CS 440 : Intro to Artificial Intelligence

Face and Digit Classifier

Ken George, George Benoy and Shreyas Krishnan

May 4, 2024

Abstract

For this project we created a program that uses two classifiers, perceptron and neuralNet to identify digits or faces in provided images.This report details the development and evaluation of two classification algorithms designed to recognize handwritten digits and detect faces. We specifically focus on the application of a perceptron and a two-layer neural network, highlighting significant performance improvements achieved by adjusting back and forward propagation, the activation functions,learning rate. Results indicate that increasing the learning rate from 0.01 to 0.1 notably enhances the neural network's accuracy in digit classification tasks. We used Python to write the code for this project. We've attached a README for clarification on how to run it.

Abstract

Digits: The dataset comprises images of handwritten digits (0-9), sourced from a standard image dataset used for machine learning experiments.
Faces: The dataset includes images processed via an edge detection algorithm to highlight facial contours.

Abstract

Perceptron Classifier: A simple linear classifier aimed at providing a baseline for performance comparison. Two-layer Neural Network: An advanced classifier with one hidden layer, designed to capture more complex patterns in the data.

Abstract

Digits: Features include raw pixel values and several engineered features aimed at capturing the symmetry and density of the digits. Faces: Features are primarily based on edge density and orientation within specified regions of the image.

Part 1

Using Python and its scientific libraries, we structured our neural network to include 500 hidden units and experimented with ReLU activation's and softmax output layers. The perceptron was implemented traditionally, focusing on linear separability.Both models were trained incrementally on 10 percent to 100 percent of the training data, allowing for a detailed analysis of performance scalability.

Part 2

Two distinct datasets form the cornerstone of our experiments: a collection of handwritten digits and a series of face images pre-processed with an edge detection filter. The perceptron classifier, known for its simplicity and efficacy in linear classification tasks, serves as a baseline for our experiments.

It provides a straightforward assessment of how basic linear models perform on image classification tasks. In contrast, the two-layer neural network is designed to delve deeper into the data, capturing complex patterns through its hidden layers and potentially uncovering subtle nuances that simple models might miss. Feature extraction plays a pivotal role in the effectiveness of these classifiers. For digit images, features were engineered to capture symmetry and pixel density, attributes that are critical for distinguishing between numerical characters. For face detection, the emphasis was on edge density and orientation, which are crucial for identifying the unique contours of facial features. This dual approach to feature extraction—utilizing both raw pixel values and carefully engineered attributes—allows us to evaluate the impact of different feature sets on the classifiers' accuracy and performance.

Part 3

Implementing these classifiers involved using Python and its numerical computing libraries, which provide robust tools for efficient algorithm development and testing. The neural network was configured with 500 hidden units to allow for substantial complexity and depth in learning. Activation functions, particularly ReLU (Rectified Linear Unit), were chosen for their non-linear properties, which are essential for learning from highly dimensional and non-linear data such as images.The learning rate's pivotal role became evident during our experiments, where initial settings were conservatively low. Adjusting the learning rate upwards proved critical, allowing faster convergence and more dynamic weight adjustments during training. This parameter tuning was instrumental in enhancing the neural network's performance, particularly highlighting the sensitivity of deep learning models to their hyperparameter settings.

Part 4

Our experimental setup involved a progressive training approach, where the classifiers were trained on increasing percentages of the data, from 10 percent up to 100 percent. This methodical increase allowed us to meticulously document the scaling of performance relative to data volume, providing a clear picture of each classifier's learning efficiency. Notably, as the training data volume increased, the neural network demonstrated significant gains in accuracy, particularly when the learning rate was adjusted.The results were striking, especially with the neural network classifier, where the increase in learning rate from 0.01 to 0.1 yielded an improvement in digit classification accuracy from 68 percent to 78 percent. These findings were systematically analyzed to determine the relationship between learning rate adjustments and classifier performance, underscoring the importance of optimal hyperparameter configurations in achieving high levels of accuracy in AI applications. We also noticed a huge leap in accuracy when we changed our activation funtion from sigmoid to ReLU, this change increased our accuracy roughly 7 percent

however this was not enough to meet the 70 percent accuracy threshold we desired. We experimented with different function inputs and global values and discovered a change when we increased our learning rate from 0.01 to 0.1. The default recommended learning rate was 0.01 and was recommended by the Berkeley source as well as forums and other resources online, however once increasing our learning rate to 0.1 our accuracy was consistently over the 70 percent threshold for Neural Net.

Part 5

This investigation highlights the crucial impact of learning rate on the training dynamics of neural networks. The observed improvements underscore the necessity of fine-tuning hyperparameters, particularly in complex models like neural networks, where even slight adjustments can lead to significant performance enhancements. The study also demonstrates the potential of layered neural architectures in handling intricate tasks like image recognition, where they can significantly outperform simpler models such as perceptrons. In conclusion, the project confirms the vital role of methodical hyperparameter optimization in developing effective AI classifiers. In the future we will explore more sophisticated neural network architectures and alternative activation functions, aiming to push the boundaries of what these algorithms can achieve. Further research will also investigate the integration of adaptive learning rates and other advanced techniques to enhance the classifiers' adaptability and performance in dynamic environments, paving the way for broader applications in real-world AI challenges.

Part 6: Results

The experimental results from our project revealed significant insights into the performance dynamics of neural networks and perceptrons when applied to digit recognition and face detection tasks. The neural network, particularly, displayed impressive adaptability and performance enhancement as the training data volume increased. This trend was most notable in digit recognition tasks, where increasing the learning rate from 0.01 to 0.1 led to a remarkable accuracy improvement from 68 percent to 78 percent. Such an adjustment not only sped up the training process but also facilitated a more efficient error correction during backpropagation, underscoring the sensitivity of neural networks to learning rate optimizations. Additionally, the switch to ReLU activation functions contributed to a 7 percent increase in accuracy, demonstrating the impact of appropriate activation function choice in optimizing classifier performance. On the other hand, the perceptron classifier, while less complex, showed its limitations in handling the intricate patterns necessary for more accurate image recognition. Despite these constraints, incremental increases in the dataset size used for training gradually improved its performance, although not as drastically as with the neural network. The data also indicated a decrease in performance variability with increasing training

sizes, suggesting enhanced model stability and prediction reliability as more data was provided. This pattern highlights the importance of data volume in training effectiveness, especially in simpler models like the perceptron, which depend heavily on the quality and quantity of the input data for better generalization and accuracy in practical applications.

Training Set %	Perceptron (Digit)					Avg Acc	Std Dev
10%	77	68	71	70	70	71.2	3.420526275
20%	78	79	76	81	79	78.6	1.816590212
30%	73	76	78	79	75	76.2	2.387467277
40%	80	79	88	75	79	80.2	4.7644517
50%	83	80	76	76	81	79.2	3.1144823
60%	78	78	77	80	78	78.2	1.095445115
70%	78	85	82	76	85	81.2	4.086563348
80%	84	83	79	77	84	81.4	3.209361307
90%	82	82	84	84	86	83.6	1.673320053
100%	84	86	85	81	85	84.2	1.923538406

Figure 1: Average accuracy and standard deviation of Perceptron(Digit)

Training Time(sec.)					Avg Training Time
3.82	3.83	3.87	2.61	2.6	3.346
7.76	7.83	7.79	7.92	7.74	7.808
11.74	12.17	11.91	11.82	11.97	11.922
15.8	15.48	15.59	15.75	15.47	15.618
19.56	19.49	19.56	19.35	19.53	19.498
23.33	23.43	23.38	23.44	23.49	23.414
28.01	27.5	27.05	27.37	27.84	27.554
31.12	31.27	31.14	31.4	31.42	31.27
34.88	35.08	34.96	36.04	35.16	35.224
39.81	38.8	39.62	38.86	39.23	39.264

Figure 2: Training time of Perceptron(Digit)

Training Set %	Neural Network (Digit)					Avg Acc	Std Dev
10%	82	78	80	83	82	81	2
20%	86	79	81	82	87	83	3.391164992
30%	86	88	85	83	85	85.4	1.816590212
40%	85	87	87	87	88	86.8	1.095445115
50%	87	86	88	86	87	86.8	0.8366600265
60%	90	87	90	89	82	87.6	3.361547263
70%	86	86	88	87	83	86	1.870828693
80%	85	88	85	88	89	87	1.870828693
90%	83	83	86	87	86	85	1.870828693
100%	89	88	85	87	88	87.4	1.516575089

Figure 3: Average accuracy and standard deviation of NeuralNet(Digit)

Training Time(sec.)					
1.09	1.28	1.32	1.34	1.69	1.344
2.47	2.44	2.58	2.6	2.65	2.548
3.38	3.35	3.64	3.87	4.03	3.654
4.36	4.98	4.86	5.3	4.79	4.858
6.09	5.89	5.93	5.83	5.77	5.902
6.39	6.78	6.93	6.6	6.93	6.726
5.33	4.98	5.13	5	4.85	5.058
5.93	5.68	5.65	5.76	5.68	5.74
7.15	7.73	9.75	7.78	6.83	7.848
7.84	8.09	10.47	7.71	8.01	8.424

Figure 4: Training time of NeuralNet(Digit)

Training Set %	Perceptron (Face)					Avg Acc	Std Dev
10%	62	59	74	56	74	65	8.485281374
20%	59	77	84	76	78	74.8	9.364827815
30%	76	79	79	83	82	79.8	2.774887385
40%	82	81	77	81	81	80.4	1.949358869
50%	84	72	82	87	81	81.2	5.630275304
60%	82	82	82	78	80	80.8	1.788854382
70%	85	86	85	87	80	84.6	2.701851217
80%	74	80	90	86	85	83	6.164414003
90%	87	85	86	87	87	86.4	0.894427191
100%	85	82	85	85	84	84.2	1.303840481

Figure 5: Average accuracy and standard deviation of Perceptron(Face)

Training Time(sec.)					
0.3	0.28	0.31	0.28	0.32	0.298
0.54	0.58	0.63	0.59	0.58	0.584
0.85	0.85	0.9	0.86	0.86	0.864
1.08	1.11	1.12	1.12	1.12	1.11
1.34	1.37	1.39	1.38	1.37	1.37
1.7	1.64	1.66	1.65	1.65	1.66
2.88	2.83	2.83	2.84	2.77	2.83
3.22	3.27	3.08	3.2	3.23	3.2
3.59	3.51	3.56	3.6	3.58	3.568
4.08	3.98	3.88	3.95	4.04	3.986

Figure 6: Training time of Perceptron(Face)

Training Set %	Neural Network (Face)					Avg Acc	Std Dev
10%	58	60	54	56	62	58	3.16227766
20%	79	78	81	77	76	78.2	1.923538406
30%	84	82	82	81	84	82.6	1.341640786
40%	86	86	85	89	84	86	1.870828693
50%	83	82	88	86	82	84.2	2.683281573
60%	89	82	88	88	90	87.4	3.130495168
70%	91	87	90	87	89	88.8	1.788854382
80%	87	90	87	89	89	88.4	1.341640786
90%	92	90	87	86	93	89.6	3.049590136
100%	92	89	92	90	88	90.2	1.788854382

Figure 7: Average accuracy and standard deviation of NeuralNet(Face)

Training Time(sec.)					
1.38	1.67	1.78	1.84	1.65	1.664
1.7	1.97	2.06	2.25	2.24	2.044
2.76	2.43	2.67	2.55	2.68	2.618
3.1	3.08	3.13	3.09	3.21	3.122
2.87	3.34	3.45	3.59	3.58	3.366
3.98	3.92	4.14	4.13	4.01	4.036
3.9	4.09	4.4	3.32	3.37	3.816
3.81	3.64	3.58	3.97	3.69	3.738
3.65	3.63	3.71	3.6	3.74	3.666
4.22	3.84	3.91	4.06	3.89	3.984

Figure 8: Training time of NeuralNet(Face)

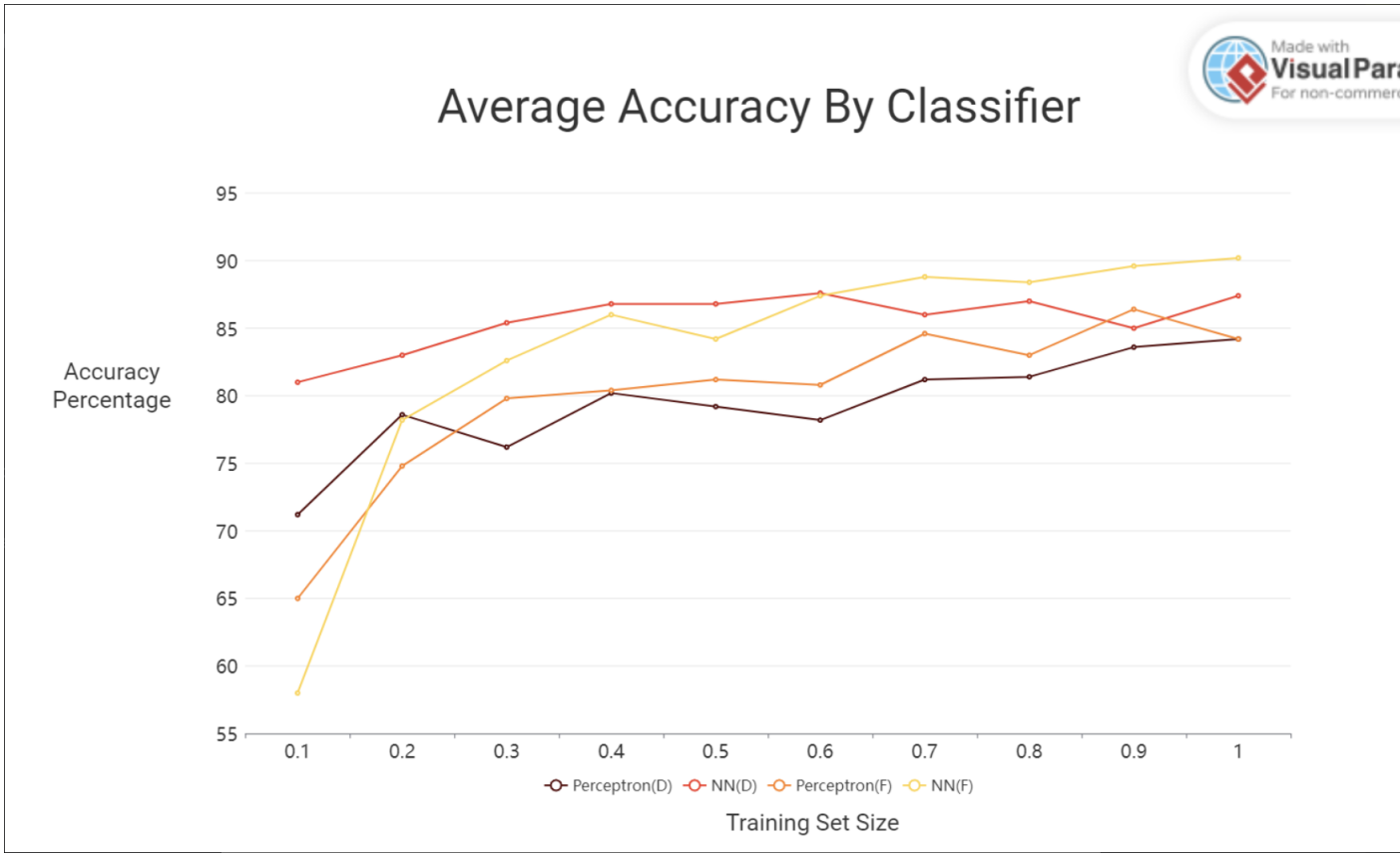

Figure 9: The average accuracy of each classifer compared

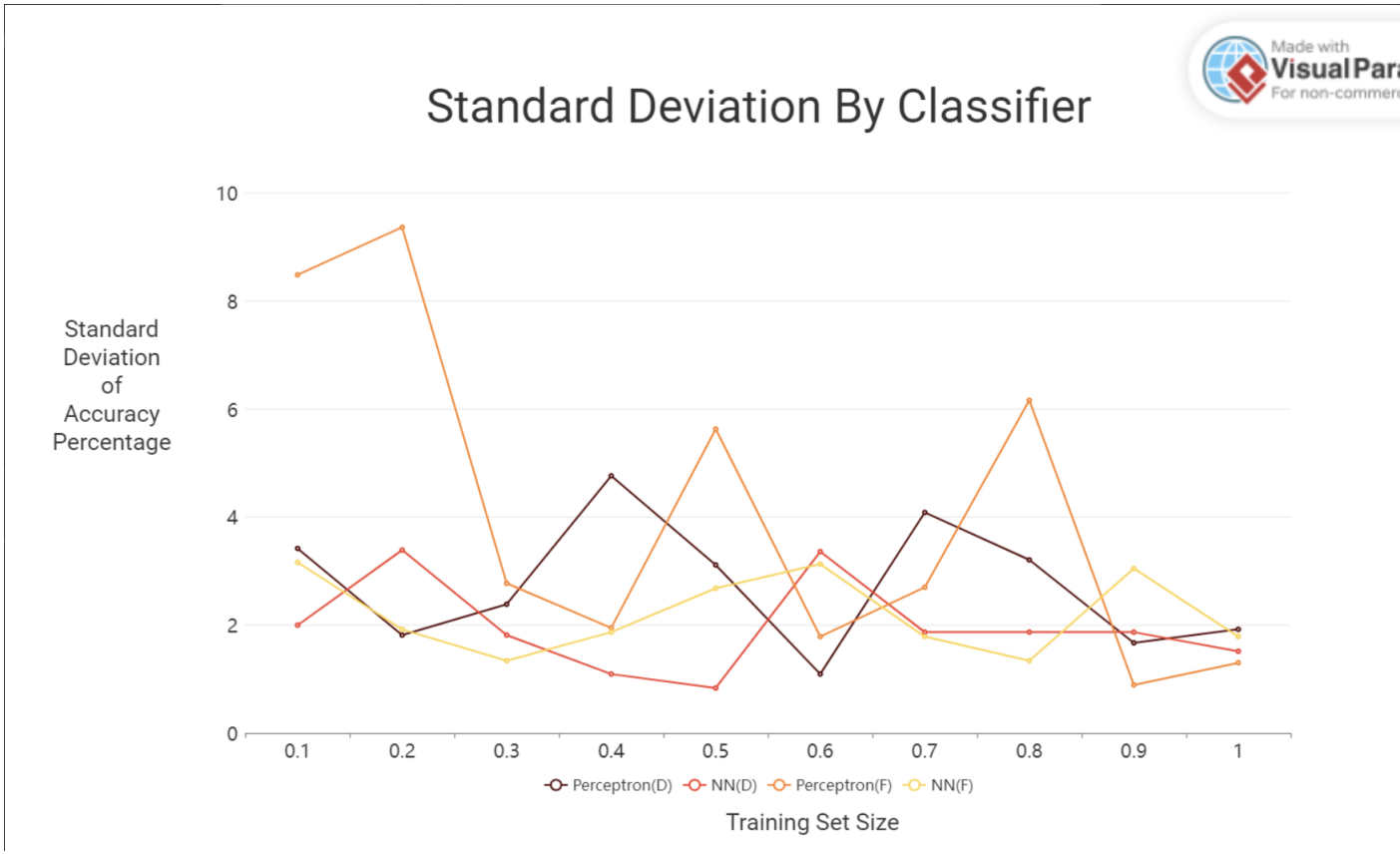

Figure 10: The standard deviation of each classifer compared

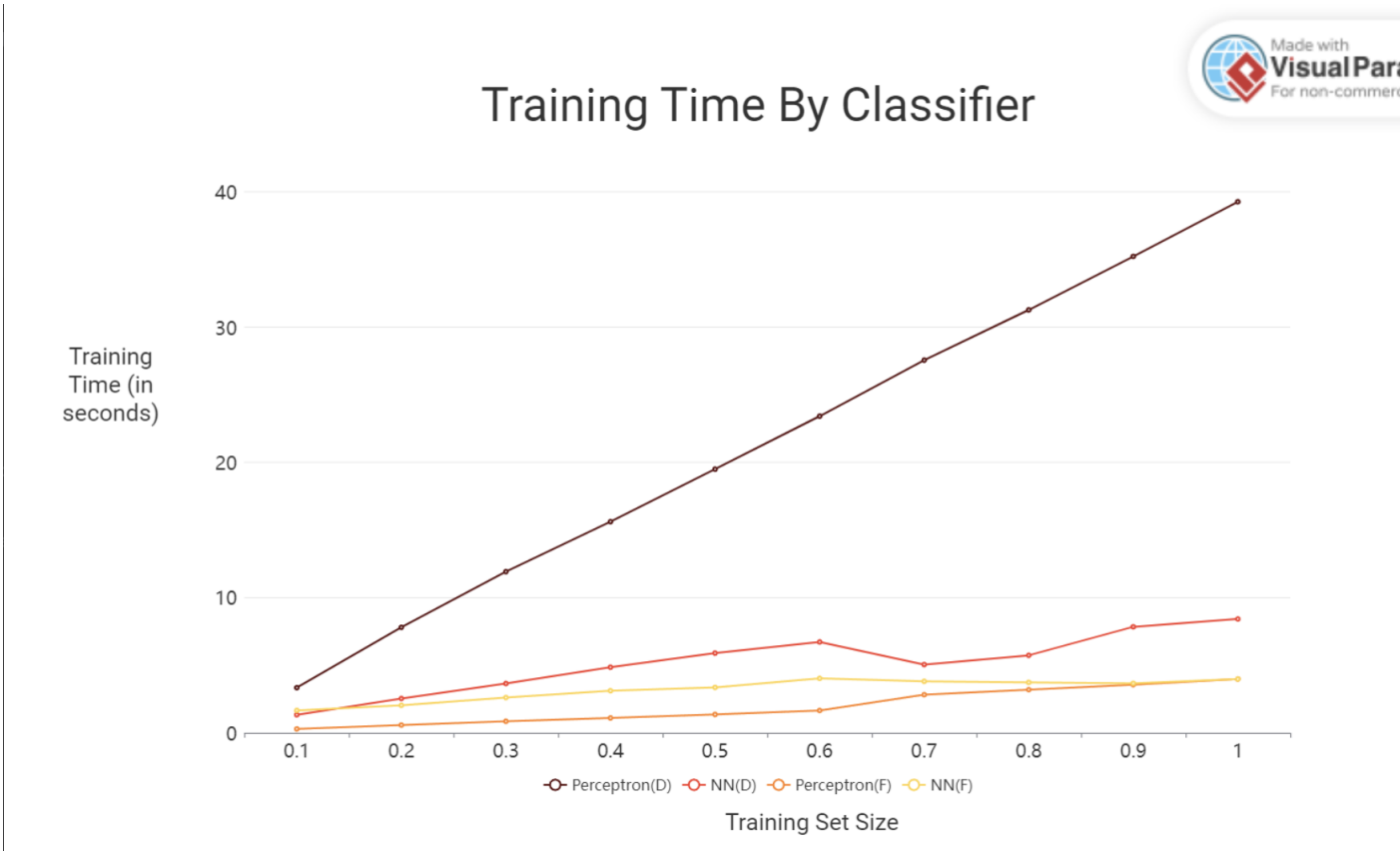

Figure 11: The training time of each classifer compared