CS 440

NETID:
sap376 - 201008878
fg236 - 219002304

*NOTE: used base code from the UC Berkeley site

Commands:
python3 dataClassifier.py

At the start of the program, the user is prompted on if they would like to use *perceptron* or *neural* (for neural network), the amount of images to train from, the number of epochs that training will perform before finishing, and whether to use prime files (best possible weights pre-generated for each corresponding classification).

Features:
Our features are based on taking the image and separating it into pixels and designating whether it is a black pixel or a white pixel. Our data per image will be data points (coordinates) of black and white pixels.
Digits: 28 by 28
Faces: 60 by 70

Final Project

Perceptron:

For our implementation of perceptron, we first initialized our weights to random values between 0 and 1 for each label. This means that each possible label has its set of weights. Then we go through each set of data points(per image) and try to classify each image based on the weights. The classification is done by applying the weights of each label to their respective data point (including the bias weight) which will generate a score for every label per image. Then we take the maximum score, and classify the image with that label. We do this for all training images until all have been classified. After classification, we compare our

guesses to the actual truth label per image. If the guess equals the truth, nothing happens, but if the guess was not correct, we update the weights by decreasing the weights of the guess label and increasing the weights of the truth label. Specifically the non-bias weights are increased/decreased by themselves multiplied by a training value and the bias weights are increased/decreased by 1 depending on if it was the truth label or guess, respectively. After all the train images are gone through, the training will go through another epoch (full iteration over training images) if the max number of epochs has not been reached. To increase the accuracy of the perceptron, the training value is decreased by 5% after each epoch from its starting value of 1. After all epochs have been completed, training is finished and validation and testing will proceed.

Analysis

When testing images with perceptron, we saw an increase in accuracy as the amount of data used for training was increased. This was the case for both digits and faces, where accuracy for digits and faces climbed from 75% and 60.2% to 85% and 88.2% respectively. We also saw an increase in runtime during training as the amount of data used increased.

Digits (25)

% train	10	20	30	40	50	60	70	80	90	100
mean	75.6	77.2	79.4	80.6	84.4	83.6	86.0	83.4	84	85.0
std	1.01 9803 9027 1855 7	1.46 9693 8456 6990 67	1.019 8039 0271 8557	0.8	1.743 5595 7741 6269 3	1.959 5917 9422 6542 4	1.26 4911 0640 6735 18	2.41 6609 1947 1891 4	1.673 3200 5306 8151 1	2.0976 17696 34030 33
time (in seconds)	21.8	44	65.6	86.8	108.6	130	151. 4	173. 6	195.8	218.2

Faces (100 iterations)

% train	10	20	30	40	50	60	70	80	90	100
mean	60.2	75.4	76.4	85.0	81.0	88.2	88.0	89.4	88.6	88.2
std	8.588364221433556	4.029888335921977	7.4993333037010 7	1.78885438199983 17	6.4807406984078 6	1.1661903789690 6	2.097617696340303 3	1.356465996625053 6	1.0198039027185557	0.4
Time (sec)	15.2	31.2	47	64	82.8	96.4	113.4	132.2	147.0	164

Neural Network:

For our Neutral Network, we implemented a numpy array where the dataset containing the images passed into the train function was used to initialize the numpy array with the training data. The weights to be multiplied with the input layer array and hidden layer array are initialized with random numbers and are named weights1 array and weights2 array which correspond with their respective layers. The training data is then run through the classify function where forward propagation is performed and then the input layer and the hidden layer are dot multiplied with weights1 and weights2 respectively. After performing dot multiplication between the hidden layer and weights2, the output layer array is produced. At the end of forward propagation, classify returns its predictions in the form of the guesses array.

After the classify function returns its predictions, backward propagation is performed in the train function where the program calculates and finds delta3 and delta2 arrays which correspond to the output layer and the hidden layer respectively. Both delta3 and delta2 arrays are dot multiplied with the preceding layers, hidden layer and input layer respectively, where their respective gradients are produced then added to a corresponding gradient variable that is used to

compute the average gradient for the layer. After the average gradient is found, it is used to update the weights during each train epoch.

Analysis

When testing images with neural network, there was an average increase in accuracy, after more training data was used. From 10% where accuracy was around 75, to 100% where the accuracy was closer to 90. Although the accuracy was not increasing strictly linearly with more training data used, it did increase at least by 10% across the entire training dataset. However, with more training data used, runtime also increased.

Digits (500 iterations)

% train	10	20	30	40	50	60	70	80	90	100
mean	79.8	87.2	88.8	87.6	89.4	90.2	90.4	89.8	90.8	89.2
std	2.135415650406262	1.720465053408553	0.9797958971132712	1.74355957741626 93	1.0198039027185557	0.9797958971132712	2.244994432064364 4	1.469693845669906 7	1.7204650534085253	1.720465053408553
time (in seconds)	10.4	19.2	23.6	32.4	34.4	40.6	48.8	57.6	56.6	61.2

Faces (100 iterations)

% train	10	20	30	40	50	60	70	80	90	100
mean	74.4	79.2	84.4	83.8	87.6	85.6	88	90.4	90.4	87.4
std	1.624807 6809	5.153639 4906	1.3564659662	2.78567765	2.3323807 5793	4.3174066 2898	1.788854 3819	0.8	1.3564659662	1.356465996 62505

	27192	900505	50536	54368237	812	458	998317		50536	36
Time (sec)	2	2	3	3	3	3.2	4	4.4	4	5.4

Discussion

Based on our results and analysis, our neural network performs on average for both faces and digit classification, better than perceptron. It has both better accuracy and better runtime. This is probably due to the use of numpy arrays in the neural network and the fact that perceptron has weights for every possible label while neural network just has 1 set of weights that will have to classify correctly each label. This will require more precision with the neural network weights and thus lead to more accurate results. Also, due to efficient space complexity, neural networks have a better runtime than perceptron. One possible improvement that we can make is to add a streak counter that tracks the accuracy of the program during each epoch and keeps a count of how often the program is predicting a certain percentage of accuracy at the end of each epoch then when streak reaches a certain number, the program will end training prematurely. This is to avoid moments when the program is running and already has a high accuracy, such as when running with 99% accuracy with the training data and the program will continue running until all epochs have been completed. This can help to prevent overcorrection for perceptron and overfitting for neural network.

For the lessons learned, we realized that the efficiency of a multi-layer neural network has far better performance with both runtime and accuracy. We also learned the efficiency of mathematical operations when done with numpy arrays, as well as their ability to handle mathematical operations across multiple vectors. The ability to implement a neural network as well as the feature extraction helped us understand the structure of the network from the lecture slides.