Gradient methods

In this exercise you will program and learn different learning algorithms. As an example we consider the logistic regression problem, which is not too simple (it is non-linear in the parameters) and not too hard (it has a unique solution).

Inputs patterns $x^n = (x_1^n, \ldots, x_d^n) \in \mathbb{R}^d$ output patterns $t^n = \{0, 1\}$ with $n = 1, \ldots, N$ and N the total number of patterns.

The model is given as (see also Bishop section 4.3) follows. The probability of $t = 1$ given x is given by

$$y = p(t = 1|x) = \sigma\left(\sum_{i=0}^{d} w_i x_i\right)$$

where we have defined $x_0 = 1$ and σ is the sigma function $\sigma(x) = (1 + e^{-x)})^{-1}$ and $p(t = 0|x) = 1 - y$. Define $y^n = p(t = 1|x^n)$. The cost to be minimizes is minus the log likelihood of the data

$$E(w) = -\frac{1}{N}\sum_{n=1}^{N}[t^n \log y^n + (1 - t^n)\log(1 - y^n)]$$

The gradient and Hessian are

$$\begin{aligned} \nabla_i E(w) &= \frac{\partial E}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N}(y^n - t^n)x_i^n \qquad i = 0, \ldots, d \\ H_{ij}(w) &= \frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{1}{N}\sum_{n} x_i^n y^n (1 - y^n) x_j^n \qquad i, j = 0, \ldots, d \end{aligned}$$

The $1/N$ term is just to make the quantities of order 1 for numerical stability.

Test the different learning methods on the MNIST data `http://yann.lecun.com/exdb/mnist/`. The data in matlab format are here: `http://www.snn.ru.nl/~bertk/comp_neurosci/mnistAll.mat`. These are images of digits 0 to 9. Select the images 3 and 7 for your two class classification problem. Scale the inputs to be in the range $[0, 1]$.

Gradient descent

The simplest idea to minimize E is by gradient descent. Start with random initial w and update according to

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Implement the learning rule in your computer program. Apply the method to the logistic regression problem. Produce plots of how E decreases with iterations, both on the training set and on the test set. Note, that the training error should always decrease, but the test error not. Test the effect of different values of η.

Experiment also with a method called early stopping. Divide the training set in 80% training set and 20 % validation set. Train on this reduced training set and monitor the

error on both the training set and the validation set. Stop the training when the error on the validation set increases. Report the error on the test set. [1]

Here are some results that I got after 10.000 iterations:

$$E_{\text{train}} = 0.0184 \qquad E_{\text{test}} = 0.0511$$

The fraction of misclassified patterns is 1.14% and 2.05% on train and test set respectively. CPU time Matlab implementation was 45 s.

Momentum

Add momentum to your gradient rule with strength α (see slides Machine Learning). Apply the method to the logistic regression problem. Produce plots of how E decreases with iterations, both on the training set and on the test set. Test the effect of different values of α, η.

Here are some results that I got after 10.000 iterations:

$$E_{\text{train}} = 0.0114 \qquad E_{\text{test}} = 0.0636$$

The fraction of misclassified patterns is 0.78% and 2.07% on train and test set respectively. CPU time Matlab implementation was 45 s.

Weight decay

Thus, we see that improving the optimization method using momentum decreases the training error, but increases the test error. This indicates that the logistic regression method is overfitting. We therefore add a weight decay term to E:

$$E(w) = -\frac{1}{N}\sum_{n=1}^{N}[t^n \log y^n + (1-t^n)\log(1-y^n)] + \frac{\lambda}{2d}\sum_{i=0}^{d} w_i^2$$

The gradient and Hessian are

$$\begin{aligned} \nabla_i E(w) &= \frac{\partial E}{\partial w_i} = \frac{1}{N}\sum_{n=1}^{N}(y^n - t^n)x_i^n + \frac{\lambda}{d}w_i \qquad i = 0,\ldots,d \\ H_{ij}(w) &= \frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{1}{N}\sum_{n} x_i^n y^n (1-y^n) x_j^n + \frac{\lambda}{d}\delta_{ij} \qquad i,j = 0,\ldots,d \end{aligned}$$

Using $\lambda = 0.1$ and the momentum method, report E (with out the regularization term) on both training and test set after convergence, number of iterations and total CPU time. Send me the code so that I can check your reported performances of the various methods.

I find after 5340 iteration on the training set $E = 0.0218$ and classification error 1.41% and on the test set $E = 0.0467$ and classification error 2.17%. Elapsed time is 26 seconds.

We now use this problem ($\lambda = 0.1$) and the results using gradient descend with momentum to compare with other methods.

[1] Note, that you are not allowed to use the test set to decide the early stopping, because then your learning algorithm uses test set information ie. in a sense optimising on the test data, which is cheating.

Newton method

Implement the Newton method

$$\Delta w = -H^{-1}(w)\nabla E$$

Apply the method to the logistic regression problem with weight decay. Produce plots of how E decreases with iterations, both on the training set and on the test set.

I find after 10 iteration on the training set $E = 0.0218$ and classification error 1.41% and on the test set $E = 0.0467$ and classification error 2.17%. Elapsed time is 31 seconds. So this method requires far less iterations, but each iteration is much more expensive, so that the net result is not worth it.

Line search

Implement the gradient method with line search. In each iteration compute the gradient at the current value of w: $d = -\nabla E(w)$. Then find numerically the value of $\gamma > 0$ such that $E(w + \gamma d)$ is minimized. This is a standard one dimensional optimization problem. For this, you can either write your own routine (for instance `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/BMVA96Tut/node16.html`) or use a standard package. The main issue is to define the initial interval within which the minimum is searched. Apply the method to the logistic regression problem. Produce plots of how E decreases with iterations, both on the training set and on the test set.

This method does not really converge well, but it is adapted with minor work to the conjugate gradient method discussed next. for the line search method I find after 224 iteration on the training set $E = 0.0239$ and classification error 1.50% and on the test set $E = 0.0465$ and classification error 2.19%. Elapsed time is 33 seconds.

Conjugate gradient descent

The conjugate gradient descent method uses the line search as a subroutine. In each step the search direction is $d = -\nabla E(w) + \beta d_{\text{old}}$. β is given by the Polak-Ribiere rule (see slides Machine Learning). Implement the conjugate gradient method. Apply the method to the logistic regression problem. Produce plots of how E decreases with iterations, both on the training set and on the test set.

I find after 106 iteration on the training set $E = 0.0218$ and classification error 1.41% and on the test set $E = 0.0467$ and classification error 2.17%. Elapsed time is 12 seconds. So this method converges fasted of all methods that we discussed.

Stochastic gradient descent

In the above, socalled batch methods, the computation of the gradient requires time linear in the size of the data set. When the data set is large, this can be a significant cost. The stochastic gradient descent method only uses a subset of the total data set (sometimes called mini batch). Implement the stochastic gradient descent method. Apply the method to the logistic regression problem. Consider constant learning rate η and mini batch sizes. Produce plots of how E decreases with iterations, both on the training set and on the test set for different values of η.

I find with mini batch size of $0.01N$ after 5000 iteration on the training set $E = 0.0243$ and classification error 1.36% and on the test set $E = 0.0499$ and classification error 2.02%. Elapsed time is 5 seconds.

Assignment

Implement all the above methods using your favorite computer language. Provide results that are competitive with the results that I stated and show the graphs. Hand in the code so that I can reproduce your results. Discuss how the results for this problem (logisitic regression on the MNIST data) are expected to change for other data sets (more samples N or more variables n) and methods (more variables w such as deep neural networks).