CNS 187 Neural Computation - Problem Sheet 4

Ling Li, ling@cs.caltech.edu

October 31, 2000

4.1 Associative Memory

I'd like to first list some questions common to the whole problem below. I am not sure with those with symbol ♠.

- **What gain should be used?** Since $g_\beta(x) = \tanh(\beta x)$ is used as the activation function, this question equals 'What value should β be?'. The answer is, for weight defined by (2) or (5), β should be better bigger than 6 for this problem.
  - (1) hints that for a stable random state ν, $g_\beta(h_i^\nu) \approx g_\beta(S_i^\nu)$[1] should approximately equals S_i^ν.[2] This urges us to find out the relation between β and x when

    $$\tanh(\beta x) = x,$$

    which is shown in Figure 1. For $|S_i^\nu| = 0.99$, β shouldn't be less than 3.

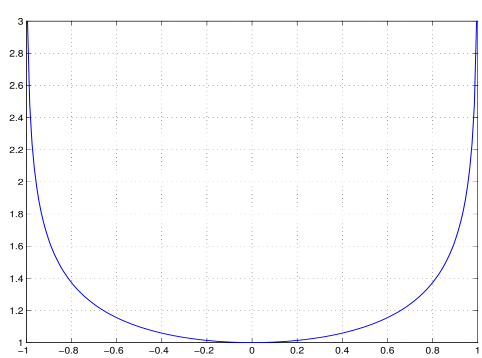
    

    Figure 1: Function $\beta = \tanh^{-1}(x)/x$.

  - Simulations with $\eta = 0.01$ shows that, when memorizing patterns of 10 digits, small β cannot ensure $|S_i^\nu| \geq 0.5$ in stable states, especially for $i = 32$. For (2), $\beta \leq 4.1$ and for (5), $\beta \leq 5.7$ don't work. If only the first 2 patterns[3] are used, $\beta \leq 2.1$ for (2) and $\beta \leq 2.2$ for (5) don't work.

[1] Since we hope the cross-talk term C_i^ν has zero mean, we get h_i^ν is approximately S_i^ν.

[2] We know that h_i^ν makes sense in a high-gain network. However, for stable states in a low-gain network, h_i^ν can be regarded as $V_i^\nu(\infty)$. So the argue also applies to low-gain networks.

[3] Why use first 2 patterns? Since from experiments in 4.1.2 and Figure 5, the first 2 patterns can be correctly memorized by the network.

- **What time step η should be used for the simulation?** Since we will work with a high-gain network, we can image that in some period of time, the $\vec{S}$-state of the network will remain the same until some V_i changes its sign and then the network enters another period of time with a new $\vec{S}$-state. This 'imaginary' is also used in 4.2.2.

  Thus for a period of simulation time that $\vec{S}$-state remains unchanged, $F_i = \sum_{j=1}^{N} W_{ij} S_j$ is constant. So

  $$\Delta V_i = \eta \dot{V}_i = \eta \left(-V_i + \sum_{j=1}^{N} W_{ij} S_j \right) = \eta \left(-V_i + F_i \right),$$

  and denoting the simulation steps by numbers $k, k+1, \ldots, k'$,

  $$V_i(k') = (1-\eta) V_i(k'-1) + \eta F_i = \cdots = (1-\eta)^{k'-k} V_i(k) + \left[1 - (1-\eta)^{k'-k} \right] F_i.$$

  This is just in the same form as the continuous version

  $$V_i(t + \Delta t) = e^{-\Delta t} V_i(t) + \left(1 - e^{-\Delta t} \right) F_i.$$

  As long as the $\vec{S}$-state doesn't change, $V_i(k)$ will be on the trajectory of $V_i(t)$, no matter what value η is. The only problem is, when the k'^{th} step of simulation causes a change in sign of one V_i, which means the $\vec{S}$-state may have changed between step $(k'-1)$ and k', there will be a simulation error which is related to the value of η. However, η really affects little in the way to stable state.

  I tried several values for η in the simulation: 0.01, 0.1, 0.2, and 0.5. The results turned out no difference with same β. $\eta = 0.1$ was used in the following experiments.

- **`Round()` or `sign()`?** For some situations, such as to memorize only one sample `pat(:,1)`, there always exists small $S_i (\approx 0)$ for some random input patterns even using $\beta = 200$. So I changed the way of standardizing output patterns in `classifyfixpt.m` from `round()` to `sign()`.

- ♠**Lyapunov energy function.** Remember $S_i = g_\beta(V_i) = \tanh(\beta V_i)$ and $V_i = g_\beta^{-1}(S_i) = \frac{1}{\beta} \tanh^{-1}(S_i)$. Since

  $$\int_0^S \tanh^{-1}(z) dz = \int_0^S \frac{1}{2} \ln \frac{1+z}{1-z} dz = \frac{1}{2} \left[(1+S) \ln(1+S) + (1-S) \ln(1-S) \right],$$

  we have

  $$G(S_i) = \int_0^{S_i} g_\beta^{-1}(z) dz = \frac{1}{2\beta} \left[(1+S_i) \ln(1+S_i) + (1-S_i) \ln(1-S_i) \right],$$

  and the Lyapunov function without external inputs is

  $$\mathcal{L} = -\frac{1}{2} \sum_{i,j} W_{ij} S_i S_j + \frac{1}{2\beta} \sum_i \left[(1+S_i) \ln(1+S_i) + (1-S_i) \ln(1-S_i) \right].$$

  One sample in Figure 2 shows that the energy decreases along the simulation. However, I am not sure whether this is always true for the discrete simulation.

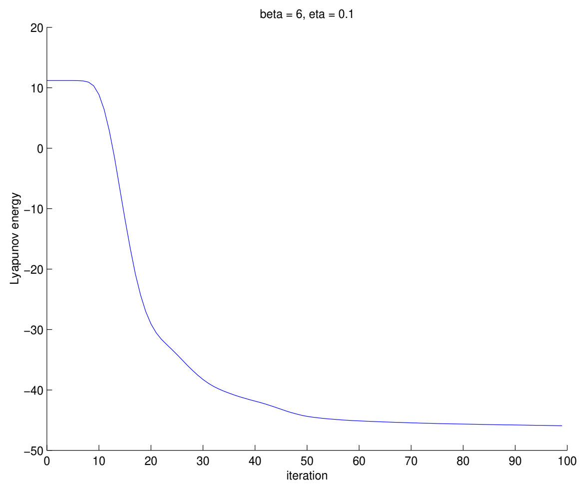


Figure 2: Lyapunov energy decreases along the simulation with random input. The weight is calculated from decorrelated patterns of 10 digits (see 4.1.5).

- **Definitions of *correct memory*.** From the definitions of correct memory and confused memory in the problem, $f(\vec{S})$ can be both correct and confused memory for one $\vec{S}$, when $\pm f(\vec{S}) = \vec{P}^k = f(\vec{P}^j)$ and $j \neq k$ (or strictly, $\vec{P}^j \neq \vec{P}^k$). I exclude such situations out of the definition of *correct memory* and define them as *confused memory*.

- ♠**How to 'rotate'?** In 4.1.5 and 4.1.6 we are asked to construct another network $\mathcal{N}^R$ with 'random' patterns R. The patterns of digits (P) are then 'rotated' to R by matrix T and then be memorized by $\mathcal{N}^R$. However, carrying out the rotation in the pattern level really has a problem: the rotated patterns may not be bounded between -1 and $+1$. One resolution is to 'saturate' the rotated pattern TR, which is changing all out-of-range values of TR to -1 or $+1$, according to their signs.

- ♠**Error bars.** For n random initial $\vec{S}$-states, we can get n memories $f(\vec{S})$ from simulation. The fraction of the volume of $\vec{S}$ space that leads to correct memory (denoted by f_r) is calculated by dividing the number of correct memories in $f(\vec{S})$ by n. The error bars in Figure (4) are standard deviations of the fractions, computed by $\sigma_r = \sqrt{f_r(1-f_r)/n}$. Since $0 \leq f_r \leq 1$, the standard deviation is bounded by

$$\sigma_r \leq \frac{1}{2\sqrt{n}}.$$

  The same rule applies to the fraction of $\vec{S}$ space that leads to confused memory and spurious memory.

- ♠**Cross-talk terms.** We have two types of weight, one with $W_{ii} \neq 0$, and the other with $W_{ii} = 0$. For stable state $\vec{S}^\nu$ in a high-gain network,

$$S_i^\nu = \operatorname{sgn}\left(\sum_{j=1}^{N} W_{ij} S_j^\nu\right).$$

Thus the essential condition of a state being stable is that

$$h_i^\nu = \sum_{j=1}^{N} W_{ij} S_j^\nu = \delta S_i^\nu (1 - C_i^\nu) \tag{1}$$

has the same sign with S_i^ν, or the cross-talk term $C_i^\nu < 1$ (here $\delta > 0$ is a constant). However, I was confused by the cross-talk terms defined in the class, since I found that C_i^ν can be defined in different ways.

– For the one that $W_{ii} \neq 0$,

$$W_{ij} = \frac{1}{N} \sum_{\mu} S_i^\mu S_j^\mu . \tag{2}$$

$$\begin{aligned} h_i^\nu = \sum_{j=1}^{N} W_{ij} S_j^\nu &= S_i^\nu \left(1 + \frac{1}{N} \sum_{\mu \neq \nu} S_i^\mu S_i^\nu \sum_{j=1}^{N} S_j^\mu S_j^\nu \right) \\ &= \frac{N+M-1}{N} S_i^\nu \left(1 + \frac{1}{N+M-1} \sum_{\mu \neq \nu} S_i^\mu S_i^\nu \sum_{j \neq i} S_j^\mu S_j^\nu \right) . \end{aligned}$$

So the cross-talk term can be defined as

$$C_i^\nu = -\frac{1}{N} \sum_{\mu \neq \nu} S_i^\mu S_i^\nu \sum_{j=1}^{N} S_j^\mu S_j^\nu \tag{3}$$

with mean

$$\overline{C_i^\nu} = -\frac{M-1}{N} .$$

This was taught in the class. However, the cross-talk term can also be defined as

$$\tilde{C}_i^\nu = -\frac{1}{N+M-1} \sum_{\mu \neq \nu} S_i^\mu S_i^\nu \sum_{j \neq i} S_j^\mu S_j^\nu \tag{4}$$

with zero mean.

– For the one with $W_{ii} = 0$, that is

$$W_{ij} = \begin{cases} 0, & \text{if } i = j; \\ \frac{1}{K} \sum_\mu S_i^\mu S_j^\mu , & \text{if } i \neq j. \end{cases} \tag{5}$$

When $M \neq N$,

$$\begin{aligned} h_i^\nu &= \frac{N-M}{K} S_i^\nu \left(1 + \frac{1}{N-M} \sum_{\mu \neq \nu} S_i^\mu S_i^\nu \sum_{j=1}^{N} S_j^\mu S_j^\nu \right) \\ &= \frac{N-1}{K} S_i^\nu \left(1 + \frac{1}{N-1} \sum_{\mu \neq \nu} S_i^\mu S_i^\nu \sum_{j \neq i} S_j^\mu S_j^\nu \right) . \end{aligned}$$

Now the cross-talk term is

$$\hat{C}_i^\nu = -\frac{1}{N-M}\sum_{\mu\neq\nu} S_i^\mu S_i^\nu \sum_{j=1}^{N} S_j^\mu S_j^\nu = \frac{N}{N-M} C_i^\nu \tag{6}$$

or

$$\tilde{\hat{C}}_i^\nu = \frac{1}{N-1}\sum_{\mu\neq\nu} S_i^\mu S_i^\nu \sum_{j\neq i} S_j^\mu S_j^\nu = \frac{N+M-1}{N-1}\tilde{C}_i^\nu. \tag{7}$$

So, which one should be used? Maybe we can analyze the stable states of the network in either way.

1. A small Matlab function calcultates the weight

```
function W = weight(p)
[N,M] = size(p);
W = (p*p'-M*eye(N))/N;
```

For different number of patterns, we can call `weight(pat(:,1:M))`.

Setting all the diagonal elements of W to zero will reduce the number of spurious stable states. We can find out this by presenting a random states $\vec{S}^\xi$ into the network:

(a) For weight (2), we have

$$\begin{aligned} h_i^\xi &= \sum_{j=1}^{N} W_{ij} S_j^\xi = \frac{M}{N} S_i^\xi + \sum_{j\neq i}\left(\frac{1}{N}\sum_\mu S_i^\mu S_j^\mu\right) S_j^\xi \\ &= \frac{M}{N} S_i^\xi \left(1 + \frac{1}{M}\sum_\mu S_i^\mu S_i^\xi \sum_{j\neq i} S_j^\mu S_j^\xi\right), \end{aligned}$$

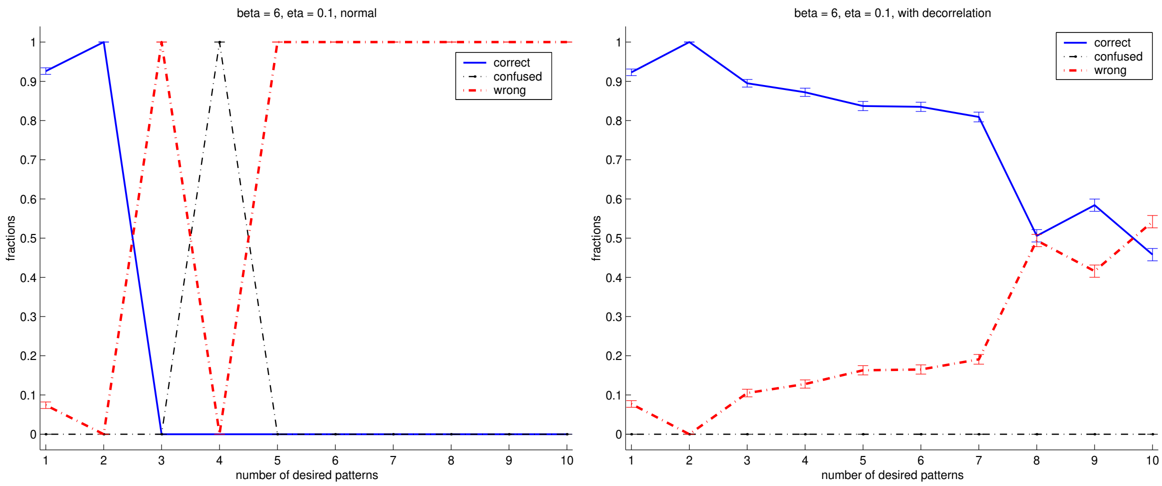
and the expectation of $\sum_\mu S_i^\mu S_i^\xi \sum_{j\neq i} S_j^\mu S_j^\xi$ is 0, and thus $h_i^\xi \approx \frac{M}{N} S_i^\xi$.

(b) For weight definition (5), we have

$$\begin{aligned} h_i^\xi &= \sum_{j=1}^{N} W_{ij} S_j^\xi = \sum_{j\neq i}\left(\frac{1}{K}\sum_\mu S_i^\mu S_j^\mu\right) S_j^\xi \\ &= \frac{1}{K} S_i^\xi \left(\sum_\mu S_i^\mu S_i^\xi \sum_{j\neq i} S_j^\mu S_j^\xi\right). \end{aligned}$$

The expectation of $\sum_\mu S_i^\mu S_i^\xi \sum_{j\neq i} S_j^\mu S_j^\xi$ is just 0.

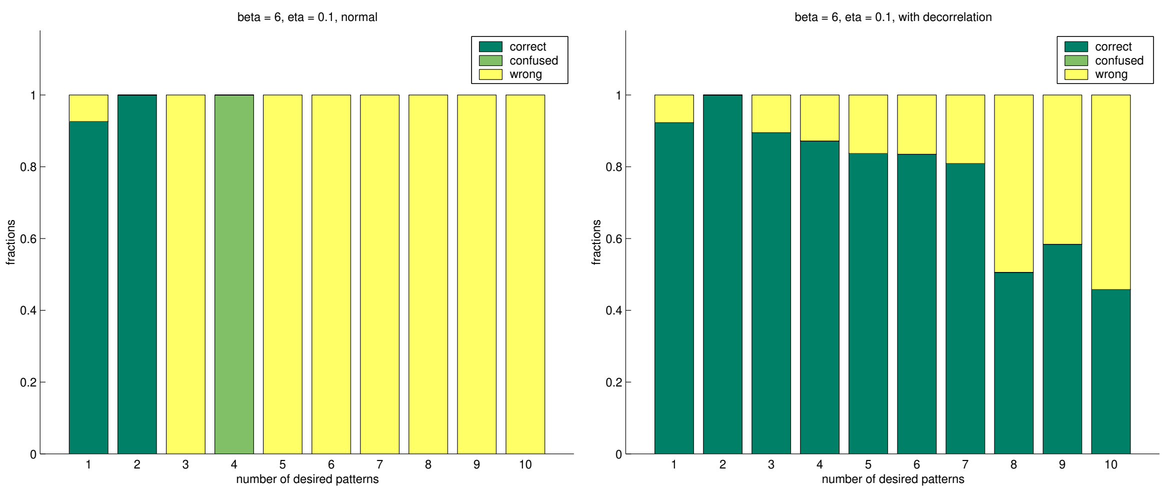
So setting all the diagonal elements of W to zero will reduce the self-feedback and thus reduce the number of spurious stable states. However, this may change the stability of desired patterns, since (6) says the cross-talk term of weight (5) is enlarged by a factor $\frac{N}{N-M}$. Thus for some C_i^ν in the range $(\frac{N-M}{N}, 1)$, $\hat{C}_i^\nu$ will be bigger than 1. Thus a stable 'point' S_i^ν when using W_{ij} as (2) becomes an unstable one when using (5). For the patterns of 10 digits, this happens to S_{74}^4 and S_{62}^0.



(a) normal

(b) after decorrelation

Figure 3: Fractions plotted with `errorbar()`.



(a) normal

(b) after decorrelation

Figure 4: Fractions plotted with `bar()`.

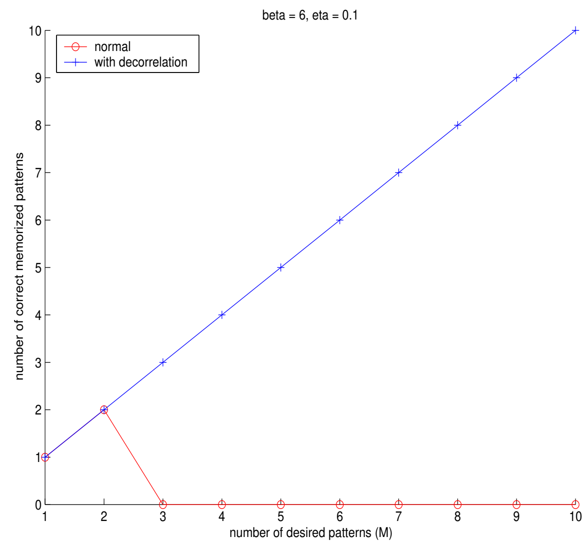


Figure 5: Number of desired patterns that are correct memories

2. With 1000 random patterns, we plot the fractions in Figure 3(a) and another version is in Figure 4(a) . The number of desired patterns that are correct memories are shown in Figure 5.

3. From $M = 3$, the network fails. The theoretical capacity of a 100 neuron network is

$$\lfloor 0.138N \rfloor = 13,$$

much bigger than 3. One of the spurious states when $M = 10$ is in Figure 6.

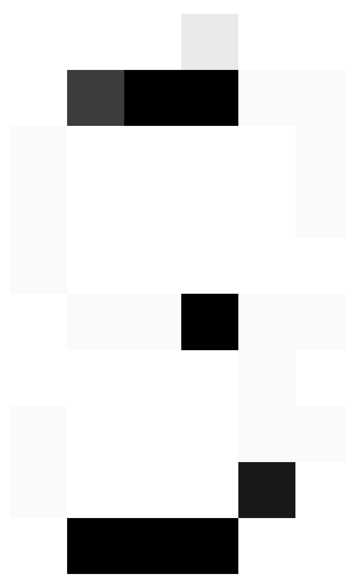


Figure 6: spurious state when $M = 10$ with random input pattern.

4. Inner product `Z=pat'*pat`; the cross-talk term can be written as

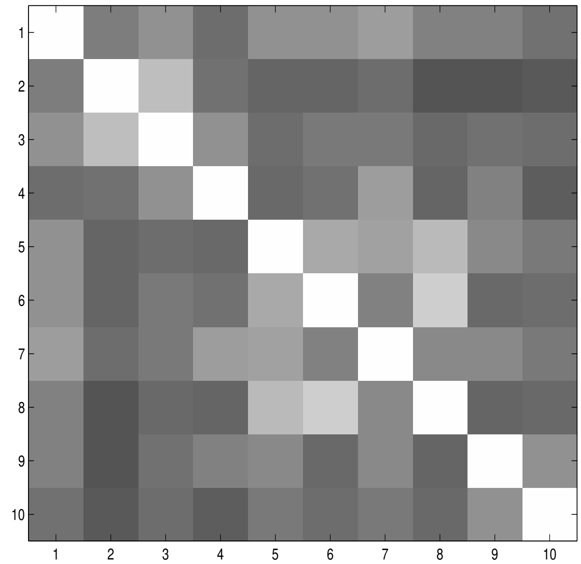


Figure 7: Inner product of desired patterns.`imagesc(pat'*pat, [0 100]); colormap('gray');`

$$C_i^\nu = -\frac{1}{N}\sum_{\mu\neq\nu} S_i^\mu S_i^\nu \sum_{j=1}^{N} S_j^\mu S_j^\nu = -\frac{1}{N}\sum_{\mu\neq\nu} S_i^\mu S_i^\nu Z_{\mu,\nu}.$$

So $Z_{\mu,\nu}$ reflects the overall difference between two patters.

5. `T=randpat(N,N]/[pat,randpat(N,N-M)];` $\hat{T} = T^{-1}$. The new Z is much close to N times the identity matrix I, which mean that the correlation between desired patterns is much weaker now.

6. The figures are in Figure 3(b), Figure 4(b), and Figure 5.

7. The largest number of correctly stored patterns of digits is $M = 10$. The procedure to test the degree of stablity of desired pattern S^ν is:

   - Convert S^ν to V^ν by `V=arctanh(S*0.995)/beta`;
   - Generate a zero mean unit variance Gaussian noise V^n by

   ```
   noise = randn(N, 1);
   noise = noise - repmat(mean(noise), N, 1);
   noise = noise ./ repmat(std(noise), N, 1);
   ```

   - Amplify or decay V^ν to make the ratio of noise to signal, which is calculated by $r_{ns} = -10\ln\sigma^2(V^\nu)$ with the unit dB, to a specified value.
   - Use $\tanh\beta(V^\nu + V^n)$ as the initial pattern for the network. This pattern will first be transformed by T to the network space and then be 'recognized'.
   - The result will transformed back to the pattern space, maybe with `round()` or `sign()`, and be compared to the original pattern V^ν.
   - Repeat these for enough times. Then we get the probability of a pattern being correctly retrieved under designated r_{ns}.

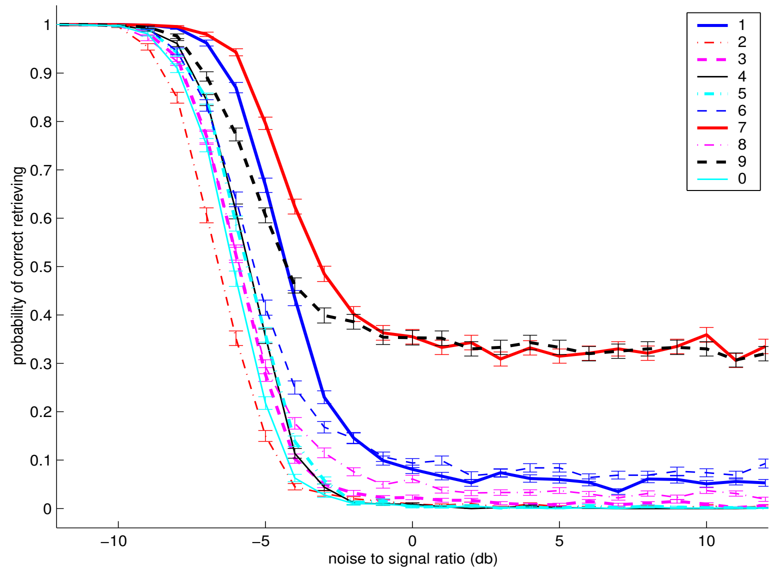


Figure 8: Probability of correct retrieving *v.s.* faintness of the signal.

We can plot the probability v.s. r_{ns} for each pattern of digit (Figure). However, 10 curves with error bars make the plot a mess. Here, by interpolation, the approximate ratio (in db) of noise to signal with specified possibility of correct retrieving is listed.

P_θ	1	2	3	4	5	6	7	8	9	0
0.9	-6.2	-8.4	-7.7	-7.4	-7.4	-7.5	-5.6	-7.8	-7.1	-7.9
0.5	-4.3	-6.6	-5.9	-5.5	-5.6	-5.4	-3.1	-5.9	-4.3	-6.1

8. First, assuming that we have a set of M orthogonal ±1 vectors of length N denoted by $\vec{R}^k$, $k = 1..M$.

   (a) Using $R = \left(\vec{R}^1, \vec{R}^1, \ldots, \vec{R}^M\right)$ as the random patterns R in 4.1.5, we get T and $\hat{T}$.

   (b) When the memorized vectors $\vec{P}^k$ are orthogonal, that is, $P^T P = NI$ or

   $$\sum_{j=1}^{N} P_j^\mu P_j^\nu = 0$$

   for $\mu \neq \nu$. From (3), we get $C_i^\nu = 0$; and for $M < N$, (6) gives $\hat{C}_i^\nu = 0$. We can also calculate that

   $$\tilde{C}_i^\nu = \frac{M-1}{N+M-1},$$

   and

   $$\tilde{\hat{C}}_i^\nu = \frac{M-1}{N-1}.$$

   (c) When $M = N$, P is a square matrix. The orthogonality of P's columns expessed by $PP^T = NI$ also hints the orthogonality of P's rows, which is $P^T P = NI$. Thus

   $$\sum_\mu P_i^\mu P_j^\mu = 0$$

when $i \neq j$. So the weight in (5) is always 0, which means the network cannot memorize any pattern. So, up to $N-1$ stored memories can be stable.

Below are some discussions about orthogonal binary vectors without proof.

- For $M > 2$ orthogonal vectors, N must be a multiple of 4. This can be proved by counting different signs between the first 3 vectors.
- When N is a multiple of 4, there may exist a Hadamard matrix of dimension $N \times N$. Thus we will have at least N orthogonal vectors.
- If N can be written as sum of several multiples of 4, i.e.,

$$N = 4N_1 + 4N_2 + \cdots + 4N_l$$

  and $1 \leq N_i \leq 7$, we can always find $4\min\{N_1, N_2, \ldots, N_l\}$ orthogonal vectors with length N, by concatenating columns of Hadamard matrix of dimension $(4N_i) \times (4N_i)$.
- You can find some resources about the Hadamard matrix at

  `http://www.research.att.com/~njas/hadamard/index.html`

4.2 Winner-Take-All Networks and Nearest-Neighbor Memories

1. For high-gain, S_i is either "on" (+1) or "off" (−1). Suppose there are m units "on". Thus

$$\sum_{i=1}^{M} S_i = -(M-m) + m = 2m - M \tag{8}$$

   and facilitate $W_{ij} = -1$ for $i \neq j$,

$$\begin{aligned} \mathcal{L} &= -\frac{1}{2}\sum_{i,j} W_{ij} S_i S_j - \sum_i S_i I_i \\ &= -\frac{1}{2}\left[-\left(\sum_i S_i\right)^2 + \sum_i S_i^2\right] - \alpha \sum_i S_i \\ &= 2\left(m - \frac{M+\alpha}{2}\right)^2 - \frac{M+\alpha^2}{2}. \end{aligned}$$

   So when

$$\frac{1}{2} < \frac{M+\alpha}{2} < \frac{3}{2}$$

   i.e.,

$$1 - M < \alpha < 3 - M, \tag{9}$$

   the Lyapunov function will get its minimum at $m = 1$.

2. Also consider the high-gain situation. Suppose at some time t_0, the network has m units "on". Use (8),

$$\dot{V}_i = -V_i + \sum_j W_{ij} S_j + \alpha = -V_i + S_i - \sum_j S_j + \alpha = -V_i + S_i + (\alpha + M - 2m).$$

During some period of time, the S state will not change while V state of the network does change. That is

$$V_i(t_0 + \Delta t) = e^{-\Delta t} V_i(t_0) + \left(1 - e^{-\Delta t}\right) F_i, \tag{10}$$

where $F_i = S_i + (\alpha + M - 2m)$ is constant during that period. With time elapses, V_i will get closer and closer to F_i, until some S_i changes.

- However, the order among V_i's will not change. That is, for $(t_0 + \Delta t)$ still in that period,

  $$V_i(t_0) < V_j(t_0) \Rightarrow V_i(t_0 + \Delta t) < V_j(t_0 + \Delta t).$$

  This is easy to be proved from (10) and $V_i < V_j \Rightarrow S_i \le S_j \Rightarrow F_i \le F_j$.
- When $m \ge 2$, i.e., there're more than 1 unit "on" at time t_0. Thus from (9),

  $$F_i \le 1 + \alpha + M - 4 < 0.$$

  So all the "off" units at time t_0 will still be "off", and all the "on" units will have their V_i's decreased and until one of them first changes to "off". That makes m decrease.
- When $m = 0$, i.e., all units are "off". We have

  $$F_i \ge -1 + \alpha + M > 0.$$

  Thus similarly with the situation that $m \ge 2$, all V_i will increase until one of them turns to be "on".

Considering together with the Lyapunov function, the state of the network will stabilize when only one unit is "on" and, since the order of V_i will not change during time, that "on" unit is the one with maximum V_i at the initial state. Thus

$$V_k \text{ in inital state } \vec{V} \text{ is the maximum} \Rightarrow \vec{S}^1 \text{ is the fixed point.}$$

3. For low-gain $\beta < 1$,

   $$S_i = \tanh(\beta V_i) \approx \beta V_i$$

   Thus

   $$\dot{V}_i = -V_i + \sum_{j=1}^{M} W_{ij} S_j + I_i + \sum_{j=1}^{N} T_{ij} P_j^{\text{in}} = -(1-\beta)V_i - \beta \sum_{j=1}^{M} V_j + H_i$$

   where $H_i = I_i + \sum_{j=1}^{N} T_{ij} P_j^{\text{in}}$. The equilibrium V_i requires for all i,

   $$(1-\beta)V_i = -\beta \sum_{j=1}^{M} V_j + H_i. \tag{11}$$

   Summing (11) for $i = 1..M$, we get

   $$(1-\beta) \sum_{i=1}^{M} V_i = -M\beta \sum_{j=1}^{M} V_j + \sum_{i=1}^{M} H_i$$

   That is

   $$\sum_{i=1}^{M} V_i = \sum_{j=1}^{M} V_j = \frac{1}{1 + (M-1)\beta} \sum_{j=1}^{M} H_j. \tag{12}$$

Plugging (12) back to (11), we get

$$V_i = \frac{1}{1-\beta}\left[H_i - \frac{\beta}{1+(M-1)\beta}\sum_{j=1}^{M} H_j\right]. \tag{13}$$

We can also use matrix form to solve this problem

$$V = \beta W V + I^{\text{in}} + TP^{\text{in}} \Rightarrow V = (I - \beta W)^{-1}(I^{\text{in}} + TP^{\text{in}})$$

4. From (13), $\sum_{j=1}^{M} H_j$ is a constant for all V_i, thus the most active unit is the one with maximum H_i. So we should design T such that $T\vec{P}^k$ has its maximum component at k^{th} position. To make the thing simple, we can use $T = P^T$. Thus the i^{th} component of $T\vec{P}^k$, $\sum_{j=1}^{N} T_{ij}P_j^k$, is $N - 2d\left(\vec{P}^i, \vec{P}^k\right)$ where $d\left(\vec{P}^i, \vec{P}^k\right)$ is the Hamming distance between $\vec{P}^i$ and $\vec{P}^k$. For P consists of distinct patterns, T apparently satisfies the requirement. So with $\vec{P}^k$ as input, H_k is the maximum among all H_i's and V_k will be the most active unit.

5. When the gain is suddenly turned from very low to very high, and simultaneously the external input P^{in} is inactive, S_i will change as described in 4.2.2. The most active unit at low-gain will be "on" and all other units will be "off". For situation in 4.2.4, S_k will be $+1$.

6. For input pattern $P^{\text{in}} = \vec{P}^k$, output $\vec{U} = (U_1, U_2, \ldots, U_N)^T$

$$\dot{\vec{U}} = -\vec{U} + \hat{T}\vec{S}^k$$

Output is $\vec{U} = \hat{T}\vec{S}^k$. So we need to make $\hat{T}\vec{S}^k = \vec{P}^k$. Thus

$$\hat{T} = \left(\vec{P}^1, \ldots, \vec{P}^M\right)\left(\vec{S}^1, \ldots, \vec{S}^M\right)^{-1}$$

In Matlab, we can use `pat*inv(2*eye(M)-1)` to get $\hat{T}$.

7. From $T = P^T$ and the relation between $\sum_{j=1}^{N} T_{ij}P_j^{\text{in}}$ and Hamming distance $d\left(\vec{T}^i, P_j^{\text{in}}\right)$, the network's behavior is to retrieve the memory that hat the minimum distance to the input pattern. Thus, it can be called "nearest neighbor memory". Similar to recognizing a person by the how she/he resembling to her/his grandmother, it has another name "grandmother cells". (why not grandfather?)

   Compared to the network used in 4.1, the "nearest neighbor memory" has no spurious states, and its behavior seems simpler. The weights in the "nearest neighbor memory" do not carry any information; the information of memorized patterns are stored in the input/output layers. Howeverm weights in an associative memory do carry information.

   If some wieghts are damaged in both networks, the "nearest neighbor memory" may not behavior as WTA. Two or more 'winners' can simutaneously exist and thus the output may be the combination of several patterns. These patterns should all be similar to the input pattern, relatively. As to the associative memory, some parts of the desired memories will be destroyed thus the retrieved patterns may not be fully same as its original.

   It seems that M can be very big, even be the number of all patterns: $M = 2^N$. To memorize M patterns with MN bits, M^2 weights are used. So the number of weights used per bit memorized is M/N. For the network in 4.1, no matter how many patters are memorized, N^2 weights are used. So the ratio there is N/M, which is usually bigger than M/N, since no more that $N-1$ memories can be simultaneously memorized by that network. All weights in the input/output layer haven't been taken into consideration.

4.3 Translation-Invariance by Fourier Transforms

1. When f_x, f_y, d_x and d_y are integers, we get $e^{\frac{2\pi}{N}ixf_x}$ and $e^{\frac{2\pi}{N}iyf_y}$ are functions with cycle N. Thus

$$\begin{aligned}
F_b(f_x, f_y) &= \frac{1}{N}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1} a(x+d_x, y+d_y)e^{\frac{2\pi}{N}ixf_x}e^{\frac{2\pi}{N}iyf_y} \\
&= \frac{1}{N}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1} a(x, y)e^{\frac{2\pi}{N}i(x-d_x)f_x}e^{\frac{2\pi}{N}i(y-d_y)f_y} \\
&= e^{-\frac{2\pi}{N}i(d_xf_x+d_yf_y)}F_a(f_x, f_y).
\end{aligned}$$

So we get

$$\|F_a(f_x, f_y)\| = \|F_b(f_x, f_y)\|\,.$$

2. It seems that the DFT can deal with translation of patterns. So, first, we should relate the desired pattern to their DFT transformation. Second, the original patterns are binary. However, the transformed patterns are no longer binary. So we have to do another transformation to make $\|F_a(f_x, f_y)\|$ be changed to a binary pattern.