MSDS 458, Section 55, Assignment 1

Alan Kessler

Classifier Neural Networks

The goal of this analysis is to understand how the hidden nodes in a simple neural network learn to represent features within the input. 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 the differences between hidden nodes as a way to ultimately generate the correct output. The challenge is less about training a successful neural network for deployment but to understand and interpret the hidden node activations as features.

The data used in the analysis represents 24 letters of the alphabet excluding the letters “V” and “Y”. Each letter is made up of 81 binary inputs that form a 9 by 9 visualization of the letter. There is only one representation of each letter considered in the analysis. This matches the intent of the analysis not being about predictive power on out-of-sample data, but rather to interpret hidden layer results. The Python script, “kessler_9x9.py”, visualizes each of the letters individually. Examples are shown in Figure 1. In reviewing the representations, it is worth noting that the letters demonstrate a level of consistency in their representation. For example, they all take up the entire 9 by 9 space and letters like “N” shown below have elements that cover entire edges. This consistency is important to the kinds of features or rules that may be represented.

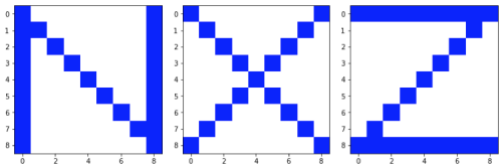

Figure 1 – Letter Representations for “N”, “X”, and “Z”

The neural network itself is comprised of an input layer of size 81, a single fully connected hidden layer with eight nodes, and an output layer of size 24 to represent the possible letter outputs. A sigmoid transfer function with alpha set to one is used to calculate the output of the latter two layers. Backpropagation is performed to train the neural network with a learning rate of 0.5 over approximately 4,400 epochs using sampling without replacement for each epoch. The resulting total sum of squared

errors reaches the set epsilon threshold of 0.05. A total of eight nodes were chosen for the hidden layer based on a review of the assignment readings and the results in which none of the nodes “died” as a result of training.

The goal of the analysis is to interpret the hidden node output and how along with the input data represents features important to classifying letters. The first step of this interpretation is to visualize the weight matrix from the input layer to the hidden layer for each hidden layer node. The red areas of the visuals indicate parts of the matrix where an input of one will increase the activation while blue areas decrease the activation for a non-zero input. It may be easy to focus interpretation efforts on the positive weights but the negative weights are equally important in analyzing the results. For each node, it is also important to view the product of a single letter input with the weight matrix. Visualizing that product shows which weights are active which is helpful in understand what is leading to whether that node is activated for a specific letter. By repeating that analysis for multiple letters, it is possible to see which letters are interacting with the hidden layer in similar ways.

These results for Hidden Node 0 are shown in Figure 2. The letters “M” and “I” are examples of letters with activations of approximately one, and “F” and “O” have activations close to zero. This node activates for letters that have pixels in the upper left-hand corner and then do not cover the entire upper boundary: “H”, “K”, “L”, “W”, “X”. The node also covers the feature where a pixel in the lower left-hand corner offset by the boundary by one is present: “M”, “U”, “Z”. Finally, the letter “I” also activates this node through positive weights corresponding the middle line in that layer. This results in the conclusion that for a neural network with this many outputs and relatively fewer hidden nodes, each node can group a number of features to be represented.

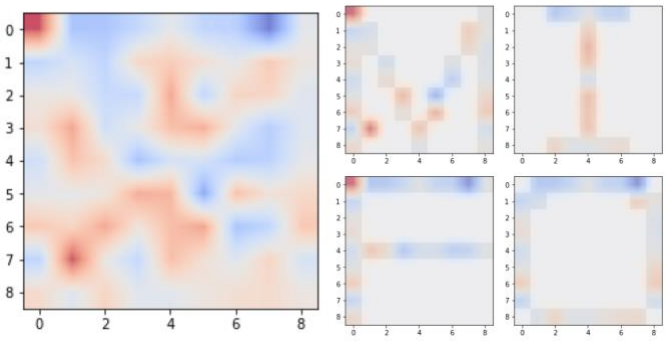

Figure 2 – Hidden Node 0 weights (right), examples of letters that activate the node (upper-right), and letters that do not (lower-right)

At first glance, Hidden Node 1 appears to represent a singular feature where letters with pixels on the upper right-most side activate the node. This is true for letters like “B”, “N”, and “O”. However, the node is also active for letters that have pixels in the middle of the “image” such as “I”, “J”, and “T”. Examples are shown in Figure 3. Letters that activate Hidden Node 2, shown in Figure 4 have strong common elements. The first is that they all have pixels in the lower left-hand corner. Many letters share this characteristic but for letters that also have pixels in a diagonal line from upper left to lower right will not activate the hidden node. This is true for the letter “X” as an example.

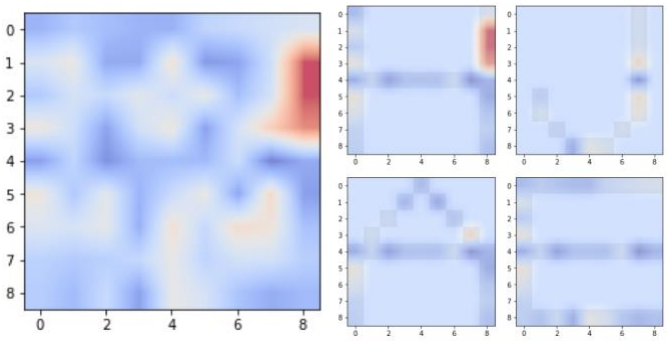

Figure 3 – Hidden Node 1 output in the same format as Figure 2

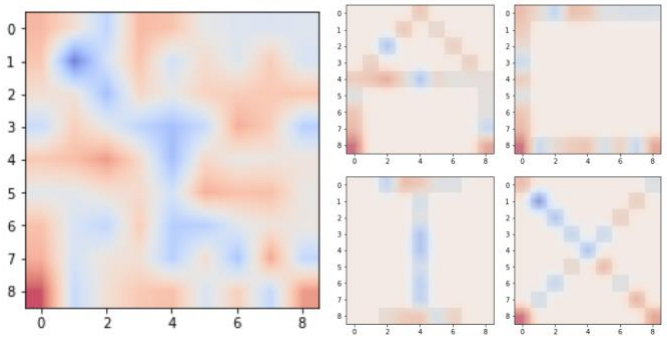

Figure 4 – Hidden Node 2 output

Hidden Node 3, shown in Figure 5, activates in most cases for letters which lines across their midsection. This is true even for the letter “B” that has pixels corresponding to positive weights for the hidden node on the right side of the image. For Hidden Node 4 in Figure 6, letters with pixels in the upper right and middle of the image will push the node to activate however, even in those cases like the letter “R”, the presence of pixels in the lower right diagonal appear to deactivate the node.

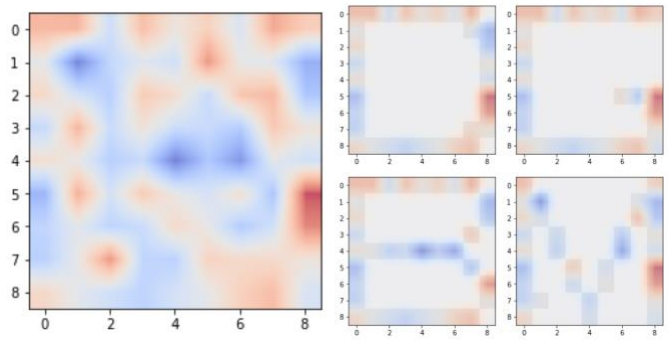

Figure 5 – Hidden Node 3 output

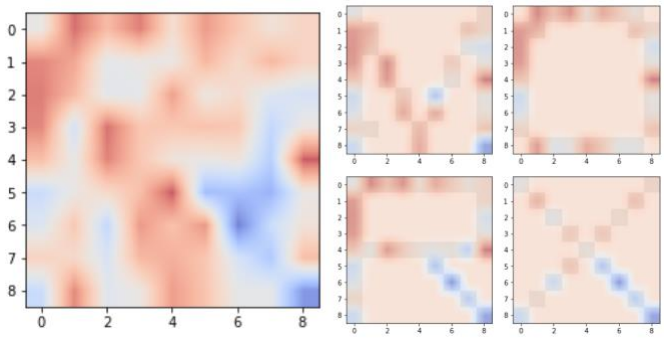

Figure 6 – Hidden Node 4 output

The same analysis continues for Hidden Nodes 5, 6, and 7 in Figures 7, 8, and 9 respectively. For Node 5, letters with pixels in the bottom row help to activate the node while pixels on the rightmost column do the opposite. Node 6 activates when pixels are present in the lower righthand corner and Node 7 activates for letters with pixels in the top row and upper left to lower right diagonal.

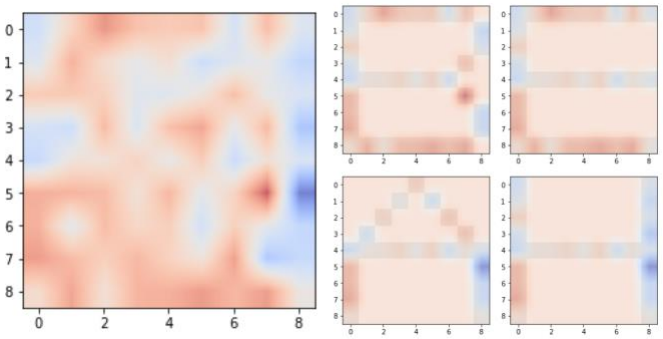

Figure 7 – Hidden Node 5 output

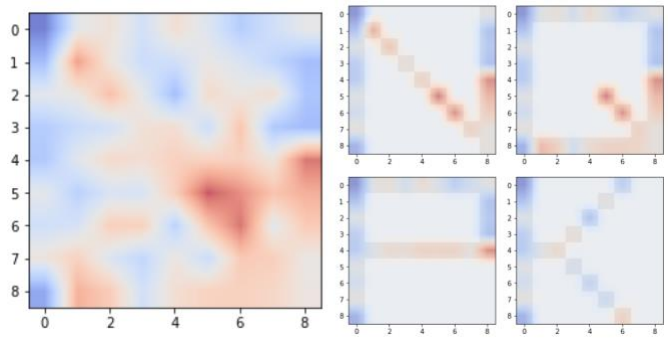

Figure 8 – Hidden Node 6 output

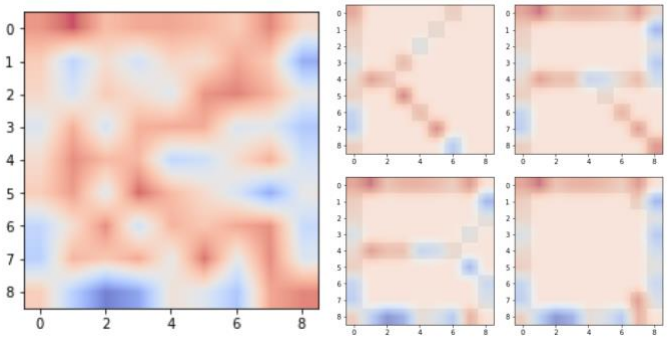

Figure 9 – Hidden Node72 output

From a programming standpoint, the numpy Python module performs the necessary calculations for the backprogagation and forward pass. 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.

By analyzing the hidden node activations and their relationship to the input and trained weights, it has been possible to visualize how each node segments the output space by what could be thought of as features of the letters. The nodes generate different segmentations that when combined work with the weights going from the hidden layer to the output layer to ultimately identify the correct letter. In the future it will be interesting to see how this interpretation varies with more complicated neural network structures and how generating features can help with generalization.