Team 3 Project 2 Final Report

Ben Hawn: Neural Network Implementation, Liuyi Jin: Back-Propagation Programming,
Colin Legge: JavaFX GUI, Bug Fixing

CSCE 315-100

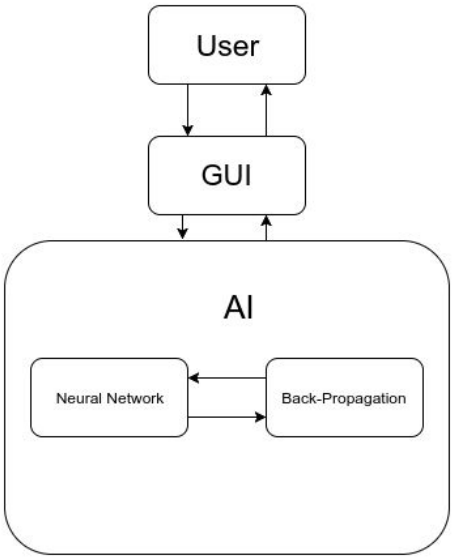
Design Document

Section 1: Purpose

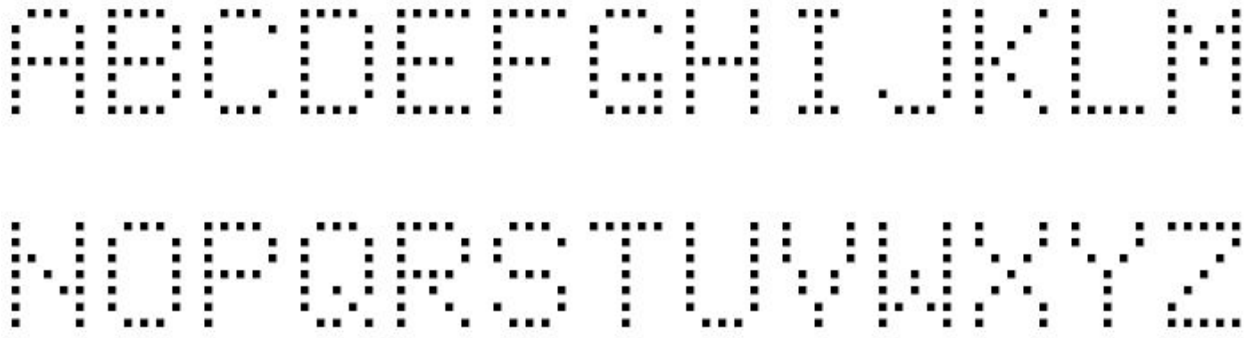
The purpose of our project is to be able to create a program that will enable a computer to be able to tell what letter is being displayed when given an input of a 5x7 dot matrix font. This is a problem that could streamline transcription from physical media to digital media if solved. The end goal is to be able to achieve this result both quickly and accurately.

Having a program that could take a scan of a page and then convert that to a text document makes preserving information and backing up data far faster and less monotonous than doing it manually. Many companies, especially those that rely on physical records as well as those stored on servers, would greatly benefit from being able to scan forms into their computer and have it automatically stored without them needing to go through the painstaking task of replicating the form digitally themselves.

Section 2: High-Level Design



Based on the above chart representing our program, the GUI takes user input, which is then passed to our AI program. The neural network and the back-propagation code then interact with each other to determine best determine what letter might have been input by the user. Upon completion, the letter that the AI thinks the user input into the GUI will then be output into the GUI. Our programming utilizes the following 5x7 dot matrix font as metric for determining what letter was input.



Section 3: Low-Level Design

Neural Network

Usage

Neural networks, or a neural net for short, is a structure of interconnected nodes called "neurons" that can be layered. These neurons take weighted inputs where each weight signifies the strength of connections in the neural net. Each neuron computes the weighted sum of its inputs and uses what is called an activation function (our sigmoid function) to determine the neurons outputs. What this means for the overall neural network is that this data structure can perform a type of machine learning using a set of given training data to manipulate the weights through backpropagation (talked about in later section) to recognize certain inputs means certain outputs. In short, the neural net can learn to identify that certain inputs mean certain outputs. (Russel and Norvig pg. 728)

Currently I am planning on making my own neural network structure rather than using a package which will use the backpropagation algorithm to train it. I plan to make the

neural network structure by writing a class NeuralNetwork which is made up of a Node/Neuron class I write. I am hoping to be able to write it so that I will be able to change the number of layers the neural network is made up of (at least up to the required 2-layers).

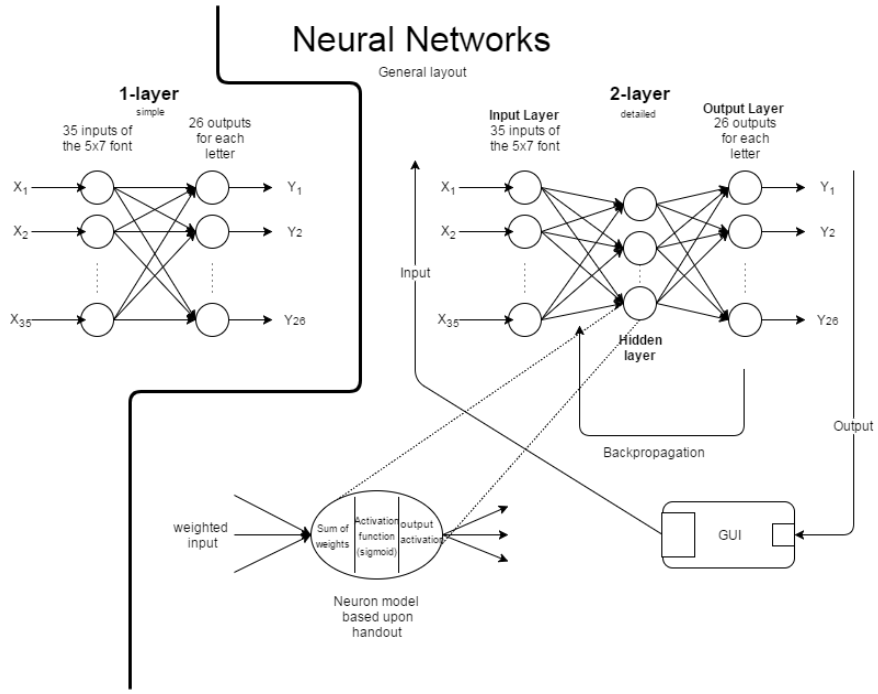
I believe we would benefit from designing our own neural net by having more control over our code and having a greater understanding of what's going/how everything works. There would be risks involved in this though, such as our neural network may be more prone to errors, have more work to do, and not be as optimized as compared to some neural net package.

This is the plan for what I will start working towards though this could be subject to change and we may use some neural net package if we encounter too many problems.

Configuration

As I mentioned earlier in the usage section, I hope to be able to write my code so that we can specify the number of layers in the neural net (at least up to the required 2-layers). This may or may not be a configurable option if we manage to implement it.

Model



The above image shows two models a 1-layer and 2-layer:

- The 1-layer model above show a general view of what the neural net will look like without going into too much detail. It shows the 35 inputs representing our 5x7 text linked to our 26 outputs.
-
- The 2-layer model goes into more detail of how the neural network works. It shows a detailed view of what the neurons will function and how it will hook into the other sections of the code.

Interaction

The 2-layer neural net shown above in the model section shows how it will interact with other sections of our code.

- With the backpropagation section:
  - The neural network part of the code will interact with the backpropagation section to train the neural net.
  -
- With the GUI section:
  - The neural networks input will be received from the GUI and the neural networks output will be displayed on the GUI.

Back-Propagation

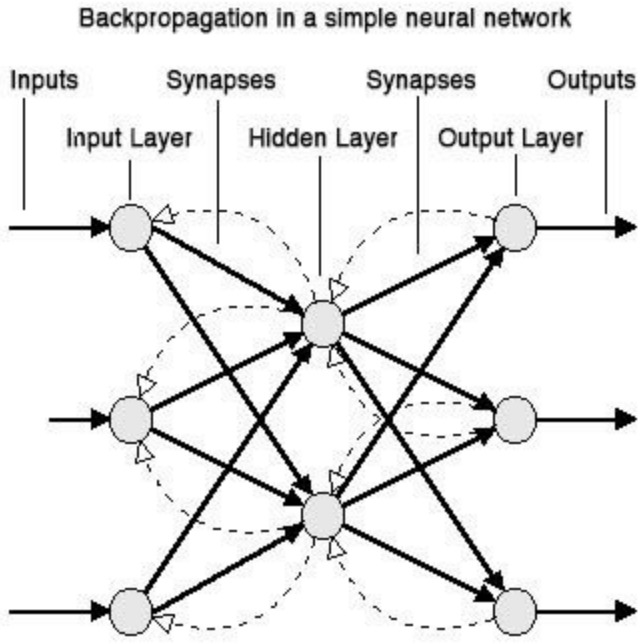
Usage

Back-Propagation algorithm is actually the most popular training algorithm that is currently used to train artificial neural network. It is preferred over other algorithms due to its efficiency and accuracy in terms of the output results.

Configuration

Since the back-propagation is pretty popular, so there are several variations of the pseudo-code of this algorithm. As indicated by professor in the class, the available pseudo-code would be used to construct final java code.

Model



The back-propagation (BP below) serves to take the previous neural networks weights and output as input to update our weights further for next time step training process.

Interaction

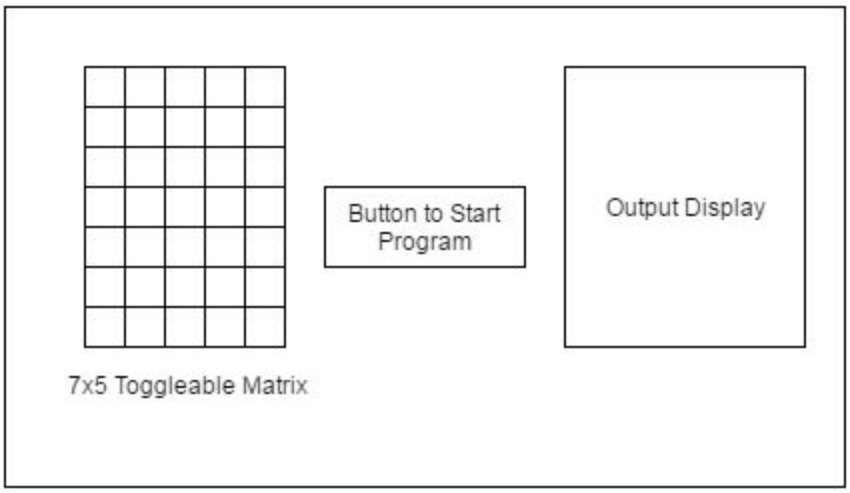
1. BP algorithm needs to take my teammate's neural network's final output and previous time-step weights
2. This procedure would certainly be employed in the process of GUI code development

GUI

Usage

My plan for the GUI is to contain a 7x5 matrix of squares or circles that will have a togglable fill color that will either be white or black. This matrix will be used by the user to simulate a 7x5 dot matrix font. There will also be a button that will be pressed after the matrix configuration has been set and the user wants the program to guess what letter they are attempting to display.

Model



The above model shows a rough representation of what we plan for the final GUI design to look like.

Interaction

The GUI will make logical 1 outputs from nodes in the matrix that are filled black, and logical 0 outputs from nodes that are filled white. These outputs will be passed to our AI program for generation of a prediction of what letter it believes the user input by toggling values in the GUI. The results of the combined neural network and back-propagation code will be output and displayed in the GUI.

Section 4: Benefits, Assumptions, and Issues

Benefits

1. The GUI has a simple design with minimal buttons and should be self-explanatory to anyone using it.
2. The Back-Propagation algorithm is a quick and efficient neural network training algorithm that minimizes the error function produced.
3. Back-Propagation is relatively easy to implement.

4. The neural network would give us more control over our code which would allow us to have a greater understanding of its functionality.

Assumptions

1. A user will be able to easily figure out the matrix has toggleable buttons
2. The AI portion of the program will be able to calculate accurate results so that the GUI can display accurate results.
3. A 1-2 layer neural network will be used.
4. The neural network will be fully connected.

Issues/Risks

1. Integration of the different parts of the project might require some trial and error.
2. Finding the correct weight-initializing scheme to work with our code will take some trial and error as well.
3. Having lots of control over the neural network could result in it being more prone to errors.

Section 5: Running & Net Training Graphs

In order to run our program on build.tamu.edu, the following commands must be entered into the console in the following order when in the directory containing our code. It is also assumed that the argument for the `readstuff()` function in line 9 of `TrainedNeuralNet.java` is "`training2.txt`". This should be the case by default, but if it is not, then it could be an issue.

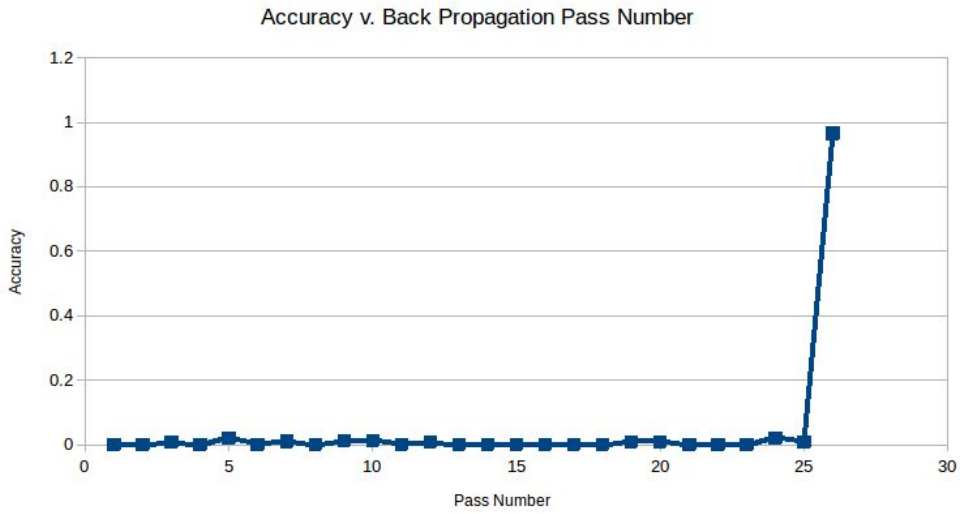
```
> javac -cp /usr/java/jre1.8.0_131/lib/ext/*:/usr/java/jre1.8.0_131/lib/*
*.java
```

```
> java GUI
```

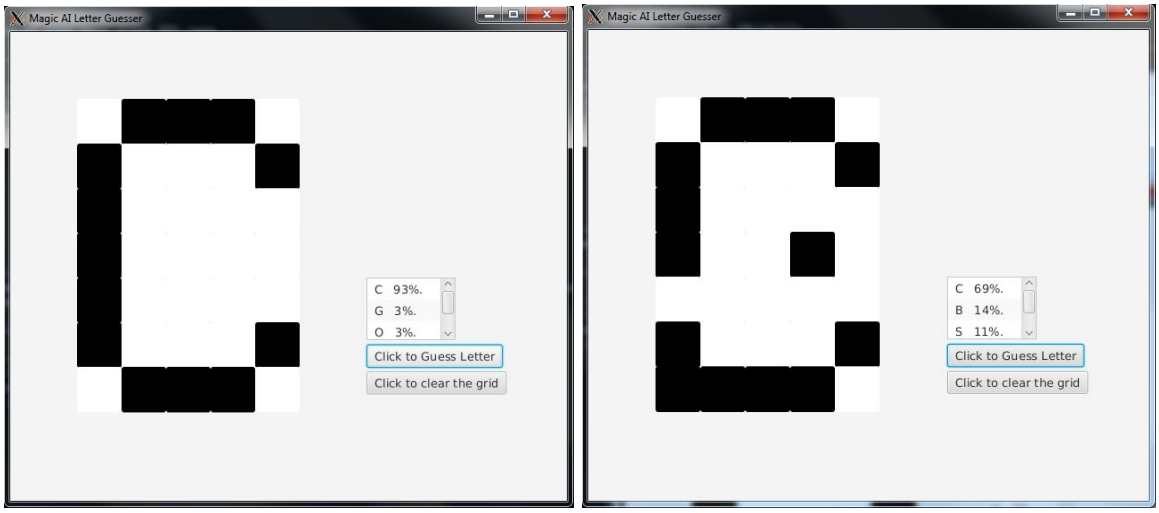
Assuming X11 Forwarding is enabled, then the program should run properly. There might be a couple libGL errors and a ES2 Prism error, but they do not affect the functionality of our program. To input a letter simply click on the white buttons of the matrix to make your letter and click on the "Click to Guess Letter" button. Then the top three letter matches corresponding to the input will be displayed along with a percent match.

Our approach to the program was to have a text file of the inputs of perfect matches to the letters of the font that would be used to teach our program what the letters of the alphabet were. Our program would then read this file and use our training algorithm with our back propagation and forward propagation algorithms in order to learn what inputs would be interpreted as which letter. Our first layer of neurons contains 32 neurons and our second layer contains 26 neurons, with each of these 26 neurons corresponding to a different letter of the alphabet. We would then take these results and display them to the user through our GUI.

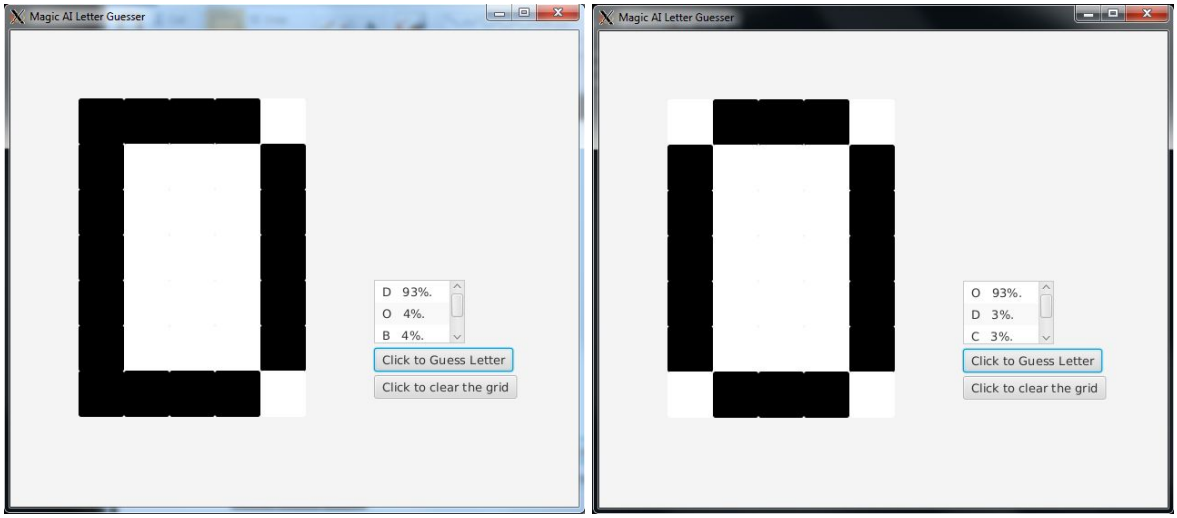
Below is a plot of the accuracy returned by our neural network plotted against the pass number of our back propagation algorithm.



The back propagation algorithm utilizes our training data and runs through it many times to learn what the representation of each letter is, until an accuracy value greater than 90% is obtained. As can be seen in the chart, a very low accuracy is reported through the first 25 passes, but on the 26th pass, our back propagation returns an accuracy of 96.47%. This data was generated by writing the accuracies for each pass to a csv file. This is indicative in the outputs that our AI program returned when guessing what letter we input. Examples of different outputs are shown below.



As shown above, our AI is still able to identify a letter, even if inputs are removed and added to the matrix. It has a less definitive match to a C, but it is still by far the best match. Similarly, our AI does not get confused between similar letters such as D and O. It is able to perfectly differentiate between these two letters despite them only being two inputs apart. However, in this case, both of the inputs return very high and almost perfect matches.



Section 6: Post-Production Notes

Throughout the course of designing and programming our AI, we came across several issues that resulted in a change in the design of our program. The biggest issue was the integration of the different parts of our project. Because the project was separated into 3 parts, each with their own inputs and outputs, it took a bit of work to make it so that our

data were the correct types or structures in order to be passed to different parts of the program.

Another change in design was addressing the nature of our sigmoid function. It simply was not accurate enough and resulted in values that ended up way outside the specified bounds of the assignment. Some more parameters were added to the sigmoid in order to achieve a more accurate guess of the letter.

There were no inherent difficulties found in constructing the GUI, and the only major hurdle in incorporating our neural net was the aforementioned change to our sigmoid function. Aside from that, the majority of our programming was in the form of trial-and-error going from task to task until our AI program was realized. Overall, this process could have been improved by designing more in the pro-production phase of the assignment, as well as collaborating more on the integration of the various parts of the program before it became necessary to integrate them. Having a plan beforehand would have saved us considerable time, and is something that we will strive to incorporate into future projects.

Each team member put in their fair share of the assignment. Colin created the GUI and assisted in bug-fixing of test cases. Ben worked to implement the neural network as well as doing small work on the GUI. Liuyi wrote the back-propagation code as well as assisting in fixing the sigmoid function. Beyond this, each team member was also involved in programming each part of the AI. There was not an aspect to the program where each member did not contribute in some way. As such, workload percentages can be represented as follows:

Colin: 33% Ben: 33% Liuyi: 33%

Our program could be improved by training our data with different variations of each letter. If there is more than one representation of each letter, our program might be better at determining which letter was input. Also, our training algorithm loops 700 times, which is fairly high. We could have done more research into what loop would be optimal and would result in accurate guesses by the AI and also not overtraining the AI. Also, the program could be extended to include lowercase letters as well.

Working on this project taught us how neural networks functions and the algorithms behind simulating intelligence. Additionally, but definitely not unimportant, was our learning to cooperate as a group to work on a long-term project. This newfound knowledge was effectively used in our project to create a simple AI that was able to guess letters.

Sources

Microsoft. *Microsoft.com.* Microsoft, n.d. Web. <https://social.msdn.microsoft.com/Forums/getfile/328312>

Russell, Stuart J., and Peter Norvig. "Artificial Neural Networks." *Artificial Intelligence: A Modern Approach.* Upper Saddle River, NJ: Prentice Hall, 2009. 728. Print.