WYSIWYG Approach to GUI for TensorFlow Deep Learning API

Behnam Saeedi, Connor Sedwick, Collin Dorsett
October 2016
Group Number: 33 Group Name: Visual Flow

Abstract

TensorFlow is a machine learning API (Application Program Interface) developed by Google in order to provide an optimized machine learning toolset for developers. This toolset is designed to be an all-in-one machine learning solution for users who do not have the time, technical skill set or resources to produce their own methods. Despite TensorFlow having useful applications and a rich manual it does not have an eloquent visualization software for its users. To address this issue we wish to develop a product that can display data between nodes in a computational graph and be as easy as placing nodes and drawing connections between them. Our target audience is individuals with little to no experience with deep learning or computer programming who specialize in fields such as data analysis and mathematics.

Problem Definition

TensorFlow, an API developed by Google is a developer tool that requires much technical knowledge to implement and run. We need to develop a GUI (graphical user interface) that allows users the ability to access TensorFlow methods, use them to create a program and visualize its control flow. We need to develop an interface that is easy to navigate while also providing useful feedback to the user if there is an error in their implementation. It should be easy to install and support multi-layered program designs as well as different types of data input such as text or graphical data. Furthermore, this software needs to support a simple drag, drop and connect functionality to enhance user experience.

Proposed Solution

Our proposed solution is to use Python scripting language to develop a software that grants developers the ability to graphically visualize the control flow of their program using a flowchart style design backed with TensorFlow API. It will also track states of the program between methods during execution to aid monitoring and modifying run-time parameters. Furthermore, this software warns users if they are missing a specific node required in their program that is required for it to compile and run correctly.

The following is how we plan to model our solution:

- Display the build space in a graphical user interface. This display is called the Scene.
- Represent variables as blocks. These blocks contain data.
- Represent methods as blocks. These blocks can be rotated in order to help with readability. Furthermore, you can adjust the visual size of this block.
- Represent classes as blocks. Classes can contain multiple method and variable blocks and could be abstracted away. when a class is abstracted, it is represented as a new block that hides its members.
- Represent layers as pages. Layers can contain multiple classes, however, each layer is independent of other layers. Layers could be disabled or enabled.
- Represent input and output arguments as circles on method blocks
- Represent channels as vectors. A channel is a connection between an input and an output that describes how data flows between method blocks.
- Probe blocks that display what is being passed through channels. This probe can modify the data that is going through a channel while the program is running. Probes can also assert certain values causing the execution to stop if a certain assertion is violated.
- Represent outputs as blocks. This block represents the final outcome. This block also ends the program execution.
- Run function, compiles and executes the code in the graphical user interface. During this operation, probes allow monitoring and modifying of the values that are passing through their channels.
- Extract function outputs the corresponding Python file based on the designed system.
- Blocks menu window. This window contains a list of all blocks including layers, classes, methods, variables and probes.
- Drag and drop functionality. The user should be able to drag and drop blocks from the block menu to the Scene.

Performance Metrics

We desire to create a graphical user interface that translates user input into executable code. Our software must follow a simple flowchart design where each node and line drawn represents a line of code that compiles and outputs relevant data. For our software to be successful it must create an executable Python file in the background that can be extracted at the push of a button. To measure whether our solution's debugging functionality works properly we will observe how well it communicates errors to the user through alerts. Running a suite of tests will allow us to ensure that the interface catches and displays errors to the user and provides advice as to how to fix the errors. Our software needs to be easy to understand. By hosting weekly trials of our software with volunteers we will gather data through questionnaires on what workspace layouts provide a more comfortable experience for users.