CS2310 Milestone #2 Report

Mengjie Mao

Title:

A digit recognition system prototyped by SIS Testbed

1. Input

In this project we use the MNIST dataset as the input for recognition. MNIST contains a training set of 60,000 handwritten digit examples, and a test set of 10,000 handwritten digit examples. Figure 1 shows five different styles extracted from MNIST dataset.

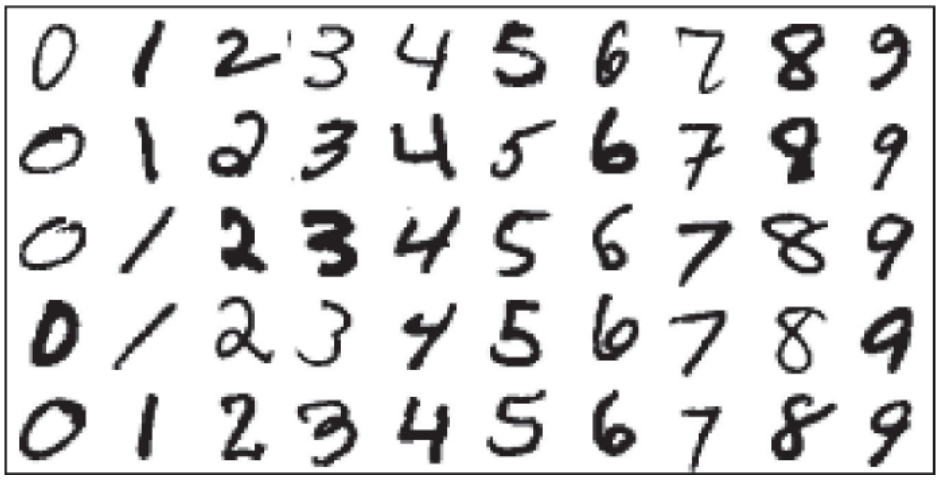

Figure 1 the black-white figure of digits from MNIST.

The package of MNIST has four files, including TRAINING SET LABEL FILE (train-labels-idx1-ubyte), TRAINING SET IMAGE FILE (train-images-idx3-ubyte), TEST SET LABEL FILE (t10k-labels-idx1-ubyte) and TEST SET IMAGE FILE (t10k-images-idx3-ubyte). Two label files share the same format:

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  60000            number of items
0008     unsigned byte   ??               label
0009     unsigned byte   ??               label
........
xxxx     unsigned byte   ??               label
```

The label is from 0 to 9.

Two image file also have the same format, as shown in following:

```
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

Pixels are organized row-wise. Pixel values are 0 to 255. 0 means background (white), 255 means foreground (black). Continuous 28x28 pixels in the image file consist of a digit.

From the label format and image format we can build a training dataset and a testing dataset for auto associative memory. The input for associative memory can be a vector consisted by 784 unsigned integers, each of which is between 0 and 255.

2. Algorithm

We use the The *Brain-state-in-a-box* (BSB) model, which is a simple, auto-associative, nonlinear, energy-minimizing neural network. A common application of the BSB model is *optical character recognition* (OCR) for printed text. The BSB model is a simple auto-associative neural network with two main operations – *training* and *recall*. The mathematical model for recall can be represented as:

$$\mathbf{x}(t+1) = S(\alpha \cdot \mathbf{A} \times \mathbf{x}(t) + \lambda \times \mathbf{x}(t))$$

where, x is an *N* dimensional real vector, and A is an *N*-by-*N* connection matrix. A $\times$ x(*t*) is a matrix-vector multiplication, which is the main function of the recall operation. α is a scalar constant feedback factor. λ is an inhibition decay constant. *S*(*y*) is the "squash" function defined as follows:

$$S(y) = \begin{cases} 1, & \text{if } y \geq 1 \\ y, & \text{if } -1 < y < 1 \\ -1, & \text{if } y \leq -1 \end{cases}$$

For a given input pattern x(0), the recall function computes above function iteratively until *convergence*, that is, when all the entries of x(*t*+1) are either "1" or "–1".

The BSB model has been implemented with C++, below two figure show the training source code recall source code:

```
bool TrainBSB(float *vec, float *wm)

{

 bool       converged;
 int        i, j, k;
 float      *wx;

 wx = (float *)malloc(BsbSize*sizeof(float));

 // Compute W*X
 for(i=0;i<BsbSize;){
   wx[i] = 0.0;
   for(j=0;j<BsbSize;){
     wx[i] += wm[i*BsbSize+j] * vec[j];
     ++j;
   }
   ++i;
 }

 converged = true;

 // Compute X- W*X
 for(i=0;i<BsbSize;){
   wx[i] = vec[i] - wx[i];
   if(fabsf(wx[i]) > 1.0E-4) converged = false;
   ++i;
 }

 // Update W = W + LR*(X- W*X) _outter_product_ X
 for(i=0;i<BsbSize;){
   for(j=0;j<BsbSize;){
     wm[i*BsbSize+j] = wm[i*BsbSize+j] + LEARN_RATE*wx[i]*vec[j];
     ++j;
   }
   ++i;
 }

 free(wx);
 return converged;
} « end TrainBSB »
```

Figure 2 the training function of BSB model

```
bool RecallBSB(float *vec, float *wm)
{
 int        i, j, k;
 bool       converged;
 float      *wx;
 wx = (float *)malloc(BsbSize*sizeof(float));

 // Compute W*X
 for(i=0;i<BsbSize;){
   wx[i] = 0.0;
   for(j=0;j<BsbSize;){
     wx[i] += wm[i*BsbSize+j] * vec[j];
     ++j;
   }
   ++i;
 }

 // Compute S(Alpha*W*X + Lamda*X)
 for(i=0;i<BsbSize;){
   wx[i] = ALPHA*wx[i] + LAMDA*vec[i];
   if(wx[i] <= -1.0)
     vec[i] = -1.0;
   else
     if(wx[i] >= 1.0)
       vec[i] = 1.0;
     else
       vec[i] = wx[i];
   ++i;
 }

 // Check convergence
 converged = true;

 for(i=0;i<TagOffset;++i){ // When converged, all tags must be 1.0
   if(vec[i] != 1.0) converged = false;
 }

 for(i=0;i<BsbSize;++i){ // When converged, all image entries must be eith -1 or 1
   if(fabsf(vec[i]) != 1.0) converged = false;
 }

 free(wx);
 return converged;
} « end RecallBSB »
```

Figure 3 the recall function of BSB model