CHALMERS UNIVERSITY OF TECHNOLOGY

FFR135 Artificial Neural Network

Example_Sheet_1

Yankun Xu (940630-0237)

(yankun@student.chalmers.se)

2017/9/21

1. Deterministic Hopfield model:

In this case, we have p random patterns, which means our storaged pattern $\zeta_i^{(\mu)} = \pm 1$ with equal probability of 0.5. Now, we have Hopfield model with synchronous updating rules: $S_i = sgn(\sum_{j=1}^{N} w_{ij} S_j)$, where S is the state of neuron, p is the number of storaged patterns and $w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \zeta_i^{(\mu)} \zeta_j^{(\mu)}$ according to Hebb's rule.

1a: When $t = 0$ (t means iteration), $S_i(t = 0) = S_j = \zeta_j^{(\mu)}$, we will successfully return the pattern if $S_i(t = 1) = \zeta_i^{(\mu)}$, but it could not always return right, the one-step probability P_{Error} is the probability we cannot return the right pattern.
Now, we follow the updating rule:

$$sgn(\sum_{j=1}^{N} w_{ij} S_j) = sgn(\frac{1}{N} \sum_{j=1}^{N} \sum_{v=1}^{p} \zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)})$$

$$= sgn(\frac{1}{N} \sum_{j=1}^{N} \zeta_i^{(\mu)} \zeta_j^{(\mu)} \zeta_j^{(\mu)} + \frac{1}{N} \sum_{j=1}^{N} \sum_{v \neq \mu}^{p} \zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)})$$

$$= sgn(\zeta_i^{(\mu)} + \frac{1}{N} \sum_{j=1}^{N} \sum_{v \neq \mu}^{p} \zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)}) \quad (\zeta_j^{(\mu)} \zeta_j^{(\mu)} = 1)$$

The part $\frac{1}{N} \sum_{j=1}^{N} \sum_{v \neq \mu}^{p} \zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)}$ is so-called cross-talk term. Then we could use $-\zeta_i^{(\mu)}$ multiply this term in order to build up a new symbol $C_i^{(\mu)}$, which equals $-\frac{1}{N} \sum_{j=1}^{N} \sum_{v \neq \mu}^{p} \zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)} \zeta_i^{(\mu)}$. What we need to do now is to investigate this $C_i^{(\mu)}$.
When $C_i^{(\mu)} < 1$, the state S_i will be satisfied, and we will get unstable bits if $C_i^{(\mu)} > 1$.
In this case, $w_{ii} \neq 0$ need to be taken into account so that we will get:

$$C_i^{(\mu)} = -\frac{1}{N} \sum_{v \neq \mu}^{p} \zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)} \zeta_i^{(\mu)} - \frac{1}{N} \sum_{j=1, j \neq i}^{N} \sum_{v \neq \mu}^{p} \zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)} \zeta_i^{(\mu)}$$

We can look at right part of $C_i^{(\mu)}$ firstly, without scalar N, this part can be consider as a random variable of binary values ± 1 with same probability 0.5 , due to the condition of large p and N, we could use centre limit theorem to get the mean $\mu = 0$ and variance $\sigma^2 \approx p/N$ of this part. Then looking at left part, $\zeta_i^{(v)} \zeta_j^{(v)} \zeta_j^{(\mu)} \zeta_i^{(\mu)}$ always equals to 1 because each neuron connects itself, so left part is $-(p-1)/N$. After combining theses two parts, we will get mean and variance of $C_i^{(\mu)}$, which equals to $(1-p)/N$ and p/N separately.

Due to $C_i^{(\mu)}$ is i.i.d, it follows Gaussian distribution, and $P_{Error} = \frac{1}{2}(1 - erf(\frac{x}{\sqrt{2}}))$ (erf means error function). Then we could get our one-step error probability:
$P(C_i^{(\mu)} > 1) = \frac{1}{2}(1 - erf(\frac{1-\mu}{\sqrt{2}\sigma})) = \frac{1}{2}(1 - erf(\frac{N+p-1}{N}\sqrt{\frac{N}{2p}})) \approx \frac{1}{2}(1 - erf(\frac{1+\alpha}{\sqrt{2\alpha}})), (\alpha = p/N)$

1b:

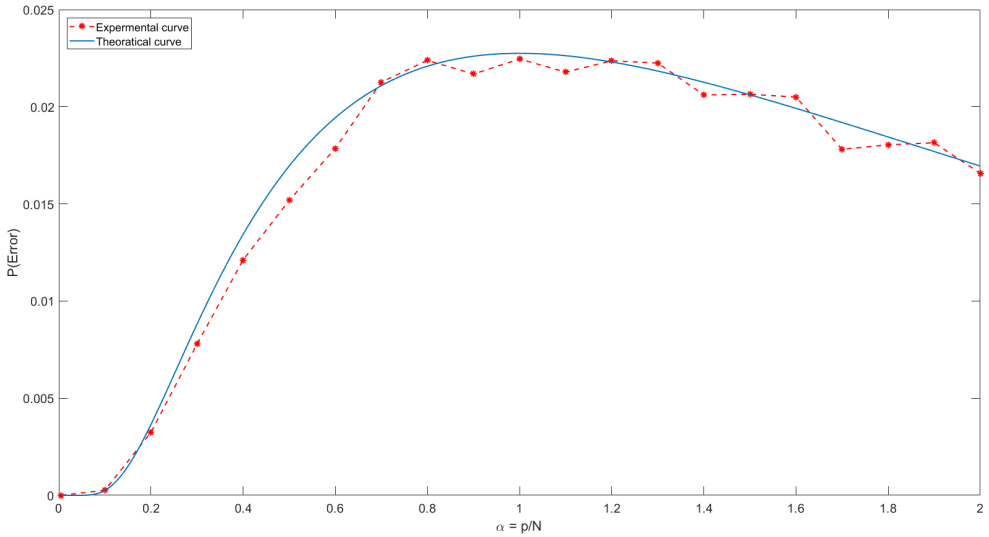


Figure 1: Experimental P_error curve and theoretical P_error curve

Compared to the case of $w_{ii} = 0$ in the lecture note, when we take into account $w_{ii} \neq 0$, the probability of one-step error is decreased significantly. According to the theoretical curve, the probability of one-step error achieves the highest value which equals around 0.023 when α $(= p/N)$ equals 1. After that, the probability will be decreased slowly along with the increasing number of patterns.

In the experiments, 10^6 bits are used to generate the probability of one-step error for each number of patterns. When I set up the parameter of the number of samples, I round the number, so I don't use full 10^6 bits for some numbers of patterns. In Figure 1 the experimental curve is very closed to the theoretical curve, but has some slight fluctuations when α is large than 0.8, I think it is caused by the increasing number of patterns.

2: Stochastic Hopfield model: phase diagram

In this task, we will use Hopfield model and random patterns to investigate the relationship between order parameter and the number of patterns and neurons, and discuss its phase diagram. We calculate the m_1 by following step: $b_i(t) = \sum_{j=1}^{N} w_{ij} S_j(t-1)$, $S_i(t) = 1$ with probability of $g(b_i)$, or $= -1$ with probability of $1 - g(b_i)$
$m_1(t) = \frac{1}{N} \sum_i S_i(t) \zeta_i^{(1)}$. In this case, we have $N = 200$ and noise parameter $\beta = 2$.

2a:

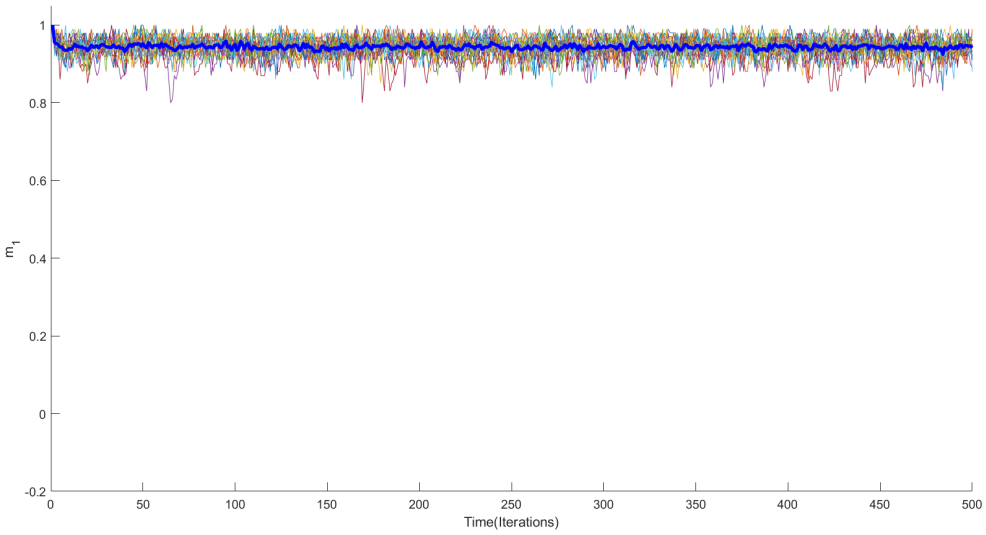


Figure 2: 20 experiments of order parameter m_1 ($p = 5$)

In Figure 2 there are 20 order parameter m_1 curves when $p = 5$, and the thickest blue line is the mean of 20 curve. It takes about 10 iterations for the network to reach steady state and the stable value is around 0.95. It is so fast because we have 200 neurons but only have 5 patterns, which means the network has low probability of one-step error and good storage capacity. Among all 20 different experiments, we get similar performance, I think this network performs well enough.

I think this case is consistent with the phase diagram in Lecture notes. In this case, $\beta^{-1} = 0.5$ which is smaller than 1 and $\alpha = p/N = 0.025$ which is quite small, so m_1 should be at middle left part of the phase diagram.

The factors have effect on the boundary of phase diagram are α and noise parameter β, and we also know that the critical value is around 0.14, which means the network could storage much more patterns if the number of neurons N is infinite. Therefore, if N is finite, the order parameter m_1 will be closed to 1 only if network stores much less patterns, in other word, m_1 will be decreased if we have finite N and fixed number of patterns.

2b:

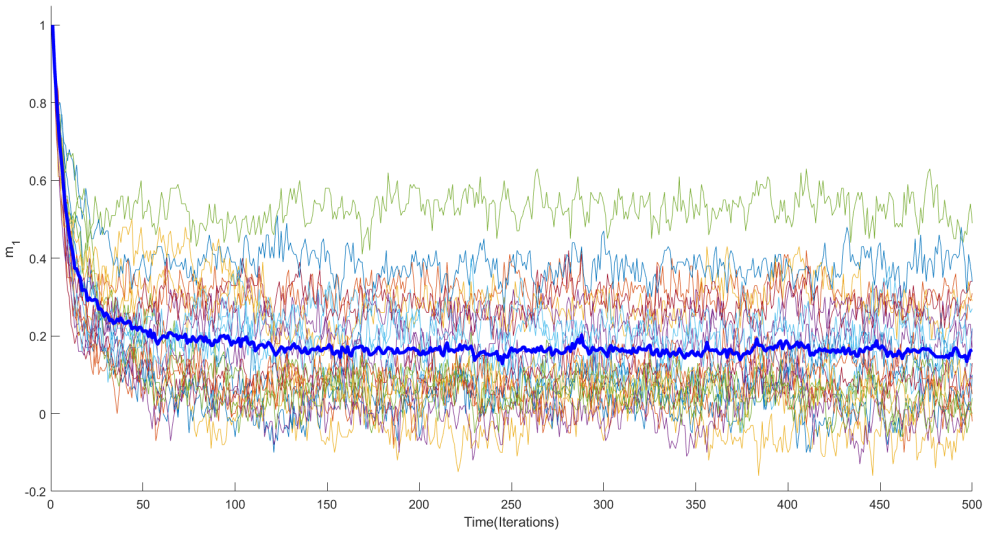


Figure 3: 20 experiments of order parameter m_1 ($p = 40$)

When $p = 40$, order parameter m_1 is decreased significantly compared to the case of $p = 5$. In Figure 3, the thickest blue line is also the mean of 20 experiments. 20 experiments have different performance, most m_1 achieve about 0 when they get steady state even if there are some experiments m_1 achieve stable value around from 0.2 to 0.5 separately, by the way all experiments achieve steady state after about 50 iterations.

This network does not perform very well, the order parameter is far away from one and closer to zero because there are too many patterns in this case. According to my understanding, if order parameter equals to one, it means our output state S_i is same as initial input pattern $\zeta^{(\mu)}$, if order parameter equals to zero, it means S_i is a random pattern which has exact 0.5 probability of error bits of $\zeta^{(\mu)}$. In this time, $\alpha = p/N = 0.2$ which is larger than critical value, which means order parameter should fall in the right blank part of the phase diagram whose order parameter equals to zero. Although there are some order parameter larger than 0 among my all 20 experiments, their values are also closer to 0, which means the network still cannot retrieve pattern successfully.

3: Back propagation

Neural network is good at supervised learning, this task is to train the weights and bias among the network with and without hidden layer to classify the data. We use training set to train and valid our result using validation set. In 3a, we use a simple network with only 2 input neurons and one output neuron, but in 3b, we will add one hidden layer with 4 neurons, which is able to improve our performance of classification significantly. In stead of using theoretical gradient descent rule, I use stochastic gradient descent to decrease computational complexity.

3a:

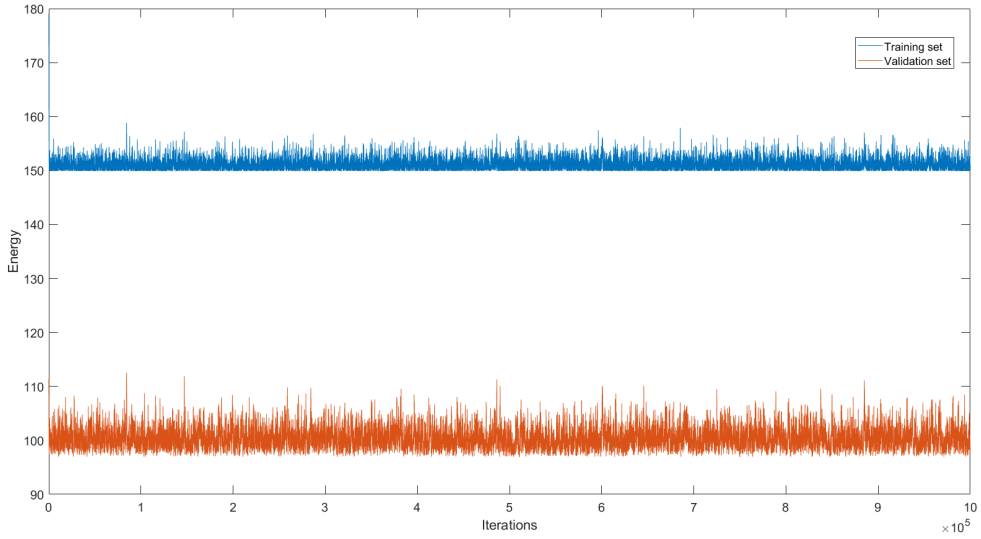
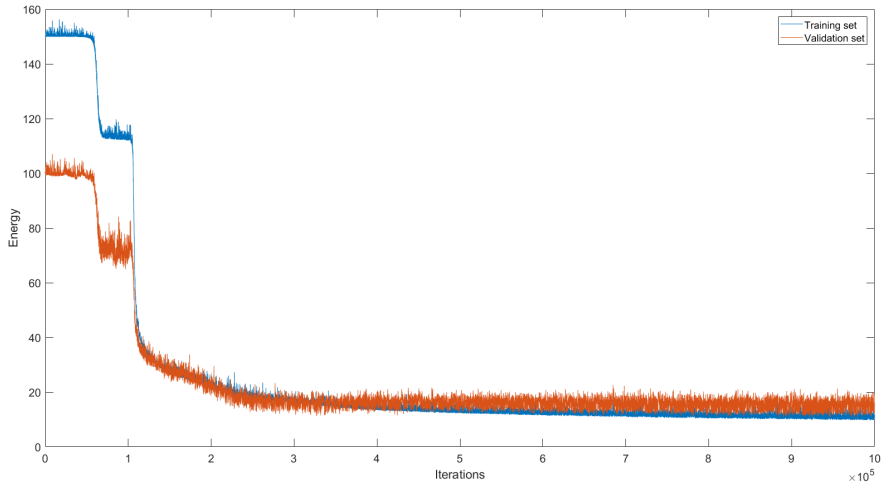


Figure 4: Energy of training and validation set (without hidden layer)

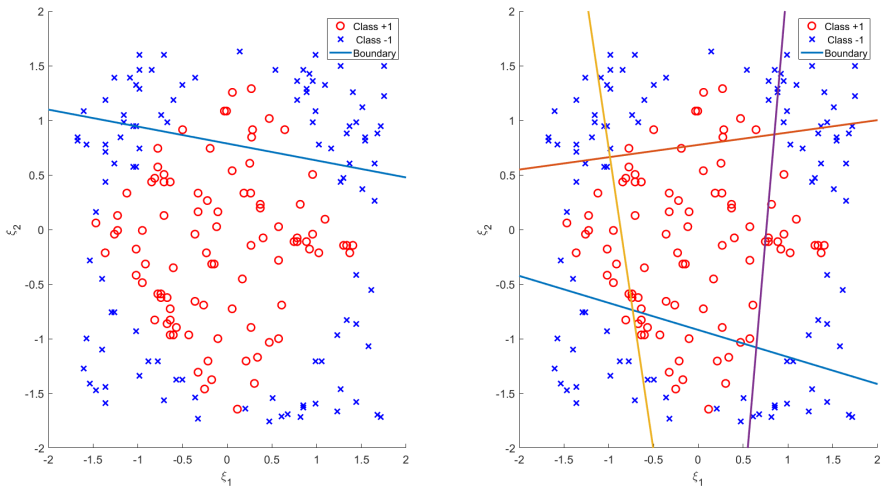
In Figure 4 we can see the energy of training set and validation set during the training, unfortunately both energy never have any decreasing trend, their values are stable at 150 for training set and 100 for validation with some fluctuations.

Classification error is calculated by the formula: $C_v = \frac{1}{2p}\sum_{\mu=1}^{p}|\zeta^{(\mu)} - sgn(O^{(\mu)})|$, after 10 independent experiments, the average of classification error is 0.4257 for training set and 0.428 for validation set, minimum error is 0.353 for training set and 0.345 for validation set. The variance of classification error is 0.0021 for training set and 0.0035 for validation set. The both average classification error around 0.42 means nearly half of data is classified incorrectly in both set, so the network does not work and energy does not go down. Although the variance of both error is small, it still indicates 10 classification errors are not very stable.

3b:



(a) Energy of training and validation set (with one hidden layer)



(b) Classification boundary in 2 networks

Figure 5: Energy and Classification illustration

In (a) of Figure 5, we can see that energy of two set is decreased significantly, both energy fall in the range of 10 to 20 finally. After 10 independent experiments, the average of classification error is 0.027 for training set and 0.049 for validation set, minimum error is 0.0133 for training set and 0.035 for validation set. The variance of classification error is 1.5e-04 for training set and 1.3e-04 for validation set. The error is much smaller than the case in 3a, both average error is less than 0.1, which means this network is trusted to classify the data. And the variance is quite small, it means this network performs always well enough no matter how many experiments are made.

This network works with one hidden layer, in the (b) of Figure 5, we can see that in 3a there is only one line to classify the data in 2-dimension space, but in 3b, one hidden layer with 4 neurons is able to generate 4 lines to classify these data, it is obviously more reasonable and powerful to make classification for the data.

Appendix: Codes for three tasks in Matlab

Task1:

```
clear ;
clc;

p = [1 linspace(20,400,20)];
N = 200; % number of nuerons
bits = 100000;
P_error = zeros(length(p),1);
error_bits = zeros(length(p),1);

for i = 1 : length(p)

    iter = round( bits / (p(i) * N));
    error_vector = zeros( p(i)*N , iter);

    for k = 1:iter
        patterns = 2 * randi([0,1],N,p(i)) - 1 ;
        weight = zeros(N);

        for j = 1:p(i)
            weightj = patterns(:,j) * patterns(:,j)'; % weight of each pattern
            weight = weight + (1/N) .* weightj;
        end

        for m = 1:p(i)
            S_j = patterns(:,m);
            S_i = sign(weight * S_j);
            S_i = S_i + (S_i==0) .* (2*randi([0,1],N,1)-1);
            error_vector(((m-1)*N+1):m*N,k) = S_i ~= S_j;
        end
    end
    error_vector = error_vector(:);
    index = randperm(length(error_vector));
    index = index(1:iter*p(i)*N);
    error_bits(i) = sum(error_vector(index));
    P_error(i) = error_bits(i) / length(index);
end

p2 = [1:400];
P_error1 = 0.5 * (1 - erf( (1 + p2./N) ./ sqrt( 2 * p2 ./ N)));

plot( p ./ N ,P_error,'r--*',p2 ./ N,P_error1,'Markersize',8,'linewidth',1.5);
xlabel('\alpha = p/N');
ylabel('P(Error)');
legend('Expermental curve','Theoratical curve','Location','northwest')
set(gca,'fontsize', 15)
```

Task2:

```
clear;
clc;

p = 40; % p=5 for 2a, p=40 for 2b
N = 200;
t = 500;
beta = 2;
m20 = zeros(t,20);
iter = 20;

for k = 1:iter
    theta = 2 * randi([0,1],N,p) - 1 ;
    weight = zeros(N);

    for i = 1:p
       weightj = theta(:,i) * theta(:,i)';
       weight = weight + (1/N) .* weightj;
    end

    for i = 1:N
        weight(i,i) =0;
    end

    states = zeros(N,t);
    m1 = zeros(t,1);

    state_0 = theta(:,1);
    states(:,1)= state_0;
    m1(1) = state_0' * state_0 / N;

for j=1:t-1
    b = weight * states(:,j);
    g = 1 ./ (1 + exp(-2 * beta .* b));

    for n = 1:N
       if rand(1,1)<g(n)
           states(n,j+1) = 1;
       else
           states(n,j+1) = -1;
       end
    end
    m1(j+1) = states(:,j+1)'*state_0 / N;
end
   m20(:,k) = m1;
end
m20_mean = mean(m20,2);
hold on
```

```
plot([1:t],m20(:,1:3));
plot([1:t],m20_mean,'b','linewidth',4)
xlabel('Time(Iterations)')
ylabel('m_{1}')
set(gca,'fontsize', 15)
ylim([-0.2 1.05]);
```

Task3:

```
clc;
clear;

train_data = importdata('train_data_2017.txt');
valid_data = importdata('valid_data_2017.txt');

train_input = train_data(:,1:2);
train_target = train_data(:,3);
valid_input = valid_data(:,1:2);
valid_target = valid_data(:,3);

% generalize
for i = 1:2
   train_input(:,i) = (train_input(:,i)-...
       mean(train_input(:,i)))/std(train_input(:,i));
   valid_input(:,i) = (valid_input(:,i)-...
       mean(valid_input(:,i)))/std(valid_input(:,i));
end

% parameters
iter = 1e6;
lr = 0.02; % learning_rate
beta = 0.5;
runs = 10;

%% 3a
% initialize
energy_t_a = zeros(iter,1);
energy_v_a = zeros(iter,1);
C_t_a = zeros(runs,1);
C_v_a = zeros(runs,1);
weight_10 = zeros(2,runs);
bias_10 = zeros(1,runs);

for r = 1:runs

    weight = 0.4*rand(2,1)-0.2;
    bias = 2*rand(1,1)-1;

    for i = 1:iter
    % feed pattern
    index = randi([1 size(train_input,1)]);
    feed_pattern = train_input(index,:)';
    pattern_traget = train_target(index);

    b = weight' * feed_pattern - bias;
    output = tanh(beta*b);
```

```
    % update
    weight = weight + lr*beta*(1-output^2)*(pattern_traget - output) .* feed_pattern;
    bias = bias - lr*beta*(1-output^2)*(pattern_traget - output);

    output_t = tanh(beta.*(train_input*weight - bias));
    energy_t_a(i) = 0.5 * sum((train_target - output_t).^2);

    output_v = tanh(beta.*(valid_input*weight - bias));
    energy_v_a(i) = 0.5 * sum((valid_target - output_v).^2);
    end
    weight_10(:,r) = weight;
    bias_10(:,r) = bias;
    % classification error of training and validation set
    C_t_a(r) = 0.5 * sum(abs(train_target-sign(output_t))) / size(train_data,1);
    C_v_a(r) = 0.5 * sum(abs(valid_target-sign(output_v))) / size(valid_data,1);

end

% analyze classification error
mean_ct_a = mean(C_t_a); % mean of C in training set
var_ct_a = var(C_t_a); % variance of C in training set
mini_ct_a = min(C_t_a); % minimum C in training set
mean_cv_a = mean(C_v_a); % mean of C in validation set
var_cv_a = var(C_v_a); % variance of C in validation set
mini_cv_a = min(C_v_a); % minimum C in validation set

%% plot
plot([1:iter],[energy_t_a energy_v_a]);
legend('Training set','Validation set')
xlabel('Iterations')
ylabel('Energy')
set(gca,'fontsize', 15)
% axis([1 runs 0 1])

%% 3b
energy_t_b = zeros(iter,1);
energy_v_b = zeros(iter,1);
C_t_b = zeros(runs,1);
C_v_b = zeros(runs,1);
weight_input_hidden_10 = zeros(2,4,runs);
bias_input_hidden_10 = zeros(4,runs);
for r = 1:runs

    % initialize weight and bias
    % we have 4 neurons in hidden layer
    weight_input_hidden = 0.4*rand(2,4)-0.2;
    weight_hidden_output = 0.4*rand(4,1)-0.2;
```

```
    bias_input_hidden = 2*rand(4,1)-1;
    bias_hidden_output = 2*rand(1,1)-1;

    for i = 1:iter
        % feed pattern
        index = randi([1 size(train_input,1)]);
        feed_pattern = train_input(index,:)';% 2*1
        pattern_traget = train_target(index);

        bj = weight_input_hidden' * feed_pattern - bias_input_hidden;
        V = tanh(beta*bj); % output of hidden layer 4*1
        bi = weight_hidden_output' * V - bias_hidden_output;
        output = tanh(beta*bi);

        % update
        weight_input_hidden = weight_input_hidden + lr*beta*...
            beta*(1-output^2)*(pattern_traget - output).* ...
            feed_pattern*(weight_hidden_output.*(1-V.^2))';
        bias_input_hidden = bias_input_hidden - ...
            lr*beta*(1-V.^2)*beta*(1-output^2)*...
            (pattern_traget - output).*weight_hidden_output;
        weight_hidden_output = weight_hidden_output + lr*beta*...
            (1-output^2)*(pattern_traget - output) .* V;
        bias_hidden_output = bias_hidden_output - ...
            lr*beta*(1-output^2)*(pattern_traget - output);

        V_t = tanh(beta.*(weight_input_hidden'*train_input' - bias_input_hidden));
        output_t = tanh(beta.*(weight_hidden_output'*V_t - bias_hidden_output));
        energy_t_b(i) = 0.5 * sum((train_target - output_t').^2);

        V_v = tanh(beta.*(weight_input_hidden'*valid_input' - bias_input_hidden));
        output_v = tanh(beta.*(weight_hidden_output'*V_v - bias_hidden_output));
        energy_v_b(i) = 0.5 * sum((valid_target - output_v').^2);

    end

    weight_input_hidden_10(:,:,r) = weight_input_hidden;
    bias_input_hidden_10(:,r) = bias_input_hidden;
    % classification error of training and validation set
    C_t_b(r) = 0.5 * sum(abs(train_target-sign(output_t'))) / size(train_data,1);
    C_v_b(r) = 0.5 * sum(abs(valid_target-sign(output_v'))) / size(valid_data,1);

end

%analyze classification error
mean_ct_b = mean(C_t_b); % mean of C in training set
var_ct_b = var(C_t_b); % variance of C in training set
```

```
mini_ct_b = min(C_t_b); % minimum C in training set
mean_cv_b = mean(C_v_b); % mean of C in validation set
var_cv_b = var(C_v_b); % variance of C in validation set
mini_cv_b = min(C_v_b); % minimum C in validation set

%% plot
plot([1:iter],[energy_t_b energy_v_b]);
legend('Training set','Validation set')
xlabel('Iterations')
ylabel('Energy')
set(gca,'fontsize', 15)
% axis([1 runs 0 1])

%% classification plot

class1 = valid_input(valid_target == 1,:);
class2 = valid_input(valid_target == -1,:);

x = linspace(-2,2,100);
% boundary in 3a
weight = weight_10(:,find(C_v_a==mini_cv_a));
bias = bias_10(:,find(C_v_a==mini_cv_a));
line = bias/weight(2) - x * weight(1)/weight(2) ; % weight' * input = bias
subplot(1,2,1)
hold on
plot(class1(:,1),class1(:,2),'ro',class2(:,1),class2(:,2),...
    'bx','MarkerSize',10,'linewidth',2);
plot(x,line,'linewidth',2.5);
legend('Class +1',' Class -1', 'Boundary')
set(gca,'fontsize', 15)
xlabel('\xi_1')
ylabel('\xi_2')
ylim([-2 2])

% boundary in 3a
% we have 4 lines due to 4 neurons in the hidden layer
weight_input_hidden = weight_input_hidden_10(:,:, find(mini_cv_b == C_v_b));
bias_input_hidden = bias_input_hidden_10(:,find(mini_cv_b == C_v_b));

lines = zeros(4,length(x));

for l = 1:4
    lines(l,:) = bias_input_hidden(l)/weight_input_hidden(2,l) - ...
    x .* weight_input_hidden(1,l)/weight_input_hidden(2,l);

end

subplot(1,2,2)
```

```
hold on
plot(class1(:,1),class1(:,2),'ro',class2(:,1),class2(:,2),...
    'bx','MarkerSize',10,'linewidth',2);
plot(x,lines,'linewidth',2.5);
legend('Class +1',' Class -1', 'Boundary')
set(gca,'fontsize', 15)
xlabel('\xi_1')
ylabel('\xi_2')
ylim([-2 2])
```