Problem 1

1. bright-or-dark— At least 75% of the pixels are on, or at least 75% of the pixels are off.

A perceptron cannot recognize this feature. A perceptron can easily be trained to recognize one of the two conditions, for example by having all of the weights be 1 and having a threshold activation function of either 7.5 or 2.5. However, once it is set to recognize one, it cannot recognize the other because the threshold would have been set at a hard limit already.

1. top-bright — A larger fraction of pixels is on in the top row than in the bottom two rows

A perceptron can recognize this feature. We can see this by having the three weights on the top row be $\frac{1}{3}$ and all the other six weights be $-\frac{1}{6}$. We would set the threshold to 0 and check if the sum is greater than 0 to indicate a positive example and otherwise a negative example.

1. connected — The set of pixels that are on is connected. (In technical terms, this means that if we deﬁne a graph in which the vertices are the pixels that are on, and there is an edge between two pixels if they are adjacent vertically or horizontally, then there is a path between every pair of vertices in the graph.)

A perceptron cannot detect this feature. There are too many variations and combinations in terms of number of pixels that are on as well as the location of where these pixels are. For example, 1 or 2 pixels on could be a path, as can 8 or 9. We cannot do anything with the total number of pixels on. We also cannot learn weights for specific squares or specific areas of the grid because paths can appear in any location. So any way we train the weights we can come up with a counter example path that the weight set incorrectly classifies by countering the training strategy. This means either by changing the number of pixels that are on, or constructing a path on a different portion of the grid than where the weights were originally tuned to correctly classify paths.

QUESTION 2

The domain of handwritten digit recognition is 14x14 pixels with intensities ranging from 0 to 255. The features for this domain are the pixel intensities themselves.

Decision trees

Decision trees that are modified to split on ranges of continuous values would be able to classify handwritten digits though they will neither be as effective nor as natural for this domain as other techniques such as using neural networks. The decision tree would need an effective algorithm to take into account lookahead because digit recognition revolves around combinations of pixel intensities. An algorithm analogous to ID3 would not work well because of the reliance of attributes on one another and the fact that nearby pixels affect recognition of what digit is formed. Because of the relatively large number of pixels, an effective decision tree would have to be deep because it would have to predict on combinations of features, not single attributes. Decision trees would also have to be pruned since their tendency to overfit combined with many features (as well as the possibility to split more than once on any given continuous attribute).

While decision trees conceivably could be constructed to do well on digit recognition, it does not lend itself naturally to this domain. An analogous algorithm to ID3 would fail miserably. So there are decision trees that can classify the digits and we’re sure there may be algorithms that adequately do lookahead and attribute combinations, but there are more natural choices for a domain that requires combinations of features to be effective.

**Boosted decision stumps**

Boosted decision stumps wouldn’t do well on this domain. Because boosted decision stumps split on only one attribute, they do not combine features and individually will not be able to capture patterns in the data well enough to classify digits. Individual stumps can do no better than chance since on their own they cannot predict what digit a particular pixel intensity is a part of.

Perceptrons

Perceptrons would be especially bad for recognizing digits. Since there are 10 possible digits to correctly classify, it presents too much variability for a single perceptron to classify. Perceptrons have more limitations beyond just being able to output binary values, which is a problem in a 10-class domain.

Multi-layer feed-forward neural networks

Multi-layer feed-forward neural networks are very suited for the domain of recognizing hand written digits. Because the digits are represented as a matrix of pixels, a neural network can propagate pixel intensities and calculate a decision boundary based on combinations of all of these features.

Moreover, the flexibility in designing feed-forward networks lends itself even further to classifying digits, allowing for custom hidden layers to compute and pass forward useful features like averaging/subsampling and feature maps downstream.

(a) Describe the network you chose to implement. This description need not be at the node by node level, but it should provide an overall view of the layers and units you created. Explain why you decided to run an experiment with this network structure.

We experimented with adding a second hidden layer with various numbers of hidden units. We ran experiments letting the two hidden layers have the same number of hidden units, ranging from a small number of hidden units to a larger number of hidden units. We also experimented with varying the number of hidden units in each layer, for example having 10 in the first and 15 in the second.

Having achieved success with one layer, we thought we could improve on this success by adding another hidden layer. We reasoned that the automatic feature engineering intuition associated with neural networks would result in the first hidden layer passing the second hidden layer an even better set of features which the second hidden layer can then do learning on.

(b) Train your network using the training and validation sets, and compare the trained network’s test performance to the previous structures. Try to explain what you ﬁnd. (You will not be evaluated on the absolute performance of your network.).

The neural networks we trained with two hidden layers universally had very poor performance on training, validation, and test sets. For all of the experiments we ran, all had training error, validation set error, and test error at 0.12 or worse. The networks would converge within three or four epochs: the errors would stop changing, indicating that the weights stopped changing. After carefully spending time carefully validating that our network was indeed constructed in the correct fashion, we determined that these poor results were not the result of a bug, but because using two hidden layers instead of one with this particular domain had very poor performance in general.

Although we