University of Michigan – Dearborn
Department of Computer and Information Science
CIS 581: Computational Learning
(Winter 2023)
Instructor: Luis E. Ortiz

Project 2: Handwritten Digits Recognition using Neural Networks

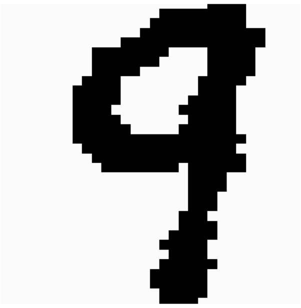
Overview

In this project you will use neural networks to learn to recognize optical handwritten digits.

Handwritten Digits Dataset

The data set of examples consists of images of handwwritten digits represented as a 32x32 matrix of pixels, each pixel taking value 1 or 0 corresponding to a black or white pixel, respectively.

The following is an example for the digit 9:



For each image, each pixel corresponds to a binary attribute, with values 0 or 1, to be treated as real-valued. So for the original data set, each input consists of $32 \times 32 = 1024$ (binary) attributes.

You are provided six (3) data files.

- `optdigits_train.dat` contains (a permuted version of) the training data.
- `optdigits_test.dat` contains the test data.
- `optdigits_trial.dat` contains an example of each digit from the test data set.

Each file is composed of one data example per line. Each line of the files but now each line contains 1025 integers separated by a single space. The first 1024 correspond to the values of a bitmap of the image, where each consecutive sequence of 32 values, starting with the value in the first column, corresponds to a row of black (0) and white (1) pixel values of the image of the handwritten digit. The value of the digit is given as the last element of the 1025 vector/row. A proper reshaping of the vector composed of the first 1024 binary values in the file-row will produce a 32x32 black-and-white image. The training, test, and trial data sets have 1934, 946 and 10 examples, respectively.

Neural Networks

You are asked to use/apply GradientDescent, ForwardProp and BackProp for evaluating and fitting neural networks. The *proxy error function* used for learning is the *mean squared error: that is, the average (over the samples) of the avergage (over the otuput units) of half of the squared error.* The *output of gradient descent* should be the *neural network with the lowest poxy error value* *on the data used for fitting over all iterations.* You have the option to use the implementation provided (*preferred* option — see **The ML Neural-Network Black-Box** section); use another library with an equivalent implementation as the one described below; or implement the methods from scracth.

Model and Implementation Details. All the neural network *units* are *sigmoid* (i.e., use sigmoid/logictic activation functions).

Neural Network Architectures: You will consider three neural network architectures as model classes:

1. A *(multi-output) perceptron* with 1024 input units and 10 output units, with *(gradient descent) learning rate* $\in \{4^i \,|\, i = 0, 1, 2, 3, 4\}$, at least
2. A *deep neural nework with the same number of units per hidden layer* $\in \{4^2, 4^3, 4^4\}$, *network depth (i.e., number of hidden layers)* $\in \{1, 2, 3, 4\}$, and *(gradient descent) learning rate* $\in \{4^i \,|\, i = -2, -1, 0, 1, 2\}$, at least
3. A *deep neural nework with* 2 *hidden layers*, with *the number of units in the first layer* $\in \{4^3, 4^4\}$ (connecting *from* the input layer), followed by a **smaller** *number of units* $\in \{4^2, 4^3\}$ *in the next layer* (connecting *to* the output layer), and *(gradient descent) learning rate* $\in \{4^i \,|\, i = -3, -2, -1, 0, 1\}$, at least

Model Selection: Use *cross validation (CV) with* 3 *folds,* at least, for model selection based on *misclassifiation error.*

Weight Initialization and Stopping Conditions: Initialize each neural network weight (including thresholds) with a number independently drawn at random from the uniform distribution over $[-1, 1]$. As discussed during lecture, set the *number of iterations of gradient descent* adaptively for each architecture in such as way that running 50 *iterations with a dataset of size* 2000 *on an architecture with* 4 *hidden layers and* 512 *units per layer takes about* 1 *minute, but with a maximum of* 1000 *iterations.*

Evaluation and Report. For each of the network architectures described above, include at least the following in your report.

1. GradientDescent **Fitting Curves:** Using all the trainning data, for the *best* model class selected,
   (a) Record
      i. the training and test proxy error function values and
      ii. the training and test misclassification error rate

      for the neural network as it is fitted at each round of GradientDescent.

   (b) Use your recorded values to plot
      i. the training and test proxy error function values together in one graph and
      ii. the training and test misclassification error rate together on a separate graph

      as a function of the number of rounds of GradientDescent. *Include these two graph plots, along with a brief discussion of the results, in your report.* Also, *tabulate and include the training and test proxy error function and misclassification error values for the best/final neural network learned in your report.*

(c) Evaluate each example in the trial data set on the best/final neural network learned and *report the corresponding output classification.*

2. [***OPTIONAL***] **CV Error Curves:** Tabulate and provide appropriate visualizations (plots) of the *average CV test misclassification errors.*

3. [***OPTIONAL***] **Learning Curves:**

   (a) Considering the first m examples in the training data *only*, with

   $$m \in \{10, 40, 100, 200, 400, 800, 1600\}, \text{ at least,}$$

   do the same asked for GradientDescent **Fitting Curves** and **CV Error Curves**.

   (b) Then, for *each type of error separately*, plot the resulting *learning curves*: that is, plot the resulting training and test error values as a function of m, on the same plot. *Include these two graph plots, along with a brief discussion of the results, in your report.*

4. [***OPTIONAL***] **Weight Parameter Interpretation:** Using all of the training data,

   (a) for the final perceptron, interpret the weight parameters for each output unit as a 32x32 (grayscale) image; and

   (b) for each of the final deep neural networks, randomly choose 10 of its units in the first hidden layer connecting from the input layer, and interpret the weight parameters for each unit chosen as a 32x32 (grayscale) image.

The ML Neural-Network Black-box

The "ML Neural-Network Black-box" is the python code for basic fitting, prediciton, and evaluation of neural network models discussed in considerable detail during lecture. Students are *strongly encouraged* to use that code to build their program to complete this project. Please use the corresponding lecture video as reference.

What to Turn In

You need to submit the following.

1. A **written report** (*in PDF*) that includes the information/plots/graphs/images requested above along with a brief discussion of the results. In your discussion, compare and contrast the different classifier models and learning algorithms.

2. All your **code and executable**, excluding the source code and script files provided (as a tared-and-gziped compressed file), with instructions on how to compile/run your program. If you implemented to methods from scratch: A platform-independent executable is preferred; otherwise, also provide instructions on how to compile your program. Please use standard tools/compilers/etc. generally available in most popular platforms.