WYSIWYG Approach to GUI for TensorFlow Deep Learning API

Behnam Saeedi, Connor Sedwick, Collin Dorsett

October 28, 2016

Contents

1 Introduction

This section gives a scope description and overview of everything included in this SRS document. The purpose for this document is described and a list of abbreviations and definitions is provided within this section.

1.1 Purpose

The purpose of this document is to give a detailed description of the requirements for the WYSIWYG Deep Learning graphical user interface software. It will illustrate the purpose and complete declaration for the development of the system. It will also explain system constraints, interface and interactions with other external applications.

1.2 Scope

The WYSIWYG Deep Learning graphical user interface is a desktop application which helps people design and test deep learning algorithms. The application should be free to download and be usable on multiple operating systems. Developers will be allowed to use their own data for input. The software will be able to interface with Google's TensorFlow API.

1.3 Glossary

- Tensorflow: TensorFlow is an open source software library for numerical computation using data flow graphs. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.
- GUI: GUI or Graphical User Interface, Is an interface that allows users to interact with a given system through graphical icons and representation of tasks as oposed to a text base representation.
- Scene In our solution, Scene is the work space where user can place different elements of their program.
- Block: A block is a a GUI feature for the user to serve a specific purpose. A block could be variable/Constant, Method, class, probe input or output.
- Block-Menu: Block-Menu is a menu that contains all of the items and functionality icons that are ready to be dragged and dropped into the scene.
- Run In our solution, the task of starting executing the program is refered to as Running the program.
- Stop In our solution, stop brings the execution of the program to a halt. Resuming the process requires running the program again. This functionality will reset the state of the execution.
- Extract The task of creating the executable version of the program which is developed by our solution is refered to as extraction of the program. In order for a user to have a stand-alone version of their implemented program, they need to extract their project.
- Variable/Constant: Variables are equivalent to programming variables. They are represented by rhombus in our graphical user interface. Constants are similar to variables, however, these values could not be changed while running the program. Constants are too represented by rhombus.
- Method: Methods are functions from the API that perform a specific task. They are represented by boxes. The circles on the edges of a method box are inputs and outputs of that method.

- Class: In our solution, class refers to an object that contains multiple variables and methods (Public or private). Classes are represented as transparent boxes around methods and variables. This box could be abstracted away to be displayes as a solid color box.
- Abstract: Abstract is feature of class that will turn the class from transparent to solid color in order to hide the content of the class.
- Channel: Channels are connections between inputs and outputs that display the direction of data flow and route the data from one block to another.
- Probe:
- input:
- Output:
- Layer:

1.4 References

1.5 Overview

The remainder of this document includes three chapters. The second chapter provides an overview of the system functionality and system interaction with other libraries. Chapter two also introduces different types of stakeholders and their interaction with the system. Further, the chapter also mentions the system constraints and assumptions about the product.

The third chapter provides the requirements specification in detailed terms and a description of the different system interfaces. Different specification techniques are used in order to specify the requirements more precisely for different audiences.

The fourth chapter deals with the prioritization of the requirements. It includes a motivation for the chosen prioritization methods and discusses why other alternatives were not chosen.

2 Overall Description

This section will give an overview of the whole system. The software will be explained in its context to show how the software interfaces with external libraries and introduce the basic functionality of it. It will also describe what type of stakeholders that will use the system and what functionality is available for each type. By the end, the constraints and assumptions for the system will be presented.

2.1 Product Perspective

The software will consist primarily of the graphical user interface which will communicate with Google's TensorFlow API. The software will need to have some way of ensuring that it has access to the TensorFlow libraries. The interface will provide to the user the basic design and flow of data through their program in a flowchart visual. The underlying code will be stored in a file which is built and saved in the background.

2.2 Product Functions

With the graphical user interface, users will be able to design an algorithm by placing shaped nodes in a build space and draw connections between the nodes to represent dependencies. Build spaces represent either individual files, classes or layers. Files will be built based on the nodes and connections drawn between them in a build space. These files can be extracted and saved to a user-designated folder at the push of a button.

Developers will be able to set probes on connections drawn between nodes to either modify values or track values as they are manipulated by their algorithm. Helpful alerts will let the user or developer know if there is an error with the way they have designed their algorithm.

2.3 User Characteristics

There are many types of users who will likely use this software. As it stands, there is only one development layout to be used for the graphical user interface. All users will be using the same desktop application whether they are students, developers, or employees using deep learning for a project.

2.4 Constraints

This project is very user based and the only constraints are imposed on usability of our software and the core library that we are trying to mask. The graphical user interface must be easily understood by the user and developer. The display of results must be reliable and dependable. The core system has its own limitations which will limit our GUI.

2.5 Assumptions and Dependencies

One assumption we have is that not all users will have prior knowledge on software programming when using this software. A user may not know what a variable is or what a function is and how it relies on variables. When piecing together an algorithm a user may not know the proper syntax that would be required when using a text editing software.

A dependency for this software is access to Google's TensorFlow libraries from the user's memory space. This will require a user to install the libraries along with TensorFlow otherwise the software will not be usable in some scenarios.

2.6 Apportioning of Requirements

In the case that the project is delayed, there are some requirements that could be transferred to the next version of the application.

3 Specific Requirements

This section contains all of the functional and quality requirements of the WYSIWYG system. It provides a detailed description of the system and all its features.

3.1 Interfaces

3.1.1 User Interfaces

The graphical user interface will begin by displaying a menu allowing the user to "Start a new project." Once this is chosen, the user will be brought to the main menu screen and build space. From this screen the user will have the ability to choose different types of nodes for either coding or debugging purposes. There will be buttons that allow users to "build" their file and "run" it.

Whenever a user wishes to pull the coded file of their algorithm in order to send it or apply it to other code they will have the ability to "Extract" the file. This will be done by pressing a button and choosing a place in memory to store it. The file should be stored in the form of a .py file.

3.1.2 Software Interfaces

The graphical user interface should communicate with Google's TensorFlow API. It should also communicate with the personal computer in the background to write the code file and executable. If an error message is thrown the interface should catch and display the error to the user in a formatted message.

3.2 Functional Requirements

This section includes the requirements that specify the fundamental actions of the software system.

ID: FR1
TITLE: Download the application
DESC: A user should be able to download the software from an application store or from GitHub. It should be free to download. The software should also come with a package of TensorFlow included.
RAT: In order for the user to download the software and have access to the require libraries.
DEPEND: None

ID: FR2
TITLE: Flowchart build space
DESC: A user should be able to open a session and place nodes on a blank build space.
RAT: In order for the user to visualize the structure of their software.
DEPEND: FR1

ID: FR3
TITLE: Data input
DESC: The software should allow the user to include a file of their own data to be used by their software.
RAT: In order for the user to test their software on their own data.
DEPEND: FR2

ID: FR4

TITLE: Layered builds
DESC: A user should be able to create multiple layers to their software by opening multiple build spaces.
RAT: In order for the user to visualize the structure of their software.
DEPEND: FR2

ID: FR5
TITLE: Background file creation
DESC: The software should create and update user named files in the background as nodes are added to the build space.
RAT: In order for the user to have a raw and formatted file of their software.
DEPEND: FR2

ID: FR6
TITLE: File compilation
DESC: The software should compile all files in the user's build directory at the press of a button.
RAT: In order for the user to create an executable in order to run the software.
DEPEND: FR4

ID: FR7
TITLE: Error handling
DESC: The software should display a text box or error message to the screen letting the user know if there is an error with their build. There should also be a helpful hint on what the user may do to fix their problem.
RAT: In order for the user to know that there is an issue with the format of their current build of their software.
DEPEND: FR4, FR5

ID: FR8
TITLE: File extraction
DESC: The software should allow the user to save their current build file to a user-defined space in memory.
RAT: In order for the user to have a raw and formatted file of their software in designated storage.
DEPEND: FR4

3.3 Performance Requirements

The requirements in this section provide a detailed specification of the user interaction with the software and measurements placed on the system's performance.

ID: QR1
TITLE: Easy installation
DESC: There should be a simple downloadable package including everything required to begin using the software after installation.
RAT: In order for the user to being using the software without having to worry about file dependencies.
DEPEND: None

ID: QR2
TITLE: Prominent node menu
DESC: The node menu for the flowchart pieces should be well-labelled and easy to understand. Nodes should be dragged and dropped onto the build space.

RAT: In order for the user to navigate the menu and find nodes easily.
DEPEND: None

ID: QR3
TITLE: Prominent feature buttons
DESC: The buttons used to build and run the software should be well-placed and recognizable. The extraction button should also indicate its purpose and be easy to find. Buttons should respond to a single click and act instantaneously.
RAT: In order to avoid confusion when trying to build and run the program.
DEPEND: None

ID: QR4
TITLE: Prominent alerts
DESC: Alerts signaled by the compiler should be displayed to the screen as soon as they are sent. When an error has been fixed its corresponding alert should disappear.
RAT: This will cut down on confusion when rebuilding a file and trying to find issues with a solution.
DEPEND: None

ID: QR5
TAG: Response time
DESC: The speed at which the interface communicates with the TensorFlow libraries and displays output to the screen.
SCALE: The time of a build.
METER: Measurements obtained by 1000 builds during testing.
MUST: No more than 3 seconds 100% of the time.
WISH: No more than 2 seconds 100% of the time.

4 Prioritization and Release Plan

In order to get a view of how to divide the requirements into different releases and what requirements should be included in which release, a prioritization of the requirements is required. This purpose of this section is to discuss the choice of prioritization methods and provide suggestions of how the release plan for these requirements could look.

4.1 Choice of Prioritization Method

5 References

6 Gantt