App. I. Artificial Neural Networks

1 General

Artificial neural networks, or connectionist models as they are sometimes referred to, have been inspired by what is known as the 'brain metaphor'. This means that these models try to copy the capabilities of the human brain into computer hardware or software. The human brain has a number of properties that are desirable for artificial systems (e.g. Schmidt, 1994):

- It is robust and fault tolerant. Even if nerve cells in the brain die (which is known to happen every day), the performance of the brain does not deteriorate immediately.

- It is flexible. This means that the human brain can adjust itself to new situations and can learn by experience.

- It can deal with information that is inconsistent, or contaminated with noise.

- It can handle unforeseen situations by applying knowledge from other domains and extrapolating this to new circumstances.

- It can deal with large amounts of input data and quickly extract the relevant properties from that data.

- It is highly parallel, hence it has a high performance.

Neural network research started in the forties. McCulloch and Pitts (1943) described the logical function of a biological neuron. They described that the transmission of neural signals is an all-or-nothing situation. A neuron fires only, if the cell has been stimulated above a certain threshold. The output signal will, in general, have a constant strength. In their paper, McCulloch and Pitts, described that networks consisting of many neurons might be used to develop the universal Turing machine (a kind of computer described by Turing (1937) that could, in principle, solve all mathematical problems). Research in neural networks was suddenly stopped following a publication by Minsky and Papert (1969). In this paper, it was shown that a relatively simple problem (the so-called XOR-problem) could not be solved by the linear algorithms used at the time. The major breakthrough which re-launched the interest in this technique has been the discovery in the eighties of a non-linear optimisation algorithm overcoming the previous limitations (Rumelhart et. al, 1986).

Neural networks have emerged in the last decade as a promising computing technique which enable computer systems to exhibit some of the desirable brain properties. Various types of networks have been applied successfully in a variety of scientific and technological fields. Examples are applications in industrial process modelling and control, ecological and biological modelling, sociological and economical sciences, as well as medicine (Kavli, 1992). Within the exploration

and production world, neural network technology is now being applied to geologic log analysis (Doveton, 1994) and seismic attribute analysis (Schultz, 1994).

In dGB-GDI neural networks are used for pattern recognition. Three approaches can be recognised in neural network pattern recognition (Lippmann, 1989): supervised training, unsupervised training and combined supervised-unsupervised training. Supervised training approaches require the existence of representative datasets. Unsupervised techniques find structure in the data themselves, thereby extracting the relevant properties. In dGB-GDI Multi-Layer Perceptrons and Radial Basis Function networks are available for the supervised training approach. Unsupervised Vector Quantisers are available in the unsupervised mode. These networks are introduced in the following sections.

2 Multi-layer perceptrons (MLP)

The most general and most widely used neural network model is the 'multi-layer perceptron (MLP)'. The basic building block of this model is the perceptron (Fig. 1), a mathematical analogue of the biological neuron, first described by Rosenblatt (1962).

The mathematical expression of the biological neuron can be written as an activation function A applied to a weighting function W, defined as:

$$W(\mathbf{y}) = \sum_{i=0}^{L} w_i y_i \,, \tag{1.1}$$

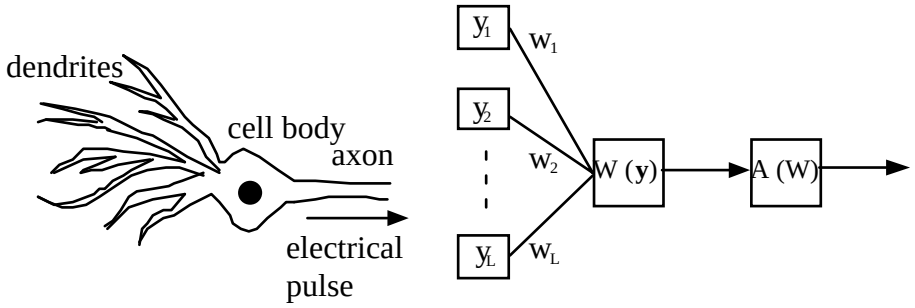

Fig. 1 *A biological neuron and a Perceptron*

where:
$\mathbf{y}$ is the neural network input vector written as y_i with $i = 1,\ldots, L$ and weighting vector w_i with $i = 1,\ldots, L$.

The activation function of the classical perceptron (Fig. 2a) can now be written in the following form:

$$A(W) = \begin{cases} 1 & W > 0 \\ 0 & W \le 0 \end{cases}. \qquad (1.2)$$

In MLPs the binary activation function is often replaced by a continuous function. The most widely used activation function is the sigmoid function (Fig. 2b). This function has the following form:

$$A(W) = \frac{2}{1 + \exp(-W)} - 1. \qquad (1.3)$$

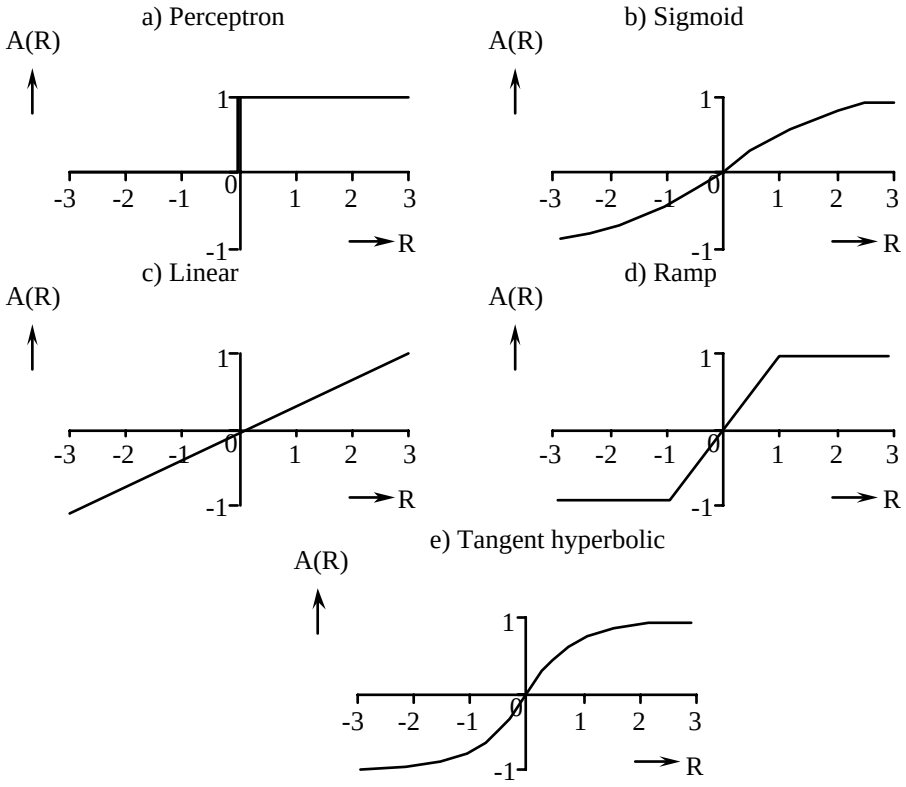

Fig. 2 *Different activation functions for MLP networks as supported in dGB's software. The prime-tangent hyperbolic function was used in this project. This function has the same mathematical expression as the tangent hyperbolic function but the update rules differ (see below).*

Other activation functions supported by the software are the linear, ramp and tangent hyperbolic functions. The linear function (Fig. 2c) is defined as:

$$A(W) = W. \qquad (1.4)$$

The ramp function (Fig. 1.4d) is given by:

$$A(W)=\begin{cases}-1 & W<-1\\ W & -1\le W\le 1.\\ 1 & W>1\end{cases} \tag{1.5}$$

The tangent hyperbolic function (Fig. 2e) is written as:

$$A(W)=\frac{\exp(W)-\exp(-W)}{\exp(W)+\exp(-W)}. \tag{1.6}$$

Two other activation functions are supported in dGB's software: the prime-sigmoid and prime-tangent hyperbolic. These functions have the same mathematical expressions as equations (1.3) and (1.6), respectively. The training algorithm treats the two types of functions differently. For the sigmoid and tangent hyperbolic functions, the derivative is used to update the weighting vector (Rich and Knight, 1991). For the prime-sigmoid and prime-tangent hyperbolic functions an offset is added to the absolute value of the derivative. This is done exclusively to avoid saturation problems during learning, where saturation means that continued learning does not lead to improved network performance. This modified procedure is used to update the weighting vector.

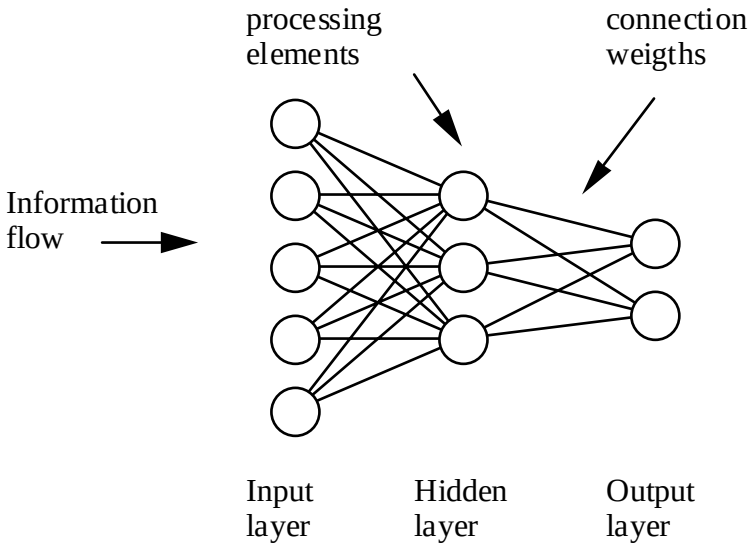

Fig. 3 *Schematic representation of a feed-forward layered neural network, such as a Multi-Layer Perceptron and a Radial Basis Function network.*

In a MLP the perceptrons are organised in layers (Fig. 3). In its simplest form, there are three layers; an input layer, a hidden layer and an output layer. There are no connections between neurons belonging to the same layer. The data flow between the layers is feed-forward. MLPs are trained on a representative dataset. This is a form of supervised learning. Known examples, consisting of input patterns and corresponding output patterns, are repeatedly offered to the network during the training phase. The 'back-propagation', learning, algorithm that is widely used to train this type of network attempts to minimise the error between the predicted network result

and the known output by adjusting the weights of the connections. The algorithm was derived independently by a number of researchers. The modern form of back-propagation is often credited to Werbos (1974), LeCun (1985), Parker (1985) and Rumelhart et. al. (1986). A fast variation of backpropagation is given by Fahlman (1988).

MLPs have two properties of interest: abstraction and generalisation. Abstraction is the ability to extract the relevant features from the input pattern and discard the irrelevant ones. Generalisation allows the network, once trained, to recognise input pattern which were not part of the training set.

3 Radial Basis Function Neural Networks (RBF)

Radial basis functions have been used for data modelling (curve fitting) by many researchers, e.g. Powell (1987) and Poggio and Girossi (1989). Recently these functions have been put in a neural network paradigm in what is called Radial Basis Function (RBF) Neural Networks (Broomhead and Lowe (1988), Moody and Darken (1988), Lee and Kil (1988), Platt (1991)). Schultz et.al. (1994) applied RBF networks in a seismic reservoir characterisation study.

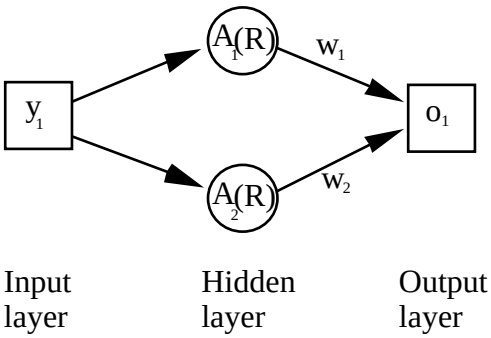

Fig. 4 *Schematic representation of a Radial Basis Function network for the case of a single input variable, two basis functions and one output variable.*

RBF networks have the same feed-forward layered architecture as MLP networks (Fig. 1.4), but the weighting function W and the activation function A are different. With RBF networks, there are only weights between output layer and hidden layer (Fig. 4). Each node in the hidden layer has a unique function, called the basis function. For the simple network of Fig. 4 with a single input, single output and two basis functions, the output is given by the sum of the two basis functions, each multiplied with its own weighting factor. In principle, any type of function can be used to act as basis function. For example, spline functions are used (Kavli, 1992), but the identification RBF network, applies only if radial basis functions are used.

Radial basis functions give local support to data points. The output of the hidden nodes, peaks when the input is near the centroid of the node, and then falls of symmetrically as the Euclidean distance between input and the centroid of a node increases (Fig. 5). The consequence of this behaviour is that RBF networks are good for data interpolation, but not good for data extrapolation.

Several different radial basis functions are in use, with the Gaussian function (Fig. 5a), being the most widely used. If the radial basis centre R is defined as:

$$R = \sqrt{\sum_{i=1}^{L} \frac{(y_i - \mu_i)^2}{\sigma_i^2}}, \tag{1.7}$$

where:
μ_i represents the centre location of each basis and σ_i indicates a scaling of the width of each basis, then the Gaussian activation function is given by:

$$A(R) = \exp\left(-\frac{R^2}{2}\right). \tag{1.8}$$

Multiplication of the activation function $A(R)$ with a weighting factor w then yields the output o (Fig. 4).

Another widely used RBF function is the so-called Inverse Multi-Quadratic Equation (IMQE, Fig. 5b), defined as:

$$A(R) = \frac{1}{\sqrt{R + k^2}}, \tag{1.9}$$

where:
k is an empirically determined smoothing factor (default 0.5 in dGB's software).

Note, that the widths in RBF functions are specified independently from each input dimension, making the functions elliptic rather than spherical. Note as well, that unlike the activation functions for MLPs no bias is included in the RBF functions.

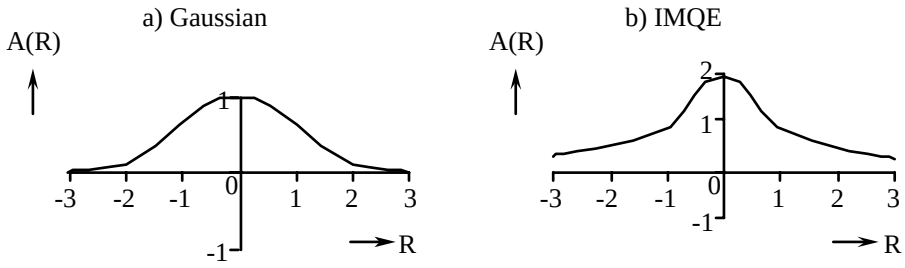

Fig. 5 *Activation functions supported in dGB's software for RBF networks. The Gaussian function has a μ of 0 and a σ of 1. The IMQE function has a μ of 0, a σ of 1 and a k of 0.5.*

Centre locations are typically determined by randomly selecting training examples from a large set of training data. The smoothing parameters and the number of nodes are typically adjusted empirically during training. RBF neural networks and MLPs have been compared by many workers. Kavli (1992) reported consistently better performance of RBF networks in five independent experiments. Another important aspect when comparing RBF networks and MLPs is the training speed. RBF networks can be trained within a fraction of the time that is required for training MLPs. RBF networks, however, generally require more nodes to obtain similar performances.

One of the training algorithms in dGB's software for RBFs is the so-called HSOL algorithm (Lee and Kill, 1989, Carlin, 1992). HSOL uses standard back propagation for updating the function parameters, but this learning algorithm also dynamically allocates new nodes in the hidden layer during training.

4 Unsupervised Vector Quantiser networks

In the preceding sections Multi-Layer Perceptrons and Radial Basis Functions neural networks were introduced. These types of network belongs to the category of supervised learning approaches. Datasets with known input and target vectors are used to train and test these networks. In this section a type of network is introduced that belongs to the category of unsupervised, or competitive learning: the Unsupervised Vector Quantiser. The general aim of competitive learning is to find structure in the data themselves and thereby extracting the relevant properties or features. In the case of the UVQ the aim is to segment (cluster, classify) the data. Similar input vectors must be classified in the same category. The classes are found by the network itself from the correlations of the input data. Therefore, these networks are sometimes referred to as self-organising networks.

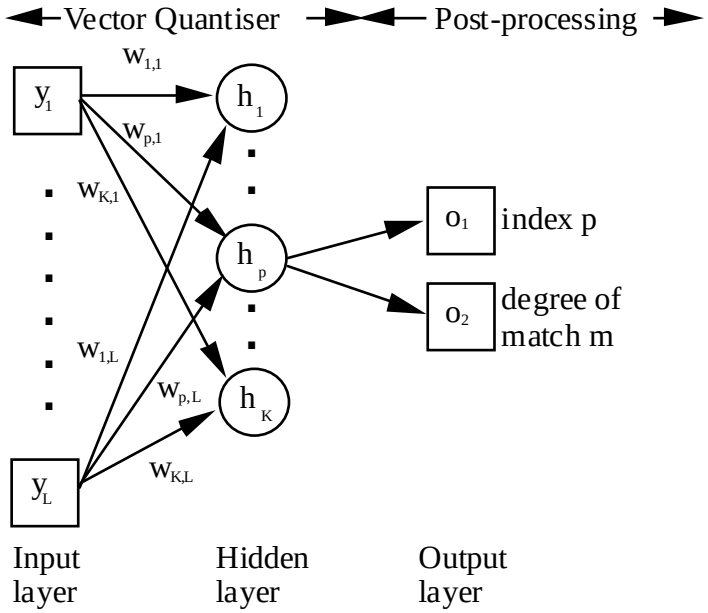

Fig. 6 *Schematic representation of the Unsupervised Vector Quantiser, as used in this study. The network consists of a vector quantiser part and a post-processing part. Two outputs are generated: the index of the winning hidden node (i.e. the class) and a degree of match, which indicates how close the input vector is located near the centre of the class.*

The UVQ that will be used is a modified version of a Learning Vector Quantiser (LVQ). Vector quantisation is an important application of competitive learning for data encoding and compression (Hertz et. al., 1991, and Haykin 1994). In vector quantisation an input vector is replaced by the index of the winning output unit. Vector quantisation requires a set of classes, or codebook to exist. Normally, a set of prototype vectors is used. The class is found by calculating the Euclidean distance to the prototype vectors. The nearest prototype vector is the winner. LVQ's are a supervised version of vector quantisation. In this case the prototype vectors are updated closer to

the input, following a successful classification and further away from it when the classification is unsuccessful.

The unsupervised vector quantiser (UVQ) is quite similar to the LVQ. The prototype vectors are in the unsupervised case initialised as random vectors. The vector closest to the input vector is updated in the direction of the input vector.

The UVQ in this study consists of a two-layer vector quantiser followed by a post-processing output-layer (Fig. 6). In the vector quantiser part of the network, a single layer of hidden nodes h_i with $i = 1,\ldots,K$, where K indicates the number of classes, is fully connected with a set of input nodes y_j with $j = 1,\ldots,L$ via excitatory connections $w_{i,j}$. For each hidden node the net output is computed as the Euclidean distance to the input:

$$h_i(\mathbf{y}) = \sqrt{\sum_{j=1}^{L}(y_j - w_{i,j})^2} \quad i = 1,...,K \cdot \tag{1.10}$$

In the learning phase the net outputs of all hidden nodes (classes) are compared in the post-processing layer. The hidden node with the smallest net output is designated the winner. The weighting vector $w_{p,j}$ associated with the winning node p is then updated according to:

$$w'_{i,j} = \begin{cases} w_{i,j} & i = 1,...,p-1,p+1,...K \quad j = 1,...,L \\ w_{p,j} + \eta\left(y_j - w_{p,j}\right) & j = 1,...,L, \end{cases} \tag{1.11}$$

where:

η is a empirically determined learning rate parameter and $w'_{i,j}$ is the updated weighting matrix. This update rule is known as the standard competitive learning rule. Updating is continued until no noticeable changes in the prototype vectors are observed.

In the application phase, the output layer consists of two nodes: one giving the index number of the winning node, and one giving a degree of match between the input vector and the prototype vector of this node. The degree of match m is computed as:

$$m = \left(1 - \frac{h_p(\mathbf{y})}{r\sqrt{L}}\right), \tag{1.12}$$

where r is the variation range for the training data.

In dGB's software, the input variables are rescaled so that they all fall in the range from -0.8 to 0.8 (therefore, r =1.6). The degree of match m can thus vary from 0 (minimum match) to 1 (perfect match).

The implication of rescaling is that all input variables will contribute equally to the classification result. In our application seismic signals are classified by feeding the UVQ network amplitudes at discrete sample positions. The samples are selected relative to a reference horizon. The rescaling procedure equalises the dynamic range at each sample position. It must be realised that some situations may exist where this approach does not yield an optimum result. For example, if, for the signals to be classified, a maximum amplitude and a zero-crossing always occur at the same sample positions, than the amplitude variations around the zero-crossing are relatively amplified.

This concludes the introduction to the type of neural networks supported in dGB's software.

5 References

Broomhead, D.S., and Lowe, D., 1988. Multivariable functional interpolation and adaptive networks, Complex systems, 2:231-355, 1988.

Carlin, M., 1992. Radial Basis Function Networks and Nonlinear Data Modelling. Proceedings of Neuro-Nimes'92, Neural Networks and their Application, EC2, France, 1992, pp.623-633.

Doveton, J.H., 1994. Geologic Log Analysis Using Computer Methods. AAPG Computer Applications in Geology, No. 2. Association of American Petroleum Geologists.

Fahlman, 1988. An Empirical Study of Learning Speed in BackPropagation Networks. Technical Report CMU-CS-88-162, 1988. LeCun, Y., 1985. Une procedure d'apprentissage pour réseau à seauil asymétrique (A learning procedure for asymmetric threshold networks). Proceedings of Cognitiva 85, 599-604, Paris.

Groot, P.F.M. de, 1995. Seismic reservoir characterisation employing factual and simulated wells. PhD thesis, Delft University Press.

Groot, P.F.M. de, Bril, A.H, Floris, F.J.T. and Campbell, A.E., 1996. Monte Carlo simulation of wells. Geophysics, Vol. 61, No. 3 (May-June 1996); P.631-638.

Haykin, S., 1994. Neural Networks, A Comprehensive Foundation. Macmillan College Publishing Company, New York.

Hertz, J., Krogh, A. and Palmer, R.G., 1991. Introduction to the theory of neural computation, Lecture notes volume I. Santa Fe intsitute studies in the sciences of complexity, Addison-Wesley Publ. Comp.

Kavli, T.Ø., 1992. Learning Priciples in Dynamic Control. PhD.Thesis University of Oslo, ISBN no. 82-411-0394-8.

LeCun, Y., 1985. Une procedure d'apprentissage pour réseau à seauil asymétrique (A learning procedure for asymmetric threshold networks). Proceedings of Cognitiva 85, 599-604, Paris.

Lee, S. and Kil, R.M., 1988. Multilayer feedforward potential function network. IEEE International Conference on Neural Networks, I-161 - I-171, San Diego, 1988.

Lee, S. and Kil, R.M., 1989. Bidirectional Continuous Associator Based On Gaussian Potential Function Network. International Joint Conference on Neural Networks, vol.1, 1989, pp.45-53.

Lippmann R.P., 1989. Pattern Classification Using Neural Networks. IEEE Communications Magazine, November 1989.

McCulloch, W.S. and Pitts, W., 1943, A logical calculus of idea's immanent in nervous activity. Bulletin of Mathematical Biophysics 5, page 115-133. Reprinted in Anderson, J.A. and Rosenfield, E., 1988. Neurocomputing: Foundations of Research, Cambridge MIT Press.

Minsky, M. and Papert, S., 1969. Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, MA.

Moody, J., and Darken, C.J., 1988. Learning with localized receptive fields, in Proceedings of the 1988 Connectionist Models Summer School. pp. 133-143, editors: Touretzky et.al., Morgan-Kaufman.

Parker, D.B., 1985. Learning-Logic, Tech.Rep.TR-47. MIT Center for Computational Research in Economic and Management Science, Cambridge, MA. Rosenblatt, F., 1962. Principles of NeurodyWintershallics: Perceptrons and the Theory of Brain Mechanisms. Washington D.C., Spartan Books.

Platt, J., 1991. A resource-allocating network for function interpolation. Neural Computation, 3(2):213-225, 1991.

Poggio, T. and Girosi, F., 1989. A theory of networks for approximation and learning. Technical report, Artificial Intelligence Laboratory, Massachusets Institute of Technology, Jul. 1989.

Powel, M.J.D., 1987. Radial basis functions for multivariable interpolation: A review, in Algorithms for Approximation. editors: Mason, J.C., and Cox, M.G., Clarendon Press, London.

Rich, E. and Knight, K., 1991. Artificial Intelligence second edition. McGraw-Hill, Inc.

Rosenblatt, F., 1962. Principles of NeurodyWintershallics: Perceptrons and the Theory of Brain Mechanisms. Washington D.C., Spartan Books.

Rumelhart, D.E., Hinton, G.E., and Williams, R.J., 1986. Learning internal representations by error propagation, Parallel Distributed Processing. Editors: Rumelhart, D.E., McClelland, J.L. and the PDP Research group, 318-362, Cambridge, MA, MIT Press.

Schmidt, W.F., 1994, Neural Pattern Classifying Systems. PhD. thesis, TU Delft, CIP-DATA Koninklijke Bibliotheek, Den Haag, ISBN 90-9006716-7.

Schultz et.al.,1994. Seismic-guided estimation of log properties, Part 1: A data-driven interpretation methodology. The Leading Edge, May 1994; Part 2: Using artificial neural networks for nonlinear attribute calibration. The Leading Edge, June 1994; Part 3: A controlled study. The Leading Edge, July 1994.

Sinvhal A. and Sinvhal H., 1992. Seismic Modelling and Pattern Recognition in Oil Exploration. Kluwer Academic Publishers.

Turing A.M., 1937. On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. (ser. 2), 42, 230-65; a correction 43, 544-6.

Werbos, P.J., 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD thesis, Harvard University, Cambridge, MA.