Technical Disclosure Commons

Defensive Publications Series

February 26, 2019

Visual User Interface (UI) Interaction Prediction

Victor Carbune

Daniel Keysers

Thomas Deselaers

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation

Carbune, Victor; Keysers, Daniel; and Deselaers, Thomas, "Visual User Interface (UI) Interaction Prediction", Technical Disclosure Commons, (February 26, 2019)
https://www.tdcommons.org/dpubs_series/1982

This work is licensed under a Creative Commons Attribution 4.0 License.
This Article is brought to you for free and open access by Technical Disclosure Commons. It has been accepted for inclusion in Defensive Publications Series by an authorized administrator of Technical Disclosure Commons.

Visual User Interface (UI) Interaction Prediction

Abstract:

This publication describes an operating system (OS) level framework functionality, which can predict a user's future interaction with first-party and third-party application software. This provides the user with an enhanced user experience (UX) by enabling first-party and third-party software to dynamically change the display in the user interface (UI) to aid the user to more-easily carry out the task at hand. The framework functionality at the OS level predicts the user's future interactions using machine learning (ML). With this prediction, the supported application software can dynamically change the UI by presenting the user with relevant application software functions in a minimalist way.

Keywords:

User interface (UI), autocorrect, predict, anticipate, suggest, forecast, raster, user action history, framework functionality, machine learning (ML), recurrent neural network (RNN), neural network, dense neural network, deep learning, artificial intelligence (AI).

Background:

When using a specific software application, a user often follows a common path of navigating the operating system's (OS) user interface (UI). This may be because the user wants to consume content in a particular way, or because the user wants to accomplish something in a particular session (e.g., editing a document, photo, or movie).

Consider the user using a word-processing application software. After the user clicks on the word-processing application software icon (button), he or she is prompted to select another icon, such as a blank document, a specific blank template, or open an existing document. One reason the application software offers these choices to the user is because, after the user initiates the opening of the word-processing application software, there are only a few possible actions that the user may take — no matter how many features the word-processing application software may have.

One may be familiar with other features that application software may offer, as illustrated in Figure 1.

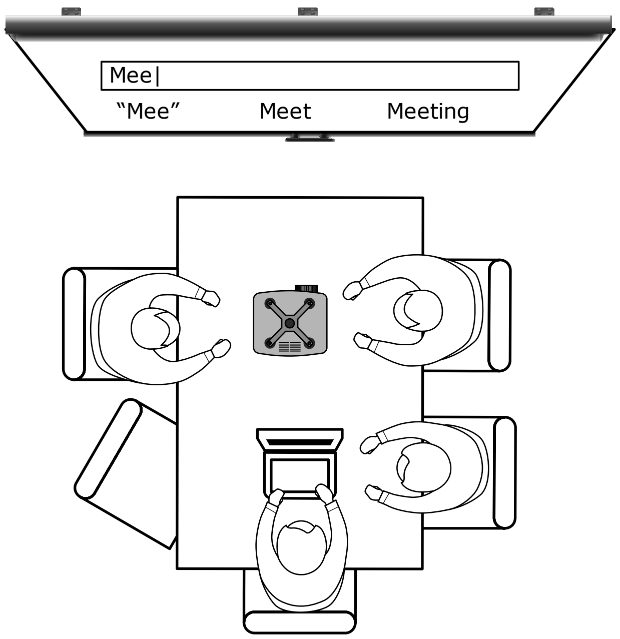


Figure 1

In Figure 1, assume Jane has called an impromptu meeting of her staff. As some members of the staff have already entered the conference room, Jane wants another member of the staff to join the meeting. On her work notebook, she opens a messaging session to contact John, who is an integral part of the meeting. As Jane starts typing a message to John to meet her in the conference room, the OS prompts her with different autofill options as she types part of the word "Meet" or "Meeting". The OS and its associated messaging application software presents Jane with the possible words that start with "Mee". Jane may quickly select one of the presented options and continue with the rest of her message.

Many applications are increasingly sophisticated and offer the user with an array of tools to carry out various tasks. As these applications integrate more functions, the ease of use may suffer due to the complexity and the sheer number of tabbed document interfaces (TDIs or tabs), toolbars, and other computer-interface design features. Application software developers are aware of the inherent drawbacks that an increased number of features may cause, such as degrading the user experience (UX). Therefore, they try to organize the software features to increase the intuitiveness of use. Nevertheless, the user still faces challenges trying to navigate these applications. After all, the user is interested in spending his or her energy accomplishing the task and not deciphering where the application software's features are located and how to best use them.

Many applications are mostly static and do not offer the user a UX similar to the example of Figure 1. Therefore, it is desirable for a larger number of applications to change the display in the UI as the user navigates the application software in a certain way. One way to do that is to change the OS's framework functionality.

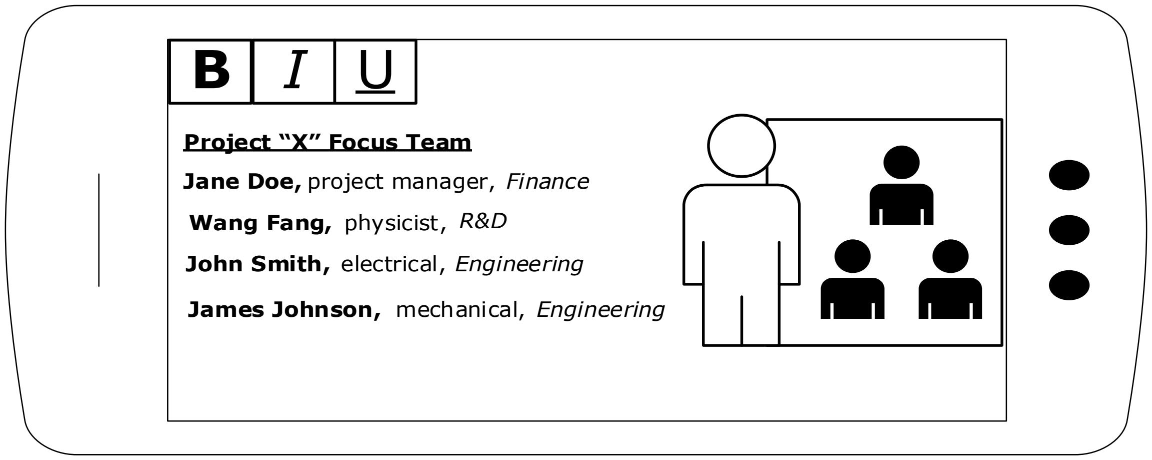
Description:

This publication describes an operating system (OS) level framework functionality, which can predict a user's future interaction with first-party and third-party application software. This provides the user with an enhanced user experience (UX) by enabling first-party and third-party software to change the display in the user interface (UI) to aid the user to more easily carry out the task at hand.

The OS's framework functionality allows the application software to adapt their user interface (UI) to better serve the user's needs in a session by rearranging visual elements by predicting visual elements of interest. To help explain the OS's framework functionality, consider the familiar word-processing application software. Various word-processing application software developers create well-thought tabs and toolbars to organize the many features of these software. Nevertheless, if the user has been predominantly typing during a particular session, the display of the many tabs and toolbars does not add much value during this session. Furthermore, many users prefer to use the application software on many platforms. For example, the user may start a word-processing session at the office's computer and continue working during the train-ride home using his or her smartphone or tablet. The screen display of embedded devices is limited and valuable "real estate". Thus, the display of many tabs and toolbars may take too much screen space and leave too-little space for the word-document itself.

This OS framework-level functionality allows this session of the application software to display, in an advantageously located part of the screen, the features that the user has been using and will most likely keep using during this session, such as **bold,** *italic,* <u>underline</u> for an editor using a word-processing application when editing text, or column insertion, merging, or deletion when editing a table..

Figure 2 illustrates how the OS aids an application software to display the features that are most relevant for the user's current session.



Project "X" Focus Team

Jane Doe, project manager, *Finance*

Wang Fang, physicist, *R&D*

John Smith, electrical, *Engineering*

James Johnson, mechanical, *Engineering*

Figure 2

Due to the familiarity of many users with the various word-processing application software, assume Jane is typing a business plan using this software, as shown in Figure 2. A major customer of Jane's company is having an issue with the latest product. The vice president (VP) of the company where Jane works has tasked her to create a focus team to find the underlying cause of the problem. Jane has been working all day at the office making phone calls and holding meetings with the proper parties. She believes that she has assembled the right team to solve the problem. Time is of the essence, and Jane has been working nonstop to present the plan to the VP for approval — first thing next morning. As Jane is riding her train to work, she decides to make a few edits to the document using her smartphone. The OS and the supported word-processing application software presents Jane with a minimalist toolbar based on the tools and features Jane has used while working on this document and, most-likely, will keep using during this session. As one can see in Figure 2, the OS and the supported word-application software dedicates most of the screen to show Jane's work, and the only three displayed features in the displayed toolbar are **bold,** *italic,* and underline text.

Furthermore, the OS and the supported word-processing application software arranges the icons inside the displayed toolbar based on a prediction on what is Jane most likely to use frequently — refer to Figure 2. Note that the most common text features Jane uses are **bold**, followed by *italic*, and then followed by underline. The toolbar arranges the icons so that Jane could easily continue following the same pattern. Jane has used the underline text in one (1) instance, in the heading of the paragraph. She has used the **bold** text in five (5) instances, in the heading of the paragraph and in the names of the assembled team. And, she has used the *italic* text in four (4) instances, in the departments of the team-members. Accordingly, the minimalist toolbar displays the icon for the **bold** text first, followed by the icon for the *italic* text, and then followed by the icon for the underline text.

Moreover, the OS and the supported application software displays the minimalist toolbar in a part of the screen so that the toolbar does not occupy the space used by the document content. In Figure 2, additionally note that the toolbar is displayed conveniently close to the text, instead of the diagram, and does not occupy any working space.

The framework functionality operates at the OS level and may consist of calls that require user-related features and a list of previous actions — specifically, the history of actions that lead to the current state. The history of actions may consist of previously displayed screens (e.g., raw pixels), a sub-part of the pixel area (e.g., bounding box) that the user has navigated to, or an action type. Examples of action types are the user clicking a button at a bounding box, scrolling to reveal more content, or navigating to move away from the current screen. In addition, a predict-interaction call may receive a list of possible interactions, which are elements that the application software provide to the framework. The framework is made available to all application software through application programming interfaces (APIs), which allow the application software to ask for the next likely user interaction.

To increase the accuracy of predicting the most helpful UI display for the user during a session, the OS level framework functionality leverages machine learning (ML), as shown in Figure 3.

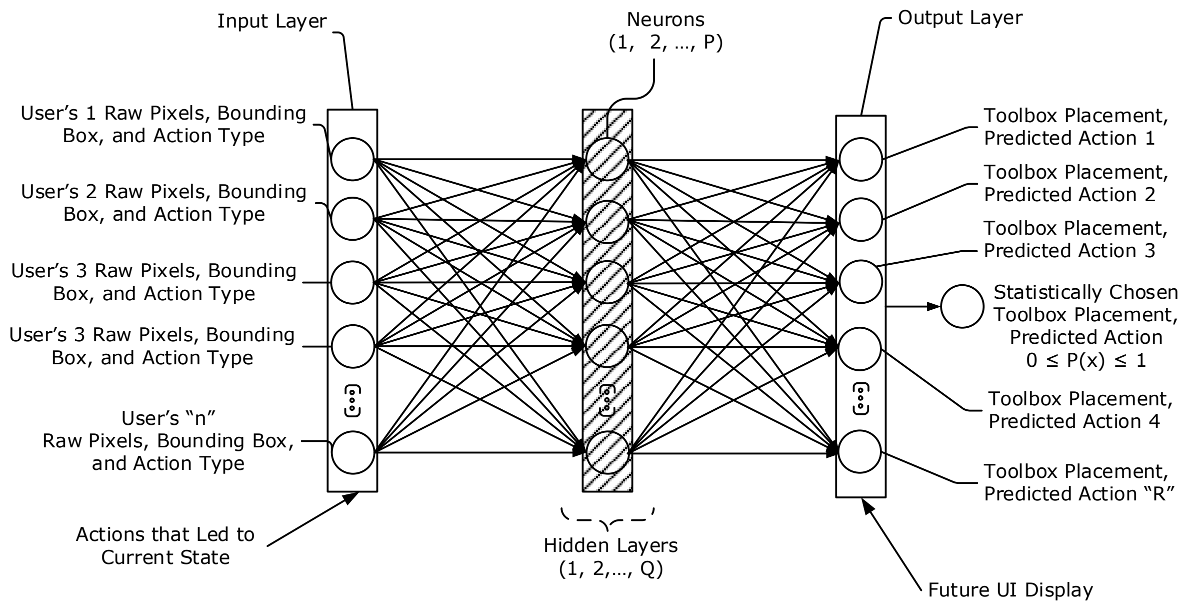

Input Layer
Neurons (1, 2, …, P)
Output Layer
User's 1 Raw Pixels, Bounding Box, and Action Type
User's 2 Raw Pixels, Bounding Box, and Action Type
User's 3 Raw Pixels, Bounding Box, and Action Type
User's 3 Raw Pixels, Bounding Box, and Action Type
User's "n" Raw Pixels, Bounding Box, and Action Type
Actions that Led to Current State
Hidden Layers (1, 2,…, Q)
Toolbox Placement, Predicted Action 1
Toolbox Placement, Predicted Action 2
Toolbox Placement, Predicted Action 3
Statistically Chosen Toolbox Placement, Predicted Action 0 ≤ P(x) ≤ 1
Toolbox Placement, Predicted Action 4
Toolbox Placement, Predicted Action "R"
Future UI Display

Figure 3

Figure 3 demonstrates a neural network used to train the model used by the framework functionality of the OS. The neural network in Figure 3 illustrates an input layer, several hidden layers, and an output layer. The input layer includes a history of steps taken that led to the current state from "n" number of users, which may be raw pixels, bounding box, action type, etc. The input data may be gathered by recording a sequence of screens from these "n" number of users. The OS keeps the identity of the "n" users anonymous and only uses the metadata for model training purposes. The specific neural network input layers may vary depending on the type of data. For example, recurrent neural network layers may be used for processing variable length sequences, while convolutional network layers may be used for processing each image per step. Once processed by different neural network layers, they can be concatenated and further passed to "Q" number of hidden layers with up to "P" number of neurons in each layer, which constitutes a feed-forward neural network. There can be a different quantity of neurons in each hidden layer. The output layer may consist of a heatmap over the screen rendered to the user. For example, each pixel may be associated with a probability score representing how likely is the user to tap to a specific region of the screen. In other cases, the output layer includes "R" number of bins with different probabilities on the future display of the UI. The bin with the closest probability to one (1) is interpreted as the correct output.

Various types and several number of neural networks may be used for machine learning. Nevertheless, recurrent neural networks (RNNs) may be the most appropriate, because they are capable of having arbitrary lengths and strings as inputs. Differently said, the RNN can consider the sequence of screens, the sequence of raw pixels, the sequence of bounding boxes, the sequence of actions taken, and so forth, all at once.

One way the framework functionality may determine the next UI display is by rendering a probability heat map of the display, as shown in Figure 4.

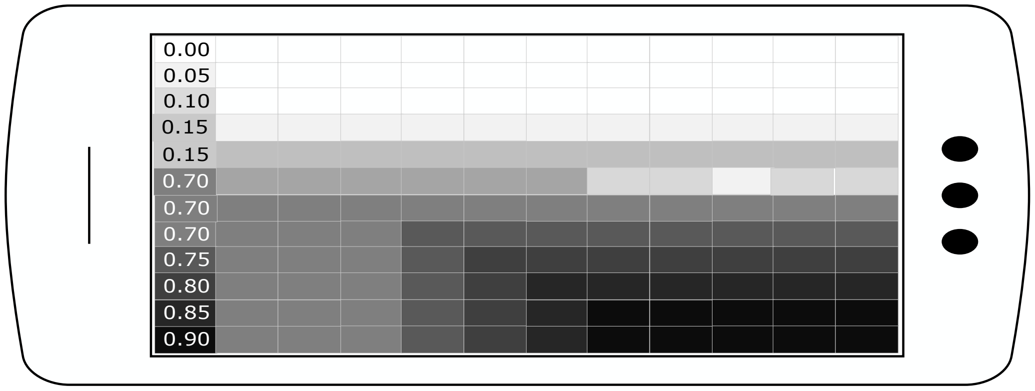


Figure 4

Figure 4 shows an exemplary raw pixel probability heat map. Given such a heat map, the future UI will display any toolbars in the white (zero (0) percent) area of the screen, because the highest probability of future user interaction with the content is at the black (90 percent) part of the screen. Refer to Figure 2 for an exemplary placement of the toolbar.

Given the large computational power that machine learning can use to train a model, the model training can be performed on a cloud, server, or other capable computing device or system. Periodic model updates are sent to each user's computing device, which allows the user's computing device to execute the model even if that device does not have the resources to update the model itself.

In addition, the framework functionality may also consider user related features. Continuing with the word-processing application software example, the software may change the prediction based on the user's age, profession, language, and so forth. For example, a teen uses a different vocabulary than someone in his or her 30s. As another example, a medical doctor uses a different vocabulary than an engineer. And, as yet another example, an American may spell and use certain words differently than an English, Australian, Indian, and most non-native English speakers. In these examples, the framework functionality may run at the output level of the natural language processing (NLP) stack working with the different vocabularies. Once the NLP system returns candidates, the visual interaction framework predicts which of the options will the user select.

Further to the descriptions above, the framework functionality, at an OS level, aids to predict and change the UI display, while the user interacts with application software other than word-processing application software. Moreover, the user may be provided with controls allowing the user to make an election as to both if and when systems, programs, or features described herein may enable collection of user information (e.g., user preferences), and if the user is sent content or communications from a server. In addition, certain data may be treated in one or more ways before it is stored or used, so that personally identifiable information is removed. For example, a user's identity may be treated so that no personally identifiable information can be determined for the user, or a user's geographic location may be generalized where location information is obtained (such as to a city, ZIP code, or state level), so that a particular location of a user cannot be determined. Thus, the user may have control over what information is collected about the user, how that information is used, and what information is provided to the user.

In summary, this framework functionality at the OS level enhances the UX with any supported application software by providing functionality that help create a more-dynamic and intuitive UI display.