AMSC808N/CMSC828V Maria Cameron

Project 1. Due Sunday, Nov. 6

The goals of this project are

- to acquire practice in working with real data;
- to explore various optimization methods for solving classification problems and understand how their performance is affected by their settings;
- to get exposure to solving PDEs using neural networks;
- to experiment with a nonlinear least squares problem.

What to submit. Please submit one report per working group of 2–3 students with figures and comments. Every group member should link her/his codes to the report pdf. These can be e.g. Dropbox links or GitHub links, etc. All optimizers should be coded from scratch.

Programming language You can use any suitable programming language. Matlab or Python are preferable. If you are going to use Matlab, you can use `mnist.mat` as input. If you are using Python, you can setup reading binary files in it. You do not have to remove the 4-pixel padding. Or you can convert data files from `mnist.mat` to any format convenient for you for reading in Python. I provide some objective functions and their derivatives in Matlab. These are short and can easily be rewritten in Python.

1 MNIST dataset

You will experiment with the MNIST dataset of handwritten digits 0, 1, ..., 9 available at http://yann.lecun.com/exdb/mnist/. The training set has 60000 28 x 28 grayscale images of handwritten digits (10 classes) and a testing set has 10000 images.

The data files are in binary format. The code `readMNIST.m` written by Siddharth Hegde and slightly modified by me reads these binary files and strips 4-pixel paddings from the images. The code `saveMNIST2mat.m` saves the resulting data to a mat file `mnist.mat`. The file `mnist.mat` and all codes I am mentioning are packaged to `Project1_aux.zip`.

1.1 Classification problem

The task is to select all images with digits 1 and all images with digits 7 from the training set, find a dividing surface that separates them, and test this dividing surface on the 1's and 7's from the test set. A sample of 1's and 7's from the training set is shown in Fig. 1.

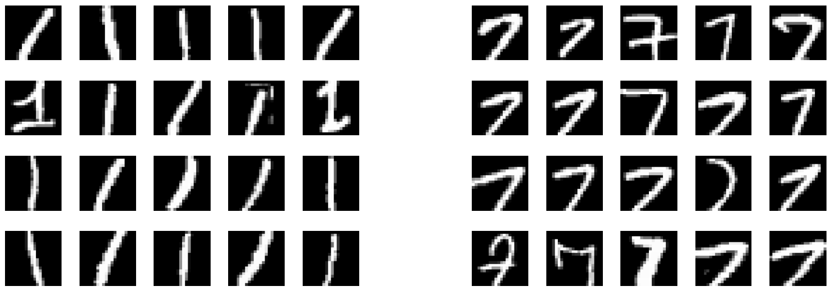


Figure 1: Samples of 20-by-20 images of 1's (left) and 7's (right) from MNIST.

1.2 Posing optimization problems

Each image is a point in $\mathbb{R}^{400}$ (the images with stripped paddings are 20-by-20). It is convenient to reduce dimensionality of data by using SVD and mapping the set to $\mathbb{R}^d$ where $d \ll 400$, e.g. $d = 3$, $d = 10$, $d = 20$ – see `mnist_2categories_hyperplane.m`. We label all images with 1 by 1 and all images with 7 by -1. The training data set `Xtrain` (or `X` for brevity) is `Ntrain`-by-`d` matrix. The vector of labels `y` is `Ntrain`-by-`1`.

My experiments on this set with the active set method have shown that it is much less efficient than the unconstrained optimization of an appropriately chosen loss function. We pose three kinds of unconstrained optimization problems.

1.2.1 A smooth loss function for the optimal hyperplane with Tikhonov regularization

In the simplest setting, we aim at finding a dividing hyperplane $w^\top x + b = 0$ with that $w^\top x_j + b > 0$ for all (almost all) x_j corresponding to 1 (labelled with $y_j = 1$) and $w^\top x_j + b < 0$ for all (almost all) x_j corresponding to 7 (labelled with $y_j = -1$). Hence, x_j is classified correctly if

$$\mathsf{sign}(y_j(w^\top x_j + b)) = 1.$$

Instead of the discontinuous sign function, we use a smooth sigmoid-type function (we call it *residual*)

$$r_j \equiv r(x_j; \{w, b\}) := \log\left(1 + e^{-y_j(w^\top x_j + b)}\right) \tag{1}$$

that is close to zero if $y_j(w^\top x_j + b) > 0$ and grows linearly in the negative range of the aggregate $y_j(w^\top x_j + b)$. For brevity, we will denote the $d+1$-dimensional vector of parameters $\{w, b\}$ by $\mathbf{w}$. We form the loss function by averaging up the residuals and adding a Tikhonov regularization term:

$$f(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n}\log\left(1+e^{-y_j(w^\top x_j+b)}\right)+\frac{\lambda}{2}\|\mathbf{w}\|^2. \tag{2}$$

Here n is the number of data points and λ is a parameter for the Tikhonov regularization. This loss function and its derivatives are encoded in functions `fun0` and `gfun0` at the bottom of the code `mnist_2categories_hyperplane.m`. The optimization problem in `mnist_2categories_hyperplane.m` is solved using the stochastic inexact Newton method. The approximations for Newton's directions are found by the conjugate gradient method. A line search algorithm is used along each proposed direction. If you set `nPCA = 3` in line 5, i.e., $d = 3$, then the dividing hyperplane is visualized.

1.2.2 A smooth loss function for the optimal quadratic hypersurface with Tikhonov regularization

As you will see, a quadratic dividing hypersurface may lead to much fewer misclassified digits. We are seeking a quadratic hypersurface of the form:

$$x^\top W x + v^\top x + b.$$

Hence, the quadratic test function is

$$q(x_j;\mathbf{w}) := y_j\left(x^\top W x + v^\top x + b\right). \tag{3}$$

The loss function is defined in a similar manner:

$$f(\mathbf{w}) = \frac{1}{n}\sum_{j=1}^{n}\log\left(1+e^{-q(x_j;\mathbf{w})}\right)+\frac{\lambda}{2}\|\mathbf{w}\|^2. \tag{4}$$

Here $\mathbf{w}$ denotes the $d^2 + d + 1$-dimensional vector of coefficients of $\{W, v, b\}$. This loss function and its gradient are available in the file `qloss.m`.

1.3 The research tasks

Set the number of PCAs $d = 20$. With this number of PCAs, you should be able to achieve good accuracy (the ratio of correctly classified test data to the total number of test data is about 99%). Implement stochastic optimizers that we have studied:

1. stochastic gradient descent (experiment with various batch sizes and stepsize decreasing strategies.;
2. stochastic Nesterov (experiment with various batch sizes);
3. stochastic Adam (experiment with various batch sizes);

4. stochastic L-BFGS (experiment with choosing stepsize, batch sizes for the gradient, and the pairs $(\mathbf{s}, \mathbf{y})$, and with the frequency of updating the pairs).

Use these optimizers to minimize the loss function (4) and find the optimal dividing quadratic hypersurface. For each solver, experiment with the appropriate settings. Compare the performance of these optimizers to each other. Run the optimizers for the same number of epochs (if you have n data points and your batch size is m then `round`(n/m) timesteps is one epoch). Which optimizer do you find the most efficient? Include a detailed discussion on the performance of these solvers in various settings in your report. Supplement your report with plots of the estimates for the loss function and the norm of its gradient. Include tables, if appropriate.

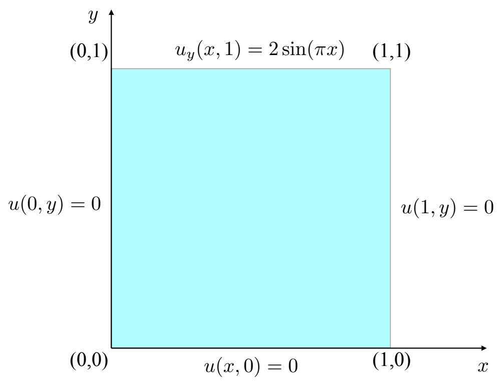
Finally, pick the most efficient solver and experiment with various numbers of PCAs using the most efficient solver. Summarize your findings in your report.

2 Solving PDE using neural networks

Set up and solve Problem 6 from Lagaris, Likas, and Fotiadis (1998), a boundary-value problem for the Poisson equation

$$u_{xx} + u_{yy} = f(x,y) \quad \text{where} \quad f(x,y) = (2 - \pi^2 y^2)\sin(\pi x), \quad (x,y) \in [0,1]^2, \tag{5}$$

with mixed boundary conditions:



The exact solution to this problem is given by

$$u_{ex}(x,y) = y^2 \sin(\pi x). \tag{6}$$

To solve this problem, you can mimic my solution to Problem 5 from Lagaris et al. (see `Lagaris.zip` for Matlab or `Largaris5.ipynb` for Python)

Use a neural network with one hidden layer and $n = 10$ neurons of the form:

$$\mathcal{N}(x,y;w) = \sum_{j=0}^{n-1} w_{3j}\sigma\left(w_{0j}x + w_{1j}y + w_{2j}\right), \tag{7}$$

where $\sigma(z)$ is a nonlinear activation function acting entrywise. Try $\sigma(z) = (1 + \exp(-z))^{-1}$ *sigmoid* and $\sigma(z) = \tanh(z)$ *hyperbolic tangent.* The total number of parameters to optimize is $4n$. It is easy to check that

$$\mathcal{N}_x(x,y;w) = \sum_{j=0}^{n-1} w_{3j} w_{0j} \sigma'\left(w_{0j}x + w_{1j}y + w_{2j}\right), \quad \mathcal{N}_y(x,y;w) = \sum_{j=0}^{n-1} w_{3j} w_{1j} \sigma'\left(w_{0j}x + w_{1j}y + w_{2j}\right),$$

$$\mathcal{N}_{xx}(x,y;w) = \sum_{j=0}^{n-1} w_{3j} w_{0j}^2 \sigma''\left(w_{0j}x + w_{1j}y + w_{2j}\right), \quad \mathcal{N}_{yy(}(x,y;w) = \sum_{j=0}^{n-1} w_{3j} w_{1j}^2 \sigma''\left(w_{0j}x + w_{1j}y + w_{2j}\right).$$

The derivatives of $\mathcal{N}$, $\mathcal{N}_x$, $\mathcal{N}_y$, $\mathcal{N}_{xx}$, and $\mathcal{N}_{yy}$ with respect to the components of w are easily found from these formulas and are available in my codes `res.m` and `Largaris5.ipynb`.

The proposed form of the solution $U(x,y;w)$ is given in Lagaris et al. in Eqs. (24)–(25). Take $N_{tr} = 49$ training points forming a uniform 7×7 `meshgrid` in $[0,1]^2$. No need to put training points on the boundary. Set up the least squares loss function

$$\mathcal{L}(w) = \frac{1}{2}\sum_{i=0}^{N_{tr}-1} \left| U_{xx}(x_i,y_i;w) + U_{yy}(x_i,y_i;w) - f(x_i,y_i)\right|^2. \tag{8}$$

Implement and apply to this problem the following *deterministic* methods

- Levenberg-Marquardt (you can use my implementation);
- Gauss-Newton;
- Gradient descent;
- Nesterov;
- Adam.

Plot the computed solution and the numerical error using level sets or a heatmap. Plot the loss function versus the iteration number in the same figure for all methods. Plot the norm of the gradient versus the iteration number in the same figure for all methods. In both cases, use a log scale for the y-axis. Comment on the performance of these methods in your report.