Functional-Link Net Computing:

Theory, System Architecture, and Functionalities

Yoh-Han Pao and Yoshiyasu Takefuji, Case Western Reserve University, Cleveland, OH 44106

Neurocomputing can support learning relationships, cluster analysis, associative recall, and optimization. Currently, these procedures are based on different algorithms requiring different specialized system architectures.

In this report, we describe a system architecture and a network computational approach compatible with our goal of devising a general-purpose artificial neural-net computer. This approach has been described previously.[1,2] We revisit it here to emphasize the advantages that accrue from simplification of net architecture.

The theoretical basis of our approach is primarily Theorem 2.4 from work by Hornik, Stinchcombe, and White[3] (see the sidebar above right). Specifically, the theorem holds that if f is a mapping $R^r \rightarrow R$ (from r-dimensional space to one-dimensional space), then we can approximate that function arbitrarily well by

$$f = \sum_{j=0}^{q} \beta_j \, G\left(\mathbf{A}_j \cdot \mathbf{x} + b_j\right)$$

where $(\mathbf{A}_j \cdot \mathbf{x} + b_j)$ represents a linear transformation of the input pattern vector $\mathbf{x}$. We can understand the mapping of the above equation in terms of a feed-forward net that must learn both the two sets of weights $\{\beta_j\}$ and $\{\mathbf{A}_j\}$ and thresholds b_j, usually through back propagation of error.

The random-vector version of the functional-link net generates $\{\mathbf{A}_j\}$ and b_j

Theoretical rationalization

Hornik, Stinchcombe, and White[3] questioned whether the many successes reported in the use of multilayer feed-forward nets were due to some deep and fundamental approximation capability of such nets or instead to good fortune and perhaps selective reporting. They investigated the matter and established that standard multilayer feed-forward networks with as few as one hidden layer using arbitrary squashing functions are capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy provided sufficiently many hidden layers are available. In that sense, multilayer feed-forward networks are universal approximators.

The following definition and theorems from Hornik, Stinchcombe, and White's work led to one result of principal interest to us. Their Definition 2.4 states that for any measurable function $G(\cdot)$ mapping R to R and $r \in N$, let $\Sigma\Pi^r(G)$ be the class of functions

$$\left\{ f: R^r \rightarrow R : f(x) = \sum_{j=1}^{q} \beta_j \cdot \prod_{k=1}^{l_j} G\left(A_{jk}(x)\right),\ x \in R^r, \beta_j \in R, A_{jk} \in A^r, l_j \in N, q = 1,2,\ldots \right\}$$

where A^r is the set of all affine functions from R^r to R. For the special case of $l_j = 1$, we have the Σ networks.

The theorems of principal interest to us are Theorems 2.2 and 2.4. Theorem 2.2 applies in the special case of $G(\cdot)$ being a squashing function $\psi(\cdot)$ such as the sigmoid activation function, but a similar result holds if ψ is any measurable function such that it can be used to approximate some squashing function on compact sets.

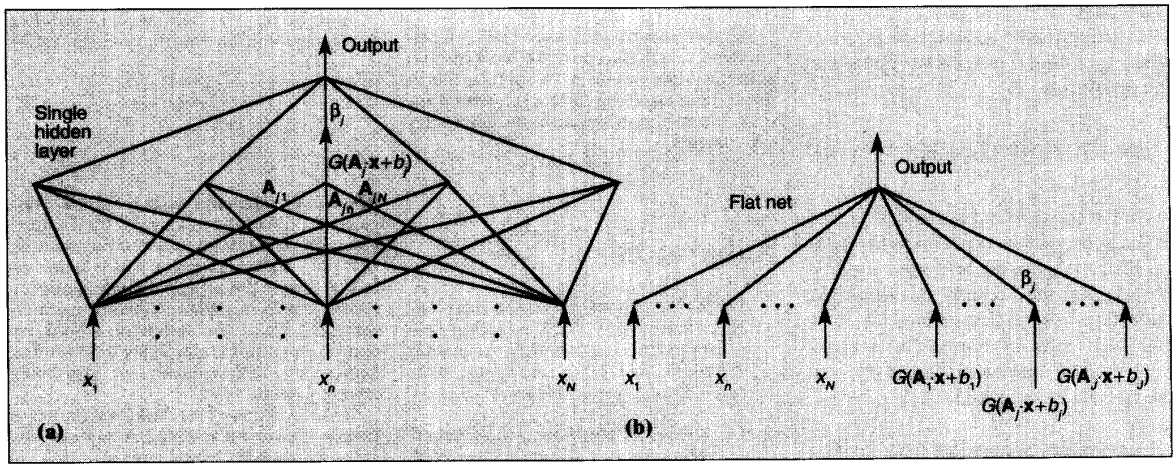

Figure 1. Comparison of hidden-layer net (a) and functional-link net (b) architectures.

Theorem 2.2 states that for every continuous nonconstant function G, every r, and every probability measure μ on (R^r, B^r), $\Sigma\Pi^r(G)$ is ρ_μ dense in M^r, where μ is a probability measure taken for convenience to describe the relative frequency of occurrence of input "patterns" μ, B^r is the Borel field of R^r, and M^r is the set of all Borel measurable functions from R^r to R.

The significance of Theorem 2.2, according to Hornik, Stinchcombe, and White, is that single hidden-layer feed-forward networks can approximate any measurable function arbitrarily well, regardless of the continuous nonconstant function G used, regardless of the dimension of the input space r, and regardless of the input space environment μ. In other words, a $\Sigma\Pi$ network can approximate as closely as necessary the mapping from R^r to R, as described by any given function f.

Theorem 2.4 states that for every squashing function ψ, every r, and every μ on (R^r, B^r), $\Sigma^r(\psi)$ is uniformly dense on compacta in C^r and ρ_μ dense in M^r (where C^r is the set of continuous functions from R^r to R, and C^r is a subset of M^r). In other words, Σ networks are also universal approximators.

More specifically, if a function f is a mapping from R^r to R, then that function can be approximated arbitrarily well by

$$\left(f = \sum_{j=1}^{q} \beta_j \, G(\mathbf{A}_j \mathbf{x} + b_j) \right)$$

This approximator (the first equation in the main text) can be implemented in terms of a single hidden-layer feed-forward net with all weights A_j and β_j to be learned (as well as the thresholds) with back propagation, or it can be implemented with a flat net, as described in the text.

randomly, and must learn only β_j. This results in a flat-net architecture for which only weights $\{\beta_j\}$ must be learned. Learning is in the nature of quadratic optimization and is extremely rapid.

In this report, we illustrate the functionalities of supervised learning and optimization and briefly mention cluster analysis and associative recall.

Supervised learning. Figure 1 compares the net architectures of the single hidden-layer net and the random-vector version of the functional-link net. In the functional-link net, the vectors $\mathbf{A}_j$ and thresholds b_j are generated randomly, not learned.

Figure 2 compares the performances of a single hidden-layer net and a functional-link net in an inferring network description of a two-dimensional surface, given only 24 examples.[4] The actual two-dimensional surface is made up of two Gaussiaus, as Figure 2a shows. Data fed to the networks consisted of 24 samplings of the surface. Acting on that data, a back-propagation net with 48 hidden-layer nodes was able to learn the

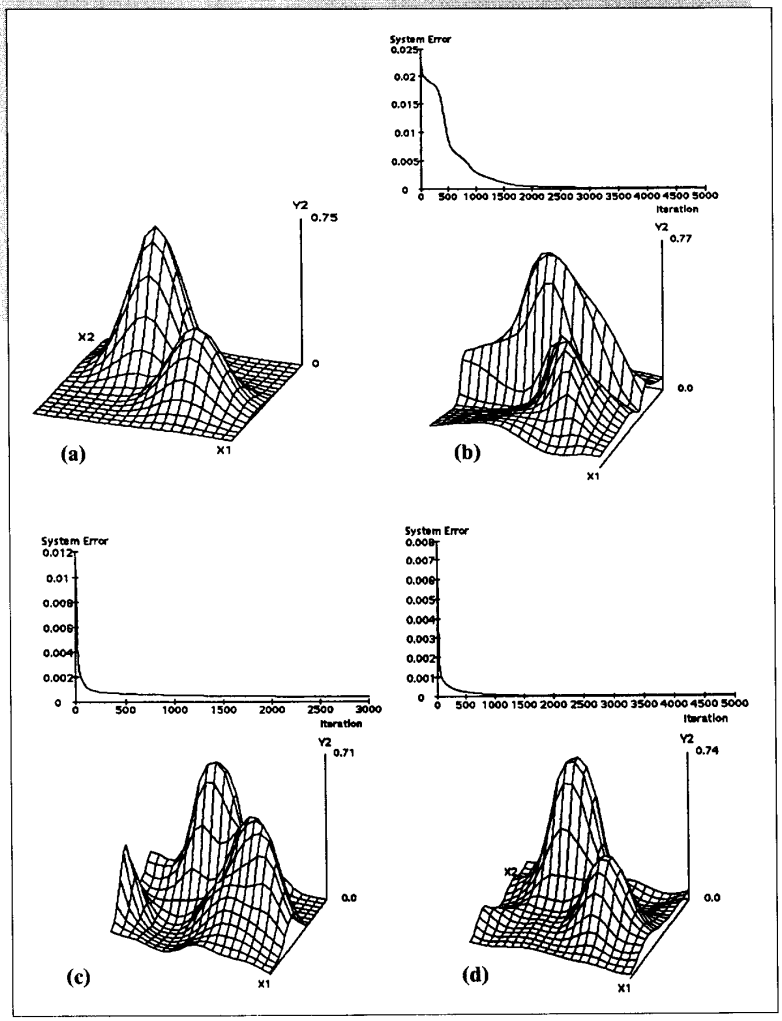

Figure 2. Comparative evaluation of learning rates and approximation capabilities: (a) surface to be learned, consisting of two Gaussians; (b) back-propagation net with 48 nodes in the hidden layer; (c) functional-link flat net with 48 enhancement nodes; (d) functional-link flat net with 200 enhancement nodes. Learning time in (d) is 0.14 that of (a).

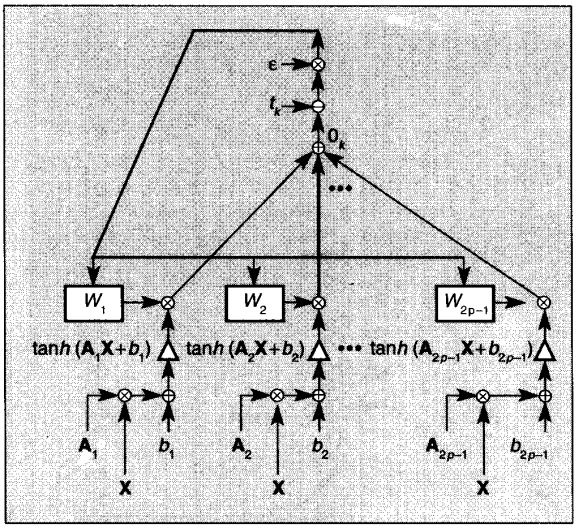

Figure 3. System architecture of the functional-link net.

surface to the extent shown in Figure 2b. The functional-link nets with 48 and 200 enhancement nodes achieved better approximations with fewer iterations in less time (Figures 2c and 2d).

System architecture. The random-vector version of the functional-link net is well suited to large-scale hardware implementation. Figure 3 schematically illustrates a network system architecture.

Optimization. The functional-link net can support many different functionalities with the same system architecture and with only minor changes in the algorithm. Figure 4 shows the use of such a net for dealing with the bipartite subgraph optimization problem.[5]

For the original 10-vertex, 27-edge graph shown in Figure 4a, the task is to construct a bipartite subgraph with the least number of edges removed. The solution shown in Figure 4b has 18 remaining edges. In such a subgraph, no member of one set is connected directly to another vertex of the same set. The objective is to find a bipartite subgraph with the maximum possible number of edges embedded or the least number removed.

The functional-link net is not shown, but in the input vector each vertex is represented by one neuron of the input layer, which is then augmented. The state of the ith vertex, whether it belongs to one subset or to another, is represented by $V_i = 1$ or $V_i = 0$. The input/output function of the ith McCulloch-Pitts neuron is given by $V_i = 1$ if $x_i >$ threshold, 0 otherwise.

The functional-link net is used to learn the unknown Lyapunov function for the bipartite subgraph problem. For each input pattern, let p be an input pattern. The net adjusts the weights β_j to minimize the system error

$$E = \sum_p E_p = \sum_p \left(\text{net}_p - \text{penalty}_p\right)^2$$

where penalty_p is the number of edges removed and is the target output, and net_p is the actual output of the functional-link net.

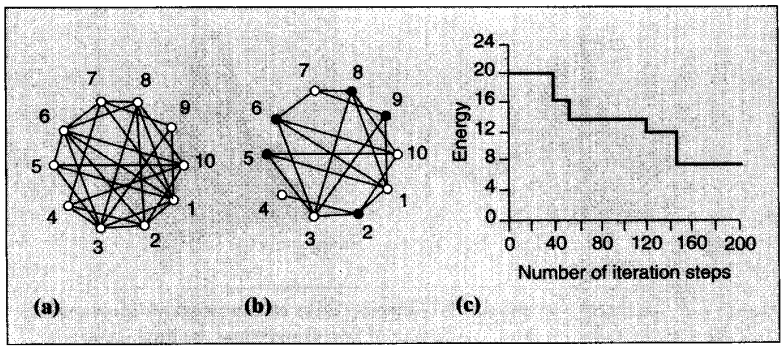

Figure 4. Solution and energy transitions for the bipartite subgraph problem: (a) original graph; (b) one solution with 18 edges embedded; (c) evolution of the optimization.

Overview of algorithms. For *mapping* or *supervised learning*, inputs are enhanced to $\mathbf{x}_p$ with elements $(x_{p1}, x_{p2}, \ldots, x_{pj}, \ldots, x_{pJ})$. The target outputs are t_{pk} and the weights are w_{kj}. For functional-link nets, the outputs t_k can be treated independently of each other. Therefore, we need only consider one output t and weights β_j. All other outputs are treated in the same manner. Initially, the algorithm assigns the weight β_j random values. It calculates the output o_p linearly as $o_p = \Sigma \beta_j x_{pj}$. For each input pattern the changes in the weights are taken to be

$$\Delta\beta_{pj} = \eta(t_p - o_p)x_{pj}$$

The changes are calculated for all the patterns in the training set, and after each such presentation the weights are updated according to

$$\beta_j(k+1) = \beta_j(k) + \sum_p \Delta\beta_{pj}$$

Updating is continued until the values of the weights w_j do not change significantly. The value of the parameter η may be increased as $(t_p - o_p)$ decreases.

For *self-organization* or *clustering*, input patterns $\mathbf{y}_p$ are enhanced to pattern $\mathbf{x}_p$ and fed to the net one at a time. Initially, there is no cluster prototype. Pattern $\mathbf{x}_1$ is therefore a cluster prototype with link weights $b_j = x_{1j}$. The algorithm introduces pattern $\mathbf{x}_2$ to the net and evaluates the distance from $\mathbf{x}_2$ to $\mathbf{x}_1$ according to the expression

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sum_i \left| x_{1_i} - x_{2_i} \right|$$

If $d(\mathbf{x}_1, \mathbf{x}_2)$ < cluster radius, then $\mathbf{x}_2$

belongs to the same cluster as $\mathbf{x}_1$ and the cluster prototype value b_j is updated.

In general, after k cluster centers have been formed, a new pattern is assigned to a particular cluster k if $d(\mathbf{x}, \mathbf{x}_k) < d(\mathbf{x}, \mathbf{x}_m)$ for $m = 1, 2, \ldots$, and $m \neq k$, and if $d(\mathbf{x}, \mathbf{x}_k) <$ cluster radius. Otherwise, a new cluster is formed. At all times, each cluster knows how many patterns are members of the cluster, and the cluster prototype is updated accordingly to

$$b_j(n+1) = \frac{n}{n+1} b_j(n) + \frac{1}{n+1} x_j$$

where n is the number of patterns already associated with that cluster.

For *associative recall*, each pattern $\mathbf{y}_p$ may be enhanced to $\mathbf{x}_p$ form. This generally results in higher memory capacity. For a set of stored patterns $\mathbf{x}_p$, the Lyapunov function is

$$E(\mathbf{x}) = -\sum_p e^{-\alpha/2} (\mathbf{x}_p - \mathbf{x})^t (\mathbf{x}_p - \mathbf{x})$$

For any cue $\mathbf{x}$, the elements x_j are changed repeatedly according to

$$\begin{aligned} \Delta x_j &= -\eta \frac{\partial E}{\partial x_j} \\ &= -\alpha\eta \sum_p (x_{pj} - x_j) e^{-\alpha/2 \sum_j (x_{pj} - x_j)^2} \\ &= \alpha\eta \sum_p (x_{pj} - x_j) E(\mathbf{x}) \end{aligned}$$

Conclusion. In the functional-link net we try to separate the issues of representation and search, the latter to be understood in terms of search for a goal or a solution. The random-vector version of the functional-link net is eminently suitable for realization with simple hardware network architectures. Numerous and significant advantages accrue from using a flat net, including rapid quadratic optimization in the learning of weights, simplification in hardware as well as in computational procedures, and uniform system architecture for all four functionalities. ■

References

1. Y.H. Pao, *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Reading, Mass., 1989, pp. 197-222.
2. Y.H. Pao and T. Goraya, "Neural Net Computation of Solutions to Ill-Formed Optimization Problems," Center for Automation and Intelligent Systems Research report, Case Western Reserve Univ., Cleveland, Ohio, 1991.
3. K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, Vol. 2, No. 5, 1989, pp. 359-366.
4. N. Mues, "Evaluation of Characteristics of the Random-Vector Functional-Link Net in Supervised Learning and Optimization Tasks," CAISR report, Case Western Reserve Univ., Cleveland, Ohio, 1991.
5. Y. Takefuji, *Neural Network Parallel Computing*, Kluwer Publishing, Hingham, Mass., 1992, pp. 1-26.

Yoh-Han Pao is the George S. Dively Distinguished Professor of Engineering at Case Western Reserve University. He has served as chair of CWRU's Electrical Engineering Department (1969-77), as director of the National Science Foundation's Electrical, Computer, and System Engineering Division (1978-80), and as founding director of CWRU's Center for Automation and Intelligent Systems Research. He is also cofounder and president of AI Ware Inc., Cleveland, Ohio.

Yoshiyasu Takefuji is on the faculty of the Electrical Engineering Department at Case Western Reserve University. His research focuses on neural network parallel computing for solving real-world problems. He is the coauthor of two books, *Digital Circuits* (Ohm-Sha Publishers, 1984) and *Neural Network Computing* (Baifukan Publishers, 1991), and the author of a third, *Neural Network Parallel Computing* (Kluwer Publishers, 1992).