Alawnah and Sagahyroon *EURASIP Journal on Embedded Systems* (2017) 2017:22
DOI 10.1186/s13639-017-0070-1

EURASIP Journal on Embedded Systems

RESEARCH Open Access

Modeling of smartphones' power using neural networks

Sameer Alawnah and Assim Sagahyroon*

Abstract

In the work presented in this paper, we use data collected from mobile users over several weeks to develop a neural network-based prediction model for the power consumed by a smartphone. Battery life is critical to the designers of smartphones, and being able to assess scenarios of power consumption, and hence energy usage is of great value. The models developed attempt to correlate power consumption to users' behavior by using power-related data collected from smartphones with the help of specially designed logging tool or application. Experiences gained while developing the model regarding the selection of input parameters to the model, the identification of the most suitable NN (neural network) structure, and the training methodology applied are all described in this paper. To the best of our knowledge, this is the first attempt where NN is used as a vehicle to model smartphones' power, and the results obtained demonstrate that NNs models can provide reasonably accurate estimates, and therefore, further investigation of their use in this modeling problem is justified.

Keyword: Smartphone, Energy, Power models, Neural networks

1 Introduction

Recent advents in battery technology have not paralleled the rapid advances in chip design and wireless telecommunication, and it is often the case that the computing power is limited by the battery capacity. This has brought the problems of power consumption, low power design, energy-efficiency, and optimal power management to the forefront of research issues pertaining to portable electronics. This problem is further aggravated for smartphones by the fact that today's consumers are expecting lighter devices that can run for long hours, and hence, designers have to rely on smaller and lighter batteries that typically have reduced energy storage capabilities.

Thus, designers and manufacturers are keen on understanding the power consumption characteristics of today's smartphones where the primary objective is to be capable of designing better future generations. Developing power models for these devices is critical since it provides designers with assessment capabilities early in the design cycle, which in turn would lead to developing sound energy management policies, and assists software developers in writing applications that are energy-efficient.

The need to try and understand the role of users' behavior and how it contributes to energy consumption with the hope of designing better systems, and making efficient use of the available energy is emphasized in two recent papers [1, 2].

The authors of these two papers studied the smartphone usage activity of a large number of users. They showed that the usage activity is quite diverse among users. This extent of usage diversity implies that mechanisms that work for the average case may be ineffective for a large proportion of the users. In the case of power modeling, usage diversity means average-case power models that may be insufficient to accurately predict power consumption for different users, and hence, a usage activity-based power model is essential to accurately predict power consumption.

The power modeling problem was addressed in [3] using pattern analysis by first segmenting users' logged data into a number of small time windows called "chunks." A chunk is a set of power-related data collected and computed during 1% of used battery capacity. Chunk data includes components such as average power, CPU utilization, frequency, and display activity. The

* Correspondence: asagahyroon@aus.edu
American University of Sharjah, Sharjah, United Arab Emirates

researchers grouped the chunks based on the hardware components accessed. Chunks with standard small deviations were kept, and others were discarded. Regression analysis was then performed to generate a model. In the reported work, only power models related to the CPU and display unit were developed. Similarly, regression-based techniques to model power were reported in [4]. Using a logger application, data was collected from different users and effective predictors were selected to develop power models. A model for energy level prediction is presented in [5]. Researchers collected data from a set of Blackberry smartphone users and then exploited energy traces within the collected dataset to build an energy emulation toolkit. Users here were divided into three groups depending on their energy consumption characteristics, that is, their daily battery charge and discharge pattern or behavior. A prediction algorithm was then used to predict energy level. Using results from the energy emulation kit, developers could then modify their designs or fine-tune certain parameters to optimize energy consumption. A battery-based power model is discussed in [6]. Based on measurement-oriented experiments, researchers excluded the hardware components with negligible power consumptions, such as the SD card, from being used in the model. The modeled components were CPU, display, GPS, Wi-Fi, cellular, and audio interfaces. A set of training programs was next used to determine the relationship between a state variable and the power consumption for each hardware component selected for the model. The authors next proposed the use of battery discharge behavior and the built-in battery voltage sensors in some smartphones to determine the average power consumption that resulted from the varying power states of the different components. The smartphone components were held in a particular state for long periods of time while the state of discharge of the battery was monitored using the built-in voltage sensors, therefore providing an estimate of the power consumption for the particular activity state. The total consumed energy within that test period or interval was then computed. This was repeated for different states, and regression techniques were used to derive models based on battery behavior. Additional work that sheds more light onto the problem at hand and discusses possible solutions or alternatives is reported in [7–10].

In this work, and in an attempt to contribute to better designs of smartphones, we will approach the power modeling problem from a user-behavior point of view. We developed a logger application (running in the background) that tracks the users' interaction with their smartphones over a period of time. It creates and logs power-related records by making use of the smart battery interface built in the device. The datasets logged over time are then used to develop power models using neural network modeling approaches. To the best of our knowledge, this is the first work that attempts to estimate smartphone power using neural network techniques. Furthermore, the large body of published work on power modeling uses a utilization-based approach where the focus is on estimating the power consumption of individual hardware components that make up the phone, using for example, performance counters, and then estimating the power of the phone when these components switch between different operating modes. Instead, in our work, we use *user activity* as the basis for developing the model. The only reported work that we came across where users' profiles are used as part of the modeling process is reported in [5]. However, they used regression-based techniques and selected a smaller and different set of parameters when compared to the set used in the work discussed here. Preliminary results related to this work are reported in [19].

In general, neural networks have some advantages when compared to regression techniques such as [20]:

- Modeling using neural networks requires less formal statistical training
- Neural network models have the ability to detect all possible interactions between predictor variables
- Neural networks can be developed using multiple and different training algorithms
- Neural networks are capable of identifying complex non-linear relationships between dependent and independent variables. Conventional regression techniques typically assume a linear relationship.

The rest of the paper is organized as follows: in Section 2 we introduce the logger application developed for this work and explain its usage; out of many logged input parameters, we also discuss the selection of the most influential parameters used to develop the NN models. In Section 3, we present the model development steps including training phase and NN configuration selection; we also assess the performance of the model. The paper is concluded in Section 4.

2 Data logging and parameters identification

Most of the power consumed is typically broken down between components that include the CPU, memory banks and controller, GSM, GPS, Bluetooth, LCD panel and touch screen, LCD backlight, Wi-Fi, audio (codec and amplifier), internal NAND flash, SD card, and camera. The system load or application heavily influences the power needs of these components. For example, if the load is a video game, more power will be drawn by

these components than, when for example, the load is simply an editing session of a text file. Hence, usage patterns and user behavior directly influence battery life.

We selected a sample of ten campus students (American University of Sharjah, www.aus.edu) and monitored their activity with the assistance of a data logger application program installed in their *Android-based* smartphones. The logger application continuously ran in the background without compromising the users' privacy. It runs continuously and starts when the mobile is turned on. We collected the maximum amount of usage-related data for synthesis and analysis as described in later sections.

The logger application logs power-related records using the smart battery interface inside the device. The datasets logged overtime are then used to develop power models using neural network techniques.

The smartphone model (Sony Acro S—an Android-based device) selected for this work contains modern lithium-ion batteries with a smart battery interface that monitors the charging and discharging process to protect it; additional information about battery capacity, current, voltage, and temperature is also provided to the power management program that is part of the operating system. It provides drivers to these interfaces as Linux virtual files to read different parameters of the battery status. Most of the recent Sony-Ericsson family of smartphones is provided with this sensor interface unit that is one of the main reasons for choosing the Sony Acro S model as a testing platform for the work discussed here. This model uses the Qualcomm MSM8260 Snapdragon processor, which is an asynchronous symmetric dual-core processor.

The logger application makes valuable use of a built-in Android mechanism called broadcast/receiver. The operating system broadcasts messages about events that are taking place, such as a battery status change, the Wi-Fi connection being turned on, or the screens being switched off. However, some relevant and power-impacting usage parameters have no broadcast actions or messages associated with them, and in this case, we use *polling* as a means of sampling changes in these parameters (examples, include "current" and "audio utilization").

Some of the parameters have Operating System counters associated with them, like CPU and memory usage, reading the difference between these counters at predefined time intervals will give us the values of them during those intervals. Other parameters like electrical current drawn from the battery and audio subsystem utilization don not have any OS counters associated with them; therefore, we need to sample their values at a relatively high rate.

The power value at any time instant is defined by $P = V \times I$, where V and I are the voltage and current at any instant in time, respectively. In an Android-based smartphone, we can obtain the voltage value at any time using an Android API; however, finding the current value is not trivial.

The smartphone used here is equipped with a TI BQ27520 Battery Fuel Gauge IC [11]. This gauge resides on the system main board and uses a 400-kHz I^2C™ interface for connection to the microcontroller port. It is capable of measuring battery charge level, voltage, current, and temperature. According to the datasheet, we can read the instantaneous current and the average current through the I^2C interface. The value of the average current is updated every second, so sampling the current at 1 Hz is enough to capture the overall average current passing through the device.

The Linux kernel used by Android phones that support the BQ27520 such as the Acro model provides a virtual file system driver for the phone through the: "/sys/class/power_supply/bq27520/" folder. For example, reading the file "current_now" will give us the instantaneous current, while reading "current_avg" will provide us with the average current.

In the logger application, we define a timer (application timer) to invoke a function every 1 s; using this function, we sample the current measurements and audio utilization since these variables do not have OS counters.

Collected usage samples are saved on the file system of the smartphone and then uploaded to a database server for further analysis rather than being processed locally on the smartphone.

After parsing the log file, we next run an application developed to extract power usage samples from the stored data. Some of the sample parameters are trivial, for example, "screen brightness," so no extra processing is needed for this parameter. On the other hand, other parameters such as "Data Activity" require normalizing with respect to the unit of time. We receive the information per sample time but we normalize this value to a 1-s interval. The processing required to extract power sample parameters can be divided into the following categories:

- No processing:
  In this case, we use the parameter values exactly as in the log file, for example, the "screen brightness" parameter.
- Normalization per 1 s:
  In this case, we have the parameter value per sample window time, but since the sample window time is not constant, we need to normalize the parameter value per 1 s.
- Weighted averaging:
  In this case, the parameter values may change several times during the sample window; hence,

we calculate the weighted average of it, an example is the RSSI (received signal strength indication) parameter.

Using the logged data, we are able to extract power data related to more than 30 parameters of a smartphone including brightness level, data activity, phone-ringing, Wi-Fi connectivity, and SMS activity. For a complete listing of the identified parameters, readers are referred to Table 5 of [12].

In developing the neural network model, a critical step is to determine if all possible input parameters are required or whether a subset would suffice to develop a reliable model. The elimination of unnecessary inputs that have negligible contribution to the prediction will lead to a simplification in the data-gathering phase and an enhancement to the model; it also eases the interpretation of results. For example, nowadays, most students are using Internet-based services, such as "Whatsapp" and "Google Talk" for text messaging; hence, SMS activity is very low. We logged only 64 SMS activities in our dataset. This number of samples is not sufficient for modeling this activity therefore we did not include SMS activity as an input to our model. Similarly, Bluetooth is rarely used in our dataset so we did not include any Bluetooth predictors in our model. We next grouped parameters together using functionality as a criterion, and then corresponding heat maps are used to eliminate some of input parameters that strongly correlate to one another within the group. For example, the heat map of Fig. 1 is used to reduce parameters that relate to data activity functionality of the device.

For modeling mobile data communications, we have Data Activity, Data Activity On, DataConOn, and GSMRSSI predictors. Figure 1 shows the heat map for the previously mentioned predictors. It is clear that Data Activity On and DataConOn are correlated. Logically, Data Activity, and Data Activity On are also correlated, since there will be no Data Activity unless Data Activity On is greater than 0. We prefer to remove Data Activity On since it is correlated with the two other predictors.

After excluding unnecessary input parameters to the model, the subset selected and used for power model generation is given in Table 1. It consists of 17 inputs or predictors. The first column of the table includes the parameter or predictor name, followed by a brief description of it in column 2; the "Generation Method" describes the technique used to obtain the value of the input parameter, either using Android broadcast Actions/Receivers or polling OS counters at a low rate or polling device information at a high rate. The range of values is specified in column 4.

The logger application is implemented in Java using more than 2300 lines of code. Figure 2 shows the cumulative distribution function (CDF) of the logger CPU time. In terms of CPU time, the average logger overhead (logger CPU time/total CPU time) is 0.62%. We also conclude that 78.14% of the samples have logger overhead less than or equal to 0.6667%, and 99.5% of the samples have logger overhead less than or equal to 5%.

3 A neural network for power estimation

The origin of the modern neural networks (NN) science was the work published by Warren McCulloch and Walter Pitts [13], who showed that neural networks could, in principle, compute an arithmetic or logical function. The elementary element of the NN is the artificial neuron. Figure 3 is a depiction of an artificial neuron.

The individual inputs $p_1, p_2, \ldots, p_R$ are each weighted by corresponding elements $w_{1,1}, w_{1,2}, \ldots, w_{1,R}$ of the weight matrix $\boldsymbol{W}$. The net input n can be computed using Eq. (1) below:

$$n = w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R + b \tag{1}$$

where b is the bias of the neuron. Equation (1) can be written in a matrix form as:

$$n = \boldsymbol{W}\boldsymbol{p} + b \tag{2}$$

Now, the neuron output can be written as $a = f(\boldsymbol{W}\boldsymbol{p} + b)$ where f is the transfer function.

The transfer function f may be a linear or non-linear function of the net input n. Our power model generation

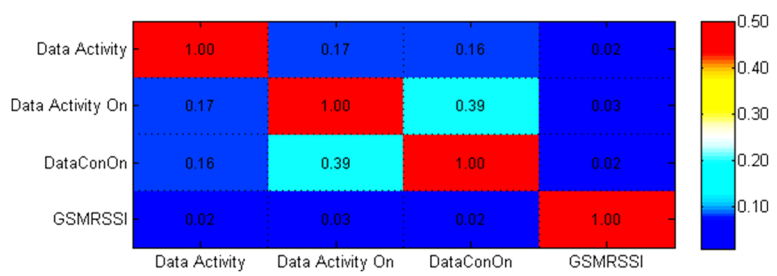


Fig. 1 Heat map relating to data activity parameters

Table 1 Input parameters to the model

Predictor name	Description	Generation method	Range
MF utilization	CPU utilization while operating at the median frequency rang	Variable polling with OS counters	0–1
HF utilization	CPU utilization while operating at the high frequency range	Variable polling with OS counters	0–1
Screen on	Fraction of time screen was on	Broadcast receivers	0–1
Screen brightness	Average screen brightness	Variable polling	20–255
Call ringing	Fraction of time smartphone was ringing	Broadcast receivers	0–1
Call off-hook	Fraction of time smartphone was in call	Broadcast receivers	0–1
Data on	Fraction of time smartphone was connected to some mobile network	Broadcast receivers	0–1
Data traffic	Average number of bytes sent/received through mobile network per second	Variable polling with OS counters	≥0
WIFI on	Fraction of time phone is connected to some WIFI network	Broadcast receivers	0–1
WIFI traffic	Average number of bytes sent/received through WIFI interface per second	Variable polling with OS counters	≥0
SD traffic	Average number of sectors read/written per second	Variable polling with OS counters	≥0
Audio on	Fraction of time audio device was active	Broadcast receivers	0–1
GSMRSSI	Average mobile received signal strength indication	Broadcast receivers	(−113)–(−48) dBm
NET HSDPA	Fraction of time mobile connected to HSDPA network (3G)	Broadcast receivers	0–1
NET EDGE	Fraction of time mobile connected to EDGE network (2.5G)	Broadcast receivers	0–1
NETGPRS	Fraction of time mobile connected to GPRS network (2G)	Broadcast receivers	0–1
GPS on	Fraction of time GPS adapter is on	Broadcast receivers	0–1

is considered to be a fitting problem. Usually, three types of transfer functions are used for these kinds of problems; others are usually used for classification problems. Table 2 lists the three transfer functions investigated in the work discussed here.

The selection of the transfer function and the number of inputs will define the structure of the neuron; the process of choosing weights and bias to generate the right output is called the training of the neuron. A single neuron has very limited modeling capabilities. Usually,

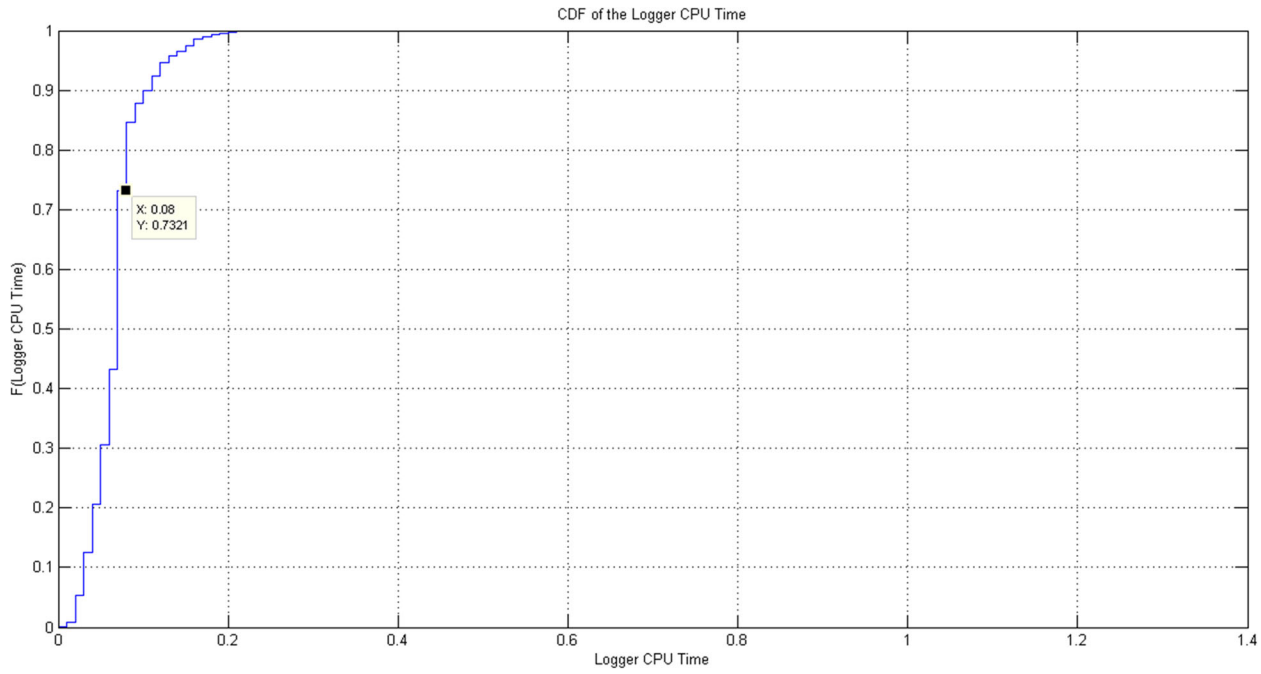


Fig. 2 CDF for the logger CPU time

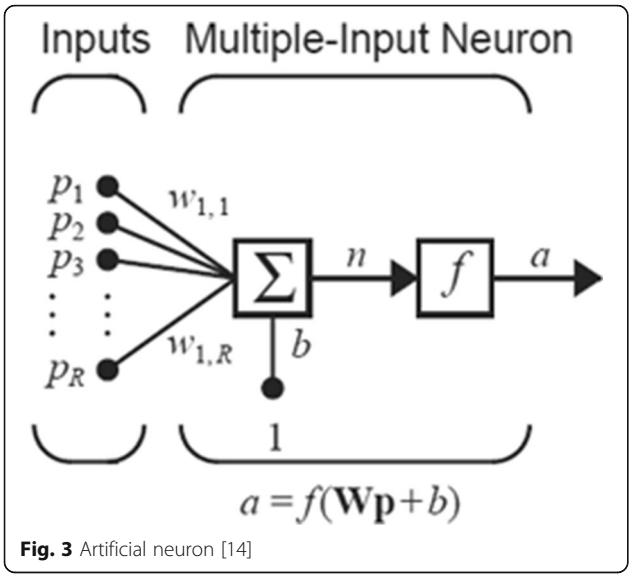


Fig. 3 Artificial neuron [14]

multiple layers of many neurons are used for modeling. The number of layers and neurons per layer and the type of connections between them defines the neural network architecture. In recent years, neural networks have been applied to model and solve various problems in the engineering field [15–18].

A. Neural network architecture and training
In this work, we will use MATLAB Neural Network Toolbox (included in the MATLAB Environment) which supports different neural network architectures. In an effort to identify the most suitable architecture, we experimented with different types of neural network architectures using various training algorithms while computing the RMSE (root mean squared error). These architectures included feed-forward back-propagation (FFBP), cascade feed-forward back-propagation (CSFFBP), and feed-forward time delay (FFBPTD). The results of the comparison are listed in Table 3. The Neural Network Toolbox has a number of training functions to train a network. For a description of these training functions, readers are referred to Table 11 of [12]. All neural network power models developed and tested in this study have 17 inputs (identified in Table 1) with one or more hidden layers with different transfer functions and one output layer. For any dataset, 70% of it is randomly selected as a training dataset, 15% is selected as validation dataset, while the rest (15%) is used as the testing dataset.
Table 3 contains the RMSE values computed using various architectures and training methods. In the table, the first column specifies the training algorithm used to train the neural network, columns 2 and 3 are for the FFBP network structure with 10 and 20 neurons in the hidden layer, respectively, and columns 4 and 5 are for the CFFBP network structures with 10 and 20 neurons in the hidden layers, respectively. The same order is applied for the FFBPTD.
It is clear that *trainbr* and *trainlm* training algorithms have the best performance over all other algorithms. The FFBPTD network architecture did not perform well with any training algorithm (high and not-improving RMSE); hence, we will not include it in any further analysis.
Table 4 contains the training time required to train each network. The organization of this table is the same as Table 3 except that the entries in this table are the training time in seconds, not the RMSE values. Table 3 shows that *trainbr* has better performance over *trainlm*; however, results in Table 4 show that *trainbr* requires two times the training time of *trainlm* with an RMSE maximum improvement of only 3.5%. Finally, *traingdm* not only requires the least time to train the network but also has the worst performance.
Thus, FFBP and CSFFBP networks trained with either *trainlm* or *trainbr* produce the best

Table 2 Transfer functions

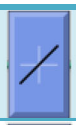
Name	Functionality	Icon	MATLAB name
Linear	$a = n$		**purelin**
Log-sigmoid	$a = \frac{1}{1+e^{-n}}$		**logisg**
Hyperbolic tangent sigmoid	$a = \frac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		**tansig**

Table 3 RMSE comparison of different architectures and training algorithms

Training function	FFBP		CFFBP		FFBPTD	
	$n = 10$	$n = 20$	$n = 10$	$n = 20$	$n = 10$	$n = 20$
trainbfg	0.209797	0.206758	0.216204	0.199763	0.632268	0.632285
trainbr	0.193328	0.175663	0.184413	0.174294	0.632267	0.632267
traincgb	0.218694	0.217632	0.21954	0.217823	0.632268	0.632271
traincgf	0.229647	0.226134	0.227723	0.213663	0.632267	0.632267
traincgp	0.225715	0.219247	0.227335	0.217196	0.632269	0.632268
traingd	0.353894	1.192112	0.336454	5.590423	0.632267	0.63227
traingdm	0.966383	0.845086	3.510545	5.550865	0.883342	0.654927
traingda	0.332997	0.39005	0.454985	0.482595	0.632267	0.632568
traingdx	0.286482	0.284275	0.306612	0.317421	0.632344	0.632833
trainlm	0.188031	0.176105	0.190761	0.180157	0.632267	0.632283
trainoss	0.233728	0.236423	0.223991	0.234473	0.632361	0.632269
trainrp	0.231186	0.239729	0.242783	0.247439	0.632274	0.806998
trainscg	0.22851	0.229321	0.232129	0.218688	0.632267	0.632268

performance. Next, we study all the combinations of these network structures and training algorithms using different transfer functions. The main goal here is to select the neural network that yields the most accurate power model.

We keep the transfer function of the output layer (*purelin*) while changing the transfer function of the hidden layer to either *logsig* or *tansig* and changing the number of neurons in the hidden layer for both FFBP and CFFBP networks. We plot the RMSE of the final model and its training time in Figs. 4 and 5,

Table 4 Training time comparison

Training function	FFBP		CFFBP	
	$n = 10$	$n = 20$	$n = 10$	$n = 20$
trainbfg	560.1874	577.9164	488.3009	885.5299
trainbr	529.1933	661.35	709.0895	944.2664
traincgb	685.6228	715.9561	686.6167	762.2569
traincgf	728.115	640.3829	1034.791	1118.346
traincgp	683.7995	674.8491	507.0572	1003.557
traingd	252.4434	9.03335	397.8197	9.829319
traingdm	10.21414	19.31096	8.943479	23.59974
traingda	73.37219	124.8304	168.5904	186.7978
traingdx	149.7136	160.7029	137.9587	162.3984
trainlm	268.3789	380.7035	576.4763	929.8259
trainoss	375.6005	345.4326	854.437	658.3433
trainrp	253.1726	268.0829	407.9637	418.6255
trainscg	393.1616	483.8855	295.2382	835.7505

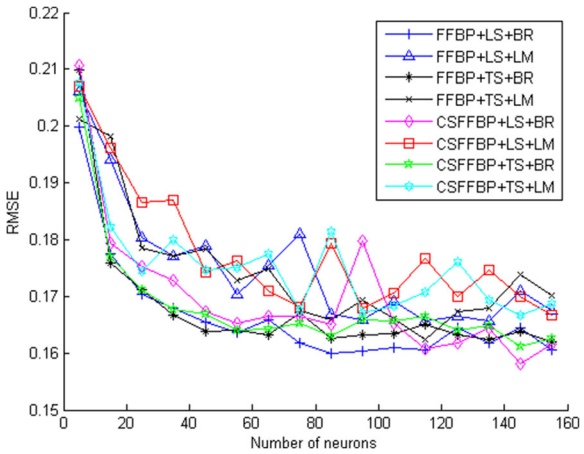


Fig. 4 RMSE of the neural network power model for various network configurations

respectively. In these figures, LS denotes the *logsig* transfer function, TS denotes the *tansig* transfer function, BR denotes *trainbr* training algorithms, and LM denotes the *trainlm* training algorithm. It is clear from the figure that *trainlm* has some random nature; the RMSE did not improve with increasing number of neurons. Furthermore, *trainbr* is more stable than *trainlm*; the RMSE improves with increasing number of neurons. We also note that FFBP has a slightly better performance than CSFFBP. We therefore select as the best NN the FFBP networks trained using *trainbr* with a hidden layer consisting of 85 neurons each, and using *logsig* as the transfer function.

Figure 5 shows the time required to train different network configurations. We observe that training time when using *trainlm* is lower than when using

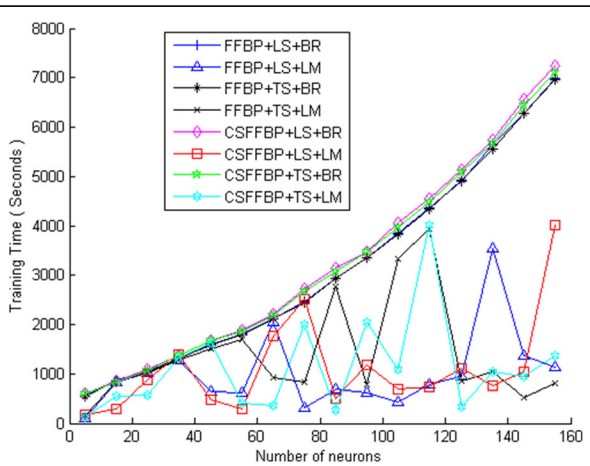


Fig. 5 Training time of neural networks power models for various configurations

trainbr, but it exhibits some randomness. For our selected configuration, the time required to train the network is about 3000 s (50 min); this is an affordable time when we want to build the model once using a typical computer. If we want to build the model for each user using his/her own mobile as computation platforms, it is better to use simpler and lower-training-time configurations even at the cost of expected accuracy.

B. Neural network performance

Drawing from the experimental exercises described above, the selected structure for our final neural network power model is depicted in Fig. 6. The training algorithm for this network is *trainbr*. The hidden layer consists of 85 neurons each using the *logsig* transfer function, while the output layer contains only one neuron that uses the *purelin* transfer function.

Figure 7 shows the performance of our neural network power model when compared to the measured. It is clear that the accuracy achieved is acceptable, with an R value of 0.96747 which is very close to the ideal value of unity.

In the selected architecture of Fig. 6, we used a neural network with one hidden layer to model the smartphone power consumption. Next, and to examine the effect of adding another hidden layer to the neural network, we added another layer and experimented by varying the number of neurons in each layer and the used transfer functions as well, while computing the RMSE for the different configurations. Results in Table 14 of [12] show that we achieve the best estimates using the network configuration where the first transfer function is *tansig* and second is *logsig*. We choose the number of neurons in the hidden layers to be 45 and 25 in the first and second hidden layers, respectively. The time required to train this networks was 3776 s (63 min).

Figure 8 depicts a performance comparison between the two-hidden-layer neural network and one-hidden-layer neural network. We note that the performance of the two-hidden-layer networks is slightly better than that of the one-hidden-layer network.

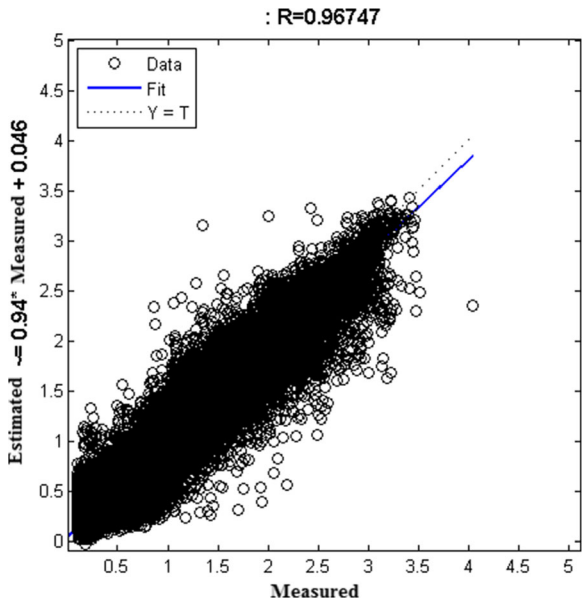


Fig. 7 Performance plot of single-hidden-layer neural network

These neural network configurations are next tested by comparing estimates obtained using the model against measured power for different users.

For brevity, in Fig. 9, we show results for the first three users. Readers are referred to Appendix A of [12] for the rest of graphs.

Figure 9 shows that the two-hidden-layer performance is better than the one-hidden-layer for the three users. This is true for all the users as depicted in Appendix A of [12].

Additionally, we developed user-level power models for each user using his/her power samples only. The samples are divided randomly into a training dataset (70%), validation dataset (15%), and testing dataset (15%). We build the power models using previously identified one-hidden-layer NN architecture and two-hidden-layer architecture. Figure 10 shows the plots of the estimated power (estimated) vs. measured power (measured) for the first three users using a one-hidden-layer NN and a two-hidden-layer NN.

From Fig. 10, the two-hidden-layer NN's performance is slightly better than the one-hidden-layer

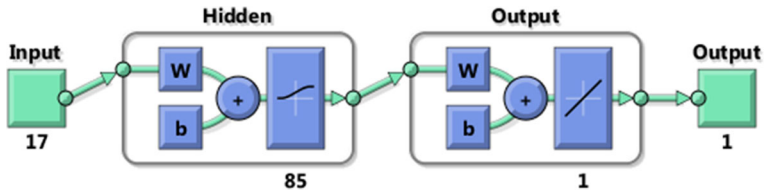


Fig. 6 Selected network structure

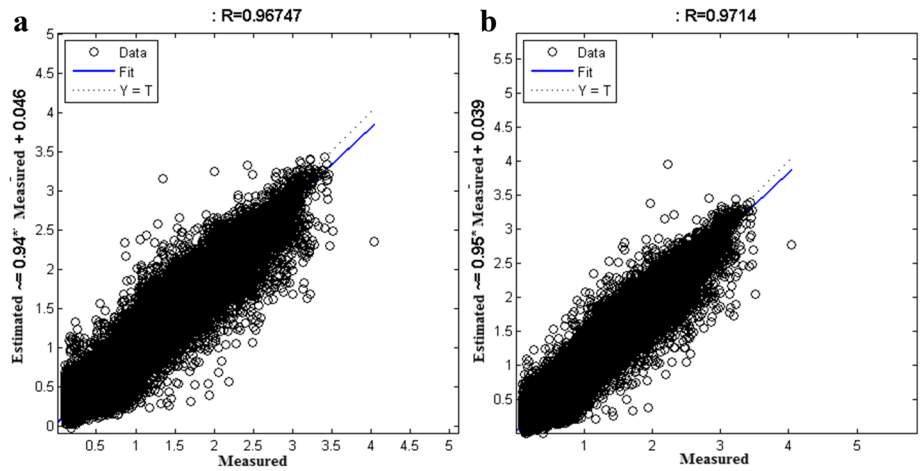


Fig. 8 Performance comparison between one-hidden-layer (**a)** and two-hidden-layer (**b)** neural networks. FFBP neural network structure is used in both **a** and **b**, the used training function is trainbr. In **a,** logsig function is used as transfer function to train the 85 neurons. In **b,** we have two layers of neurons; the first layer consists of 45 neuron with tansig transfer function, while the second layer consists of 25 neurons with logsig transfer function

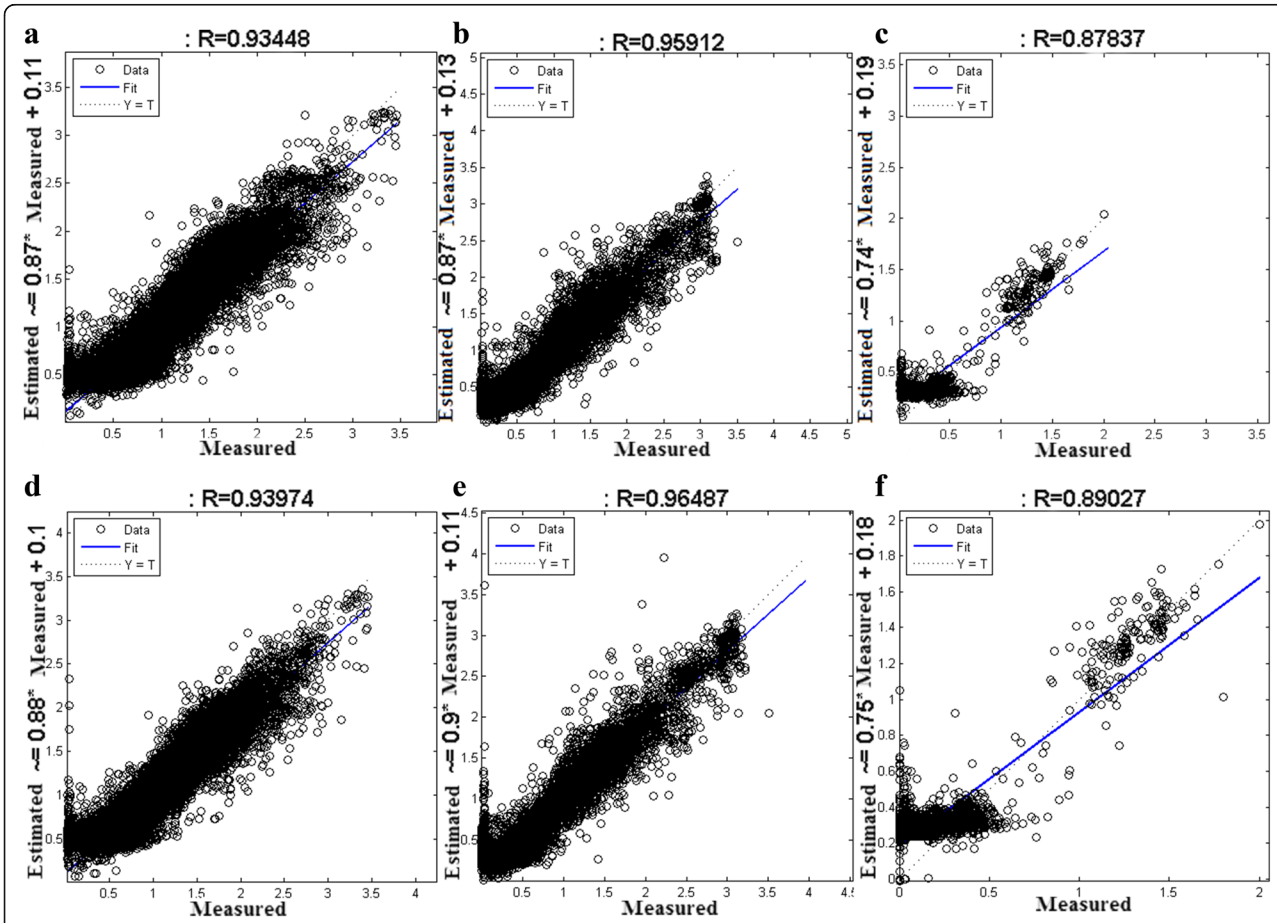


Fig. 9 Device-level performance plots for the first three users. **a**–**c** are one-hidden-layer NN performance for users 1, 2, and 3, respectively. **d**–**f** are two-hidden-layer performance for users 1,2, and 3, respectively

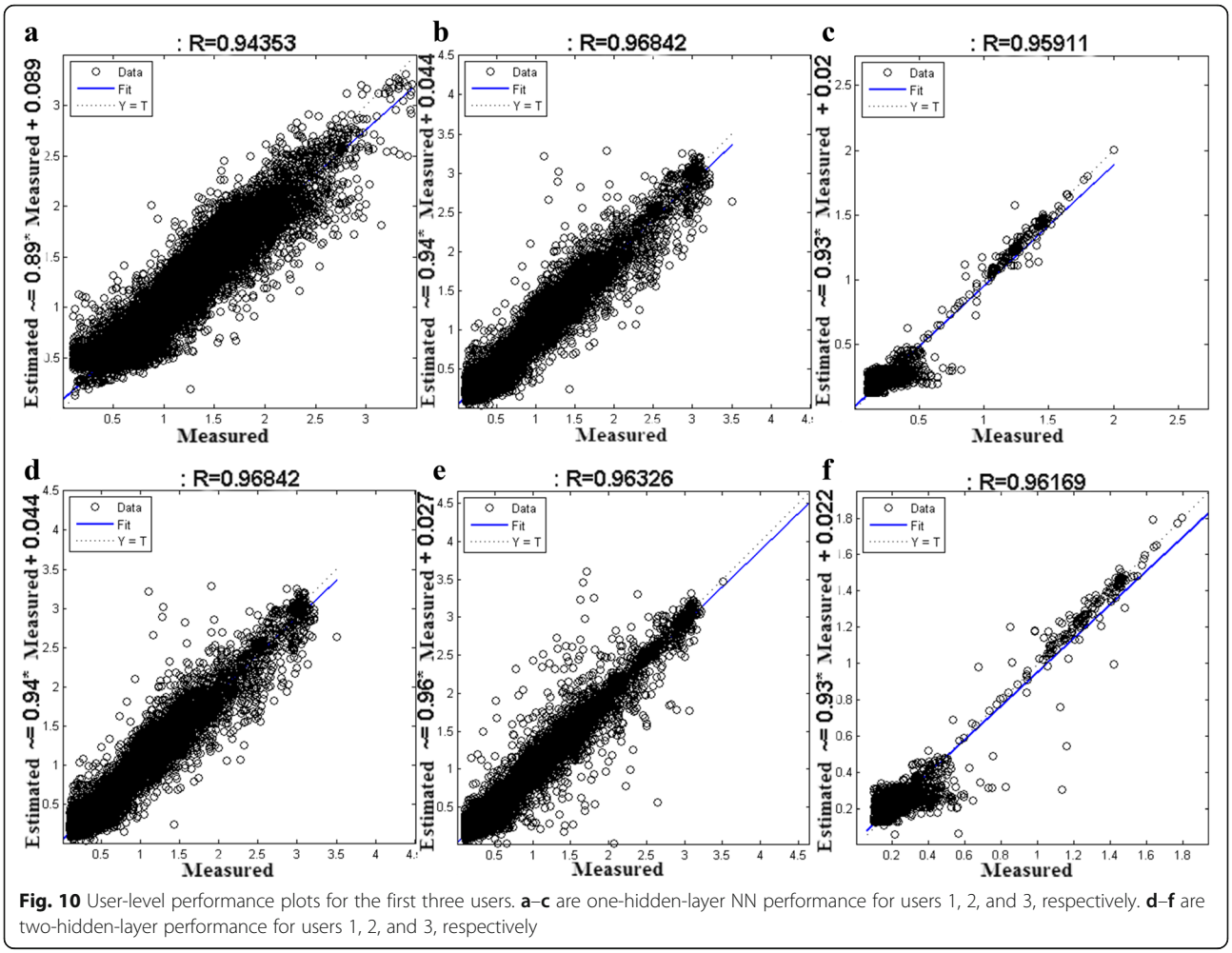


Fig. 10 User-level performance plots for the first three users. **a**–**c** are one-hidden-layer NN performance for users 1, 2, and 3, respectively. **d**–**f** are two-hidden-layer performance for users 1, 2, and 3, respectively

NN for users 1 and 3 (the R value we got when using the two-hidden-layer models is a little higher than when using the one-hidden-layer model). For user 2, the one-hidden-layer NN model performance is better than the two-hidden-layer NN model. Hidden layers help the neural network to recognize more patterns. Choosing the size of the hidden layers can be difficult and most of the time is based on empirical observations. In this work and from the results shown in Figs. 9 and 10, we can conclude that the two-hidden-layer NN power models are more accurate but require more training time. On the other hand, one-hidden-layer NN models have accuracy that is comparable to the two-hidden-layer NN models but with less training time.

Comparing the plots in Figs. 9 and 10, we can say that user-level power models have higher accuracy than device-level power models. This is expected since in user-level power models, we used only the user data in order to construct separate model for each individual user of the device, while in device-level power models, we construct the model using data from all users and test it on each user individually. Device-level power model is a general model that can be used to model power consumption behavior of all users included in the dataset and can be generalized to model power consumption behavior of any user of the device.

4 Conclusions

The power consumed by a smartphone is highly influenced by end-user usage patterns and interest. Battery energy life is very dependent on the nature of applications running on the device and other activities invoked by the user. In this work, we described an attempt to model smartphone power using input parameters that are derived from power-related data that is in turn collected in real time when the devices were on use. We were able to study the feasibility of using neural network techniques in generating reliable power models.

From the results, we can conclude that the two-hidden-layer NN power models are the most accurate models, but they require more training time. It also observed that one-hidden-layer NN models have accuracy

that is comparable to the two-hidden-layer NN models with less training time. User-level power models are built based on the usage patterns for each user while device-level power models are built based on the usage patterns of all users of a smartphone model. User-level power models perform better than device-level models. This is expected, since only the user data is used to construct and test the model, this data is more representative of the user's behavior than the general data. However, device-level power models are still useful in providing an insight into the consumption characteristics of the device.

Competing interests
The authors declare that they have no competing interests.

Received: 10 October 2015 Accepted: 4 January 2017
Published online: 02 February 2017

References

1. Hossein Falaki et al., "Diversity in smartphone usage," in Proceedings of the 8th international conference on Mobile systems, applications, and services, San Francisco, CA, USA, 2010, pp. 179–194
2. Qiang Xu et al., "Identifying diverse usage behaviors of smartphone apps," in Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference, Berlin, Germany, 2011, pp. 329–344
3. Jaymin Lee, Hyunwoo Joe, and Hyungshin Kim, "Smart phone power model generation using use pattern analysis," in IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 2012, pp. 412–413
4. Alawnah, S. and Sagahyroon, A., "Modeling smartphone power," in Proceedings of IEEE EUROCON Conference, Zagreb, Croatia, 2013, pp. 369–374
5. Alex Shye, Benjamin Scholbrock, and Gokhan Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," in Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, 2009, pp. 168–178
6. Lide Zhang et al., "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, Scottsdale, AZ, USA, 2010, pp. 105–114
7. Rajesh Palit, Ajit Singh, and Kshirasagar Naik, "Modeling the energy cost of applications on portable wireless devices," in Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems, Vancouver, Canada, 2008, pp. 346–353
8. Mian Dong and Lin Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in Proceedings of the 9th international conference on Mobile systems, applications, and services, Bethesda, MD, USA, 2011, pp. 335–348
9. Abdulhakim Abogharaf, Rajesh Palit, Kshirasagar Naik, and Ajit Singh, "A methodology for energy performance testing of smartphone applications," in 7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland, 2012, pp. 110–116
10. A Naci et al., Adaptive and flexible smartphone power modeling. Journal of Mobile Networks and Applications **18**, 5 (2013)
11. TI. (2013, March) BQ27520-g3 Fuel gauge with integrated LDO. [Online]. http://www.ti.com/product/bq27520-g3. Accessed Mar 2013
12. Sameer Alawnah, "Modeling smartphone power", MSc thesis, Computer Engineering Program, American University of Sharjah, 2013; available at: https://dspace.aus.edu. Accessed Mar 2013
13. WS McCulloch, W Pitts, A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics **5**, 115–133 (1943)
14. MT Hagan, HB Demuth, MH Beale, *Neural network design* (Pws Pub, Boston, 1996)
15. A. Suissa, O. Romain, J. Denoulet, K. Hachicha, and P. Garda, "Empirical method based on neural networks for analog power modeling", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume: 29, Issue: 5, 2010, pp. 839–844
16. M Carolin, E Fernandez, Estimation of energy yield from wind farms using artificial neural networks. IEEE Transactions on Energy Conversion **24**, 2 (2009)
17. Nwulu, N.I., and Agboola A., "Modeling and predicting electricity consumption using artificial neural networks", Proceeding of the 11th Intl. Conference on Environment and Electrical Engineering, 2012.
18. K Youngseo et al., Artificial neural network model of SOS-MOSFETs based on dynamic large-signal measurements. IEEE Transactions on Microwave Theory and Techniques **62**, 3 (2014)
19. Alawnah, S., and Sagahyroon, A, "Smartphones power", in Proceedings of the International Conference on Electrical and Information Technologies, Marrakech, Morocco, 2015
20. J Tu, Advantages and disadvantages of using ANN versus logic regression for predicting medical outcomes. Journal of Clinical Epidemiology **94**, 11 (1996)