Backpropagation Neural Network
Hand Written Recognizing Implementation

Fangzhou Li, Diwen Lu

OUTLINE

For this project, we learned to use practical algebraic methods to implement how to recognize hand written digits by utilizing backpropagation neural network. Since both we were new to this knowledge, we could hardly understand the prompt at the beginning. To formulate the most basic understanding of this project, we looked up a lot of documents and videos of explaining what neural network is. Finally, we could start coding, with fully understanding why and how we use backpropagation neural network.

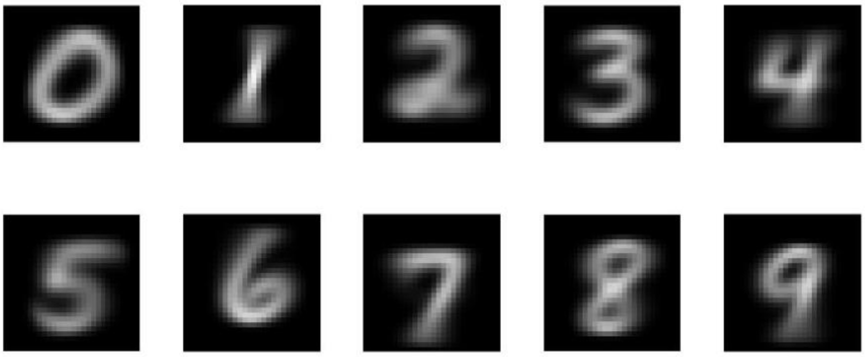
Together, we finished coding for the part of training. We discussed about how we should train the network: the number of layer, the number of neuron, the order of training set, etc. We randomly picked some values mentioned above initially, but for minimizing the time we spent in running a single train set, we decided to train with 1 hidden layer, 50 neurons, and training from 0 to 9 respectively.

However, the problem occurred which after we trained 1, the training result of 0 would be destroyed. More precisely speaking, after training the digit 0, if we tested the digit 0, we got our final output extremely close to [1 0 0 0 0 0 0 0 0 0]. Then we trained the digit 1. Though we got the final output of *test1* close to [0 1 0 0 0 0 0 0 0 0], the final output of *test0* was no longer [1 0 0 0 0 0 0 0 0 0]. Luckily, we found that if we only trained the digits 0 and 1, though the result of *test0* would be incorrect, we found out the output of the first entry of the output vector was slightly larger than other entries, such as [0.1 1 0 0 0 0 0 0 0 0]. This showed that, there was the effect of training 0, but was too small. Therefore, we decided to iteratively train 1000 rows from one train matrix, which means we trained 1000 digits of 0, …, 1000 digits of 9, and repeated this process for 100 times. The result became satisfied.

We wrote different codes for the project but with the same ideas and principles. Therefore, the codes' styles are very different but basically doing the same things. We did different parts of work on figuring out the dependence of parameters. Fangzhou Li dealt with how the number of layer and the number of training sets influenced the error (correct ratio), and Diwen Lu dealt with how the number of neuron influenced the error.

MATHEMATICAL DERIVATIONS

After we downloaded the training and test sets from the webpage of Greenbaum and Chartier, we implement the program to print out the images below:



By Fig.2., the activation function is given by sigmoidal function $OUT = F(NET) = \frac{1}{1+e^{-NET}}$. Then we have $\frac{\partial OUT}{\partial NET} = \frac{\partial F(NET)}{\partial NET} = \frac{\partial\left(\frac{1}{1+e^{-NET}}\right)}{\partial NET} = \frac{0*(1+e^{-NET})-1*\left(e^{-NET}*(-1)\right)}{(1+e^{-NET})(1+e^{-NET})} = \frac{e^{-NET}}{(1+e^{-NET})(1+e^{-NET})} = \frac{1+e^{-NET}-1}{(1+e^{-NET})(1+e^{-NET})} = \frac{1}{1+e^{-NET}} - \frac{1}{(1+e^{-NET})(1+e^{-NET})} = \frac{1}{1+e^{-NET}} * \left(1 - \frac{1}{1+e^{-NET}}\right) = OUT(1-OUT)$. Therefore, the derivative expression given in Fig.2. is correct. Because we notice that

$\lim_{NET\to\infty} OUT = \lim_{NET\to\infty} F(NET) = \lim_{NET\to\infty} \frac{1}{1+e^{-NET}} = 1$, and $\frac{1}{1+e^{-NET}}$ is an increasing

function with its range from 0 to 1 and with the value at NET = 0 being $\frac{1}{2}$. Thus,

when NET approaches to negative ∞, OUT is close to 0 from left, whereas when NET is large, say, approaching ∞, then OUT is close to 1 from left. For activation function, we have other options as well, such as *identity*, *binary step*, *tanh*, *arctan*, and *softsign* functions. These functions can map all the real number set onto a small interval.

When we are to calculate the delta on all the neurons in last hidden layer, D_j the vector of deltas for the hidden layer. D_k is the set of deltas at the output layer. W_k the set of weights at the output layer. O_j is the output vector of layer j. I is the vector with all components equal to 1. Without loss of generality, suppose there are n neurons in each hidden layer.

Wk = $\begin{bmatrix} W11,k & W12,k & \dots & W1\ 10,k \\ W21,k & W22,k & \dots & W2\ 10,k \\ \vdots & \vdots & \dots & \vdots \\ Wn1,k & Wn2,k & \dots & Wn\ 10,k \end{bmatrix}$, Dk = $\begin{bmatrix} \delta k1 \\ \delta k2 \\ \delta k3 \\ \vdots \\ \delta kn \end{bmatrix}$, Oj = $\begin{bmatrix} Oj1 \\ Oj2 \\ Oj3 \\ \vdots \\ Ojn \end{bmatrix}$, I = $\begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$

Then $Dj = DkWk^t \otimes [Oj \otimes (I - Oj)] =$

$$\begin{bmatrix} W11,k & W12,k & \dots & W1\,10,k \\ W21,k & W22,k & \dots & W2\,10,k \\ \vdots & \vdots & \dots & \vdots \\ Wn1,k & Wn2,k & \dots & Wn\,10,k \end{bmatrix} \begin{bmatrix} \delta 1,k \\ \delta 2,k \\ \delta 3,k \\ \vdots \\ \delta 10,k \end{bmatrix} \otimes \left(\begin{bmatrix} Oj1 \\ Oj2 \\ Oj3 \\ \vdots \\ Ojn \end{bmatrix} \otimes \left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} - \begin{bmatrix} Oj1 \\ Oj2 \\ Oj3 \\ \vdots \\ Ojn \end{bmatrix} \right) \right).$$

Then $Dj = \begin{bmatrix} \delta 1,j \\ \delta 2,j \\ \delta 3,j \\ \vdots \\ \delta n,j \end{bmatrix} = \begin{bmatrix} (\sum_{i=1}^{10} W1i,k * \delta i,k) * \big(Oj1 * (1 - Oj1)\big) \\ (\sum_{i=1}^{10} W2i,k * \delta i,k) * \big(Oj2 * (1 - Oj2)\big) \\ (\sum_{i=1}^{10} W3i,k * \delta i,k) * \big(Oj3 * (1 - Oj3)\big) \\ \vdots \\ (\sum_{i=1}^{10} Wni,k * \delta i,k) * \big(Ojn * (1 - Ojn)\big) \end{bmatrix} \Leftrightarrow$

$$\delta p,j = OUTp,j(1 - OUTp,j)\left(\sum_{i=1}^{10} Wpi,k * \delta i,k\right).$$

Dependence on Parameter

In training the neural network, we have one input layer, one hidden layer, and one output layer, and two weight matrix V and W connecting the three layers for simplicity. Because in our experiment, the contribution of increasing number of hidden layers is far less than that of increasing number of neurons in each hidden layer. We have 60000 training images in total. This could be a very time-consuming task. Therefore, we draw a subset of these images, 500 training images for each digit, and then we have 5000 training images in total to train. We test our 10000 testing digits based on the weights matrix outputted by training process.

The Error of the 2-norm of an error vector for all testing images, with the entry i being the 2-norm of the difference between i^{th} testing image output vector with its target output vector. Here, we are also using correct ratio of test cases which is descriptive.

# of Layer = 1
# of Neuron = 50

# of Train Set	20 (*10000)	50	100	200	400
Correct Ratio	10.14%	24.55%	72.31%	85.87%	92.73%
Error	120.48	88.69	57.93	42.98	35.10

# of Neuron = 50
# of Train Set = 100 (*10000)

# of Layer	1	2	3	4	5
Correct Ratio	72.31%	79.13%	69.42%	66.03%	40.32%

Error	57.93	53.58	60.28	64.32	70.38

# of Layer = 1
# of Train Set = 800 images per digit, 8000 in total

# of Neuron	1	30	50	70	90
Correct Ratio	17.59%	60.89%	83.02%	91.16%	92.86%
Error	98.5839	79.5765	53.7042	39.0051	34.8199

The correct ratio becomes larger (error becomes smaller) if the number of training set is larger, which explains that the more we train the network, the more situation it can deal with. However, for the number of layer, it is not always good to have a large amount of them. Observing the number of layer, we see that if there are two hidden layers, the correct ratio is the highest. Due to the over-fit situation, after passing the peak, the more layers it has, the lower correct ratio it becomes. The same happens to the number of neuron, we tried the neuron number of 784, which gave us the correct ratio 10%.

TRAINING

```
% Load database.
load mnist_all.mat;
% Inputs hidden layer number and neuron number.
prompt1 = 'Input your layer number: ';
l = input(prompt1);
prompt2 = 'Input your neuton per layer: ';
n = input(prompt2);
% Initialize all values needed.
W_in = (2*rand(784, n, 1)-1);
W_out = (2*rand(n, 10, 1)-1);
W_hid = zeros(n, n, l - 1);
for i = 1 : l - 1
    W_hid(:, :, i) = (2*rand(n, n, 1)-1);
end

OUT_hid = zeros(l, n);
OUT_out = zeros(1, 10);
delta_hid = zeros(l, n);
delta_out = zeros(1, 10);


% Training all digits
for itr = 1 : 100
    fprintf('Iterating for %d...\n', itr);
```

```
for count = 1 : 10
    if count == 1
        p = im2double(train0);
        TARGET = [1 0 0 0 0 0 0 0 0 0];
    elseif count == 2
        p = im2double(train1);
        TARGET = [0 1 0 0 0 0 0 0 0 0];
    elseif count == 3
        p = im2double(train2);
        TARGET = [0 0 1 0 0 0 0 0 0 0];
    elseif count == 4
        p = im2double(train3);
        TARGET = [0 0 0 1 0 0 0 0 0 0];
    elseif count == 5
        p = im2double(train4);
        TARGET = [0 0 0 0 1 0 0 0 0 0];
    elseif count == 6
        p = im2double(train5);
        TARGET = [0 0 0 0 0 1 0 0 0 0];
    elseif count == 7
        p = im2double(train6);
        TARGET = [0 0 0 0 0 0 1 0 0 0];
    elseif count == 8
        p = im2double(train7);
        TARGET = [0 0 0 0 0 0 0 1 0 0];
    elseif count == 9
        p = im2double(train8);
        TARGET = [0 0 0 0 0 0 0 0 1 0];
    elseif count == 10
        p = im2double(train9);
        TARGET = [0 0 0 0 0 0 0 0 0 1];
    end

    for i = 1 : 1000
        % Initialize OUT for the hidden layers.
        OUT_hid(1, :) = 1 ./ (1 + exp(-p(i, :) * W_in));
        for j = 1 : l - 1
            OUT_hid(j + 1, :) = 1 ./ (1 + exp(-OUT_hid(j, :) *
W_hid(:, :, j)));
        end
        % Initialize OUT for the output layer.
        OUT_out = 1 ./ (1 + exp(-OUT_hid(l, :) * W_out));
        % Calculate delta array for the output layer.
        delta_out = OUT_out .* (1 - OUT_out) .* (TARGET - OUT_out);
```

```
            % Updata the W_out.
            W_out = W_out + 0.1 * OUT_hid(l, :)' * delta_out;
            % Calculate and updata all W_hid.
            delta_hid(l, :) = (delta_out * W_out') .* (OUT_hid(l, :) .*
(ones(1, n) - OUT_hid(l, :)));
            for m = 1 : l - 1
                W_hid(:, :, l - m) = W_hid(:, :, l - m) + 0.1 *
OUT_hid(l - m, :)' * delta_hid(l - m + 1, :);
                delta_hid(l - m, :) = (delta_hid(l - m + 1, :) *
W_hid(:, :, l - m)') .* (OUT_hid(l - m, :) .* (1 - OUT_hid(l - m,
:)));
            end
            % Update W_in.
            for j = 1 : 784
                for k = 1 : n
                    W_in(j, k) = W_in(j, k) + 0.1 * delta_hid(1, k) *
p(i, j);
                end
            end
        end
    end
end
```

TESTING

```
correct = 0;
total = 0;
for itr = 1 : 10
    if itr == 1
        [row, ~] = size(test0);
        t = im2double(test0);
    elseif itr == 2
        [row, ~] = size(test1);
        t = im2double(test1);
    elseif itr == 3
        [row, ~] = size(test2);
        t = im2double(test2);
    elseif itr == 4
        [row, ~] = size(test3);
        t = im2double(test3);
    elseif itr == 5
        [row, ~] = size(test4);
        t = im2double(test4);
```

```
    elseif itr == 6
        [row, ~] = size(test5);
        t = im2double(test5);
    elseif itr == 7
        [row, ~] = size(test6);
        t = im2double(test6);
    elseif itr == 8
        [row, ~] = size(test7);
        t = im2double(test7);
    elseif itr == 9
        [row, ~] = size(test8);
        t = im2double(test8);
    elseif itr == 10
        [row, ~] = size(test9);
        t = im2double(test9);
    end

    total = total + row;

    for i = 1 : row
        Output = 1 ./ (1 + exp(-t(i, :) * W_in));
        for j = 1 : l - 1
            Output = 1 ./ (1 + exp(-Output * W_hid(:, :, j)));
        end
        Output = 1 ./ (1 + exp(-Output * W_out));
        [m, n] = max(Output);
        %disp(Output);
        if n == itr
            correct = correct + 1;
        else
            fprintf('Expected Value: %d\n', itr - 1);
            disp(Output);
        end
    end
end

fprintf('Correct Ratio: %f\n', (correct / total));
```