Detecting Abnormal Start-Ups, Unusual Resource Consumptions of the Smart Phone: A Deep Learning Approach

ZHENG Xiaoqing[1], LU Yaping[2], PENG Haoyuan[1], FENG Jiangtao[1], ZHOU Yi[1], JIANG Min[2], MA Li[2], ZHANG Ji[2], and JI Jie[2]

(1. School of Computer Science, Fudan University, Shanghai 201203, China;
2. Software R&D Center/Terminal Business Division, ZTE Corporation, Shanghai 201203, China)

Abstract: The temporal distance between events conveys information essential for many time series tasks such as speech recognition and rhythm detection. While traditional models such as hidden Markov models (HMMs) and discrete symbolic grammars tend to discard such information, recurrent neural networks (RNNs) can in principle learn to make use of it. As an advanced variant of RNNs, long short-term memory (LSTM) has an alternative (arguably better) mechanism for bridging long time lags. We propose a couple of deep neural network-based models to detect abnormal start-ups, unusual CPU and memory consumptions of the application processes running on smart phones. Experiment results showed that the proposed neural networks achieve remarkable performance at some reasonable computational cost. The speed advantage of neural networks makes them even more competitive for the applications requiring real-time response, offering the proposed models the potential for practical systems.

Keywords: deep learning; time series analysis; convolutional neural network; RNN

DOI: 10.12142/ZTECOM.201902006

http://kns.cnki.net/kcms/detail/34.1294.TN.20190529.0953.002.html, published online May 29, 2019

Manuscript received: 2018-01-17

1 Introduction

Deep learning algorithm emerged as a successful machine learning technique a few years ago. With the deep architectures, it became possible to learn high - level (compact) representations, each of which combines features at lower levels in an exponential and hierarchical way [1]–[3]. A stack of representation layers, learned from the data in order to optimize the given objective, make deep neural networks gain advantages such as generalization to unknown examples [4], discovering disentangling factors of variation and sharing learned representations among multiple tasks [5]. The recent successes of the deep convolutional neural networks (CNNs) are mainly based on such ability to learn hierarchical representation for spatial data [6]. For modeling temporal data, the recent resurgence of recurrent neural networks (RNN) has led to remarkable advances [6]–[11]. Unlike the spatial data, learning both hierarchical and temporal representation is one of the long-standing challenges for RNNs in spite of the fact that hierarchical structures naturally exist in many temporal data [12]–[15].

Forecasting future values of the observed time series in fact plays an important role in nearly all fields of science and engineering, such as economics, finance, business intelligence, and industrial applications. There has been extensive research on using machine learning techniques for time-series forecasting. Several machine learning algorithms were presented to tackle time series forecasting problem, such as multilayer perceptron, Bayesian neural networks, K-nearest neighbor regression, support vector regression, and Gaussian processes [16]. The effectiveness of local learning techniques is explored for dealing with temporal data [17]. In this study, we tried to detect abnormal start-ups, unusual CPU and memory consumptions of the application processes running on smart phones using RNNs, which falls in line with the recent efforts to analyze time series data in order to extract meaningful statistics and other characteristics of the data with the deep learning approach.

The paper is organized as follows. First, we give an overview of the research goals, and then try to convey an intuition of the key ideas in Section 2. Section 3 presents RNN-based models to detect unusual CPU, memory consumptions and abnormal start-ups of the application process running on the smart phones. Section 4 reports results of a number of experiments on real-life devices and shows the effectiveness of the proposed models. The conclusion and future work are summarized in Section 5.

2 Background

Recurrent neural networks are a class of artificial neural networks that possess internal state or short-term memory due to recurrent feed-back connections that make them suitable for dealing with sequential tasks, such as speech recognition, prediction and generation [18]–[20]. Traditional RNNs trained with stochastic gradient-descent (SGD) have difficulty learning long-term dependencies (i.e. spanning more than ten time-steps lag) encoded in the input sequences due to vanishing gradient [21]. This problem has been partly addressed by using a specially designed neuron structure, or cell, in long short-term memory (LSTM) networks [21], [22] that keeps constant backward flow in the error signal; second-order optimization methods [23] preserve the gradients by approximating their curvature; or using informed random initialization [24] which allows for training the networks with momentum and stochastic gradient-descent only.

In conventional LSTM each gate receives connections from the input units and the outputs of all cells, but there is no direct connection from the Constant Error Carrousel (CEC) it is supposed to control. All it can observe directly is the cell output, which is close to zero as long as the output gate is closed. The same problem occurs for multiple cells in a memory block: when the output gate is closed none of the gates has access to the CECs they control. The resulting lack of essential information may harm network performance. Gers, Schraudolph, and Schmidhuber [25] suggested adding weighted "peephole" connections from the CEC to the gates of the same memory block. The peephole connections allow all gates to inspect the current cell state even when the output gate is closed. The information can be essential for finding well-working network solutions. During learning no error signals are propagated back from gates via peephole connections to the CEC. Peephole connections are treated like regular connections to gates except for updating timing.

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural networks (a variant of LSTM), introduced by Cho et al. [9]. Their performance on polyphonic music modeling and speech signal modeling was found to be similar to that of long short-term memory. GRUs have been shown to exhibit better performance on smaller datasets because they have fewer parameters than LSTM, as they lack an output gate. There are several variations on the full gated units, with gating done using the previous hidden state and the bias in different combinations.

3 Deep Neural Network-Based Models

Two deep neural network-based models will be described in this section. Both tasks mentioned above are time series forecasting, which uses a model to predict future values based on previously observed values. RNNs and their variants are used to perform those tasks since RNNs can recognize patterns that are defined by temporal distance.

3.1 Unusual CPU and Memory Consumption Detection

The proposed unusual CPU and memory consumption detection model aims at detecting unusual CPU and memory consumption of an application from its resource consumption series and runtime policy in a given time period. Model analyzes the resource consumption data of the application in the specified time period, including a series of its CPU consumption, its memory usage, as well as its runtime policy, and then outputs the probabilities of the existence of unusual CPU consumption and unusual memory consumption if there are any. Since it is a classification problem with time series analysis, we proposed a detection model based on LSTM network to model the time series, and followed by a 3-layer neural network to perform the classification.

3.1.1 Problem Formalization

The proposed model consists of two parts: unusual CPU consumption detection model and unusual memory consumption detection model. For unusual CPU consumption detection model, given an input time series $X=\left(X_1,X_2,\dots X_t,\dots,X_T\right)$, where X_t represents a sampling point including the current CPU consumption and runtime policy of an application and T is the length of the input series, the model analyzes the time series, predicts the probability of the existence of unusual resource consumption, and finally assigns the input into one of the two classes: unusual resource consumption or non-unusual resource consumption. Unusual memory consumption detection model has the same structure as the unusual CPU consumption detection model, but the sampling points X_t in its input series $X=\left(X_1,X_2,\dots X_t,\dots,X_T\right)$ only includes the current memory consumption data.

3.1.2 Normalization of Input Series

At each sampling point, the current CPU consumption is expressed as a percentage, ranging from 0 to 100, and the current memory consumption ranges from 0 to 10^6 KB. These values are all positive and too large for a neural network based model. Therefore, we introduced a normalization step before the proposed model processes the input data. We calculated the average and standard variance of the CPU consumption $(avg_{cpu},stdv_{cpu})$ and the memory consumption $(avg_{mem},stdv_{mem})$ of all the sampling points in the training set, and then normalized the resource consumption values in each sampling point by subtracting the average from the original value, and then dividing the result by the standard variance:

$$X_t[cpu] \leftarrow \left(X_t[cpu] - avg_{cpu}\right)/stdv_{cpu}, \tag{1}$$

$$X_t[mem] \leftarrow \left(X_t[mem] - avg_{mem}\right)/stdv_{mem}. \tag{2}$$

3.1.3 Time Series Modelling

For precisely detecting the unusual resource consumption, the amount of consumption is important, as well as the order and trends of changes in the input series. As **Fig. 1** shows, in order to leverage such information, we apply a bi-directional LSTM network to model the input series and generate a vector representation for it, including a forward LSTM and a backward LSTM.

Both LSTM networks contain a cell state as the representation of its input series, which is initialized as a zero vector before it starts to read the input series (i.e., $C_0 = 0$). The forward LSTM reads the input series X from its left to its right. When it reads a sampling point X_t at timestep t, it first computes its forget gate to decide what information should be forgotten:

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, X_t\right] + b_f\right), \tag{3}$$

where h_{t-1} is the output of the last timestep, and $\sigma(\cdot)$ represents the sigmoid function. Then, it computes its input gate to decide what information should be added into its current cell state:

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, X_t\right] + b_i\right). \tag{4}$$

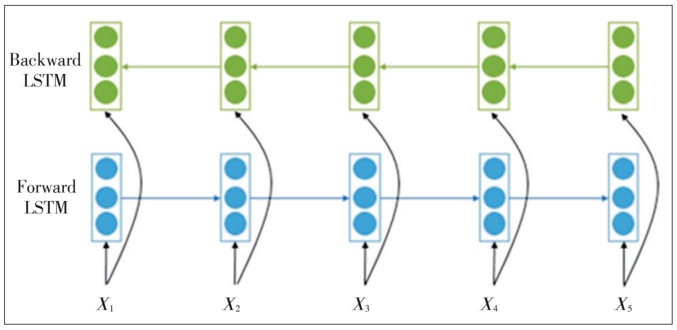
Then, it updates its cell state and calculates its output at the current timestep:

$$\tilde{C}_t = tanh\left(W_C \cdot \left[h_{t-1}, X_t\right] + b_C\right), \tag{5}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{6}$$

$$h_t = \sigma\left(W_o \cdot \left[h_{t-1}, X_t\right] + b_o\right) * tanh\left(C_t\right). \tag{7}$$

By continuously updating the cell state, after the LSTM reads all the sampling points, the cell state vector and the output vector at the last timestep contain the information of the whole input series. Notice that an LSTM only reads the input series from one direction, resulting that the earlier inputs usually have less impact on the final cell state and the output, we adopted a backward LSTM to read the input series from its right to its left, and concatenate the final output vector from the forward LSTM and backward LSTM as the vector representation of the input series:



▲**Figure 1. The bi-directional LSTM networks used in the model.**

$$r = \left[h_T^{fw}, h_T^{bw}\right]. \tag{8}$$

We applied a 3-layer neural network to compute the probability of the existence of unusual consumption. Preliminary experiments show that 3-layer neural network achieved a good trade-off between training speed and performance. The network takes the vector representation of the input series as input, and outputs a 2-dimension vector $o = \left(o_1, o_2\right)$ by a softmax layer, where o_1 represents the probability of the existence of unusual consumption, and o_2 represents the probability of the normal case:

$$o = softmax\left(U \cdot \sigma\left(W \cdot r + b\right) + d\right). \tag{9}$$

We trained our model by minimizing the negative log-likelihood of all the samples of the training set, and the model parameters are optimized by Adam [26], with hyper-parameters recommended by the authors (i.e., learning rate = 0.001, β_1 = 0.9, β_2 = 0.999).

3.2 Abnormal Start-Up Prediction

To save the limited resource of a smart phone, a possible way is to stop application processes when they are abnormally started. By predicting whether a start-up of an application process is abnormal, memory and computational resources could be allocated more efficiently, which further optimizes the performance of the smart phone. The start-up prediction problem can be viewed as a binary classification problem. We proposed a hybrid system, consisting of a rule-based model and a deep learning-based model.

In this section, we first formalize the prediction problem and introduce a set of rules to label all start-ups with either NORMAL or ABNORMAL tag by leveraging the full information contained in the given data set. Then a hybrid system is constructed to solve the prediction problem. Finally, the deep learning model is described in more details.

3.2.1 Problem Formalization

The abnormal start-up prediction models aim at predicting the probability whether a start-up event or *am_proc_start* event is abnormal given its previous logs $\{s_1, \cdots, s_i\}$, where each log s consists of an event name $e \in V_E$, a timestamp t and an argument list a. The used event names and their arguments are listed in **Table 1**. The vocabulary of the package and that of the component are denoted by V_P and V_C. The start type vocabulary is denoted by V_{ST}, in which *ACTIVITY* is a special type indicating that this start-up is brought up by an user. To sum it up, the log data set is defined by $D = \{(s_i, y_i)|s_i = \{s_{ij}\}, s_{ij} = (e_{ij}, t_{ij}, a_{ij})\}$.

However, in the raw data set, namely $D_{raw} = \{s_i\}$, the labels are not given. Thus before defining a machine learning model, the data should be labeled based on human knowledge. One possible way is to label those data manually, which takes a lot of human efforts as well as other resources. Alternatively, we proposed to use a set of rules based on human knowledge or engineer experience to label the data automatically, which is very cost-effective.

The data can be labeled with six steps or six rules from *R1* to *R6* (they are omitted here for security reason). Following those rules, the raw data are tagged with binary labels. The rules can be divided into two parts according to whether a rule can be used in the inference. The first part consists of rule {*R*1, *R*2, *R*3}, which infers the label with previous logs without any future information. The rule 4 labels the data with future information, while these data are not available in the usage. Thus the rule *R*4 as well as the rules with lower priority form the second part. The first part can be tackled with trivial rules, namely rule {*R*1, *R*2, *R*3}, while the second part should be determined by a probabilistic machine learning model.

3.2.2 Hybrid Model

As shown in the previous section, the abnormal start-up prediction is defined as a binary classification problem, aiming at predicting whether a start-up of an application process is abnormal with a given set of logs. We proposed a hybrid model to solve this problem.

The model consists of two parts, a rule-based part and a deep learning-based part. The targeted start-up, as well as the previous logs, is first fed into the rule-based model, which is defined by the rule {*R*1, *R*2, *R*3}, and then generates one of the three possible results, namely *NORMAL*, *ABNORMAL*, and *UNDETERMINED*. The first two are deterministic ones, and will be directly outputted by the hybrid model. The last result indicates that the rule-based model is incapable of predicting the results, because it is determined by {*R*4, *R*5, *R*6}. Therefore, such *UNDETERMINED* data will be further fed into the deep learning model, and the deep learning model will output a probability, reflecting the likelihood of whether the start-up is *NORMAL*.

▼Table 1. Events and arguments

Event	Argument list
am_proc_start	Package; component; start type
am_proc_died	package

3.2.3 Deep Learning Model

The deep learning model aims at providing the probability of a *NORMAL* start-up which cannot be determined by rules based on previous logs. In this situation, the deep learning model is designed to predict whether the process about to start-up will die in the future, namely in one second.

Unlike the rule-based model, the deep learning model cannot map a log to a predicted class directly. Because the deep learning model is capable of fitting data and logs without preprocessing, it could be full of noise, which in turn may result in a bad performance since noise could also be learned by the deep learning model. To reduce such noise in the input of the deep learning model, a feature extraction component is introduced to compute a set of features from the original logs by leveraging human priority. The necessity of these features are demonstrated by preliminary experiments. There are five features extracted listed in **Table 2**.

There are three layers in the neural networks. The first is the five parallel embedding layer, each of which transforms a respective feature into its dense vector representation. Then these vectors are combined with an addition operation, and this layer is designed to combine these parallel features into a joint feature. Finally, we use a logistic regression with this joint feature as its input to estimate the probability of whether the start-up is *NORMAL*. The training parameters are defined as all the embedding layers and the parameters of logistic regression. In this training process, the cross-entropy is used to compute the loss function. To learn the parameter of the deep learning model, Adam optimizer is used.

4 Experiments

We conducted experiments on both tasks to evaluate our model. In the following section, we will first describe the datasets we used and then show the performance of the proposed model on both tasks.

4.1 Data Sets and Preprocessing

For unusual CPU and memory consumption detection task, we prepared a dataset including 992 896 series of resource con-

▼Table 2. Features extracted from original logs to be inputted into the deep learning model

Features	Comments
Event	*am_proc_start* / *am_proc_died*
Package	which package to start
Component	which component to start
Start type	the startup mode
*R*5	whether *R*5 is satisfied

sumption in the training set, and 653 233 series in the testing set. The length of these series ranges from 1 to 12, and the average length is about 9.

Unusual CPU consumption occurs in 2.0% of the series in the training set, and unusual memory consumption occurs in 1.6% of the series in the training set. The proportion of the positive and negative samples are too large. In order to prevent our model from tilting, we used a sample strategy in the training process to ensure that the model learn an equal number of positive and negative samples. During testing, we normalized the input series with the average and standard variance calculated from the training set.

4.2 Unusual CPU and Memory Consumption Detection

We trained our model on an Nvidia GPU card. The final hyper-parameter configuration is listed in **Table 3**.

We evaluated the performance of our model from three perspectives: the precision, the recall, and the F1-score:

$$precision = \frac{Number\ of\ all\ detected\ unusual\ consumptions}{Number\ of\ all\ cases\ the\ model\ reports\ as\ unusual\ consumption}, \quad (10)$$

$$recall = \frac{Number\ of\ all\ detected\ unusual\ consumptions}{Number\ of\ unusual\ consumptions\ in\ the\ test\ set}, \quad (11)$$

$$F1 - score = \frac{2*precision*recall}{precision + recall}. \quad (12)$$

The performance results of the unusual CPU and memory consumption detection models are shown in **Table 4**.

Experimental results illustrate that our proposed model has achieved a high performance on the testing set (over 90% on all aspects), proving the effectiveness of applying LSTM networks into time series analysis problems in improving the performance of smart phones.

4.3 Abnormal Start-up Prediction

We collected logs generated by smart phones, and labelled the logs with rules defined in Section 3.2.1. **Table 5** shows how many logs are determined by each rule. Please note that future information is involved in {$R4$, $R5$, $R6$}, and the logs labelled by these rules are related to the deep learning model. The labelled data set is split into a training set and a test one with a ratio of 4:1.

The deep learning model is implemented with Tensorflow. We set the embedding dimensionality of all features to 20 and the size of mini-batch to 64. The hyper-parameters for Adam optimizer were set to their default values. The results of our experiment is listed below in **Table 6**. It shows that our model demonstrated extremely well with high performance on the machine labelled data set.

▼**Table 3. Model hyper-parameter configuration**

Hyper-parameter	Value
Size of the LSTM output	2 × 50
Size of the hidden layer in classifier	50
Learning rate	0.001
Batch size	32

▼**Table 4. Performance of unusual CPU and memory consumption detection model**

	CPU	Memory
Precision	0.963	0.972
Recall	0.983	0.998
F1-score	0.973	0.985

5 Conclusions

In this paper, we have described a deep neural network-based model for detecting abnormal application start-ups, and unusual CPU and memory consumptions of the application processes running on Android systems. A variant of recurrent neural network architecture with multiple layers was implemented and tested systematically. The experiment results showed that the proposed neural networks performed reasonably well on the two tasks, offering the potential of the proposed networks for practical time serious analyzing and other similar tasks. The number of parameters in neural network-based models is usually much less than other competitors, such as conditional random fields (CRFs). Besides, only simple four-arithmetic operations are required to run the neural network-based models after the models are well trained. So neural network-based models are clearly run considerably faster and require much less memory than that of other models. The speed advantage of those neural networks makes them even more competitive for the applications requiring real-time response, especially for the applications deployed on the smart phones.

▼**Table 5. The statistics of the labelled logs**

Rules	Number of logs labelled by each rule
$R1$	95 138
$R2$	269 348
$R3$	390 203
$R4$	19 557
$R5$	82 912
$R6$	207 881
$R4$, $R5$, $R6$	310 350
Total	1 065 039

▼**Table 6. The performance of our models**

Data set	Accuracy (%)
Deep learning model (for the part with future information involved)	98.93
Hybrid model	99.69

References

[1] BENGIO Y. Learning Deep Architectures for AI [J]. Foundations and Trends in Machine Learning, 2009, 2(1): 1–127. DOI: 10.1561/2200000006
[2] LECUN Y, BENGIO Y, HINTON G. Deep Learning [J]. Nature, 2015, 521 (7553): 436–444. DOI: 10.1038/nature14539
[3] SCHMIDHUBER J. Deep Learning in Neural Networks: An Overview [J]. Neural Networks, 2015, 61: 85–117. DOI: 10.1016/j.neunet.2014.09.003
[4] HOFFMAN J, TZENG E, DONAHUE J, et al. One-Shot Adaptation of Supervised Deep Convolutional Models [EB/OL]. (2013-12-21)[2018-04-15]. http://arxiv.org/abs/1312.6204
[5] KINGMA D P, WELLING M. Auto-Encoding Variational Bayes [EB/OL]. (2013-12-20)[2018-04-15]. https://arxiv.org/abs/1312.6114
[6] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. ImageNet Classification with Deep Convolutional Neural Networks [J]. Communications of the ACM, 2017, 60 (6): 84–90. DOI: 10.1145/3065386
[7] MIKOLOV T, SUTSKEVER I, DEORAS A et al. Subword Language Modelling with Neural Networks [EB/OL]. (2012) [2018-04-15]. http://www.fit.vutbr.cz/~imikolov/rnnlm/char.pdf
[8] Graves A. Generating Sequences With Recurrent Neural Networks [EB/OL]. (2013-08-04) [2018-04-15]. https://arxiv.org/abs/1308.0850
[9] CHO K, MERRIENBOER B van, GULCEHRE C, et al. Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation [C]// Conference on Empirical Methods in Natural Language Processing. Doha, Qatar, 2014. DOI: 10.3115/v1/D14-1179
[10] SUTSKEVER I, VINYALS O L, Le Q V. Sequence to Sequence Learning with Neural Networks [M]//Advances in Neural Information Processing Systems. Cambridge, USA: The MIT Press, 2014: 3104–3112
[11] VINYALS O, TOSHEV A, BENGIO S, et al. Show and Tell: A Neural Image Caption Generator [C]//IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Boston, USA, 2015: 3156–3164. DOI: 10.1109/CVPR.2015.7298935
[12] MOZER M C. Induction of Multiscale Temporal Structure [C]//Proc. 4th International Conference on Neural Information Processing Systems. San Francisco, USA: Morgan Kaufmann Publishers Inc., 1991: 275–282.
[13] HIHI S E, BENGIO Y. Hierarchical Recurrent Neural Networks for Long-Term Dependencies [C]//Proc. 8th International Conference on Neural Information Processing Systems. Cambridge, USA: MIT Press, 1995: 493–499
[14] LIN T, HORNE B G, TINO P, et al. Learning Long-Term Dependencies in NARX Recurrent Neural Networks [J]. IEEE Transactions on Neural Networks, 1996, 7(6): 1329–1338. DOI: 10.1109/72.548162
[15] KOUTNÍK J, GREFF K, GOMEZ F, et al. A Clockwork RNN [C]//31st International Conference on Machine Learning. Beijing, China, 2014: 1863–1871
[16] AHMED N K, ATIYA A F, GAYAR N E, et al. An Empirical Comparison of Machine Learning Models for Time Series Forecasting [J]. Econometric Reviews, 2010, 29(5/6): 594–621. DOI: 10.1080/07474938.2010.481556
[17] BONTEMPI G, BEN TAIEB S, LE BORGNE Y A. Machine Learning Strategies for Time Series Forecasting [M]//BONTEMPI G, BEN TAIEB S, LE BORGNE Y A. eds. Business Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013: 62–77. DOI:10.1007/978-3-642-36318-4_3
[18] ROBINSON A J, FALLSIDE F. The Utility Driven Dynamic Error Propagation Network: CUED/FINFENG/TR.1 [R]. Cambridge, UK: Cambridge University, Engineering Department, 1987
[19] WERBOS P J. Generalization of Backpropagation with Application to a Recurrent Gas Market Model [J]. Neural Networks, 1988, 1(4): 339– 356. DOI: 10.1016/0893-6080(88)90007-x
[20] Williams R J. Complexity of Exact Gradient Computation Algorithms for Recurrent Neural Networks: NUCCS-89-27 [R]. Boston USA: Northeastern University, College of Computer Science, 1989
[21] Hochreiter S, Bengio Y, Frasconi P, et al. Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies [M]// Kremer S C, Kolen, J F eds. A Field Guide to Dynamical Recurrent Networks. Hoboken, USA: IEEE Press, 2001
[22] HOCHREITER S, SCHMIDHUBER J. Long Short-Term Memory [J]. Neural Computation, 1997, 9(8): 1735–1780. DOI: 10.1162/neco.1997.9.8.1735
[23] MARTENS J, SUTSKEVER I. Learning Recurrent Neural Networks with Hessian-Free Optimization [C]//28th International Conference on Machine Learning. Bellevue, USA, 2011: 1033–1040
[24] SUTSKEVER I, MARTENS J, DAHL G E, et al. On the Importance of Initialization and Momentum in Deep Learning [C]//30th International Conference on Machine Learning. Atlanta, USA, 2013: 1139–1147
[25] GERS F A, SCHRAUDOLPH N N, SCHMIDHUBER J. Learning Precise Timing with LSTM Recurrent Networks [J]. Journal of Machine Learning Research, 2002, 3:115–143
[26] KINGMA D, Ba J. Adam: A Method for Stochastic Optimization [EB/OL]. (2014-12-22) [2018-04-15] https://arxiv.org/abs/1412.6980

Biographies

ZHENG Xiaoqing (zhengxq@fudan.edu.cn) received the Ph.D. degree in computer science from Zhejiang University, China in 2007. After that, he joined the faculty of School of Computer Science at Fudan University, China. He did research on semantic technology during his stay at the Information Technology Group, Massachusetts Institute of Technology (MIT), USA as an international faculty fellow from 2010 to 2011. His current research interests include natural language processing, deep learning, data analytics and semantic web. He published more than 30 academic papers in various journals and conferences, including ACL, IJCAI, AAAI, WWW, EMNLP, etc.

LU Yaping received his M.S. degree in industrial management engineering from Shanghai Jiao Tong University, China in 1988 and that in computer science from the University of Texas at Arlington, USA in 1996. He was the chief engineer of the North America R&D Center of ZTE's Terminal Business Division from 2015 to 2018. He worked as a senior staff engineer at Motorola Inc from 1997 to 2006, and a solution architect at i2 Technologies from 2006 to 2009 separately before he joined ZTE (USA) in 2010. His research interests include artificial intelligence, operations research, and their applications.

PENG Haoyuan received the M.S. degree in software engineering from Fudan University, China in 2015. He has been doing research on machine learning and natural language processing, supervised by Professor ZHENG Xiaoqing.

FENG Jiangtao received the M.S. degree in software engineering from Fudan University, China in 2019. He has been doing research on machine learning and natural language processing, supervised by Professor ZHENG Xiaoqing.

ZHOU Yi received the B.S. degree in software engineering from Fudan University, China in 2017. He has been doing research on machine learning and natural language processing, supervised by Professor ZHENG Xiaoqing.

JIANG Min was the system software engineer with the Software R&D Center of ZTE's Terminal Business Division. His research areas include Android system, embedded operating system, virtualization technology and machine learning.

MA Li received his B.S. degree from the University of Electronic Science and Technology, China in 1993. He has been working in ZTE Corporation since 1999, where he has held important software development, design and management positions in the Network Division, Technology Center and Mobile Device Division. He is currently working as the chief software engineer in the ZTE Terminal Business Division. His research interests and development field include terminal system design, AI engine deployment and optimization, algorithm R&D, and other core terminal technologies.

ZHANG Ji received the B.S. degrees in microelectronics and software engineering from Xidian University, China in 2013. He is currently a software engineer with ZTE Corporation. His research areas include Android system, embedded operating system, virtualization technology and machine learning.

JI Jie received his master's engineering degree in computer science software engineering from Xidian University, China in 2008. He is currently a software designer in the Research and Development Team of ZTE Corporation. His research interests include wireless communication technology, AI in embedded systems, and smart phone operating system.