Josh Veltri (jkv13)
Justin Wang (jsw104)

Programming Writeup 2

(a) What is the area under ROC of the ANN with no hidden units on each dataset? Set the weight decay coefficient γ=0, and train to convergence. This approximates the perceptron, which uses a step function instead of a sigmoid. How does this compare to the decision stump/tree results in the previous assignment?

The assignment did not specify whether this experiment should be run with or without cross-validation. The numbers in the table below were collected using cross-validation. Convergence was considered to have occurred when no weights or biases changed by more than 0.0075 between two training epochs. For voting and volcanoes, the accuracy measures are slightly higher than, but generally comparable to, the accuracies of the depth-1 decision tree from the previous assignment. For spam, the accuracy of the single-neuron network is much lower than that of the depth-1 decision tree.

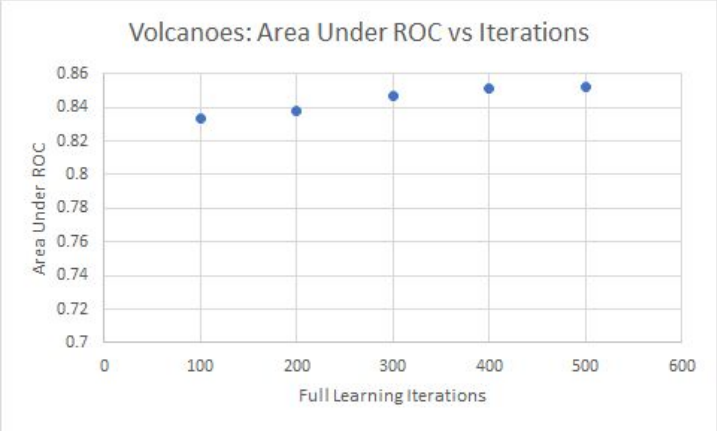
Dataset	**Area Under ROC**	**Accuracy**
voting	0.99158	0.98832
volcanoes	0.82205	0.78794
spam	0.69580	0.51697

(b) For volcanoes and spam, and explore how the AROC changes as learning iterations are increased. Fix the number of hidden units to 5 and γ=0.01 for these experiments. Plot AROC results for at least three values of learning iterations evenly spaced between 100 and 10,000. Compare your results to the "perceptron" in part (a). Does the introduction of hidden units lead to improved accuracy? How many iterations does this ANN need to converge compared to the "perceptron"?

Volcanoes:

For volcanoes, we observed that with 5 hidden using and a weight decay of 0.01, we typically saw convergence after around 400 full training iterations (that is after showing the network each training example 400 times). Therefore, we tested the ROC at every 100 training iterations between 100 and 500.

Learning Iterations	**Area under ROC**
100	0.83331
200	0.83761
300	0.84713
400	0.85164
500	0.85190

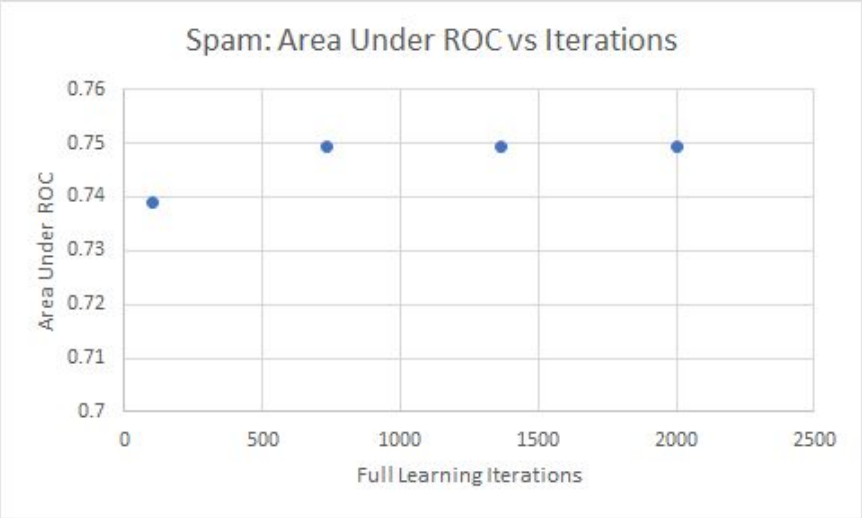


We see that the on the volcanoes data set, the introduction of the hidden units increased the area-under-ROC slightly relative to the area-under-ROC of the single neuron (from 0.82205 to between 0.833 and 0.852). The increases were greater with more learning iterations, but the total increase was still modest. We also observed a modest increase in the accuracy as a result of using the hidden layer -- when the weights were allowed to converge, the accuracy of the network using hidden neurons was 0.82126, an increase from 0.78794 in the single-neuron case. Convergence of all weights and biases is significantly increased from around 65 training iterations in the single-neuron case to around 400 iterations in the 5-hidden-neurons case.

Spam:

For spam, we observed that with 5 hidden units and using a weight decay of 0.01, we saw convergence after around 2,000 full training iterations (that is, after showing the network each training example 2,000 times). Therefore, we tested the ROC every ~633 training iterations between 100 and 2,000.

Learning Iterations	**Area under ROC**
100	0.73892
733	0.74934
1,366	0.74944
2,000	0.74940

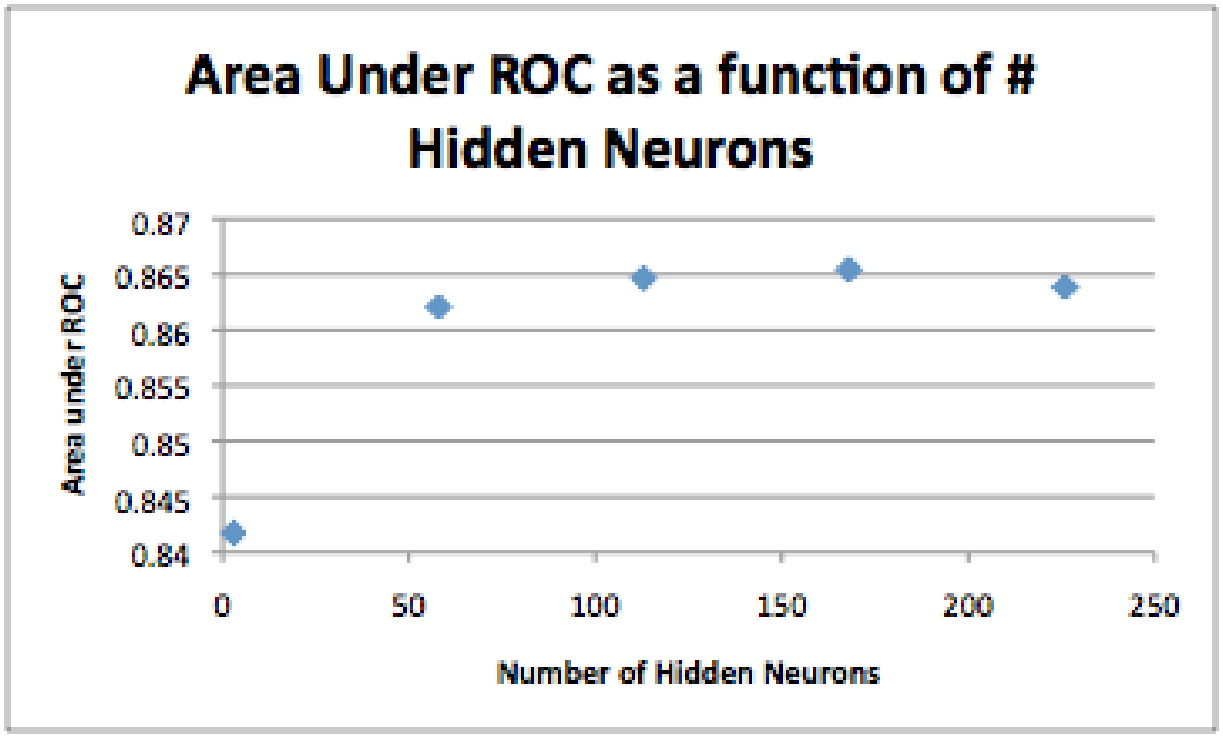


We see that the on the volcanoes data set, the introduction of the hidden units increased the area-under-ROC slightly relative to the area-under-ROC of the single neuron (from 0.69580 to between 0.738 and 0.750). The increases were least with few total learning iterations, but the increase leveled out and there was little difference between the results with 733, 1,366, and 2,000 iterations. When using the five hidden neurons, we observed no significant differences in the accuray from the 0.517 value observed with the single neuron. This probably indicates that more hidden neurons are needed. Convergence of all weights and biases is significantly increased from around 45 training iterations in the single-neuron case to around 2000 iterations in the 5-hidden-neurons case.

(c) Explore how the AROC changes as the number of hidden units are varied on volcanoes. Plot AROC results for at least three values of hidden units evenly spaced between 3 and f, where f is the number of input units. Set γ=0.01 and the learning iterations to 100f , with a minimum of 10,000. Compare the ROC and training times to the results in parts (a) and (b). Warning: This experiment may take a long time to run.

Volcanoes:
For volcanoes, there were 226 input units. Thus, each experiment was run with 22,600 learning iterations.

Hidden Units	**Area under ROC**
3	0.841718492311
58	0.862078830261
113	0.864671955186
168	0.865438974728
226	0.863863014456



For the volcanoes data set, we saw a significant spike in the area under the ROC for increasing the number of hidden neurons from 3 to 58, but saw diminishing returns the more neurons we added to the hidden layer.

Adding hidden neurons overall definitely produced better AROC results than in B where the number of hidden units was stagnant at 5. This is illustrated by the first experiment where we have 3 hidden neurons in the network producing worse AROC results than the fully converged or close to converging examples in part B. However, after increasing the number of hidden neurons to 58, the AROC is greater than all of the experiments in part B regardless of the number of training iterations in B.

This is further illustrated by comparing these results to part A where the number of hidden units was stagnant at 0. However, this time all of the experiments in part C have greater AROC than the experiment in part A since every experiment in part C has more hidden neurons than part A. This further illustrates that adding hidden neurons to the hidden layer is beneficial for the AROC of the network.

There are some cons however to just adding neurons indefinitely to maximize the AROC. We found that adding more hidden neurons to the hidden layer significantly increased the training times of the neural network. For a hidden layer with only 3 neurons we saw convergence after around 500 training iterations, but with a hidden layer of 226 neurons, we saw convergence at around 30 training iterations or less. However, the reduction in training iterations was not enough to compensate for the increase in time complexity from adding neurons to the hidden layer. Additionally, we saw that the network actually produced slightly worse results for 226 hidden neurons than 168 hidden neurons. This suggests that after a certain point, adding more hidden neurons is actually detrimental to the network!

(d) Write down any further insights or observations you made while implementing and running the algorithms, such as time and memory requirements, the complexity of the code, the effect of parallelizing code if you tried that, etc. As in the previous assignment, you can also test your code with classification problems from your research. If you do so, please describe the problem and any observations you were able to make. Especially interesting insights may be awarded extra points.

Backpropagation is computationally expensive. The most expensive step includes a matrix multiplication requiring $n * m$ scalar multiplications per example for a layer with n inputs and m outputs. A full training iteration, in which we show all C training examples to the network and choose new weight values, is then $O(C * m * n)$. In a multi-layer network, the layer with the largest $m * n$ will drive the computational complexity of the entire network. In our experiments, we had at most two layers and the second layer had only a single output. Therefore, the computational complexity of a training the network is driven by the number of hidden nodes we place in the hidden layer and the number of training examples. The space complexity of backpropagation through a layer is considerably lower -- just $O(m * n)$ for a layer with n inputs and m neurons, since we need to pass a matrix of old weights (size $m * n$) and a vector of bias derivatives (size m) backwards from one layer to the next. While this can be quite large if there are a large number of inputs or a large number of nodes in the layer, these matrices and vectors

do not need to be stored beyond a single training example. Thus the value of C, which we saw is one of the drivers of the computation complexity, does not impact the memory requirement.

The pooled area-under-ROC computation is also surprisingly computationally complex. Our initial implementation computing this for n examples was $O(n^2)$ because we initially used an inefficient method to compute the FP-rate/TP-rate pairs. This turned out to be too complicated to run quickly for the spam data set. We found a way to compute the FP-rate/TP-rate pairs in $O(n)$ time, but the overall area-under-ROC computation is still $O(n * log(n))$ because those pairs must be sorted after they are generated. However, this was a very significant improvement given that the spam data set has over 70,000 examples.