Inleiding Machine Learning

Exercisesperceptron

1. Beschouw een kostenlandschap gegeven door $E = a_1x^2 + a_2y^2$ met $a_1 = 1, a_2 = 20$.
   - Implementeer de gradient descent leerregel en laat numeriek convergentie zien. Varieer de leerparameter η. Bereken de leerparameter η zodanig dat de convergentie in zowel x als y richting even snel is.
   - Voeg momentum toe aan de leerregel en check dat convergentie sneller gaat.
   - Geef een algoritme voor de Newton methode en argumenteer je resultaat.
2. - Write a computer program that implements the perceptron learning rule. Take as data p random input vectors of dimension n with binary components. Take as outputs random assignments ± 1. Take $n = 50$ and test empirically that when $p < 2n$ the rule converges almost always and for $p > 2n$ the rule converges almost never. Here is the template of a Matlab program.
   - Reconstruct the curve $C(p, n)$ (seen perceptron handout) as a function of p for $n = 50$ in the following way. For each p construct a number of learning problems randomly and compute the fraction of these problems for which the perceptron learning rule converges. Plot this fraction versus p.
   - apply the perceptron to classify the 3's and the 7's from the MNIST data . Get the data from `http://www.cs.nyu.edu/~roweis/data/mnist_all.mat`.

Here is some Python code to get you started.

```
import numpy as np
import scipy.io  # use scipy.io module to read in .mat files

DATA = 'mnist'  # set to either 'mnist' or 'random'


if DATA == 'random':
    p = 50  # number of patterns
    n = 50  # number of dimensions
    xi = np.random.randint(2, size=(p, n + 1))  # generate random patterns (0s and 1s)
    xi[:, -1] = -1  # set label as n+1-th dimension
    z = 2 * np.random.randint(2, size=(p, 1)) - 1  # randomly multiply by 1 or -1 (to create two
data classes/labels)
    x = np.multiply(xi, z)
elif DATA == 'mnist':
    data_file = scipy.io.loadmat('mnist_all.mat')  # load mnist_all.mat (if in same folder.!)
    X3 = np.transpose(data_file['train3'])  # fetch and transpose training 3's
    X7 = np.transpose(data_file['train7'])
    n = int(np.size(X3, 0))  # number of pixels
    x3 = np.float32(X3) / -256.0  # go to float between 0 and 1, multiply 3s by -1
    x7 = np.float32(X7) / 256.0
    x3 = np.vstack([x3, np.squeeze(-1 * np.ones((np.size(x3, 1), 1)))])  # add label as n+1-th
```

```
dimension
    x7 = np.vstack([x7, np.squeeze(np.ones((np.size(x7, 1), 1)))])
    x = np.transpose(np.hstack([x3, x7]))  # create variable x with all training data
    p = int(np.size(x, 0))  # number of patterns

w = np.random.randn(1, n + 1)
eta = 1.0
maxiter = 1000
```

Here is some Matlab code to get you started. Set DATA='random' to generate the random perceptron data. Set DATA='mnist' to load the MNist data.

```
%% Matlab loading data template, inl. ML


clear all
DATA = 'mnist';% classes 3 and 7
% DATA = 'random';

switch DATA
    case 'random',
        p = 50; % number of patterns
        n = 50; % number of dimensions
        xi = randi(2, p, n+1) - 1; % inputs xi(:,1:n)=0,1 xi(:,n+1)=-1;
        xi(:, n+1)= - 1;
        z = 2 * randi(2, p, 1) - 3; % output z(:,1)=-1,1
        x = xi .* (z * ones(1, n + 1));

    case 'mnist',
        load mnist_all.mat
        X3 = train3'; % read in all train images of 3s and transpose
        X7 = train7';
        n = size(X3, 1); % the number of pixels
        x3 = double(X3) / -256.0; % go to floats between 0 and 1, input data of 3's multiplied
by class label -1
        x7 = double(X7) / 256.0;
        x3(n+1,:) = -1; % set class label as (n + 1)-th dimension/row
        x7(n+1,:) = 1;
        x = [x3, x7]';  % concatenate the two data sets in one variable
        p = size(x, 1);

end;

w = randn(1, n + 1); % weights w(1:n) threshold w(n+1)
eta = 1;
maxiter = 1000;

%% write your perceptron learning algorithm
```