Digit Recognizer

Min Cheol Kim, Katie Hahm
CS221 (MNIST database, Kaggle competition)

Problem Statement: Create an AI that can take in a single hand-written digit and output the value of that digit

Introduction

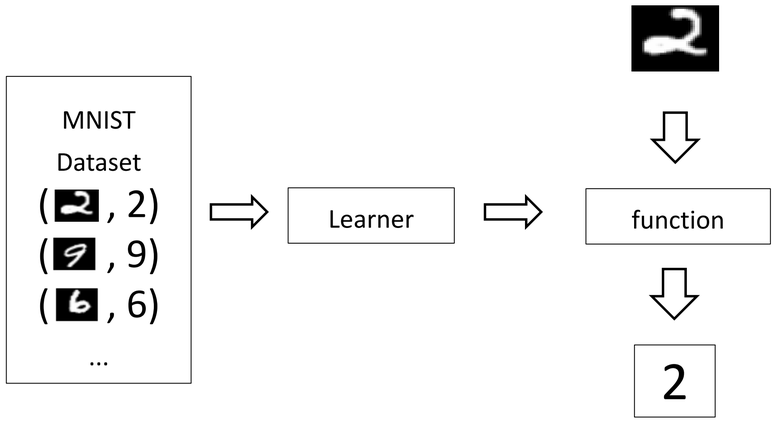
Interpreting handwritten pieces of writing is a active area of research in image analysis for betterment of uses in postal-address recognition, automatic bank-check recognition, and more. This project will focus on a small aspect of that endeavor, classifying a single handwritten digit into one of 0, 1, 2 …9. Our interest in this subject was started by a competition that we saw on Kaggle.com on classifying these digits using the MNIST dataset.

The MNIST dataset contains several ten thousands of handwritten images with their correct labels as training data. There has been many proposed approaches, including support vector machines (SVMs), neural networks, convolutional neural networks, etc. Using mostly neural networks and convolutional neural networks, one group was able to get an error rate of 0.23% on the testing set of 10,000 images. For our final project, we wanted to learn about some of these approaches and actually implement them from scratch.

Definitions

- **Input:** an upright, grey-scaled image of a single handwritten digit from 0-9
- **Output**: The value of the digit the computer determines
- **Performance measure:** Percent accuracy in predictions of the testing set.

Overview



Experimental Setup

In this project, we used three different approaches to the problem:

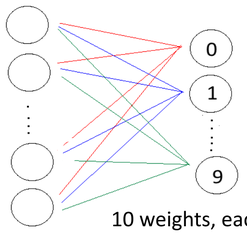
1. A multiclass linear classifier, as we learned briefly in lecture, that uses a hinge loss function and 10 different set of weights representing each digit.
2. A multilayered perceptron, with a single or multiple hidden layers, using the raw pixel values as inputs and the classification scores as output.
3. An autoencoder paired with the multiclass linear classifier mentioned in 1), using a neural network to perform feature learning.

Once these ideas were implemented, we shifted our focus to testing and tuning our hyperparameters, along with thinking about how to explain the results.

Approaches

Multiclass Linear Classifier

Labels = {0, 1, 2, … , 9}
Weights = {w_0, w_1, w_2, … , w_8, w_9}
Predictor $f_w(x) = \arg\max_y \{ w_y \cdot \varphi(x) \}, y \in \{0, 1, \ldots, 9\}$
$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max_{y'} \{ \mathbf{w}_{y'} \cdot \phi(x) - \mathbf{w}_y \cdot \phi(x) + \mathbf{1}[y' \neq y] \}$.



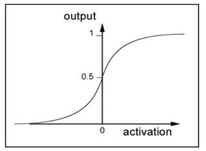
Output nodes represent digits

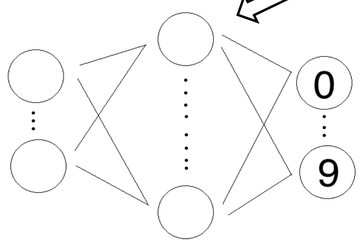
Pixel values as input nodes

10 weights, each mapping to one output node

Multilayered Perceptron

- Feedforward neural network
- Sigmoid as the activation function
- Quadratic cost function for error calculation
- Backpropagation with mini-batch stochastic gradient descent to update weight/bias matrices



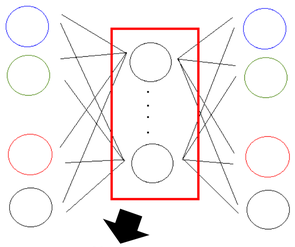


Pixel values as input nodes

Output nodes represent digits

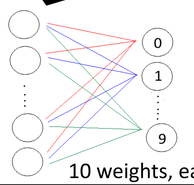
Single hidden layer

Autoencoder / Multiclass Linear Classifier



Pixel values as input nodes

Output nodes attempt to match input values



Hidden nodes from the auto-encoder are used as input nodes in the linear classifier

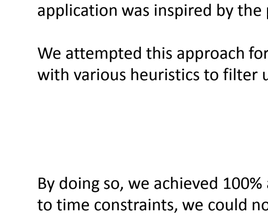
Output nodes represent digits

10 weights, each mapping to one output node

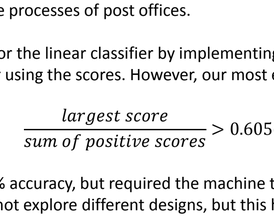
Analysis

Best Results

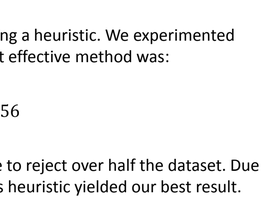
Linear Classifier
Best Performance: 0.89
NumIters: 40
Eta = 0.5



Multilayered Perceptron
Best Performance: 0.96
# Hidden Nodes: 100 (1 layer)
NumIters = 20
Eta = 3.0



Auto-encoder/LC
Best Performance: 0.87
Hidden Nodes: 400 (1 layer)
NumIters: 10
Eta = 3.0



Discussion/Error Analysis

We were not given the full MNIST dataset; therefore, it was difficult to compare our results to the literature values available for classifying this dataset. We found that using the multiclass linear classifier gave a surprisingly high accuracy, and that using multilayered perceptron gave a reasonable accuracy (from literature).

However, we found it odd that the autoencoder yielded lower accuracies than the multiclass linear classifier. We believe that this is caused by our context of using supervised and unsupervised learning. Because we had a large set of labeled data, a linear classifier trained on the whole set performed better than a linear classifier being trained on learned features.

We conducted a follow-up experiment where we reduced our labeled training data drastically to size 50, while treating the rest of our original data as unlabeled. The difference is small, but this demonstrates that the use of the autoencoder for feature learning is beneficial under certain environments.

	Linear Classifier	Autoencoder
Accuracy	0.54	0.56

Extension

In practical applications of digit recognizing machines, it is important to achieve a near perfect recognition of handwriting. Time is also an issue in practical applications, as it would be ideal for the computer to read the images quickly. Therefore, we performed another follow-up test where the linear classifier would reject all images that are uncertain to achieve accuracy and speed. This application was inspired by the processes of post offices.

We attempted this approach for the linear classifier by implementing a heuristic. We experimented with various heuristics to filter using the scores. However, our most effective method was:

$$\frac{largest\ score}{sum\ of\ positive\ scores} > 0.6056$$

By doing so, we achieved 100% accuracy, but required the machine to reject over half the dataset. Due to time constraints, we could not explore different designs, but this heuristic yielded our best result.

Challenges

We documented some challenges we encountered.

- Speed of execution: because the MNIST dataset is large, we had to pay close attention to the decisions we made regarding the type of variables, number of loops, and others that may cause time delay. One solution was to use numpy matrix operations to speed up calculations.
- There were many iterations we had to run to find the optimal number of iterations, eta, etc. We had to spend a lot of time waiting for the computer to execute our code. Tuning the neural networks were difficult.