Reconstruction of Digit 4 using Boltzmann Machine

Sulaimon Oyeleye

May 2, 2019

Contents

1 Data Description

Using the MNIST dataset consisting of handwritten digit images comprising of 60,000 examples for the training set and 10,000 examples for testing, I picked 20 random images of digit 4 by copying 20 rows representing digit 4 which is my focus on this project. All digit images have been size-normalized and centered in a fixed size image of 28 x 28 pixels. In the original dataset each pixel of the image is represented by a value between 0 and 255, where 0 is black, 255 is white and anything in between is a different shade of grey.

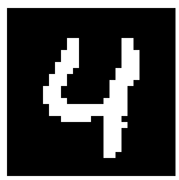
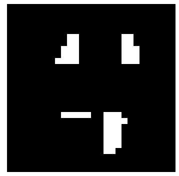


Image Information

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.

2 GBM Architecture

By taking an image, we feed our Boltamnn Machine with image having the middle part of it being taken off as seen in the image below and having the top and bottom fixed. Thus, for a 28 by 28 making 784 pixel. I could exclude the middle part for 281 to 504 pixels(i.e the middle 8 rows).



Next, I simulate the Boltzmann machine using a totally connected network. The neural networks with neurons that are connected not only to other neurons in other layers but also to neurons within the same layer. For every sweep, we visit all neurons and a total of 8 visits was made.
When neuron i is visited with current state x_i, I compute

$$V_i = \sum_j wijxj$$

, then we can calculate our probability for this visit at i by

$$p_i = \frac{1}{1 + exp(-V_i)}$$

and update the probability, $P(new\ \ x_i = 1) = p_i$.
The co-activities is computed for both the clamped, $C_{i,j}$, and unclamped, $\hat{C_{i,j}}$, cases as the sum average of $x_i x_j$ over the 8 visits.
Using a small $\varepsilon = 0.01$, weight were updated using

$$new\ w_{i,j} = w_{i,j} + \varepsilon(C_{i,j} - \hat{C_{i,j}})$$

Same is then done to image 2 and eventually the process is repeated for the all the images.

3 Results

Automatic learning was implemented using MATLAB(This can be found in the appendix). My default choose of learning options uses 5 epochs. Below is the presented images of the 20 reconstructed images.

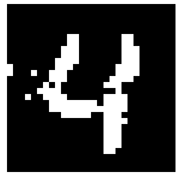


Figure 1: Image 1

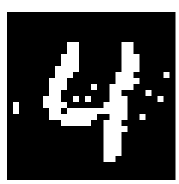


Figure 2: Image 2

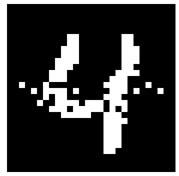


Figure 3: Image 3

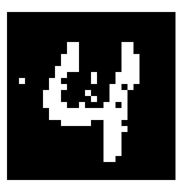


Figure 4: Image 4

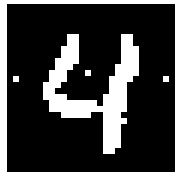


Figure 5: Image 5

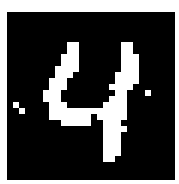


Figure 6: Image 6

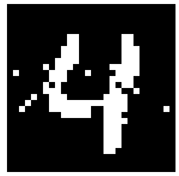


Figure 7: Image 7

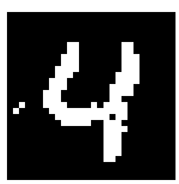


Figure 8: Image 8

Figure 9: Image 9

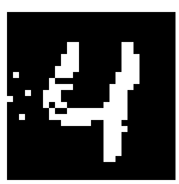


Figure 10: Image 10

Figure 11: Image 11

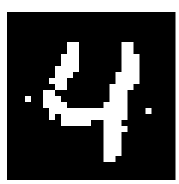


Figure 12: Image 12

Figure 13: Image 13

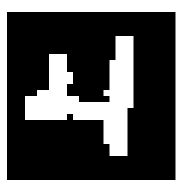


Figure 14: Image 14

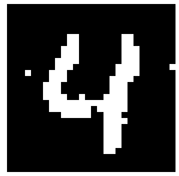


Figure 15: Image 15

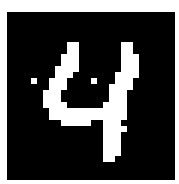


Figure 16: Image 16

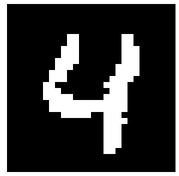


Figure 17: Image 17

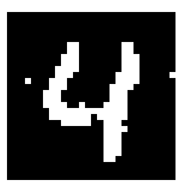


Figure 18: Image 18

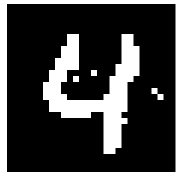


Figure 19: Image 19

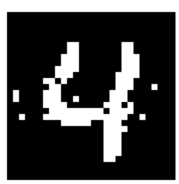


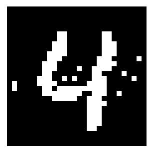
Figure 20: Image 20

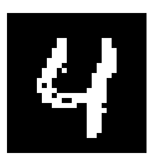
4 Impact of Various Leaning Options

Here, I consider how various parameters affect our automatic learning. Specifically, I analyze cases for varying the number of epochs, epsilon and the number of visits.

4.1 Number of Epoch

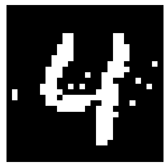
Here, I look at the effect of increasing the number of epoch by twice the default number I used in the code. Below is the output of the default(top image) and the increase epoch output(bottom image), which happens to be better.

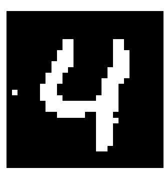




4.2 Epsilon

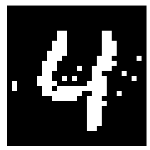
Epsilon used was 0.01, after changing it to 0.05. The processing time is still roughly same. Below is the output of the default(top image) and the updated output(bottom image), which happens to be much better.

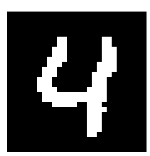




4.3 Number of Visit

Instead of 8 visits I decided to double that and try out 16 visits. As, expected the more the visit the better our result. Below is the output of the default(top image) and the updated output(bottom image) which is almost perfect.





5 Appendices

5.1 Appendix: Boltzmann Machine Source Code

```
[a,b]=size(ori);
N=b;
epsilon=0.01;
epoch=5;
update_size=7;
H=28*2;
out=28*8;
W=initweights(280*2,N,H);


for p=1:a
%%%%%%%%binarize input
for i=1:b
   if ori(p,i)<=0;
   img(i)=0;
   else
   img(i)=1;
   end
end

%%%%initialize configuration of neurons and weights%%%%%%
x=zeros(1,N+H);
counter=0;
for i=1:N
taboo=[281:504];
    if not(ismember(i,taboo))
    counter=counter+1;
    x(counter)=img(i);
    end
end
f_current=x;
c_current=x;
for i=1:28*8
c_current(i+counter+H)=img(280+i);
end

f_record=zeros(update_size,N+H);
c_record=zeros(update_size,N+H);

for l=1:epoch
count=0;
order=counter+randperm(H);
f_current=generate(f_current,counter+H+1,W);
r_image=[x(1:280), f_current(counter+H+1:N+H), x(281:counter)];
imshow(reshape(r_image,28,28)')
    for i=1:length(order)
    %%%%%%simulate for free case%%%%%%%%%
    f_ener=energy(W,order(i),f_current);
    f_alpha=1/(1+exp(-f_ener));
       if rand<f_alpha
       f_current(order(i))=1;
       else
       f_current(order(i))=0;
       end

    %%%%%%simulate for clamped case%%%%%%%%%
    c_ener=energy(W,order(i),c_current);
    c_alpha=1/(1+exp(-c_ener));
       if rand<c_alpha
       c_current(order(i))=1;
       else
       c_current(order(i))=0;
       end

       if mod(i,update_size)==0
       f_current=generate(f_current,counter+H+1,W);
       count=count+1;
       j=update_size;
       disp(sprintf('Image %d: Epoch %d: updating weight %d th time',p,l,count))
       else
       j=mod(i+update_size,update_size);
       end
       f_record(j,:)=f_current;
       c_record(j,:)=c_current;

   %%%%%weights update%%%%%%%%%%
       if mod(i,update_size)==0
       free=zeros(N+H,N+H,update_size);
       clamped=zeros(N+H,N+H,update_size);
           for ii=1:N+H
                for jj=counter+1:N+H
                    for kk=1:update_size
                        if not(ii==jj) & ii<=counter+H
                        free(ii,jj,kk)=coact(f_record(kk,ii),f_record(kk,jj));
                        clamped(ii,jj,kk)=coact(c_record(kk,ii),c_record(kk,jj));
                        end
```

```
                    end
                    if not(ii==jj) & ii<=counter+H
                    W(ii,jj)=W(ii,jj)+epsilon*(mean(clamped(ii,jj,:))-mean(free(ii,jj,:)));
                    W(jj,ii)=W(ii,jj);
                    end
                end
           end
       end

    end
end
end

f_current=generate(f_current,counter+H+1,W);
r_image=[x(1:280), f_current(counter+H+1:N+H), x(281:counter)];
imshow(reshape(r_image,28,28)')

function e=energy(W,i,x)
e=0;
[a,b]=size(W);
for j=1:b
   e=e+W(i,j)*x(j);
end
end


function W=initweights(counter,N,H)
W=zeros(N+H,N+H);
    for i=1:counter+H
        for j=counter+1:N+H
            if not(i==j) &  i<=counter+H
            W(i,j)=random('Normal',0,0.1)/sqrt(784);
            end

        end
    end
W=W+W';
end


function a=generate(x,c,W)
a=x;
for i=c:length(x)
ener=energy(W,i,x);
alpha=1/(1+exp(-ener));
       if rand<alpha
       a(i)=1;
       else
       a(i)=0;
       end
end
end

function a=coact(x,y)
    a=y*x;
end
```

5.2 Appendix: Testing code images

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Save complete testing image in a variable "test". No need to remove
% middle four lines, the program does it for you.
%
% Write the following in the command line to perform the desired task after
% running this code.
%
%1. display complete source image
%    imshow(reshape(img,28,28)')
%
%2. display incomplete test image:
%    imshow(reshape(prexlmg,28,28)')
%
%3. display reconstructed image:
%    imshow(reshape(r_image,28,28)')
test=ori(1,:);
%%%%%%%%binarize input
for i=1:length(test)
   if test(i)<=0;
   img(i)=0;
   else
   img(i)=1;
   end
end

%%%%construct incomplete image%%%%%%
prex=zeros(1,N+H);
counter=0;
for i=1:N
```

```
taboo=[281:504];
    if not(ismember(i,taboo))
    counter=counter+1;
    prex(counter)=img(i);
    end
end
preximg=[prex(1:280), prex(counter+H+1:N+H), prex(281:counter)];
f_current=prex;
%%%%%%simulate for free case%%%%%%%%%
for i=1:H
ener=energy(W,counter+i,f_current);
f_alpha=1/(1+exp(-f_ener));
       if rand<f_alpha
       f_current(counter+i)=1;
       else
       f_current(counter+i)=0;
       end
end
f_current=generate(f_current,counter+H+1,W);
r_image=[x(1:280), f_current(counter+H+1:N+H), x(281:counter)];

function a=generate(x,c,W)
a=x;
for i=c:length(x)
ener=energy(W,i,x);
alpha=1/(1+exp(-ener));
       if rand<alpha
       a(i)=1;
       else
       a(i)=0;
       end
end
end

function e=energy(W,i,x)
e=0;
[a,b]=size(W);
for j=1:b
   e=e+W(i,j)*x(j);
end
end
```