Improving Software-Reduced Touchscreen Latency

Niels Henze
University of Stuttgart
Stuttgart, Germany
niels.henze@vis.uni-stuttgart.de

Huy Viet Le
University of Stuttgart
Stuttgart, Germany
huy.le@vis.uni-stuttgart.de

Sven Mayer
University of Stuttgart
Stuttgart, Germany
sven.mayer@vis.uni-stuttgart.de

Valentin Schwind
University of Stuttgart
Stuttgart, Germany
valentin.schwind@vis.uni-stuttgart.de

MobileHCI '17, September 4–7, 2017, Vienna, Austria
ACM 0-12345-67-8/90/01.
https://doi.org/10.1145/3098279.3122150

Abstract

The latency of current mobile devices' touchscreens is around 100ms and has widely been explored. Latency down to 2ms is noticeable, and latency as low as 25ms reduces users' performance. Previous work reduced touch latency by extrapolating a finger's movement using an ensemble of shallow neural networks and showed that predicting 33ms into the future increases users' performance. Unfortunately, this prediction has a high error. Predicting beyond 33ms did not increase participants' performance, and the error affected the subjective assessment. We use more recent machine learning techniques to reduce the prediction error. We train LSTM networks and multilayer perceptrons using a large data set and regularization. We show that linear extrapolation causes an 116.7% higher error and the previously proposed ensembles of shallow networks cause a 26.7% higher error compared to the LSTM networks. The trained models, the data used for testing, and the source code is available on GitHub.

Author Keywords

Touchscreen; touch input; latency; lag; prediction; machine learning; LSTM; multilayer perceptron.

ACM Classification Keywords

H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous

Introduction

All input and output devices have latency. The latency of touchscreens is especially apparent as input and output are combined in a single location (see Figure 2). While a user drags an object or draws a line across a touchscreen, the object or line follows the finger. The higher the touchscreens' latency and the faster the finger, the larger the gap between the finger and the object or line that follows. Previous work developed to use tools to measure the latency of touchscreens [2, 5] and showed that current mobile devices have a latency of a latency of 50-150ms [12, 16].

The effect of touch latency has widely been explored by previous work. Ng et al. developed a low-latency system to investigate how much latency is noticeable and how it affects users' performance. The researchers combined a high-speed projector, a high-speed camera, and an FPGA to build a touchscreen with around 1ms latency [16]. The apparatus has been used for a number of studies [1, 11, 15, 16]. It has been shown that latency down to 2ms is still noticeable for pen [15] and finger input [11]. Furthermore, Jota et al. also showed that latency down to 25ms reduces users' performance when dragging an object across the screen [11]. Recently, Cattan et al. showed that training could help to partially reduce the negative effect of touch latency for a tracking task. The authors conclude, however, that latency should still be considered as a major hindrance since learning to compensate for latency requires a sizable effort, it may not occur or transfer to every type of tasks, and it may have a high cognitive cost [4].

Cattan et al. also built a low latency touchscreen using an optical marker-based tracking system to investigate the effect of linearly extrapolating the position of a user's finger on a tabletop to reduce latency [3]. The authors showed that linear extrapolation did not improve participants' performance for devices with more than 42ms latency. The authors concluded that the linear extrapolation is strongly constrained, but that the approach can be successfully applied to counteract a large part of the negative effect of latency on users' performance.

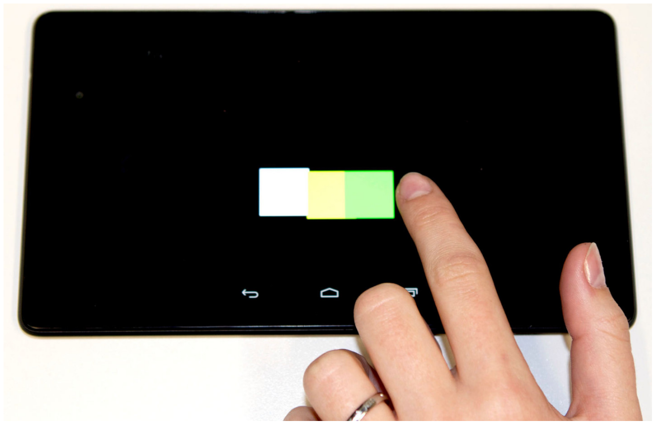
Figure 1: The effect of 100ms (white), 67ms (yellow), and 33ms (green) latency when a user drags an object across the screen.

A characteristic of all extrapolation methods is that the extrapolation has a certain error. When using extrapolation to reduce touchscreen latency, this error can become apparent as jitter. In previous work, researchers investigated the effect of extrapolation on dragging performance and subjective assessment [7]. In contrast to previous work, Henze et al. used an off-the-shelf tablet with 100ms latency. They collected a data set to train an ensemble of shallow neural networks which extrapolate the movement of a finger on a touchscreen. Comparing the trained ensemble with other approaches including linear extrapolation, they showed that the ensemble has a lower error compared to all other tested approaches. As the error increases the further the network looks into the future, Henze et al. could only show that extrapolating the next 33ms increases participants performance. Extrapolating further into the future did not further increase participants performance. In addition, the extrapolation error negatively affected participants' subjective rating [7].

In this late-breaking work, we show how we further reduce the error when extrapolation a fingers movement on a touchscreen using more recent machine learning techniques. In the following, we first describe the used data set, the architecture of the neural networks, and the training procedure. Afterward, we provide an assessment of the networks' performance by comparing them to previous work. We close the paper with a discussion of promising improvements that could further reduce the error as well as interesting potential studies.

Neural Network Architecture & Training

In the last couple of years, we witness the third wave of machine learning. Recent machine learning research demonstrated impressive performance for a very wide range of tasks. This success in machine learning was mainly enabled by combining new training algorithms, new network architectures and moving the training on graphic cards. Combined, these three aspects enable to train models that outperform previous approaches for almost all areas. In previous work, Henze et al. used an ensemble of neural networks that consists of a single hidden layer with 96 neurons [7]. While these networks outperformed polynomial interpolation and using an ensemble further decreased the error, the approach used techniques that have been the state-of-the-art over 20 years ago. Thus, we aimed to explore if more recent machine learning approaches can reduce the error.

Input and Output Vectors

Our overall approach resembles the approach used in previous work [7]. We take a stream of x/y touch positions from the touchscreen. Each touch event comes together with a timestamp. At each point in time, we can use the previous positions to predict the subsequent positions. For training, we take n (e.g. n=17) touch events as the input and output of the network. We take the first 11 touch events as the input for the network. As output for the network, we use the remaining pre-processed touch events by individually summing up their x- and y-component.

We first scale the touch events to a pixel density of 323 ppi (the pixel density of a Nexus 7) to make the events resolution invariant. We make a sequence of touch events rotation invariant by computing the angle of the vector between the last two touch events that go into the network and rotate the whole segment to align the vector with the line were x=y (45°). Afterward, we make the touch events position invariant by computing the first derivation of the x/y position, thereby deriving the speed of the finger along both axes from two sequential touch positions. In contrast to Henze et al. [7] we do not further normalize the input as we assumed that this is not needed if the training set is sufficiently large and diverse. The x- and the y-component of the preprocessed touch events are part of the input vector that goes into the network.

Architecture

We used python 3.6 and TensorFlow 1.1[1] to build the model and to train the network. TensorFlow provides the unique possibility to reduce the size of a trained model and to compile the model for deployment on Android and iOS devices by supporting quantization and lower precision arithmetic

[1]TensorFlow - An open-source software library for Machine Intelligence tensorflow.org

that reduce model size. We explored different hyperparameters for the network, including different numbers of network layers, the number of neurons in the layers, activation functions, and cost functions. We finally decided for a multilayer perceptron and an LSTM network, two fairly standard architectures, as they already showed a clear improvement regarding error compared to previous work.

We use a multilayer perceptron with four hidden layers. While the multilayer perceptron has already been described in the 1960s, using more than two layers only recently became possible through increased computing power, improved optimization algorithms, and new activation functions. In addition to the x/y positions, we add the rotation angle, the time between the touch events that go into the network, the time the algorithm should look into the future, as well as the screen size of the used device in inch. Our network has 33 input neurons (speed in x- and y-direction, the time between the touch events, the rotation angle, the screen size of the device in inches, and the amount of time the network should look into the future) computed from a sequence of 11 touch events. The hidden layers consist of 4096, 2048, 1024, 512 neurons. As neurons, we use rectified linear units, the most popular activation function for deep neural networks [14].

We use a recurrent network consisting of long short-term memory (LSTM) cells proposed by Hochreiter and Schmidhuber in 1997 [10]. The input vector consists of three values that are sequentially fed to the network. We feed the network with one x/y position after the other. We add the time between two following touch events to the input vector depending on the time the network should look into the future. When the network should look two touch events into the future, for example, we add the time between the touch events two events into the future of the x/y position. We simply use the default LSTM layers provided by TFLearn[2]. The network consists of two stacked LSTM layers with 512 and 256 cells.

Training procedure

We use the Xavier initializer described by Glorot and Bengio [6] to initialize the weights. The biases are initialized with random values from a normal distribution. For the multilayer perceptron, we use the sum of the Euclidean norm and the square of the Euclidean norm. We use L2 regularization with a factor of 0.5 and a 0.5 dropout to improve the networks ability to generalize beyond the training data. For the LSTM network, we use a mean squared loss function and 0.75 for dropout.

For both types of networks, we use the Adam optimizer [13] as it has been suggested that Adam might be the best overall choice compared to other optimizers [17]. Batches of 500 samples are used for backpropagation to reduce training time and to increase the networks' robustness. We randomize the samples between the epochs. We use a 0.00005 learning rate for the multilayer perceptron and a 0.0001 learning rate for the LSTM network.

TTo train the networks, we use data collected with an in the large approach by Henze et al. [8, 9]. Thus we use data from the same Android drawing application[3] from various smartphones and tablets. We started with data from 25,719 Android devices collected between August 8th, 2015 and December 23, 2016. We manually determined the screen size for all device types with more than 40 devices in the entire data set. We randomly selected a subset of 10,000 devices to reduce processing time. This resulted in 3,739

[2]TFLearn: Deep learning library featuring a higher-level API for TensorFlow tflearn.org

[3]The drawing application Pen & Paper in Google Play: play.google.com/store/apps/details?id=net.nhenze.penandpaper

	Writing			Drawing			Fitts' Law		
Approach	33.33ms	66.67ms	100ms	33.33ms	66.67ms	100ms	33.33ms	66.67ms	100ms
no extrapolation	15.4px	29.7px	42.5px	40.2px	77.9px	110.1px	31.7px	52.6px	66.3px
Linear extrapolation	7.8px	21.1px	38.5px	25.1px	67.9px	123.3px	16.9px	39.6px	69.0px
Neural networks [7]	6.0px	14.5px	24.8px	16.1px	36.5px	60.2px	11.4px	26.1px	43.4px
Multilayer perceptron	5.6px	13.0px	23.1px	14.3px	32.8px	55.4px	8.7px	16.8px	27.8px
LSTM network	5.1px	13.0px	23.0px	13.2px	31.9px	54.3px	7.4px	15.6px	25.4px

Table 1: Prediction error for the different approaches. Apart from the results for the multilayer perceptron and the LSTM network, the table is equivalent to the results presented in Henze et al. [7]. 10px are equivalent to 0.79mm and 10mm are equivalent to 127.20px.

devices with 116 different device types. Users produced a total of 1,123,632 strokes on theses devices consisting of 28,647,419 touch events. From this data set, we retrieved sequences of touch events to derive training samples. We excluded all samples that contain touch events with more than 99ms between them.

Preprocessing and filtering resulted in a final dataset with 2,074,655 samples for extrapolation the finger's position in 33ms, which is one magnitude more than what previous work used. We further divided the samples into 90% for training and 10% for testing. For both types, we trained three networks, each extrapolating the finger's position in two, four or six touch events from now. This is equivalent to the position in 33ms, 67ms, and 100ms assuming a constant rate 60Hz for new touch events. We trained the networks for around 500 epochs. Training of each network took on average eight hours using two ZOTAC GeForce GTX 1080 AMP! Edition.

Network Performance

To assess the performance of the trained networks and compare it to previous work we used the same validation set used in previous work (see Henze et al. [7] for a detailed description). The validation set consist of data produced on a Nexus 7 tablet with a 7.02inch (178mm) display and a resolution of 1920 $\times$ 1200 pixels. The device provides touch events and updates the screen with a constant rate of 60Hz. It has a latency of 100ms. Eight participants contributed data with three tasks: a Fitts' Law task that replicated the design by Jota et al. [11], a writing task where each participant wrote 10 phrases, and a drawing task where each participant draw a scene from their holiday.

As shown in Table 1, both networks outperform the linear extrapolation proposed by Cattan et al. [3] as well as the ensemble of neural networks [7]. For the multilayer perceptron, the average error in millimeters is 0.75mm for 33ms, 1.64mm for 66.67ms, and 2.80mm for 100ms. Compared to the ensemble of neural networks, the multilayer perceptron reduces the average error for the three tasks (Writing, Drawing, and Fitts' Law) from 11.2px to 9.5px for 33ms prediction, from 25.7px to 20.8px for 6ms prediction, and from 42.8px to 35.4px for 100ms prediction. Linear extrapolation causes a 107.8% higher error and the ensemble of neural networks causes a 21.5% higher error compared to the trained multilayer perceptron.

For the LSTM network, the average error in millimeters is 0.68mm for 33ms, 1.60mm for 66.67ms, and 2.70mm for 100ms. Compared to the ensemble of neural networks, the multilayer perceptron reduces the average error for the three tasks (Writing, Drawing, and Fitts' Law) from 11.2px to 8.6px for 33ms prediction, from 25.7px to 20.2px for 67ms prediction, and from 42.8px to 34.2px for 100ms prediction. Linear extrapolation causes an 116.7% higher error and the ensemble of neural networks causes a 26.7% higher error compared to the trained multilayer perceptron.

Figure 2 shows the error of both networks for the three tasks. Apparently, the error seems to increase super-linear, similar to the other prediction approaches. Comparing the two networks, the LSTM network outperforms or is on par with the multilayer perceptron for all tasks. Overall, the multilayer perceptron has a 4.3% higher error rate compared to the LSTM.

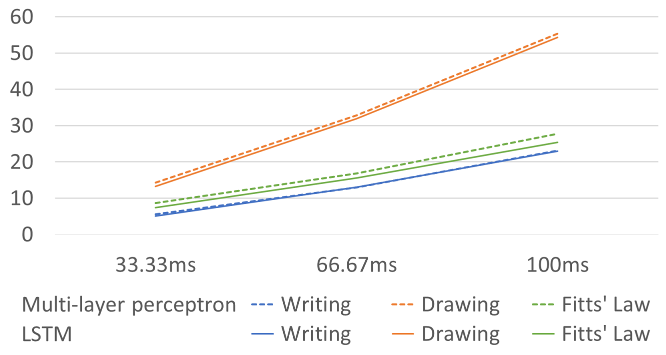

Figure 2: The development of the error for the three tasks with increasing extrapolation time.

Conclusion & Future Work

In this late-breaking work, we showed how to reduce the error when extrapolation a fingers movement on a touchscreen using state of the art machine learning techniques. We trained a networks that outperform previous machine learning-based approaches by 26.7%. Besides the reduced error, the models were trained on over 100 different device types. We assume that the model could therefore also be usable on other devices. The models, the data used for testing, as well as the source code, is available on GitHub[4].

We could envision a number of further improvements. Previous work ensured that the prediction ends on the screen, which we do not do in our assessment. This would easily reduce the error by a few percent. As previous work used an ensemble of shallow neural networks, we could use an ensemble of LSTM networks or multilayer perceptrons. A systematic hyperparameter search that additionally considers, for example, L1 regularization, other optimizers, or different architectures might also further reduce the error.

Apart from further improving the extrapolation, we are also interested in assessing the performance of the models in further studies. We intend to determine the effect on throughput by conducting a controlled study using a dragging task. On the other hand, we would like to study the performance with actual users by integrating the model into the drawing application we used for data collection. We assume that extrapolation not only has a positive effect on abstract tasks but also improves the drawing experience.

Acknowledgements: Supported by the DFG within the SimTech Cluster of Excellence (EXC 310/2) and the SFB/TRR 161. Also supported by the MWK Baden-Württemberg within the Juniorprofessuren-Programm.

[4]github.com/interactionlab/MobileHCI17-Touch-Extrapolation

REFERENCES

1. Glen Anderson, Rina Doherty, and Subhashini Ganapathy. 2011. User Perception of Touch Screen Latency. In *Design, User Experience, and Usability. Theory, Methods, Tools and Practice*. Springer, 195–202. DOI: http://dx.doi.org/10.1007/978-3-642-21675-6_23

2. François Bérard and Renaud Blanch. 2013. Two Touch System Latency Estimators: High Accuracy and Low Overhead. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces (ITS '13)*. ACM, New York, NY, USA, 241–250. DOI: http://dx.doi.org/10.1145/2512349.2512796

3. Elie Cattan, Amélie Rochet-Capellan, Pascal Perrier, and François Bérard. 2015. Reducing Latency with a Continuous Prediction: Effects on Users' Performance in Direct-Touch Target Acquisitions. In *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces (ITS '15)*. ACM, New York, NY, USA, 205–214. DOI: http://dx.doi.org/10.1145/2817721.2817736

4. Elie Cattan, Amélie Rochet-Capellan, Pascal Perrier, and François Bérard. 2017. Does Practice Make Perfect?. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 5619–5629. DOI: http://dx.doi.org/10.1145/3025453.3025585

5. Jonathan Deber, Bruno Araujo, Ricardo Jota, Clifton Forlines, Darren Leigh, Steven Sanders, and Daniel Wigdor. 2016. Hammer Time!: A Low-Cost, High Precision, High Accuracy Tool to Measure the Latency of Touchscreen Devices. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 2857–2868. DOI: http://dx.doi.org/10.1145/2858036.2858394

6. Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Yee Whye Teh and Mike Titterington (Eds.), Vol. 9. PMLR, Chia Laguna Resort, Sardinia, Italy, 249–256. http://proceedings.mlr.press/v9/glorot10a.html

7. Niels Henze, Markus Funk, and Alireza Sahami Shirazi. 2016. Software-reduced Touchscreen Latency. In *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI '16)*. ACM, New York, NY, USA, 434–441. DOI: http://dx.doi.org/10.1145/2935334.2935381

8. Niels Henze and Martin Pielot. 2013. App Stores: External Validity for Mobile HCI. *interactions* 20, 2 (March 2013), 33–38. DOI: http://dx.doi.org/10.1145/2427076.2427084

9. Niels Henze, Alireza Sahami Shrazi, Albrecht Schmidt, Martin Pielot, and Florian Michahelles. 2013. Empirical Research through Ubiquitous Data Collection. *Computer* 46, 6 (June 2013), 74–76. DOI: http://dx.doi.org/10.1109/MC.2013.202

10. Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

11. Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor. 2013. How fast is fast enough?: a study of the effects of latency in direct-touch pointing tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2291–2300. DOI: http://dx.doi.org/10.1145/2470654.2481317
12. Topi Kaaresoja and Stephen Brewster. 2010. Feedback is... late: measuring multimodal delays in mobile device touchscreen interaction. In *International Conference on Multimodal Interfaces and the Workshop on Machine Learning for Multimodal Interaction*. ACM, 2. DOI: http://dx.doi.org/10.1145/1891903.1891907
13. Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980
14. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444. DOI: http://dx.doi.org/10.1038/nature14539
15. Albert Ng, Michelle Annett, Paul Dietz, Anoop Gupta, and Walter F. Bischof. 2014. In the Blink of an Eye: Investigating Latency Perception During Stylus Interaction. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 1103–1112. DOI: http://dx.doi.org/10.1145/2556288.2557037
16. Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. 2012. Designing for low-latency direct-touch input. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 453–464. DOI: http://dx.doi.org/10.1145/2380116.2380174
17. Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *CoRR* abs/1609.04747 (2016). http://arxiv.org/abs/1609.04747