# Autoencoder based API Recommendation System for Android Programming

Jinyang Liu
*Academy for Advanced Interdisciplinary Studies*
Peking University
Beijing, China
jyliu.aais@pku.edu.cn

Ye Qiu
School of Electronics Engineering & Computer Science
Peking University
Beijing, China
qiuye2014@pku.edu.cn

Zhiyi Ma
School of Electronics Engineering & Computer Science
Peking University
Beijing, China
mazhiyi@pku.edu.cn

Zhonghai Wu
School of Software and Microelectronics
Peking University
Beijing, China
wuzh@ss.pku.edu.cn

***Abstract*—As a typical example of modern Information Technologies, Android platform and Apps are widely used by smartphone users all over the world. Thus, the research of designing models for assisting programmers in writing Android codes is of great importance and value, and recommending API usages is a stereotype task in this aspect. This paper applies Autoencoder neural networks into the model of API recommendation system for Android programming, and designs new Autoencoder based Android API recommendation system. This paper carries out experiments on the collected Android code dataset and verifies the effectiveness of the newly designed models compared with classical recommendation models.**

***Keywords—Software Engineering, Android, Recommendation System, Autoencoder.***

## I. INTRODUCTION

Nowadays, smartphones are used by people all over the world, and a great portion of them are using smartphone with Android operation system and applications. A huge number of people are working on Android platform to design and develop new applications, however, the work of them is filled with great difficulties: to complete a code file, or even just a code sequence is a hard mission that needs rich knowledge and good comprehension of programming language. Therefore, to design and build computer-based models that make computer help programmers in their work has been a popular research topic. As API usage is a core part in Android programming, this paper makes insight into the task of recommending API usages for programmers in Android programming. The work and achievements of this paper are as following:

This paper applies a new kind of neural network: Autoencoder into the design of Android API Recommendation System. In the experiments the paper shows, the effectiveness of the model is proved.

The rest part of this paper is organized as following: Part II introduces the API recommendation task in detail, and describes the data preprocess method. Part III is the full introduction of the recommendation model raised by this paper. Part IV shows the experiments and its results in terms of the effectiveness of models in this paper, Part V introduces the related works of this paper, and part VI is the conclusion of this paper's work and achievements.

## II. TASK INTRODUCTION AND DATA PREPROCESSING

### A. Task Introduction

This paper focuses on the task of an Android API Recommendation System: Given an uncompleted Android code snippet, the system can extract the API usage information of it, and provide recommendations of other APIs that are likely to be used in the same code file. Fig. 1 is a graphic explanation of this task.

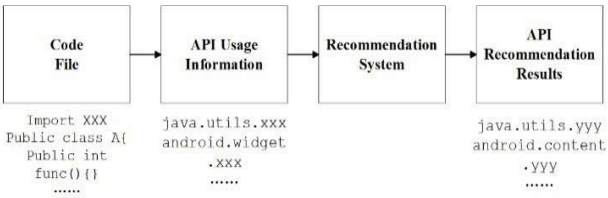


Fig. 1. The task that API Recommendation system executes. The system extracts the API usages from given code file, and generates API list that are recommended to be used in the code file.

### B. Data Preprocessing

As Android codes are written in Java language, the recommendation system acquires the information of API usages of codes from their imports. Fig. 2 shows the imports in an Android code file to be converted to API usage information. After that process, a code file is converted into an API usage list consisted of the APIs used in this code. Each API is given an identical numerical ID, and the code can be represented with a 0-1 vector: each dimension is corresponded to a specific API, and the value of that dimension (0 or 1) represents the usage of the API in this code (not used or used). The size of the vector is the number of all APIs in the database, or the number of APIs

for the recommendation system to learn and generate recommendation results with, because there is always a huge number of APIs in big database, and most of them has extremely low appearing frequency, which means that the Recommendation System can't take them into account. In the rest part of this paper, the 0-1 vector generated from a code file is called a **code vector**.

```
import android.app.Activity;
import android.content.Context;
import android.content.res.Configuration;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.View;
import android.view.ViewGroup;
```

Fig. 2. The imports in Android (Java) codes contain the API usages of this code file.

## III. Autoencoder in Recommendation System

Autoencoder[1] is a kind of neural networks. It is a self-supervised model for extracting features from data, or reconstructing input data from broken data, which is similar to the recommendation task. A classical Autoencoder network has full-connected input and output layers that have the same size, and a full-connected hidden layer with smaller size. When being trained, the model uses same input data and output data. Fig. 3 is the structure of Autoencoder. Sedhain et al. [1] points out that Autoencoder can be used in recommendation system. For example, in the task of this paper, after training a network with the code vectors in code database, the network can doing recommendation work in the following way: Given an input of 0-1 code vector, the output of the network can be regarded as 'Recommendation Score Vector'. In other words, the value of each dimension of the output vector is the recommendation score of the corresponded API. The APIs are ranked by the scores, and APIs with high ranks will be the recommendation results.

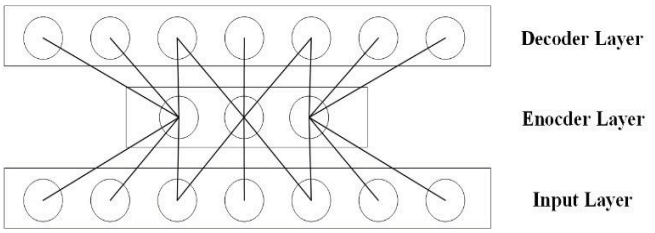


Fig. 3. The network structure of Autoencoder. The Layers are full-connected layers, and only part of the links of units are displayed in the figure.

When training Autoencoder network, same input and output data are used, so vanilla Autoencoder suffers severe over-fitting trouble in practical use. To lighten over-fitting, Denoising Autoencoder (DAE)[2] is raised. Different from vanilla Autoencoder, a noising unit is added to the network, which adds 'noise' to input data when the network is being trained. By training network with noised input and raw output, the network gets the ability of 'Denoising' (remove noise from data), and its robustness and ability of resisting over-fitting is increased. Fig. 4 is the structure graph of DAE. There are several methods to add noise in the noising unit. Considering the practical use for the task, the model in this paper uses 'Mask Noising', that is, for each value in the input vector, the unit set it to 0 with a certain probability (the probability itself is a parameter) separately. It is worth to be mentioned that the Noising Unit only works at the training process, not in the testing process. The work of Li et al. [3] is an example of application of DAE into recommendation system.

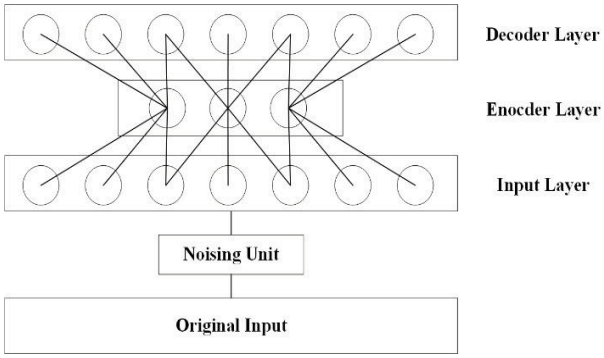


Fig. 4. The structure of DAE (Denoising Autoencoder) network. A noising unit is added between the original input and input layer.

Autoencoder or DAE network only has one hidden layers, which restricts its scale (number of parameters). To build bigger model and extract deeper features, Vincent et al. [4] designs a developed Autoencoder network derived from DAE: Stacked Denoising Autoencoder (SDAE). SDAE is actually a stack of DAE, it has several hidden encoder layers and decoder layers, and has a mirror symmetric network structure. Fig. 5 shows the structure of SDAE. Comparing with DAE, SDAE model has more parameters, and have better feature extracting ability. Its main shortage is the difficulty of its training process.

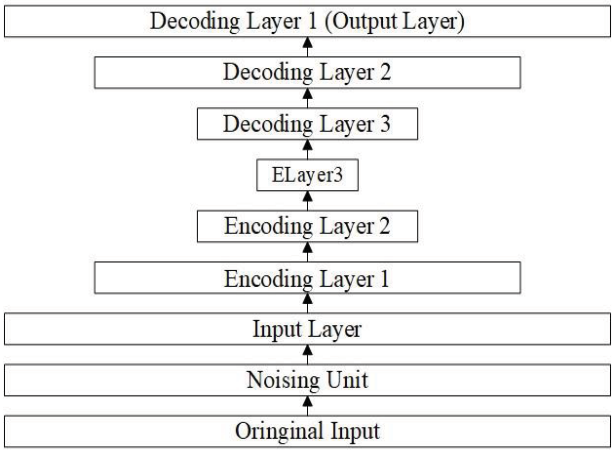


Fig. 5. The network structure of a 3-stacked SDAE (Stacked Denoising Autoencoder) network. The sizes of layers are mirror symmetric.

The full design of the Android API Recommendation System this paper raises is as following: The recommendation model is an Autoencoder type network, trained with code vectors in code database. It is combined with a API usage extraction module to generate input vector for new code file, and a output module that ranks the scores in the output vector of recommendation model and lists out ranked recommendation

results. In the experiments of this paper, 3 kinds of Autoencoder networks (Vanilla Autoencoder, DAE and SDAE) are tested for the Recommendation System.

## IV. Experiments and Analysis

This part introduces the dataset used in this paper for experiments, describes the model configurations and train details, displays and analyzes the experimental results of the recommendation models this paper designs in comparison with classical recommendation models.

### A. Experimental Dataset

The dataset this paper uses and trains models on consists of Android projects and code files from GitHub Java Corpus an open source Java code dataset collected from GitHub, and some other Android projects manually collected from GitHub, then it is split into train set and test set. In the whole dataset of 3584 Android projects, this paper randomly selects 10% of them as test set, and the rest part is used as training set. It is worth to be noted that the work of this paper splits the training and test set by projects instead of code files, because that there are always more similarities between codes inside one certain project, but the model is trying to obtain features from programming language level other than the specificity of projects. The statistical numbers of the database are in Table 1. The model only learns with APIs that appear more than 10 times, and there are total 15689 APIs that meet the requirement.

TABLE I. Statistical Data of Android Code Dataset

| Set | Projects | Code Files |
|---|---|---|
| Train | 3225 | 226110 |
| Test | 359 | 12740 |
| Overall | 3584 | 238850 |

### B. Model Configuration and Train Details

As mentioned before, this paper tests 3 kinds of Autoencoder network for the Android Recommendation System: Vanilla Autoencoder, DAE and SDAE. The size of input and output layers are the same in all networks: 15689, the number of APIs to be recommended. For Vanilla Autoencoder and DAE, hidden layers with size 500, 1000 or 1500 are tested. For SDAE, this paper tests 3-stacked SDAEs (SDAE with 3 Encoder layers and 3 Decoder layers). The sizes of Encoder layers in SDAEs are set to 1000-500-200 or 1000-1000-1000 (Decoder layers are corresponded). For DAE and SDAE, noise rate (the probability of mask single dimension) is set to 0.2, 0.3 or 0.5. All Vanilla Autoencoder and DAE networks are trained for 200 epochs. SDAE networks are first trained layer-by-layer for 200 epochs each(a pair of Encoder layer and Decoder layer are trained together), then the whole networks are trained together for fine-tuning for 200 epochs. All networks are trained with batch size 64 and initial learning rate 0.001, using clip norm of threshold 5. At the 100th and 150th epoch, learning rate is multiplied by 0.1.

For Comparison, this paper also does experiments of classical Collaborative Filtering Model [5, 6]. This paper uses user-based Collaborative Filtering. For the parameter of number of most-similar code files to be used for generating recommendation scores, 30, 50 or 100 is tested.

### C. Evaluation Method

This paper mainly uses recall rate to evaluate the effectiveness of the models. In the test process, for each test case (a code file in the test set), 20% of the API usages in it are randomly chosen and masked, in another word, removed (at least 1 of them will be removed). Then, a model gives recommendation results using the code vector formed by the rest part of API usages. X APIs that have most recommendation scores are chosen as recommendation candidates. X is 10 if the number of removed APIs is less than or equal to 5, or double of the number of removed APIs if it is more than 5. Let $\mathbf{A_i}$ be the set of removed APIs test case i, $\mathbf{B_i}$ be the set of recommendation candidates for test case i, Finally, the recall rate is:

$$recall\ rate = \frac{\sum_{i \in T} |A_i \cap B_i|}{\sum_{i \in T} |A_i|}$$

where T represents the whole test set. As the APIs are randomly masked, the test process will be run 10 times for each model (the whole test set will be iterated for 10 times), and calculate the average recall rate.

### D. Experimental Results and Analysis

The test recall rates for Autoencoder, DAE and SDAE models with different parameters are in Table 2 (Autoencoder), Table 3 (DAE), and Table 4 (SDAE).

TABLE II. Test Recall Rates of Autoencoder Models

| Hidden Layer Size | Recall Rate |
|---|---|
| 500 | 0.3063 |
| 1000 | 0.3160 |
| 1500 | **0.3195** |

TABLE III. Test Recall Rates of DAE Models

| Hidden Layer Size | Noise Rate | Recall Rate |
|---|---|---|
| 500 | 0.2 | 0.7417 |
| 500 | 0.3 | 0.7465 |
| 500 | 0.5 | 0.7422 |
| 1000 | 0.2 | 0.7487 |
| 1000 | 0.3 | **0.7507** |
| 1000 | 0.5 | 0.7411 |
| 1500 | 0.2 | 0.7501 |
| 1500 | 0.3 | 0.7483 |
| 1500 | 0.5 | 0.7385 |

TABLE IV. Test Recall Rates of SDAE Models

| Network Structure | Noise Rate | Recall Rate |
|---|---|---|
| (1000,500,200) | 0.2 | 0.7107 |
| (1000,500,200) | 0.3 | **0.7197** |

| | | |
|---|---|---|
| (1000,500,200) | 0.5 | 0.7132 |
| (1000,1000,1000) | 0.2 | 0.6968 |
| (1000,1000,1000) | 0.3 | 0.7117 |
| (1000,1000,1000) | 0.5 | 0.7176 |

From the experimental results, it is easy to see that noising unit performs a key role for guaranteeing the performance of recommendation model. Without noising unit, a vanilla Autoencoder can only gain 30%+ test recall rate, but an DAE network with just an additional noise unit can provide recall rate of about 75%. Among the Autoencoder models, DAE network with medium size (1000) and noise rate (0.3) performs best on the evaluation by recall rate (75.07%). It is worth to be noticed that, although SDAE networks have more layers and parameters, their performance is no better than DAE networks (the best recall rate among them is 71.97%). It can be explained with the following reasons:

1. When data is transferred through more layers, there is more loss of information of original input data, so using deeper features to reconstruct input data is not probably better than using shallower features with only one Encoder layer.

2. The number of parameters of SDAE is greater than that of DAE, which means that over-fitting problem is more serious in SDAE models.

3. The training difficulty of SDAE with multiple layers is much more than DAE.

Table 5 shows the best recall rates of each type of models, including CF (Collaborative Filtering), Vanilla Autoencoder, DAE and SDAE. In conclusion, DAE and SDAE models both perform better than classical CF recommendation model, and DAE model has the best effectiveness of recommendation. Autoencoder network, especially DAE network, will be good fundamental framework for Android API Recommendation System.

TABLE V. BEST TEST RECALL RATES OF DIFFERENT MODEL TYPES

| Model Type | Best Recall Rate |
|---|---|
| CF | 0.6939 |
| Autoencoder | 0.3195 |
| DAE | **0.7507** |
| SDAE | 0.7197 |

## V. RELATED WORKS

API Recommendation has been a popular research task since years before. [7, 8] are early works of using statistical methods in analysis of API usages and recommendation. Heinemann et al .[9] builds an API recommendation model taking context into account. The work of Thung et al. [10] is an API recommendation model using feature requests (textual descriptions). Asaduzzaman et al. [11] and Li et al. [12] provide works of recommending API parameters. In recent years, there are also many works of applying modern machine learning or deep learning models into this task. Nguyen et al. [13] designs an API recommendation model with Hidden Markov model, and Gu et al. [14] makes use of deep learning models. There are also works with classical statistical methods in recent years, and [12, 15, 16] are some examples.

For the introduction and theory of Autoencoder network, and its application in recommendation system, [2, 4] are works introducing structure of Autoencoder, [1, 3] are applications of it in recommendation system.

## VI. CONCLUSION

Dealing with the task of API recommendation in Android domain, this paper gives a new design of recommendation system with Autoencoder network, and validates the effectiveness and performance of the models through detailed experiments. The experiments of this paper shows that DAE (Denoising Autoencoder) network works well in the recommendation system, with best test recall rates, which proves itself to be effective framework of Android API Recommendation System. The model also has considerable development potential, to combine additional context information of codes with API usage list, utilizing various kinds of neural networks. In future work, the authors of this paper will make attempts on this topic.

## ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (No. 61672046).

## REFERENCES

[1] Sedhain, S., Menon, A.K., Sanner, S., and Xie, L.: 'Autorec: Autoencoders meet collaborative filtering', in Editor (Ed.)^(Eds.): 'Book Autorec: Autoencoders meet collaborative filtering' (ACM, 2015, edn.), pp. 111-112

[2] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A.: 'Extracting and composing robust features with denoising autoencoders', in Editor (Ed.)^(Eds.): 'Book Extracting and composing robust features with denoising autoencoders' (ACM, 2008, edn.), pp. 1096-1103

[3] Li, S., Kawale, J., and Fu, Y.: 'Deep collaborative filtering via marginalized denoising auto-encoder', in Editor (Ed.)^(Eds.): 'Book Deep collaborative filtering via marginalized denoising auto-encoder' (ACM, 2015, edn.), pp. 811-820

[4] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A.: 'Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion', Journal of machine learning research, 2010, 11, (Dec), pp. 3371-3408

[5] Linden, G., Smith, B., and York, J.: 'Amazon. com recommendations: Item-to-item collaborative filtering', IEEE Internet computing, 2003, (1), pp. 76-80

[6] Su, X., and Khoshgoftaar, T.M.: 'A survey of collaborative filtering techniques', Advances in artificial intelligence, 2009, 2009

[7] Xie, T., and Pei, J.: 'MAPO: Mining API usages from open source repositories', in Editor (Ed.)^(Eds.): 'Book MAPO: Mining API usages from open source repositories' (ACM, 2006, edn.), pp. 54-57

[8] Hou, D., and Pletcher, D.M.: 'An evaluation of the strategies of sorting, filtering, and grouping API methods for code completion', in Editor (Ed.)^(Eds.): 'Book An evaluation of the strategies of sorting, filtering, and grouping API methods for code completion' (IEEE, 2011, edn.), pp. 233-242

[9] Heinemann, L., Bauer, V., Herrmannsdoerfer, M., and Hummel, B.: 'Identifier-based context-dependent api method recommendation', in Editor (Ed.)^(Eds.): 'Book Identifier-based context-dependent api method recommendation' (IEEE, 2012, edn.), pp. 31-40

[10] Thung, F., Wang, S., Lo, D., and Lawall, J.: 'Automatic recommendation of API methods from feature requests', in Editor (Ed.)^(Eds.): 'Book Automatic recommendation of API methods from feature requests' (IEEE Press, 2013, edn.), pp. 290-300

[11] Asaduzzaman, M., Roy, C.K., Monir, S., and Schneider, K.A.: 'Exploring API method parameter recommendations', in Editor (Ed.)^(Eds.): 'Book Exploring API method parameter recommendations' (IEEE, 2015, edn.), pp. 271-280

[12] Li, L., Bissyandé, T.F., Klein, J., and Le Traon, Y.: 'Parameter values of Android APIs: A preliminary study on 100,000 apps', in Editor (Ed.)^(Eds.): 'Book Parameter values of Android APIs: A preliminary study on 100,000 apps' (IEEE, 2016, edn.), pp. 584-588

[13] Nguyen, T.T., Pham, H.V., Vu, P.M., and Nguyen, T.T.: 'Recommending API usages for mobile apps with hidden markov model', in Editor (Ed.)^(Eds.): 'Book Recommending API usages for mobile apps with hidden markov model' (IEEE, 2015, edn.), pp. 795-800

[14] Gu, X., Zhang, H., Zhang, D., and Kim, S.: 'Deep API learning', in Editor (Ed.)^(Eds.): 'Book Deep API learning' (ACM, 2016, edn.), pp. 631-642

[15] Nguyen, A.T., Hilton, M., Codoban, M., Nguyen, H.A., Mast, L., Rademacher, E., Nguyen, T.N., and Dig, D.: 'API code recommendation using statistical learning from fine-grained changes', in Editor (Ed.)^(Eds.): 'Book API code recommendation using statistical learning from fine-grained changes' (ACM, 2016, edn.), pp. 511-522

[16] Pham, H.V., Vu, P.M., and Nguyen, T.T.: 'Learning API usages from bytecode: a statistical approach', in Editor (Ed.)^(Eds.): 'Book Learning API usages from bytecode: a statistical approach' (ACM, 2016, edn.), pp. 416-427