Introduction

In Assignment 1, I implemented a simple, single-layer classifier neural network. Using sixteen different letters as inputs, generated as 9x9 grids with each cell either “on” or “off” to visualize the letter, I passed the eighty-one inputs through six hidden hidden layers and, finally, into six distinct output classes. While my analysis was not primarily concerned with the effectiveness of the network design, the network performed fairly well nonetheless, correctly predicting the classes of fourteen of the sixteen input letters.

To understand the inner workings of the network, I analyzed the hidden-to-output weights for the final iteration of the learning process. By examining the weights in indexed format (index with base 100 along the hidden node axis), I could begin to recognize which features of the input arrays each hidden node was primarily focused on when activating (or deactivating) the output nodes. Of course, any explanation of what features the hidden nodes were responding to was not quite accurate, as the network did not see things in terms of “vertical lines across the top”, but rather was responding to groups of nodes in various sections of the eighty-one node input array; however, for purposes of clarity and mental visualization, the deviation from precision seemed warranted.

In this paper, I have introduced a small, but important change to the network-- the output of our previous model is used as inputs in this new model. So, with our eighty-one inputs from our 9x9 grid, we add the nine output values from the previous learning iteration. In this paper, we will examine how this affects output accuracy, hidden-to-output weights, and network efficiency.

Output Results Comparison

As mentioned above, in Assignment 1 (A1, hereafter), the network correctly predicted the output classes for fourteen of the sixteen input letters. For letter ‘C’ (Class o3), the network did not strongly predict any class for ‘C’, the highest output activation was 0.1. For letter ‘G’, the network strongly misclassified the letter into the class for ‘B’ instead of the correct class ‘C’. Clearly, the simple network had issues with that particular class.

Node	o0	o1	o2	o3	o4	o5	o6	o7	o8
Class	A	B	C	D	E	I	K	L	M
A	0.93	0.03	0.03	0.00	0.07	0.00	0.10	0.07	0.05
B	0.01	0.96	0.06	0.06	0.01	0.02	0.03	0.00	0.01
C	0.00	0.07	0.08	0.13	0.12	0.10	0.04	0.03	0.00
D	0.00	0.05	0.06	0.84	0.00	0.03	0.04	0.02	0.02
E	0.04	0.06	0.08	0.01	0.90	0.04	0.06	0.04	0.00
F	0.04	0.07	0.07	0.01	0.88	0.02	0.07	0.03	0.00
G	0.00	0.86	0.08	0.14	0.09	0.06	0.00	0.00	0.00
H	0.89	0.07	0.03	0.00	0.02	0.00	0.07	0.04	0.07
I	0.00	0.02	0.10	0.15	0.03	0.90	0.07	0.04	0.01
J	0.00	0.03	0.09	0.08	0.01	0.91	0.01	0.12	0.05
K	0.02	0.00	0.04	0.05	0.01	0.01	0.85	0.02	0.11
L	0.05	0.00	0.05	0.05	0.02	0.05	0.04	0.86	0.05
M	0.03	0.03	0.03	0.12	0.00	0.02	0.06	0.04	0.91
N	0.03	0.03	0.03	0.12	0.00	0.02	0.06	0.04	0.91
O	0.00	0.05	0.06	0.84	0.00	0.03	0.04	0.02	0.01
P	0.01	0.91	0.05	0.06	0.02	0.01	0.04	0.00	0.00

Table 1- Actual Output Activation from A2 Network

The new network did not fare any better- the results are nearly identical to the network from A1, as seen in Table 1 above. The letter ‘G’ is still misclassified, and the letter and class ‘C’ are still black boxes to the network. Considering these results, we’ll begin to look for any benefits of the additional network complexity, beyond output accuracy.

Hidden Weight Patterns

For reference, Table 2 shows the hidden-to-output weights from the A1 network, colored along the hidden nodes, with red signifying significantly negative weights and blue signifying significantly positive weights. (Again, we used a weight index, rather than the actual weights, in order to examine the relative power of each weight without getting caught up in the actual values.) As mentioned in the previous report, while a few patterns emerge with regards to feature extraction, the network is not strongly identifying any major, higher dimensional features.

Node	Class	H0	H1	H2	H3	H4	H5
o0	A	-989	-336	-868	10	23	180
o1	B	267	-689	45	735	-427	106
o2	C	-108	-19	19	-113	-118	-22
o3	D	1131	-572	316	-732	84	-14
o4	E	-3	554	-892	671	-170	-33
o5	I	95	321	40	-884	-482	59
o6	K	-216	-309	6	653	78	-549
o7	L	271	284	-254	-678	73	-497
o8	M	-1346	-133	688	-563	37	-131

Table 2- Final Hidden-to-Output Weight Index from A1 Network

In Table 3, however, we begin to see that the network has produced quite different results with regards to these weights. Each node appears to be much more “confident” in the effect it should have on each output node-- the relative weights are much stronger than in A1. Additionally, each hidden node is much more likely to “commit” to a particular feature or set of features. For instance, node H0 strongly deactivates classes A, B, and E, while strongly activating classes D, I, K, and L. The nodes corresponding to horizontal lines across the center clearly do not agree with node H3, and the weight choices emphatically reflect that.

Node	Class	H0	H1	H2	H3	H4	H5
o0	A	-2185	-655	260	-8101	48	-1108
o1	B	1323	1052	-346	-10011	-51	-912
o2	C	187	-10	-127	-78	-136	223
o3	D	1278	680	306	6475	-422	855
o4	E	1235	-1042	-144	-9798	-115	1466
o5	I	-391	-63	-1189	5048	31	517
o6	K	1651	-571	245	3644	-143	-1334
o7	L	-2054	-618	162	4477	-81	1393
o8	M	-1944	328	-67	9243	-30	-2001

Table 3- Final Hidden-to-Output Weight Index from A2 Network

Interestingly, class C remains a mystery to each of the hidden nodes, with none of them offering strong activation or deactivation of output o2. I am still unsure of why this might be.

Time to Converge

With eta set at 0.5 in both models, signifying the SSE threshold at which the model stops learning, an interesting finding is how much faster this new method causes the model to converge. Model A1 stopped running after ~4,800 iterations, while Model A2 took only 3,500. On a small, simple network with such straightforward data, that delta did not make much difference in computational complexity or run time; however, in a larger, more complicated network, that amount of performance improvement would make a significant difference in costs. By passing output back into the network as inputs, we give the network a “jump start” at each iteration, speeding the pace at which it approaches convergence.

Conclusions

While this model did not technically add another layer to our original network, it emulated the multilayer learning process by passing previously learned information as inputs into the learning process. As such, it more confidently identified input features and sped up learning time, allowing it to converge on fewer iterations. While accuracy did not improve, that can be attributed to network design-- the number of hidden nodes could be adjusted to likely produce better results.