Final Report of Term Project—ANN for Handwritten Digits Recognition

Weiran Wang*

May 5, 2009

Abstract

In this paper we present an Artificial Neural Network to tackle the recognition of human handwritten digits. The ANN proposed here is experimented on the well-known MNIST data set. Without any preprocessing of the data set, our ANN achieves quite low classification error. Combined with clustering techniques, we can build artificial intelligence system which can automatically segment individual digit from images and find its corresponding label.

1 Introduction

Automatic recognition of human handwritten digits was a mysterious job to me when I was an undergraduate student. Different people have very different writing style, even digits of a same person written in different time are not identical. How does artificial intelligence deal with the infinite possibility of different shapes of digits, given only an image? Since now I have taken the Machine Learning course and acquired knowledge in this field, I am able to tackle this problem with my own matlab program.

*wwang5@ucmerced.edu
2 Methods

The program I implement will mainly focus on identifying 0-9 from segmented pictures of handwritten digits. The input of my program is a gray level image, the intensity level of which varies from 0 to 255. For simplicity, input images are pre-treated to be of certain fixed size, and each input image should contain only one unknown digit in the middle. These requirements are not too harsh because they can be achieved using simple image processing or computer vision techniques. In addition, such pre-treated image data set are easy to obtain. In my implementation, the popular MNIST data set ([1]) is a good choice. Each image in MNIST is already normalized to 28x28 in the above sense and the data set itself is publicly available. The MNIST data set is really a huge one: it contains 60000 training samples and 10000 test samples. And it has become a standard data set for testing various algorithms.

The output of my program will be the corresponding 0-9 digit contained in input image. The method I use is Artificial Neural Network (ANN). Unlike lazy learning method such as Nearest Neighbor Classifier that stores the whole training set and classify new input case by case, ANN will implicitly learn the corresponding rule between image of handwritten digits and the actual 0-9 identities. To achieve the effect of dimensionality reduction, I make use of multilayer network. The input layer contains the same number of units as the number of pixels in input image. In our case it is 28x28=784. Then the hidden layer containing 500 units with sigmoid activation is employed to find a compact representation of input images. Hopefully, this compact representation is easier for the final classification purpose. In the end, the output unit contains 10 units in accordance with
10 different classes. Preferably, I want the output units provide the conditional probability (thus the output of each unit is between 0 and 1, and the outputs of all 10 units will sum to 1) of each class to which each input belongs, and the unit that has the maximum output will determine the class label. As a result, softmax activation is the desirable choice for output units. Figure 1 shows the architecture of the ANN I am using here.

The crucial part of this project is training the neural network. Since my goal is typically a classification problem, so the desirable objective function will be the multiple class cross entropy[2] between the network output and the target class labels. There is a somewhat subtle point in this program. For target output, I use the “1 out of n” scheme, but slightly change the values. The unit corresponding to the right class label of each input has value 0.91, while other units have the same value 0.01. I do not set them to be 1s and 0s because extreme value are hard to be achieved by activation functions.

The usual training algorithm introduced by [4] is error back propagation, in which user need to specify at least learning rates
apart from computing gradients with respect to weights. However, for the numerical optimization algorithm of my neural network, I choose Conjugate Gradient method because it not only allows me to get rid of choosing learning rate but also has good convergence property. Meanwhile, there are available matlab function minimize.m from internet[3]. With the help of this function, the problem reduces to computing the gradient of the objective function with respect to weights. And this is a simple task using δ rule[4]. For programing such a large neural network with matlab, I learned a lot from a good example given by Geoffery Hinton[5].

During each training epoch, the 60000 training samples are evenly divided into 60 batches. Weights are updated after processing each batch, resulting in 60 weights updates in every training epoch. In my implementation, I set a fixed number of training epochs–2000. Optimal weights are chosen based on best classification performance on training set. At the end of training process, the optimal weights for the neural network will be kept and used to generate class labels for new inputs (test set).

3 Results

Given the above configurations, I then run the matlab code. It takes nearly 4 days for my program to finish 2000 training epochs. Figure 2 shows the error rate of classification using the neural network, on both the training set and test set.

We can see that, the error quickly decreases on training set, and becomes almost 0 since 200 epochs. The error on test set first decreases with that of training set, then fluctuates a little. The smallest error rate 1.88% on test set is achieved at epoch
Figure 2: An illustration of the architecture of my ANN. The layers from bottom to top are input layer, hidden layer, and output layer, respectively.

4 Discussion

The MNIST data set is a popular data set, on which various classification algorithms has been tested. The state of art on this data set is large Convolutional Neural Network ([1]) with unsupervised pretraining. It achieved an error rate of 0.39% on test set. However, to make a fair comparison, it’s more beneficial to compare performances of different algorithms without any preprocessing on the data set. Table 1 lists the error rates of several algorithm, applied on the original data set.

Note that, the performance of Nearest Neighbor Classifier will be quite good given enough training samples. In our case, it even beat neural network with 300 hidden units. Thus the error rate
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Classifier</td>
<td>12</td>
</tr>
<tr>
<td>Neural Network, 300 hidden units, mean squared error</td>
<td>4.7</td>
</tr>
<tr>
<td>Nearest Neighbor Classifier</td>
<td>3.09</td>
</tr>
<tr>
<td>Support Vector Machine (Gaussian Kernel)</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Table 1: Error rates (in percentage) of several algorithms on test set.

1.88% I achieved is reasonably good: it beat the nearest neighbor classifier, and shows large improvements to smaller hidden layer. And it certainly outperforms Nearest Neighbor Classifier by a large margin on testing time: just putting new test sample in input layer and feed forward is going to be much faster than finding nearest neighbor in such a huge data set. I believe that with even more hidden units, it’s possible for ANN to achieve smaller error rate than Support Vector Machine.

In the rest of this section, I want to demonstrate the real application of the above ANN. Let’s first consider a simpler situation: suppose a big picture of a sequence of digits is given, and it contains more than one digits in it. Digits in that image may have different sizes and orientations. How to automatically locate and segment digits? Figure 3 shows a typical example of the situations I am considering.

Many efficient algorithms deal with this problem from a clus-
Figure 4: The segmentation result of Figure 3. Five different clusters are identified with different colors.

tering point of view. The idea is simple, though. Pixels that are close to each other (in the sense of location, intensity, texture...) should be segmented into the same cluster. The most popular clustering methods during years maybe the spectral clustering methods[7], which are based on spectral graph theory. I am going to tackle this problem with a similar in spirits but still different clustering method – Gaussian Blurring Mean-Shift.

In my case, input image are grayscale, This desirable characteristic allows me to cluster only the coordinates of the non-zero pixels while ignoring the intensities of them. In this algorithm, I only need to give one parameter – the width of the gaussian kernel, measured by number of pixels. During a set of experiments, I find that choosing kernel width 5 gives very stable and accurate result. Usually it takes 10 mean-shift iterations to converge.

Figure 4 shows the segmentation result for Figure 3.

After segmentation, I know the pixel locations of each digits. Thus it is straight forward to normalize each digits into image of size 28x28. Figure 5 gives the normalized images. These images can easily be sent to the ANN with optimal weights and predict their labels. In the case shown in the figure, my ANN classifies all the 5 digit correctly.
For more complex images containing not only digits but also non-digit symbols, it is still possible to segment symbols using GBMS. Then we can set threshold of the distance between test image and training images to distinguish whether the test image is a digit or not. After that, digit images can be sent to ANN and be classified.

5 Conclusion

In conclusion, I implement a large multilayer artificial neural network for human handwritten digits. I train the ANN with cross entropy using error back propagation to obtain optimal weights values. I also find useful applications for the ANN I generate. By doing this project, I have practiced using what I have learned in the Machine Learning course. My program probably does not beat the state of the art in handwritten digit recognition. However, I have for the first time observed the practical problems of using the powerful Artificial Neural Networks, for example, designing the architecture of an ANN, choosing appropriate activation functions for each layer, error back propagation, convergence issues, stopping criteria, generalization ability . . . This experience will definitely be helpful for my future research.
6 Acknowledgments

I want to thank Dr. David Noelle for his course Machine Learning. I really learn a lot of new things from his lectures. I am also grateful to Chao Qin and Jianwu Zeng for their useful discussions with details of training ANN.

References


Appendix

The codes for Neural Network are attached here.

---

```matlab
% batchsize: 100.

clear all;
load mnist_all;

% START OF MAKING BATCHES FOR TRAINING SET

digitdata = [ ];
digitid = [ ];
digitdata = [digitdata;train0];
digitid = [digitid;0*ones(size(train0,1),1)];
digitdata = [digitdata;train1];
digitid = [digitid;1*ones(size(train1,1),1)];
digitdata = [digitdata;train2];
digitid = [digitid;2*ones(size(train2,1),1)];
digitdata = [digitdata;train3];
digitid = [digitid;3*ones(size(train3,1),1)];
digitdata = [digitdata;train4];
digitid = [digitid;4*ones(size(train4,1),1)];
digitdata = [digitdata;train5];
digitid = [digitid;5*ones(size(train5,1),1)];
digitdata = [digitdata;train6];
digitid = [digitid;6*ones(size(train6,1),1)];
digitdata = [digitdata;train7];
digitid = [digitid;7*ones(size(train7,1),1)];
digitdata = [digitdata;train8];
digitid = [digitid;8*ones(size(train8,1),1)];
digitdata = [digitdata;train9];
digitid = [digitid;9*ones(size(train9,1),1)];
digitdata = double(digitdata)/255;
totnum=size(digitdata,1);
fprintf(1, 'Size of the training dataset: %d
', totnum);
fprintf(1, 'Dimension of original input: %d
', size(digitdata,2));
rand('state',0); %so we know the permutation of the training data
numdims = size(digitdata,2);
batchsize = 100;
numbatches=totnum/batchsize;
trainbatchdata = zeros(batchsize, numdims, numbatches);
trainbatchid = zeros(batchsize, 1, numbatches);
for i=1: numbatches
    trainbatchdata(:,:,i) = digitdata(randperm(totnum, i*batchsize-i*batchsize+1));
    trainbatchid(:,:,i) = digitid(randperm(totnum, i*batchsize-i*batchsize+1));
end
clear digitdata;
clear digitid;
clear randnumperm;
%END OF MAKING BATCHES FOR TRAINING SET

% END OF MAKING BATCHES FOR TRAINING SET

% START OF MAKING BATCHES FOR TESTING SET

digitdata = [ ];
digitid = [ ];
digitdata = [digitdata; test0];
digitid = [digitid;0*ones(size(test0,1),1)];
digitdata = [digitdata; test1];
digitid = [digitid;1*ones(size(test1,1),1)];
digitdata = [digitdata; test2];
digitid = [digitid;2*ones(size(test2,1),1)];
digitdata = [digitdata; test3];
digitid = [digitid;3*ones(size(test3,1),1)];
digitdata = [digitdata; test4];
digitid = [digitid;4*ones(size(test4,1),1)];
digitdata = [digitdata; test5];
digitid = [digitid;5*ones(size(test5,1),1)];
digitdata = [digitdata; test6];
digitid = [digitid;6*ones(size(test6,1),1)];
```

---

10
digitdata = [digitdata; test7];
digitid = [digitid; 7*ones(size(test7,1),1)];
digitdata = [digitdata; test8];
digitid = [digitid; 8*ones(size(test8,1),1)];
digitdata = [digitdata; test9];
digitid = [digitid; 9*ones(size(test9,1),1)];
digitdata = double(digitdata)/255;
totnum=size(digitdata,1);
fprintf(1, ‘Size of the test dataset: %d
’, totnum);
rand(‘state’,0); % so we know the permutation of the training data
randomorder2 = randperm(totnum);

numdims = size(digitdata,2);
umbatches = totnum/batchsize;
tesbatchdata = zeros(batchsize, numdims, numbatches);
testbatchid = zeros(batchsize, 1, numbatches);
for i=1: numbatches
testbatchdata(:,:,i) = digitdata(randomorder2(1+(i-1)*batchsize:i*batchsize),:);
testbatchid(:,:,i) = digitid(randomorder2(1+(i-1)*batchsize:i*batchsize),:);
end

clear digitdata;
clear digitid;
clear -regexp ‘test’;

% This program tunes an ANN which has 1 hidden unit with erro back propagation.
% The architecture is: input layer(326 units) --> hidden layer 1 (100 units) --> output layer (10 units, corresponding to 10 classes of digits).
% Units in hidden layer have sigmoid activation.
% Output layer use softmax activation (Desirable objective function for multiple classes).
% Weights of the autoencoder are going to be saved in mnist_weights.mat.
% Error rates of classification are saved in mnist_errors.mat.
% You can also set maxepoch, default value is 2000.
% This is modified from code provided by Ruslan Salakhudinov and Geoff Hinton.
% Permission is granted for anyone to copy, use, modify, or distribute this
% program and accompanying programs and documents for any purpose, provided
% this copyright notice is retained and prominently displayed, along with
% a note saying that the original programs are available from our web page.

maxepoch = 2000;
fprintf(1,’
Fine-tuning MLP by minimizing cross entropy error. \n’);

hd = 500; % Number of units in hidden layer.
od = 10; % Number of different classes.

% PREINITIALIZE WEIGHTS OF THE MLP

w1 = randn(numdims+1,hd)*0.5;
w2 = randn(hd+1,od)*0.5;
load mnist_weights_init
save mnist_weights_init w1 w2

% We need to store the weights when they give best performance on the test set. Thus need the variable to record the smallest error rate on test set thus far.
best_performance = 1.0;
for epoch = 1:maxepoch
...
% COMPUTE TRAINING CLASSIFICATION ERROR %

err = 0;

[numcases numdims numbatches] = size(trainbatchdata);
N = numcases;
for batch = 1: numbatches
  data = trainbatchdata(:,:,batch);
  target = trainbatchid(:,:,batch);
  data = [data ones(N,1)];
  w1probs = 1./(1 + exp(-data*w1));
  w1probs = [w1probs ones(N,1)];
  output = w1probs*w2;
  output = exp(output);
  s = sum(output,2);
  output = output./repmat(s,1,od); % Softmax activation.
  [m, I] = max(output,
  I = I-1; % Convert index to id.
  err = err + sum(I~=target)/N;
end
train_err = [train_err, err/numbatches];

% END OF COMPUTING TRAINING CLASSIFICATION ERROR %

% COMPUTE TEST CLASSIFICATION ERROR %

err = 0;

[numcases numdims numbatches] = size(testbatchdata);
N = numcases;
for batch = 1: numbatches
  data = testbatchdata(:,:,batch);
  target = testbatchid(:,:,batch);
  data = [data ones(N,1)];
  w1probs = 1./(1 + exp(-data*w1));
  w1probs = [w1probs ones(N,1)];
  output = w1probs*w2;
  output = exp(output);
  s = sum(output,2);
  output = output./repmat(s,1,od); % Softmax activation.
  [m, I] = max(output,
  I = I-1; % Convert index to id.
  err = err + sum(I~=target)/N;
end
test_err = [test_err, err/numbatches];
if test_err(end) < best_performance
  best_performance = test_err(end);
  save mnist_best_weights w1 w2
end

% END OF COMPUTING TEST CLASSIFICATION ERROR %

% COMBINE 10 MINIBATCHES INTO 1 LARGER MINIBATCH %

max_iter = 3;
VV = [w1(:); w2(:)];
Dim = [numdims; hd; od];
[X, fX] = minimize(VV, 'CG_MNIST', max_iter, Dim, data, target);
w1 = reshape(X(1:(numdims+1)*hd), numdims+1, hd);
xxx = (numdims+1)*hd;
w2 = reshape(X(xxx+1:end), hd+1, od);

% END OF CONJUGATE GRADIENT WITH 3 LINESEARCHES %

% save mnist_weights w1 w2
% save mnist_errors train_err test_err
% figure(100);
% clf;
% plot(1:epoch, train_err, 'b.-');
CG\_MNIST.m:

function [f, df] = CG\_MNIST(VV, Dim, data, targetid)

% This program implement the Error Back Propagation and minimize the cross
% entropy with conjugate gradient method.
% This is modified from code provided by Ruslan Salakhutdinov and Geoff Hinton.
% Permission is granted for anyone to copy, use, modify, or distribute this
% program and accompanying programs and documents for any purpose, provided
% this copyright notice is retained and prominently displayed, along with
% a note saying that the original programs are available from our
% web page.

l1 = Dim(1);  % Input dimension.
l2 = Dim(2);  % Hidden dimension.
l3 = Dim(3);  % Output dimension.
N = size(data,1);

% Do deconvolution.
w1 = reshape(VV(1:(l1+1)*l2),l1+1,l2);
xxx = (l1+1)*l2;
w2 = reshape(VV(xxx+1:end),l2+1,l3);
data = [data ones(N,1)];
w1probs = 1./(1 + exp(-data*w1));
w1probs = [w1probs ones(N,1)];
output = w1probs*w2;
output = exp(output);
s = sum(output,2);
output = output./repmat(s,1,l3);% Softmax activation.
f = 0;
t = ones(size(output))*0.01;
for i = 1:N
    t(i,targetid(i)+1) = 0.91; % Target vector=[0.01,...,0.91,0.01,...0.01]
end
f = -sum(sum(t.*log(output./t))); % Cross entropy.

Ix2 = output-t;
dw2 = wprobe*Ix2;
Ix1 = (Ix2*w2').*w1probs.*(1-w1probs);
Ix1 = Ix1(:,1:end-1);
del = data.*Ix1;
df = [dw1(:); dw2(:)];