TENSORSNAP: A BLOCK CODING INTERFACE FOR MACHINE LEARNING

A Research Paper submitted to the Department of Computer Science
In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

By

Eric Stein

April 28, 2022

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISOR
N. Rich Nguyen, Department of Computer Science

TensorSnap: A Block Coding Interface For Machine Learning

CS4980 Capstone Report, 2022

Eric Stein
Computer Science
University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
es7jyz@virgina.edu

***Abstract*—Machine learning (ML), one of the fastest-growing technologies of the past 30 years, has revolutionized and benefited humanity in countless applications, from self-driving cars to oncology [1 & 2]. However, developing with ML has requirements that are either hardware-related or knowledge-related; such requirements hinder the democratization of ML. This project aims to address this problem by producing TensorSnap, a block-based ML platform that allows users to learn about and develop with ML without the need for text-based programming or high-performance graphics cards (GPUs) or central processing units (CPUs). Future work can further the impact of TensorSnap by providing user tutorials and by streamlining the ML deployment process.**

I. INTRODUCTION

A. Background

One of the fastest-growing technologies of the past 30 years is ML [2]. ML allows for computer algorithms to "learn through data," by recognizing and leveraging patterns that lead to desired outcomes [1]. Applications of ML include those that are difficult to address through traditional programming and are better addressed through practice and repetition, such as self-driving cars, image recognition, natural language processing, and virtual assistants. In certain cases, ML-based solutions are more effective than human-based solutions; for example, Google developed an ML model that detects breast cancer accurately 89% of the time, whereas pathologists have a 74% accuracy. The ML market is expected to grow from $21.17 billion in 2022 to $209.91 billion by 2029.

However, using ML comes with numerous strict requirements that decrease its accessibility and, thus, democratization. Some such requirements are hardware-related. For example, training ML models often requires high-performance GPUs or CPUs, as well as a sufficient amount of memory (RAM) [9]. Such hardware requirements increase the barrier of entry to ML by requiring prospective developers to pay large sums of money before being able to develop. In low-income areas or third-world countries, the effects of such requirements are exacerbated.

Other requirements are knowledge-related. For instance, the most popular ML libraries, such as TensorFlow, require developers to have a text-based coding background [8]; yet, less than 0.5 percent of the world's population knows how to code [4]. Thus, to begin using ML, one must first learn a text-based programming language, such as Python or JavaScript. This requirement prevents those familiar with the theory of ML but unfamiliar with text-based programming from being able to independently develop models, thus making them reliant on text-based programmers and hindering prototype iteration.

B. Prior Art

This project builds upon the work in TensorFlow and Magenta. TensorFlow is an open-source ML library developed by Google Brain that enables the development, training, and deployment of ML models, including neural networks [8]. Magenta, also developed by Google, is an open-source collection of pre-built models used for generative tasks, such as neural style transfer and instrument transfer [6].

A project that allows users to interface with pre-built ML models using block coding is AI blocks [3]. This project, developed at the Massachusetts Institute of Technology, provides access to multiple pre-built models for certain tasks, such as text classification, speech-to-text, and text-to-speech. However, it does not allow users to perform common ML tasks that produce entirely new models, such as defining neural networks and fitting models to preprocessed data.

Only one project has been found that seeks to accomplish the goal of this project: to create a comprehensive block-coding interface to ML. This project, eCraft2learn, is being developed at Oxford University and aims to provide a block-based "interface to both AI cloud services and deep learning functionality" [7]. Though it does provide the ability to define custom neural networks, eCraft2learn has several ways in which it can be improved to increase its utility and bridge the gap between text-based ML and block-based ML. For example, it lacks access to some of the more complex options afforded by TensorFlow, such as model layer types and training metrics. Furthermore, eCraft2learn does not provide a convenient means of visualizing data or predictions in certain formats, such as image or audio. Combining such features with those in eCraft2learn

and AI blocks allows for the creation of a more complete block-based ML tool.

II. MATERIALS AND METHODS

A. Materials

The software used for this project include programming languages, libraries, frameworks, integrated development environments (IDEs), and cloud services. One programming language used was *Snap!*, a block-based programming language developed at the University of California Berkeley with the purpose of educating younger students about computer science. It was used in this project to create visual block-based interfaces for various ML-related tasks; the functionality of these blocks was developed using JavaScript.

The main library used for ML functionality was TensorFlow [8], and a library used to add pre-trained models was Magenta [6]. The main framework used was React.js, a user-interface (UI) development tool. The IDE used was Webstorm, a web development IDE published by JetBrains. The cloud service used was Firebase, a suite of cloud-related tools developed by Google that includes web hosting.

B. Methods

Developing TensorSnap involved following an iterative process with four steps. In the first step, high-priority TensorFlow functionality was identified for addition into TensorSnap. The first feature selected to port into TensorSnap was the ability to define, compile, fit, and use deep neural networks, a key facet of TensorFlow that affords a wide range of utility. Being able to define the architecture for a deep neural network requires the ability to customize the types of layers in the network, their parameters, and their order relative to each other. Similarly, compiling a model is a highly-customizable process. Fitting, or training, a model requires the ability to import, create, preprocess, and/or restructure data. Similarly, using a model to predict values or make decisions requires flexibility with formatting input data. These requirements were kept in mind when moving onto the next step in the iterative cycle.

In the second step, user-friendly interfaces were designed in *Snap!* Doing so involved creating custom blocks that accept various parameters and user input. In the case of adding deep neural network functionality, to allow for a high degree of flexibility without overwhelming the user, many dropdown-style inputs were used [Fig. 1]. Furthermore, to support modularity and allow users to differentiate between different blocks and their purposes, certain parameters were defined as individual blocks. For example, in the case of deep neural networks, metrics and layers took the form of individual blocks; visually, this allows users to "construct" models by dragging layers into a model definition block (such as "sequential") rather than typing out layer names.

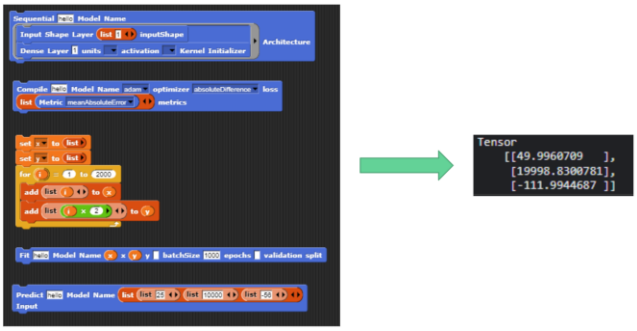

Fig. 1. Linear regression in TensorSnap.

In the third step, the functionality for the previously-designed interfaces was implemented using JavaScript in the internal code of the TensorSnap web app. This implementation followed an object-oriented approach and used the TensorFlow.js library. In most cases, such implementations involved the following approach: (1) user input from *Snap!* is processed and reformatted as necessary, (2) the internal state of TensorSnap is consulted and updated as necessary (e.g., currently loaded models and tensors/data), (3) the appropriate TensorFlow.js functions are called, and (4) output is produced and displayed to the user. In the case of deep neural networks, model and data storage was abstracted away for the user, allowing them to refer to models and tensors by name in their blocks.

Finally, in the fourth step, the interfaces and corresponding implementations in the previous steps were connected. To do so, a small snippet of JavaScript code was added directly to the *Snap!* blocks to allow them to call the appropriate JavaScript functions in the TensorFlow web app. Certain *Snap!* blocks, such as the layer blocks in the deep neural network example, were processed as input by the corresponding JavaScript implementation and thus did not provide any functionality on their own.

In addition to the iterative approach used to add core TensorFlow functionality into TensorSnap, a similar approach was followed to add various pre-built generative models from Magenta. One such model, a neural style transferrer, allows users to apply the style of one image onto another without having to define and train their own model [Fig. 2]. Another model transforms a given audio file by playing the same sounds using one of four instruments: flute, saxophone, trumpet, and violin.

Fig. 2. Neural style transfer in TensorSnap.

Moreover, several entirely new blocks were developed to increase modularity and usability. These include utility blocks that generate image files or audio files when given raw data. This allows users to decide how they would like to output the predictions of their models or visualize their input data. It also helps conceptualize the

various stages of ML (e.g., data preprocessing, training, prediction, output) and allows for the development of a highly customizable ML pipeline.

III. RESULTS

TensorSnap is now a fully-functioning block-coding window into TensorFlow and the world of ML. The custom blocks allow for the development of highly-configurable deep neural networks, transformation of data, and more. It is currently hosted at https://tensor-snap.web.app/

Several sample projects were developed to showcase the use cases of TensorSnap. One such project illustrates how to perform linear regression [Fig. 1], a common ML use case in which various parameters are used to predict a numerical value, such as the price of a house. In this example, a simple sequential model with a single dense layer is defined, compiled, trained, and used to predict the y-values for the equation y=2x. When trained, the only (x,y) pairs supplied were those in which x was between 1 and 2000. As seen in Fig. 1, the resulting model is able to make accurate predictions when supplied with input that was not in the training set.

Another sample project showcases the pre-built neural style transfer model [Fig. 2]. This project consists of four blocks: one to import the image to transform and convert it into a tensor, one to import the style image and convert it into a tensor, one to perform the neural style transfer (and save the result to a tensor), and one to generate an image file from the model's output.

A third sample project showcases the pre-built neural instrument transfer model [Fig. 3]. Similarly, four code blocks were used in this project: one to convert an audio file (imported using default *Snap!* functionality) into a tensor, one to perform the neural instrument transfer, one to play the audio of the result in the browser, and one to generate an audio file of the result.

Fig. 3. Neural instrument transfer in TensorSnap.

Finally, a fourth sample project illustrates TensorSnap being used to perform classification on the MNIST dataset [Fig. 4], a popular ML dataset containing hand-drawn images of numerical digits. First, a sequential model with five layers is constructed and compiled. Next, a dataset is generated by taking a subset of the MNIST dataset. After that, the model is fit to the data. Finally, to test the model, the user can either generate more datasets from the MNIST dataset or can draw a number in *Snap!*'s "stage" area.

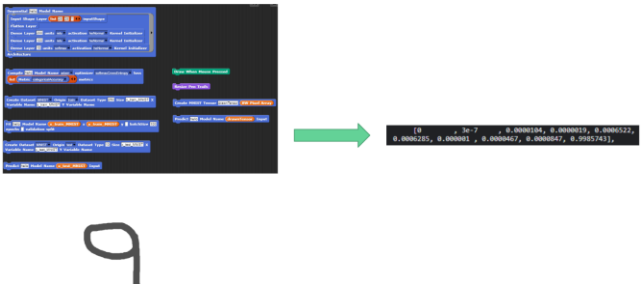

Fig. 4. MNIST classification in TensorSnap.

IV. DISCUSSION

A. Challenges

The biggest challenge in the development of TensorSnap was balancing the need for simplicity and the need for usefulness. Simplicity was required because the target audience of TensorSnap consists of non-programmers, programming novices, or ML students. Usefulness was required so that TensorSnap would be a viable block-based alternative to TensorFlow and other text-based ML tools. However, exposing TensorFlow's functionality in a block-based form without overwhelming the user required a large amount of planning and iteration.

Several decisions were made to promote both simplicity and usefulness. One decision was to allow for a large degree of customization by exposing many parameters for common TensorFlow operations. However, users were not required to fill each of these parameters; if certain parameters remained unfilled, the implementation would use default values. Thus, users have access to advanced functionality but are not required to use it to perform ML operations. Another decision was to split blocks with a large amount of functionality into smaller blocks so that each block would only be responsible for a single operation. This allows users to separate their code into multiple blocks, understand the purpose of each block, and combine blocks to create more advanced ML programs.

Another major challenge was ensuring that each of TensorSnap's major dependencies – *Snap!*, TensorFlow.js, Magenta.js, and node.js (the JavaScript engine used) – were compatible with each other. Initial incompatibility issues resulted in errors when running certain functions, such as Magenta's neural instrument transfer. After experimenting with different combinations of versions for each of these dependencies, compatibility was obtained.

B. Connection to Coursework

The most relevant course at the University of Virginia for the development of TensorSnap was Machine Learning. By teaching both the fundamentals of ML and how to perform ML using TensorFlow (in Python), this course provided the background and experience necessary to design an ML interface and perform ML using TensorFlow.js.

Designing and developing a new, modular tool for ML also had major implications in Software Development Essentials. This course taught about the importance of modularity, high cohesion, low coupling, and more. These principles were used both when designing TensorSnap's block interface and when implementing the interface in the JavaScript of the TensorSnap web app.

Finally, Human-Computer Interaction (HCI) offered an introduction into user experience (UX) design and the design thinking process. Knowledge from this course assisted in designing intuitive interfaces, offering useful error messages, developing a prototyping iteration cycle, and more.

C. Future Work

Though the major goals of TensorSnap have been achieved, work remains to be done to improve its utility and usability. To improve utility, additional TensorFlow functionality can be exposed by creating new blocks and corresponding implementations that use TensorFlow.js. Furthermore, new blocks that assist in deploying models produced in TensorFlow – such as by exporting models as files or hosting them online via custom application programming interfaces (APIs) – can be created. Such blocks would further decrease the barrier of entry to ML by allowing non-programmers to seamlessly put their models into production and connect to their models from a number of other applications.

To improve usability, future work can add "help" messages directly into the *Snap!* interface, thereby offering immediate explanations about the purpose of each block. Similarly, a tutorial can be created that teaches users how to use TensorFlow for a variety of tasks and offers example code for each task. In addition, error messages and model output can be added and refined to better indicate the reason for errors and the internal state at different stages of the ML process.

Consequently, TensorSnap now provides a foundation upon which a more powerful and robust block-based ML development platform can be created. With the proper improvements, TensorSnap will be better able to achieve its goal of expediting the democratization of ML.

V. ACKNOWLEDGEMENTS

Assistant professor N. Rich Nguyen for assembling TensorSnap's team, defining its vision and major project goals, and guiding its development.

Professor Glen Bull and MakeToLearn lab manager Jo Watts for showcasing TensorSnap in the University of Virginia's "Designing Art, Music, and Games" course.

VI. REFERENCES

[1] "45 Statistics, Facts & Forecasts on Machine Learning [2022]," Oct. 25, 2020. https://research.aimultiple.com/ml-stats/

[2] K. D. Foote, "A Brief History of Machine Learning," DATAVERSITY, Dec. 03, 2021. https://www.dataversity.net/a-brief-history-of-machine-learning/

[3] R. Williams, "AI Blocks," MIT Media Lab. https://www.media.mit.edu/posts/ai-blocks/

[4] "Coding in 2016 is like reading in 1816," Prenda, Oct. 19, 2016. https://www.prendacodeclub.com/blog/coding-2016-like-reading-1816/

[5] F. B. Insights, "Machine Learning Market Size [2022-2029] Worth USD 209.91 Billion | Exhibiting a CAGR of 38.8%," GlobeNewswire News Room, Apr. 04, 2022. https://www.globenewswire.com/news-release/2022/04/04/2415724/0/en/Machine-Learning-Market-Size-2022-2029-Worth-USD-209-91-Billion-Exhibiting-a-CAGR-of-38-8.html

[6] "Magenta," Magenta. https://magenta.tensorflow.org/

[7] "Making deep machine learning neural nets." https://ecraft2learn.github.io/ai/AI-Teacher-Guide/chapter-6.html

[8] "TensorFlow," TensorFlow. https://www.tensorflow.org/

[9] "What to consider when choosing a laptop for machine learning," Zindi. https://zindi.africa/learn/what-to-consider-when-choosing-a-laptop-for-machine-learning