Improving Application Launch Performance in Smartphones Using Recurrent Neural Network

Andre Luiz Nunes Martins
Sungkyunkwan University
Suwon, South Korea
andreluiz@skku.edu

Cesar A. V. Duarte
Sungkyunkwan University
Suwon, South Korea
cesar@skku.edu

Jinkyu Jeong
Sungkyunkwan University
Suwon, South Korea
jinkyu@skku.edu

ABSTRACT

Mobile phones became indispensable tools in our lives, with Android being the most used mobile OS. These devices depend on managing application lifecycles to improve launch performance, but the management of application processes is not done in an efficient way. The standard low memory killer, that is responsible for freeing memory, does not consider any user information and it frequently kills applications that are going to be launched. Context-awareness presents several possibilities to make mobile systems more efficient and user-driven in terms of user experience. In this paper, we introduce a context-based launcher using recurrent neural network (RNN), a special branch of neural networks capable of remembering dependencies, taking in consideration not just previous inputs, but also previous outputs, providing high accuracy without any extra sensor context. Our system guarantees that most of the applications are ready to use in the background, substantially improving the launch time enabling a better user experience. Experimental results demonstrate that the novel scheme can reduce application launch latency in a significant manner.

CCS Concepts

• Information systems → Information systems application → Mobile information processing systems.

Keywords

App launch performance; App prediction; Mobile computing; Recurrent Neural Networks.

1. INTRODUCTION

The communication on the go brought by mobile devices has modified the lifestyle of our entire society: it has transformed work routines, how we organize our daily schedules, develop and maintain social ties. The continuous increase of smartphone users in market has led to a massive boost in the number of mobile applications. Reports show that the mobile app stores, Apple App and Google Play, store together more than 4 million apps [20]. With mobile devices being a part of our lives, users expect longer battery life, pervasive internet access, fast response time, and recent contents [8].

ICMLT 2018, May 19--21, 2018, JINAN, China
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6432-4/18/05…$15.00
DOI: https://doi.org/10.1145/3231884.3231897

In Android, the most used mobile OS, the management of the application processes is not done in an efficient way. The low memory killer that is responsible for freeing memory when it reaches a certain threshold is based in the least recently used processes (LRU). The LRU algorithm does not consider any user information and it frequently kills applications that are going to be used. That process directly affects the user experience that will constantly have to deal with long latency in application launches when interacting with the device [19].

In terms of user experience in smartphones, one of the most crucial performance parameters is the application launch time. There are two methods to launch an application in Android OS. The first one, which is defined as cold start, happens when the system needs to create the new application because it is absent in the background. Second, the warm launch, in which the process is resumed because it was already present in the background [6, 12]. The startup time of a mobile application can be significantly changed depending if the app has to be created or simply resumed. Application launch time is characterized as the time between the users touching the screen icon to the point when the activity is displayed on the screen. And this launch time latency is known to affect a lot the user interaction with the mobile apps. Figure 1 presents a simplified schema of the android application lifecycle and transitions between the states.

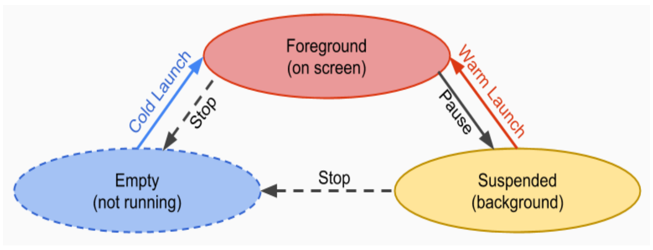

Figure 1: Simplified lifecycle of android application.

To address the challenge of providing best performance we want to manage mobile applications in a context-aware and resource-efficient fashion by harnessing individual application usage behavior. We formulate the problem as series prediction based in contextual information from the user using Recurrent Neural Networks. The prediction model and application preloader guarantee that the application process would be already in the background when needed. That would allow the application to run a warm start with lower latency.

Our experiments are carried out using traces, containing user's app usage, from the RiceLab user study [7, 15]. Substantial application latency reduction was obtained as a result of adopting our novel approach for application prediction and preloading.

The rest of this paper is organized as follows. In Section 2 we analyze previous work related to understanding app prediction, preloading and memory management. Section 3 describes our application prediction model using Recurrent Neural Networks. In section 4 we present our experimental results. In Section 5 we discuss the implementation challenges and limitations. Section 6 concludes this paper with future work.

2. RELATED WORK

Several experimental studies were performed with the goal of building the basis of context-awareness for mobile systems scheduling [5, 16–18, 22]. These projects looked for an analytical understanding of how people interact with applications given a certain context such as temporal and geographical data, the last used applications and many others. Falaki et al. [5] investigates usage traces and tries to label the level of activities that are found to considerably vary among users.

In another previous project, Rice LiveLab user study [17], evaluates 34 people, who received smartphones and mobile data access for 14 months. All along this period, their phone usage was monitored, resulting in a very extensive trace dataset. As discussed in [13], the majority of user interactions with the device are found to be quick. As expected, the effect of sluggish start up can cause even more negative impact for the users considering that 80% of the interactions are shorter than 2 minutes.

Improvements for faster application launch have been studied and implemented for a long time in non-mobile computer systems. Content prefetching and caching to reduce page load times, are commonly used in browsers and other web systems [23]. Recently, there's been an increase in efforts to bring this idea to mobile world. As an initial step for the development of our solution we analyzed the improvement and drawbacks of these related projects, to clearly understand the unfulfilled gaps.

This paper relates to two different areas of prior studies:

2.1 Application Prediction

With the increasingly availability of apps, several different solutions have been proposed for exploiting contextual-awareness in application prediction. One of the biggest challenges are the adaptability of prediction under a cold start and the privacy of the users. In this section we analyze and compare previous works that aim to predict the next application to be used.

The FALCON system contains a context source manager, a launch predictor and a prelaunch event dispatcher, implemented for the Windows Phone OS [23]. FALCON offers an architecture that observes and utilizes context to run predictive app launch actions. The launch predictor, the central component of this implementation, applies user context parameters to predict application start. The launch predictor converts non-treated data sources into relevant features for the selection engine. Then the system infers and figures it out which features it should use along with which application to prelaunch. Formerly, the dispatcher loads the apps and the prelaunch routine is executed for the selected applications based in the information that was passed to it.

FALCON used various context sources such as location, time, accelerometer and gyroscope. The prediction of application to be launched, using context, can achieve latency reductions of around 50% when compared with LRU and launch with fresh app content. These elements are sufficiently low overhead and negligible processing and power consumption. Although presenting good results FALCON implementation was limited to Windows phone and the implementation done to the OS, did not allow much flexibility.

In another study, the Predictive Practical Prefetch (PREPP) [7, 15], is a practical system to prefetch in mobile devices and it have been designed to enhance performance of mobile devices allowing reduction in the startup time. At the core, PREPP relies on two parts:

1. App Prediction by Partial Match (APPM), a prediction system that without use of any privacy-sensitive contextual information grasps the probability distribution of the apps likely to be started after.
2. Time Till Usage (TTU), a model that by measuring the time period before application interaction estimates the correct time for prefetching.

Combining the output of these two individual systems, PREPP is able to decide when to prefetch content, without doing any app or OS modifications. The two systems are complementary and, in the end, able to obtain the following results:

- Over 80% accuracy for the next app to be used (top 5 ranking prediction). Delivering content with 90 seconds freshness on average for users;
- Generates minimum extra energy cost.

In spite of the fact that it presents good results, PREPP focus on prefetching the web content but does not approach the issue and app launch latencies, making clear that there is still room for solutions that tackle the application prediction and management in a broader sense.

Other research modeled the prediction of subsequent apps as a supervised learning problem, achieving a prediction accuracy as high as 90.2% [2]. This study uses log samples with more than 60 million records of anonymous users collected in 2014, obtained from the Yahoo Aviate. This prediction model utilizes Parallel Tree Augmented Naive Bayesian Network (PTAN) with basic and session features. Table 1 presents a summary of the accuracy of the models described.

Table 1. Comparison of prediction accuracy of previous algorithms

Algorithm	Prediction Accuracy
Falcon	70.1%
PREPP	74.3%
PTAN	90.2%

2.2 Mobile Application Management

Prefetching and memory management in mobile devices has received considerable attention lately, and several studies were developed with the goal of improving performance in this area [7, 15]. Following this trend CAS (Context-aware Application Scheduler) is a solution which functions by predicting the application that will be used and when a user will launch it. Simulations, based on usage traces, show that CAS performs better than Android scheduler, Android Low Memory Killer (LMK) and Android 6.0 [14].

CAS focus on consumption aspect of application management and it is shown to outperform two other complementary approaches, BFC (Background to Foreground Correlation) and HUSH [3]. BFC computes the amount of user interaction for each application while they interact with the phone and it is based on suppressing background applications based in a threshold. HUSH places applications in the inactive state in case they are not often used in foreground and elongates the period of the inactive states exponentially. The algorithm is able to obtain screen-off energy saving of more than 15%. CAS adopts the parameter γ to treat both metrics, energy efficiency and fast app launch time, as a unified measure and it has different adjusting points depending on the balance of the parameter γ. The background and the total energy saving compared to the LMK reach 51% and 25% on average but the parameters need to be adjusted in an Ad-hoc manner. These approaches address the energy-inefficient activities, handling the background processes, but this still does not answer when the user will use the application again [9].

Li et al. proposed a reinforcement learning based optimization for deciding which app to kill allowing the system to consequently improve application startup performance. In the low memory killer optimization process, they present an algorithm which learns applying trial-and-error exploration, measuring the reward of each killing process and creating the policy to perform process-killing tasks. The reinforcement learning LMK, makes better choices while choosing the process to be discarded, showing results that outperform the standard low memory killer from Android system [10].

While other approaches have contributed to improve app predictions, most of them focus on app recommendations, killing processes and power saving, not in reducing the application cold starts. Our goal is to attain highest prediction accuracy while exploring a new anticipatory model and make use of that prediction to improve the application launching performance by preloading the applications.

3. PREDICTION MODEL

We define the question of app prediction as following: By analyzing the application usage sequence for a certain user, can we anticipate which application he is going to open after? We establish parallels between our problem and text translation, where the previous word sequence can be used to determine the most probable next character. Similarly, each app can be viewed as a word and the sequence as a long text and apply this machine learning techniques to our problem [4].

3.1 LSTM Model

Recurrent neural networks (RNNs) are models that capture the patterns in sequences via a chain like structure composed by gates. These networks combine the input with their current state and a function to output a new prediction [21]. Since the development of LSTM in 1997, systems based on the architecture and similar to the Gate Recurrent Unit (GRU) have demonstrated unprecedented performance on diverse tasks, such as speech recognition, text-to-speech synthesis, translation, etc. So, we look into these model characteristics to understand how to adapt to our needs [10].

Long Short-Term Memory networks are a particular type of RNNs are extremely efficient for learning long-term dependencies [11]. Its design allows it to adjust and remember patterns that consider long intervals of time. In that manner it handles the long-term dependency problem. Like all other recurrent neural networks, it has the form of a conglomerate of repeating blocks, resulting in an overall simple structure [3]. The main difference manifests in the fact that it has four interacting layers. Figure 2 presents the basic structure of a LSTM network.

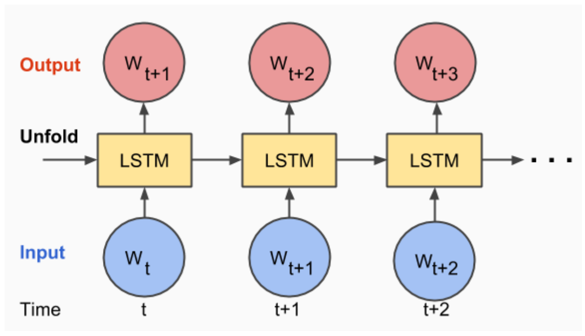

Figure 2: Long Short-Term Memory (LSTM) Network

The core idea behind LSTMs is that it has the astonishing ability to add or discard information for the cell state or memory unit. In this way, LSTMs can selectively remember or forget things and capture the temporal behaviors of a user.

3.2 User Trace Dataset and Training

Our machine learning model utilizes the training datasets provided by the LiveLab project at Rice University [17]. The traces were collected from 34 volunteers during a period of 14 months. The data of the users is composed by the list of applications used in each phone. The traces contain the time and duration of each app session along with phone calls made and received, charging state, accelerometer readings, CPU and disk utilization, periodical cell tower signal state, periodical connection Wi-Fi access point state, time that the logger was running, web browsing history and GPS.

Our LSTM-based prediction system was developed using the TensorFlow framework. TensorFlow is an open source machine learning system that operates at large scale in diverse environments allowing developers to test novel optimizations and training algorithms [1]. The chosen features to feed our network as inputs in the Livelab traces are app id, the time the app is launched, battery status and location. The data is then reshaped to a 3-D array being the first dimension the length of the data itself and the second dimension the number of steps. The system is then trained using 80% of each user's data, which is feed to the system in batches. Figure 3 shows the accuracy of our model during the training process.

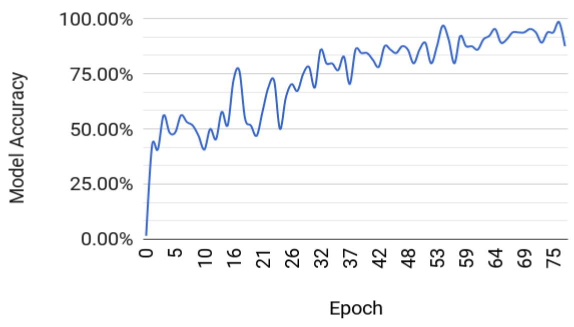

Figure 3: Model Accuracy during each training iteration.

We train our network individually for each one of the 34 users, getting results that exceeded our expectations. The overall accuracy had surpassed 92.8%. Initially we used LSTM only feed the current features as input, which showed poor performance for a TOP-1 prediction, as we change to a Time Series, stacking previous features, where the long-term recurrence is a perfect feature for it, our model, presents prominent results. Our architecture has only 3 layers to avoid overfitting. As described above our model learns the patterns combining the user features and considering the temporal aspect it is used to predict and preload the next application to be used. That allows the app to be ready for launching whenever the user requests it.

4. EXPERIMENTS

In this section, we analyze the advantages of our prediction and preloading approach using the LiveLab traces. The system considers the current context to make a prediction and guarantee that the app is loaded before a user actually invokes the app.

4.1 Testing Environment

We model the application memory consumption and Android Low Memory Killer and develop a comprehensive simulator based on a mobile device with 2GB RAM. Due to the irrelevance of the system processes for this application, we excluded the home screen manager, the phone calls, messaging service and other similar system applications.

The simulation evaluates the execution of different apps on the phone, checking the process creation and changes in the memory usage that are controlled by probability distributions initialized with randomized parameters. The simulation process is simplified in such a way that actions are taken at the point the memory thresholds are reached [10]. The methodology used in our simulation is presented in Algorithm 1.

Algorithm 1 Pseudo code for the simulation methodology

```
function LSTMAppLoader($Trace$)
    for each app in $Trace$ do
        $nextApp \leftarrow predictNextApp(traceFeatures)$
        if $nextApp == NULL$ then
            $break$
        else
            loadNextApp(nextApp)
            appLaunch()
            launchStatistics()
        end if
    end for
end function
function PredictNextApp($nextApp$)
    $predictedApp \leftarrow callLSTMPredictor$
    if $app \subset AppListLMK$ then
        Return $NULL$
    else
        Return $predictedApp$
    end if
end function
function LoadNextApp($nextApp$)
    $freeMemory \leftarrow totalMemory - nextAppMemory$
    if $freeMemory > threshold$ then
        load(nextApp)
        ADD nextApp to AppListLMK
    else
        freeMemory()
        load(nextApp)
        ADD nextApp to AppListLMK
    end if
end function
```

To compare with our system, we evaluate the standard Android LMK using the same traces. The standard LMK operation can be summarized as follows: applications with no service processes are killed at the point which the memory threshold of 192MB is hit and apps that contain service components start to be killed at 144MB memory threshold.

4.2 Results

The average launch latency and number of cold launches are the metrics used to evaluate our model performance. Figure 4 shows the performance of the standard LMK, a second baseline, that corresponds to the LRU LMK, which performs process-kills based in the order the apps were used, and finally our LSTM prediction model.

As we see from the results, our proposed scheme outperforms all the Android LMK policies. All results considered, our approach accomplishes on average a 44% reduction on application launch latency when compared to the standard Android LMK, and 12% in relation to the LRU Low Memory Killer.

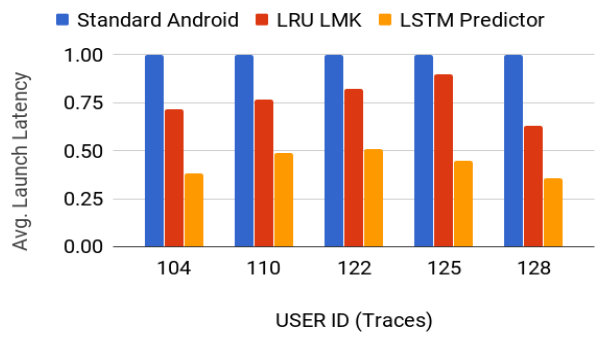

Figure 4: Detail simulation results (5 randomly selected users).

We also measured the number of cold launches that happened while running all the different techniques. The obtained results show that the proposed scheme outperforms the others in all the observed users, obtaining a reduction of 42% in the number of cold launches in relation to the LRU LMK. Less free memory is available when our algorithm runs but that helps in having more applications in the background leading to faster response times.

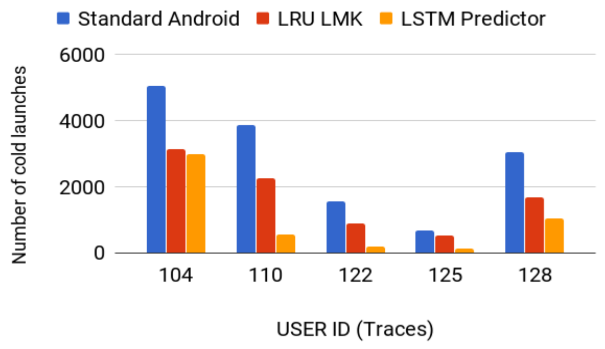

Figure 5: Number of cold launches (5 randomly selected users).

5. DISCUSSION

There are some issues to be resolved before implementing our solution into a real device. The issues are threefold.

First, the application unload or process killing should be avoided in case the application is needed anytime soon. For that our system should not simply consider the next application but possible a batch of application and a window of time in which they are likely to use. That would minimize the loss and avoid the killing of important apps, while guaranteeing good launching times. Second, to load applications and make our solution work natively we need to override the application manager and virtually disable or make Android run with substantially relaxed memory limits.

Finally, the LSTM model is able to reach a really good prediction accuracy, surpassing the current state-of-art. But in this process we do not consider the prediction behavior adaptation in case of new app installations. For that matter a different strategy should be studied and this condition added to our model.

6. CONCLUSION

This research paper proposes a new schema for application prediction and preloading, which utilizes collected information in a smartphone and generates customized app predictions using a recurrent neural network. Based on the data collection analysis, we found that several features such as last application, time and battery level can be used to anticipate the next application to be started and improve the app launch performance, creating a better experience for the user. Our system had a performance that surpassed the performance of other previous models, from the literature, in terms of prediction accuracy and showed significant improvement in the app launch time, reducing the amount of cold starts.

In the future we plan to further develop the Android implementation of our system, considering the effects new application installations, predict application considering the top-k applications to mitigate the effects of misprediction and also measure the battery consumption effects of the prediction and preloading process.

7. ACKNOWLEDGMENTS

This research was supported partly by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2016M3C4A7952587) and partly by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MISP) (NRF-2017R1C1B2007273).

8. REFERENCES

[1] Abadi, M. et al. 2016. TensorFlow: A System for Large-scale Machine Learning. *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, USA, 2016), 265–283.

[2] Baeza-Yates, R. et al. 2015. Predicting The Next App That You Are Going To Use. *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining* (New York, NY, USA, 2015), 285–294.

[3] Chen, X. et al. 2015. Smartphone Background Activities in the Wild. *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking* - MobiCom '15 (2015).

[4] Dietterich, T.G. 2002. Machine Learning for Sequential Data: A Review. *Lecture Notes in Computer Science* (2002) 15–30.

[5] Falaki, H. et al. 2010. Diversity in smartphone usage. *Proceedings of the 8th international conference on Mobile systems, applications, and services* - MobiSys '10 (2010).

[6] Gorman, M. 2004. Understanding the Linux Virtual Memory Manager. Prentice-Hall PTR (2004).

[7] Higgins, B.D. et al. 2012. Informed mobile prefetching. *Proceedings of the 10th international conference on Mobile systems, applications, and services* - MobiSys '12 (2012).

[8] Lee, J. et al. 2017. CAS: Context-Aware Background Application Scheduling in Interactive Mobile Systems. *IEEE Journal on Selected Areas in Communications*. 35, 5 (2017), 1013–1029. DOI=https://doi.org/10.1109/jsac.2017.2676918.

[9] Lee, J. et al. 2016. Context-aware Application Scheduling in Mobile Systems: What Will Users Do and Not Do Next? *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (New York, USA, 2016), 1235–1246.

[10] Li, C. et al. 2017. Optimizing low memory killers for mobile devices using reinforcement learning. *13th International Wireless Communications and Mobile Computing Conference* (IWCMC) (Jun. 2017), 2169–2174.

[11] Lipton, Z.C. et al. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. (2015).

[12] Nagata, K. et al. 2013. Measuring and Improving Application Launching Performance on Android Devices. *First International Symposium on Computing and Networking* (2013).

[13] Nicholson, A.J. and Noble, B.D. 2008. BreadCrumbs: forecasting mobile connectivity. *Proceedings of the 14th ACM international conference on Mobile computing and networking* (Sep. 2008), 46–57.

[14] Optimizing for Doze and App Standby | Android Developers: https://developer.android.com/training/monitoring-device-state/doze-standby.html. Accessed: 2017-11-13.

[15] Parate, A. et al. 2013. Practical prediction and prefetch for faster access to applications on mobile phones. *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing* - UbiComp '13 (2013).

[16] Rahmati, A. et al. 2015. Practical Context Awareness: Measuring and Utilizing the Context Dependency of Mobile Usage. *IEEE Transactions on Mobile Computing*. 14, 9 (2015), 1932–1946.

[17] Shepard, C. et al. 2011. LiveLab. *ACM SIGMETRICS Performance Evaluation Review*. 38, 3 (2011), 15. DOI=https://doi.org/10.1145/1925019.1925023.

[18] Shin, C. et al. 2012. Understanding and prediction of mobile application usage for smart phones. *Proceedings of the 2012 ACM Conference on Ubiquitous Computing* - UbiComp '12 (2012).

[19] Singh, A. et al. 2016. A method to improve application launch performance in Android devices. *International Conference on Internet of Things and Applications* (IOTA) (Jan. 2016), 112–115.

[20] The 2017 Mobile App Market: Statistics, Trends, and Analysis: https://www.business2community.com/mobile-apps/2017-mobile-app-market-statistics-trends-analysis-01750346. Accessed: 2017-11-13.

[21] The Unreasonable Effectiveness of Recurrent Neural Networks: http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Accessed: 2017-11-13.

[22] Verkasalo, H. 2008. Contextual patterns in mobile service usage. Personal and Ubiquitous Computing. 13, 5 (2008), 331–342. DOI=https://doi.org/10.1007/s00779-008-0197-0.

[23] Yan, T. et al. 2012. Fast app launching for mobile devices using predictive user context. *Proceedings of the 10th international conference on Mobile systems, applications, and services* - MobiSys '12 (2012).