Applied data science. Lab 2. Peter Varshavsky

Problem 1.

1. The neural network gets the correct answer most of the time. I trained 100 networks for each of the Boolean operators and tested them against the correct outputs with the following results.

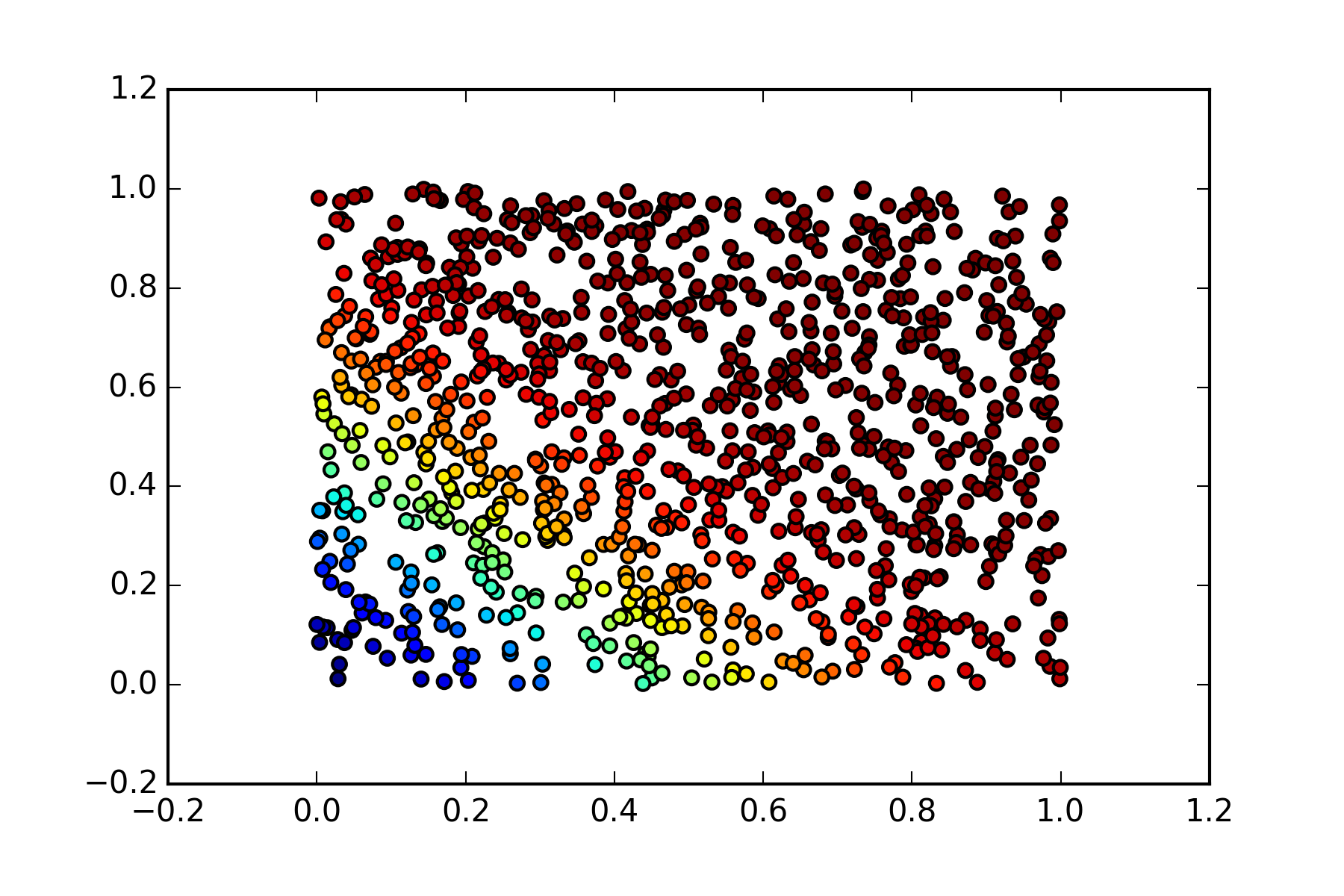
|  |  |  |  |
| --- | --- | --- | --- |
| not | and | or | nor |
| Fully accurate in 100 runs | Iteration 6  Error 0.750044  0.252743  0.252743  0.252743  0.252771  Iteration 88  Error 0.500115  0.489517  0.489311  0.005619  -0.008091 | Iteration 35  Error 0.751248  0.764003  0.764009  0.764010  0.764310 | Fully accurate in 100 runs |

These results are not reproducible because I did not set seed before the simulations, but they do show that the neural networks model the Boolean operations well.

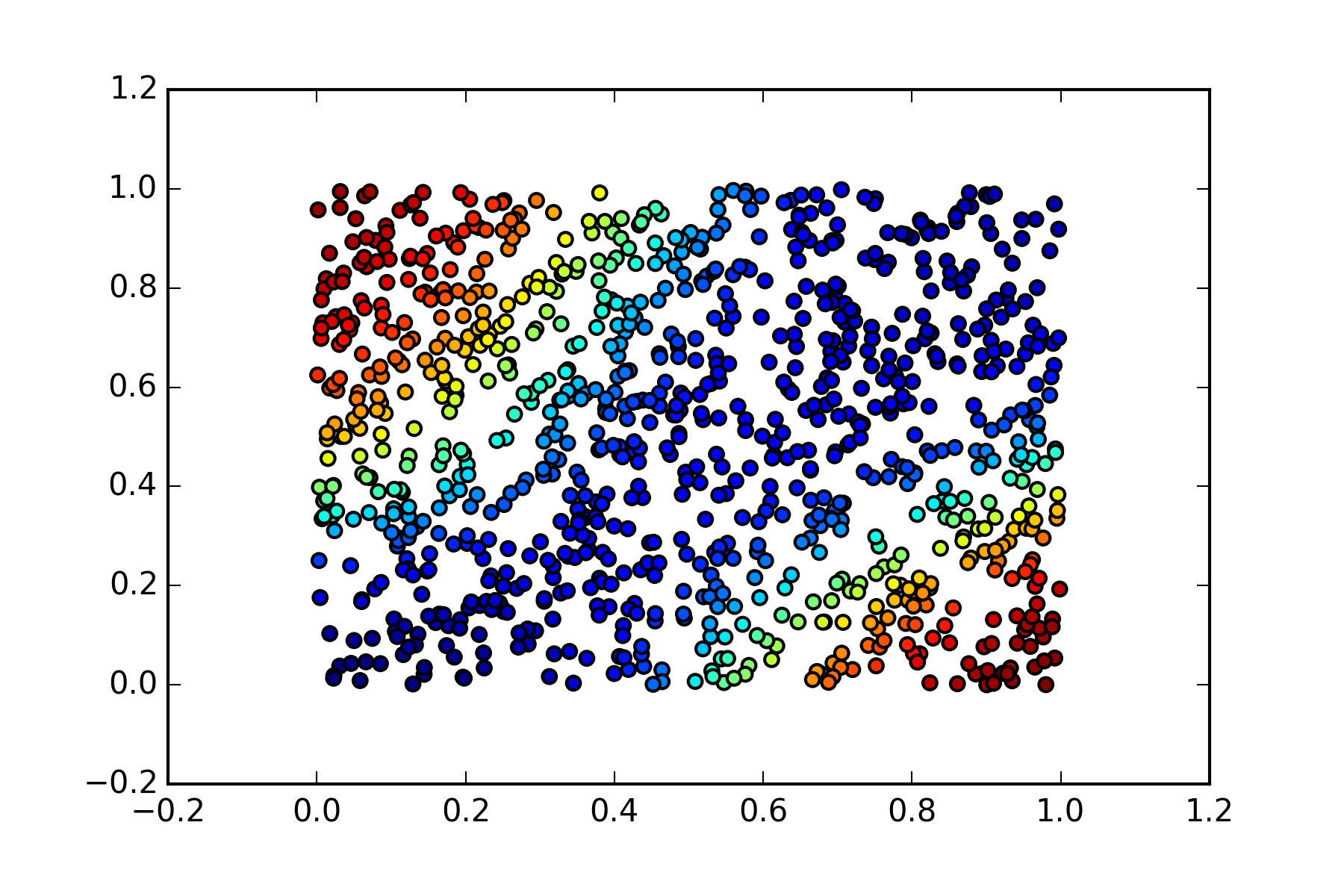
The code we were given uses a neural network with 2 input nodes, one hidden layer with 4 nodes and 1 output node, denoted (2, 4, 1). In addition it has bias nodes. Hidden layers allow the network to create nonlinear decision boundaries. Because OR, NOR, AND, and NOT are linearly separable, they can be modeled with a simpler network without a hidden layer. For example a (2,1) network would suffice. XOR, however is not linearly separable, so a hidden layer is required.

To illustrate the decision making by the neural networks, I plotted the the predictions made by OR and XOR networks trained on the lattice points of the ((0,0), (1,1)) square and then tested on a uniform random sample of points from that square.

Here is a plot of a (2,1) neural network with sigmoid output layer trained for OR operator and tested on a uniform random sample from the square ((0,0), (1,1)). This network is essentially the same as a logistic regression with two predictors.

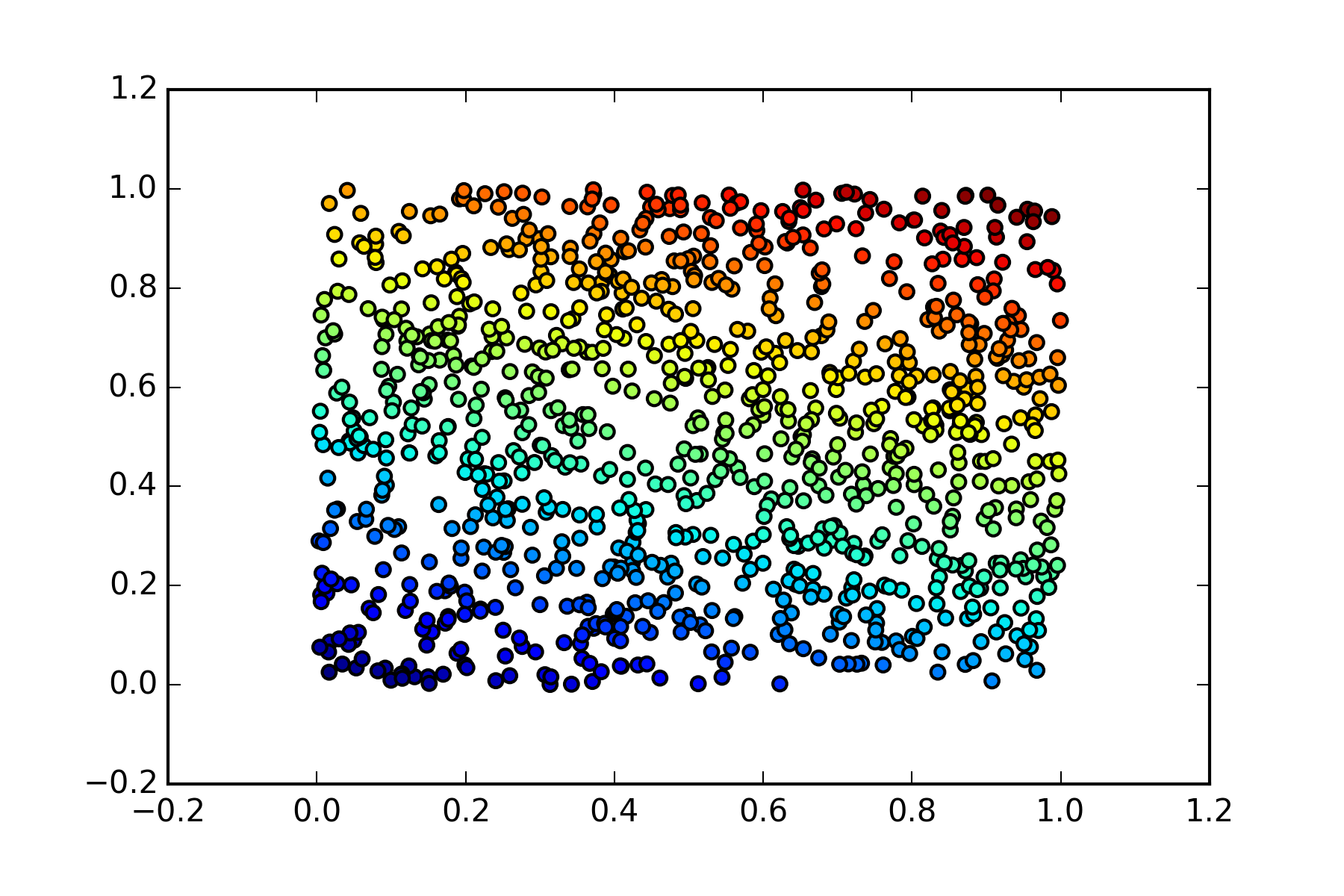


Predictions computed by a (2,4,1) XOR network with a sigmoid output layer can be seen to be not linearly separable in the following image.



1. Changing the numbers of hidden layers for OR, NOR, and AND does not significantly change the prediction quality, but adds computational expense if the number of hidden nodes is increased. The network for XOR becomes ineffective if the hidden node is taken away.

The image below shows the prediction made by a linear XOR network. Its prediction is a complete failure: 0.5020612, 0.50036557, 0.50138001, 0.49968437.



1. Composite Boolean operators
   1. (A AND B) OR C

The output of a (3,4,1) network is given below. It is a successful approximation of the correct result.

(1, 1, 1) [ 1.02140037]

(1, 1, 0) [ 0.98898213]

(1, 0, 1) [ 0.99397006]

(0, 1, 1) [ 0.99427853]

(1, 0, 0) [ 0.00162767]

(0, 1, 0) [ 0.00071983]

(0, 0, 1) [ 1.00415994]

(0, 0, 0) [ 0.00085638]

* 1. (NOT (A OR B)) AND C

The output of a (3,4,1) network is given below. It is successful.

(1, 1, 1) [-0.00046274]

(1, 1, 0) [ 0.00061117]

(1, 0, 1) [ 0.00018912]

(0, 1, 1) [ 0.00039458]

(1, 0, 0) [ 0.0001218]

(0, 1, 0) [-0.00019796]

(0, 0, 1) [ 1.0001051]

(0, 0, 0) [ 7.74848717e-05]

* 1. NOT ((A OR B) AND C)

The output of a (3,4,1) network is given below. It is successful.

(1, 1, 1) [ 0.]

(1, 1, 0) [ 1.]

(1, 0, 1) [ 1.11022302e-15]

(0, 1, 1) [ -1.11022302e-15]

(1, 0, 0) [ 1.]

(0, 1, 0) [ 1.]

(0, 0, 1) [ 1.]

(0, 0, 0) [ 1.]

Although a network with one hidden layer fit all of the composite operators in this assignment, and a number of other three dimensional Boolean operators, I do not believe the (*n*, 4, 1) network, where n is the number (or dimension) of inputs is universal. A paper by Martin Anthony (<http://www.cdam.lse.ac.uk/Reports/Files/cdam-2003-01.pdf>) states that any Boolean function can be computed by a 2-layer neural network with no hidden layers, but that would require a larger input layer than simply the dimension of the input of the Boolean function. There is also a lower limit on the number of nodes in a network of any architecture that would not be satisfied for some Boolean functions by (*n*, 4, 1) networks for a high enough *n*.

Problem 2.

1. For this problem I changed the cleaner.py code to output 40x40 pixel square images. I used 20 images of myself and 17 images of other faces manually cropped tightly to position eyes roughly in the vertical middle of the image. This decision was due to the fact that neural networks train and make predictions pixel by pixel (with complicated interactions introduced by the hidden layers), so their performance can be greatly improved if pixels in the same part of the image belong to the same facial features. With much larger training set, it may be possible to train a network to recognize faces in different parts of the image, but that would be wasting the resources – if face is in different part of the image, it makes more sense to use a feature finding algorithm to crop to face, and then attempt facial recognition.

The program was written to take all cropped and resized images of mine and others’ faces and assign 66% of them to the training set and the remaining 33% to the test set. On repeat executions, the program would train and test on different subsets of images.

The neural network given in Professor Schles’ code used linear output layer. I was not satisfied with the outputs using the linear layer, but got meaningful results when building a network with a softmax two-node output layer with value (1,0) for my face and (0,1) for not my face.

Some of the images had drastic sidelight or I used my palm to conceal part of my face. These were misclassified more often than evenly lit straight-on pictures.

Based on one execution of the training and testing, using two hidden layers with 80 (40\*40\*0.05) nodes in each, the network achieved perfect accuracy on the training set and 62% (8 out of 13) accuracy on the test set with all errors being false positives, that is the network tended to classify other faces as my faces. The network correctly identified a negative once recognizing that I am not Caetano Veloso. To better understand performance, the code should be executed multiple times retraining and retesting the network on different subsets of the images.

Using one hidden layer with 48 hidden nodes, the back propagation iterations were quicker, but more iterations were required to converge to the error threshold. The performance was exactly the same with perfect prediction accuracy on the test set and 62% accuracy on test set with all errors being false positives. The network correctly identified a negative once recognizing that I am not an 80-year-old Groucho Marx.

1. The network did not perform well with multiple faces, which is understandable since it does not learn to recognize my face, rather it learns a relationship between pixel locations and training output values. Once my face is not centered in the image and is occupying a smaller part of the image, it loses predictive value for the network.

OUTPUT WITH 2 HIDDEN LAYERS OF 80 NODES EACH

WITH A 2-NODE OUTPUT SOFTMAX LAYER

##### TEST SET PREDICTIONS #####

0Result (0.0000, 1.0000): (0.9565, 0.0435) groucho\_old.png

1Result (1.0000, 0.0000): (1.0000, 0.0000) pv16copy.png

1Result (1.0000, 0.0000): (1.0000, 0.0000) pv2copy.png

1Result (1.0000, 0.0000): (1.0000, 0.0000) pv23copy.png

1Result (0.0000, 1.0000): (0.4789, 0.5211) caetanocopy.png

1Result (1.0000, 0.0000): (0.9955, 0.0045) pv24copy.png

1Result (1.0000, 0.0000): (0.9935, 0.0065) pv10copy.png

1Result (1.0000, 0.0000): (0.9997, 0.0003) pv3copy.png

1Result (1.0000, 0.0000): (0.6955, 0.3045) pv21copy.png

0Result (0.0000, 1.0000): (0.9565, 0.0435) oldgrouchocopy.png

0Result (0.0000, 1.0000): (0.5525, 0.4475) thelonious.png

0Result (0.0000, 1.0000): (0.9257, 0.0743) coltranecopy.png

0Result (0.0000, 1.0000): (0.9594, 0.0406) picassocopy.png

##### TRAINING SET PREDICTIONS #####

Result (0.0000, 1.0000): (0.0038, 0.9962) frank-zappacopy.png

Result (1.0000, 0.0000): (0.9985, 0.0015) pv1copy.png

Result (1.0000, 0.0000): (0.9974, 0.0026) pv20copy.png

Result (0.0000, 1.0000): (0.0042, 0.9958) turingcopy.png

Result (0.0000, 1.0000): (0.0051, 0.9949) harpocopy.png

Result (1.0000, 0.0000): (0.9999, 0.0001) pv6copy.png

Result (1.0000, 0.0000): (0.9929, 0.0071) pv12copy.png

Result (0.0000, 1.0000): (0.0048, 0.9952) garbuscopy.png

Result (1.0000, 0.0000): (0.9952, 0.0048) pv13copy.png

Result (1.0000, 0.0000): (1.0000, 0.0000) pv22copy.png

Result (0.0000, 1.0000): (0.0072, 0.9928) einsteincopy.png

Result (1.0000, 0.0000): (0.9955, 0.0045) pv11copy.png

Result (1.0000, 0.0000): (0.9990, 0.0010) pv9copy.png

Result (1.0000, 0.0000): (0.9936, 0.0064) pv8copy.png

Result (1.0000, 0.0000): (0.9931, 0.0069) pv5copy.png

Result (0.0000, 1.0000): (0.0051, 0.9949) stvincentcopy.png

Result (1.0000, 0.0000): (0.9965, 0.0035) pv4copy.png

Result (1.0000, 0.0000): (0.9949, 0.0051) pv14copy.png

Result (0.0000, 1.0000): (0.0047, 0.9953) chaplincopy.png

Result (1.0000, 0.0000): (0.9989, 0.0011) pv7copy.png

Result (0.0000, 1.0000): (0.0044, 0.9956) milescopy.png

Result (0.0000, 1.0000): (0.0034, 0.9966) matissecopy.png

Result (0.0000, 1.0000): (0.0040, 0.9960) grouchocopy.png

Problem 3.

For animals I used 13 images of each animal 75% of each were randomly assigned to the training set, remaining to the test set. The training set was then shuffled. The images were cropped to face and downsampled to 40x40 pixels.

The results here are from a (1600, 80, 80, 5) network with error threshold of 0.0008 and a softmax output layer.

Out of 20 test cases, 6 were classified correctly. A confusion table would provide better representation of classification errors, but I ran out of time.

OUTPUT:

Picture cat\_7.png has a cat

Picture croc\_4.png was misclassified

Picture gorilla\_1.png was misclassified

Picture dog\_3.png has a gorilla

Picture dog\_12.png has a alligator

Picture dog\_8.png has a alligator

Picture croc\_5.png has a giraffe

Picture giraffe\_13.png has a giraffe

Picture giraffe\_2.png has a giraffe

Picture croc\_3.png was misclassified

Picture gorilla\_5.png has a gorilla

Picture croc\_9.png has a cat

Picture cat\_8.png has a giraffe

Picture gorilla\_2.png has a dog

Picture giraffe\_9.png has a giraffe

Picture cat\_13.png has a cat

Picture cat\_5.png has a giraffe

Picture giraffe\_10.png has a giraffe

Picture gorilla\_11.png was misclassified

Picture dog\_5.png has a cat

Problem 4.

a. Neural networks without hidden layers are the same as a generalized linear model where the link function of the glm is the activation function of the output layer. As such, the relationships they model (decision boundaries in the case of classification) are linear in parameters (weights). The introduction of hidden layers allows for nonlinear combinations of parameters and thus nonlinear decision boundaries, which is very useful in image recognition (face, handwriting, etc.). The parameters, however, are not useful for inference, since their combinations can not be used to explain causal relationships or even correlations in the model. Thus they can be useful if our only goal is predictive quality, but not useful when we would like to learn something about the relationships between the covariates.

b. It is hard to imagine statistics disappearing so long as we are interested in inference. Machine learning techniques expand the field, but they are not replacing classical statistics completely, since much of classical statistics is a subset of machine learning, and many machine learning methods came from statistics.