Chapter 7

Neuronic Equations

1 Introduction

The synthesis problem of Boolean network is, by giving a series of patterns, to make the net transits in that ordered sequence. Many works in neural network theory in the past concerned mainly about nets whose elements related only to the state of the last time interval. In other words, a state is function of the last moment only, i.e.,

$$\mathbf{x}(t+1) = F\,[\,\mathbf{x}(t)\,] \qquad \ldots(1)$$

But in general, neuronic equations can be formed like:

$$x_j(t+\tau) = \sigma\left[\sum_{r=0}^{L}\sum_{i=1}^{N} w_{ji}^{(r)} x_i(t-r\tau) - S_j\right] \qquad \ldots(2)$$

where *i* means the *i*-th neuron, τ is the time interval, and r is the time index.
It means that the states of a nets should be described generally, related to its past, not only one but up to L times iterations before. So the first one (1) is just a special case of (2) with L = 0. But it has been showed that (not provided here), we can always reduce the general case (L ≠ 0) to the first one (L = 0) by enlarging the number of neurons.

Another significant aspect is that any neuronic equation (NE) of the form proposed by E.R. Caianiello in 1961 can be completely *linearized* in tensor space.

By the above two, the Boolean function neural nets discussed below are indeed general case. All the properties of those NEs covered above should also be found in nets of the simpler ones (L = 0 and linearized).

2 Linearized in tensor space

For a net of N elements, the state of the net is:

$\mathbf{x} = (x_1, x_2, \ldots, x_N)$ $\qquad x_i = \pm 1.$

Assume for each neuron *i*, the function is:

$$\mathbf{x}_{i,t+1} = \sigma\,[\,g_i(\mathbf{x}_t)\,] \qquad \ldots(3)$$

or $\mathbf{x}_{t+1} = \sigma\,[\,\boldsymbol{g}(\mathbf{x}_t)\,]$ in vector form. Here, g_i is any real function such that $g_i \neq 0$ for

any input. σ is the hard-limiting activation function ($\sigma(g) = 1$ if $g > 0$ and $\sigma(g) = -1$ if $g < 0$).

The vector **x** (N dimension) can be rewritten in tensor power space, which is now a 2^N dimensional vector. One convenient definition is:

$$\eta = \begin{pmatrix} \eta_0 \\ \eta_0 \\ \vdots \\ \eta_\alpha \\ \vdots \\ \eta_{2^N} \end{pmatrix} \equiv \left(\frac{1}{\sqrt{2}}\right)^N \underbrace{\begin{pmatrix} 1 \\ x_N \end{pmatrix} \times \begin{pmatrix} 1 \\ x_{N-1} \end{pmatrix} \times \cdots \times \begin{pmatrix} 1 \\ x_1 \end{pmatrix}}_{\substack{N-times \\ called\ \ direct\ \ product}} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_3 \\ x_1 x_3 \\ \vdots \\ x_1 \cdots x_N \end{pmatrix} \qquad \ldots(4)$$

Then one has the η- expansion:

$$x_{i,t+1} = \sum_{\alpha=0}^{2^N-1} f_{i,\alpha} \eta_{\alpha,t} = \sigma\left[g_i\left(x_t\right)\right] \qquad \ldots(5)$$

That is, a nonlinear function in x-space can be expanded into linear function in η-space. The expansion (5) can be obtained by the three trivial remarks:

$$\begin{cases} \sigma(FG) = \sigma(F)\sigma(G) \\ \sigma(\sigma(F)) = \sigma(F) \\ [\sigma(F)]^2 = 1 \end{cases}$$

Now for the N-d vector state $\mathbf{x} = (x_1, x_2, \ldots, x_N)$, as $x_i = \pm 1$, there are totally 2^N possible states. You can write them all down, e.g, for N=3 :

$$\begin{pmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \end{pmatrix}$$ one column = 1 possible **x**-state.

So we define matrix φ_N in the same way (it's a N x 2^N matrix)

$$\varphi_N = \left(\frac{1}{\sqrt{2}}\right)^N \begin{pmatrix} 1 & -1 & \ddots & \ddots & & & -1 \\ 1 & 1 & \ddots & \ddots & \ddots & & -1 \\ \vdots & \vdots & & \ddots & \ddots & \ddots & \vdots \\ 1 & \vdots & & & \ddots & \ddots & -1 \\ 1 & 1 & & & & \ddots & -1 \end{pmatrix} \qquad \ldots(6)$$

In the η-space, one get another matrix, Φ_N, by specifying the value $x_i = \pm 1$ to the direct product in (4) :

Direct product definition:

$$A=\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \& B=\begin{pmatrix} b_{n1} & \cdots & b_{1l} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nl} \end{pmatrix} \Rightarrow A\times B=\begin{pmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{pmatrix}$$

$$\text{so} \quad \Phi_N=\left(\frac{1}{\sqrt{2}}\right)^N \underbrace{\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\times\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\times\cdots\cdots\times\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}}_{N-times} \qquad \ldots(7)$$

Then Φ_N has properties: $\Phi_N = \ (\Phi_N)^T = (\Phi_N)^{-1}$

$$(\Phi_N)^2 = I$$

$$\det(\Phi_N) = (-1)^N$$

e.g, for N=3, Φ_N is an 8x8 symmetric square matrix:

$$\Phi_3=\left(\frac{1}{\sqrt{2}}\right)^N \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix} \begin{matrix} \cdots & 1 \\ \cdots & x_1 \\ \cdots & x_2 \\ \cdots & x_1x_2 \\ \cdots & x_3 \\ \cdots & x_1x_3 \\ \cdots & x_2x_3 \\ \cdots & x_1x_2x_3 \end{matrix} \qquad \ldots(8)$$

Particularly, one will find that φ_N is included in Φ_N .i.e., φ_N corresponds to the rows of Φ_N which are related to x_i alone (rather than x_ix_j…). Or you can also say by expanding each x-column in φ_N to η-column instead.

So φ_N and Φ_N are related as:

$$\varphi_{N,t+1} = f_N \Phi_{N,t} = \varphi_{N,t} P_N \qquad \ldots(9)$$

where f_N is a Nx2^N matrix with component $f_{i,\alpha}$ in (5). P_N is a permutation matrix. By definition, permute $\varphi_{N,t}$ into $\varphi_{N,t+1}$.

But for computation convenience, we can enlarge (9) to :

$$\Phi_{N,t+1} = F_N \Phi_{N,t} = \Phi_{N,t} P_N \qquad \ldots(10)$$

F_N is now a 2^Nx2^N matrix. Here is again, f_N corresponding to some rows of F_N just like the φ_N to Φ_N.

Assume starting with time t=0, then from (10):

$$\Phi_{N,1} = F_N \Phi_{N,0} = \Phi_{N,0} P_N$$

$$\Rightarrow \Phi_{N,t} = \left(F_N\right)^t \Phi_{N,0} = \Phi_{N,0}\left(P_N\right)^t$$

If we let the initial set of states $\Phi_{N,0} = \Phi_N$ in (7), immediately:

$$F_N = \Phi_N P_N \Phi_N \qquad \ldots(11)$$

It means that F_N can be constructed, if P_N is given.

3 Permutation Function

P_N is a permutation matrix and can be used to describe the state transition of the net. It describes transition in this way: (from (10))

$$\underset{\Phi_{N,t+1}}{\begin{pmatrix} & \overset{j}{\begin{bmatrix} \\ \\ \\ \end{bmatrix}} & \end{pmatrix}} = \underset{\Phi_{N,t}}{\begin{pmatrix} & \overset{i}{\begin{bmatrix} \\ \\ \\ \end{bmatrix}} & \end{pmatrix}} \underset{P_N}{\begin{pmatrix} & \overset{j}{0} & & & \\ 0 & 1 & \cdots & \cdots & 0 \\ & \vdots & & & \end{pmatrix}} i$$

Recall that each column = a possible state η (which is expanded from $\mathbf{x}$).

If at time t+1, the state of column j = the state of column i at time t. Then it's a transition from j to i ($j \rightarrow i$). The element $(P_N)_{i,j}$ is set to 1, and all other elements in the same row / column should be zero.

For transition of a series of pattern, e.g. N=3, index the patterns with columns of initial matrix Φ_N. For example, you want the column in Φ_N to transit in order:

$1 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 6 \rightarrow 8 \rightarrow 1 \rightarrow 5 \rightarrow \ldots$. (loop again)

P_N will look like this:

$$P_3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Then, as long as you "sit on" one particular column of Φ_N and see it's transition by applying P_N, it will reproduce the wanted sequence you specified. e.g.:

$$\Phi_3 = \Phi_{3,0} = \left(\frac{1}{\sqrt{2}}\right)^N \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

$$\Phi_{3,1}=\left(\tfrac{1}{\sqrt{2}}\right)^N\begin{pmatrix}1&1&1&1&1&1&1&1\\1&-1&-1&1&1&-1&-1&1\\1&-1&1&-1&-1&-1&1&1\\1&1&-1&-1&-1&1&-1&1\\-1&1&1&-1&1&-1&-1&1\\-1&-1&-1&-1&1&1&1&1\\-1&-1&1&1&-1&1&-1&1\\-1&1&-1&1&-1&-1&1&1\end{pmatrix}\quad\Phi_{3,2}=\left(\tfrac{1}{\sqrt{2}}\right)^N\begin{pmatrix}1&1&1&1&1&1&1&1\\1&1&-1&-1&-1&1&-1&1\\-1&-1&-1&1&1&1&-1&1\\-1&-1&1&-1&-1&1&1&1\\1&-1&1&-1&1&1&-1&-1\\1&-1&-1&1&-1&1&1&-1\\-1&1&-1&-1&1&1&1&-1\\-1&1&1&1&-1&1&-1&-1\end{pmatrix}$$

$$\Phi_{3,3}=\left(\tfrac{1}{\sqrt{2}}\right)^N\begin{pmatrix}1&1&1&1&1&1&1&1\\-1&-1&1&-1&-1&1&1&1\\1&1&-1&-1&-1&1&1&-1\\-1&-1&-1&1&1&1&1&-1\\1&-1&-1&-1&1&-1&1&1\\-1&1&-1&1&1-1&-1&1&1\\1&-1&1&1&-1&-1&1&-1\\-1&1&1&-1&1&-1&1&-1\end{pmatrix}\quad\Phi_{3,4}=\left(\tfrac{1}{\sqrt{2}}\right)^N\begin{pmatrix}1&1&1&1&1&1&1&1\\-1&-1&-1&1&1&1&1&-1\\-1&-1&1&1&-1&-1&1&1\\1&1&-1&1&-1&-1&1&-1\\1&-1&-1&1&-1&1&-1&1\\-1&1&1&1&-1&1&-1&-1\\-1&1&-1&1&1&-1&-1&1\\1&-1&1&1&1&-1&-1&-1\end{pmatrix}$$

As it has been showed, the first column (or any column) repeats the wanted sequence of patterns.

So, by this method, one can generate any wanted sequence of maximal length 2^N, with *no repetition* in the sequence.

For a more direct approach, one can use the wanted, ordered sequence η^1, η^2, $\eta^\alpha\ldots\eta^{2\wedge N}$, (expand the sequence $x^1,x^2,\ldots, x^{2\wedge N}$ into η-space first!) to form the matrix Φ directly, i.e., each pattern η^α is a column of Φ. This time Φ no longer share the properties with Φ_N, but it do still satisfy equation similar to (11):

$$F_N=\Phi P_N\Phi^{-1} \qquad \ldots(12)$$

The P_N here, is like: $\begin{pmatrix}0&0&\cdots&0&1\\1&0&&&0\\0&1&\ddots&&\vdots\\\vdots&&\ddots&\ddots&\vdots\\0&\cdots&\cdots&1&0\end{pmatrix}$

Because it only need to *shift* the columns once a time. But Φ is not normalized and Hermitian any more, so you need to find Φ's inverse first.

The advantage of this approach is that the pattern η^α need not be distinct (when we can't find the Φ^{-1}, we use the pseudoinverse instead). Also the length of sequence can be less than 2^N. (e.g. you want η^1, η^2,…η^k , then just add any other (2^N-k) states (indeed *any* η can do, they doesn't matter!) to form the $2^N x 2^N$ Φ matrix. And just change the P_N correspondingly.)

The main concern of us about synthesis is, not how the matrixes permute the wanted sequence, but is how those matrixes relates to the real neural network configuration. As mention before, F_N can be obtained by (11) or (12). And the $Nx2^N$

matrix f_N can be extracted by

$$f_N = \varphi_N P_N \Phi_N \,, \qquad \ldots(13)$$

then one immediately get (5) $x_{i,t+1} = \sum_{\alpha=0}^{2N} f_{i,\alpha} \eta_{\alpha,t}$. We can claim that the synthesis problem is solved.

4 Discussions

Among the computation, $\Phi_{N,t+1} = F_N \Phi_{N,t} = \Phi_{N,t} P_N$, P_N is the *key* to move/transit the whole column of Φ_N. Without the middle part of the above equations, the transition done by matrix permutation is purely mathematics and nothing to do with the neural. (For example, you can construct an Φ matrix, which is not an expansion of the φ matrix, i.e., those η-states can not map to any possible x-states, and you can still permute this Φ in the way you like.)

The critical point is, from $P_N \Rightarrow F_N$. Although its just move from the L.H.S of Φ_N to R.H.S, but the left/right product meaning are so different, that:

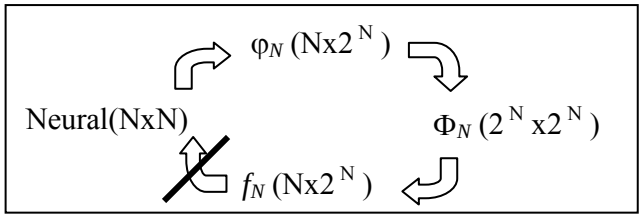
$$\begin{pmatrix} & j & \\ & \begin{bmatrix} \\ \\ \\ \end{bmatrix} & \\ \Phi_{t+1} & & \end{pmatrix} = \begin{pmatrix} & i & \\ & \begin{bmatrix} \\ \\ \\ \end{bmatrix} & \\ \Phi_t & & \end{pmatrix} \begin{pmatrix} \\ P_N \\ \\ \end{pmatrix}$$

$$\begin{pmatrix} & j & \\ & \begin{bmatrix} \\ \\ \\ \end{bmatrix} & \\ \Phi_{t+1} & & \end{pmatrix} = \begin{pmatrix} \\ F_N \\ \\ \end{pmatrix} \begin{pmatrix} & j & \\ & \begin{bmatrix} \\ \\ \\ \end{bmatrix} & \\ \Phi_t & & \end{pmatrix}$$

For the product with F_N, any column-j related only to itself (in the past). So you can drop all the other dummy states but one. You can also shorten the column (from $2^N \rightarrow$N) by only taking N rows among them. Finally it's realized that the resulting equation, is identical to $x_{i,t+1} = \sum_{\alpha=0}^{2N} f_{i,\alpha} \eta_{\alpha,t}$

Which is the tensor expansion introduced in (5).

So, it is the case:



We expand first in order to get a more powerful description about the nets (i.e., linearity), then compute and reduce it to the tensor expansion of f_N. But in general we can't go any further. We can't find general solution of g_i in (5), which represented the real physical network configuration. On the other hand, this method to solve synthesis problems is error intolerable.