Practical Processing of Mobile Sensor Data for Continual Deep Learning Predictions

Kleomenis Katevas*
Cognitive Science Research Group
Queen Mary University of London, UK
k.katevas@qmul.ac.uk

Ilias Leontiadis, Martin Pielot,
Joan Serrà
Telefónica Research, Barcelona, Spain
name.surname@telefonica.com

ABSTRACT

We present a practical approach for processing mobile sensor time series data for continual deep learning predictions. The approach comprises data cleaning, normalization, capping, time-based compression, and finally classification with a recurrent neural network. We demonstrate the effectiveness of the approach in a case study with 279 participants. On the basis of sparse sensor events, the network continually predicts whether the participants would attend to a notification within 10 minutes. Compared to a random baseline, the classifier achieves a 40% performance increase (AUC of 0.702) on a withheld test set. This approach allows to forgo resource-intensive, domain-specific, error-prone feature engineering, which may drastically increase the applicability of machine learning to mobile phone sensor data.

Keywords

Mobile Sensing; Recurrent Neural Networks; Push Notifications; Sensor Data Processing

1. BACKGROUND AND MOTIVATION

Machine learning can turn our mobile phones into sophisticated sensing and inference tools. Data captured from mobile phones cannot only be used to infer the location or level of acceleration of our phone, but also high-level information about, *e.g.*, the environment, health & well-being, and emotional states of the phone user [11, 15, 17].

Traditional machine learning classifiers cannot typically handle raw sensor inputs, such as the level of activity as it is reported from the acceleration sensor. Therefore, sensor events have to be converted into features in order to become a relevant input for the classifier, such as the mean level of acceleration during a specified time window. Choosing which features to compute is an inherently time-consuming and creative task. Other than experience and domain knowledge, little guidelines exist on how to arrive to the best, or even to a sufficiently good set of features. Consequently, during feature extraction, important information may not be modeled and thus remain unused by the classifier. In addition, the extracted features may not be generic, in the sense that reusing them for a related but different task may be sub-optimal.

Deep learning proposes to solve this problem by learning the feature sets and the classifier at the same time, in a supervised way, and for a specific domain [2]. A cascade of neural network layers is employed, where each subsequent layer can learn more complex information, typically in a hierarchical fashion. The model implicitly identifies and learns predictive 'features' from the available dataset and for the task at hand.

Deep learning is significantly outperforming state-of-the-art methods in several domains, such as image classification [4, 8]. A number of deep-learning architectures expect their input to be of a fixed size and format. However, in the context of mobile phone sensors, events which are predictive may be sparse and occur at irregular intervals. For example, in some use cases, the events of unlocking the screen or opening an app can be important predictors. These events, however, occur only rarely and asynchronously, making them hard to map into a fixed data format. Thus, there is no direct way to feed those events into a network that expects a stable-sized input. Recurrent neural networks (RNNs) [2] are more suited for variable-length sequential data, such as the one produced by mobile sensors. Nonetheless, they are typically designed for constant rate, synchronous sequences [3].

According to Lane *et al.* [10]: "*If deep learning could lead to significantly more robust and efficient mobile sensor inference, it would revolutionize the field by rapidly expanding the number of sensor apps ready for mainstream usage*". To achieve that, research is beginning to look into how deep learning models can deal with sparse and asynchronous sequences. For instance, Lee *et al.* [13] propose a phased-triggered RNN that uses a time gate to down-sample and discretize continuous sensor input, but is not capable of 'de-sparsifying' sparse sensor data. DeepSense [21] is, to our knowledge, the only work that inputs time series mobile sensor data into an RNN. In this work, the attendance to large time spans is achieved by using a combination of convolutional and RNN layers. The framework outperformed the baselines in several tasks (*i.e.*, car tracking, activity recognition, and user identification). Even though the authors did not report any results with a wide range of mobile sensors,

*Part of the work was done during a student internship at Telefónica Research.

EMDL'17, June 23, 2017, Niagara Falls, NY, USA
© 2017 ACM. ISBN 978-1-4503-4962-8/17/06...$15.00
DOI: http://dx.doi.org/10.1145/3089801.3089802

they claim that their framework can be directly applied to almost all other sensors, such as microphone, Wi-Fi signal, barometer, and light sensor.

In this paper, we propose a pipeline for the practical processing of sparse sensor data from mobile phones for the use in a deep learning classifier. Our goal is to enable continual predictions on the basis of sensor data streams, *i.e.*, at each moment in time, the network should allow to make an estimation about the user's contextual state. The main points we tackle are:

- *Data sparsity.* We propose a data format in which sparse sensor events are represented by positive numbers whereas absence of events is represented by zeros.
- *Temporal sparsity and asynchrony.* To improve performance, we propose a time-based compression method, which reduces the sparsity of the dataset.
- *User and class imbalance.* We consider and study four ground truth weighting strategies, used in the training of our deep learning models.

We demonstrate the effectiveness of our approach in a case study on a dataset of 279 mobile phone users, where sensor data and other events are used to continually predict whether the user will attend timely to a mobile phone notification. To the best of our knowledge, this is the first work that describes how to fuse a wide range of mobile sensors to predict the user's context using recurrent neural networks.

2. DEEP LEARNING PIPELINE

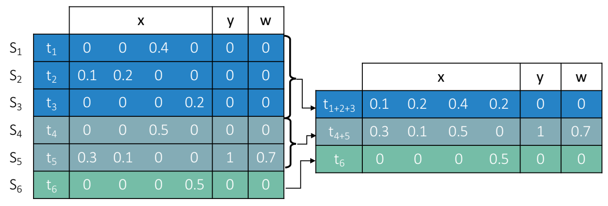
A deep neural network [2] is a series of fully connected layers of units (nodes) capable of mapping an input vector (raw data) into an output vector (*e.g.*, inferred classes). A major difference with traditional machine learning is that instead of using manually crafted features as an input, deep networks are capable of using raw data (*e.g.*, images, audio, text). An RNN is a specific type of deep network that takes sequential data as an input [2, 3]. RNNs can be stateful, *i.e.*, having an internal memory that allows them to remember past information. The most-used RNN architecture is the so-called long short-term memory (LSTM) network [5]. It has repeatedly proven to be one of the best performing off-the-shelf approaches to sequence modeling.

2.1 Prediction / Ground Truth

RNNs typically learn from a continual series of events and ground truth labels. However, in the case of mobile sensor data, the ground truth labels can be sparse. For instance, in our case study, the ground truth is the comparably rare event of attending to a notification (only 1.45% of the samples include ground truth labels). Thus, it is required that a prediction is happening continually, while the collected sensor data stream is being aggregated and the model is trained to do so, even in the absence of continual ground truth at the learning stage.

2.2 Sensor Data Collection

Mobile sensor data can be categorized into *continuous*, where the sampling rate is fixed (*e.g.*, accelerometer, light, *etc.*) and *event-driven*, where new data are reported when an event occurs (*e.g.*, battery level drops, a notification is received, *etc.*). When high precision from continuous sensors is not required, these data can be transformed into *periodical*, by aggregating the data on custom time intervals (*e.g.*, mean and maximum acceleration on every 10 minutes). Table 1 describes the periodical and event-driven sensors of our case study.



		x				y	w
S_1	t_1	0	0	0.4	0	0	0
S_2	t_2	0.1	0.2	0	0	0	0
S_3	t_3	0	0	0	0.2	0	0
S_4	t_4	0	0	0.5	0	0	0
S_5	t_5	0.3	0.1	0	0	1	0.7
S_6	t_6	0	0	0	0.5	0	0

	x				y	w
t_{1+2+3}	0.1	0.2	0.4	0.2	0	0
t_{4+5}	0.3	0.1	0.5	0	1	0.7
t_6	0	0	0	0.5	0	0

Figure 1: Example of compressing sensor data.

2.3 Normalization and Capping

Our input consists of real-valued sensor readings and one-hot encoded vectors. Before feeding the input into the network, we normalize it by re-scaling all the elements to lie between 0 and 1. In sensor data, we typically find highly-skewed, long-tail distributions. We empirically tested different thresholds above which the values are capped, and ended up using the 95th percentile of the input data.

The time stamp of each data entry can be confusing for the RNN, as the value constantly increases over time (usually in epoch format, *i.e.*, milliseconds since 1st January 1970). Thus, we replace it with the *time delta*, the time difference in minutes between the current and the previous sensor event. By capping the value at 60 minutes, we also avoid outliers in situations like the device is switched off for some time or the battery runs out.

2.4 Fusing Sensors and Ground Truth

RNNs are typically designed for synchronous data (*e.g.*, audio, text, time series). While some of the sensors are sampled in regular intervals, most inputs in mobile data are event-driven. Therefore, they exhibit irregular bursts (see Table 1 for some examples). Apart from introducing asynchronicity, event-driven inputs also result in extremely sparse input vectors.

We organize the data in a form of sensor events, stored in a two dimensional matrix (Fig. 1). Each row represents a sensor event (S_i), whereas each column represents a sensor measurement (x). Ground truth labels are also represented as a column in the matrix (y), using $w = 0$ when a ground truth label is not available. Since mobile phone sensors can be asynchronous and event-based, at every time step not all sensors can possibly provide data and ground truth labels. Therefore, we represent missing values with 0. To alleviate the issue of using 0 for both a missing value and a true 0 measurement, we re-scale data to range between 0.05 and 1.

2.5 Structuring the Data for Training

The data is structured along several dimensions:
Input sample: Each sample i contains the input data of a single instance for a single user: a tuple $S_i = (x_i, y_i, w_i)$, where x_i is a sensor data value, y_i contains the ground truth label, and w_i contains the weight of this sample, used in the error or loss function. Notice that not all samples contain a ground truth label (Fig. 1).

	Sensor	Description
Periodical	Accelerometer	Mean and maximum linear acceleration.
	Battery	Percentage of the device's battery drain per hour.
	Data	Network data activity in kb/sec (total received, total transmitted, cellular received, cellular transmitted).
	Light	Mean light level in lux.
	Noise	Mean noise levels in dB.
	Semantic Location	Location visited by the user, classified as *Home*, *Work*, *Single* (visited once), *Repeated* (visited regularly), *Passing* (passed by for a short time), and *Unknown*.
Event-driven	App	Name and category of the app that was opened by the user.
	Audio Music	Change in audio playback state (Music, No Music).
	Audio Source	Change in audio output of the device (Speaker, Headphones).
	Charging State	Charging state of the device (Charging, Not Charging).
	Notification	Post or removal of notification.
	Notif. Center	Event of accessing the device's notification center.
	Ringer	Change of the ringer mode (Normal, Silent, Vibrate).
	Screen	Change of the device's screen state (On, Off, Unlocked).
	Screen Orientation	Change of screen orientation (Portrait, Landscape).

Table 1: Periodical and event-driven sensor data collected by a smartphone device.

Sequences: To train RNNs we need to provide for each user a time-ordered sequence of input samples. These samples are used to build an internal state that determines how past events affect future time slots. They are also used to back-propagate the error when training the RNN [2]. The number of steps to perform this back-propagation in time (*sequence length*) is a parameter of the model.
Batches: Modern deep learning techniques allow us to train a network in batches by interleaving multiple sequences together. Among others, batching allows to further exploit the power of matrix multiplication on the GPU and to avoid loading all data into memory at once. The *batch size* has implications for the robustness of the error that is propagated in the learning phase [6]. Figure 2 shows an example of 3 batches that encode 3 sequences of 5 samples each (15 samples per batch in total).
User buckets: By using stateful RNNs, the internal RNN state is kept between two subsequent batches, potentially allowing it to learn sequences that are larger than the sequence length. To do so, we need to make sure that two subsequent batches interleave the same users with the same order. Therefore, we assign them into buckets: each bucket contains all the batches that are required to encode the data of its users. If the users within a bucket have a different number of sequences, we zero-pad their data and sort them so that the minimum zero padding is needed. Figure 2 shows an example of a single bucket that encodes the data of 3 users.
Prediction: The suggested arrangement into buckets and batches is only required in the training phase. For predictions we can even provide a single sample of a single user and the network will make a prediction based on the previous samples of that user.

3. PERFORMANCE IMPROVEMENTS

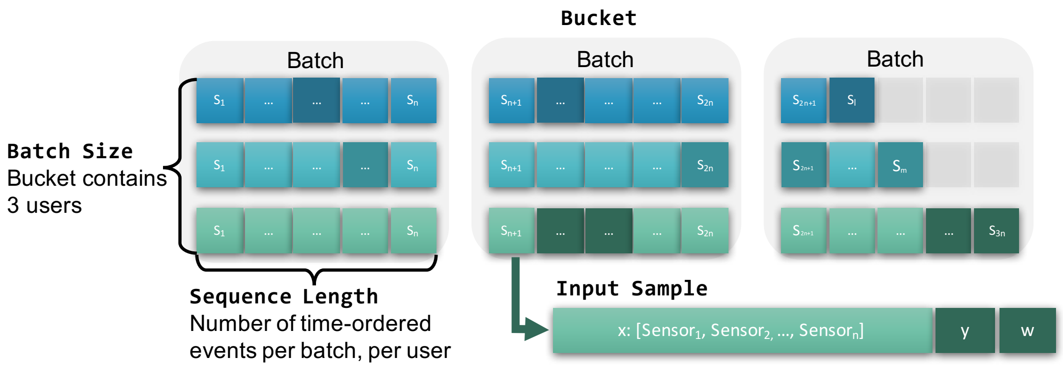
3.1 Time-Based Sparse Data Compression

Batching with a single sensor event per sequence sample has two significant drawbacks: i) sub-optimal training where each event results in a training sample with very limited information contained in it, and ii) it is imposing a challenge to the RNN's internal states that now have to accommodate longer sequences to represent the same temporal context. Therefore, we perform an opportunistic, lossless compression of the input data: consecutive input samples are combined when there is no clashing information between them. The time delta for the merged samples is updated to indicate the overall elapsed time. More specifically, data from a subsequent sample S_{i+1} can be merged into an existing sample S_i only if all of the following rules are valid for all given input sensors j (Fig. 1):

- $S_i[j] = 0$ or $S_i[j] = S_{i+1}[j]$. In other words, we can only merge the next sample into the current one if the current value is zero (no existing data) or is equal to the value of the following sample.
- $S_i[j]$ does not contain ground truth (sample weight is not zero).
- The time delta between the merged samples is not larger than a threshold T (we do not set T in our experiments as our periodical sensors are configured to a fixed sampling rate of 10 minutes).

While this compression process results in a much denser input, there are some drawbacks. Firstly, a prediction is slightly delayed until a compressed sample has been generated. Smaller T values can be used to shorten this delay. Secondly, the time information about the inter-arrival time of the compressed events is distorted. Finally, sensors that trigger multiple times with the same value can be compressed into a single event. However, performing a time-based compression presents a number of advantages that outweigh the previous drawbacks:

1. Models train faster. With smaller sequences we have less samples to feed into the classifier. If those samples keep the same information (as it is the case), the process results in faster training times with no performance drop.
2. We have less elements in the sequence. This is important since the attention to past time spans of current RNN architectures is limited, a phenomenon known as the vanishing gradients problem [14]. Therefore, by compressing longer time spans into smaller sequences we can feed more information into the RNN.



Bucket
Batch
Batch
Batch
Batch Size
Bucket contains 3 users
Sequence Length
Number of time-ordered events per batch, per user
Input Sample
x: [Sensor1, Sensor2, ..., Sensorn]
y
w

Figure 2: Preparing data for training. Users are first split into buckets and then split into batches where multiple users are interleaved. Within each batch, a sequence of user data is provided. Some (but not all) of the samples contain the ground truth that will be used to train the model.

3. The sequence size is so small that we can even think of not deploying any further processing on the phone (including the deep network) and send that information to a server performing the remaining operations.

3.2 Sample weights

Weights are traditionally used by machine-learning models to fight class imbalance: instances with significantly fewer samples typically get higher weights to force the model into considering them equally. In practice, the weights represent the contribution of each sample towards the loss function. However, in our case the weights are not only used to balance the different labels, but also for a more important task.

As described, most of the generated samples simply contain sensor readings; there are very few samples that contain labeled data. Nevertheless, even if there are unlabeled sensor readings, all samples should go through the RNN as this will keep updating the internal RNN states. In other words, even if we don't want to make a prediction at time step t, this sample might affect a future time step $t + i$. An additional benefit is that by inputting every sequence, the network will make a prediction at every input and, in fact, we want to train the network like this. In the example of Figure 2, we see that the whole sensor input is passed through the network but the network only learns from the highlighted samples.

Therefore, we need a way to indicate to the classifier that a given sample should be used to update the internal states (*i.e.*, affect the past memory) but it should not be considered by the loss function in training time. To do so, we mark samples without a ground truth label with zeroed weights, whereas for instances that contain ground truth weights are calculated based on the number of instances of this label within each user. We consider 4 different strategies: i) no weights, therefore we resort to a simple binary indicator of whether to use the sample or not, ii) inverse frequency weighting [12], iii) inverse log-frequency weighting [12], and iv) inverse square root frequency weighting.

4. CASE STUDY: PREDICTING REACTIVENESS TO NOTIFICATIONS

Notifications are alerts that try to attract the mobile phone user's attention to new content, such as unread emails or social network activity. While notifications help to avoid missing important content [16], they can have substantial negative effects. They disrupt and impair work performance, even when they are are not attended [18]. Constant exposure to notifications can negatively affect well-being [9], as they induce symptoms of hyperactivity and inattention. At the same time, notifications are essential for people to keep up with expectations towards responsiveness [16]. The research community is therefore investigating ways to reduce the negative effects of notifications. One approach that the community follows is to predict how reactive a user would be to a notification [20] to enable intelligent ways of handling them. In this section, we present a case study where we predict whether a user will react (click or dismiss it) to a mobile phone notification within a 10 minutes window.

4.1 Data collection

Our dataset contains mobile phone use logs from 279 Android phone users for an average duration of four weeks during summer 2016. The participants' ages ranged from 18 to 66 years ($M = 37.7$, $SD = 11.1$), with a balanced gender split (52.7% female and 47.3% male). The data was collected through an app which was running in the background while passively collecting rich sensor data about the user's context and phone usage. Participants registered the app to listen for notification and accessibility events, which allowed it to log what notifications participants received and after how much time they opened the corresponding app.

Table 1 presents a list of all sensors used in this study. Based on the time stamp of each entry, we extracted some simple information such as the time delta (explained in Sec. 2), the day of the week (1–7), the hour of the day (0–23), as well as a variable that indicates whether the current day is a working day or not (0–1). Basic demographics such as age and gender were also self-reported using a questionnaire at the beginning of the study and included in the dataset.

We computed the ground truth using 1 when a notification arrives and the user opens the app that originated the notification in less than 10 minutes, and 0 if the user either removed it from the notification center or just ignored it. We excluded all system and keyboard type notifications events from the ground truth, where a consequent action was not usually required by the user. The resulting dataset contains over 26 million phone usage events, about 1 mil-

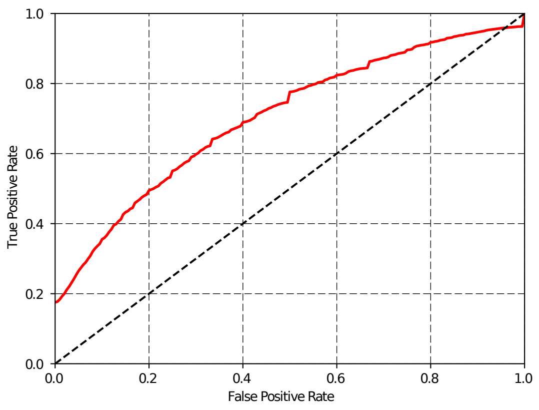


Figure 3: ROC curve of the test set using the time-based compressed data and logarithmic weights (AUC = 0.702).

	Valid	Test	Unknown Test
Frequency	0.679	0.681	0.669
Square root of frequency	0.697	0.698	0.667
No weights (binary)	0.706	0.700	**0.696**
Logarithm of frequency	**0.713**	**0.702**	0.691

Table 2: AUC per weight type using the time-based compressed data.

	Valid	Test	Unknown Test
Baseline	0.495	0.499	0.511
Uncompressed	0.688	0.678	0.671
Compressed	**0.713**	**0.702**	**0.691**

Table 3: AUCs using logarithmic weights.

lion events after applying the sensor data compression, and about 388 thousand ground truth labels.

4.2 Data Analysis

For the analysis, we split the 4-week sequential dataset into training (first two weeks), validation (3rd week), known test (4th week) and unknown test (4th week). The difference between the two test sets is that the unknown test set includes 22 new users that the model has never seen before. Following the pipeline explained in Sec. 2, we applied one-hot encoding to all categorical sensors, replaced all NaN values with zeros, applied normalization and capping, and finally applied time-based compression to the dataset.

To implement our model we used Keras v2.0.3 [1] with Theano v0.9 [19]. As an input layer we used a fully-connected time-distributed linear layer with 50 parametric rectified linear units [4]. Two stateful LSTMs were used as hidden layers with 500 units. A final dense layer and a sigmoid activation function was applied to obtain output probabilities. We trained our model using standard cross-entropy loss [2] and the Adam optimizer [7] with default parameters.

As a baseline, we used a probability-based dummy classifier. On the basis of the training set, it determines, for each user, the probability that a notification of a certain category will be clicked within 10 minutes. In the prediction phase, it uses this probability as threshold in a random prediction. For example, if a user responded to 80% of the WhatsApp messages within 10 minutes, the prediction will yield about 80% positive predictions.

4.3 Results

In Table 3, we report the area under the curve (AUC) of the classifier using both the compressed and uncompressed datasets. The AUC is computed per user and per app category, and then averaged. Overall, we achieved an AUC of 0.70 in the test set and 0.69 in the unknown test set. Similar accuracies in both test sets suggest that the model is resilient to users outside the training set, which would be a very desirable property. By applying the time-based compression, we achieved a 95% size reduction of the dataset and a 3.5% relative improvement when predicting notification attendance. In addition, the model training time improved significantly from 1.3 hours to 2.8 minutes per epoch. We note that without the normalization and capping part described in Sec. 2, the model presented some convergence issues. In Fig. 3 we report the performance of the classifier in a Receiver Operating Characteristic (ROC) curve.

In Table 2 we compare the accuracy of the model using the four considered types of weights (Sec. 3.2). Apart from using the inverse of the frequency, which performed worse, we do not observe a substantial effect. Logarithmic weighting performed best in validation and test. Binary weights outperformed the rest in the case of the unknown test. However, due to its small size ($n = 22$), the unknown test set was subject to high variance, preventing us from drawing clear conclusions.

5. CONCLUSIONS AND FUTURE WORK

We introduce a practical approach for preparing time series mobile sensor data for deep learning applications. We demonstrate its effectiveness in a case study with 279 participants. An RNN trained on data prepared with our approach achieved a 40% performance increase with respect to a probabilistic random baseline in the task of predicting whether a notification would be attended within 10 minutes. We find that the model generalizes to unknown users without significant performance loss.

The proposed data processing approach enables running continual predictions on mobile sensor data streams. The proposed time-based compression further enables practical implementations, where the phone collects and compresses the data, and then sends it to server to run predictions. Future work includes the comparison of the performance to canonical approaches, the improvement of the compression strategy, and the potential application of more sophisticated deep learning techniques, such as transfer learning, or unsupervised learning with the use of generative adversarial networks.

6. ACKNOWLEDGMENTS

The authors wish to thank the participants of the study. They also wish to thank Alexandros Karatzoglou for the useful discussions.

7. REFERENCES

[1] F. Chollet. Keras. https://github.com/fchollet/keras, 2015.

[2] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT Press, Massachusetts, USA, 2016.

[3] A. Graves. Generating sequences with recurrent neural networks. ArXiv: 1308.0850, 2013.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In *Proc. of the IEEE Int. Conf. on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[6] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: generalization gap and sharp minima. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2017.

[7] D. P. Kingma and J. L. Ba. Adam: a method for stochastic optimization. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*, 2015.

[8] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pages 1097–1105. Curran Associates Inc., 2012.

[9] K. Kushlev, J. Proulx, and E. W. Dunn. "silence your phones": Smartphone notifications increase inattention and hyperactivity symptoms. In *Proc CHI '16*, pages 1011–1020. ACM, 2016.

[10] N. D. Lane and P. Georgiev. Can deep learning revolutionize mobile sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 117–122. ACM, 2015.

[11] N. D. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. T. Campbell. A survey of mobile phone sensing. *Comm. Mag.*, 48(9):140–150, Sept. 2010.

[12] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[13] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 3882–3890. Curran Associates, Inc., 2016.

[14] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 1310–1318, 2013.

[15] M. Pielot, T. Dingler, J. S. Pedro, and N. Oliver. When attention is not scarce - detecting boredom from mobile phone usage. In *Proc. UbiComp '15*, UbiComp '15, pages 825–836. ACM, 2015.

[16] M. Pielot and L. Rello. Productive, anxious, lonely - 24 hours without push notifications. In *MobileHCI '17*, 2017.

[17] S. Servia-Rodríguez, K. K. Rachuri, C. Mascolo, P. J. Rentfrow, N. Lathia, and G. M. Sandstrom. Mobile sensing at the service of mental well-being: A large-scale longitudinal study. In *Proc. WWW '17*, pages 103–112, 2017.

[18] C. Stothart, A. Mitchum, and C. Yehnert. The attentional cost of receiving a cell phone notification. *Journal of experimental psychology: human perception and performance*, 41(4):893, 2015.

[19] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[20] L. D. Turner, S. M. Allen, and R. M. Whitaker. Interruptibility prediction for ubiquitous systems: Conventions and new directions from a growing field. In *Proc UbiComp '15*. ACM, 2015.

[21] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. *arXiv preprint arXiv:1611.01942*, 2016.