Neural Networks: Advanced

Dokuz Eylül University
Computer Engineering Department

25 September 2000 – 12 January 2001

□ Contents:

□ What is Neural Network?

A system with *neurons* connected with *synapses*,

where

- synapse:

  transmits input state value (usually either one or zero) with its weight value being multiplied to the input; and

- neuron:

  takes the weighted sum of fan-in inputs, and outputs its function value of the specified *transfer function* $f(x)$.

Typically categorized into two forms:

1. Layered (feed forward)
2. Fully-connected

Keywords:

- **neuron, synapse, weight, transfer-function, layered, feed-forward, fully-connected**

□ Perceptron:

An example of single-layered feed forward neural network.

Suppose:

- M inputs: $x_1, \cdots x_M$;
- N outputs: $y_1, \cdots y_N$;
- each N output neurons are connected with all M input neurons;
- weight values (of the synapse) from (input) neuron j to (output) neuron k is w_{jk};

then

$$y_i = f(\sum_{k=1}^{N} w_{ki} x_k), \tag{1}$$

where

$$f(x) = \mathrm{sgn}(x) \quad \text{or} \quad f(x) = \tan^{-1}(x). \tag{2}$$

Keywords:

- **perceptron**

□ The simplest example.

1. AND with two inputs and one output

   e.g.,

   * $w_{00} = w_{01} = 0.7$

2. How about OR?

   e.g.,

   * $w_{00} = w_{01} = 1.4$

3. How about XOR?

   – Impossible by single layer perceptron.

Keywords:

- **synaptic plasticity**

□ Perceptron with hidden layers

Adding intermediate layer(s) called *hidden layer(s)* between input and output.

- Suppose K hidden layers.
- weight from neuron i of the k-th layer to neuron j of the next layer is denoted as w_{ij}^k.
- XOR revisited

  e.g.,

  * $w_{00}^0 = w_{01}^0 = 0.7, \quad w_{10}^0 = w_{11}^0 = 1.4,$

    $w_{10}^1 = -1.4, \; w_{11}^1 = 1.4.$
- Separability
  - If we can separate the solution space by hyperplane,

    (To be more specific, we can separate 2D solution space of 2-input AND/OR by a straight line, but not for XOR.)

    then we can solve the problem by single perceptron.
  - Multi-layer allows us to separate the space by more flexible surface, not necessarily the plane.

Keywords:

- **hidden layer, separability, solution-space**

□ Back Propagation Algorithm

Definition of variables:

- L layers of neurons.
- k_r neurons in the r-th layer. $\quad r = 1, 2, \cdots, L.$
- l input neurons ($k_0 = l$).
- w^r_{jk}: weight to the j-th neuron in the r-th layer from the k-th neuron in the previous layer.
- N training pairs: $(\mathbf{x}(i), \mathbf{y}(i)), \quad i = 1, 2, \cdots, N.$
- $y^r_k(i)$: the output of the k-th neuron in the r-th layer when the i-th training sample is given.
- fan-in's to j-th neuron in the r-th layer when the i-th training sample is given.

  $$h^r_j(i) \equiv \sum_{k=1}^{k_{r-1}} w^r_{jk} y^{r-1}_k(i) + w^r_{j0} = \sum_{k=0}^{k_{r-1}} w^r_{jk} y^{r-1}_k(i),$$

  where $y^{r-1}_0 = 1$ and

  $$(y^0_1(i), \cdots, y^0_l(i)) = (x_1(i), \cdots, x_l(i)) = \mathbf{x}(i).$$

- Each output neuron's error:

  $$e_m(i) \equiv y_m(i) - \hat{y}_m(i), \quad i = 1, 2, \cdots, N.$$

  where $(\hat{y}_1(i), \cdots, \hat{y}_{k_L}(i)) = \mathbf{y}(i)$

- Error function to be minimized:

  $$D = \frac{1}{2} \sum_{i=1}^{N} \sum_{m=1}^{k_L} e_m(i)^2$$

The algorithm:

1. *Initialization:* Initialize all the weights with small random values.

2. *Forward computations:* For each of the training vectors

   $$\mathbf{x}(i), \quad i = 1, 2, \cdots, N$$

   compute all the

   $$y_j^r(i) = f(h_j^r(i))$$

   for $j = 1, 2, \cdots, k_r$ and $r = 1, 2, \cdots, L$.

3. Backward computations: For each $i = 1, 2, \cdots, N$, and $j = 1, 2, \cdots, k_L$ compute

   $$\delta_j^L(i) = e_j(i) f'(h_j^L(i))$$

   then compute

   $$\delta_j^{r-1}(i) = e_j^{r-1}(i) f'(h_j^{r-1}(i))$$

   for $r = L, L-1, \cdots, 2$ and $j = 1, 2, \cdots, k_r$ where

   $$e_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r$$

4. Update the weights for $r = 1, 2, \cdots, L$ and $j = 1, 2, \cdots, k_r$

   $$w_{jk}^r\,[\text{new}] = w_{jk}^r\,[\text{old}] + \triangle w_{jk}^r,$$

   where

   $$\triangle w_{jk}^r = -\epsilon \sum_{i=1}^{N} \delta_j^r(i) y_k^{r-1}(i).$$

Rationale:

From the *gradient descent*'s principle,

$$\triangle w_{jk}^r = -\epsilon \frac{\partial D}{\partial w_{jk}^r} \tag{3}$$

where

$$D = \sum_{i=1}^{N} \varepsilon(i) \tag{4}$$

and

$$\begin{aligned} \varepsilon(i) &= \frac{1}{2} \sum_{i=1}^{N} \sum_{m=1}^{k_L} e_m(i)^2 && (5) \\ &= \frac{1}{2} \sum_{i=1}^{N} \sum_{m=1}^{k_L} (y_m(i) - \hat{y}_m(i))^2. && (6) \end{aligned}$$

Hence

$$\frac{\partial \varepsilon(i)}{\partial w_{jk}^r} = \frac{\partial \varepsilon(i)}{\partial h_j^r} \frac{\partial h_j^r}{\partial w_{jk}^r}. \tag{7}$$

Here we define

$$\frac{\partial \varepsilon(i)}{\partial h_j^r} \equiv \delta_j^r(i). \tag{8}$$

Then

$$\triangle w_{jk}^r = -\epsilon \sum_{i=1}^{r} (i) \delta_j^r(i) y_k^{r-1}(i). \tag{9}$$

□ Genetic Algorithm (GA)

An example:

- Neural network with 100 input neurons and 10 output neurons with no hidden layer:
  - hand-written-characters of the figures from zero to nine drawn on the 10x10 pixels are to be recognized
  - by activating only the i-th neuron of the output remaining other neurons not activated.

From a biological analogy, the followings are defined.

Chromosome:
Each chromosome is comprised of 1000 *genes*, representing 1000 weights of one *individual* neural network.

Population:
A number of individuals, say 100, create a population.

Initialization:
All genes of all chromosomes are initialized at random, taking a value ranging from zero to one. This construct the first *generation*.

(cont'd)

Fitness:

The ability of each individual is evaluated and assigned its fitness value. In our current problem, for example, each individual given a set of weights is evaluated by giving each output neuron 0.1 if the output is correct and 0 otherwise. Note that in the first generation, every individual performs rather badly but diferrently, since weights are determined at random.

Selection:

Two parents are selected to produce one offspring. Individuals of better performance should be more likely to be selected. For example, (1) selection is always from upper half of the population in which individuals have been sorted with an order of their fitness, or (2) selection is made sot that the probability is proportional to the fitness value. The production of an offspring is by employing the following two operations: *crossover* and *mutation*

Crossover:

A same position of the two parent chromosomes are determined at random (each time when they are selected), and cross them over.

Mutation:

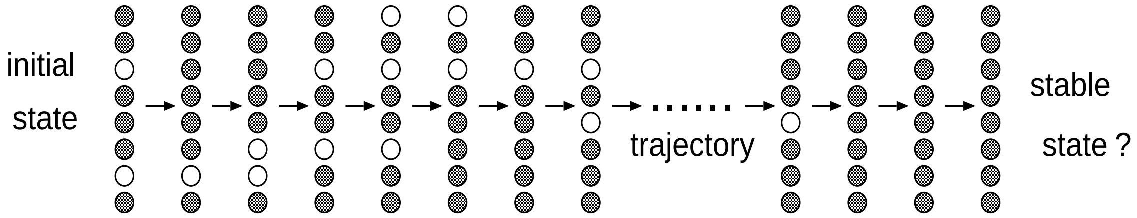
For each gene of the offspring chromosome, produce one integer random number from 0 to 99. If the value is zero, then the gene is replaced with the other weight value also created at random each time.

Algorithm:

1. *Initialization:* Initialize all the weight values of all the individuals in the first generation at random so that it lies on $[0, 1]$.
2. *Evaluation:* Evaluate fitness of all the individuals of the population.
3. *Reproduction:*
   (1) Select two parents from the population.
   (2) Crossover the two parents and produce one offspring.
   (3) Mutate the offspring.
   (4) Repeat (1) to (3) above until new generation of same size is obtained.
4. Repeat 2. to 3. above until the perfect solution is obtained or prespecified generations is reached.

□ Associative Memory as a Dynamical System

associative memory = dynamical system

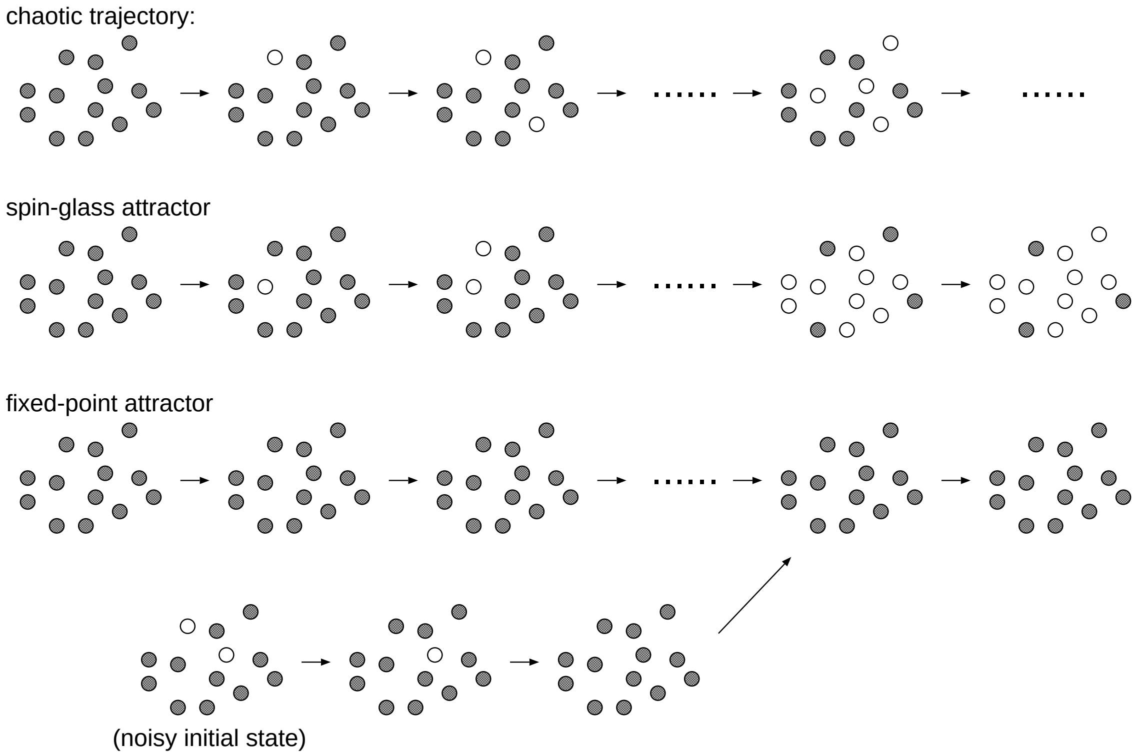


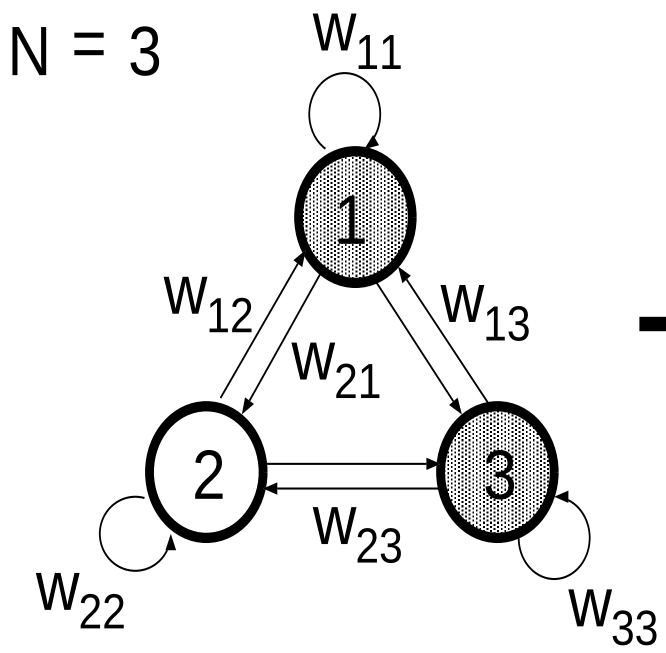
converge

initial state $\longrightarrow$ equilibrium state

u+du u

□ A category of trajectories





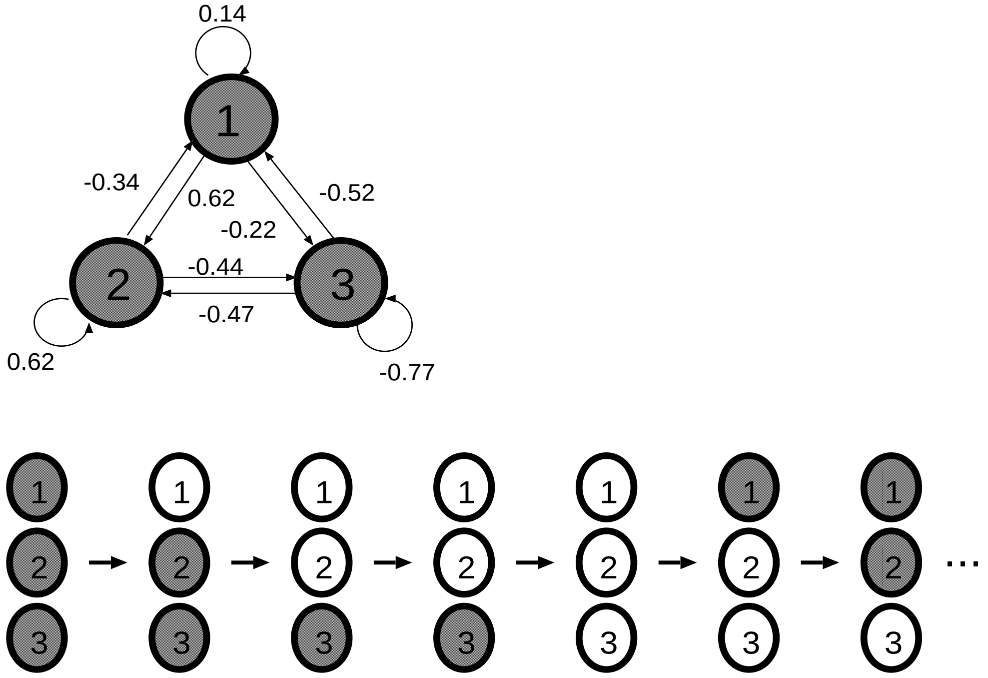
weight matrix

$$\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{pmatrix}$$

□ State Transition

$$s_i(t+1) = sgn\left(\sum_{j\neq i}^{N} w_{ij} \cdot s_j(t)\right).$$

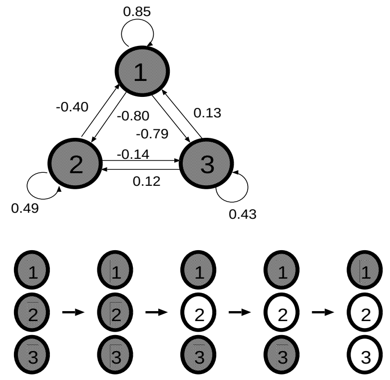
Example 1



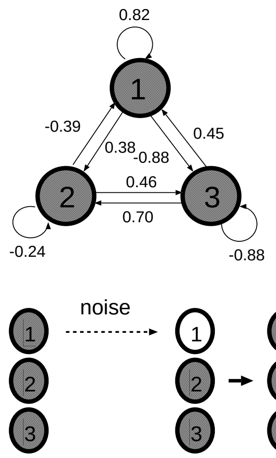
An example of state transition in the above toy example:

$$
\begin{aligned}
s_1(1) &= sgn\,(\, w_{11} \cdot s_1(0) + w_{12} \cdot s_2(0) + w_{13} \cdot s_3(0)\,) \\
&= sgn\,(\, 0.14 \cdot 1 + (-0.34) \cdot 1 + (-0.52) \cdot 1\,) \\
&= -1
\end{aligned}
$$

Example 2



□ Example 3



□ Hebbian Weights

$$w_{ij} = \frac{1}{N} \sum_{\nu=1}^{p} \xi_i^\nu \xi_j^\nu \ (i \neq j), \ w_{ii} = 0.$$

- In the above toy example:

$$\begin{pmatrix} 0 & 0.11 & 0.11 \\ 0.11 & 0 & 0.11 \\ 0.11 & 0.11 & 0 \end{pmatrix}$$

- An interpretation of Hebb's rule:
  - A strong weight affects neurons at both ends to be same states.
- Many other solutions of a *weight configuration* to store a same set of patterns exist.
  - Each
    - can *recall* from
      - partial information;
      - incorrect information
    - but has a different
      - *basin of attraction*;
      - *storage capacity*.

□ The Other Topics of Associative Memory

- Memory of a set of Motion Pictures

$$w_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \xi_i^{\mu+1} \xi_j^{\mu}. \tag{10}$$

- How to forget already stored patterns?

$$w_{ij} = w_{ij} - \frac{\lambda}{N} \xi_i \xi_j. \tag{11}$$

e.g., repeat equation(11) 1000 times with $\lambda = 0.01$.

- Storage capacity (How many patterns can be stored?)

· Under Hebb's rule

$$p/N < 0.138 \tag{12}$$

· E. Gardner asserts (1988).

$$p/N < 2 \tag{13}$$

In order to obtain $p/N \approx 2$:

· ξ_i^μ and $\Sigma_j w_{ij}\xi_j^\mu$ should be a same sign for all $\mu = 1, \cdots, p$ (then all stored memories are stable.)

· So,

$$\xi_i^\mu \cdot \sum_j w_{ij}\xi_j^\mu > 0. \tag{14}$$

or more preferably

$$\xi_i^\mu \cdot \sum_j w_{ij}\xi_j^\mu > (\kappa > 0). \tag{15}$$

(the bigger κ, the more stable)

· Algorithm for obtaining $p/N \approx 2$:

IF $\Sigma_j w_{ij}\xi_j^\mu \neq 1$ for $\forall\mu$ $\quad(\kappa = 1)$
THEN $w_{ij} = w_{ij} + \delta w_{ij}$

where

$$\delta w_{ij} = \frac{1}{N}\xi_i^\mu\xi_j^\mu(1 - \delta_{ij}) \cdot \theta(\kappa - \gamma_i^\mu) \tag{16}$$

$$\gamma_i^\mu = \frac{\Sigma_j w_{ij}\xi_i^\mu\xi_j^\mu}{\Sigma_j(w_{ij})^2} \tag{17}$$

- How can we observe phenomena?

1) By displaying $S_i(t)$ at every update time $t = n\triangle t \quad (n = 1, 2, \cdots)$ on a PC screen.
2) By plotting Hamming distances from initial input.
3) By plotting a similarity measure of $S_i(t)$ to the initial input.

$$m^\mu(t) = \frac{1}{N} \sum_{i=1}^{N} \xi_i^\mu S_i^\mu(t). \tag{18}$$

Try each of the above three methods

· Fixed Points

starting with one of stored patterns that is given some noises (1);

· Convergence to Stored Pattern

starting with non-noisy one of stored patterns (1)-(3);

· Chaotic Trajectory

starting with random initial state under symmetric random weights (2) or (3);

· Trajectory to Spingrass Atractor

same as above but under asymmetric random weights (2) or (3);

· Motion Picture

– observing equation (refeq:motion) (2) or (3).

- How appropriate a configuraition of weights is?

$$f = \sum_{\mu=1}^{p} \sum_{t=1}^{t_0} m^\mu(t) \tag{19}$$

□ Home work

- Using fully connected neural networks
  - with 200 neuronsl;
  - with p random[1] patterns stored by Hebb's rule;
  - with equal probability of 1 and –1 in patterns.[2]

(1) Study capcity dependency on the number of patterns stored.

e.g., f (y-axis) vs. p (x-axis)

(2) Study size of basin of attraction's dependency on the number of patterns stored.

e.g., f (y-axis) vs. #-of-noises (x-axis) with parameter p.

[1]To produce integer random number, e.g.,

```
my_random_number(int n)
{
r1=(double)rand();
r2=(double)n;
r=(int)((r1/RAD_MAX)*r2);
return(r);
}
```

[2]We use random binary patterns, but reproducibility of these random patterns is preferable. So, for example:

```
# define SEED
...
randomize()
{ srand(SEED); }
...
main()
{
...
randomize();
generate_pattenrs();
...
}
```

Chapter 3

Solving Combinatorial Problem by Neural Networks

□ Solving Combinatorial Problem

- The fully connected neural network model so far we've studied can also solve combinatorial problems.

  e.g.

  (1) Multiflop problem:

  - ∗ Only one out of n neurons is on with all the others off at a moment.
  - ∗ When all weight values are -2,
    - · a unit that is set to 1 inhibits the other units; or
    - · if the network is started with all neurons set to zero, then the excitation of every neuron is zero, which makes the first neuron to be asynchronously selected flip its state to 1 and no other neuron can change its state after this first neuron has been set to 1

  (2) The eight rooks problem:

  - ∗ A generalization of multiflop;
  - ∗ If the connection of each neuron to other neurons in the same row or column have the weight -2, all others have a weight zero and all neurons have the threshold -1, then
    - · Any neuron set to 1 inhibits any other neurons in the same row or column; or
    - · if a row or column is all set to 0 when one of its neuron is selected it will immediately switch its state to 1

(3) The eight queens problem:

* A generalization of eight rooks problem,
* taking diagonal lines into consideration too.
* This, however, does not always work due to many *local minima.* surrounding the solution

  $\Downarrow$

  using stocastic (probabilistic) neurons

□ A new scheme.

- Stochastic Neurons:

  Instead of using **deterministic neurons** where

$$S_i(t+1) = sgn(\sum_j^N w_{ij} S_j(t)) \tag{20}$$

  stochastic neurons where

$$Pr(S_i(t+1)) = f(\sum_j^N w_{ij} S_j(t)) \tag{21}$$

  are employed, in which

$$f(x) = \frac{1}{1 + \exp(-2\frac{x}{T})} \tag{22}$$

  and T is called *temperature*.

- Energy:

  A quantity called *energy* is defined corresponding to a configuration of the states of neurons as

$$E(t) = -\frac{1}{2} \sum_{ij}^{i \neq j} w_{ij} S_i(t) S_j(t)) \tag{23}$$

  and network state changes so that $E(t)$ minimizes.

Definition 1 (Boltzman machine) *A Boltzman machine is a Hopfield network composed of n neurons with states $x_1 \cdots x_n$. The state of neuron i is updated asynchronously according to the rule*

$$x_i = \begin{cases} 1 & \textit{with probability} \quad p_i \\ 0 & \textit{with probability} \quad 1 - p_i \end{cases} \tag{24}$$

where

$$p_i = \frac{1}{1 + \exp\{(-\sum\limits_{j=1}^{n} w_{ij} x_j - \theta_i)/T\}} \tag{25}$$

and T is a positive temperature constant and θ_i is the threshold of the neurons. Then the energy is defined as:

$$E = -1/2 \sum_i \sum_j w_{ij} x_i x_j + \sum_i \theta_i x_i \tag{26}$$

Note that if T is very small $p_i \approx 1$ for $\sum\limits_i w_{ij} x_j - \theta_i$ is positive and $p_i \approx 0$ for negative.

- Examples
  - the smallest hopfield network — the flip-flop

    two neurons with

$$w_{12} = w_{21} = -1, \quad w_{11} = w_{22} = 0,$$
$$\theta_i = -0.5$$
$$\Downarrow$$
$$E_{00} = E_{11} = 0, \quad E_{01} = E_{10} = -0.5$$

– the multi-flop

$$\begin{aligned} E &= (\sum_{i=1}^{n} x_i - 1)^2 \\ &= \sum_{i=1}^{n} x_i^2 + \sum_{i \neq j} x_i x_j - 2 \sum_{i=1}^{n} x_i + 1 \\ &= \sum_{i \neq j} x_i x_j - \sum_{i=1}^{n} x_i + 1 \\ &= \frac{1}{2} \sum_{i \neq j} (-2) x_i x_j + \sum_{i=1}^{n} (-1) x_i + 1 \end{aligned}$$

– The eight rooks problem

$$\begin{aligned} E_1 &= \sum_j (\sum_i x_{ij-1})^2 \\ E_2 &= \sum_i (\sum_j x_{ij-1})^2 \\ E &= E1 + E2 \end{aligned}$$

□ Traveler's Salesman Problem (TSP)[3]

– as an NP-complete[4] combinatorial problem

∗ A salesman should visit every city once but only once such that his traveling route is the shortest[5].

- Representation of a rout by a $N \times N$ matrix

⋆ The rout 1-3-6-4-5-2 of 6-city TSP is represented as

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{28}$$

where "$i\alpha$-element is one" means that the i-th city is in the α-th location in the tour.

- Implementation by a fully connected neural network with N neurons.
  - $S_{i\alpha}(t)$ corresponds to the $i\alpha$-element of the matrix.
  - Starting with a random configuration of the network state, it is expected to converge to the matrix corresponding to the shortest route.

[3] We can apply the problem to "how we connect each device in LSI in which many, say 10^5, divices are included."

[4] Nondeterministic polynomial

[5] With N cities, the number of all the possible route is

$$(N-1)! = \sqrt{2\pi N}\exp(N\ln N - N) \tag{27}$$

i.e., *not deterministic in polynomial time.*

Then the energy of our neural network is

$$E = -\frac{1}{2} \sum_{i,k}^{i \neq k} \sum_{\alpha,\beta}^{\alpha \neq \beta} w_{i\alpha,k\beta} S_{i\alpha} S_{k\beta} \tag{29}$$

This can be expanded with appropriate A, B, and C as

$$\begin{aligned} E = & - \frac{1}{2} \sum_{i,k,\alpha}^{i \neq k} d_{ik} S_{i\alpha} (S_{k(\alpha-1)} + S_{k(\alpha+1)}) \\ & - \frac{A}{2} \sum_{i,\alpha,\beta}^{\alpha \neq \beta} S_{i\alpha} S_{i\beta} - \frac{B}{2} \sum_{i,k,\alpha,}^{i \neq k} S_{i\alpha} S_{k\alpha} \\ & - \frac{C}{2} \sum_{i,\alpha} (S_{i\alpha} S_N)^2 \qquad (30) \\ & \qquad (31) \end{aligned}$$

Then

- ⋆ 1st term means total length of the tour;
- ⋆ 2nd term vanishes iff the raw includes only one “1”;
- ⋆ 3rd term vanishes iff the column includes only one “1”;
- ⋆ 4th term takes its minimum when exactly N neurons are active.