Alternative Dynamic Network Structures for Non-linear System Modelling

K. P. Dimopoulos[1] and C. Kambhampati[2]
[1]*CITY Liberal Studies, Thessaloniki, Greece, Affiliated Institution of the University of Sheffield, U.K.,*
[2]*The University of Hull, Hull, U.K.*
[1]*k.dimopoulos@city.academic.gr,* [2]*C.Kambhampati@hull.ac.uk*

Abstract

Hopfield Neural Networks have been used as universal identifiers of non-linear systems, because of their inherent dynamic properties. However the design decision of the number of neurons in the Hopfield network is not easy to make, in order for the network model to have the necessary complexity, extra neurons are required. This poses a problem since the role of the states that these neurons represent is not clear.

Adding a hidden layer in the Hopfield network model increases the complexity of the model without posing the extra states problem. Alternatively breaking the problem down by having different interconnected Hopfield networks modeling each state, also increase the complexity of the problem.

A comparison between the three approaches (traditional Hopfield, Hopfield with a hidden layer, and multiple interconnected Hopfield networks) indicates equivalence between the three structures, but with the alternative cases having increased connectivity in the feedback matrix, and limited connectivity in the weight matrices.

1. Introduction

Hopfield Neural Networks (HNNs) have been very popular in literature as universal non-linear models. The dynamic properties of the HNNs makes them ideal for capturing the complete non-linear dynamics of any unknown non-linear process, as long as accurate estimates of the process states exist. However, very often when HNNs represent the dynamics of a process, they often have more internal states than the process they are representing. The extra states are hard to initialise, as they do not represent any of the process real states. Nevertheless, the extra complexity offered by the extra neurons is sometimes necessary for the HNN to approximate the process correctly.

In this paper we propose ways to increase the complexity of the neural network model, with out increasing the external states, thus alleviating the problem.

2. HNNs as non-linear systems

Hopfield Neural Networks are neural networks that exhibit dynamic properties due to internal/external feedback. [1-5]. These properties allow for the internal dynamics of unknown processs to be identified in contrast to the usual input-output identification that Feed Forward Neural Networks are limited to. Hopfield Networks have of the form:

$$\left.\begin{aligned} \dot{x} &= -Bx + W\sigma(x) + \Gamma u \\ y &= x_1 \end{aligned}\right\} \qquad (1)$$

or in a more precisely:

$$\left.\begin{aligned} \dot{x}_i &= -\beta_i x_i + \sum_{j=1}^{n} w_{ij}\sigma(x_j) + \gamma_i u \\ y &= x_1 \end{aligned}\right\} \qquad (2)$$

where $x \in \mathbb{R}^n$ is a vector with the states of the network, $\dot{x} \in \mathbb{R}^n$ is the first derivative of x, $B \in \mathbb{R}^{n\times n}$ is the feedback matrix (usually a diagonal matrix), $W \in \mathbb{R}^{n\times n}$ is the weight matrix, $\Gamma \in \mathbb{R}^n$ is the input matrix, and $\sigma(x)$ is a sigmoidal function like $\tanh(x)$. With out loss of generality, from now on it will be assumed that $h(x) = x_1$.

The equations that characterise a Hopfield network are those of a non-linear control affine system:

$$\left.\begin{aligned} \dot{x} &= f(x) + g(x)u \\ y &= h(x) \end{aligned}\right\} \qquad (3)$$

Therefore, Hopfield networks are a class of non-linear control affine systems, and thus all analysis of such systems can be also applied to them.

2.1. Relative Order of HNN

Relative order is an invariant property of physical systems. It defines the number of times the input has to be integrated in order to affect the output. According to Isidori [6]:

The system (3) *is said to have relative order r at a point* x^0 *if*

- $L_g L_f^k h(x) = 0$ *for all x in a neighbourhood of* x^0 *and all* $k<r-1$
- $L_g L_f^{r-1} h(x^0) \neq 0$.

Where $L_f h(x)$ *is the Lie derivative of h(x) in the direction of f(x) defined as:*

$$L_f h(x) = \sum_{j-1}^{n} \frac{\partial h}{\partial x_j} f_j(x)$$

Since the Hopfield network is a control affine system of the form (3), the concept of the relative order, is also be applicable it. By applying the relative order definition to the HNN, Dimopoulos and Kambhampati have proved that the structure of the HNN can determine its relative order, and vice versa [4], and that the relative order of the HNN will be upper limited by the number of states the neural network has. Specifically:

Relative Order Proposition:

For the Hopfield network in (2)*;*

- *If* $\gamma_i = 0, i = 1, \ldots (n-1)$, *and* $\gamma_n \neq 0$, *the relative order of the system is* $r \in [2, n]$, *if the following conditions are satisfied:*

$$\left.\begin{array}{l} w_{1,n} = 0 \\ w_{1,3}, \ldots w_{1,(n-1)}, w_{2,n} = 0 \\ \quad \vdots \quad \vdots \quad \vdots \quad \vdots \\ w_{(r-3),(r-1)}, \ldots w_{(r-3),(n-1)}, w_{(r-2),n} = 0 \\ w_{(r-2),r}, \ldots w_{(r-2),(n-1)}, w_{(r-1),n} \neq 0 \end{array}\right\}$$

  If $\gamma_1 \neq 0$, *then the relative order r is 1.*

- *If* $\gamma_i = 0, i = 1, \ldots, m, \gamma_j \neq 0, j = (m+1), \ldots n$, *the relative order of the system is r, with* $r \in [2, m+1]$, *if the following conditions are satisfied:*

$$\left.\begin{array}{l} w_{1,(m+1)}, \ldots, w_{1,n} = 0 \\ w_{1,3}, \ldots, w_{1,(n-1)}, w_{2,(m+1)}, \ldots, w_{2,n} = 0 \\ w_{2,4}, \ldots, w_{2,(n-1)}, w_{3,(m+1)}, \ldots, w_{3,n} = 0 \\ \quad \vdots \quad \vdots \quad \vdots \; \vdots \\ w_{(r-3),(r-1)}, \ldots, w_{(r-3),(n-1)}, w_{(r-2),(m+1)}, \ldots, w_{(r-2),n} = 0 \\ w_{(r-2),r}, \ldots, w_{(r-2),(n-1)}, w_{(r-1),(m+1)} \neq 0 \end{array}\right\}$$

  If $\gamma_r \neq 0$, *then the relative order r will be 1.*

- *If* $\gamma_i = 0, i \in [1, m) \cup (m, n], \gamma_m \neq 0$, *then the relative order of the system is r, where* $r \in [2, m) \cup (m, n]$.

Proofs of the above propositions can be found in [4].

2.2. Zero Dynamics of HNN

Zero dynamics are another invariant property of physical systems. In many instances they play a role exactly similar to that of the 'zeros' of the transfer function in a linear system [6]. Zero dynamics are the differential equations that describe the internal behaviour of the non-linear system when the system's output is 'forced' to be zero.

Let us assume a Hopfield network of the form of (2) and further assume that this network has a relative order $r<n$ and has the appropriate structure as defined in the relative order proposition. Then because of the structure of the weight matrices, the Hopfield will look like:

$$\left.\begin{array}{l} \dot{x}_i = -\beta_i x_i + \sum_{j=1}^{i+1} w_{ij} \sigma(x_j) \quad i = 1, \ldots, r \\ \dot{x}_i = -\beta_i x_i + \sum_{j=1}^{n} w_{ij} \sigma(x_j) + \gamma_i u \quad i = r+1, \ldots, n \\ y = x_1 \end{array}\right\}$$

Since the output of the network is the first state, and since each of the first *r* states is a function of only the first *r* states, then the first *r* states will be zero, and the zero dynamics of the Hopfield will be given by:

$$\dot{x}_i = -\beta_i x_i + \sum_{j=r+1}^{n} \left(w_{ij} - \frac{\gamma_i}{\gamma_r} w_{rj} \right) \sigma(x_j) \quad i = r+1, \ldots n$$

In order for the inverse of the model to be stable, these zero dynamics have to be stable. Stability of these can be in the same way as stability of a Hopfield can be determined; by linearising and examining the Eigen-values. The linearised zero dynamics are given by:

$$\dot{z} = Az + Bu$$

with B=0 and A given by:

$$A = \begin{bmatrix} h_{r+1,r+1} - \beta_{r+1} & \cdots & h_{r+1,n} \\ \vdots & h_{i,i} - \beta_i & \vdots \\ h_{n,(r+1)} & \cdots & h_{n,n} - \beta_n \end{bmatrix}$$

$$h_{ij} = \left(w_{ij} - \frac{\gamma_i}{\gamma_r} w_{rj} \right) \sigma'(x_j) \quad i, j = r+1, \ldots n$$

When recurrent networks represent the dynamics of a system, they often have more internal states than the system they are representing. Suppose that this is the case with a network with n states that is modelling a non-linear process of m states where $r<m<n$ (r being the relative order of both process and model). Then the non-linear process will have m-r zero dynamics, while the network model will have n-r zero dynamics. Therefore the extra states of the network model represent extra zero dynamics. Thus it is possible for the network to have more zero dynamics than the process it models. Nevertheless, training an n state network to identify an m state process, where $m<n$, has the same effect as with training the same network to identify an n state process using limited information (say the first m states) about the non-linear process. Since we have no information about the last n-m states, it is difficult not only to decide the role they play but also how to initialise them.

3. HNN with a hidden layer

Extra neurons are needed in order to increase the complexity of the Hopfield network, without increasing the number of states. This can be achieved by adding a hidden layer inside the Hopfield network. In this case, the network is composed out of three layers. The input and the output layer are composed of n neurons, i.e. the number of states the network is trying to model. The hidden layer is composed of m neurons. It is these neurons that will provide the extra complexity to the model. This architecture was proposed in [7] and is described by the following equations:

$$\left.\begin{aligned} \dot{x} &= -Ax + C\sigma(Dx + E) + Fu \\ y &= x_1 \end{aligned}\right\} \qquad (4)$$

where $A \in \mathbb{R}^{n \times n}$ is the diagonal feedback matrix, much the same with the feedback matrix in (2), $C \in \mathbb{R}^{n \times m}$ and $D \in \mathbb{R}^{m \times n}$ are the weight matrices, $E \in \mathbb{R}^m$ is a bias matrix, $F \in \mathbb{R}^n$ is the input weight matrix and $x \in \mathbb{R}^n$ is the state vector. In a more detailed form (4) becomes:

$$\left.\begin{aligned} \dot{x}_i &= -a_i x_i + \sum_{j=1}^{m} c_{ij} \sigma \left[\sum_{k=1}^{n} (d_{ik} x_k + e_k) \right] + f_i u \\ y &= x_1 \end{aligned}\right\} \qquad (5)$$

3.1. Equivalence between Simple HNN and HNN with a hidden layer

Comparing the description of the hidden layered network in (2) with the description of the Hopfield network of (5) it is clear that the proposed structure has many attributes of the original structure. To begin with, both structures utilise external feedback. Each state is directly connected to itself, separately from the connection with the state vector. Secondly the method of connection of the input to the network is identical in both structures. Finally, both structures use the weighted sigmoid of a function of time ($x(t)$ in the case of the Hopfield, $Dx(t)+E$ in the case of the hidden layer structure). Therefore one can expect that the non-linear properties investigated in the previous chapter will also hold for this structure.

Consider the proposed structure for a moment. Given that

$$\dot{x} = -Ax + C\sigma(Dx + E) + Fu$$

from (4) let us define a new auxiliary variable $\xi = Dx + E, \quad \xi \in \mathbb{R}^m$. Then the derivative of this new variable is given by $\dot{\xi} = D\dot{x}$, and substituting (4) into this we obtain:

$$\dot{\xi} = -DAx + DC\sigma(Dx + E) + DFu \qquad (6)$$

Now let us define a new state vector $z = [x, \xi]^T \in \mathbb{R}^{n+m}$. By combining (4) and (6) we can see that:

$$\dot{z} = \begin{bmatrix} \dot{x} \\ \dot{\xi} \end{bmatrix} = -\begin{bmatrix} A & 0 \\ DA & 0 \end{bmatrix} z + \begin{bmatrix} 0 & C \\ 0 & DC \end{bmatrix} \sigma(z) + \begin{bmatrix} F \\ DF \end{bmatrix} u \quad (7)$$

But this is similar to the form of the Hopfield but with the matrices B, W and Γ given by:

$$B = \begin{bmatrix} A & 0 \\ DA & 0 \end{bmatrix}, \quad W = \begin{bmatrix} 0 & C \\ 0 & DC \end{bmatrix}, \quad \Gamma = \begin{bmatrix} F \\ DF \end{bmatrix}$$

It could be noted that the B matrix is no longer a diagonal matrix. Although the first n states are fed-back to themselves, the last m states are fed with only the first n states. This hints to a special significance of the first n states. Under closer inspection of (7) it can be seen that the first n states are the 'revealed' states or the approximations to the states that the network is trained for.

Another important observation is that the weight matrix W is sparse. The 'revealed' states are not directly connected here. Instead there is an indirect connection through the auxiliary states and the feedback matrix.

To summarise, the proposed hidden layered architecture of n input neurons and m hidden ones is equivalent to a Hopfield structure of $N = n + m$ neurons but with the feedback matrix not being diagonal, and the weight matrix being sparse.

4. Multiple Interconnected HNNs

The proposed structure is composed of many simple Hopfield Neural Networks, each modelling an aspect of the problem, thus breaking the main problem to smaller ones. This structure was inspired from multiple or stacked networks [8-11]. These types of networks have been successfully employed in various applications ranging from image recognition to pattern prediction. In some of those cases the networks where trained to solve the same problem and the final decision was taken by means like voting. In our case, smaller networks are employed for different parts of the problem, the solution to which rises from the combination of the smaller networks.

As an example consider a system with n states, similarly to (3):

$$\left.\begin{aligned} \dot{x}_1 &= f_1(x) + g_1(x)u \\ \dot{x}_2 &= f_2(x) + g_2(x)u \\ &\vdots \\ \dot{x}_n &= f_n(x) + g_n(x)u \\ y &= x_1 \end{aligned}\right\} \qquad (8)$$

We can consider this as n input-output problems instead of one input-state problem with n tracking parameters. Specifically we can train n smaller networks each modelling one state of (8) but with n inputs (the regular input u and the other n-1 states) instead of just u. Therefore the resulting structure for the k^{th} network modelling the k^{th} process state will look like:

$$\left.\begin{aligned} \dot{z}_{k,1} &= -\beta_{k,1} z_{k,1} + \sum_{j=1}^{m_k} w_{k,(1,j)} \sigma(z_{k,j}) \\ &\quad + \sum_{\substack{j=1 \\ j \neq k}}^{n} \gamma_{k,1,j} z_{j,1} + \gamma_{k,1,n+1} u \\ &\vdots \qquad\qquad \vdots \\ \dot{z}_{k,i} &= -\beta_{k,i} z_{k,i} + \sum_{j=1}^{m_k} w_{k,(i,j)} \sigma(z_{k,j}) \\ &\quad + \sum_{\substack{j=1 \\ j \neq k}}^{n} \gamma_{k,i,j} z_{j,1} + \gamma_{k,i,n+1} u \\ &\vdots \qquad\qquad \vdots \\ \dot{z}_{k,m_k} &= -\beta_{k,m_1} z_{k,m_k} + \sum_{j=1}^{m_k} w_{k,(m_k,j)} \sigma(z_{k,j}) \\ &\quad + \sum_{\substack{j=1 \\ j \neq k}}^{n} \gamma_{k,m_k,j} z_{j,1} + \gamma_{k,m_k,n+1} u \end{aligned}\right\} \qquad (9)$$

where m_k is the number of neurons for this network and the state of the process is modelled by the first state of the network. In the case where the weights have three indexes, the first one identifies the network, the second the state, and the last the connection they belong to. Therefore the weight $w_{3,6,4}$ is a weight belonging to the third network connecting the sixth state to the fourth state.

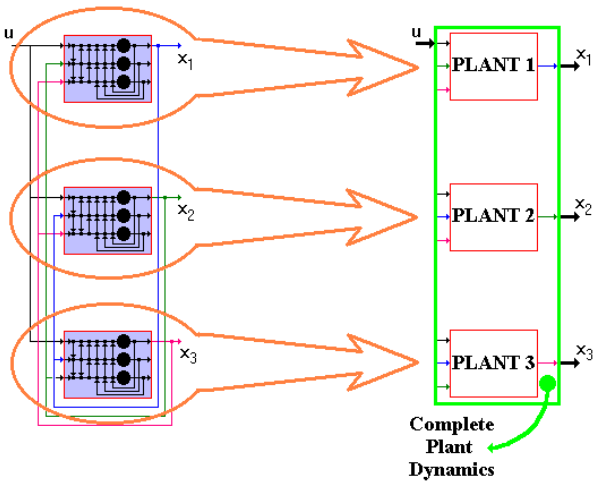

Figure 1: Multiple Interconnected network structure modeling a non-linear process

4.1. Equivalence between Simple HNN and multiple interconnected HNNs

As it can be seen from (9), the proposed structure is composed out of smaller Hopfield networks. As far as each network is concerned, the process that it is trying to model has multiple inputs and a single output. Therefore from a collective point of view the collection of the networks are modelling a collection of processes each with many inputs but one output, as it can be seen in Figure 1. This has the advantage that it is possible to train each network independently of the rest. Since each network structure has to model only one variable, it is much easier to train (Figure 2).

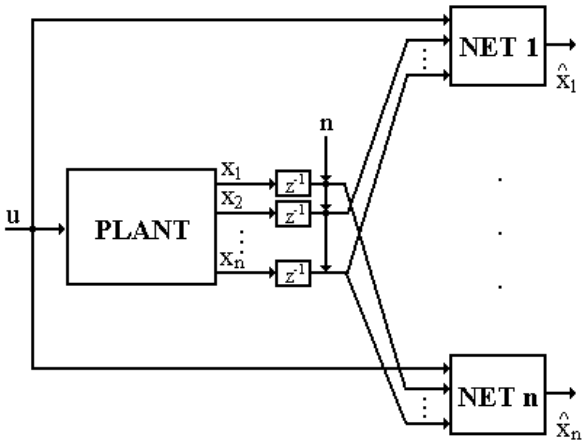

Figure 2: Training of a multiple interconnected network structure.

Once all the networks are trained, it is a simple matter of putting together the building blocks to create the complex representation required. A second advantage of this structure is noise immunity. Given that noise was included in the states that were fed to the networks as inputs during training, the overall structure will be immune to disturbances. Since each network output will be an approximation to the real state of the process, it can be therefore considered to be the state of the process with some noise added. This can be seen more clearly if we consider the following. Let at time t_1, $x_k(t_1)$ be the k^{th} state of the process, and $\hat{x}_k(t_1)$ be the k^{th} state of the network. Let $n(t)$ be the function which defines the difference between $\hat{x}_k(t)$ and $x_k(t)$. Assuming that the network is trained, then at any time this difference must be in a neighbourhood of zero:

$$\hat{x}_k(t) - x_k(t) = n(t), \quad n(t) \in (-\varepsilon_0, \varepsilon_0)\ \forall t$$

Solving this for $\hat{x}_k(t)$ we get

$$\hat{x}_k(t) = x_k(t) + n(t) \qquad (10)$$

This expresses that at any time the state approximation of the network model equals the state of the process plus a small value. In (10), $n(t)$ can be approximated by noise with zero average (Figure 2).

Now, let us consider the structure in Figure 1. Each of the individual networks will be described by (9). Define a state vector ξ as the composition of all the states of the networks:

$$\begin{aligned}\xi &= \left[\xi_1, \ldots \xi_k, \ldots \xi_n\right]^T \\ &= \left[\, z_{1,1} \ldots z_{1,m_1}, \ldots, z_{k,1} \ldots z_{k,m_k}, \ldots, z_{n,1} \ldots z_{n,m_n} \,\right]^T\end{aligned} \qquad (11)$$

and let the matrices Γ_{ij}, C_i, W_i, be:

$$\Gamma_{i,j} = \begin{bmatrix} \gamma_{i,1,j} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{i,m_i,j} & 0 & \cdots & 0 \end{bmatrix}, C_i = \begin{bmatrix} \gamma_{i,1,n+1} \\ \gamma_{i,2,n+1} \\ \vdots \\ \gamma_{i,m_i,n+1} \end{bmatrix}, W_i = \begin{bmatrix} w_{i,1,1} & \cdots & w_{i,1,m_i} \\ \vdots & \ddots & \vdots \\ w_{i,m_i,1} & \cdots & w_{i,m_i,m_i} \end{bmatrix}$$

Then by combining n networks of the form (9) and substituting for the states with (11) we get the description:

$$\dot{\xi} = \begin{bmatrix} B_1 & \Gamma_{12} & \cdots & \Gamma_{1n} \\ \Gamma_{21} & B_2 & \cdots & \Gamma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \Gamma_{n1} & \Gamma_{n2} & \cdots & B_n \end{bmatrix} \xi + \begin{bmatrix} W_1 & 0 & \cdots & 0 \\ 0 & W_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W_n \end{bmatrix} \sigma(\xi) + \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{bmatrix} u \qquad (12)$$

This description is a Hopfield network. Nevertheless there are subtle differences between this representation and (2). The feedback matrix retains its diagonal, but now there are additional feedback connections between the states. The Γ matrices have zero elements everywhere except the first column. This hints to a special significance for these states. As a matter of fact these are the first states of each network and therefore according to our design of the multiple structure, the approximating states to the non-linear states. Thus each state in the new structure has feedback connections to itself (because of the B matrices) and to the states that approximate the states of the process (because of the Γ matrices). Furthermore, the weight matrix is a sparse matrix. The distributed system approach of the multiple network structure can be clearly seen here: the weight matrix provides connections between 'neighbourhoods' of

states. In each case, these sates correspond to the states of a network in the multiple network structure alone. The connections between these 'neighbourhoods' of states are left to weights in the feedback matrix. Finally the input matrix is in exactly the same format as in the original Hopfield configuration of (2).

The similarities between the two proposed structures should be noted. In both cases, the feedback matrices have increased in connectivity, allowing information to pass directly from one state to the other. On the other hand, the connectivity in the weight matrices has been limited and fewer states are required to pass through the sigmoid function. A first glance to this indicates that it is not important. Nevertheless, most neural network applications are implemented in software rather than in hardware because of practical implications. Calculating the sigmoid of a state and then multiplying it with a weight is less efficient than just multiplying the two numbers together, not just because of the extra calculation need, but because calculating the sigmoid (a non-linear function) of a number is also computationally expensive.

5. Alternative Neural Networks as non-linear systems

The network structures presented in this section are equivalent to the Hopfield network, with the feedback matrix no longer a diagonal matrix, and the weight matrix being sparse. The equivalent equations (7) and (12) of the alternative structures still describe control affine non-linear systems and thus the concepts discussed in the previous part still apply to them. As a mater of fact, with little effort, that analysis can be easily extended to cover the alternative structures.

5.1. Relative order of alternative structures

Consider the propositions about the relative order of the Hopfield network. There the relative order is defined from the structures of the weight and the input matrices. In the case of simple Hopfield networks the feedback matrix is not taken into account, because it connects a state with itself. In order to extend the propositions for the cases of the alternative network structures, the effect of the feedback matrix has to be taken into account. Here the effect of the weight matrix in relation with that of the feedback matrix has to be considered. The feedback matrix is associated with the states directly, while the weight matrix is associated with the threshold function, which is a sigmoid. So generally, both matrices are multiplied with a function of the states, and therefore whatever issues arise for the structure of one, will also arise for the structure of the other. Consider a fully connected network:

$$\begin{aligned}
\dot{x}_1 &= -\beta_{11}x_1 - \ldots - \beta_{1n}x_n + w_{11}\sigma(x_1) + \ldots + w_{1n}\sigma(x_n) \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
\dot{x}_i &= -\beta_{i1}x_1 - \ldots - \beta_{in}x_n + w_{i1}\sigma(x_1) + \ldots + w_{in}\sigma(x_n) \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
\dot{x}_n &= -\beta_{n1}x_1 - \ldots - \beta_{nn}x_n + w_{n1}\sigma(x_1) + \ldots + w_{nn}\sigma(x_n) + \gamma_n u \\
y &= x_1
\end{aligned}$$

which can be re-arranged as:

$$\begin{aligned}
\dot{x}_1 &= [-\beta_{11}x_1 + w_{11}\sigma(x_1)] + \ldots + [-\beta_{1n}x_n + w_{1n}\sigma(x_n)] \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
\dot{x}_i &= [-\beta_{i1}x_1 + w_{i1}\sigma(x_1)] + \ldots + [-\beta_{in}x_n + w_{in}\sigma(x_n)] \\
&\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
\dot{x}_n &= [-\beta_{n1}x_1 + w_{n1}\sigma(x_1)] + \ldots + [-\beta_{nn}x_n + w_{nn}\sigma(x_n)] + \gamma_n u \\
y &= x_1
\end{aligned}$$

Considering the above structure, it can be seen that the network is characterised by functions of the form $-\beta_{ij}x_j + w_{ij}\sigma(x_j)$. This also indicates to a similar behaviour of the structures of the feedback and weight matrices as far as the relative order is concerned. In order for the relative order to belong in the region of [*2*,*n*] the combination of the feedback matrix and the weight matrix must give rise to a structure similar to that of the structure of the weight matrix in the **Relative Order Proposition**.

5.2. Zero dynamics of the alternative structures

Previously we demonstrated that the extra states of a Hopfield Neural Network could describe extra zero dynamics. It has also been demonstrated that in the equivalent structure of the hidden layered Hopfield, the first *n* states approximate the process states, and the last *m* states (where *m* is the number of neurons in the hidden layer) are extra states. Therefore these states describe extra zero dynamics.

Similarly, in the equivalent form of the multiple interconnected networks, only the first state of each network approximates a state of the non-linear process. Since the remaining states in each network also appear in the equivalent structure, they must also describe extra zero dynamics.

Generally, from the point of view of a Hopfield network with more states than the process it is modelling, we can say that the Hopfield network is trying to model a process with an equal number of

states but with using limited knowledge of the process states to train. In the case of the HNN with a hidden layer, the extra states are internal, and do not appear in the output of the network. Therefore no matter the number of hidden neurons, the network will always try to model a process with the same number of observable states. Similarly, in the multiple interconnected Hopfield structure, each sub-network is modelling a state of the process and it is that state that is used to connect it to the other sub-networks. The other states of each sub-network only appear in that network and not in the output of the resulting structure. Therefore, the resulting structure will always model a process with the correct number of states.

5.3. Which network?

We have argued that a Hopfield network can be used to model the dynamics of a non-linear process. When extra complexity in the network model is needed, there exist three alternatives. The first is to use a Hopfield network but with extra neurons. *This gives rise to the problem of identifying the role of the extra neurons when these do not represent extra zero-dynamics*. If the number of neurons of the network is *N*, then the number of variables (V_S) that need training (and therefore the dimensionality of the training problem) is: *N* variables from the feedback matrix, *N*×*N* variables from the weight matrix, and *N* variables from the input matrix, a total of:

$$V_S = N + N \times N + N = N^2 + 2N$$

A second solution is to add a hidden layer to the Hopfield model, thus increasing its complexity. This structure has been seen to be equivalent to the first, with the number of input / output neurons and the hidden ones equal to the total number of neurons in the first case. i.e. a hidden layer Hopfield with *n* input / output neurons and *m* hidden (where *n* is also the number of states of the process), is equivalent in form to a simple Hopfield with $N = n + m$ neurons. But because the equivalent structure is sparse, the number of variables that need training (V_L) can be seen from (4) as *n* variables from the *A* matrix, *n*×*m* variables from the *C* matrix, *m*×*n* from the *D* matrix, *m* from the *E* matrix, and *n* from the *F* matrix. A total of

$$V_L = n + n \times m + m \times n + m + n = 2n + m + 2(nm)$$

But since $m = N - n$ the total number of variables in terms of number of modelled states and equivalent neurons is

$$V_L = 2n + N - n + 2[n(N-n)] = 2n + N - n + 2Nn - 2n^2 \Rightarrow$$
$$V_L = n + N + 2Nn - 2n^2$$

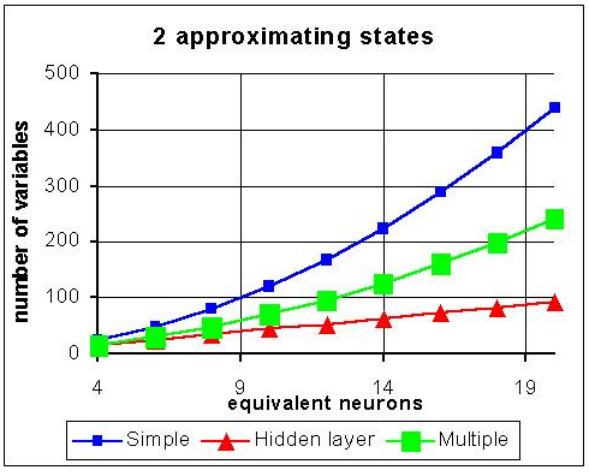

Figure 3: The three cases with varying number of neurons in their equivalent forms, when the process has 2 states

Finally, the third option is to use many simpler interconnected Hopfield networks, each modelling one state of the process. Assuming that each sub-network has *m* neurons, then the overall structure is equivalent to the first case, with the total number of neurons in the equivalent form $N = m \cdot n$ where *n* is again the number of states of the process (and consecutively number of sub-networks in the structure). Then the number of variables that need training (V_M) are the number of sub-networks times the number of variables in each sub-network:

$$\left.\begin{array}{l} V_M = n \cdot \left(2m + m^2\right) \\ m = \dfrac{N}{n} \end{array}\right\} \Rightarrow V_M = n \cdot \left(2\frac{N}{n} + \frac{N^2}{n^2}\right) \Rightarrow$$
$$V_M = 2N + \frac{N^2}{n}$$

These three equations have serious implications on the choice of network model. In the first case the number of variables can be seen to be independent of the number of process states, but it is proportional to the square of the number of neurons in the network. Therefore as the number of extra neurons increases, the number of variables that need training increases proportionally to the square of that value. In the second case, the relation between the number of variables that need training and the total number of neurons in all layers is of a first degree, but is also proportional to the

square of the number of process states. Finally, in the case of the multiple interconnected networks, the number of variables that need training is proportional to the square of total number of neurons in all the networks, but is inversely proportional to the number of states of the process. The implications of these can be seen more clearly in Figure 3 and Figure 4.

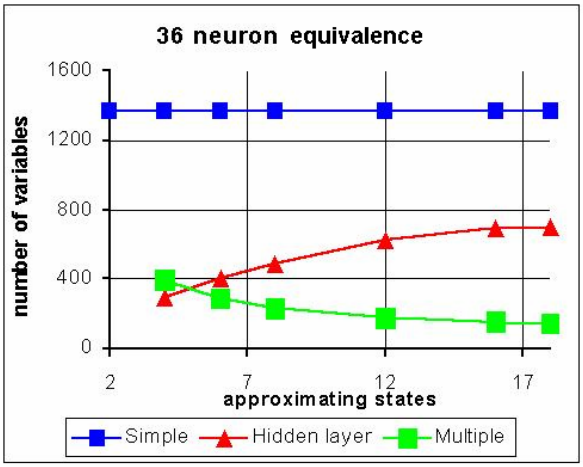

Figure 4: The three cases with varying number of states in the process, when their equivalent forms have 36 neurons.

Where the number of states in the process that will be modelled are not many, but increased complexity is needed, the most efficient structure in terms of number of variables that need training is the Hopfield with the hidden layer. Then the effect of a small *n* in the equation will be minimal, and at the same time it is possible to increase the complexity of the model by adding neurons in the hidden layer since their effect is linear.

On the other hand, where increased complexity is needed in modeling a process with many states, the multiple interconnected network structure would be most efficient, since the number of variables that need training will be inversely proportional to the number of states.

At this point we should note that while minimising the number of variables that need training is desirable, there should be a limit in the minimisation. As someone would expect there is a trade off between number of variables and complexity of the model.

6. Simulations

To test the modelling abilities of the proposed structures, a well-known non-linear process is selected, and multiple neural networks of different configurations were trained as models for this process.

6.1. The non-linear process

The process selected for the simulations is a well-known, well-used non-linear system [3, 4, 12-18]. The Single Link Manipulator (SLM) is essentially a pendulum, and the control problem is to control at any point in time, its position and velocity. Given a weightless rod of length *l*, a dimensionless mass *m* is placed at one end and the other is pivoted to a fixed point in space in such way as to allow the rod to move in only one direction. The friction coefficient at the pivot point is *v*. This system is illustrated in Figure 5. Given an input torque *u* at the pivot, then at an angle θ, using Newton's Laws we can get:

$$\sum T = I \cdot \ddot{\theta} \Leftrightarrow u - mgl\sin\theta - v\dot{\theta} = ml\ddot{\theta} \Leftrightarrow$$

$$\ddot{\theta} = -g\sin\theta - \frac{v}{ml}\dot{\theta} + \frac{1}{ml}u$$

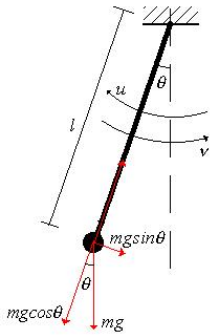

Figure 5: The Single Link Manipulator

Transferring the above equation in to state space by setting x_1 equal to the position and x_2 equal to the velocity, and by setting *m=2kg*, *l=1m* and *v=6kg* m^2/s we get:

$$\left.\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -9.8\sin x_1 - 3x_2 + 0.5u \\ y &= x_1 \end{aligned}\right\} \qquad (14)$$

This system has a relative order of 2. Once the system is linearised around the equilibrium point $(x_o, u_o) = (0,0)$, we see that the system is stable, and that it has eigenvalues at $-1.5 \pm j2.748$.

6.2. Specifying the training algorithm

In order to compare the proposed neural network structures with simple HNN, five sets of training

experiments were performed using a simple Genetic Algorithm (GA). Each set composed of ten trials for the same structure, with each trial starting from a different random number (seed for the random number generator). The same ten seeds were used for all sets. In all tests direct encoding was used to design the gene sequences that encode the neural networks: The sequence was composed out of real values, each uniquely corresponding to a weight in the matrices.

In all cases, the fitness was defined based on the Normalised Mean Square Error (NMSE) [4] of the states as defined by:

$$NMSE = \frac{1}{N}\sum_{i=1}^{N}\left(100\frac{(\max_i - \min_i)}{P}\sum_{j=1}^{P}(x_{ij}^{p} - x_{ij}^{m})^2\right)$$

Where N is the number of states of the system, P is the number of training patterns, max_i is the maximum output value of output i, min_i is the minimum value output of output i, x^p is the system output and x^m is the model output.

Furthermore, training was stopped when either the NMSE of the best network dropped bellow 5×10^{-6}, or when the algorithm finished the 300th generation, whichever criteria was reached first. This was because from preliminary tests it was observed that the NMSE did not improve significantly after 300 generations. In all cases a population of 50 networks was used. A mutation rate of 40% was used in all GA simulations. Although this percentage seems large for a mutation rate for a GA, it is not the percentage for each gene in the gene sequence, but it is the possibility that one gene in the gene sequence will mutate. This ensures that at most only one gene in a genotype will mutate. This is wanted because in many cases the genomes are sort in length, and therefore if two genes mutate, a large percentage of the genome changes.

The pseudo-code describing the training algorithm follows:

```
1. Create and initialise a new population
   of 50 networks.
2. For every network in the population,
   test it and calculate the NMSE.
3. Short the population from smallest
   NMSE to largest.
4. If the stopping criteria are reached
   END SIMULATION. Else go to 5.
5. Do crossover and mutation operations
   using the 10 networks with the lowest
   NMSE, creating 18 new networks
   substituting the 18 networks with the
   highest NMSE. Go to 2.
```

The stopping criteria are described above.

6.3. Training the models

The first set of experiments involved training simple HNNs with five neurons, as models of the SLM. This way the network model has more states than the process. This will form the basis of comparison with the proposed structures. The results of this set are displayed in table *T1*, in the appendix. Figure 6, shows the phase portrait of the best network in this set.

In order to test the Hopfield network with the hidden layer, a network with two output neurons (one for each process state) and three nodes in the hidden layer was used. As we have seen this is equivalent to the five-neuron simple HNN. For this set of simulations a GA was used to train the network similarly to the previous case. For reasons of comparison, the same random number seeds where used to produce ten networks. The results for this set are presented in table *T2*, in the appendix. Figure 7, shows the phase portrait of the best network in this set.

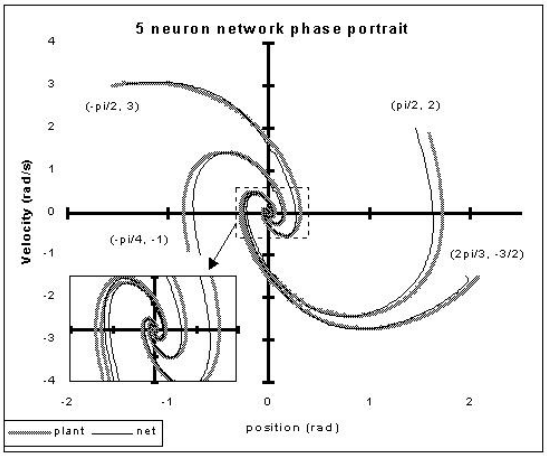

Figure 6: The phase portrait of the best 5-neuron network compared with the process, for different initial conditions.

In the final set of simulations the multiple networks architecture was tested. Since the non-linear process has two states, a double Hopfield network was trained with each network modelling a state. Training the networks was done with a GA, with each sub-network trained not separately, but in parallel with each other. Similarly with all previous sets, the training procedure was repeated ten times with ten different initial seeds for the random number generator. The results for this set are presented in the appendix, table *T3*. Figure 8, shows the phase portrait of the best network in this set.

6.4. Discussion of results

In Figure 6, Figure 7 and Figure 8, the phase portraits of the best networks of each set, are illustrated for different initial conditions. In order to illustrate the ability of the networks to extrapolate, the initial conditions used in the phase portraits are outside the region used for training the networks. This region is illustrated in the attached detail, in each figure.

We can clearly see how the networks are quite capable of imitating the phase portraits of the process in the region they were trained in, while outside that region they do not perform as well. In particular, the five-neuron network originally seems to perform very well inside the training region, but on a closer look, one can clearly see that the performance is not as good as those in the next two cases.

In the case of the Hopfield network with hidden layer, the performance is equally good in both regions (Figure 7). Using networks of this type does not compromise the ability to identify the non-linear dynamics between the two regions.

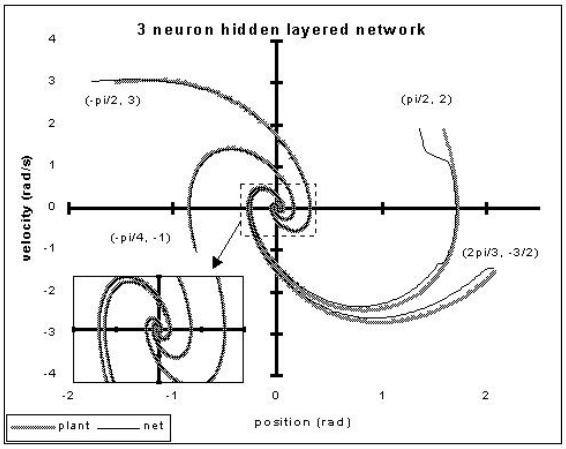

Figure 7: The phase portrait of the best Hopfield with a hidden layer compared with the process, for different initial conditions.

Furthermore, in Figure 8 the phase portrait of the best double network structure is illustrated (case *T2-1*). Comparing this with the previous phase portraits the advantages of this structure become evident. This network outperforms all the previous cases, inside and outside the training regions.

Comparing the results obtained from training a HNN with a hidden layer and that of its equivalent five-neuron network, it can be seen that on average the Hopfield with the hidden layer performs worst than the equivalent five-neuron case. Under closer examination though, it becomes evident that more networks of the first case have test errors in the order of 10^{-4} than in the second.

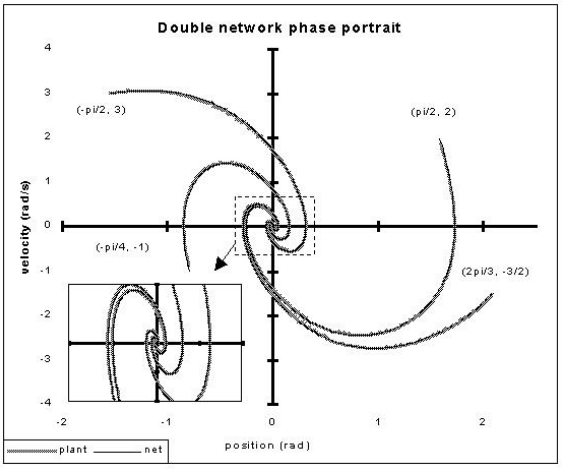

Figure 8: The phase portrait of the best double Hopfield compared with the process, for different initial conditions.

A second alternative is to use multiple interconnected networks, each modelling a state of the process. Earlier it was shown that this structure is also equivalent to a larger Hopfield network. In order to investigate the approximating properties of this structure a number of double networks were trained to approximate the process. In each case, the each sub-network was composed of two neurons, and was trained to model a state of the process. This structure is roughly equivalent to the five-neuron case, and the Hopfield with the hidden layer. Comparing this case with its equivalents, one can see that on average the double structure outperforms both previous cases, for both train and test input sets.

By inspection of the tables T1, T2 and T3 located at the appendix, we can see that the simple HNN with 5 neurons has the largest average training error, followed by the HNN with the hidden layer. The best average training error (lowest value) is achieved by the double HNNs. Under closer inspection, we can see that the training error of all double HNNs is located in the range of 10^{-5}, while this is also true about most HNN with a hidden layer. In contrast, there is only 1 case (case 4 of table T1) of a simple HNNs where the training error is so low. This is a clear indication that the alternative networks structures are easier to train.

In order to ensure that the networks are not over-trained (perform very well only for data used during training, and not very well for other data), a test error for all networks is calculated, using data unseen during training. It is usual to assume that if a Neural Network (NN) has a small training error but a large test error,

then the NN has over-trained since it no longer performs well with other data from what it was trained with.

By inspecting table T3 we can clearly see that the double HNN structure again outperforms the other two structures, having an average test error in the range of 10^{-4}, while the average training error is in the range of 10^{-5}. This is true for all double HNNs, indicating that none of the double HNNs have over-trained.

The average test error of the HNN with the hidden layer is above that of the simple HNN, and after closer inspection of tables T2 we can see that there are five cases where the test error is much larger than the training error (cases 2, 5, 7, 9 and 10). Therefore in these five cases the networks have over-trained.

Finally, in table T1, there is only one case when the test error in much larger than the test error (case 4). It is very interesting to observe case 9 in table T1. In this case, the test error is very high in respect to all other cases. However this is also true about the training error. Therefore in this case the network has not over-trained; rather it not trained very well.

The test errors indicate that there multiple HNN structures are also harder to over-train, than simple HNNs and HNNs with a hidden layer. However, the opposite can be claimed about HNNs with a hidden layer.

7. Conclusions

It has been shown that any extra states of a network trained as a model of a non-linear process, are connected to the non-linear zero dynamics, and are therefore essential for a good approximation. This has the effect of making the training more difficult by increasing the dimensions of the search space, and therefore the number of iterations for the training algorithm necessary to lower the training error to a satisfactory point. One way of perceiving the effect of training an m state network to identify an n state process, where $m>n$, is similar to training the same network to identify an m state process using limited information (say the first n states) about the non-linear process. Since we have no information about the last m-n states, it is difficult not only to decide what the roles of the extra states are, but also how to initialise them.

Two alternatives have been proposed. The first is to use a Hopfield network with as many states as the process, but to increase its complexity with the use of a hidden layer. It was shown that, a structure like this is equivalent to a simple HNN with a total number of neurons equal to the number of neurons in both input and hidden layers of the proposed structure. However, only some of the neurons are directly connected. The advantage of this network type is that while it increases the complexity of the network, the problem associated with choosing initial conditions is no longer present.

The second alternative investigated, was to use multiple interconnected networks, each modelling a single state of the process, thus breaking down the problem. Such a structure with n networks and m neurons was found to be equivalent to a single Hopfield neural network with n times m neurons but with the feedback matrix including extra feedback between the states, and the weight matrix being sparse.

In a comparison of the number of variables needed to be trained in equivalent forms in each case, it was found that in all cases as the number of equivalent neurons increase the number of variables increase with a higher rate in the case of the simple HNN, with a slower rate in the Hopfield with the hidden layer, and with much slower rate in the case of the interconnected Hopfield networks.

Experimental results indicated that both proposed structures (multiple interconnected Hopfield networks and HNN with hidden layer) have better approximation capabilities than the Simple Hopfield network of equivalent number of neurons, and better extrapolation capabilities. In addition multiple HNN structures are easier to train and harder to over-train, while HNNs with hidden layer are easy to train, but there is a high probability that the network will over-train.

8. References

[1] J. J. Hopfield, "Neural Networks and Physical Systems With Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences of the United States of America-Biological Sciences*, vol. 79, pp. 2554-2558, 1982.

[2] K. P. Dimopoulos and C. Kambhampati, "Apriori Information in Network Design," in *Dealing with Complexity: A Neural Network Approach.*, M. Karny, K. Warwick, and V. Kurkova., Eds., 1998.

[3] K. P. Dimopoulos and C. Kambhampati, "Multiple Interconnected Hopfield Networks for Intelligent Global Linearising Control," presented at IJCNN, Washington D.C. USA, 1999.

[4] K. P. Dimopoulos, C. Kambhampati, and R. J. Craddock, "Efficient recurrent neural network training incorporating a priori knowledge," *Mathematics and Computers in Simulation*, vol. 52, pp. 137-162, 2000.

[5] N. J. Dimopoulos, "A study of the asymptotic-behavior of neural networks," *IEEE Transactions On Circuits and Systems*, vol. 36, pp. 687-694, 1989.

[6] A. Isidori, *Nonlinear Control Systems. An Introduction*, 2nd ed: Springer-Verlag, 1989.

[7] C. Kambhampati, A. Delgado, J. D. Mason, and K. Warwick, "Stable Receding Horizon Control Based on

Reccurent Networks," presented at IEE Control Theory Applications, 1997.
[8] S. W. Lee and S. Y. Kim, "Integrated segmentation and recognition of handwritten numerals with cascade neural network," *IEEE Transactions On Systems Man and Cybernetics Part C- Applications and Reviews*, vol. 29, pp. 285-290, 1999.
[9] J. Lis, "The synthesis of the ranked neural networks applying genetic algorithm with the dynamic probability of mutation," *From Natural to Artificial Neural Computation*, vol. 930, pp. 498-504, 1995.
[10] D. H. Wolpert, "Stacked Generalization," *Neural Networks*, vol. 5, pp. 241-259, 1992.
[11] D. V. Sridhar, R. C. Seagrave, and E. B. Bartlett, "Process modeling using stacked neural networks," *Aiche Journal*, vol. 42, pp. 2529-2539, 1996.
[12] A. Delgado and C. Kambhampati, "Differential Geometric Control of Non-linear System with Recurrent Neural Networks," Department of Cybernetics, University of Reading, Reading 1994 1994.
[13] A. Delgado, C. Kambhampati, and K. Warwick, "Approximation of non-linear systems using dynamic recurrent neural networks," presented at International conference on identifications in engineering systems, Univ. of Wales, Swansea, U.K., 1996.
[14] A. Delgado and C. Kambhampati, "Global linearising control using recurrent networks," presented at European Robotics and Intelligent systems Conference (EURICON), Malaga, Spain, 1994.
[15] A. Delgado, "Input/output linearisation of control affine systems using neural networks," in *Department of Cybernetics*. Reading: University of Reading, 1996.
[16] A. Delgado, C. Kambhampati, and K. Warwick, "Input/output linearization using dynamic recurrent-neural networks," *Mathematics and Computers in Simulation*, vol. 41, pp. 451-460, 1996.
[17] C. Kambhampati, R. J. Craddock, M. Tham, and K. Warwick, "Internal model control of nonlinear systems through the inversion of recurrent neural networks," *World Congress on Computational Intelligence, Alaska.*, 1998.
[18] A. Delgado, C. Kambhampati, and K. Warwick, "Dynamic recurrent neural networks for system identification and control," *IEE Proc. Control Theory Appl.*, vol. 142, pp. 307 - 314., 1995.

9. Appendix

T1. Simple HNN with 5 neurons

TEST	Seed	train error	test error	Epochs
1	4534	2.435E-04	1.625E-03	300
2	456	1.591E-04	3.465E-03	300
3	1514	3.994E-04	2.058E-03	300
4	1534	4.231E-05	6.959E-03	300
5	4210	9.162E-04	9.884E-03	300
6	53	1.059E-03	7.134E-03	300
7	2731	1.877E-04	7.421E-04	300
8	1102	1.054E-04	3.125E-03	300
9	273	2.633E-03	1.165E-02	300
10	666	1.681E-03	9.343E-03	300
	Aver	7.426E-04	5.599E-03	300

T2. HNN with 3 Hidden neurons

TEST	Seed	train error	test error	Epochs
1	4534	7.016E-05	4.062E-04	300
2	456	3.179E-05	6.462E-02	300
3	1514	2.680E-05	9.659E-04	300
4	1534	4.770E-05	6.611E-04	300
5	4210	7.588E-05	1.910E-03	300
6	53	5.707E-04	3.750E-03	300
7	2731	3.718E-05	6.734E-03	300
8	1102	6.692E-04	8.315E-03	300
9	273	4.215E-05	1.256E-02	300
10	666	5.007E-05	6.255E-03	300
	Aver	1.622E-04	1.062E-02	300

T3. Double HNN with 2 neurons in each network

TEST	Seed	train error	test error	Epochs
1	4534	1.67E-05	1.22E-04	300
2	456	1.47E-05	1.02E-04	300
3	1514	2.03E-05	1.40E-04	300
4	1534	3.86E-05	2.90E-04	300
5	4210	1.47E-05	1.08E-04	300
6	53	2.81E-05	2.30E-04	300
7	2731	2.54E-05	1.94E-04	300
8	1102	2.22E-05	1.59E-04	300
9	273	3.99E-05	2.46E-04	300
10	666	3.46E-05	1.98E-04	300
	Aver	2.55E-05	1.79E-04	300