REFERENCES

THAI

จุฑาทิพย์ เพชรเชิดศักดิ์, "การใช้ข่ายงานนิวรัลชนิดป้อนไปข้างหน้าแบบหลายชั้นสำหรับการระบุระบบ การประมาณฟังก์ชัน และการควบคุมขั้นสูง," *วิทยานิพนธ์ ภาควิชาวิศวกรรมเคมี* (2542)

ชมพูนุท พิภพลาภอนันต์ ; "เพอร์เวเพอเรชันของของผสมน้ำ-บิวทานอล," *วิทยานิพนธ์ ภาควิชาวิศวกรรมเคมี*, จุฬาลงกรณ์มหาวิทยาลัย (2537)

ณรงค์พันธุ์ รัตนปนัดดา; "การสร้างแบบจำลองฟัซซี และการออกแบบตัวควบคุมเครื่องปฏิกรณ์เพอร์เวเพอเรทีพเมมเบรน," *วิทยานิพนธ์ ภาควิชาวิศวกรรมเคมี*, จุฬาลงกรณ์มหาวิทยาลัย (2546)

นงลักษณ์ พลรักษา ; "การทำนายค่าอัตราการไหล และความหนาแน่นของโพลิเมอร์โดยใช้ข่ายงานนิวรอน, " *วิทยานิพนธ์ ภาควิชาวิศวกรรมเคมี*, จุฬาลงกรณ์มหาวิทยาลัย (2543)

ไพศาล กิตติศุภกร, *เอกสารคำสอนวิชา 2105-619 การควบคุมอัตโนมัติขั้นสูง*, ภาควิชาวิศวกรรมเคมี จุฬาลงกรณ์มหาวิทยาลัย (2543)

ลลิตา อัตนโถ ; "การประยุกต์ใช้กระบวนการเพอร์เวเพอเรชันเพื่อการแยกน้ำออกจากสารสกัดหยาบจากใบขี้เหล็ก," *วิทยานิพนธ์ ภาควิชาวิศวกรรมเคมี*, จุฬาลงกรณ์มหาวิทยาลัย (2543)

อรลัดดา มูลศาสตรสาทร; "การออปติไมซ์ และควบคุมเครื่องปฏิกรณ์เพอร์เวเพอเรทีฟเมมเบรน," *วิทยานิพนธ์ ภาควิชาวิศวกรรมเคมี*, จุฬาลงกรณ์มหาวิทยาลัย (2545)

Aziz, N., Hussian, M.A. and Mujtaba, I.M.; "Performance of Different Types of Controllers in Tracking Optimal Temperature Profiles in Batch Reactors," *Comput. Chem. Eng.*, **24**, 1069 (2000)

Aziz, N., Hussain, M. A. and Mujtaba, L.M. *Process Systems Engineering 2003*, Elsevier, 708 (2003)

Baker, W. R., *Membrane Technology and Applications*., McGraw-Hill, (2000)

Bhat N.V. and T.J. McAvoy.; "Use of Neural Nets for Dynamic Modeling and Control of Chemical Process Systems." *Computer Chem Eng.*, **14** , 573 (1990)

Chitra, S. P. ;" Neural net application in chemical engineering," *AI Expert* , **November** , 20 (1992)

Chovan, T., Catfolis, T. and Meert, K. ; "Neural network architecture for process control based on the RTRL algorithm," AIChE J. **42(2),** 493 (1996)

David, M, O., Gref, R. and Nguyen, Q, T.; "Pervaporation-esterification coupling, I. Basic kinetic model," *Trans. Inst. Chem. Res.* , **28**, 335(1991)

Dirion, L., et al.; "Design of a neural controller by inverse modeling," *Computers Chem. Engng.* , **19S**, S797 (1995)

Emmanouilides, C. and Petrou, L. : " Identification and control anaerobic digesters using adaptive, onlin trained neural networks," *Computers Chem. Engng.* , **21 (1),** 113 (1997)

Feng X., and Huang, M.; Studies of a membrane reactor : esterification facilitated by pervaporation. *Chem. Eng. Sci.* , **51**,4763(1996)

Fredic M. Ham and Ivica Kostanic, *Principles of Neurocomputing for Science & Engineering*, McGraw-Hill Companies, Inc., (2000)

Gontarski C.A., Rodrigues P.R, Mori M. and Prenem L.F.; " Simulation of an industrial wastewater treatment plant using artificial neural networks," *Computers Chem. Engng.* , **24**, 1719 (2000)

Hussain, M. A., Kittisupakorn, P. and Daosud W.; " Implementation of neural-network-based inverse-model control strategies on an Exothermic reactor," *Science Asia*, **27**, 41 (2001)

Keurentjes, J, T. F,, Janssen, G. H. R., and Gorissen, J. J. ; "The esterification of tartaric acid with ethanol: kinetics and shifting the equilibrium by means of pervaporation," *Chem. Eng. Sci.,* **49** , 4681(1994)

Kiitisupakorn, P.,Hussain, M. A. and Petcherdsak J.; " Studies on the use of neural networks in nonlinear control strategies," *J. Chem. Engng. Jpn.* , **34 (4)**, 453 (2001)

Kittisupakorn, P., Moolasartsatorn, O., Arpornwichanop, A. and Kaewpradit, P., *Process Systems Engineering 2003*, Elsevier, 888 (2003)

Kurtanjek, Z. ," Modeling and control by artificaial neural network in biotechnology," *Computers Chem. Engng.* , **18 (suppl.)** ,S627 (1994)

Lanouette R., Thibault J. and Jacques L. Valade, "Process modeling with neural networks using small experimental datasets," *Computers Chem. Engng.* , **23**, 1167 (1999)

Lee, M. and Park, S. ;" A new scheme combining neural feedforward control with model predictive control," *AIChE J.* , **38(2),** 193 (1992)

Loh, A. P., Looi, K. O. and Fong, K. F. ; "Neural network modeling and control strategies for a pH process," *J. Proc. Contr.* **5(6),** 355 (1995)

Lou, K. N. and Perez, R. A. ; " Anew system identification technique using kalman filtering and multilayer neural networks," *Artificial Intelligence in Engineering.* , **10**, 1(1996)

Lui, Q., Zhang, Z., and Chen H. ; "Study on the coupling of esterification with pervaporation," *J. Mem. Sci.* ,**182**, 173 (2001)

MacMurray, J. C. and Himmelblau, D. M. ; " Modeling and control of paced distillation column using artificial neural networks," *Computers Chem. Engng.*, **19(10),** 1077 (1995)

Matouq, M., Tagawa, T., and Goto, S.; "Combined process for production of methyl tert-butyl ether from tert-butyl alcohol and methanol," *J. Chem. Eng. Jpn.*, **27**, 302(1994)

Nahas, E. P. Henson, M. A. and Seborg, D. E. ; " Nonliinear internal model control strategy for neural network models," *Computers Chem. Engng.*, **16**, 1039 (1992)

Narendra, K. S. annd Parthasarathy, K.; "Identification and control of dynammical systems using neural networks," *IEEE Trans. Neural Network* ,**1** , 4(1990)

Nikolaou, M. and Hanagandi, V. ; "Control of nonlinear dynamical systems modeled by recurrent neural networks," *AIChE J.* , **39(11),** 1890 (1993)

Nikravesh, M., Farell, A. and Stanford, T. G. ; "Model identification of nonlinear time variant processes via artificial neural network," *Computers Chem. Engng.,* **20 (11)**,1277 (1996)

Okamoto, K. I., Yamamoto, M., Otoshi, Y., Semoto, T., Yano, M., Tanaka K. and Kita, H. ; "Pervaporation-aided esterification of oleic acid," *J. Chem. Eng. Jpn.*, **26,** 457(1993)

Pao,Y.H., Phillips, S. M. and Sobajic , D. J. ; "Neural Net Computing and the Intelligent Control of Systems," *Int. J. Contr.*, **56**, 263 (1992)

Pollard, J. F., Broussard, M. R., Garisson, D. B. and San, K. Y.; " Process identification using neural networks," *Computers Chem. Engng.* , **16**, 253 (1992)

Psichogios, D. M. and Ungar, L. H. ; " Direct and indirect model based control using artificial neural networks," *Ind. Engng. Chem. Res.,* **30**, 2564 (1991)

Ramchandran, S. and Rhinehart, R. R. ; "A very simple structure for neural network control of distillation," *J. Proc.Contr.* , **5(2)**, 115 (1995)

Scott, G. M. and Ray, W. H.; "Experience with model based controllers based on neural network process models," *J. Proc. Contr.* **3(3),** 179 (1993)

Shene C., Diez C. and Bravo S.; "Neural network for the prediction of the state of Zymomonas mobilis CP4 batch fermentation," *Computers Chem. Engng.* , **23**, 1097 (1999)

Syu, M. and Chang, J. B. ; " Recurrent backpropagation neural network adaptive control of Penicillin Acylase Fermentation by Arthrobactor viscosus, " *Ind. Engng. Chem. Res.* , **36**, 3756 (1997)

Temeng, K. O. , Schnelle, P. D. and McAvoy, T. J. ; " Model predictive control of an industrial packed bed reactor using neural networks," *J. Proc. Contr.* **5 (1),** 19 (1995)

Thompson, M. L. and Kramer, M. A. ; "Modeling chemical processes using prior knowledge and neural networks," *AIChE J.* **40 (8)** ,1328 (1994)

Tsen, A. Y., et al. ; "Predictive control of quality in Batch polymerization using hybrid ANN models," *AIChE J.* , **42(2)** , 455 (1996)

Ungar. L. H., Powell, B. A. and Kamens S. N. ;" Adaptive networks for fault diagnosis and control," *Computers Chem. Engng.* **14(4/5),** 561(1990)

VanCan, H. J. et al. ; "Design and real time testing of a neural model predictive controller for a nonlinear system," *Chem. Engng. Sci.,* **50(15)** , 2419 (1995)

Waldburger, R. M., and Widmer, F. ; "Membrane reactions in chemical production processes and the application to pervaporation-assisted esterification," *Chem. Eng. Technol.* ,**19** ,117 (1996)

You, Y. and Nikolaou, M. , "Dynamic process modeling with recurrent neural networks," *AIChE J.,* **39(10),** 1654 (1993)

Zhu.: Minet, R. G.; and Tsotsis, T. T. ; "A continuous pervaporation membrane reactor for study of esterification reactions using a composite polymeric/ceramic membrane," *Chem. Eng. Sci.,* **51**,4103

APPENDIX

APPENDICES

APPENDIX A

NEURAL NETWORK TOOLBOX

MATLAB abbreviated from MATrix LABoratory is a technical computing environment for high-performance numeric computation and visualization. MATLAB integrates numerical analysis, matrix computation, signal processing, and graphics in an easy-to-use environment where problems and solutions are expressed just as they are written mathematically - without traditional program meaning. MATLAB 's functionality and versatility with the addition of optional application-specific toolboxes can be extended. Toolboxes are comprised of suites of MATLAB functions (M-files) written by world-class authorities on each of the respective topics. These toolboxes cover a variety of disciplines as illustrated by the following list:

- Matlab Toolbox
- Signal Processing Toolbox
- Optimization Toolbox
- Neural Network Toolbox
- Fuzzy Logic Toolbox
- Model Predictive Control
- Control System Toolbox,
- System Identification Toolbox

A.1 Neural Network Toolbox

In Neural Network Toolbox, it provides many useful toolbox functions supporting programmers to do their programs easier. Four main toolbox functions used for neural network programming are as follow:

- Network creation functions
- Weight/bias initialization functions
- Training functions
- Performance functions

A.2 Training Functions

Gradient descent and gradient descent with momentum are backpropagation training algorithm. However, these two methods are often too slow for practical problems. Consequently, high performance algorithms which can converge from ten to one hundred times faster than those two algorithms.

These faster algorithms fall into two main categories. The first category uses heuristic techniques, which were developed from an analysis of the performance of the standard steepest descent algorithm. One heuristic modification is the momentum technique. The others, which are more heuristic technique, are variable learning rate backpropagation, TRAINGDA, and resilient backpropagation, TRAINRP.

The second category of fast algorithms uses standard numerical optimization techniques. Three types of numerical optimization techniques for neural network training: conjugate gradient (TRAINCGF, TRAINCGP, TRAINCGB, TRAINSCG), quasi-Newton (TRAINBFG, TRAINOSS), and Levenberg-Marquardt (TRAINLM) are provided.

A.2.1 Gradient Descent Learning Rule

1) Variable Learning Rate (TRAINGDA, TRAINGDX)

With standard steepest descent, the learning rate is held constant throughout training. The performance of the algorithm is very sensitive to the proper setting of the learning rate. If the learning rate is set too high, the algorithm may oscillate and become unstable. If the learning rate is too small, the algorithm will take too long to converge. It is not practical to determine the optimal setting for the learning rate before training, and, in fact, the optimal learning rate change during the training process, as the algorithm moves across the performance surface.

The performance of the steepest descent algorithm can be improved if we allow the learning rate to change during the training process. An adaptive learning will attempt to keep the learning step size as large as possible while keeping learning stable. The learning rate is made responsive to the complexity of the local error surface.

2) Resilient Backpropagation (TRAINRP)

Multilayered networks typically use sigmoid transfer functions in the hidden layers. These functions are often called squashing functions, since they compress an infinite input range into a finite output range, Sigmoid functions are characterized by the fact that their slope must approach zero as the inputs get large. This causes a problem when using steepest descent to train a multilayered network with sigmoid functions, since the gradient can have a very small magnitude, and therefore cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.

The purpose of the resilient backpropagation (Rprop) training algorithm is to eliminate these harmful effects of the magnitudes of the partial derivatives. Only the sign of the derivative is used to determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value.

Rprop is generally much faster than the standard steepest descent algorithm. It also has the nice property that it requires only a modest increase in memory requirements. We do need to store the update values for each weight and bias, which is equivalent to storage of the gradient.

A.2.2 Conjugate Gradient Algorithms

The basic backpropagation algorithm adjusts the weights in the steepest descent direction (negative of the gradient). This is the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms a search is performed along the conjugate directions, which produces generally faster convergence than steepest descent directions. In this section, four different variations of conjugate gradient algorithm are presented.

1) Flecher-Reeves Update (TRAINCGF)

All of the conjugate gradient algorithms start out by searching in the steepest descent direction (negative of the gradient) on the first iteration.

$$Po = -go$$

A line search is then performed to determine the optimal distance along the current search direction:

$$X_{k+1} = X_k + \alpha_k p_k$$

Then the next search direction is determined so that it is conjugate to various search directions. The general procedure for determining the new search direction is to combine the new steepest descent direction with the previous search directions

$$P_k = -g_k + \beta_k P_{k-1}$$

The various versions of conjugate gradient are distinguished by the researcher in which the constant Pk is computed. For the Fletcher-Reeves update procedure is

$$\beta_k = \frac{g^T_{k-1} g_k}{g^T_{k-1} g_{k-1}}$$

This is the ratio of the norm squared of the current gradient to the mean squared of the previous gradient.

2) Polak-Ribiere Update (TRAINCGP)

Another version of the conjugate gradient algorithm was proposed by Polak and Ribire. As with the Fletcher-Reeves algorithm, the search direction at each iterationis determined by

$$P_k = -g_k + \beta_k P_{k-1}$$

For the Polak-Ribiere Update, the constant *Pk* is computed by

$$\beta_k = \frac{g^T_{k-1} g_k}{g^T_{k-1} g_{k-1}}$$

This is the inner product of the previous change in the gradient with the current gradient divided by the norm squared of the previous gradient. The TRAINCGP routine has performance similar to TRAINCGF. It is difficult to predict which algorithm will perform best on a given problem. The storage requirements for Polak- Ribiere (four vectors) are slightly larger than for Fletcher-Reeves (three vectors).

3) Powell-Beale Restarts (TRAINCGB)

For all conjugate gradient algorithms, the search direction will be periodically reset to the negative of the gradient. The standard reset point occurs when the number of iterations is equal to the number of network parameters (weights and biases), but there are other reset methods which can improve the efficiency of training. One such reset methods was proposed by Powell, based on an eariler version proposed by Beale. For this technique we will restart if there is very little orthogonality left between the current gradient and the previous gradient. This is tested with the following inequality:

$$|g^T_{k-1} g_k| \geq 0.2||g_k||^2$$

If this condition is satisfied, the search direction is reset to the negative of the gradient. The TRAINCGB routine has performance which is somewhat better than TRAINCGP for some problems, although performance on any given problem is difficult to predict. The storage requirements for the Powell-Beale algorithm (six vectors) are slightly larger than for Polak-Ribiere (four vectors).

4) Scaled Conjugate Gradient (TRAINSCG)

Each of the conjugate gradient algorithms which we have discussed so far require a line search at each iteration. This line search is computationally expensive, since it requires that the network response to all training inputs be computed several times for each search. The scaled conjugate gradient algorithm (SCG) was designed to avoid the time consuming line search. This algorithm is too complex to explain in a few lines, but the basic idea is to combine the model-trust region approach, which is used in the Levenberg-Marquardt algorithm describe later, with the conjugate gradient approach. The TRAINSCG routine may require more iteration to converge than the other conjugate gradient algorithms, but the number of computations in each iteration is significantly reduced because no line search is performed. The storage requirements for the scaled conjugate gradient algorithm are about the same as those of Fletcher-Reeves.

A.2.3 Quasi-Newton Algorithms

1) BFGS Algorithm (TRAINBFG)

Newton's method is an alternative to the conjugate gradient methods for fast optimization. The basic step of Newton's method is

$$X_{k+1} = X_k - A_k^{-1} g_k$$

where A_k, is the Hessian matrix (second derivatives) of the performance index at the current values of the weights and biases. Newton's method often converge faster than conjugate gradient methods. Unfortunately, it is complex and expensive to compute the Hessian matrix for feedforward neural networks. There is a class of algorithms that are based on Newton's method but which do not require calculation of second derivatives. These are called quasi-Newton (or secant) methods. They update an approximate Hessian matrix at each iteration of the algorithm. The update is computed as a function of the gradient. The algorithm has been implemented in the TRAINBFG routine.

2) One Step Secant Algorithm (TRAINOSS)

Since the BFGS algorithm requires more storage and computation in each iteration than the conjugate gradient algorithms, there is need for a secant approximation with smaller storage and computation requirements. The one step secant (OSS) method is an attempt to bridge the gap between the conjugate gradient algorithms and the quasi-Newton (secant) algorithms. This algorithm does not store the complete Hessian matrix; it assumes that at each iteration the previous Hessian was the identity matrix. This has the additional advantage that the new search direction can be calculated without computing a matrix inverse

A.2.4 Levenberge-Marquardt (TRAINLM)

Like the quasi-Newton methods, the Levenberge-Marquardt algorithm was designed to approach second order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares (as is typical in training feedforward networks), then the Hessian matrix can be approximated as

$$H = J^T J$$

and the gradient can be computed as

$$g = J^T e$$

where J is the Jacobian matrix, which contain first derivatives of the network errors with respect to the weights and biases, and e is a vector of network errors. The Jacobian matrix can be computed through a standard bachpropagation technique that is much less complex than computing the Hessian matrix.

The Levenberg-Marquardt algorithm uses this approximation to the Hessian matrix in the following Newton like update:

$$X_{k+1} = X_k - [J^T J + \mu I]^{-1} J^T e.$$

When the scalar μ is zero, this is just Newton's method, using the approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. Newton 's method is faster and more accurate near an error minimum, so the aim is to shift towards Newton 's method as quickly as possible. Thus, μ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way, the performance function will always be reduced at each iteration of the algorithm.

A.3 Speed and Memory Comparison of Training Functions

It is very difficult to know which training algorithm will be the fastest for a given problem. It will depend on many factors, including the complexity of the problem, the number of data points in the training set, the number of weights and biases in the network, and the error goal. In general, on networks which contain up to a few hundred weights the Levenberg-Marquardt algorithm will have the fastest convergence. This advantage is especially noticeable if very accurate training is required.

The quasi-Newton methods are often the next fastest algorithms on networks of moderate size. The BFGS algorithm does require? storage of the approximate Hessian matrix, but is generally faster than the conjugate gradient algorithms. Of the conjugate gradient algorithms, the Powell-Beale procedure requires the most storage, but usually has the fastest convergence. Rprop and the scaled conjugate gradient algorithm do not require a line search and have small storage requirements. They are reasonably fast, and are very useful for large problems. The variable learning rate algorithm is usually much slower than the other method, and has about the same storage requirements as Rprop, but it can still be useful for some problems.

For most situations, Levenberg-Marquardt algorithm is recommended to try first. If this algorithm requires too much memory, then try the BFGS algorithm, or one of the conjugate gradient methods. The Rprop algorithm is also very fast, and has relatively small memory requirements.

For most situations, Levenberg-Marquardt algorithm is recommended to try first. If this algorithm requires too much memory, then try the BFGS algorithm

TRAINBFG, or one of the conjugate gradient methods. The Rprop algorithm TRAINRP is also very fast, and has relatively small memory requirements.

APPENDIX B

BACKPROPAGATION ALGORITHM

B.1 Conclusion of the Backpropagation Algorithm

- Weight Initialization

Set all weights and node thresholds to small random numbers. Note that the node threshold is the negative of the weight from the bias unit (whose activation level is fixed at 1).

- Calculation of activation function

1. The activation level of an input unit is determined by the instance presented to the network.

2. The activation level O_j of a hidden and output unit is determined by

$$O_j = F(\Sigma W_{ji} O_i + \theta j)$$

where W_{ji} is the weight from an input O_i , θj is the node threshold, and F is the sigmoid function:

$$F(a) = \frac{1}{1 + e^{-a}}$$

- Weight Training

1. Start at the output units and work backward to the hidden layers recursively. Adjust weights by

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji}$$

where $W_{ji}(t)$ is the weight from unit i to unit j at time t (or the iteration) and ΔW_{ji} is the weight adjustment.

2. The weight change is computed by

$$\Delta W_{ji} = \eta \delta_j O_i$$

where η is a trial-independent learning rate ($0<\eta<1$, e.g., 0.3) and δ_j is the error gradient at unit j. Convergence is sometimes faster by adding a momentum term:

$$W_{ji}(t+1) = W_{ji}(t) + \eta \delta_j O_i + \alpha [W_{ji}(t) - W_{ji}(t-1)]$$

where $0<\alpha<1$.

3. The error gradient is given by:

- For the output units:

$$\delta_j = O_j (1 - O_j)(T_j - O_j)$$

where T_j is the desired (target) output activation and 0, is the actual output activation at output unit j.

- For the hidden unit

$$\delta_j = O_j (1 - O_j) \Sigma \delta_k W_{kj}$$

where δ_k is the error gradient at unit k to which a connection points from hidden unit j.

4. Repeat iterations until convergence in terms of the selected error criterion. An iteration includes presenting an instance, calculating activations, and modifying weights.

The name "backpropagation" comes from the fact that the error (gradient) of hidden units are derived from propagating backward the errors associated with output units since the target values for the hidden units are not given. In the backpropagation network, the activation function chosen is the sigmoid function, which compresses the output value into the range between 0 and I. The sigmoid function is advantageous in that it can accommodate large signals without saturation while allows the passing of small signals without excessive attenuation. Also, it is a smooth function so that gradients can be calculated, which are required for a gradient descent search.

B.2 Example of Calculation

To solve the exclusive-or problem, we build a backpropagation network as shown in Figure B.1. The network will be trained on the following instances:

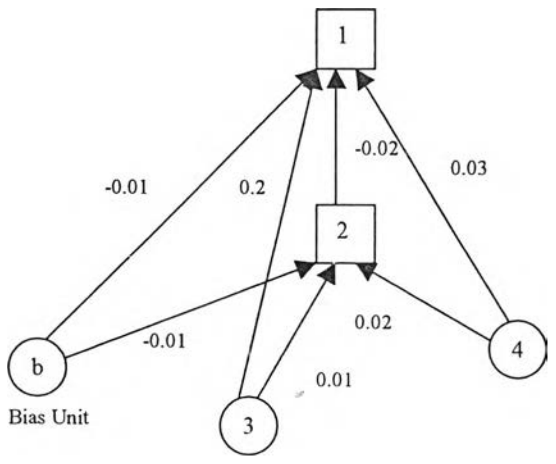

Figure B1 A backpropagation network for learning the exclusive-or function

Inputs	Output
(1,1)	0
(1,0)	1
(0,1)	1
(0,0)	0

The weights are initialized randomly as follows:

$W_{13} = 0.02,\ W_{14} = 0.03,\ W_{12} = -0.02,\ W_{23} = 0.01,$

$W_{24} = 0.02, W_{1b} = -0.01,\ W_{2b} = -0.01$

Calculation of Activation: Consider a training instance with the input vector = (1,1) and the desired output vector =(0)

$O_3 = O_4 = 1$

$O_2 = 1 / [1 + e^{-(1x0.1 + 1x0.2 - 1x0.1)}] = 0.505$

$O_1 = 1 / [1 + e^{-(0.505x\text{-}0.02 + 1x0.2 + 1x0.03 - 1x0.01)}] = 0.505$

Weight Training: Assume that the learning rate $\eta = 0.3$.

$\delta_j = 0.508(1 - 0.508)(0 - 0.508) = -0.127$

$\Delta W_{13} = 0.3 \times (-0.127) \times 1 = -0.038$

$\delta_2 = 0.505(1\text{-}0.505)(-0.127 \times \text{-}0.02) = 0.0006$

The rest of the weight adjustments are omitted. Note that the threshold (which is the negative of the weight from the bias unit) is adjusted likewise. It takes many iterations like this before the learning (training) process stops. The following set of the final weights gives the mean squared error of less that 0.01: $\Delta W_{23} = 0.3 \times 0.0006 \times 1 = 0.002$

$W_{13} = 4.98$, $W_{14} = 4.98$, $W_{12} = -11.30$, $W_{23} = 5.62$,

$W_{24} = 5.62$, $W_{1b} = -2.16$, $W_{2b} = -8.83$

Backpropagation has been applied to classification tasks, speech synthesis from text adaptive robotic control, scoring of bank loan applications, system modeling, data compression, and many others.

VITA

Piyanuch Thitiyassok was born in Bangkok, Thailand, on September 27, 1975, the daughter of Banleng and Pratummas Thitiyasook. She graduated high school from Satri Wittaya School, Bangkok, Thailand and received the degree of Bachelor of Science in Chemistry department from Srinakharinwirot (Prasanmitr) University in 1997. She entered the Graduate School of Chulalongkorn University to pursue the Master of Engineering in Chemical Engineering in 2001.