5 mins. The max call length by any user went all the way till 412 mins. So this gives us about 412 expanded features from Conversation
.
We expanded PhoneLock using a similar criteria i.e how many time was the phone locked for a certain length. Eg how many time was the phone locked by a given user on a given day for 5 mins. The max time a phone was kept locked was about 600 mins. So this gives us about 600 expanded features from Phone Lock.

Again, a similar criteria performed on Phone Charge gave us about 600 expanded features.

After this feature expansion exercise and including the already present features, we get a total of 1622 features. Each feature containing 3872 rows. Although number of rows were further filtered described in results section below.

V. Neural network

1. Motivation

Models of regression and classification that involve linear combinations of fixed basis functions are useful analytical tools, but they often run into trouble when the model we desire to approximate has high dimensionality. The relationship between human behavior and phenomena such as stress is most likely high dimensional and nonlinear. Therefore we use neural networks to approximate a model of the relationship between behavior and stress.

2. Training

i) Gradient descent

To train a neural network, we wish to find a weight matrix w which minimizes the error function $E(\mathrm{w})$. Since we cannot find an analytical solution to $dE(\mathrm{w})=0$, we use gradient descent to optimize the weight matrix. We choose the weight update to be a small step in the direction of the negative gradient. The gradient descent update rule is given by:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} - \eta \nabla E(\mathrm{w}^{(\tau)})$$

where the parameter η is the learning rate. The above rule applies to the *batch gradient descent* method, which evaluates the gradient based on the entire training set before updating the weight matrices.

An online version of gradient descent, whose weight update rule is given by:

$$\mathrm{w}^{(\tau+1)} = \mathrm{w}^{(\tau)} - \eta \nabla \mathrm{E}_n(\mathrm{w}^{(\tau)})$$

is known as stochastic gradient descent, and updates the weight matrix based on each training example. Stochastic gradient descent simplifies the gradient by computing one for each training example, but has been demonstrated to perform on par with batch methods, while taking less running time.

ii) Backpropagation

In order to evaluate the gradient of $E(\mathrm{w})$ for a feed-forward network, we use error backpropagation. The backpropagation algorithm is derived by applying the chain rule for partial derivatives and obtaining the derivatives of the error function with respect to the the neural network weights.

We omit the full derivation of the backpropagation algorithm, which was referenced from Section 5.3.1 of Bishop.

3. **Algorithm**

Based on the algorithm provided by Bishop, we build a neural network with the following functions.

First, the activation function for the hidden units are given by

$$\mathrm{h(a)} = \tanh(\mathrm{a})$$

where

$$\tanh(\mathrm{a}) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

The hyperbolic tangent function has its derivative given by

$$\mathrm{h'(a)} = 1 - \mathrm{h(a)}^2$$

We use a euclidean error function given by

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

where y_k is the activation output unit k and t_k is the corresponding target value.

The forward propagation is performed by:

$$a_j = \sum_{i=0}^{D} w_{ji}{}^{(1)} x_i$$

$$z_j = \tanh(a_j)$$

$$y_k = \sum_{j=0}^{M} w_{kj}{}^{(2)} z_j$$

Finally, to obtain the gradients, we first compute

$$\delta_k = \mathrm{y}_k - \mathrm{t}_k$$

and

$$\delta_j = (1 - z_j{}^2) \sum_{k=1}^{K} w_{kj}\, \delta_k$$

and compute

$$\frac{\partial E_n}{\partial w_{ji}{}^{(1)}} = \delta_j x_i \qquad \frac{\partial E_n}{\partial w_{kj}{}^{(2)}} = \delta_k z_j$$

Finally, we added regularization to the neural network.

VI. Implementation details & Challenges

a. External software: We did not not use any external software other than STATA to help with the data collection and merging processes. The preprocessing of data was done via MySQL, and Java, and STATA, and the neural network was implemented in MATLAB.

b. Training and testing code: The training and testing codes are implemented as MATLAB functions.

Neural network implementation

1. **Random initialization**

We encountered a symmetry problem in our initial implementation of the neural network. Initializing the parameters to random positive values between 0 and 1, while an intuitive choice, causes a problem known as symmetry, in that all activation values of the hidden layer take on similar values. In this case, the neural network will be unable to learn. Since we used the hyperbolic tangent function, an initialization of all parameters to positive values has a high likelihood of causing all of the hidden unit layer activations to take on the value of 1. This means that the error signals propagated to these units will take the same value, which means that the weight updates for the hidden units will be identical. This causes the neural network to "get stuck".

We resolved the issue by initializing the parameters to random values within a range of -0.12 to 0.12. The specific values were reached via trial and error of best performance, and recommendations from literature.

2. Stochastic Gradient Descent

We found that performing Stochastic Gradient descent with one training example at a time resulted in poor performance. Therefore we implemented minibatch and were able to speed up our training time, but we ran into issues with bad local minima, so switched back to the batch implementation.

Data processing implementation

The Java files are described as below:
The Java files contained in the package edu.cs174.studentlifedataadaptor are the files that parses the raw data from their individual csv / json files and stores them the database.

The Java files contained in the package edu.cs174.studentlifedataadaptor.dao are the files that pull the required data out of the database and perform the processing needed on the data. Eg as follows:

a. query the database and get the list of dates for which we have self reported stress levels data
b. for these dates query the database and get the Conversation logs. Extract the duration in minutes or hours as per requirement. If there are multiple logs for a given user and on a given day, then they can all be added up here

The Java files contained in package edu.cs174.studentlifedataadaptor.featureexpansion does the expansion of the features as described in above sections.

All these Java files use third party jar files / libraries as follows:

a. ant
b. ant-launcher
c. antlr-2.7.6
d. asm
e. cglib-2.1.3
f. commons-collections-2.1.1
g. commons-lang-2.4
h. commons-logging-1.0.4
i. dom4j-1.6.1
j. hibernate3
k. jms
l. joda-time-2.7
m. json-simple-1.1.1
n. jta
o. log4j-1.2.17
p. mysql-connector-java-3.1.14
q. opencsv-3.1

VII. Results

For our final results, we used a data set of 1270 total examples, produced by removing all training examples which did not have the full set of feature data, meaning there was at least one sensor data (out of the five features, phone-lock, phone-charge, sleep, dark, conversation) from the user that was unable to be collected for the corresponding time period. Afterwards, we expanded these existing

features by adding minutely histogram features of phone-lock, phone-charge, and conversation, adding ~600, ~600, and ~400 features, respectively for a total of 1622 features.

1. **Test of Objective using user separation**

Firstly, we needed to address the issue arising from the fact that our dataset did not differentiate between the 60 different users from whom the data were collected. This meant that if it was the case that certain users always produced the same input-stress pairs, then our test set could potentially be a partial replicate of our training set, because the same user will likely appear in both the training and test sets. To test for this possibility, we ran two separate validation runs, first taking the first 10 users by userID as the test set and the other 50 users as the training set, and in the second case taking the last 10 users by userID as the test set and the rest of the users as the training set. Figures A and B show the results:

Figure A: First 10 users as test set

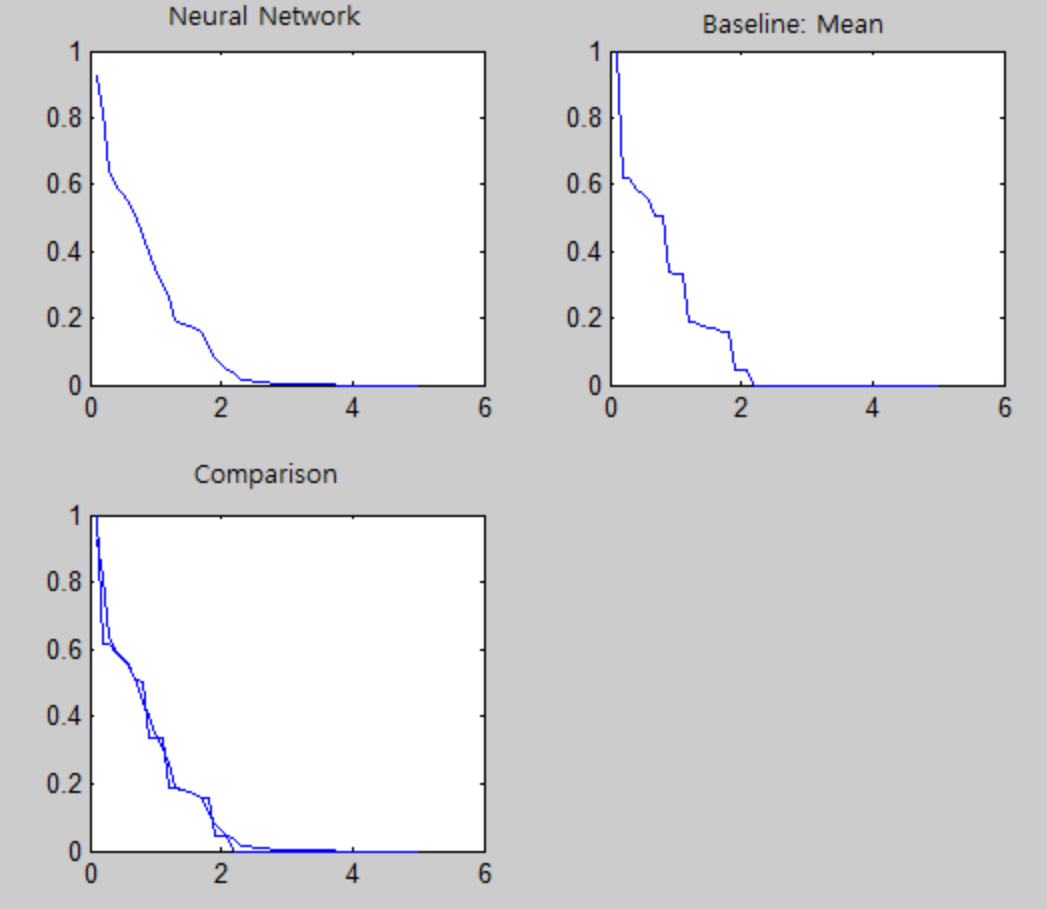
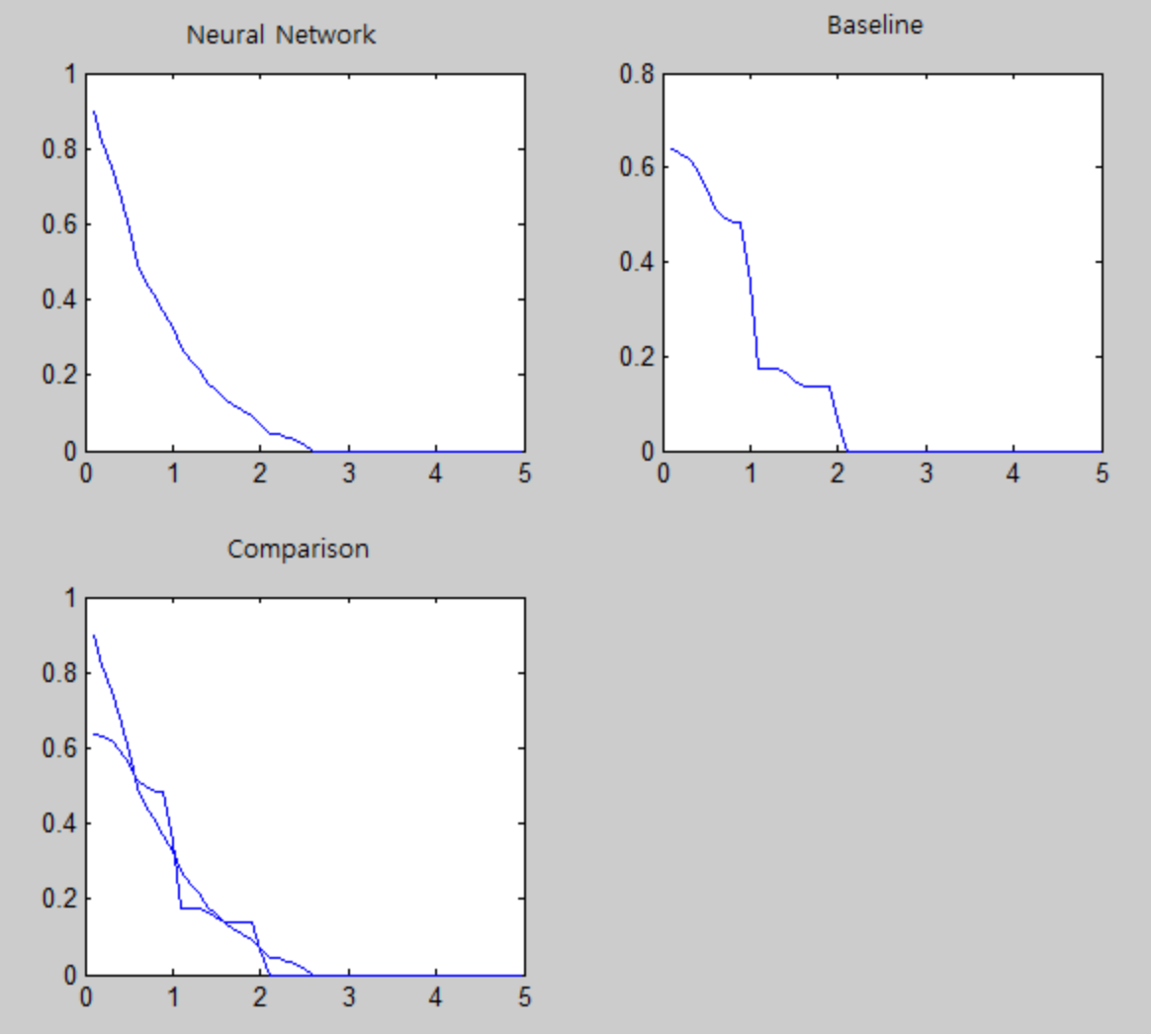


Figure B: Last 10 users as test set



Because both validation results returned errors roughly equal to 1 for a training error of 1, and they performed close to the baseline, we decided to go ahead with the mixing the datapoints of different users in our main dataset.

2. Optimal Hidden Units, Learning Rates, and Regularization

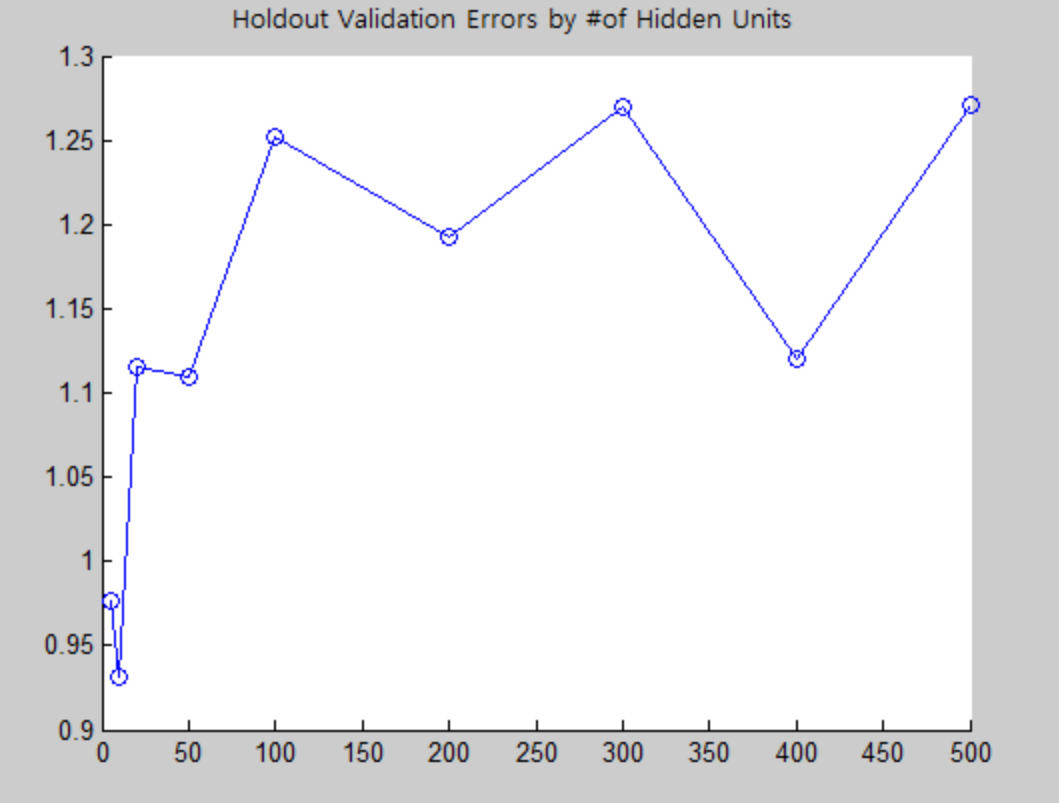
i) Learning rate

The process of finding the optimal hyperparameters was carried out starting with the learning rate. We found that the learning rate was better manually picked because variations in the number of hidden units caused different bad local optima or training failure (values going to infinity). Therefore we used the learning rates 0.1~0.01 keeping in mind consistency for comparisons. However in general, most training runs shown in the following results were conducted with learning rate=0.01 except when the learning rate of 0.1 was found to be feasible.

ii) Hidden units

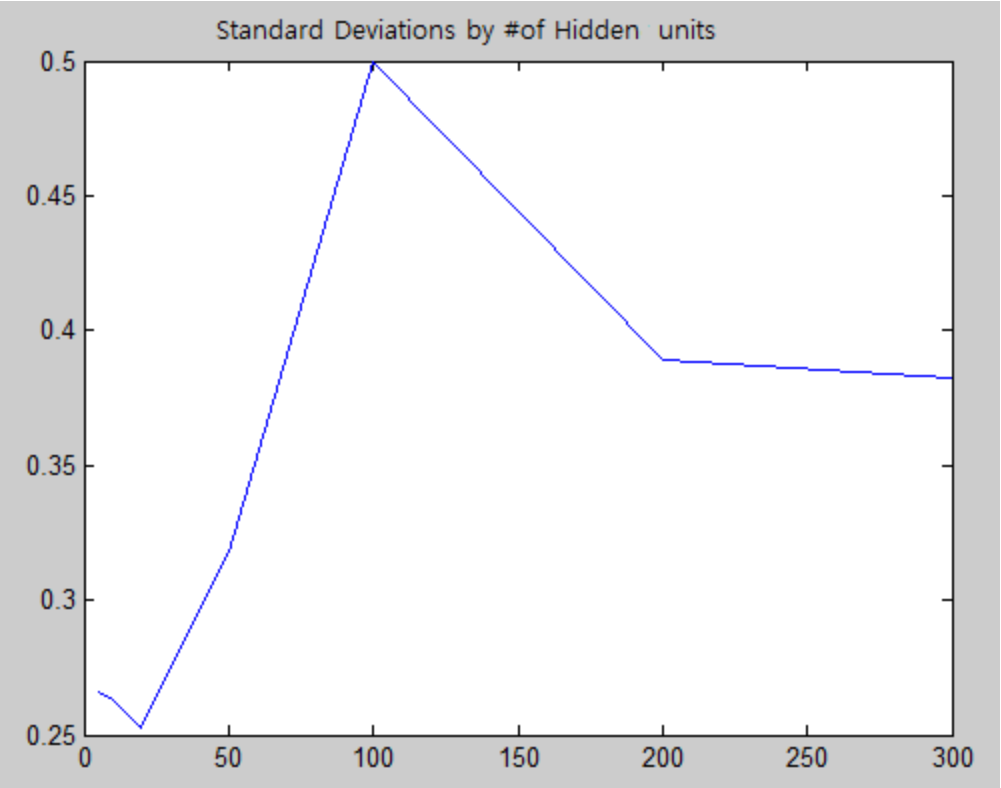
In order to find the optimal number of hidden units, we looked at the holdout validation error and the standard deviation results returned by our training & tests. Figure C shows the plot of holdout validation errors for different numbers of hidden units.

Figure C:



We found that the validation error generally increased for increasing numbers of hidden units. However, before deciding on 5, for example, as the number of hidden units based on this result, we looked at another criteria, standard deviation. Figure D shows the plot of standard deviation for different numbers of hidden units.

Figure D:



Noticeably, we observed that for numbers of hidden units less than 50, the standard deviation was very low, at 0.3 or lower. Looking at the predicted values for the test examples, we confirmed that indeed that predicted values were mostly very close to the mean. However this was not desirable because the variance of the Y column is close to 1, and a low standard deviation simply means that the model is minimizing the MSE by finding a mean value. Therefore we began to look at standard deviation as a criteria for our test performance, postponing the decision on the optimal number of hidden units.

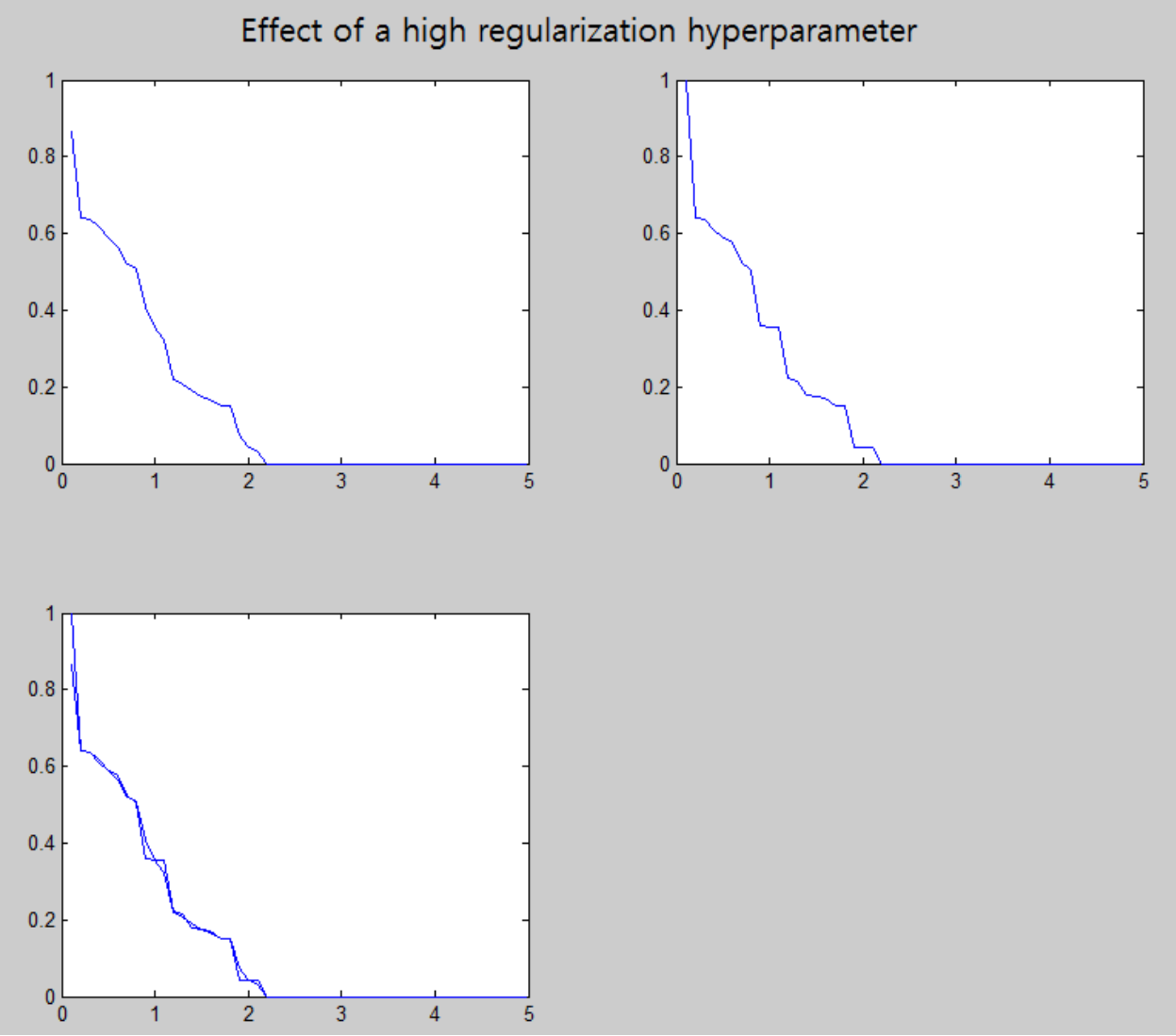
We saw that lower number of units fits the base well because the variance and std is lower meaning it predicts mostly the mean.

iii) To explore the implications of standard deviation values further, we tested whether they had any correlation with the lambda regularization hyperparameter. The following tables shows a comparison of lambda=0 vs lambda=1000 on the standard deviation.

Hidden Units	Training error	Lambda	Standard deviation
5	**1**	**0**	**0.0372**
5	**1**	**1000**	**0.2657**

We found that by forcing the lambda to a high value, we could drive down the standard deviation and also bring our neural networks predictions very close to the mean of the Y's, our baseline. Figure E shows the effect of lambda=1000 on the comparison with the baseline.

Figure E:



Based on this result, we were able to conclude that underfitting causes the model to predict the mean value of the Y's.

Finally, we tested the validation errors and standard deviations that result from training to a lower error threshold (0.7 as opposed to 1 before). The table shows our results:

Hidden units	**Training error**	**Standard Deviation**	**Test error**
5	**0.7**	**0.4218**	**0.8594**
100	**0.7**	**0.4799**	**1.0836**
200	**0.7**	**0.4158**	**1.2154**

We found that the number of hidden units ceased to have a significant effect on the standard deviation. The standard deviation also increased to over 0.4 for all hidden units, as opposed to before when we had values 0.2~0.3 for some numbers of hidden units. However we did see that choosing 100 hidden units gave a slightly higher standard deviation.

Our final figure shows the results of training to a training error of 0.5. The validation error was 1.2085, and the standard deviation was 0.4557.

Figure F : Trained to 0.5 training error, 100 hidden units.

VIII. Conclusions & Discussion

From Figure F, we can see that the neural network did not outperform a baseline of taking the mean of the target variables as the predicted value for all examples. However we were able to find a different fit for the data from the baseline, with a relatively similar MSE near 1, and standard deviation of ~0.4 as opposed to 0 given by the baseline.

However, the fact that we were not able to outperform the mean baseline, coupled with the fact that we found that imposing high regularization on the neural network caused it to underfit and match the baseline, led us to conclude that the dataset as constructed by us had too high variance and was not enough to capture the model we had hoped for. Some obvious issues were feature expansion, and dealing with missing sensor values, resulting in a smaller dataset size.

Although our results indicate that it is difficult to predict stress data from the raw sensor data of sleep, conversation, light/dark and phone lock/charge, we hope to have an opportunity to perform a deeper analysis of this problem in the future with a more expansive set of features.

IX. Possible Future Work

1. Apply more feature expansion techniques to construct a richer dataset
2. Increase the size of the dataset
3. Apply more advanced neural network techniques
4. Put the application on a web-server

X. References:

1. Discussions with Professor Lorenzo Torresani.

2. *Wang, Rui, Fanglin Chen, Zhenyu Chen, Tianxing Li, Gabriella Harari, Stefanie Tignor, Xia Zhou, Dror Ben-Zeev, and Andrew T. Campbell. "StudentLife: Assessing Mental Health, Academic Performance and Behavioral Trends of College Students using Smartphones." In Proceedings of the ACM Conference on Ubiquitous Computing. 2014.*

3. *S. E. Taylor, W. T. Welch, H. S. Kim, and D. K. Sherman. Cultural differences in the impact of social support on psychological and biological stress responses. Psychological Science, 18(9):831–837, 2007*

4. *N. D. Lane, M. Mohammod, M. Lin, X. Yang, H. Lu, S. Ali, A. Doryab, E. Berke, T. Choudhury, and A. Campbell. Bewell: A smartphone application to monitor, model and promote wellbeing. In Proc. of PervasiveHealth , 2011*

5. *CS65 Smartphone Programming. http://www.cs.dartmouth.edu/~campbell/cs65/cs65.html*

6. *Depression. http://www.nimh.nih.gov/health/topics/depression/index.shtml*

7. *StudentLife Dataset 2014. http://studentlife.cs.dartmouth.edu/*

8. *C. M. Aldwin.Stress, coping, and development: An integrative perspective. Guilford Press, 2007*

9. *Eftekhar B, Mohammad K, Ardebili HE, Ghodsi M, Ketabchi E. Comparison of artificial neural network and logistic regression models for prediction of mortality in head trauma based on initial clinical data. BMC Medical Informatics and Decision Making 2005;5:3. doi:10.1186/1472-6947-5-3.*

10. *Bishop, Christopher M. Pattern Recognition and Machine Learning*

11. *Ng, Andrew. Machine Learning Course Materials (Stanford, Coursera)*

12. *http://www.mysql.com/*
13. *http://ant.apache.org/*
14. *http://asm.ow2.org/*
15. *https://github.com/cglib/cglib*
16. *http://commons.apache.org/proper/commons-collections/*
17. *http://commons.apache.org/proper/commons-lang/*
18. *http://hibernate.org/*
19. *http://www.mysql.com/*
20. *http://opencsv.sourceforge.net/*