MSDS 458, Section 55, Assignment 2

Alan Kessler

Emulate Deep Network

The goal of this analysis is to build on the work from the first assignment by incorporating concepts from deeper neural networks. This is accomplished by taking an intermediate step of using a gray box model as a way to include identifiable features in the neural network. This is similar in nature to interpreting the results of a deep convolutional neural network. The data used in the assignment is a collection of letters from the English alphabet in the form of arrays. Through this analysis, it is possible to understand and interpret features that can be incorporated into an additional simple neural network. Like the first analysis, the challenge is less about training a successful neural network for deployment but to understand what is happening with the gray box and hidden layer nodes.

That data used in the analysis is the same as the first assignment: 9 by 9-pixel representations of 24 letters of the alphabet. The data is loaded in the program used for the entire analysis named “kessler_9x9_graybox.py”. The difference in data used for this assignment is that random variants have been incorporated. The original data is used but ten additional sets of letters are added. For these variants, each pixel has a 20% probability of changing from a 1 to a 0 or from a 0 to a 1. The result is a collection of differing letters but the consistency in representations remain. A larger input data size results in the model taking longer to converge but should result in the neural network generalizing better to small differences in the input data. In the first assignment, error was represented by the sum of squared errors, but due to this change in the number of observations, it is now represented as average squared error.

The gray box structure is shown in Figure 1. It involves creating an initial neural network that classifies letters into a series of large classes. The hidden node activations from that neural network can be incorporated with the original input data to train another neural network that will classify inputs as letters themselves. The result is that the input from the gray box include identifiable features based on the large classes. For this analysis, there are four classes selected by judgment:

Class 0 (Line through the Middle): A, B, E, F, H, P, R, S
Class 1 (Mostly Circular): C, D, G, O, Q, U
Class 2 (Includes Diagonals): J, K, M, N, W, X, Z
Class 3 (Other): I, L, T

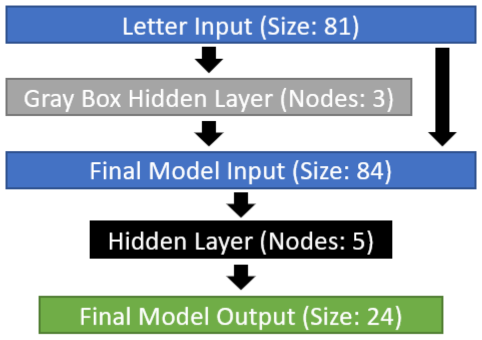
Letter Input (Size: 81)
Gray Box Hidden Layer (Nodes: 3)
Final Model Input (Size: 84)
Hidden Layer (Nodes: 5)
Final Model Output (Size: 24)

Figure 1 – Gray Box Architecture

The first neural network in the gray box set up is comprised of an input layer of size 81, a single fully connected hidden layer with three nodes, and an output layer of size 4 to represent the possible class outputs. A sigmoid transfer function with alpha set to one is used to calculate the output of the hidden layer. Backpropagation is performed to train the neural network with a learning rate of 0.5 over approximately 40 epochs using sampling without replacement for each epoch. It is important to keep in mind that with the variants included that a single epoch contains many more letters than the first analysis. The average squared error has a threshold of 0.02 for classifying to the four classes.

The activations from the hidden layer are concatenated with the original input data to generate a new input layer of size 84. The new neural network has a single fully connected hidden layer with five nodes and an output layer of size 24 to represent all of the possible letters. The random variants make fitting this model more difficult and the backpropagation is stopped after approximately 390 epochs when the average squared error reaches 0.4. Five nodes are chosen to make the total number of hidden nodes add up to eight like the previous analysis.

The first step in interpreting the hidden node output is to review the hidden node weights for the gray box activations. Like the first analysis, this is accomplished by visualizing the weight matrix from the input to hidden layer and the product of the weights and the input. The code used in the analysis focuses on visualizing the non-variant versions of the letters. The results for Hidden Node 0 are shown in Figure 2. The weights on the left show positive response for the border area but a strong negative response to pixels in the center of the image. This correlates strongly to Class 1 (Mostly Circular). Class 1 letters like “D” (shown on the right) activate the node while “A” and “X’ do not. The split is not complete as shown for “W” which is not part of Class 1 but activates Hidden Node 0.

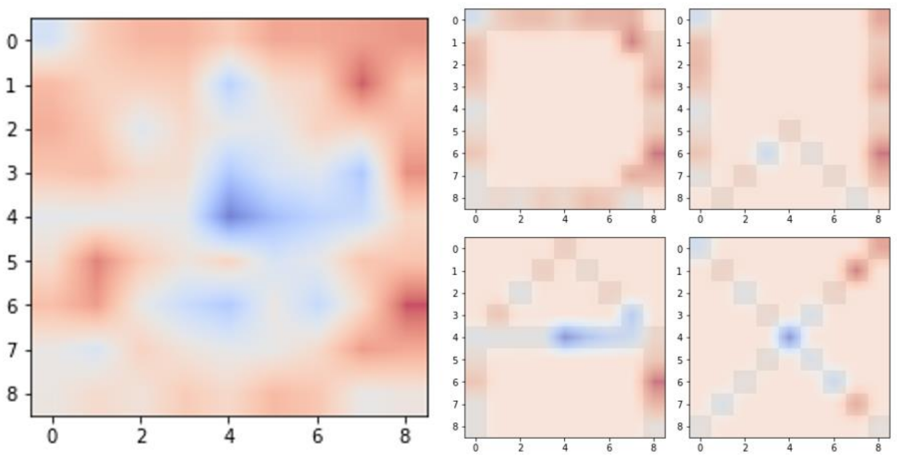

Figure 2 – Hidden Node 0 weights (right), examples of letters that activate the node (upper-right), and letters that do not (lower-right)

Hidden Node 1 shown in Figure 3 correlates strongly to Class 0 (Line through the Middle). It activates for every member of the class and does not activate for letters with only one or two pixels in the center line such as “I”.

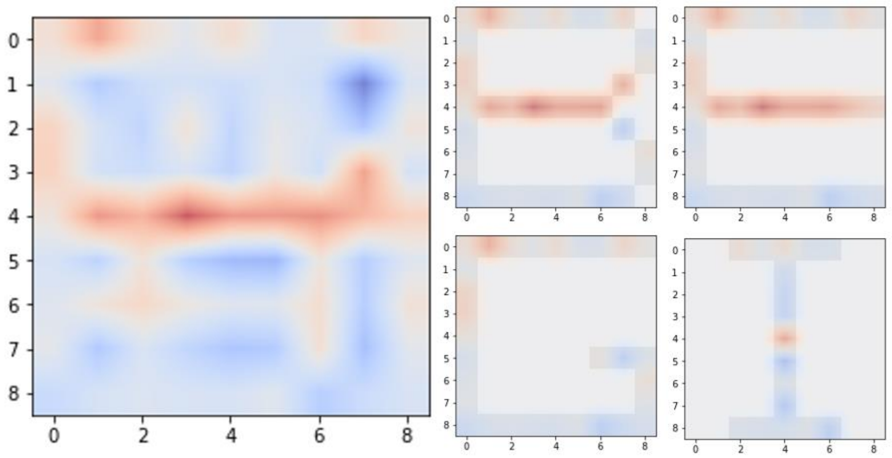

Figure 3 – Hidden Node 1 Weights

Figure 4 visualizes Hidden Node 2. This Hidden Node activates for members of Class 3. Based on this analysis the hidden node activates for the gray box do a good job of segmenting the letters into groups that will ultimately improve the next neural network’s ability to converge.

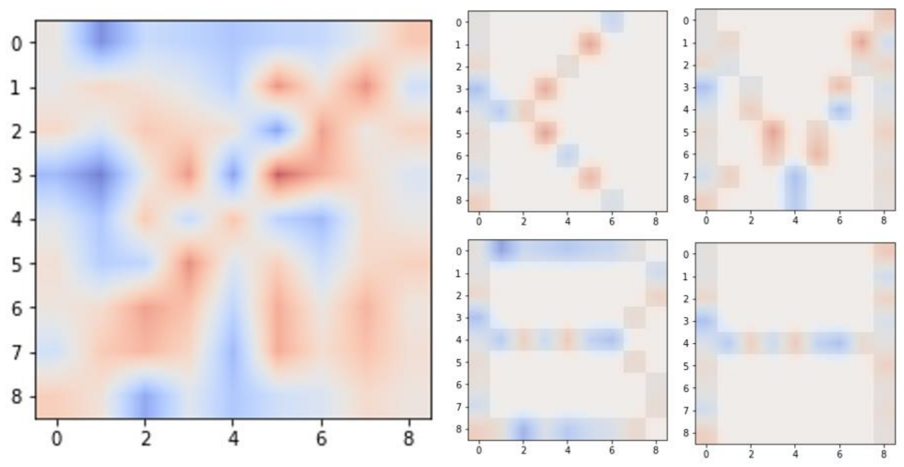

Figure 4 – Hidden Node 2 Weights

The hidden node activations from the gray box are helpful but do not completely cover the differences between letters in the same class. The hidden node in the final model is needed to further segment the output space. These final hidden nodes can cover the elements not considered by the larger classes to converge. The analysis into these letters is similar to that of the first analysis. Figure 5 shows the weight matrices for each of the remaining five hidden nodes in the final model.

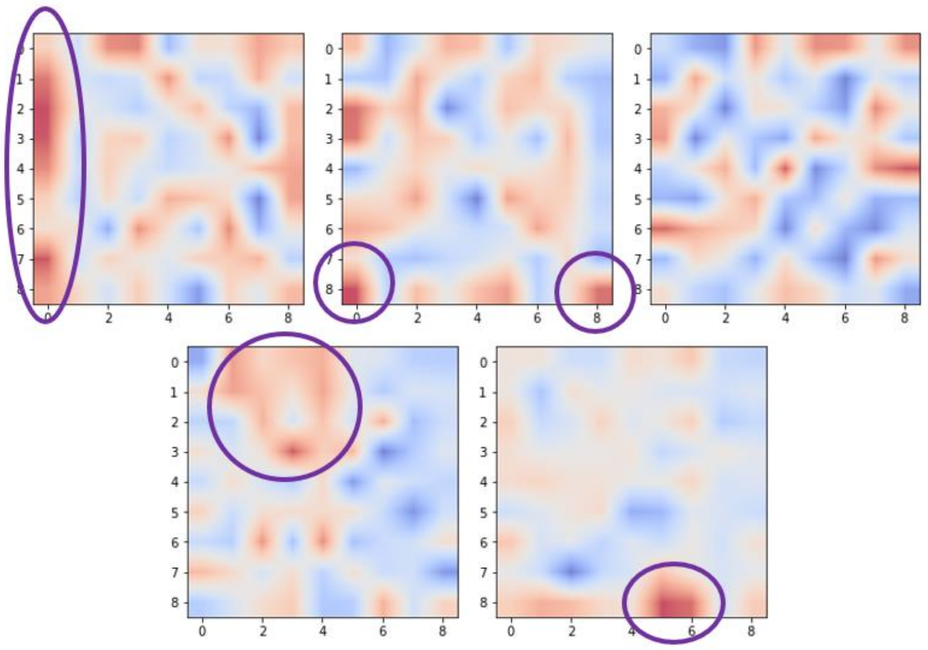

Figure 5 – 5 Hidden Nodes of Final Model with Highlights Points of Interest

The circles show areas where the presence of pixels activate the node. Note that the visuals do not include the weights given to the gray box outputs. The first node shown in the upper left activates for letters with pixels on the left-most boundary. The next node activates for pixels in the corners. A similar pattern emerges for last two nodes as well. The most interesting result is that the third hidden node activates for all non-variant letters effectively rendering it a “dead node” with respect to the original input data. While in the first analysis, all eight nodes providing input, the presence of the gray box resulted in needing fewer fully connected hidden layer nodes.

In terms of Python programming, the numpy Python module performs the necessary calculations for the backprogagation and forward pass. Numpy also serves as the means by which randomizations are generated. Named tuples are used to store information about a given forward pass run and are implemented with the standard collections module. Matplotlib is used to visualize the data and results some of which are saved in pandas data frames as well.

Using a gray box structure involving classifying to larger classes before identifying single letters emulates the behavior of deep neural networks. Identifying features is a prominent role of convolutional neural networks and is what make them efficient relative to large fully connected neural networks. Not only are the features more interpretable in this example, but even with simple data and a simple structure, the number of hidden nodes needed to build a successful model is reduced.