LMU Munich
Frauenlobstraße 7a
D–80337 München
Institut für Informatik

Master Thesis

Intent Prediction with Vectorized Sequential Android UI Tree Data

August Oberhauser

august.oberhauser@campus.lmu.de

Course of Study:	Medieninformatik
Examiner:	Prof. Dr. Sven Mayer
Supervisor:	Florian Bemmann, M.Sc.

Commenced:	June 20, 2023
Completed:	January 8, 2024

Abstract

The interaction of a user with an end device such as a smartphone or a computer is very diverse and difficult to predict. Nevertheless, user-specific (personalized) as well as global (collaborative) patterns can possibly be worked out with the help of preceding user interactions. These could be used to predict the intention of a user or a group of users. It is interesting to know to what level of detail these predictions can be made reliably. By making use of the continous on-device data an attempt can be made to gain more insights in the user behavior or even forecast their next actions.

It suggests itself to implement this with the help of user interactions in sessions on Android devices. For this purpose, the Sequential UI Tree data of the device could be tracked, filtered and labeled and then trained with a machine learning model to find similar interaction sequences and then make predictions. These can then be very coarse, such as predicting the next app. Or they can be very detailed, e.g., determining the next user action, such as filling out a form field.

A concept will be developed on how a model for predicting user intent could be built and how it could be applied to the user session. To this end, possibilities for collecting and vectorizing sequential UI trees (e.g., from the Android Accessibility Service) will be discussed (e.g., via Recurrent Neural Network (*RNN*) [28] [4] [14], *Seq2Seq* Model [5], *Screen2Vec* Model [22], *Intention2Text* [31], *Html2Vec* [30]), which are designed to predict the user intent. Here, privacy and feature pre-filtering in UI data plays an important role. After that, personalized as well as collaborative data can be used in a hybrid approach. This model should then be made available to the user in an Android app service and, depending on the level of detail, suggest upcoming apps or actions to the user at a suitable time. It should also be considered whether the user can contribute to the learning process and improve suggested actions through feedback (labeling). The performance of the model can be measured, for example, by indicators such as the amount of training data and time spent on the learning process. The effectiveness can be evaluated by accuracy metrics in predicting, for example, app categories [24] or complete test sequences via Rico [6] or ERICA [7].

Furthermore, the machine learning model could provide the following benefits in addition to intent prediction:

- reduction of the complexity and size of the UI tree
- creation of user groups that have similar behavior when using digital UI systems [17]
- elimination of technical expertise on individual features that would be required to manually compare user sessions [16]
- consideration of a user's history over time (sequential)
- comparison of user interactions without providing privacy invasive information
- supporting app developers to improve their app design and usability
- application in psychology and market research
- pre-loading of processes on devices (energy savings) [29]

As listed above, many fields of application can profit by elaborating such a system. It would be exciting to know, how the concrete concept would look like and if it can be implemented successfully e.g. to improve the user experience on end-user devices.

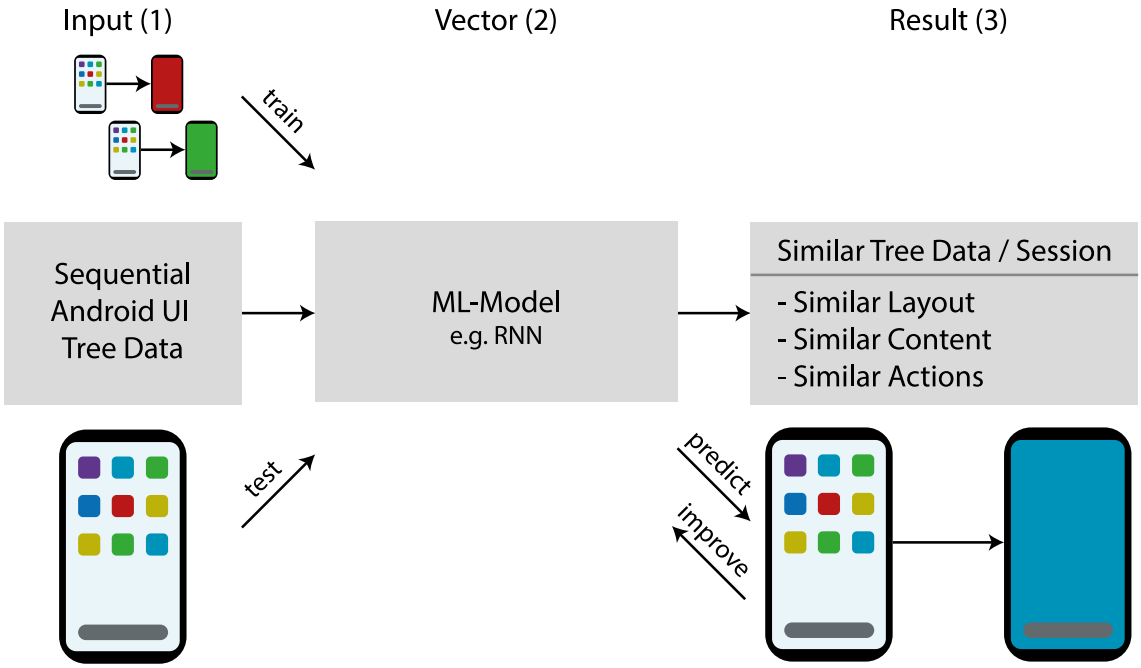


Figure 1: Possible procedure using a Machine-Learning algorithm to predict the next intent from a beginning user session: The input (1) can be a sequence of Android tree data. With help of a Machine-Learning-Model (2) (e.g. RNN) a vector representation can be trained and then predict the most probable action or screen (3) from a given starting sequence, but also can be improvde through the users feedback.

Contents

List of Figures

List of Tables

Glossary

accuracy

[TODO]: Accuracy explained in Udemy

. 30, 32

Big Data Extremely large and complex data sets which can only be processed with modern computing soft- and hardware. 20

one-hot A combination of multiple binary values, where one of them is 1 and the others are 0. 9, 21, 22, 24

Protocol Buffers A language-neutral, platform-neutral extensible mechanisms for serializing structured data, developed by Google, see https://protobuf.dev. 30

rooting Method to gain privileged access to the operating system Android. 32

Acronyms

Application Programming Interface (API) . 17

Artificial Neural Network (ANN) . 22

Continuous Bag-of-Words (CBOW) . 30

Convolutional Neural Network (CNN) . 28, 30

Graphical User Interface (GUI) . 30

Graph Neural Network (GNN) . 28

Long Short-Term Memory (LSTM) . 32, 33

Machine Learning (ML) Scientific approach to form statistical models without the need to explicitly program it. 20, 21

Neural Network (NN) . 30

Operating System (OS) . 17, 19, 32

Recurrent Neural Network (RNN) . 22, 30

Residual Neural Network (ResNet) . 28

Social Networking Service (SNS) . 32

Unidirectional Data Flow (UDF) . 17, 18

User Interface (UI) . 7, 17, 18, 19, 22, 30

1 Introduction

This is a typical human-computer interaction thesis structure for an introduction which is structured in four paragraphs as follows:

1.1 Necessarity of Vectors for Android UI

Motivation for transforming Android UI tree data to vectors

- open source code for predicting next user click / action - evaluate and compare existing approaches - provide tools to reduce screen sequences to vectors - how to work with multidimensional (multi-modal) data in RNNs

- Low Button depth: number of clicks until one gets to the action [19]

- Explain intent, and the other words in the title

Contributions: [32] • An analysis of a large-scale dataset of mobile user click se- quences that reveals rich factors and complexity in modeling click behaviors, which contributes new knowledge to under- stand mobile interaction behaviors. • A Transformer-based deep model that predicts next element to click based on the user click history and the current screen and time. The model does not rely on a vocabulary of prede- fined UI elements and provides a general solution for model- ing arbitrary UI elements for click prediction. • A thorough experiment that compares our deep model with multiple alternative designs and baseline models, and an analysis of model behaviors and benefits that the model can bring to improve mobile interaction.

Contributions: [22] Screen2Vec: a new self-supervised technique for generating more comprehensive semantic embeddings of GUI screens and components using their textual content, visual design and layout patterns, and app meta-data. (2) An open-sourced GUI embedding model trained using the Screen2Vec technique on the Rico [9] dataset that can be used off-the-shelf. (3) Several sample downstream tasks that showcase the model's usefulness.

[TODO]: P1.1. What is the large scope of the problem?

[TODO]: P1.2. What is the specific problem?

[TODO]: P2.1. The second paragraph should be about what have others been doing

[TODO]: P2.2. Why is the problem important? Why was this work carried out?

[TODO]: P3.1. What have you done?

[TODO]: P3.2. What is new about your work?

[TODO]: P4.1. What did you find out? What are the concrete results?

[TODO]: P4.2. What are the implications? What does this mean for the bigger picture?

2 Theoretical Framework

Before starting with rehabilitate the main topic, some basic concepts, technical terms and underlying theory has to be explained. This is important to faster read through the thesis without the need to interrupt the flow of thought. In the following, a set of established theories and terms is explained which are not directly related to the thesis formulation, but essential to comprehend numerous contexts.

2.1 Android UI

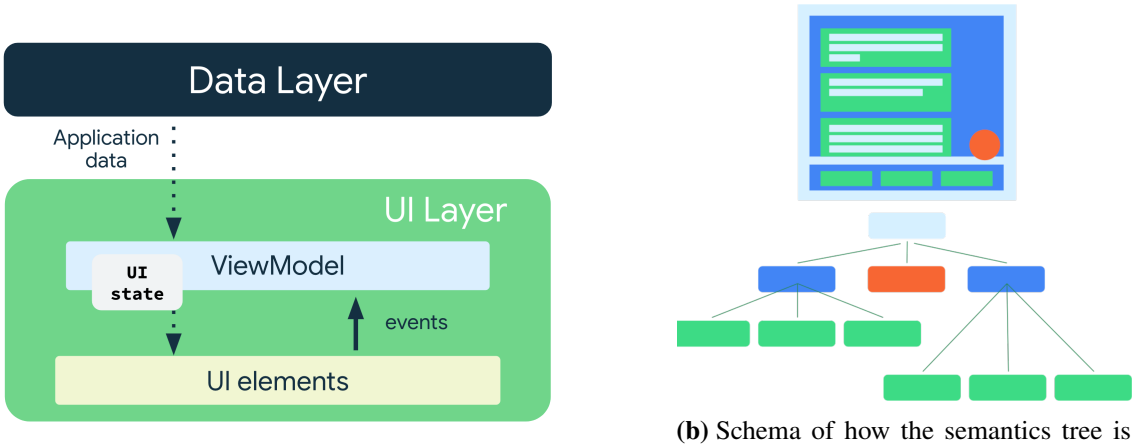
To be able to use any of the things displayed on the mobile device, some concepts of UI programming must be shown. A UI enables the user to view the applications data on the screen but also to interact with the device especially on mobile devices [12].

The main challenge is to bring the application data in the right format, so that the display can interpret the instructions to draw the elements. Each mobile phone with the Android Operating System (OS) has a basic set of native functions through which the UI can be drawn and updated, a so-called Application Programming Interface (API). These functions can be very general to instruct drawing a whole component such as an alert box, or they can be very specific as drawing single rectangles in a canvas.

The rough transition from the application data (data layer) to the display data (ui layer) is depicted in figure 2.1a. The application data is transformed, concatenated or filtered to be saved in a view model, which represents the state for each view. The view model is then layed out to multiple UI elements. E.g. they are loaded in the Android activity via view layouts [9] or composed in a declarative approach [11]. It is generally advised to use a Unidirectional Data Flow (UDF). This ensures that the data is only changed in one place and doesn't get out of sync between UI elements, the view model and the data layer. The UI can also register user inputs (like a button press) and report them back to the view model. The view model then updates the application data, if needed, and then also reports the current UI state back to the UI elements to be rerendered.

2.1.1 Data tree structure

The UI elements (i.e. the composition) themselves are hierarchically structured in a tree. This allows the renderer to calculate relative distances, floatings and skip processing hidden or overlapped elements.



(a) Diagram of UDF in app architecture [12]

(b) Schema of how the semantics tree is related to the UI hierarchy [10]

Figure 2.1: Structure of the Android UI

With the Android Layout Inspector (figure 2.2) a view hierarchy tree can be visually inspected while displaying its position and layout on the Android screen. Also, the layout attributes can be validated. This tool allows to debug complex UIs especially when using nested components and display them in a simplistic way. Note that this tool is only available if one has access to the app's source code.

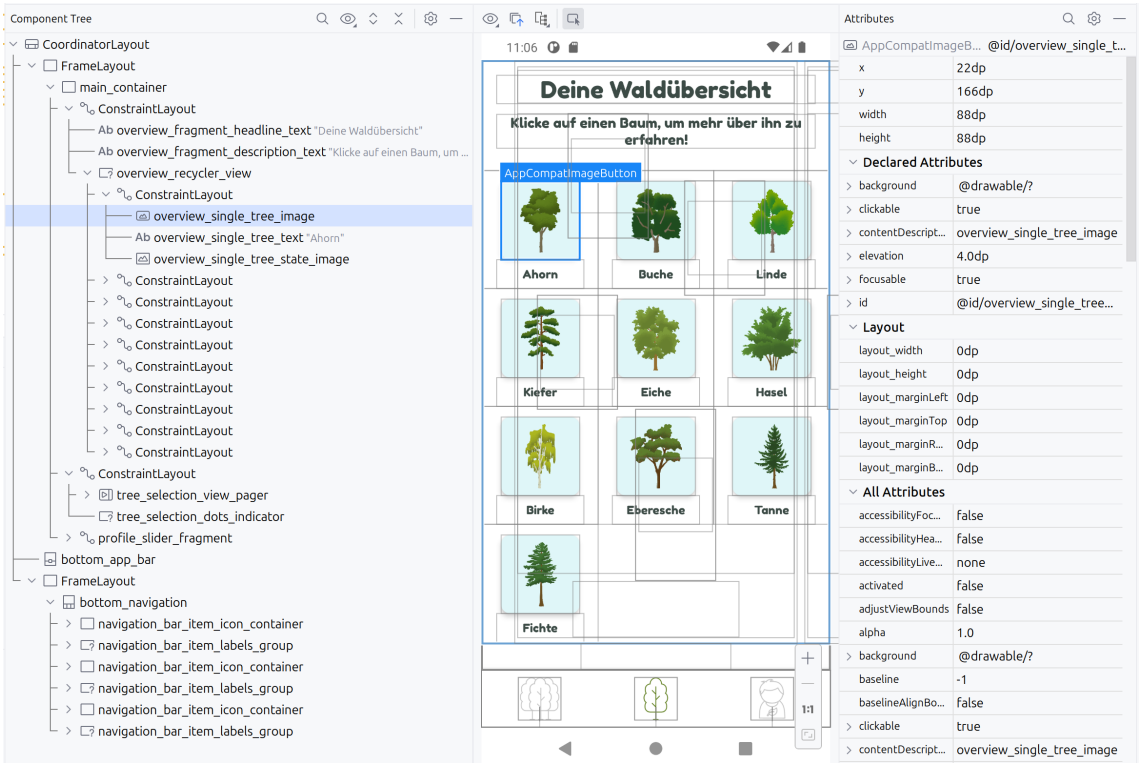


Figure 2.2: The Android Layout Inspector tool using the example of "App ins Grüne" [1] by the Media Informatics Group of the LMU in Munich.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
<hierarchy>
    <node index="0" text="" resource-id="" class="android.view.ViewGroup"
          package="de.lmu.treeapp" content-desc="" checkable="false"
          checked="false" clickable="false" enabled="true" focusable="false"
          focused="false" scrollable="false" long-clickable="false"
          password="false" selected="false" visible-to-user="true"
          bounds="[0,137][1440,2923]">
        <node index="0" text="" resource-id=""
              class="androidx.viewpager.widget.ViewPager" package="de.lmu.treeapp"
              content-desc="" checkable="false" checked="false" clickable="false"
              enabled="true" focusable="true" focused="false" scrollable="true"
              long-clickable="false" password="false" selected="false"
              visible-to-user="true" bounds="[4,141][1436,2707]">
            <node index="22" text="Eiche" resource-id=""
                  class="android.widget.TextView" package="de.lmu.treeapp" ... />
        </node>
        <node index="1" text="" resource-id="" class="android.view.ViewGroup" ... >
            <node NAF="true" index="0" text="" resource-id=""
                  class="android.widget.FrameLayout" package="de.lmu.treeapp" ... />
        </node>
    </node>
</hierarchy>
```

Listing 2.1: Android Accessibility Node in XML.

2.1.2 Android Accessibility Service

If an app is running in production on a users device, meaning that the app is compiled and publicly available, the ways of accessing the Android UI tree are limited. This behavior is of course wanted for safety and privacy reasons. Nonetheless, if desired, a user can explicitly allow certain apps to gain access to the semantics tree of your Android OS. This is especially useful for providing accessibility services for impaired users (like done with the TalkBack app). Or – as in our case – the setting can be used to enable services which collect data for user studies or scientific experiments.

This semantics tree is deviated from the existing UI tree. It can be fed via special semantic properties while composing the UI, e.g. by specifying the `contentDescription` property of an icon [10]. Providing semantics is not limited to native platforms as shown by Flutter [3] and React Native [13]. In figure 2.1b a schema is presented, which shows how the elements of the semantics tree are spanned compared to the components on the UI layer.

To take advantage of the semantics tree, a custom accessibility service can be built, which can run in the background. This service tracks all UI changes and has access to the current view hierarchy of the screen, which also inherits the semantic tree. By altering the code of the *AccessibilityNodeInfoDumper* one can extract the view hierarchy to a locale or remote database [27]. In the code section 2.1 a small fraction of a view hierarchy is shown. It contains nested nodes with various attributes which represent the components of the combined UI and semantics hierarchy.

2.2 Machine Learning

Machine Learning (ML), a term spread by Arthur Lee Samuel, is a method of data analysis, more precisely a scientific approach to form statistical models without the need to explicitly program it [25]. It uses algorithms to iteratively learn how data is structured. In contrast to statistical inference or manually crafted statistical models respectively, ML can solve tasks by automation of model building. Its advantages lie in finding hidden relations and patterns from the context, without having any or only a small pre knowledge of the data, thus it is a strong tool for generalization or abstraction of large datasets, also known as Big Data. ML can be applied to the following fields among others: email and spam filtering, fraud detection, cybersecurity, web search engines, recommender systems (like known from Netflix or Amazon), advertising, translators and text generation, pattern and image recognition. The data driven approach also comes with some drawbacks: the outcome heavily depends on the provided data. It can include biases and therefore may acquire forms of discrimination or unfair treatment. Nonetheless ML has a lot of potential to uncover hidden connections in large datasets.

[TODO]: explain Tensors, Datasets

2.2.1 Preprocessing

Preprocessing describes the step after one acquired their data, but before training the ML model. This step is not to be underestimated. A ML model can perform significantly better when certain preprocessing steps are applied [2].

To be able to preprocess, we have to know with what kind of data we handle with. Data entries can occur in different forms, but we can break them down in three main types:

- **Categorical values**: a value is always assigned to a class with a fixed pool of predetermined classes. E.g. letters, words, brands, animals, chemical elements
- **Continuous values**: the value can be fractional and may lies in between a lower and an upper bound. E.g. temperature, velocity, geographic position
- **Integer values**: the value is a whole number and may also lie in between a lower and an upper bound. E.g. revolutions per minute, product number, annual sales

For all types – discrete, continuous, and categorical values – we have a wide variety of options for preparing them in order to be subsequently processed by a ML model [15].

2.2.1.1 Feature selection

Feature selection is a crucial step to successfully develop a model with the desired results. It can improve learning performance, thus reduces time, increases computational efficiency, decreases memory storage, and helps build better generalization models [21]. Also, it may be a valid approach to get around missing data and can help structure the data by removing unnecessary clutter.

On the technical sight, feature selection can be differentiated for two goals: supervised and unsupervised learning [21]. However, principles from the supervised feature selection can be applied in the unsupervised domain, resulting in a semi-supervised filter selection. For classifiers and regression problems (unsupervised) following methods can be applied. Multiple features can be compared by calculating their correlation. If one feature is uncorrelated to all other features, this may be an indicator that this feature can be dropped, as it may doesn't contribute to the resulting model (*filter method*). However, this can only be stated for linear correlations, thus it may contribute to the result in an unpredicted way. Also, if two features correlate too much to each other, one feature may is redundant and can be dropped. A good approach is also to reduce the dimensionality of the input data, e.g. by replacing one-hot encoded features of the same domain with embeddings (*embedded method*). This is also called *feature extraction*. Feature extraction can also be used in unsupervised feature selection. Clustering is a common approach to reduce the number of input dimensions by gaining insights of which classes can be merged and which need to stay. A more computational but promising solution is to filter the features by optimizing the model result, also called a *wrapper method*. By gradually removing and adding features and calculating the models performance, one can determine which inputs are important and which may only steal computing time.

[TODO]: formulate or remove

semantic or legal feature selection Such as Filtering privacy invasive details Remove sensitive data

Parameterizing the vectorization process a) Vector length b) Weighting of features c) Manipulating individual parameters of model

2.2.1.2 Missing data

Some data entries may are missing. Therefore, you have two approaches to get around these missing values. One can drop these values by removing the column or row. This is only recommended if you are not relying on this data entry, or this the whole feature is not expected to be important enough to bring any value to the model's performance. Further you can fill the data with a default value like zero or calculate a reasonable value from the surrounding data entries by taking their "mean, median, or interpolation" [15]. The second approach can only be applied to numerical data.

2.2.1.3 Normalization and Standardization

This is only applicable for numerical data. Many ML models work better or exclusively with normalized data. This means that the values have to be in a certain range, most commonly are from **0** to **1** or from **-1** to **1**. This can be achieved by dividing all values with the difference of the minimum and maximum value and shift the output accordingly [15].

Sometimes this is not enough, e.g. if having a few extreme values, and an approach is desired which better reflects the average data. Here the standardization, also called z-score normalization, comes into play. This method scales the values so that the mean value is placed at **0** and the standard deviation is placed at **1**.

2.2.1.4 Padding

Padding is a technique to adapt input sequences or matrices of different dimensions to the same size. For example if one screen only has five UI elements, and the next one has twenty, the input size varies significantly. Therefore, there exists three approaches to overcome this problem [26]. The first one is to extend all inputs of a specific feature to the longest available input dimension of this feature. Then every sequence has to be filled with additional values, almost always **0** is used for that, until it matches the longest dimension (*same padding*). Another technique called *valid padding* cuts all data values after reaching the smallest dimension of all inputs of the feature. One can also use a combination to e.g. pad all inputs with zero to the mean dimension, but throw away all exceeding values as they aren't expected to contribute to a better result. *Causal padding* is a form of *same padding*, but it prepends the fill values in front of the sequence. This is helpful if having Recurrent Neural Network (RNN)s, which rely on a lengthier inputs in order to predict their next value, without need of filtering out short sequences.

2.2.1.5 Categorical Values and One-Hot-Encoding

A categorical value must be treated differently than numerical ordinal values. This is demonstrated best on a concrete example. Imagine a dataset with pictures of animals, and we want to categorize their species. Then the values of all possible species like `dog` and `cat` is called *vocabulary*. To write it in a table, one could add a column called `species` and write their species as a *string* (cf. table 2.1a). Unfortunately a Artificial Neural Network (ANN) cannot work with *strings*, but only with numerical values. So one could think that mapping each species to a number can solve this issue. Indeed, this is possible for ordinal categories, which have a strict linear order, such as ratings or gradings. They are then treated the same as **integer values**. But animals aren't structured ordinal, thus each species must be treated equally. To reflect that we split the species column in multiple sub-columns, so that each new one represents one species, see table 2.1b. If the animal belongs to that species, one inserts a **1** or **True** and if not, a **0** or **False** is filled. This is also called one-hot-encoding.

I_d	**Img**	**Species**
0	*blob*	cat
1	*blob*	dog
2	*blob*	cat
3	*blob*	horse

(a) Raw data table

I_d	**Img**	**Cat**	**Dog**	**Horse**
0	*blob*	1	0	0
1	*blob*	0	1	0
2	*blob*	1	0	0
3	*blob*	0	0	1

(b) One-hot encoded species

Table 2.1: One-hot-encoding of categories, I_d is the index of the data entry

2.2.2 Supervised vs Unsupervised vs Semisupervised

2.2.2.1 Supervised Learning

Supervised: Classification and regression Uses **labeled** examples: Input and output is known

Learns by comparison of the output it is provided with the output the model *predicts*.

Steps: - Data acquisition - Data cleaning / Preprocessing (Panadas) - Split into Training Data, Validation data, and Test data (cannot adapt the model after using the test data) - Train the model with the train data - Evaluate the model with the test data, then can adapt the model by the developer - Last deploy the model to production

2.2.2.2 Unsupervised Learning

Clustering Reinforcement learning

self-supervised

2.2.3 Under and Overfitting

2.3 Artificial Neural Nets

- Uses biological neuron systems as paradigm to generate mathematical models - can solve tasks by abstraction or generalization of data relations

Activation Functions Cost function Gradient - Regression: Continuous Values - Classification: Multiple class - One Class

2.3.1 Classes of Neural Nets

2.3.1.1 Deep Neural Nets

Neural Net with more than one layer - Dense Layer

2.3.1.2 Convolutional Neural Nets

2.3.1.3 Recurrent Neural Networks and LSTMs / GRU

LSTM 4 dimensional

Limitations to only 3 dimensions, needs flattening

Sample dimension (X -> y) Time (Step) Dimension Feature Dimension Data, Quantity dimension, such as Image dimensions, or multiple nodes

TimeDistributedLayer

2.3.1.4 Dense Layer

2.3.1.5 Embedding Layer

As mentioned in section 2.2.1.5 categories can be represented using one-hot encoding. The conversion can result in innumerable amount of columns, which is equivalent each having its own feature dimension. To reduce the dimensionality of such encoding, so-called categorical *embeddings* can be introduced. This means that each single species is represented by a vector. The length of the vector can be selected freely, it is called the *embedding dimension*. A lookup-table is created which encodes each category with randomly initialized weights of size of the embedding dimension (cf. table 2.2b). Now the embedding vector for each species is looked up and replaced in the data table resulting in an embedded representation (cf. table 2.2a). During training of the model the weights of the look-up table are successively updated (like for the dense layer 2.3.1.4) to reduce the loss of the overall model. An embedding would result in an one-hot-encoding, if the embedding dimension and the vocabulary size are equal and the conversion matrix (lookup-table) is the identity matrix.

I_d	**Img**	**SP_1**	**SP_2**
0	*blob*	0.1	0.6
1	*blob*	0.4	0.8
2	*blob*	0.1	0.6
3	*blob*	0.5	0.5

(a) Species encoded with embedding

Species	I_s	**SP_1**	**SP_2**
0	cat	0.1	0.6
1	dog	0.4	0.8
2	horse	0.5	0.5

(b) Lookup table for the 2-dimensional embedding of species

Table 2.2: Embedding of categories, I_d is the index of the data entry, I_s is the index of the species

According to [2] these steps can be removal of emoticons, elimination of stopwords and stemming for text based models.

Category Embedding before LSTM

Embedding dimension is about the actual voc_size, but not too large. Dimension near the actual average length of features (?)

2.3.1.6 Autoencoders

Encoder, Decoder

2.3.2 Tensorflow and Keras

Layers FlattenLayer

Positive Integer to Dense Vectors of fixed size

2.4 Evaluation and Metrics

2.4.1 Mean Squared Error

2.4.2 F1 Score

3 Related Work

[TODO]: introduce in related work, differentiate the importance of datasets

Describe relevant scientific literature related to your work.

3.1 Datasets of UI Trees

Requirements on a good data set - Google and Samsung - need to have correct data - needs enough data to train a NN - need enough features to be able to recognize patterns - up to date - Publicly available - Variable length of app sessions, define one session of activating the screen until it is turned off.

Missing - System to feed in in real time - Dataset which is across multiple apps, also tracks the system -

3.1.1 ERICA

ERICA is a design and interaction mining application, which allows gathering *interaction traces* by capturing the users activity on Android apps [7]. This is accomplished through a web-based interaction layer in contrast to the other common approach of using *accessibility services* directly. They justify that approach by the lack of need to install additional applications, as only a browser is required. A further reason is the response latency of the commonly used *UiAutomator*, which cannot collect the data in time. Also they argue that capturing and simultaneously interacting with the apps may overload the user device and challenges the user experience. Therefore the much more powerful servers take the task of capturing the UI trees. The apps are hosted on multiple physical devices with a modified Android OS directly connected with the server. ERICA captures UI screens and user flows by tracking UI changes. They then used this data to form k-mean clusters from the UI elements (visual and textual features) and the interactive elements (icons and buttons). Based on the clusters they then build classifiers and trained an AutoEncoder (2.3.1.7) to determine the flows from the test dataset. The authors worked out 23 common user flows (from over a thousand popular Android apps) which aim to provide complementary, promising or new design patterns and trends.

3.1.2 Rico / RicoSCA

Rico [6] (spanish for “rich”) is the successor of ERICA. It aims to help perform better at designing and support the creation of adaptive UIs. As far as known to date this is the largest collection of mobile app designs and traces with covering 72k UI screens in 9.7k Android apps. Like its predecessor Rico uses a web-based approach to collect user traces. It enables the applications like searching for designs, generation of UI layouts and code, modeling of user interactions, and prediction of user perception. It exposes visual, textual, structural, and interactive design properties of more than 72k unique UI screens. Unfortunately the dataset doesn't include interaction traces for app to app transitions or interactions with the Android OS itself. In table 3.1 a collection of all view hierarchy attributes is shown with their meaning. These were extracted by iterating over all view hierarchy files contained in the traces of the dataset. This gives insights in what attributes were recorded in the Rico dataset and what relevance they may have during training the model. The authors of Rico used their dataset to train a 64-dimensional UI layout vector **??** with an AutoEncoder 2.3.1.7. For their input they converted the UI layout hierarchy to an image with colored bounding boxes differentiating images and text. This has the advantage to be able to deal with the high dimensions inside the UI tree. But the conversion also most likely discards lots of meaningful information hidden in the UI tree semantics.

The RicoSCA dataset has been formed out of the research topic of mapping language instructions to mobile UI action sequences [23]. They removed screens whose bounding boxes in the view hierarchies are inconsistent with the screenshots with the help of annotators. The process of filtering reduced the Rico dataset to 25k more concise and meaningful screens.

3.1.3 Mobile UI CLAY Dataset

The Google researchers Gang Li et al. [20] present a a so-called *CLAY* pipeline which is able to denoise mobile UI layouts from incorrect nodes or adding further semantics to it. As basis they used the Rico **??** dataset for a subject of improvement. They state that recording results are dynamic and can get out of sync with the actual screen of the user. That leads to 37.4% of screens which contain invalid objects. This induces invisible or misaligned objects, or objects which are not clickable (greyed out). The researchers filtered invalid objects by training a Residual Neural Network (ResNet) model with the screenshots to classify nodes as as invalid if their bounding boxes don't match. Also they introduced two models: a Graph Neural Network (GNN) and a Transformer model to each determine the view type (also related to the view class). For that they considered the view hierarchy attributes as well as the screenshots via a Convolutional Neural Network (CNN). They claim they outperform heuristic approaches for detecting layout objects without a visual valid counterpart and also can recognize their types in more than 85%. This pipeline could help to improve intent prediction algorithms as less inconsistent data is applied to the model.

3.2 Vector models

- Compress a huge data set to a concise model - Vector Representation enables

Advantages: - “small” or smaller than the data set itself - No need to have pre knowledge about the topic, just need input an output (labels) for unsupervised NN -

Key	Type	Shape	Description
			Per View
activity_name	string	(1)	Name of the activity: e.g. “com.my_app.AppName.MainActivity”
is_keyboard_deployed	bool	(1)	Indicates if the keyboard is shown
request_id	int	(1)	Id used by the crawler to request the view
			Per Node
abs-pos	bool	(1)	Indicates if position in *bounds* is relative or absolute; if *true*, *rel-bounds* is set
adapter-view	bool	(1)	Indicates that children are loaded via an adapter, see [8]
ancestors	[string]	(None)	Ancestors of current node, e.g. “android.view.View”
bounds	[integer]	(4)	Absolute or relative boundaries, dependent on *abs-pos*
children	[node]	(None)	Child nodes
class	string	(1)	“com.my_app.lib.ui.views.DropDownSpinner”
clickable	bool	(1)	User can interact by press / click
content-desc	string	(1)	(Accessibility) description of the node “Interstitial close button”
draw	bool	(1)	Indicates if this node is drawn on the canvas
enabled	bool	(1)	Indicates if this node is in the enabled state
focusable	bool	(1)	Indicates if this node can be focused
focused	bool	(1)	Indicates if this node can is currently in focus
font-family	string	(1)	States the font family, e.g. “sans-serif”
long-clickable	bool	(1)	Indicates if this node has a long press action
package	string	(1)	States which packages the node belongs to “com.my_app.mypackage”
pressed	bool	(1)	Indicates if this node can is currently pressed
rel-bounds	[integer]	(4)	Relative boundaries, if *abs-pos* is set to *true*
resource-id	string	(1)	The unique resource identifier for this view “android:id/navigationBarBackground”
scrollable-horizontal	bool	(1)	Indicates if this node can be scrolled horizontally
scrollable-vertical	bool	(1)	Indicates if this node can be scrolled vertically
selected	bool	(1)	Indicates if this node can is currently selected
text	string	(1)	Text value if this node is a textual element
text-hint	bool	(1)	Explanation text for text boxes or icons
visibility	string	(1)	Indicates if this node is hidden, e.g. “visible”, “gone”
visible-to-user	bool	(1)	Indicates if this node can be seen in the viewport by the user

Table 3.1: Collection of attributes of a *view hierarchy* record, extracted from all interaction traces of the Rico [6] dataset.

3.2.1 Doc2Vec and Word2Vec

[18]

3.2.2 Screen2Vec

Toby Jia-Jun Li, Lindsay Popowski et al. [22] wrote a Neural Network (NN) called Screen2Vec which embeds the UI components while preserving the semantics. It is claimed that they are among the first to develop a NN for mobile screens which takes textual, visual design, and layout patterns and app context meta-data into account. As inspiration they used the Word2Vec **??** to predict result by considering the context and map them to a Continuous Bag-of-Words (CBOW). The self-supervised **??** model consists of two pipeline levels. The outer level (Graphical User Interface (GUI) screen level) combines embeddings of GUI components, layout hierarchy and app descriptions. The inner level is only present for the GUI components as they contain nested embeddings for the screen text and the class type. The screen text (in the inner level) as well as the app description (in the outer level) is processed using a pretrained Sentence-BERT model. The layout hierarchy is converted to a colored image encoding the text and image boundaries with colors (like in 3.1.2). With such model, a vector can be calculated for each screen which then can be compared to each other, e.g. by the euclidean distance between the pixel representations, or comparing the distance in the view hierarchy representation. When taking all features into account, both, the euclidean and the hierarchical approach, get an accuracy of around 0.85 that the correct screen is among the first 1% of the models predictions. In around half of the cases the predicted screen ("Top-1 Accuracy") is the correct one.

Such an approach of representing an Android layout and context in a vector can be used as pretrained embedding for feeding a RNN predicting upcoming screens.

3.2.3 Screen2Words

Similar to [22] Screen2words considers the screen context from the view hierarchy to create a screen embedding. The goal is to provide a summarization for an unseen screen by usage of it's app and UI context. They describe such a technique as multi-modal, as it "leverages input from multiple data sources", like screenshot images, textual labels and UI tree structures. As basis for their training data the Rico-SCA 3.1.2 dataset is used, to remove inaccurate view hierarchies. Although the public code did not provide a direct way to parse the Protocol Buffers of Rico-SCA. In addition they hired 85 labelers to manually annotate each of the 22k existing screens with multiple summarizations, resulting in more than 100k phrases.

As input for their summarizing NN they used a flattened view hierarchy which contained padded embeddings for categorical values, like the view class. Also the textual components of the screen were extracted and encoded with a pre-trained GloVE Word embedding. The GloVE encoding was also applied to the app description. The description was combined with the textual components and the other screen attributes to serve as input of a Transformer Encoder 2.3.1.6. Simultaneously each screenshot was embedded though a CNN, which was then concatenated with the Transformer Encoder to form the overall encoder of the Screen2words model. The screen summaries were used as labels for the model and were also embedded with the GloVE encoding.

As human validation they made a study consisting of more than a thousand participants. These had to give a star rating from one to five to assess the quality of the screen summarization. They also trained different graduations of their model by removing some of their modalities. That showed that using all modalities brought the best results (3.4 stars mean) in their rating. Using only the screenshots (pixel) modality the ratings were lower by almost one star indicating that adding more modalities contribute to improve the screen to word vectorization.

[TODO]: why this paper is important for this work

- can give insights in evalutation methods - uses good embedding for various input modalities - has a solution to embed words

3.2.4 Intention2Text

[31]

3.2.5 Html2Vec

[30]

3.2.6 Tree2Vec

3.2.7 Activity2Vec

3.3 Time Series / Sequence models

- one more dimension - allows predicting unseen states - back propagation -> see technical part -

RNNs: 9_Personalizing session based recommendations with RNNs [28] 10_Bansal_Hybrid RNN Recommender system [4] [14]

3.3.1 Seq2Seq Model

[5]

3.3.2 Click Sequence Prediction / PathFinder

Seokjun Lee et al. [19] propose a technique called *PathFinder* which aims to predict the sequence of user clicks in Android mobile apps. The user input and the contextual data is collected via the Android Accessibility Services **??**, so the users OS does not need to be modified or rooted. They collected the data from 55 students of their university with a sequence tracing tool and collected near 2 million button clicks from over a thousand apps. They follow a collaborative and content-based approach which takes both all the users data as well as the individual preferences into account. The *button depth* describes the number of clicks or taps until the user gets to their target screen. In average nearly the user has 16 buttons as candidates to press as the next action. With a personalized UI a the *button depth* should decrease significantly. The next user click is dependent on very recent but also on previous clicks happened a longer time ago, e.g. taking a picture relates to uploading it later to their Social Networking Service (SNS). The authors train a Long Short-Term Memory (LSTM) model to predict the next button, which will be clicked on. PathFinder predicts the most probable three buttons with a 0.76 F-measure.

In contrast to this work, *PathFinder* does not take into account the complete view hierarchy or other spatiotemporal information. Just the previous and the current app and the click history with their button properties are considered. Also as far as known the dataset and code is not publicly accessible.

3.3.3 Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning

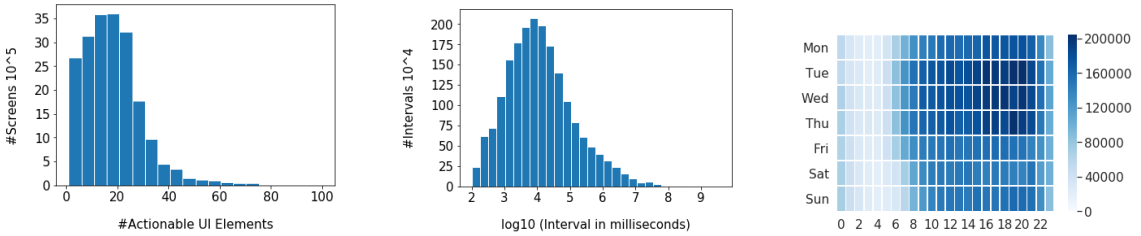
Xin Zhou and Yang Li [32] extend the work of [19]. They expect to optimize the UI experience by recommending the users next click interaction based on their findings. They gathered a dataset of 20 million clicks from 4k mobile users. The goal was to overcome the challenge of accurate but also scalable click sequence modeling. That means that apps are not limited in their composition and the screens get increasingly diverse and there's no predefined set of UI elements. Also the users click behavior is very individual and heavily dependents on situational factors.

Based on the Transformer architecture in 2.3.1.6 they created a deep learning model which has 48% accuracy for predicting the next user click and 71% accuracy for the three most probable *actionable* objects. The researchers differentiated three main inputs for embedding their elements visible on the screen: the text content, the type of view and the bounding box. All the elements are then passed to a Transformer Encoder representing a single screen. Together with the click event as well as the time encoding the screen embedding serves as one concatenated input step. The encoded time can be very different as it doesn't follow a regular sample interval, but is recorded as soon as the screen is changing 3.1b. Multiple input steps then form a sequence fed in a second Transformer Encoder which contains all past screens and clicks. The current screen embedding and time are then passed to a pointer (M-layer perceptron) which calculates the most probable *actionable* elements to be clicked on. They consider a UI element as *actionable*, if it is currently `clickable`, `visible` and `enabled` (cf. table 3.1). More complex screens can have much more actionable elements, which makes the prediction much more difficult 3.1a.

This paper solves a lot of problems previous attempts had. The dataset includes cross-app transitions which make 26% of all clicks, which are also considered in their model. The current context was taken into account such as the time of the day and day in the week 3.1c which adds a lot of

semantics. Further they used a transformer model with self-attention, which reduces the training times significantly compared to LSTM. Compared to the approach tested in this paper, they also predict the concrete element that will be clicked on instead of the absolute screen coordinates.

Unfortunately as far as the investigation permits, neither the dataset nor the code were publicly provided.



(a) Number of actionable elements per screen

(b) The distribution of time intervals between click events

(c) The distribution of time intervals between click events

Figure 3.1: View element insights [32]

4 Methodology

[TODO]: Describe methodology

Start with your overall approach to the research. What research problem or question did you investigate? What type of data did you need to answer it? Quantitative, qualitative, or mixed? Primary or secondary? Experimental or descriptive?

4.1 Data Aquisition

How you collected and analyzed your data Describe the specific methods you used for data collection and analysis. How did you collect and analyze your data? What tools or materials did you use? How did you ensure the quality and accuracy of your data?

- Why RICO, rather use a dataset with accessibility service. - ERICA employs a human-powered approach over an automated one: - More realistic results - required user input, which cannot emulated, Google Captcha, real data - humans detect the completion of UI updates [7] - erica is quite outdated

Any tools or materials you used in the research. The type of research you conducted

E.g. Google Scholar, Google Research, Tensorflow, Keras, Udemy, Open Source, Reproducible

4.2 Methodological variety

Explain why you chose these methods over others. How do they relate to your research question and literature review? How do they address the limitations or gaps in existing research? How do they suit your research design and objectives?

- No similar approach - No dataset present with consecutive sequential app usages - Many different approaches to solve this problem: - Encoder, Decoder, etc... - A Study can follow

4.3 Methodological choices

Evaluate and justify your methodological choices. Why you chose these methods How did they affect the outcome of your research? What challenges or difficulties did you encounter and how did you overcome them? How can you ensure the credibility and generalizability of your findings?

[TODO]: overall goal is more than just interaction traces

Goal: Make the algorithm as independent of the data as possible. Find general rules to feed the data. Applicable to other research fields, not just UI traces. Use LSTM, so predict something unseen, in contrast to RICO or ERICA, which only categorize the current context

4.4 Research Biases

How you mitigated or avoided research biases

How this thesis is working? Apparatus, Procedure, Utilities

5 Results

5.1 Datasets

- Problem with sequential data sets

5.1.1 Rico

- Too less frames.
- No transition between apps.

5.2 Preprocessing Android UI tree data

5.2.1 Filtering privacy invasive details

- rico doesn't use logins or any privat data - gestures can tell more about the user -

5.2.2 Normalization, Feature selection

Dealing with variable length data tf.io.VarLenFeature()

5.3 Model

Multiple approaches

[TODO]: create graphic for each approach

AutoEncoder:

- Encoder -> Decoder -> LSTM -> Decoder
- Encoder -> LSTM -> Decoder
- LSTM -> Encoder -> Decoder (AutoEncoder)

Decoder can either only decode to x and y or to whole UI tree.

5.4 Evaluation

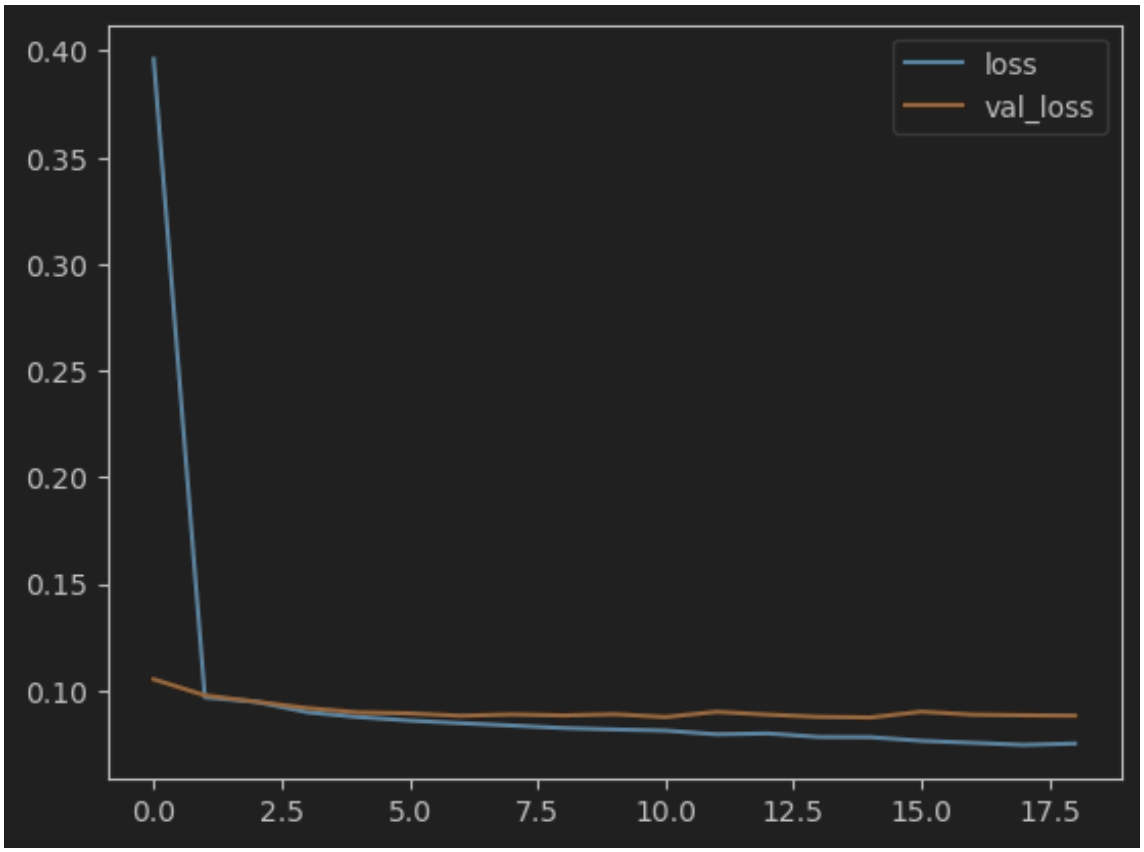


Figure 5.1: Model loss vs validation loss

5.4.1 Mean Squared Error

5.4.2 F1 Score

5.5 Limitations

Dataset Dataset is not through different apps, only in one app. Dataset is not detailed enough in the time steps, or not containing all data Dataset is not long enough Dataset has no paid apps or apps with login, which most services require Dataset has wrong data see [20]

Preprocessing Need more time to validate what are the core parameters to predict the next user intent

Model needs more investigation on what data is needed How many neurons are required to achieve this Play around with different layers, also Convolutional and pretrained embeddings

6 Application of Android UI tree vectors

6.1 Automation and testing of Android apps

6.2 UI design similarities

6.3 Action prediction models, User behavior modeling

6.4 Behavioral analyses for smartphone usage patterns

7 Conclusion and Future Work

Summary

Outlook

Future directions for research in this area

ChatGPT – Image Recognition – Limitation als Ausblick

Generate Dataset which overcomes the limitations

Make a study with actual feedback on a prediction system, visualization -> Learn faster and directly, Reinforced learning

Work out user flows, like in ERICA, but without the need to separate it from the interaction tree

Take in visual and textual context (semantics).

Make dataset with app overlapping traces. Use dataset with preprocessing such as RicoSCA or Clay. Also use accessibility service, as phones are much more powerful. No need for web interface. -> Reinforced directly on the phone, more privacy.

[TODO]: check if references are still on their own page

References

[1] Contributors of 'App ins Grüne'. *App Ins Grüne*. 2020. URL: https://github.com/mimuc/app-ins-gruene (visited on 01/01/2024).

[2] Saqib Alam, Nianmin Yao. "The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis". In: *Computational and Mathematical Organization Theory* 25 (2019), pp. 319–335.

[3] Flutter Authors. *Semantics class*. 2023. URL: https://api.flutter.dev/flutter/widgets/Semantics-class.html (visited on 01/01/2024).

[4] Saumya Bansal, Niyati Baliyan. "Remembering past and predicting future: a hybrid recurrent neural network based recommender system". In: *Journal of Ambient Intelligence and Humanized Computing* (2022), pp. 1–12.

[5] François Chollet. *A ten-minute introduction to sequence-to-sequence learning in Keras*. 2017. URL: https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html (visited on 02/15/2023).

[6] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, Ranjitha Kumar. "Rico: A mobile app dataset for building data-driven design applications". In: *Proceedings of the 30th annual ACM symposium on user interface software and technology*. 2017, pp. 845–854.

[7] Biplab Deka, Zifeng Huang, Ranjitha Kumar. "ERICA: Interaction mining mobile apps". In: *Proceedings of the 29th annual symposium on user interface software and technology*. 2016, pp. 767–776.

[8] Android Developers. *AdapterView*. 2023. URL: https://developer.android.com/reference/android/widget/AdapterView (visited on 01/01/2024).

[9] Android Developers. *How Android draws Views*. 2023. URL: https://developer.android.com/guide/topics/ui/how-android-draws (visited on 01/01/2024).

[10] Android Developers. *Semantics in Compose*. 2023. URL: https://developer.android.com/jetpack/compose/semantics (visited on 01/01/2024).

[11] Android Developers. *Thinking in Compose*. 2023. URL: https://developer.android.com/jetpack/compose/mental-model (visited on 01/01/2024).

[12] Android Developers. *UI layer*. 2023. URL: https://developer.android.com/topic/architecture/ui-layer (visited on 01/01/2024).

[13] React Native developers. *Accessibility*. 2023. URL: https://reactnative.dev/docs/accessibility (visited on 01/01/2024).

[14] Mauro Di Pietro. *Modern Recommendation Systems with Neural Networks*. 2022. URL: https://towardsdatascience.com/modern-recommendation-systems-with-neural-networks-3cc06a6ded2c (visited on 03/08/2023).

[15] Bao Tram Duong. *Fine-Tuning Inputs: Data Preprocessing Techniques for Neural Networks*. 2023. URL: https://baotramduong.medium.com/data-preprocessing-for-neural-network-0b398b43d309 (visited on 12/30/2023).

[16] Alireza Ghods, Diane J Cook. "Activity2vec: Learning adl embeddings from sensor data with a sequence-to-sequence model". In: *arXiv preprint arXiv:1907.05597* (2019).

[17] Kasthuri Jayarajah, Youngki Lee, Archan Misra, Rajesh Krishna Balan. "Need accurate user behaviour? pay attention to groups!" In: *Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing*. 2015, pp. 855–866.

[18] Quoc Le, Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing, Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, June 2014, pp. 1188–1196. URL: https://proceedings.mlr.press/v32/le14.html.

[19] Seokjun Lee, Rhan Ha, Hojung Cha. "Click Sequence Prediction in Android Mobile Applications". In: *IEEE Transactions on Human-Machine Systems* 49.3 (2019), pp. 278–289. DOI: 10.1109/THMS.2018.2868806.

[20] Gang Li, Gilles Baechler, Manuel Tragut, Yang Li. "Learning to Denoise Raw Mobile UI Layouts for Improving Datasets at Scale". In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. https://github.com/google-research/google-research/tree/master/clay. New Orleans, LA, USA: Association for Computing Machinery, May 2022, pp. 1–13. URL: https://doi.org/10.1145/3491102.3502042.

[21] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, Huan Liu. "Feature Selection: A Data Perspective". In: *ACM Comput. Surv.* 50.6 (Dec. 2017). ISSN: 0360-0300. DOI: 10.1145/3136625. URL: https://doi.org/10.1145/3136625.

[22] Toby Jia-Jun Li, Lindsay Popowski, Tom Mitchell, Brad A Myers. "Screen2vec: Semantic embedding of gui screens and gui components". In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 2021, pp. 1–15.

[23] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, Jason Baldridge. "Mapping natural language instructions to mobile UI action sequences". In: *arXiv preprint arXiv:2005.03776* (2020).

[24] Google LLC. *Choose a category and tags for your app or game*. 2023. URL: https://support.google.com/googleplay/android-developer/answer/9859673 (visited on 02/15/2023).

[25] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9.1 (2020), pp. 381–386.

[26] Georgios Nanos. *Deep Neural Networks: Padding*. June 2023. URL: https://www.baeldung.com/cs/deep-neural-networks-padding (visited on 01/01/2024).

[27] Android Open Source Project. *AccessibilityNodeInfoDumper*. 2012. URL: https://android.googlesource.com/platform/frameworks/testing/+/jb-dev/uiautomator/library/src/com/android/uiautomator/core/AccessibilityNodeInfoDumper.java (visited on 01/01/2024).

[28] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, Paolo Cremonesi. "Personalizing session-based recommendations with hierarchical recurrent neural networks". In: *proceedings of the Eleventh ACM Conference on Recommender Systems*. 2017, pp. 130–137.

[29] Zhihao Shen, Kang Yang, Wan Du, Xi Zhao, Jianhua Zou. "Deepapp: a deep reinforcement learning framework for mobile application usage prediction". In: *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*. 2019, pp. 153–165.

[30] Peihuang Wu, Jiakun Zhao. "Distributed representations of html page". In: *2022 IEEE 2nd International Conference on Power, Electronics and Computer Applications (ICPECA)*. IEEE. 2022, pp. 160–164.

[31] Che-Hsuan Yu, Hung-Yuan Chen, Fang-Yie Leu, Yao-Chung Fan. "Understanding mobile user intent using attentive sequence-to-sequence RNNs". In: *Wireless Internet: 12th EAI International Conference, WiCON 2019, TaiChung, Taiwan, November 26–27, 2019, Proceedings 12*. Springer. 2020, pp. 51–61.

[32] Xin Zhou, Yang Li. "Large-Scale Modeling of Mobile User Click Behaviors Using Deep Learning". In: *Proceedings of the 15th ACM Conference on Recommender Systems*. RecSys '21. Amsterdam, Netherlands: Association for Computing Machinery, 2021, pp. 473–483. ISBN: 9781450384582. DOI: 10.1145/3460231.3474264. URL: https://doi.org/10.1145/3460231.3474264.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature