An Online Graphical User Interface Application to Remove Barriers in the Process of Learning Neural Networks and Deep Learning Concepts Using Tensorflow

Justin Li[1] and Yu Sun[2]

[1]Troy High School, 2200 East Dorothy Ln, Fullerton, CA 92831
[2]California State Polytechnic University,
Pomona, CA, 91768, Irvine, CA 92620

Abstract

Over the years, neural networks have become increasingly important and complex due to the rising popularity of artificial intelligence technologies. It allows for complex decision prediction making, and is an essential part in the modern AI industry. However, due to the complex nature of neural networks, a lot of complex math and logic has to be well understood along with a proficiency in programming in order for one to make anything practical with this technology. This is unfortunate, however, that many do not have the required high level math skill, or the proficiency in coding, blocking a lot of people from reaching and experimenting with this technology. My method attempts to eliminate the complexity that developing neural networks bring, and bring a clearer picture of what the user may be creating and working with. With the help of modern web technologies such as JavaScript and tensorflow.js, I was able to create a GUI program that can create, train, and test a neural network right on a browser, and without writing any code with a comparable result [13].

Keywords

Neural network, deep learning, CNN.

1. Introduction

From self-driving vehicles to advances in healthcare applications, Deep Learning has been revolutionizing today's society [2]. For its ability to learn from large amounts of unstructured and unlabeled data, Deep Learning possesses the capability to perform complicated tasks such as driving, translating, and even performing image recognition [4]. And as the goal of Deep Learning is to simulate the process of a learning human brain, a multi-layered Neural Network is used at its core [5]. It functions as universal function approximators, which allows it to be trained for any circumstances given an appropriate set of input and output data-set. This is also the main reason for the popularity and potential of Deep Learning, as its flexibility allows for it to adapt to real world situations [8]. Such flexibility enables Deep Learning to automate jobs never thought was possible before, such as self-driving and image-colorization [3]. Fields such as Healthcare even started adopting this technology to create diagnosis for Breast Cancer based on related data [1].

However, all these benefits come with a catch, and that is that Deep Learning involves complex math knowledge such as Multivariate Calculus, Linear Algebra, and Statistics. Knowledge such as Algorithms and Programming are also required in order to implement Deep Learning [7]. However, as the data shows in an article, in 2019, there are only 23.9 million out of 7.71 billion who are programmers and software developers worldwide, which is less than 1% of the population. Because technologies such as weather predictions depend on this sort of technology, the result of the incompetence in this field will result in less innovation in this field, thus technologies such as smart stock trading will cease to improve [11].

Some preexisting softwares and applications have allowed users to create and view their neural networks visually <evidence needed>, but most have confusing layouts, complicated user interfaces, or lack the ability to create complex neural networks with custom data entries. Tools such as the Tensorflow Playground <source needed>, only allows the user to choose from a defined set of training data. This prevents the user from testing the architecture's efficiency on real world data that the architecture may be used on. These types of implementation pose a limit for the user on their data variability, neural network complexity, and usefulness in general. A second problem with these software is that they generally have very complex and hard to understand user interfaces, making it very hard for beginners to experiment with and use. Static user interfaces, such as the one from Tensorflow, limits the user to the type and complexity of neural networks they can create, and thus making it impossible for more advanced users to create more sophisticated architectures [14].

Taking the pros and cons of previous methods into consideration, our goal is to create a simple, easy to use, modular, and highly expandable software for creating neural networks visually. With that in mind, our implementation features a minimalism design, a highly expandable and easy to use layout, as well as the ability to run directly on a web browser. Compared to existing methods, our method is a lot cleaner and organized, while being highly functional and expandable.

In two application scenarios, we demonstrate how the above combination of features increases the experiences and speed for users to use this app. First we show the ease of use and functionality of our method through a comprehensive application test. Second, we compared the speed at which a user can create a functional and well performing neural network with our method, previous methods, and traditional method.

The rest of the paper is organized as follows: Section 2 provides the details on challenges that I encountered during design and development; Section 3 focuses on the details of my solution and well as the solution to the problems described in Section 2; Section 4 presents the relevant details about the experiments regarding the solution, comparing it to older methods and alternate methods; Section 5 gives more details on the alternate methods that was used to compare to my method. Finally, Section 6 gives the conclusion as well as planned future works on this project.

2. Challenges

In order to build the project, a few challenges have been identified as follows.

2.1. Parsing custom data

Letting the user choose their own data to train the neural network on is a very useful feature, as it allows for a wider and more expansive training environment [9]. However, it is not always clear if the user's data structure will match the neural network's input structure, or is even a valid dataset. Therefore checks will have to be performed on both the network and the neural network to ensure no failure occurs during training.

2.2. Creating a usable, minimalistic user interface

User interface is a crucial part of our method, as it is aimed towards beginners. And while minimalistic user interfaces are great at being easy to use and simple to understand, they often lack functionality, or take up too much space. The way of expression is also important, as we need to represent an abstract idea of a neural network, a system of matrix dot products and vectors, into a graphical visual that is accurate, easy to understand and customizable. Other aspects such as tutorials and controls have to all be as intuitive as possible.

2.3. Making a custom architecture that is trainable

Training a neural network well can sometimes be the hardest part of AI development. Things such as training hyper-parameters and activation have to be chosen wisely for a good performing network, and those are dependent on the data. These hyper-parameters can be chosen through calculations and parameters, but are often left to the user to decide, which can be quite daunting to beginners. Other things like training optimizer and loss also influences the network's performance, and using the wrong ones can have devastating effects on the training's outcome.

3. SOLUTION

Through the interaction with the graphical neural network representation, the different aspects and features of the network are read in and interpreted to create a tensorflow neural network that depicts exactly what is shown on screen. During the network creation, the user interacts with the user interface, manipulating their network architectures and uploads their training input and outputs. The graphical network is then interpreted, and a tensorflow neural network of equivalence is created automatically. During training, the program checks the validity of the input and output shapes of both the neural network and the training data. After the check is complete, training begins, and the program reports the training progress to the user graphically. After training, the final loss for the neural network is shown.

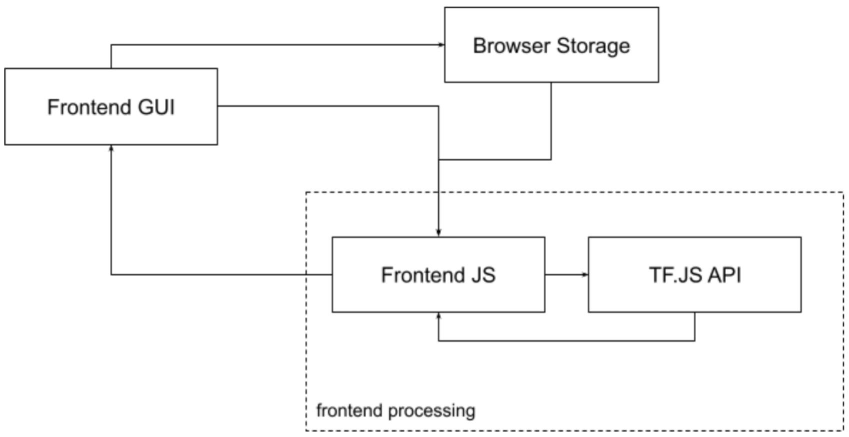
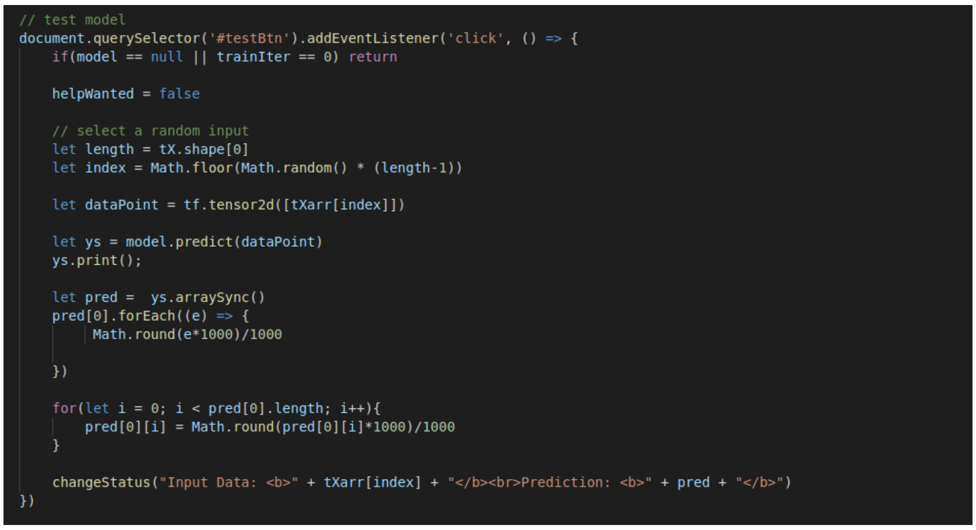


Figure 1. Overview of the system



```
// test model
document.querySelector('#testBtn').addEventListener('click', () => {
    if(model == null || trainIter == 0) return

    helpWanted = false

    // select a random input
    let length = tX.shape[0]
    let index = Math.floor(Math.random() * (length-1))

    let dataPoint = tf.tensor2d([tXarr[index]])

    let ys = model.predict(dataPoint)
    ys.print();

    let pred =  ys.arraySync()
    pred[0].forEach((e) => {
         Math.round(e*1000)/1000

    })

    for(let i = 0; i < pred[0].length; i++){
        pred[0][i] = Math.round(pred[0][i]*1000)/1000
    }

    changeStatus("Input Data: <b>" + tXarr[index] + "</b><br>Prediction: <b>" + pred + "</b>")
})
```

Figure 2. A segment of the code

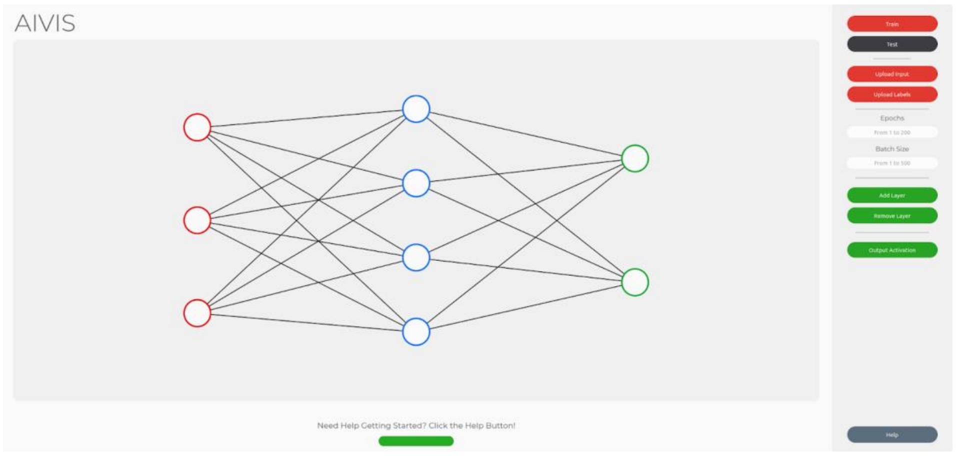


Figure 3. Screenshot and UI

In order to create the graphical interface to be scalable and user friendly, I used HTML CSS to style the interface, as well as JavaScript to provide its functionality. The user interacts with the HTML + CSS site, and uploads their input and label datasets. When the datasets are uploaded, they are stored in the browser's local storage for the ease of access and modification. When the train button is clicked, the application attempts to create and compile the neural network in TFJS based on what the user defined on the front-end. It then automatically partitions the data-set into training set, validation set, as well as testing sets. The program then uses the TFJS library to train the neural network, and sends analytic info to the front-end after every epoch, things like training and validation losses. After training is complete, the user can then choose to test the network by pressing test, which then the program will select a random data-point from the testing data-set, and run it through the neural network, and send back the output.

4. Experiment

4.1. Experiment 1

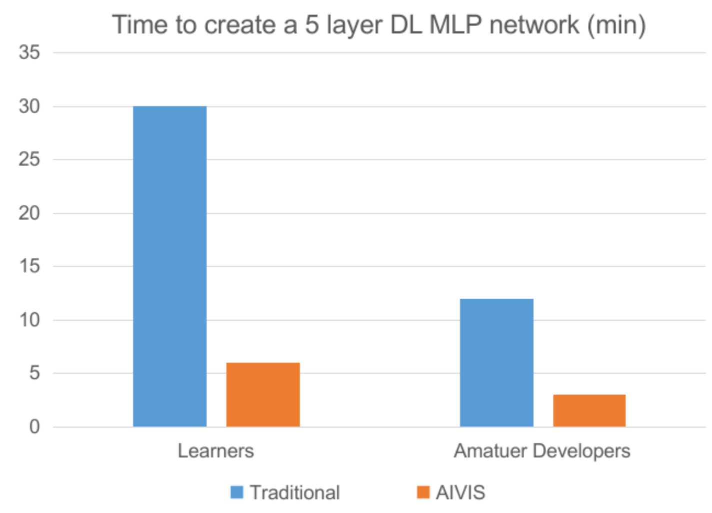
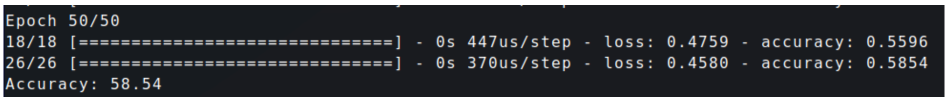


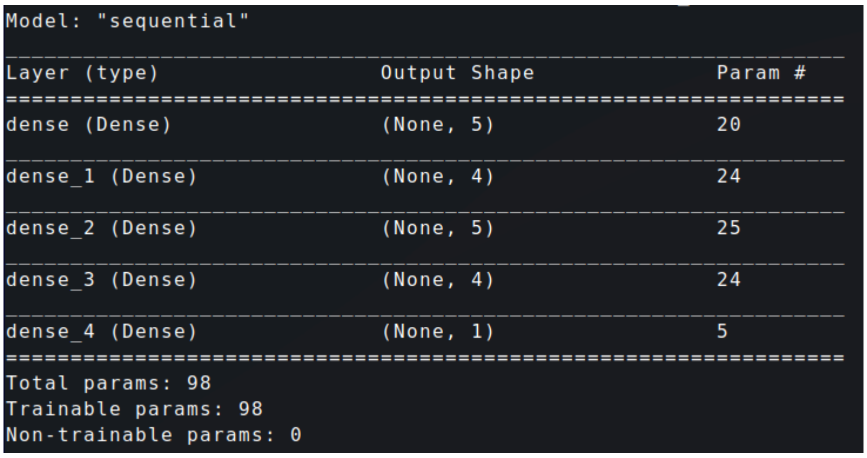
Figure 4. AIVIS is better for speed prototyping for both amateurs and beginners

My solution proves that the output resulting from this method is comparable, and even sometimes better than traditionally made neural networks.



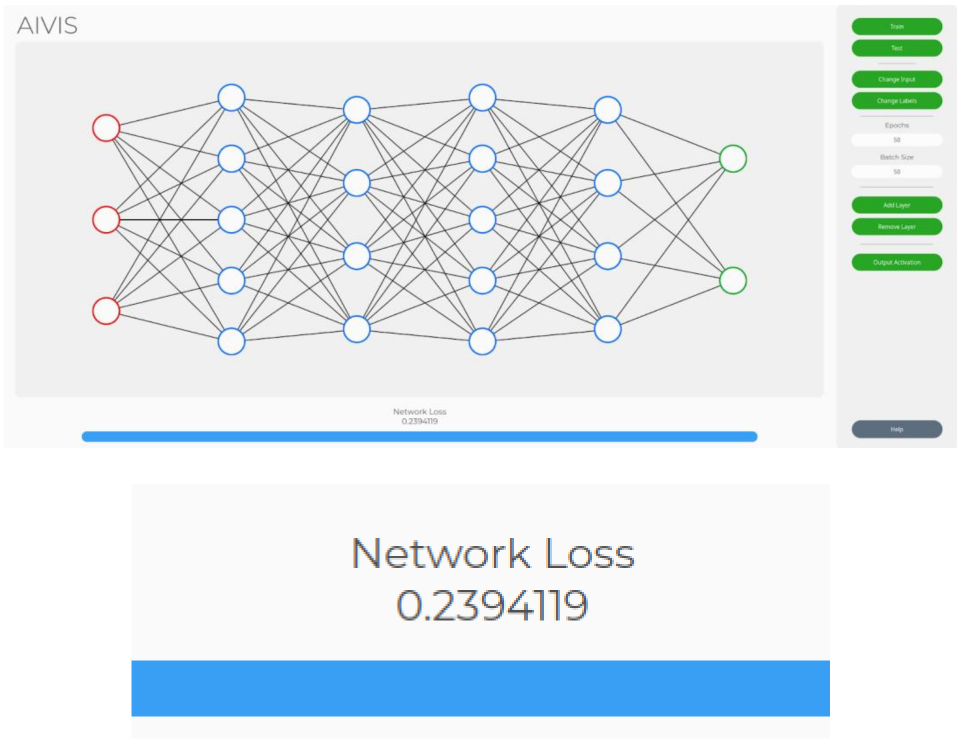
```
Epoch 50/50
18/18 [==============================] - 0s 447us/step - loss: 0.4759 - accuracy: 0.5596
26/26 [==============================] - 0s 370us/step - loss: 0.4580 - accuracy: 0.5854
Accuracy: 58.54
```

Figure 5. Final loss of traditional network after 200 epochs and 25 batch size



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 5)                 20
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 24
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 25
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 24
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 5
=================================================================
Total params: 98
Trainable params: 98
Non-trainable params: 0
```

Figure 6. Neural network structure of traditional network



Epochs 50
Batch Size 50
Network Loss
0.2394119

Figure 7. Final loss of the model trained with AIVIS, with the same hyper parameters

After about 50 training cycles with the same training parameters, data, and architecture, the average loss of the model trained with our method was about 30% lower than traditional. The optimizer, loss, and activation in each model is the same, but somehow the training results show that the visual made neural network performs better overall than the traditional one. Testing the same neural network out, it seems that AIVIS was better at avoiding overfitting, as the output of the neural network trained in AIVIS had more variety than that trained traditionally.

4.2. Experiment 2

Our solution not only dramatically decreases the amount of time to create neural networks, it also simplifies the steps needed to get a network working.

experiment settings: indicate that experiment design is scientific.

1. 1 participant
2. Has knowledge in neural networks and coding

The average amount of steps required to create and train a neural network in our solution is around 20-30 steps, depending on how complex the neural network is. However, when coding in python with Tensorflow, the user needs around at least 70 lines of code, and also needs to prepare the data, including partitioning and prepossessing it. Overall, the much simpler design and nature of our solution results in a faster development speed, as well as less room for technical errors.

4.3. Experiment 3

Even though our solution can offer a wide variety of neural networks, AIVIS can only create fully connected layers as of the moment, with many limitations on it due to graphical inadequacy.

Our solutions can solve simple problems such as predicting the best color of text to go above a colored background, XOR classifications, and other simpler problems. However, when more neurons are needed for a specific problem such as MNIST, our solution is currently incapable of doing so, as our graphical user interface does not allow for such a big network.

The experiments showed that AIVIS is a viable way to create neural networks graphically, with a much faster prototyping time and a better result than traditional neural networks. This is due to the minimalist nature of the program, where the user can easily and quickly change hyper parameters and train over and over again.

The result was surprising, as I was not expecting the graphical method to outperform the traditional method with the same architectures and training parameters. A possible source to this difference may be in how the code was run, and also how the weights are initialized. But currently with my code analysis, there is no difference between the two methods' way of weight initialization.

5. Related Work

Neutron is a program that allows users to visually see their neural network in a node based fashion. It has a simple user interface and works with a variety of neural network types. It's way of representation is modular and dynamic, which is something our method needs to improve on. However, Neuron lacks the ability to modify or create neural networks.

Tensorflow Playground is an online application that allows users to tinker with neural networks of different sizes without hassle.

Neuron is a program which aims to solve regression, time series, binomial and multi-nominal classification problems for businesses and professional needs. It has a way to visualize data that the user is working with, and is suited towards companies which need rapid prototyping. However, Neuron is not an open source software, and is more aimed towards big businesses and professional work, and therefore does not suit an average user or student very well. Even though AIVIS may not perform as well in either functionality, efficiency, or scalability as Neuron, it is more education oriented and meant for small scale development. And while the free version of Neuron lacks the ability to export trained neural networks, AIVIS is planned to add that feature in a later version.

6. Conclusions

The goal is to create a simple, light weight, and easy to use GUI application for creating neural networks in order to mitigate the flaws of traditional neural network development. And when comparing the result of our method and traditional methods, our method has shown to be over three times faster in creating and training neural networks, while retaining network performance and accuracy. Our method shows that neural networks can be created and trained faster with comparable, and sometimes better, performance than traditional methods. As the use of a graphical interface provides a more intuitive and simple usage experience.

The current limitation of our method is that neural network training is not optimal due to the lack of customizability in hyper-parameters [15]. Users don't have much choice and selections over how the networks get trained, when to stop training (early stopping), what gradients to use, etc… Our method also lacks scalability due to its limiting UI design, making it more suitable as a demonstration tool than a real developer tool. The training speed and network complexity is also extremely lacking, as the neural network is trained on the client's side instead of on a dedicated server. This compromises the amount of complexity the neural network can be, as well as limiting the training speed by a wide magnitude.

As the user interface is not very well designed in this solution, our future work will feature a better, more modular, and more scalable interface; allowing for more complex neural networks to be made. Training will also be moved to a back-end dedicated server, which can potentially increase training and compiling speed by 300 to 700% depending on the network's complexity. Features such as exporting an integrate neural network file, such as TensorFlow's h5 files, will also be included for a more integrated and streamline workflow [6].

REFERENCES

[1] Steiner, David F et al. "Impact of Deep Learning Assistance on the Histopathologic Review of Lymph Nodes for Metastatic Breast Cancer." The American journal of surgical pathology vol. 42,12 (2018): 1636-1646. doi:10.1097/PAS.0000000000001151

[2] Daniels, Norman. "Justice, health, and healthcare." American Journal of Bioethics 1.2 (2001): 2-16.

[3] Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful image colorization." European conference on computer vision. Springer, Cham, 2016.

[4] Honneth, Axel, and Avishai Margalit. "Recognition." Proceedings of the Aristotelian society, supplementary volumes 75 (2001): 111-139.

[5] Piramuthu, Selwyn, Michael J. Shaw, and James A. Gentry. "A classification approach using multi-layered neural networks." Decision Support Systems 11.5 (1994): 509-525.

[6] van Der Aalst, Wil MP, et al. "Workflow patterns." Distributed and parallel databases 14.1 (2003): 5-51.

[7] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

[8] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444.

[9] Hu, Yu Hen, and Jeng-Neng Hwang, eds. "Handbook of neural network signal processing." (2002): 2525-2526.

[10] A. G. Salman, B. Kanigoro and Y. Heryadi, "Weather forecasting using deep learning techniques," 2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS), 2015, pp. 281-285, doi: 10.1109/ICACSIS.2015.7415154.

[11] Molina, Gabriel. "Stock trading with recurrent reinforcement learning (RRL)." CS229, nd Web 15 (2016).

[12] Feindt, M., and U. Kerzel. "The NeuroBayes neural network package." Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 559.1 (2006): 190-194.

[13] Smilkov, Daniel, et al. "Tensorflow. js: Machine learning for the web and beyond." arXiv preprint arXiv:1901.05350 (2019).

[14] Abadi, Martín. "TensorFlow: learning functions at scale." Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming. 2016.

[15] MacKay, David JC. "Hyperparameters: optimize, or integrate out?." Maximum entropy and bayesian methods. Springer, Dordrecht, 1996. 43-59.