Neural Network to Recognize Character from Sample Images

CSCE 420 FINAL PROJECT

Author: Charlemagne Wong
Dr. Daugherity
April 25, 2019

Problem and Significance:

A Neural Network is an artificial intelligence algorithm that models how a living organism's brain functions to predict an output. The advantages of a neural network are that it allows us to develop a model for predicting outputs from inputs that would be otherwise be too complicated to calculate from developing a traditional algorithm. Instead, we train the neural network with examples to further refine its ability to output the correct result. Essentially, a neural network is a pattern recognition algorithm used to predict unexpected inputs and outputs.

In the scope of this assignment, we are to write a neural network algorithm to predict the letter in an image. Each image is a 14 by 9 array where each cell is either a black or white pixel. By feeding training examples to the neural network, we would be able to predict new input images containing a character with high accuracy.

Restrictions and Limitations

The program can only accept capitalized letters (A – Z) and would result in undefined behavior if fed another character not specified in that domain or any other image. Additionally, the image must be contained in a 14x9 array, thus limiting the resolution to 126 pixels. The program cannot take an image with color and the algorithm is not trained to recognize if a pixel is a shade of white and black, thus resulting in undefined behavior if the program contains such pixels. Finally, the program is taught with the font "IBM EGA 9x14", so any input that is not that font may have difficult in predicting the correct output. The restrictions and limitations of the program is set based on the scope and the instructions given for this assignment

Approach

When developing the algorithm to the neural network, I used matrices to represent the neurons and the weights. A row in the matrix is a single neuron and the columns for that row are the individual weights from the previous layer of neurons. Using this method, I was able to increase performance in predicting and back propagating the neural network.

To train the neural network, I used back propagation for every epoch and then recorded the accuracy after each training session from a sample set. Each epoch is a set of shuffled 26 images, where each image represents a capital letter in the font. Each of these images of fonts are shifted randomly in the x and y position. The sample set used to test the accuracy of each image are 300 random character images A - Z, with each image randomly shifted in the x and y axis. This sample set is constant throughout each epoch to ensure consistency in testing.

To test the number of noise (flipping a random pixel from black to white or vice versa), I tested each image by generating a list of image characters from A – Z. These characters are randomly shifted once and will not be re-shifted when adding noise to each image. I then flipped a random bit without repetition for each character until the neural network failed to predict it.

Sample Run:

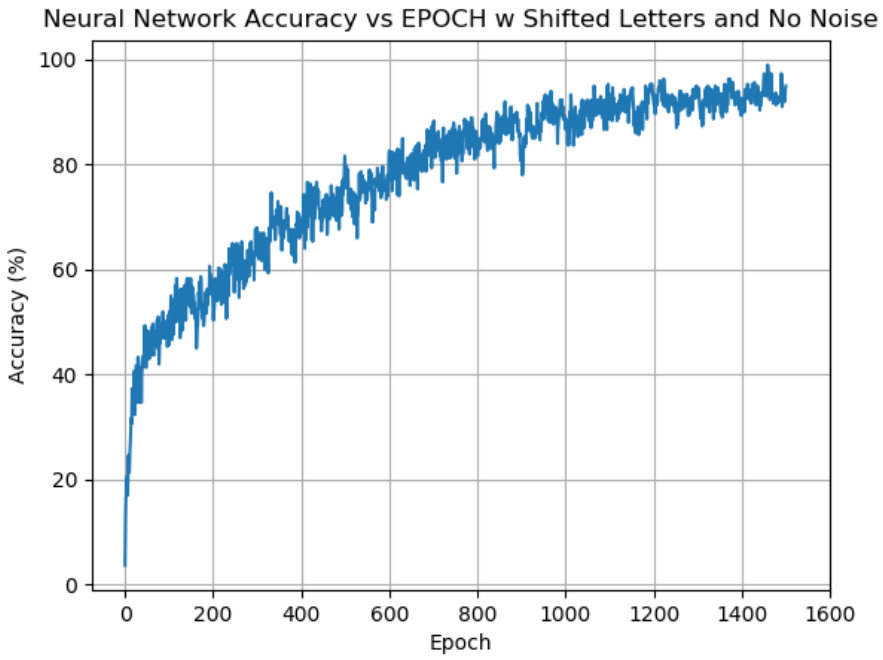

Figure 1: Accuracy of neural network after training an epoch of 26 characters

Final Accuracy after 1500 epochs: 96.66%

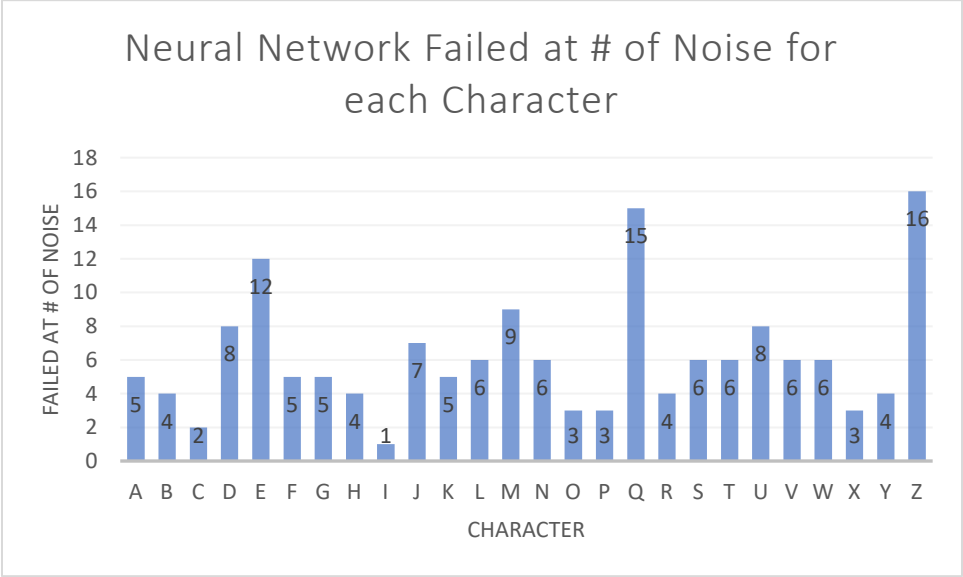

Figure 2: The number of random noises to prevent the neural network from recognizing a character.

To read the figure, ‘A’ with failure at 5 means that the algorithm fails when at least 5 pixels are flipped

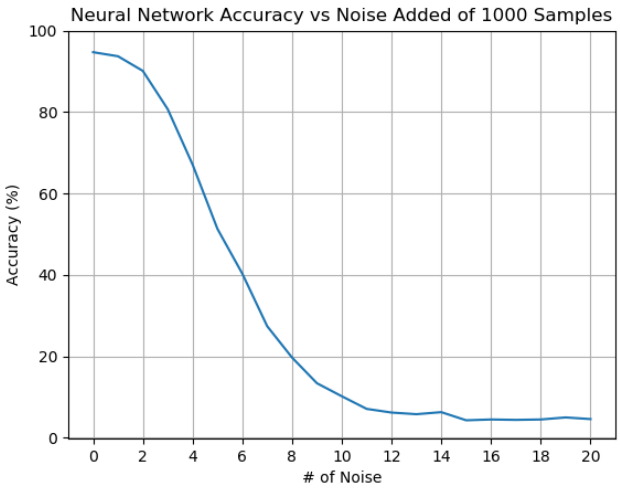

Figure 3: A graph of the accuracy of the neural network from a sample of 1000 images when adding random noises

Results and Analysis:

In Figure 1, it is shown that as the number of epochs increase, the accuracy of the program generally increases as well. There is a large variance in accuracy within short intervals. However, this is typically corrected once more training data is fed to the algorithm.

Using the trained neural network, I tested to see the number of random flips in the image are necessary to fail the recognition. As shown in Figure 2, I found that some letters, particularly E, Q, and Z are very robust to noise. However, characters such as C and I are especially prone to noise. This is likely due to their similar appearances to other characters and the low resolution of the images.

Finally, I also tested to see how the number of noises affects the accuracy of the neural network as shown in Figure 3.

Conclusion

After developing the neural network and training it, it was able to accurately predict the characters from the images from the scope of the project. However, the neural network is shown to be inaccurate for a few characters with noise, so its accuracy when fed a different font is questionable. However, given the scope of the assignment, its accuracy of 96.66% where each image is shifted is a good value for this neural network.

Future Research

A potential improvement is to modify the alpha value of the algorithm if the program correctly identifies the output. Additionally, more research can be done to study how the number or neurons and layers affect the results if given the same number of training data.

When developing this program, I observed that simply adding more layers or neurons did not result in a more accurate results given the same number of epochs. This may be because the program requires more examples to output the correct result.

Instructions:

The main program is called 'FinalProject.py' and it runs on Python 3.6. The first 30 lines of the python file contains Boolean flags to test various components of the algorithm, including training the algorithm and testing the robustness of the program by adding random noise. Note that the GRAPHING flags should be left false if the Python interpreter does not contain matplotlib. This was not added in the program because due to the risk of exceeding the file size limit on the submission. However, if this flag is on, the program will graph and save such graph into the working directory.

Bibliography

Buzzanca, Marco, IBM Fonts, (2016), farsil (GitHub repository), https://github.com/farsil/ibmfonts