Predicting Typing Speed with Smartwatch

Anuj Kumar
College of Information and Computer Sciences
University of Massachusetts Amherst
anujkumar@umass.edu

Abstract

With the growth in the popularity of smartwatches, there's a new avenue for smart devices to tap into to aid their users. As smartwatches are expected to do more, ventures have been made into their technical capabilities. This project aims at gauging one such aspect. With the proper neural network pipeline, I show that it is possible to predict the typing speed of a person wearing a smartwatch in real time. This can then be used to determine if a person is distracted or depressed and perform in-time intervention.

1. Introduction and Previous Work

Over recent years, there has been an increasing demand for smartwatches to assist better towards the daily lives of people. Recent works have seen a smartwatch tracking the trajectory of a user's arms [9], as well as controlling the various appliances in a smart home [8].

One of my recent projects on similar lines was ClInKs [6]. It was done as part of the Neural Networks CMPSCI 682 course in Fall 2017. In that project, I showed that with an end-to-end affine + recurrent neural network pipeline, it is possible to an extent to predict the characters being typed on a keyboard using the sensor data from smartwatches.

The outcome of that project, that typing is detectable by smartwatch sensors, is the basis towards this project. The aim of this project is to detect the typing speed of a user using only a smartwatch. Through initial experiments, it was determined that typing speed is the major indicator towards a person's mental presence. Thus, by detecting real time typing speed, I hope to be able to sample a person's mental presence and well-being.

My implementation is publicly accessible on my GitHub repository[1].

This report was submitted for the Independent Study course CMPSCI 696 at the University of Massachusetts Amherst. Spring 2018.

[1] https://github.com/anujkumar93/PredictTypeSpeed

2. Architecture and Dataset

For this project, the data was collected from two people. Each person wore two smartwatches, an ASUS 3ZenWatch 2 on their left wrist and a LG G Watch W100 on their right wrist. The accelerometer, gyroscope and gravity sensors were recorded from both watches using the modified version of the publicly available app Sensor Data Logger [5]. The app was modified to not only demand (from watch) and record data at the highest frequency for each sensor (50 Hz) but also to log all data into a neat csv upon exiting the app.

It was further modified to also store live data every 16 seconds (configurable) by modifying relevant files in parallel. The folder in which these files were stored was then synced live with my Dropbox account. This was achieved through the Instant Sync feature of the Android app FolderSync [1]. A folder on my laptop was then synced with the said Dropbox folder using Dropbox daemons for Ubuntu. This was done because the network architecture would eventually run on the laptop and live data was required.

The configurable value of 16 seconds was chosen empirically after observing the internet speed on both devices, and the time it took the files to sync between mobile and laptop. Also, due to the specific nature of the university (UMass) wifi network, it is not possible to set up a REST webservice to receive live data as the security settings prevents communication between two devices. This was also confirmed with commercial apps like SSHDroid [2].

I augment the collected time series data from both the smartwatches by also converting each axis of each sensor into the frequency space using the Fast Fourier Transform. Since the smartwatch readings aren't synchronized, we get different number of sampled points for the same duration (4 seconds in my case) for different sensors, or even for the same sensors at different times. This necessitates that I convert them to some uniform dimension. Hence, I calculate the statistical features mentioned in Table 1 for each axis of each sensor in the time space as well as the frequency space.

The resulting input has 612 features (size of dimension 1). I collect data for around 22178 characters, and club them into 4 second windows to form one input sample. I thus

Table 1: Statistical Features

1. Mean: $\mu_s = \frac{1}{T}\Sigma_{i=1}^{T} s_i$
2. Standard Deviation: $\sigma_s = \sqrt{\frac{1}{T}\Sigma_{i=1}^{T}(s_i - \mu_s)}$
3. Coefficients of variation: $c_v = \frac{\sigma_s}{\mu_s}$
4. Peak-to-peak amplitude: $max\{s1, ..., sT\}$ - $min\{s1, .., sT\}$.
5-9. Percentiles: 10^{th}, 25^{th}, 50^{th}, 75^{th}, 90^{th}
10. Interquartile range: difference between the 75^{th} and 25^{th} percentiles.
11. Lag-one-autocorrelation: $\frac{\Sigma_{i=1}^{T-1}(s_i-\mu_s)(s_{i+1}-\mu_s)}{\Sigma_{i=1}^{T}(s_i-\mu_s)^2}$
12. Skewness: $\frac{\frac{1}{T}\Sigma_{i=1}^{T}(s_i-\mu_s)^3}{(\frac{1}{T}\Sigma_{i=1}^{T}(s_i-\mu_s)^2)^{3/2}}$, measure of asymmetry of the signal probability distribution.
13. Kurtosis: $\frac{\frac{1}{T}\Sigma_{i=1}^{T}(s_i-\mu_s)^4}{(\frac{1}{T}\Sigma_{i=1}^{T}(s_i-\mu_s)^2)^3} - 3$, degree of the peakedness of the signal probability distribution.
14. Signal power: $\Sigma_{i=1}^{T} s_i^2$
15. Log-energy: $\Sigma_{i=1}^{T} \log s_i^2$
16. Zero crossings: number of times the signal crosses its median.
17. Correlation between each pair of axes: $\frac{\Sigma_{i=1}^{T}(s_i-\mu_s)(v_i-\mu_v)}{\sqrt{\Sigma_{i=1}^{T}(s_i-\mu_s)\Sigma_{j=1}^{T}(v_i-\mu_v)}}$

have 1223 input samples which I divide into 90:10 split for training and testing.

The ground truth label is captured using a modified version of Ubuntu keylogger app logkeys [3]. The app was modified to include each character with its timestamp on a new line, as well as the log file to be readable by another user, while it's being written on by the app itself.

I used a straightforward affine network with seven hidden layers. The network was implemented in PyTorch. The first 6 layers have output size of 512 and the last layer has an output size of 256. The final predict layer then takes this 256-dimensional input and outputs a single number, which we use as our predicted output for typing speed. Each of the first 6 layers are followed by a ReLU activation layer [7] and a batch normalization layer [4]. I take the loss to be the MSE loss with respect to the true typing speed, and train the network with the SGD optimizer.

3. Experiment and Results

The non-real time network was trained and tested on the input sample containing data from both smartwatches for about 50 epochs. The results on the test set are included in Fig 1 (a). The plot shown here is calculated for about 110 samples. The red curve shows the predicted typing speed (in char/min) and the blue curve is the ground truth. The accuracy for one sample is defined as whether the predicted value is within 15% of the actual value. Thus, for this scenario the test accuracy was around 45% (i.e. the predicted speed was within 15% of the actual speed 45% of the times).

However, in between my evaluations, I ran into technical

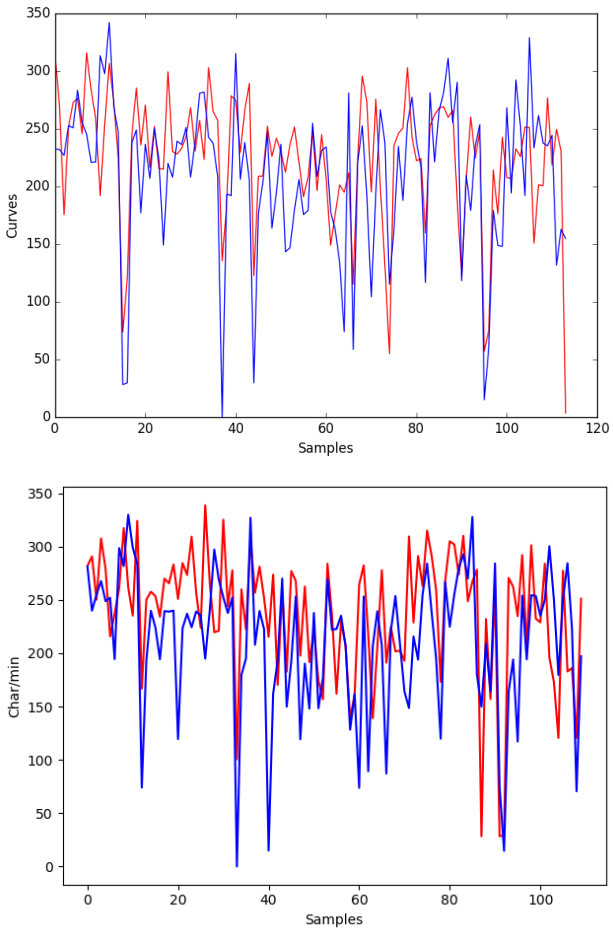

Figure 1

problems with one of my smartwatch-phone set pair. I had to scale back to using only one smartwatch-phone pair. This turned out to be a good thing because running the model again on this data yielded similar results. On further deliberation, it's easy to see that it's not fair to expect a user to wear one smart wearable device on each hand.

The new data (with 306 features) was trained with the same network for 50 epochs with a lower batch size and the predicted results for the test set are shown in Figure 1 (b). The accuracy 46% was also similar in this case. Thus, it is deduced that data from one smartwatch is sufficient to predict the typing speed. The loss curve after training on this set is also included in Figure 2.

4. Conclusion

The performance of the model as shown in Figure 1 is good in general. It follows the ground truth curve, especially in places of abrupt change. The accuracy of 46% of the model on the test set can also be significantly bumped

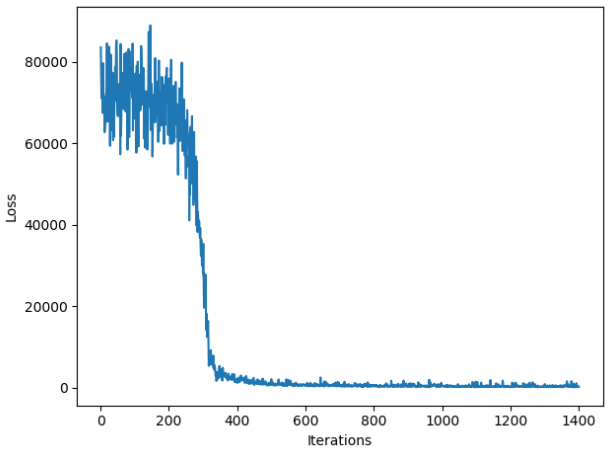

Figure 2: The loss curve during the training of network

up if we choose a higher tolerance of variation of say 20% (instead of 15). In conclusion, the built feature space from live data is a good fit for the affine network to predict the true typing speed of a user. This can then be used to infer different aspects of the mental presence of a user.

References

[1] Android. FolderSync : Available on Play Store for direct installation. `https://play.google.com/store/apps/details?id=dk.tacit.android.foldersync.lite`. [Online;].

[2] Android. SSHDroid : Available on Play Store for direct installation. `https://play.google.com/store/apps/details?id=berserker.android.apps.sshdroid`. [Online;].

[3] Badger. Key Frequency Logger - open-source Git Project. `https://github.com/bagder/keyfreq`. [Online;].

[4] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[5] A. Kumar. Sensor Data Logger : open-source Android Wear app, available on GitHub. Also available on Play Store for direct installation. `https://github.com/Steppschuh/Sensor-Data-Logger`. [Online;].

[6] A. Kumar. ClInKS: Classification of Individual KeyStrokes with Smartwatches. `https://github.com/anujkumar93/DetectKeystroke/blob/master/30517967_Report.pdf`, 2017. [Online;].

[7] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[8] Q. Pu, S. Gupta, S. Gollakota, and S. Patel. Whole-home gesture recognition using wireless signals. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13, pages 27–38, New York, NY, USA, 2013. ACM.

[9] S. Shen, H. Wang, and R. Roy Choudhury. I am a smartwatch and i can track my user's arm. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16, pages 85–96, New York, NY, USA, 2016. ACM.