Research article

N-hidden layer artificial neural network architecture computer code: geophysical application example

Jide Nosakare Ogunbo, Olufemi Adigun Alagbe, Michael Ilesanmi Oladapo, Changsoo Shin

Abstract

We provide a MATLAB computer code for training artificial neural network (ANN) with \(N+1\) layer (N-hidden layer) architecture. Currently, the ANN application to solving geophysical problems have been confined to the 2-layer, i.e. 1-hidden layer, architecture because there are no open source software codes for higher numbered layer architecture. The restriction to the 2-layer architecture comes with the attendant model error due to insufficient hidden neurons to fully define the ANN machines. The N-hidden layer ANN has a general architecture whose sensitivity is the accumulation of the backpropagation of the error between the feedforward output and the target patterns. The trained ANN machine can be retrieved by the gradient optimization method namely: Levenberg-Marquardt, steepest descent or conjugate gradient methods. Our test results on the Poisson’s ratio (as a function of compressional and shear wave velocities) machines with 2-, 3- and 4-layer ANN architectures reveal that the machines with higher number of layers outperform those with lower number of layers. Specifically, the 3- and 4-layer ANN machines have \(\geq 97\%\) accuracy, predicting the lithology and fluid identification in the oil and gas industry by means of the Poisson’s ratio, whereas the 2-layer ANN machines poorly predict the results with large error as 20%. These results therefore reinforce our belief that this open source code will facilitate the training of accurate N-hidden layer ANN sophisticated machines with high performance and quality delivery of geophysical solutions. Moreover, the easy portability of the functions of the code into other software will enhance a versatile application and further research to improve its performance.

1. Introduction

Pattern recognition and nonlinear function approximations are the two basic applications of artificial neural network identified by Hagan et al. (1996). Although the foundation of artificial neural network (ANN) is domiciled in biological neural science, it has been widely applied in several disciplines to solve and approximate complex functions. Recently ANN has continued to enjoy increasing attention and demand to approximate and predict the complex nonlinear relationships between dependent and independent variables. Hagan et al. (1996) listed some of the areas that the ANN has been successfully applied; and they include the aerospace (autopilot), automotive, banking (card readers), defense (signal/image identification), electronics (machine vision), entertainment (animations), currency price prediction, insurance (policy application evaluation), manufacturing (product design and analysis), medical prosthesis design, robotics, speech recognition, securities (market analysis), telecommunications (real-time translation of spoken language), transportation (vehicle scheduling) and geophysical oil and gas exploration.

An excellent review of the applications of ANN in geophysics is provided in Poulton (2002) who emphasized the use of neural network as intelligence amplification toolkit and not a replacement of human effort. In the processing, analyzing and interpretation of geophysical data where patterns are recognized, the ANN is expected to enhance exploration process from the combination of speed and computer efficiency (Poulton, 2002). Specifically, the ANN has been applied to solve geophysical problems namely: for tentatively attenuating noise in the 3D GPR data (Ouadfeul and Aliouane, 2014); for trace editing and picking of the first break from refraction seismic data (McCormack et al., 1993); Baronian et al. (2002). Baronian et al. (2007) used the ANN for seismic velocity analysis. Bescoby et al. (2006) applied ANN to enhance magnetic data interpretation for archeological studies. ANN based autopicker for micro-earthquake and

© 2020 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).
hydrofrac data were developed by Aminzadeh et al. (2011) and Maity and Aminzadeh (2012) respectively; Du et al. (2018) compared the use of ANN and rock physics in the estimation of shear wave velocity in a heterogeneous reservoir. Source wavelet estimation for adaptive minimum prediction-error deconvolution was achieved via Hopfield field neural networks (Wang and Mendel, 1992). Boadu (1998) reportedly inverted fracture density from field P- and S-wave seismic velocities using ANN. Other areas the ANN has found usefulness in geophysics applications are the multiattribute transforms for the prediction of log properties from seismic data (Hampson et al., 2001; Dorrington and Link, 2004); for automated lithology classification for coal exploration (Horrocks et al., 2015); permeability predictions (Huang et al., 1996); for modeling complex relationships between petrophysical parameters affecting nuclear magnetic resonance (NMR) relaxation times and cementation factor (Boadu, 2001). Thermal conductivity of sedimentary rocks are predicted using ANN (Goutorbe et al., 2006). Cai (1994) used ANN to predict recovery ratio in Haji Jiang Oil Field. Recently, Ghorbani et al. (2017) used the so-called boost learning system to detect and categorize mechanisms for micro seismic events. Hamidian et al. (2018) merged wavelet transform with neuro fuzzy approach to locate damaged dams.

The adaptation of the biological neural network for solving geophysical problems is predicated on pattern recognition. The extensive applications of ANN in geophysics as reported in the literature cited above suggests that patterns of complex nonlinear relationship between dependent and independent variables exist in geophysical phenomena. For example in the use of seismic attribute for well-log prediction in Dorrington and Link (2004), the seismic attributes and well-log can be taken as dependent and independent variables respectively and the ANN supplies complex nonlinear relationship between them. And because these dependent variables may not have simple parametric relationships with the independent variables in the patterns, the approximation of a well-trained network is usually a reliable extrapolator (Poulton et al., 1992; Dorrington and Link, 2004). Although only results of the applications have been indicated in the aforementioned literature, the few areas of applications referenced suggests that the ANN can still be used to solve more geophysical problems as long as the patterns between the dependent and independent variables can be established. However, most of the published articles only slightly touched on the details of the ANN and presented results from the ANN toolbox used.

Following the taxonomy reported in Lippmann (1987), the artificial neural network that will be designed in this research is the one with continuous valued input, supervised and feedforward multilayer perceptron, solved by the backpropagation techniques. The feedforward perceptron architecture will be presented, the general framework for the backpropagation algorithms of N+1-layers will be given and coded in MATLAB programming language (e.g. batched Levenberg-Marquardt, steepest descent, or conjugate gradient). Finally, an ANN machine for Poisson’s ratio from compressional and shear wave velocity will be produced which can be useful for fluid identification in the oil and gas exploration. Since there is much left to understand about the ANN performance, we envisage that the open source code will afford users, researchers, geoscientists in artificial neural network explore the advantages/benefits of N+1-layer over two layer ANN using this software as a tool for the experiments.

We therefore present an N+1-layer open source code for ANN experiments. Although the software is written in MATLAB, it uses none of the MATLAB in-built ANN functions, and can be easily translated to other programming languages for users who are constrained by the expensive MATLAB license. The ANN concept and algorithms will be explained in a unified mathematical framework and the implementation of the perceptron backpropagation will be presented in MATLAB code. The unified mathematical framework will extend the two-layer MATLAB algorithm to an N+1-layer which can serve as a research tool to better study the ANN performance. The code is intended to broaden both the appeal and application of ANN in solving geophysical problems (and by extension, solve ANN problems in any other fields of research), especially those with patterns and routines.

In the following sections, the theory of ANN feedforward and backpropagation will first be discussed. Next, the formulation of the unified algorithm for sensitivity (Jacobian) matrix will be presented, which will then be incorporated into gradient based optimization algorithm (Levenberg-Marquardt, steepest descent, or conjugate gradient). Finally, an ANN machine for Poisson’s ratio from compressional and shear wave velocity will be produced which can be useful for fluid identification in the oil and gas exploration. Since there is much left to understand about the ANN performance, we envisage that the open source code will afford users, researchers, geoscientists in artificial neural network explore the advantages/benefits of N+1-layer over two layer ANN using this software as a tool for the experiments.

2. Theory: feedforward artificial neural network and backpropagation algorithms

2.1. Feedforward ANN architecture

An ANN machine can be trained by introducing a vector of patterns into the machine that are coded by some transfer (activation) functions, which are eventually outputted. Whereas the ANN machine is represented by some weight matrix \( W \) and bias vector \( b \) the numerical values of the input patterns vector \( p \) and the output vector \( a \) are referred to as the neurons. Typically the input neurons are multiplied by network weights and added to the bias, and it is sent through a transfer (or activation) function \( f \), resulting in an output of \( a = f(Wp + b) \). Examples of transfer functions used in the literature are: hard limit, linear, log-sigmoid, hyperbolic tangent sigmoid (Hagan et al., 1996). In relation to geophysical problems, the input neurons can be understood as individual numeric amplitude (as in a seismic trace) or different combination of attributes or patterns (as in AVO slopes and intercepts). The network layer as used by Hagan et al. (1996) is identified as containing the weight matrix, the summation and multiplication operations, the bias vector, the transfer function and the output vector. By this definition the input is not considered as a layer and the output layer is the layer whose output is the network output. Hidden layers are other layers apart from the output layer. Figure 1 shows a two-layer network with each layer having its weight matrix \( W \), bias vector \( b \), net input vector \( a \) and an output vector \( a \). The length of the input is also known as dimension \( R \). For example, if an input pattern vector for a single layer network is \( p = [-1 0 0 1] \), then it is said to be four-dimensional, i.e. \( R = 4 \). Thus the length of column of the weight matrix \( W \) that multiplies \( p \) is \( R \) while the row of \( W \) is the number of neurons in the hidden layer, \( S \). Hence the size of \( W \) is \( S \times R \). Therefore, for a multilayer (\( N \)-layer) perceptron, i.e. \( (N-1) \) hidden layers, with multiple weights—\( W^1_{u_1c_1}, W^2_{u_2c_2}, \ldots, W^N_{u_Nc_N} \)—where the superscript represents the layer number and the subscripts \( u \) and \( c \) are the connecting and source neurons respectively; the size of each layer weight is \( S^{n-1} \times S^n \) by \( S^n \) where \( n = 1, 2, 3, \ldots, N \) layer. In this case, the output of a hidden layer is the input to the next hidden layer and are mathematically connected by the usual matrix-vector multiplication until the final result is outputted; this is the feedforward process for the multilayer perceptron network. The network architecture representation for...
different layers can be identified as follows: $R - S^1$ for 1-layer network; $R - S^1 - S^2$ for the 2-layer network; $R - S^1 - S^2 - S^3$ for the 3-layer network and $R - S^1 - S^2 - S^3 - \ldots - S^{N+1}$ for the $N+1$-layer network which we plan to implement to generalize the process.

### 2.2. N-layer backpropagation algorithms

The problem of the ANN is similar to the usual nonlinear inversion optimization problem (Ogunbo and Zhang, 2014; Ogunbo et al., 2018; Ogunbo, 2019) whose objective function can be defined in Eq. (1) as:

$$\begin{align*}
\mathcal{O}(W, b) &= \|t - f(Wp + b)\|^2 + \tau \|J(W, b)\|^2.
\end{align*}$$

(1)

In a more compact form, in Eq. (2), where the machine parameters are represented by variable $x$:

$$\begin{align*}
\mathcal{O}(x) &= \|t - f(Wp + b)\|^2 + \|\Delta x\|^2,
\end{align*}$$

(2)

where $t$ is the target pattern corresponding to its pair input pattern $p$; $f(Wp + b)$ is the feedforward output from the machine, $f$ is the appropriate activation function, $W$ and $b$ are the weight matrix and bias vector; $r$ is regularization parameter and $I$ is identity matrix.

### 2.3. Unified algorithm for the sensitivity matrix for optimization

Minimization of Eq. (2) yields Eq. (3):

$$\begin{align*}
\Delta x_k &= J^T(x_k) J(x_k) + \tau I)^{-1} J^T(x_k) v(x_k),
\end{align*}$$

(3)

where, $x_k$ is the model parameters at k-th iteration; $J(x_k)$ is the sensitivity (Jacobian) matrix from $x_k$, $v(x_k) = e_M = t - f(Wp + b)$ is the data misfit at k-th iteration, $e_M$ is the j-th element of the error for the qth input target pair. For $Q$ number of patterns and $S^M$ number of neurons in the output layers, we further identify the error vector as:

$$\begin{align*}
v^T = [v_1, v_2, \ldots, v_N] = [e_{M,1}, e_{M,2}, \ldots, e_{M,S^M}]
\end{align*}$$

and the parameter vector as:

$$\begin{align*}
S^T = [x_1, x_2, \ldots, x_N] = [w^1_{1,1}, w^1_{1,2}, w^1_{1,3} \ldots w^1_{1,L}, b^1_{1,1}, b^1_{1,2}, b^1_{1,3} \ldots b^1_{1,L}, \\
\ldots, w^M_{1,1}, w^M_{1,2}, w^M_{1,3} \ldots w^M_{1,L}, b^M_{1,1}, b^M_{1,2}, b^M_{1,3} \ldots b^M_{1,L}]
\end{align*}$$

where the total length of the error vector (row of $J(x_k)$) for the multilayer network is $N = Q \times S^M$ and the total length of the parameter vector (column of $J(x_k)$) for the multilayer network is given in Eq. (4) as:

$$\begin{align*}
n = S^1 (R + 1) + S^2 (S^1 + 1) + \ldots + S^M (S^{M-1} + 1).
\end{align*}$$

(4)

Thus the Jacobian matrix $J(x)$ is now formed as defined in Eq. (5):

$$\begin{align*}
\frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \quad \frac{\partial e_{1,2}}{\partial w^1_{1,2}} & \quad \ldots & \quad \frac{\partial e_{1,1}}{\partial w^1_{1,L}} \\
\frac{\partial e_{2,1}}{\partial w^1_{2,1}} & \quad \frac{\partial e_{2,2}}{\partial w^1_{2,2}} & \quad \ldots & \quad \frac{\partial e_{2,1}}{\partial w^1_{2,L}} \\
\vdots & \quad \vdots & \quad \ddots & \quad \vdots \\
\frac{\partial e_{M,1}}{\partial w^1_{M,1}} & \quad \frac{\partial e_{M,2}}{\partial w^1_{M,2}} & \quad \ldots & \quad \frac{\partial e_{M,1}}{\partial w^1_{M,L}}
\end{align*}$$

$$\begin{align*}
\frac{\partial e_{1,1}}{\partial b^1_{1,1}} & \quad \frac{\partial e_{1,2}}{\partial b^1_{1,2}} & \quad \ldots & \quad \frac{\partial e_{1,1}}{\partial b^1_{1,L}} \\
\frac{\partial e_{2,1}}{\partial b^1_{2,1}} & \quad \frac{\partial e_{2,2}}{\partial b^1_{2,2}} & \quad \ldots & \quad \frac{\partial e_{2,1}}{\partial b^1_{2,L}} \\
\vdots & \quad \vdots & \quad \ddots & \quad \vdots \\
\frac{\partial e_{M,1}}{\partial b^1_{M,1}} & \quad \frac{\partial e_{M,2}}{\partial b^1_{M,2}} & \quad \ldots & \quad \frac{\partial e_{M,1}}{\partial b^1_{M,L}}
\end{align*}$$

$$\begin{align*}
\frac{\partial e_{1,1}}{\partial w^2_{1,1}} & \quad \frac{\partial e_{1,2}}{\partial w^2_{1,2}} & \quad \ldots & \quad \frac{\partial e_{1,1}}{\partial w^2_{1,L}} \\
\frac{\partial e_{2,1}}{\partial w^2_{2,1}} & \quad \frac{\partial e_{2,2}}{\partial w^2_{2,2}} & \quad \ldots & \quad \frac{\partial e_{2,1}}{\partial w^2_{2,L}} \\
\vdots & \quad \vdots & \quad \ddots & \quad \vdots \\
\frac{\partial e_{M,1}}{\partial w^2_{M,1}} & \quad \frac{\partial e_{M,2}}{\partial w^2_{M,2}} & \quad \ldots & \quad \frac{\partial e_{M,1}}{\partial w^2_{M,L}}
\end{align*}$$

$$\begin{align*}
\frac{\partial e_{1,1}}{\partial b^2_{1,1}} & \quad \frac{\partial e_{1,2}}{\partial b^2_{1,2}} & \quad \ldots & \quad \frac{\partial e_{1,1}}{\partial b^2_{1,L}} \\
\frac{\partial e_{2,1}}{\partial b^2_{2,1}} & \quad \frac{\partial e_{2,2}}{\partial b^2_{2,2}} & \quad \ldots & \quad \frac{\partial e_{2,1}}{\partial b^2_{2,L}} \\
\vdots & \quad \vdots & \quad \ddots & \quad \vdots \\
\frac{\partial e_{M,1}}{\partial b^2_{M,1}} & \quad \frac{\partial e_{M,2}}{\partial b^2_{M,2}} & \quad \ldots & \quad \frac{\partial e_{M,1}}{\partial b^2_{M,L}}
\end{align*}$$

$$\begin{align*}
\frac{\partial e_{1,1}}{\partial w^3_{1,1}} & \quad \frac{\partial e_{1,2}}{\partial w^3_{1,2}} & \quad \ldots & \quad \frac{\partial e_{1,1}}{\partial w^3_{1,L}} \\
\frac{\partial e_{2,1}}{\partial w^3_{2,1}} & \quad \frac{\partial e_{2,2}}{\partial w^3_{2,2}} & \quad \ldots & \quad \frac{\partial e_{2,1}}{\partial w^3_{2,L}} \\
\vdots & \quad \vdots & \quad \ddots & \quad \vdots \\
\frac{\partial e_{M,1}}{\partial w^3_{M,1}} & \quad \frac{\partial e_{M,2}}{\partial w^3_{M,2}} & \quad \ldots & \quad \frac{\partial e_{M,1}}{\partial w^3_{M,L}}
\end{align*}$$

$$\begin{align*}
\frac{\partial e_{1,1}}{\partial b^3_{1,1}} & \quad \frac{\partial e_{1,2}}{\partial b^3_{1,2}} & \quad \ldots & \quad \frac{\partial e_{1,1}}{\partial b^3_{1,L}} \\
\frac{\partial e_{2,1}}{\partial b^3_{2,1}} & \quad \frac{\partial e_{2,2}}{\partial b^3_{2,2}} & \quad \ldots & \quad \frac{\partial e_{2,1}}{\partial b^3_{2,L}} \\
\vdots & \quad \vdots & \quad \ddots & \quad \vdots \\
\frac{\partial e_{M,1}}{\partial b^3_{M,1}} & \quad \frac{\partial e_{M,2}}{\partial b^3_{M,2}} & \quad \ldots & \quad \frac{\partial e_{M,1}}{\partial b^3_{M,L}}
\end{align*}$$

(5)

In Eq. (6), the weight $x_l$
\[ [J] = \frac{\partial \mathbf{s}}{\partial x_i} = \frac{\partial \mathbf{s}}{\partial x_j} \times \frac{\partial x_j}{\partial w_m} = \mathbf{s}_i = \frac{\partial \mathbf{s}}{\partial w_m} = \mathbf{s}_m. \]  

(6)

In Eq. (7), the bias \( s_i \) 

\[ [J] = \frac{\partial s}{\partial x_i} = \frac{\partial s}{\partial x_j} \times \frac{\partial x_j}{\partial w_m} = \mathbf{s}_i = \frac{\partial s}{\partial w_m} = \mathbf{s}_m. \]  

(7)

where the Marquardt sensitivity is \( s_i = \frac{\partial \mathbf{s}}{\partial x_i} \) and the output of layer \( m \) is \( \mathbf{s}_m \), and \( h = (q - 1)\mathbf{s} + k \), for the \( q \)th pattern. The sensitivity is backpropagated from the output layer to the first layer through a recurrence relationship in Eqs. (8) and (9): 

\[ \mathbf{S}^M = -\mathbf{F}'(\mathbf{n}^M) \mathbf{S}^M \]  

(8)

\[ \mathbf{S}^M = -\mathbf{F}'(\mathbf{n}^M) (\mathbf{W}^{m+1})^T \mathbf{S}^{m+1} \]  

(9)

The total sensitivity matrices for each layer are then formed by augmenting the matrices computed for each input as given in Eq. (10):

\[ \mathbf{S} = \mathbf{S}^1 \mathbf{S}^2 \cdots \mathbf{S}_L. \]  

(10)

where \( \mathbf{S}^M \) is the output sensitivity that initializes the recurrence relation; \((\mathbf{W}^{m+1})^T \) is the transpose of the weight matrix; and \( \mathbf{F}'(\mathbf{n}^M) \) is given in Eq. (11) as:

\[ \mathbf{F}'(\mathbf{n}^M) = \begin{pmatrix} f'(\mathbf{n}^M) & 0 & \cdots & 0 \\ 0 & f'(\mathbf{n}^M) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'(\mathbf{n}^M) \end{pmatrix}. \]  

(11)

where \( f'(\mathbf{n}^M) \) is the derivative of the activation function for the individual layer output \( \mathbf{n}^M \).

2.4. Algorithms: Levenberg-Marquardt, steepest descent and conjugate gradient

Eq. (3) is the general optimization problem to be solved; and it can be solved by any of the gradient-based methods. We have solved Eq. (3) by the Levenberg-Marquardt, steepest descent and the conjugate gradient methods. The unified Eq. (3) suggests once the Jacobian (sensitivity) matrix is available then any of the optimization methods can be used to find the machine weights and biases. Solving Eq. (3) using direct matrix inversion gives the Levenberg-Marquardt solution and the parameter update at \( k + 1 \) iteration is simply given as:

\[ x(k + 1) = x - \Delta x. \]

From Eq. (3) setting 

\[ A = \mathbf{F}'(x_i)J(x_i) + \tau \mathbf{I}, \]

and 

\[ b = \mathbf{F}'(x_i)y(x_i), \]

then Eq. (3) can also be solved using the steepest descent method by following these steps:

1) \( r = b - Ax \) (initial \( \Delta x \) can be set to zero)
2) \( p = r \)
3) Find the step length using, \( \alpha = \frac{\mathbf{F}'(x_i)}{\partial \mathbf{F}'(x_i)/\partial p} \)
4) Update \( \Delta x \), using \( \Delta x = \Delta x + \alpha p \)

The process repeats itself until the end of predefined number of steepest descent iterations.

The conjugate gradient method is similar to the steepest descent algorithm but from the second conjugate gradient iteration \((i \geq 2)\), 

\[ p_i = \frac{\mathbf{F}'(x_i) - \mathbf{F}'(x_{i-1})}{\tau \mathbf{F}'(x_{i-1})}, \]

the step length is found at \( i \)th conjugate gradient iteration using:

\[ \alpha = \frac{\mathbf{F}'(x_i) - \mathbf{F}'(x_{i-1})}{\partial \mathbf{F}'(x_i)/\partial p}. \]

In the conjugate gradient algorithm above, the process is terminated either at the prescribed maximum iteration or when the normalized sum of \( r \) is less than a tolerance value. Also, in the conjugate gradient algorithm, \( p \) is reset to \( r \) when the conjugate gradient iteration is a multiple of the length of the parameter \( x \) to prevent problem associated with accumulated round-off error in the conjugate gradient iteration loop (Shewchuk, 1994).

2.5. Methodology flowchart

Figure 2 shows the flowchart of the methodology implemented for training the \( N \)-hidden layer ANN machine. There are three basic components: the preliminary parameter settings, the pattern-sensitivity matrix loop and the epoch/iteration loop. At the preliminary level, input-target patterns, initial weights and biases (according to Eq. 4), number of patterns, number of epoch or iterations for training, and the activation functions are supplied. In the pattern-sensitivity matrix loop, for each epoch, the input pattern passes through the feedforward algorithm to produce an output. Error between the feedforward output and the target patterns is stored, which is backpropagated and accumulated for all the input-target patterns to form the sensitivity matrix. In the epoch/iteration loop, the optimization performance of the ANN is quantified using the mean squared error (MSE) of all input-target patterns error. Inversion is then performed, to retrieve the weights and biases, using any of the Levenberg-Marquardt, steepest descent, or conjugate gradient optimization algorithm. The process continues till the end of specified iteration (epoch) number.

3. The \( N \)-hidden ANN code structure

The code is structured into five basic MATLAB functions namely:

- FEEDFORWARD, ERROR VECTOR, BACKPROPAGATION, PERFORMANCE INDEX and WEIGHTS BIAS OPTIMIZATION. The FEEDFORWARD function computes the output of the network which is subtracted from the target output using the ERROR VECTOR function. The error vector is backpropagated from the output to the first layer. The mean squared error (MSE) is calculated in the PERFORMANCE INDEX function. The current weights and biases are finally updated in the WEIGHTS BIAS OPTIMIZATION function by means of either conjugate gradient, steepest descent or Levenberg-Marquardt optimization scheme. The amount of the regularization factor is reduced or increased if the performance is good or poor by the Levenberg-Marquardt divisor or multiplier respectively.

A PARAMETER file text is prepared to facilitate the training. The PARAMETER file consists of three file names for the input patterns, target patterns and the initial weights respectively. After supplying these file names, the dimension of the input pattern (\( R \)), the number patterns and number of layers are next given. The number of neurons in each hidden layer and the output layer are also specified. The types of activation functions for each layer (hidden and output) are indicated with numerical flag 1 for purelin or 2 for logsig activation functions. Although we have given two types of activation functions the users can easily include more activation flag but should ensure these functions and their derivative are numerically defined in the appropriate sections of the codes. The last line of the PARAMETER file consists of the total number of it-
4. ANN machine for Poisson's ratio

Eq. (12) indicates that the Poisson's ratio is a function of the compressional P-wave velocity ($V_p$) and the shear wave velocity ($V_s$). Specifically, the Poisson's ratio is the function of the ratio of the velocities. For fluid identification in the oil and gas exploration, the Poisson's ratio ($\sigma$) is usually employed. This is particularly important as lithological identification since $V_s = 0$ in fluid.

$$\sigma = 0.5 \left( \frac{\left( \frac{V_p}{V_s} \right)^2 - 2}{\left( \frac{V_p}{V_s} \right)^2 - 1} \right). \quad (12)$$

In the ANN machine formulation the $\frac{V_p}{V_s}$ in Eq. (12) is the input pattern while the Poisson's ratio ($\sigma$) is the target pattern. Various ANN architectures for 2-, 3- and 4-layers are used to design different ANN machines for the Poisson's ratio for Eq. (12), see Table 1. The first and the last numbers in the ANN architecture in Table 1 are the dimension of the input pattern and number of target (or output) neurons; and their values
are 1 because there is a one-to-one mapping of the $\frac{V_p}{V_s}$ to $\sigma$. But the numbers between these input-target neurons represent the number of neurons in the hidden layers. The number of weights and biases indicates the sophistication of ANN architecture. The steepest descent algorithm is used for the machine trainings. The MSE is used to assess the performance of an ANN with respect to the others. A large MSE suggests poor performance compared to low MSE. In Table 1, the MSE is given in the unit of base 10 logarithm (or log10(MSE)) to aid the interpretation. For each architecture, the network training is halted after five hundred epochs and the log10(MSE) values in Table 1 are those of the 500th epoch. Figure 3 shows the plots of log10(MSE) with epoch, for the 2-, 3-, and 4-layers ANN machines. In this example of Poisson's ratio, it is observed that for a fixed numbered-layer ANN architecture case, the performance improves as a function of the number of neurons in the hidden layers. Also, comparing across different number of layers, the 4-layer ANN outperforms the 2- and 3-layer ANN. Specifically, the 4-layer with ANN architecture of 1-4-4-4-1 has the best performance at the end of the training. Figure 4 displays the data fit between the true Poisson's ratio and the machine results for 2-, 3- and 4-layer ANN. The results show that indeed the machine with the low log10(MSE) outperform those with higher values.

The best performing machines for the 2-, 3- and 4-layers in Table 1 are subsequently used for testing and validating their performances on the test data that were not used for the training. The velocity ranges used are $4500 \leq \frac{V_p}{V_s} \leq 8000 \text{ m/s}$ and $1000 \leq V_s \leq 3000 \text{ m/s}$. Figure 5a and b show the velocity ratio, $\frac{V_p}{V_s}$ and the true Poisson's ratio plots respectively. The weights and biases from the training of the ANN machines are used.

| Number of Layers | Number of Hidden Layers | ANN Architecture | Number of weights and biases | Log10(MSE) |
|------------------|-------------------------|------------------|-------------------------------|------------|
| 2                | 1                       | 1-1-1            | 4                            | -2.93      |
| 2                | 1                       | 1-2-1            | 7                            | -3.27      |
| 3                | 1                       | 1-3-1            | 10                           | -3.60      |
| 3                | 2                       | 1-1-1-1          | 6                            | -3.10      |
| 3                | 2                       | 1-2-2-1          | 13                           | -4.60      |
| 3                | 2                       | 1-3-1-1          | 17                           | -3.60      |
| 3                | 2                       | 1-3-2-1          | 17                           | -4.50      |
| 3                | 2                       | 1-3-3-1          | 22                           | -5.49      |
| 3                | 2                       | 1-4-4-1          | 33                           | -5.55      |
| 4                | 3                       | 1-1-2-3-1        | 19                           | -3.84      |
| 4                | 3                       | 1-2-2-2-1        | 19                           | -3.82      |
| 4                | 3                       | 1-3-3-3-1        | 34                           | -5.43      |
| 4                | 3                       | 1-4-4-4-1        | 53                           | -5.60      |

Table 1. Performance measure, in terms of Log10(MSE) of different ANN architectures for Poisson's ratio machine.

Figure 3. Network training logarithm mean squared error (MSE) versus the epoch for (a) 2-layer (b) 3-layer and (c) 4-layer ANN for the Poisson’ ratio.
on the test data and the generated Poisson's ratio results are presented in Figure 6a, b and c for the 2-, 3- and 4-layers respectively. Visibly, they all bear striking resemblance with the true Poisson's ratio plot in Figure 5b. However, to assess the performance of the machines on the test data set, the percentage error between the computed and the true Poisson's ratio data are shown in Figure 6d, e and f for the 2-, 3- and 4-layers respectively. The 3- and 4-layer ANN machines (Figure 6e and f respectively) have comparable performances with about 3 % error for small $\frac{V_p}{V_s}$. However, the 2-layer machine in Figure 6d generally perform poorly compared with 3- and 4-layers machines, with as large percentage error.

Figure 4. Fit between the machine results and the true Poisson's ratio data for (a) 2-layer (b) 3-layer and (c) 4-layer ANN.

Figure 5. (a) True velocity ratio test data and (b) true Poisson's ratio test data.
as 10–20 % in predicting the Poisson’s ratio. Thus the 2-layer ANN machine will not reliably identify lithology or fluid content as function of $V_p$ and $V_s$, but either the 3- or 4-layer ANN machine will make the prediction within $\geq 97$ % accuracy. Our results agree with the findings in Pattichis et al. (1995), that reported a superior performance of the 3-layer compared with the 2-layer machines.

5. Discussion

The popularity of the 2-layer artificial neural network (ANN) has been attributed to its ability to approximate quite a few nonlinear functions. The statement is debatable and questionable even though much research efforts have been invested in improving the performance of the 2-layer ANN, for example learning speed of the 2-layer ANN (Nguyen and Widrow, 1990). However, we suppose that because many researchers have continued to use the 2-layer ANN, so much have been researched to improve its performance and so when compared with the networks of higher number of layers, the 2-layer networks appear to outperform the others. Nevertheless, the 2-layer architecture presents a simple and unsophisticated ANN architecture compared with ANN architectures of higher number of layers. The obvious limitation of such a simple 2-layer architecture is the inherent large error in the trained machine because there are not enough neurons to fully capture the sophistications of complex machines. Our experiments have demonstrated that as the number of layers and the number of neurons in the hidden layers increase, the total number of weights and biases sufficiently support the sophistication of the trained machine, leading to better performances on the test data. With the use of our open source code for the $N$-hidden layer ANN architecture, researchers may discover ways of speeding up the processes for higher numbered layer ANN machines.

6. Conclusions

We present an open source MATLAB code for the $N$-hidden layer artificial neural network (ANN) for training high performance ANN machines with greater accuracy than the 2-layer (1-hidden layer) that is popularly used. A significant number of the ANN applications in geophysics have been confined to the 2-layer ANN architecture. This

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{poisson_ratio_prediction.png}
\caption{Poisson’s ratio prediction by (a) 2-layer (1-3-1), (b) 3-layer 1-4-4-1 and (c) 4-layer 1-4-4-4-1 ANN architectures. Percentage error (d) between Figures 5b and 6a, (e) between Figures 5b and 6b and (f) between Figures 5b and 6c.}
\end{figure}
choice of the 2-layer architecture is not only restrictive but a potent source of error in the eventual ANN machines. One reason the majority of the ANN machines have been restricted to the 2-layer architecture is the unavailability of an open source code to train the ANN machines with higher number of layers. Our research makes available the open source MATLAB code that is needed to train sophisticated ANN machines with any number of layers. The software performs the feedforward, backpropagation, and computes the sensitivity for all input-target patterns for all the N-hidden layers. And the optimization is done by any of the Levenberg-Marquardt, steepest descent or conjugate gradient methods. Another benefit is that the code is portable and can be easily incorporated into another related software for bigger ANN related project. The experiments we perform on generating ANN Poisson’s ratio machines from the 2-, 3- and 4-layers indicate that the higher number layers perform better than the 2-layer machine. This offers higher reliability on the 3- and 4-layer ANN architectures as prediction machines using the test data, leading to more accurate predictions of lithology and fluid identification by the Poisson’s ratio as a function of the ratio compressional and shear wave velocities. Nevertheless, the current limitation of the N-hidden layer is the total randomness in defining the initial weights and biases. The future research therefore should focus on optimizing the initial weights and biases for improved learning speed of the N-hidden layer ANN machines.

Declarations

Author contribution statement

J. N. Ogunbo: Conceived and designed the experiments; Performed the experiments; Analyzed and interpreted the data; Wrote the paper.

O. A. Alaghe, M. I. Oladapo: Contributed reagents, materials, analysis tools or data.

C. Shin: Analyzed and interpreted the data; Contributed reagents, materials, analysis tools or data.

Funding statement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Competing interest statement

The authors declare no conflict of interest.

Additional information

Data associated with this study has been deposited at https://github.com/JesusIsGod/N_Hidden_Layer_ANN_Code.

Acknowledgements

This project was executed during the Brain Korea (BK21) Research Program at the Department of Energy Resources Engineering, Seoul National University, South Korea. We equally appreciate the reviewers’ comments and recommendations that improved the quality of the paper and presentation.

References

Aminzadeh, F., Maity, D., Tafﬁ, T.A., Brouwer, F., 2011. Artiﬁcial neural network based autopicker for micro-earthquake data. SEG, Expanded Abstracts. In: Annual International Meeting, pp. 1625–1626.

Baronian, C., Riahi, M.A., Lucas, C., Mokhtari, M., 2007. A theoretical approach to applicability of artiﬁcial neural networks for seismic velocity analysis. J. Appl. Sci. 7 (23), 3659–3668.

Boadu, F.K., 1998. Inversion of fracture density from ﬁeld seismic velocities using artiﬁcial neural networks. Geophysics 63 (2), 534–545.

Boadu, F.K., 2001. Petrophysical parameters affecting NMR relaxation times and cementation factor: artiﬁcial neural network analysis. J. Environ. Eng. Geophys. 6 (3), 107–114.

Bescoby, D.J., Cawley, G.C., Chroston, P.N., 2006. Enhanced interpretation of magnetic survey data from archaeological sites using artiﬁcial neural networks. Geophysics 71 (5), H45–H53.

Cai, Y., 1994. The Artiﬁcial Neural Network for Research of the Recovery Ratio of Oil ﬁelds. SEG Technical Program expanded abstracts 1994, pp. 791–793.

Do, Q., Yasin, Q., Ismail, A., 2018. A comparative analysis of artiﬁcial neural network and rock physics for the estimation of shear wave velocity in a highly heterogeneous reservoir. SEG, Expanded Abstracts. In: 88th Annual Meeting International Exposition, pp. 2246–2249.

Dorrington, K.P., Link, C.A., 2004. Genetic-algorithm/neural-network approach to seismic attribute selection for well-log prediction. Geophysics 69 (1), 212–221.

Ghorbani, S., Barari, M., Hosseini, M., 2017. A modern method to improve of detecting and categorizing mechanism for micro seismic events data using boost learning system. Civ. Eng. J. 3 (9), 715–726.

Goutorbe, B., Lucazenu, F., Bonneville, A., 2006. Using neural networks to predict thermal conductivity from geophysical well logs. Geophys. J. Int. 166, 115–125.

Hagan, M.T., Demuth, H.B., Beale, M., 1996. Neural Network Design. China Machine Press.

Hamidian, D., Salajegheh, J., Salajegheh, E., 2018. Damage detection of irregular plates and regular waves by wavelet transform combined adaptive neuro fuzzy interference system. Civ. Eng. J. 4 (2), 305–319.

Hampson, D.P., Schueke, J.S., Quirein, J.A., 2001. Use of multivariate transforms to predict log properties from seismic data. Geophysics 66 (1), 220–236.

Horrocks, T., Holden, E., Wedge, D., 2015. Evaluation of automated lithology classiﬁcation architectures using highly-sampled wireline logs for coal exploration. ASEG-PESA 2015 1-4.

Huang, Z., Shewchuk, J.R., 1994. An Introduction to the Conjugate Gradient Method without the Need for Human Intelligence. Technical report, University of California, Berkeley.

Huang, Z., Shimeld, J., Williamson, M., Katsube, J., 1996. Permeability prediction with artiﬁcial neural networks. Geophysics 61 (2), 422–436.

Lippmann, R.P., 1987. An introduction to computing with neural nets. IEEE ASSP Mag. 4–22.

Maity, D., Aminzadeh, F., 2012. Novel hybrid ANN autopicker for hydrofract data: a comparative study. SEG, Expanded Abstracts. In: Annual International Meeting, pp. 1–5.

McCormack, M.D., Zaucha, D.E., Dunek, D.W., 1993. First-break refrac tion event picking and seismic data trace editing using neural networks. Geophysics 58 (1), 67–78.

Nguyen, D., Widrow, B., 1990. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. Int. Jt. Conf. Neural Network. IEEE, 1990.

Ogunbo, J.N., 2019. Mono-model parameter joint inversion by Gramian constraints: EM methods example. Earth Space Sci. 6, 741–751.

Ogunbo, J.N., Marquis, G., Wang, W., 2018. Joint inversion of seismic traveltine and frequency-domain airborne electromagnetic data for hydrocarbon exploration. Geophysics 83 (2), U9–U22.

Ogunbo, J.N., Zhang, J., 2014. Joint Seismic Traveltine and TEM Inversion for Near Surface Imaging. SEG Technical Program Expanded Abstracts, 2014, pp. 2104–2108.

Oudeiﬁld, S., Aliouane, L., 2014. Noise attenuation from 3D GPR data using artiﬁcial neural network. SAGEEP 2014, 166–170.

Pattichis, C.S., Charalambous, C., Middleton, L.T., 1995. Efﬁcient training of neural network models in classiﬁcation of electromyographic data. Med. Biol. Eng. Comput. 33, 499–503.

Poulton, M.M., 2002. Neural networks as an intelligence ampliﬁcation tool: a review of applications. Geophysics 67 (3), 979–992.

Poulton, M.M., Sternberg, B.K., Glass, C.E., 1992. Location of subsurface targets in geophysical data using neural networks. Geophysics 57 (12), 1534–1544.

Shewchuk, J.R., 1994. An Introduction to the Conjugate Gradient Method without the Agonizing Pain. School of Computer Science, Carnegie Mellon University, p. 58.

Shewchuk, J.R., 1994. An Introduction to the Conjugate Gradient Method without the Agonizing Pain. School of Computer Science, Carnegie Mellon University, p. 58.

Wilmotovski, B.M., Chen, Y., 1999. Efﬁcient algorithm for training neural network with one hidden layer. IEEE 1725–1728.