Performance Evaluation of Training Algorithms in Backpropagation Neural Network Approach to Blast-Induced Ground Vibration Prediction*

[1]C.K. Arthur, [1]V.A. Temeng and [1]Y.Y. Ziggah
[1]University of Mine and Technology, P. O. Box 237, Tarkwa

Arthur, C. K., Temeng, V. A. and Ziggah, Y. Y. (2020) "Performance Evaluation of Training Algorithms in Backpropagation Neural Network Approach to Blast-Induced Ground Vibration Prediction", *Ghana Mining Journal*, Vol. 20, No. 1, pp. 20 - 33.

Abstract

Backpropagation Neural Network (BPNN) is an artificial intelligence technique that has seen several applications in many fields of science and engineering. It is well-known that, the critical task in developing an effective and accurate BPNN model depends on an appropriate training algorithm, transfer function, number of hidden layers and number of hidden neurons. Despite the numerous contributing factors for the development of a BPNN model, training algorithm is key in achieving optimum BPNN model performance. This study is focused on evaluating and comparing the performance of 13 training algorithms in BPNN for the prediction of blast-induced ground vibration. The training algorithms considered include: Levenberg-Marquardt, Bayesian Regularisation, Broyden–Fletcher–Goldfarb–Shanno (BFGS) Quasi-Newton, Resilient Backpropagation, Scaled Conjugate Gradient, Conjugate Gradient with Powell/Beale Restarts, Fletcher-Powell Conjugate Gradient, Polak-Ribiére Conjugate Gradient, One Step Secant, Gradient Descent with Adaptive Learning Rate, Gradient Descent with Momentum, Gradient Descent, and Gradient Descent with Momentum and Adaptive Learning Rate. Using ranking values for the performance indicators of Mean Squared Error (MSE), correlation coefficient (R), number of training epoch (iteration) and the duration for convergence, the performance of the various training algorithms used to build the BPNN models were evaluated. The obtained overall ranking results showed that the BFGS Quasi-Newton algorithm outperformed the other training algorithms even though the Levenberg Marquardt algorithm was found to have the best computational speed and utilised the smallest number of epochs.

Keywords: Artificial Intelligence, Blast-induced Ground Vibration, Backpropagation Training Algorithms

1 Introduction

Artificial Neural Network (ANN) developed by Warren McCulloch and Walter Pitts in 1943 is one of the widely used supervised learning approaches which was inspired by the structural complexity of the human brain (Yegnanarayana, 2009). In the 1960's the concept of the backpropagation algorithm for neural network training was introduced which was thereafter made popular by Rumelhart *et al*. (1989) and hence the name Backpropagation Neural Network (BPNN). The BPNN can thus be described as feed forward neural network which comprises of the input layer, the hidden layer and the output layer. The role of the input layer is to receive information from the real world. These received input data are assigned weights which define the strength of the connection between input and hidden layer neurons with an added bias term. The weighted inputs are then sent to neurons in the hidden layer which are then transformed by a nonlinear activation function. The resulting output from the hidden layer is then sent to the output layer where a linear activation function is employed to produce the final output. It is worth mentioning that, in the training process, after each forward pass through a network, backward pass is performed by backpropagation with the aim of minimising the error between the estimated network value and the expected measured values by adjusting the model's parameters (weights and biases). These forward and backward passes are repeated until the network error converge at a minimum predetermined threshold value.

Studies have shown that the critical task in developing an effective and accurate BPNN model depends on selecting an appropriate training algorithm and fine-tuning certain factors such as the transfer function, number of hidden layers and number of hidden neurons (Zhu *et al.*, 2005; Huang *et al.*, 2006; Huang *et al.*, 2011). Despite the numerous contributing factors for the development of a BPNN model, the training algorithm plays a key role in the BPNN final outputs. This is because, it has been proven that a BPNN with one hidden layer is enough to correctly fit any continuous data (Hornik *et al.*, 1989; Park and Sandberg, 1991). Additionally, the number of hidden neurons to be used is mostly determined by the sequential trial and error procedure in the model training (Braspenning *et al.*, 1995; Sheela and Deepa, 2013, Anifowose *et al.*, 2017). The common practice in the case of the activation function is that the logistic or hyperbolic is usually used in the hidden layer while a linear function is used in the output layer (Garson, 1998; Beale *et al.*, 2017). Therefore, this study is focused on

**Manuscript received March 20, 2020*
Revised version accepted June 19, 2020
https://doi.org/10.4314/gm.v20i1.3

evaluating and comparing the performance of thirteen (13) training algorithms in BPNN for the prediction of blast-induced ground vibration. The BPNN training algorithms found in literature and applied in this study include the Levenberg-Marquardt (trainLM), Bayesian Regularisation (trainBR), Broyden–Fletcher–Goldfarb–Shanno (BFGS) Quasi-Newton (trainBFGS), Resilient Backpropagation (trainRP), Scaled Conjugate Gradient (trainSCG), Conjugate Gradient with Powell/Beale Restart (trainCGB), Fletcher-Reeves Conjugate Gradient (trainCGF), Polak-Ribiére Conjugate Gradient (trainCGP), One Step Secant (trainOSS), Gradient Descent algorithm with Adaptive Learning Rate (trainGDA), Gradient Descent with Momentum (trainGDM), Gradient Descent (trainGD) and Gradient Descent with Momentum and Adaptive Learning Rate (trainGDX) (Beale *et al*., 2019).

The motive of this study is that, only few of these training algorithms have been applied by researchers in the development of a BPNN model. For instance, researchers such as Kişi and Uncuoğlu (2005) investigated the use of the trainLM, trainCGF and the trainRP for streamflow forecasting and the lateral stress in cohesionless soil determination. These three training algorithms were compared based on their convergence velocities in training and performance in testing. The results showed that, although the trainLM algorithm was found to be faster and having better performance than the other algorithms in training, the trainRP Algorithm had the best accuracy in the testing period. Ceke *et al*. (2009) also investigated the predictive performance of six training algorithms in predicting mean glandular dose based on measurable parameters in mammography. The algorithms compared included the trainSCG, trainCGB, trainBFGS, trainOSS, trainLM and trainRP. Their prediction results showed that the neural network model trained with trainLM algorithm had best results compared to those trained with the other training algorithms. Sandhu and Chhabra (2011) also investigated the predictive performance of trainSCG, trainCGB algorithm, trainCGF algorithm, trainCGP algorithm in reusability evaluation of procedure-based software systems. The results obtained showed that the trainSCG algorithm was the best. In Kaur and Salaria (2013) trainBR, trainLM, trainGDX were compared in developing a neural network for software effort estimation. The trainBR was noted to have performed more creditably than the other algorithms for software effort estimation. Sharma and Venugopalan (2014) in brain hematoma classification compared the performance of trainGD, trainGDM, trainRP, trainSCG, trainCGF, trainCGP, trainBFGS and trainLM algorithms. It was found that trainLM and trainSCG outperformed the other algorithms. In Baghirli (2015) the predictive abilities of the trainLM, trainBR and the trainSCG algorithms were investigated pertaining to the accuracy of the multistep ahead monthly wind speed forecasting. Kayri (2016) also investigated the predictive capabilities of the trainLM and trainBR algorithms on neural networks using social data. The trainBR algorithm showed better performance than the trainLM algorithm.

It can therefore be realised from the afore-discussed instances that there is the need to explore and evaluate the performance of the training algorithms outlined by Beale *et al*. (2019). In this study, the performance of BPNN trained with the 13 algorithms are evaluated to predict blast-induced ground vibration. The motive is that literature has shown that the BPNN trained with trainLM is the most widely and successfully used method for blast-induced ground vibration (Khandelwal and Singh, 2007; Khandelwal and Singh, 2009; Saadat *et al*., 2014; Taheri *et al*., 2017; Arthur *et al*., 2020a). Therefore, the main contribution of this study was to bring to light how training algorithms affect the predictive performance of BPNN in blast-induced ground vibration prediction. This will further enhance and bring up new dimension when applying the BPNN for blast induced ground vibration prediction.

2 Resources and Methods Used

2.1 Resources

The study was carried out in a Manganese Mine in Ghana with an area extension of latitude 5˚16′ North and longitude 1˚59′ West as shown in Fig. 1.

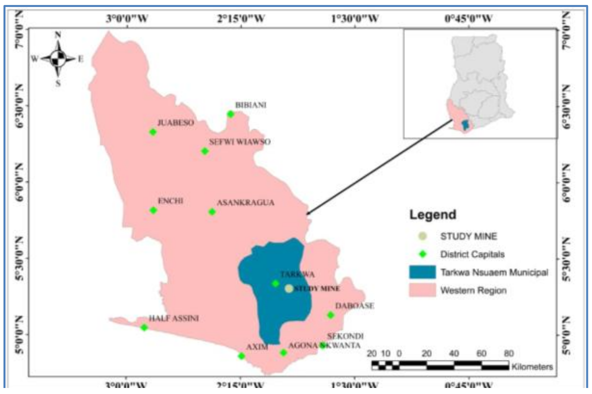

Fig. 1 Study Area

The Mine adopts the use of drill and blast techniques to fragment the in-situ rock mass into appropriate rock sizes. In this regard, drill rigs and emulsion are used in the drilling and blasting processes respectively. The fragmented rocks are either hauled to the crusher or waste dump using

CAT 777F, Komatsu HD 465, and Volvo AD35 rear dump trucks.

For the purpose of this study, a total of 210 historic instances of blast data were collected from the Mine. The blast data comprised of the following parameters: number of blast holes, maximum instantaneous charge (kg), distance between blasting point and monitoring station (m), hole depth (m), powder factor (kg/m^3) and Peak Particle Velocity (PPV) (mm/s). These recorded parameters are deemed significant in affecting the levels of blast-induced vibrations in literature It is noteworthy that PPV is the most preferred parameter for evaluating blast-induced ground vibration (Iramina *et al.*, 2018; Arthur *et al.*, 2020b). However, for the development of the various models as presented in this study, number of blast holes, maximum instantaneous charge (kg), distance between blasting point and monitoring station (m), hole depth (m), powder factor (kg/m^3) were used as the input parameters while the PPV (mm/s) values served as the output parameter. Table 1 shows the statistical description of the input and output parameters used in this study. In order to construct the various models in this study, the collected datasets were divided into two sets: training and testing sets. A total of 130 data points representing 62% of the collected datasets were used as the training sets while the remaining 80 data points representing 38% were used as the testing datasets to independently assess the performance of the trained models.

2.2 Backpropagation Training Algorithms Used

In this section, concise descriptions of the training algorithms is presented. The architectural description of the BPNN is presented here as they are extensively applied and explained in several blast-induced ground vibration studies (Khandelwal and Singh, 2007; Khandelwal and Singh, 2009; Saadat *et al.*, 2014; Taheri *et al.*, 2017; Arthur *et al.*, 2020a).

2.2.1 Levenberg-Marquardt

The trainLM algorithm is an iterative technique for finding the minimum of a multivariate error function E (Equation (1)) that is expressed as the sum of squares of the difference between the actual output y_i and target output t_i (Adeoti and Osanaiye, 2013).

$$E = \frac{1}{2}\sum\left(y_i - t_i\right) \tag{1}$$

The trainLM was designed to approach second order speed without having to compute the Hessian matrix. Nevertheless, the Hessian matrix (H) as well as the gradient (g) can be approximated using Equations (2) and (3) respectively, when the performance function has a form of sum of squares.

$$H = J^T J \tag{2}$$

$$g = J^T e \tag{3}$$

where J is the Jacobian matrix containing the first derivatives of the network errors with respect to the biases and weights, and e is the network error vector. The Jacobian matrix can be computed through a standard backpropagation technique that is much less complex than computing the Hessian matrix (Baghirli, 2015). The trainLM algorithm uses this approximation to the Hessian matrix in the following Newton-like update (Equation (4)).

$$w_{i+1} = w_i - \left[J^T J + \mu I\right]^{-1} J^T e \tag{4}$$

where w represents connection weights, μ is the damping term and I is the identity matrix. The trainLM uses the combination of Gauss-Newton method and gradient descent in its iterative process. When the μ is zero, it becomes a Gauss-Newton method, using the approximate Hessian matrix. When the μ is large, it becomes a gradient descent method having 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 (Baghirli, 2015).

Table 1 Statistical Description of Parameters

Parameters	Type	Unit	Min	Max	Average	Std Dev
Number of blast holes	Inputs	-	19	355	122.50	52.37
Maximum instantaneous charge		kg	11.60	123.49	90.08	19.54
Distance from blasting point to monitoring station		m	573	1500	915.01	234.62
Hole depth		m	3.73	12.58	10.45	1.14
Powder factor		kg/m^3	0.10	0.97	0.69	0.15
Peak Particle Velocity	Output	mm/s	0.13	1.65	0.79	0.32

2.2.2 Bayesian Regularisation

The trainBR is a training algorithm that updates the weights and bias values according to Levenberg-Marquardt optimisation (Kaur and Salaria, 2013). It minimizes a combination of squared errors and weights, and then determines the correct combination to produce a network that generalizes well (Kaur and Salaria, 2013). According to Foresee and Hagan (1997), the method of improving generalisation is referred to as regularisation.

The aim of training is to reduce the sum of squared error, E_D. This implies that, the training objective function is $F = E_D$. However, regularisation adds an additional term, E_W. The objective function is then expressed as shown in Equation (5) (Foresee and Hagan, 1997).

$$F = \beta E_D + \alpha E_W \tag{5}$$

where E_W is the sum of squared of the network weights; E_D is the sum of network errors; α and β are the objective function parameters. Foresee and Hagan (1997) state that, the relative size of the objective function parameters dictates the emphasis for training. If $\alpha << \beta$, then the training algorithm will drive the errors smaller and if $\alpha >> \beta$, training will emphasise weight size reduction at the expense of network errors, thus producing a smoother network problem. However, the main problem with implementing regularisation is setting the correct values for the objective function parameters. The α and β factors are defined using the Bayes' rule. The procedure for finding the correct values of α and β is explained by Foresee and Hagan (1997).

2.2.3 Broyden-Fletcher-Goldfarb-Shanno Quasi-Newton

The trainBFGS algorithm is a Quasi-Newton second-derivative line search family method for solving unconstrained optimization problem (Ibrahim *et al.*, 2014). The trainBFGS uses quadratic Taylor approximation of the objective function in a *d*-vicinity of *x* (Biglari and Ebadian, 2015) as expressed in Equation (6).

$$f(x+d) \approx q(d) = f(x) + d^T g(x) + \frac{1}{2} d^T H(x) d \tag{6}$$

where *g(x)* is the gradient vector and *H(x)* is the Hessian matrix. The necessary condition for a local minimum of *q(d)* with respect to *d* results in the linear system presented in Equation (7).

$$g(x) + H(x)d = 0 \tag{7}$$

which in turn gives the Newton direction *d* (Equation (8)) for line search.

$$d = -H(x)^{-1} - g(x) \tag{8}$$

The exact Newton direction (which is subject to defining in Newton-type methods) is reliable when the Hessian matrix exists and positive definite with the difference between the true objective function and its quadratic approximation not being large.

In Quasi-Newton methods, the idea is to use matrices which approximate the Hessian matrix and/or its inverse, instead of exact computing of the Hessian matrix (as in Newton-type methods). The matrices are normally named $B \approx H$ and $D \approx H^{-1}$. The matrices are adjusted on each iteration and can be produced in many different ways ranging from very simple techniques to highly advanced schemes. The trainBFGS method uses an updating formula which converges to the approximate of the Hessian matrix *H(x*)* as expressed in Equation (9).

$$B_{i+1} = B_i - \frac{B_i s_i s_i^T B_i}{s_i^T B_i s_i} + \frac{y_i y_i^T}{y_i^T s_i} \tag{9}$$

where

$$s_i = x_{i+1} - x_i$$
$$y_i = g_{i+1} - g_i$$

As a starting point, B_0 can be set to any symmetric positive definite matrix, for example and very often, the identity matrix. The trainBFGS method exposes super linear convergence; resource-intensity is estimated as $O(n^2)$ per iteration for *n*-component argument vector.

2.2.4 Resilient Backpropagation

The trainRP algorithm is a training algorithm for neural networks that work similarly to the standard backpropagation algorithm. The difference however is in the way the connecting weights are updated (Prasad *et al.*, 2013). For the backpropagation, the update is computed using the magnitude of the partial derivative as expressed in Equation (10).

$$\Delta w_{jk}(m) = \alpha \times x_j(m) \times \delta_k(m) \tag{10}$$

where α is the learning rate, $x_j(m)$ denotes the inputs propagating back to the *i*th neuron at time step *m* and δ_k is the corresponding error gradient. For the trainRP, an individual delta Δ_{jk} which determines the size of the weight w_{jk} update for each connection is computed. The learning rule expressed in Equation (11) is used in calculating Δ_{jk}.

$$\Delta_{jk}(m)=\begin{cases}\Delta_{jk}(m-1)\times\eta^{+}, \\ \text{if } \frac{\partial E}{\partial w_{jk}}(m-1)\times\frac{\partial E}{\partial w_{jk}}(m)>0 \\ \Delta_{jk}(m-1)\times\eta^{-}, \\ \text{if } \frac{\partial E}{\partial w_{jk}}(m-1)\times\frac{\partial E}{\partial w_{jk}}(m)<0 \\ \Delta_{jk}(m-1), \text{ else}\end{cases} \quad (11)$$

where $0<\eta^{-}<1<\eta^{+}$. It is noteworthy that, for the trainRP, the weight update is not influenced by the magnitude of the derivatives, but by the behaviour of the sign of the two succeeding derivatives.

Every time the partial derivative of the corresponding weight w_{jk} changes its sign indicating the last update was too big and that the algorithm has jumped over a local minimum. The update-value Δ_{jk} is then decreased by the factor η^{-}. If the derivative retains its sign, the updated value is slightly increased in order to accelerate convergence in shallow regions (Riedmiller and Braun, 1992; Prasad *et al*., 2013).

The update rule for weights is the same as that expressed in Equation (12), except that if the partial derivative changes sign, the previous update-step leading to a jump over the minimum is reverted to Equation (13). When a change of sign has occurred, the adaptation process is restarted. The update-values and weights are changed every time the whole pattern set has been presented to the network.

$$\Delta w_{jk}(m)=\begin{cases}-\Delta_0, \text{ if } \frac{\partial E}{\partial w_{jk}}(m)>0 \\ +\Delta_0, \text{ if } \frac{\partial E}{\partial w_{jk}}(m)<0 \\ 0, \text{ else}\end{cases} \quad (12)$$

$$\Delta_{jk}(m)=-\Delta_{jk}(m-1), \quad \text{if } \frac{\partial E}{\partial w_{jk}}(m)\times\frac{\partial E}{\partial w_{jk}}(m-1)<0 \quad (13)$$

2.2.5 Fletcher-Reeves Conjugate Gradient

The trainCGF algorithm is a variation of the Conjugate Gradient method developed by Fletcher and Reeves (1964). The algorithm can train any network if its weight, net input, and transfer functions have derivative functions. Backpropagation is used to calculate derivatives of performance with respect to the weight and bias vectors *M*. Each vector *M*$_i$ is adjusted according to Equation (14).

$$M=M+a(dM) \quad (14)$$

where *dM* is the search direction with *a* being the parameter selected to minimise the performance along the search direction. The line search function is used to locate the minimum point. The first search direction is the negative of the gradient of performance. In succeeding iterations, the search direction is computed from the new gradient and the previous search direction according to Equation (15).

$$dM=-gM+\beta(dM_{old}) \quad (15)$$

where *gM* is the gradient. The parameter β can be computed in several different ways. For the Fletcher-Reeves variation of conjugate gradient it is computed using Equation (16).

$$\beta_k=\frac{g_k^T g_k}{g_{k-1}^T g_{k-1}} \quad (16)$$

where $g_{k-1}^T g_{k-1}$ is the norm square of the previous gradient and $g_k^T g_k$ is the norm square of the current gradient.

2.2.6 Polak-Ribiére Conjugate Gradient

The trainCGP algorithm is another version of the conjugate gradient method proposed by Polak and Ribiére (1969). As with the trainCGF algorithm, the search direction *(p)* at each iteration is determined by Equation (17).

$$p_k=-g_k+\beta_k p_{k-1} \quad (17)$$

For the Polak-Ribiére update, the constant β_k is computed using Equation (18).

$$\beta_k=\frac{\Delta g_{k-1}^T g_k}{g_{k-1}^T g_{k-1}} \quad (18)$$

Equation (18) is the inner product of the previous change in the gradient with the current gradient divided by the norm squared of the previous gradient.

2.2.7 Conjugate Gradient with Powell/Beale Restarts

According to Sandhu and Chhabra (2011), the search direction for all conjugate gradient algorithms is occasionally reset to the negative of the gradient. When the number of network's weights and biases equal the number of iterations, the standard reset point has occurred. However, there are other reset approaches that can improve the training efficiency. One of these is the Powell/Beale Restart approach (Powell, 1977; Beale, 1972). This technique restarts if there is very little orthogonality left between the current gradient and the previous gradient (Sandhu and Chhabra, 2011). Equation (19) is used as a test to determine when to reset the search direction to the negative of the gradient.

$$\left|g_{k-1}^{T} g_k\right| \geq 0.2\left\|g_k\right\|^2 \qquad (19)$$

where g_k is the gradient of the *k*th iteration. If this condition is satisfied, the search direction is reset to the negative of the gradient. This algorithm can train any network if its weight, net input, and transfer functions have derivative functions.

Backpropagation is used to calculate derivatives of performance with respect to the weight and bias vectors *M*. Each vector M_i is adjusted using Equation (14). The line search function is used to locate the minimum point.

2.2.8 Scaled Conjugate Gradient

The trainSCG algorithm belongs to a class of Conjugate Gradient methods developed by Møller (1993). The trainSCG avoids the use of line search in its computation unlike the other conjugate gradient algorithms that require a line search for each iteration. The trainSCG combines the model-trust approach and the conjugate gradient approach (Sandhu and Chhabra, 2011). During computation, the trainSCG algorithm denotes the quadratic approximation to the error *E* in a neighbourhood of a point *w* by $E_{qw}(y)$ (Equation (20)).

$$E_{qw}(y) = E(w) + E'(w)^T y + \frac{1}{2} y^T E''(w) y \qquad (20)$$

Hence, to determine the minimum of $E_{qw}(y)$, the critical points for $E_{qw}(y)$ must be found. The critical points are the solution to the linear system defined by Møller (1993). The Scaled Conjugate Gradient algorithm can train any network as long as its weight, net input, and transfer functions have derivative functions (Sandhu and Chhabra, 2011).

2.2.9 One Step Secant Backpropagation

The trainOSS method is an attempt to bridge the gap between the conjugate gradient algorithms and the quasi-Newton (secant) algorithms (Mukkamala *et al*., 2003). This algorithm does not store the complete Hessian matrix. It however 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 the matrix inverse (Mukkamala *et al*., 2003). The algorithm can train any network if its weight, net input, and transfer functions have derivative functions. Backpropagation is used to calculate derivatives of performance with respect to the weight and bias vectors *M*. Each vector M_i is adjusted according to Equation (14) as in conjugate gradient algorithms. The line search function is used to locate the minimum point. The first search direction is the negative of the gradient of performance. In subsequent iterations, the search direction is computed from the new gradient and the change in the weights and gradients from the previous iteration according to Equation (21).

$$dM = -gM + Ac\left(M_{step}\right) + Bc\left(dgM\right) \qquad (21)$$

here *gM* is the gradient, M_{step} is the change in the weights of the previous iteration, *dgM* is the change in the gradient from the last iteration whereas Ac and Bc are the combinational scalar products of *gM*, M_{step} and *dgM*

2.2.10 Gradient Descent

For the trainGD algorithm, the weights and biases are updated in the direction of the negative gradient of the performance function (Moini and Lakizadeh, 2011). Backpropagation is used to calculate derivatives of performance function, *Q* with respect to the weight and bias vectors *M*. Each vector M_i is adjusted according to the gradient descent as expressed in Equation (22).

$$dM = \alpha \times \frac{dQ}{dx} \qquad (22)$$

where α is the learning rate. The learning rate is multiplied by the negative of the gradient to determine the changes to the weights and biases. The larger the learning rate, the bigger the step leading to unstable algorithm. However, the smaller the learning rate the longer time it takes the algorithm to converge.

2.2.11 Gradient Descent with Adaptive Learning Rate

With standard trainGD algorithm, the learning rate is held constant throughout training. The performance of the algorithm is very sensitive to the proper setting of the learning rate (Peteiro-Barral and Guijarro-Berdiñas, 2013). If the learning rate is set too high, the algorithm can oscillate and become unstable. If the learning rate is too small, the algorithm takes 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 changes during the training process, as the algorithm moves across the performance surface. The performance of the trainGD algorithm can be improved if the learning rate can change during the training process. Thus, the trainGDA algorithm. An adaptive learning rate attempts 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 (Peteiro-Barral and Guijarro-Berdiñas, 2013).

An adaptive learning rate requires some changes in the training procedure used by trainGD algorithm. First, the initial network output and error are calculated. At each iteration new weights and biases are calculated using the current learning rate. New outputs and errors are then calculated.

2.2.12 Gradient Descent with Momentum

The trainGDM allows a network to respond not only to the local gradient, but also to recent trends in the error surface acting like a lowpass filter (Garcez *et al.*, 2008). Momentum allows the network to ignore small features in the error surface. Without momentum, a network can get stuck in a shallow local minimum. With momentum a network can slide through such entrapment.

The trainGDM algorithm depends on two training parameters: namely the learning rate, α and the momentum constant γ. The momentum constant defines the amount of momentum which is set between 0 (no momentum) and values close to 1 (lots of momentum). A momentum constant of 1 (one) results in a network that is completely insensitive to the local gradient and therefore, does not learn properly. Backpropagation is used to calculate derivatives of performance function Q with respect to the weight and bias vectors M. Each vector M_i is adjusted according to gradient descent with momentum as expressed in Equation (23).

$$dM = \gamma \times \left(dM_{previous} \right) + \alpha \times (1-\gamma) \times \frac{dQ}{dM} \tag{23}$$

where $dM_{previous}$ is the previous change to the weight or bias.

2.2.13 Gradient Descent with Momentum and Adaptive Learning Rate

The trainGDX algorithm combines adaptive learning rate with momentum training. It is similar to the trainGDA except that it has the momentum coefficient γ as an additional training parameter (Galaviz *et al*., 2010). The algorithm can train any network as long as its weight, net input, and transfer functions have derivative functions. Backpropagation is used to calculate derivatives of performance Q with respect to the weight and bias vectors M. Each vector M_i is adjusted according to gradient descent with momentum as expressed in Equation (24).

$$dM = \gamma \times \left(dM_{previous} \right) + \alpha \times \gamma \times \frac{dQ}{dM} \tag{24}$$

where $dM_{previous}$ is the previous change to the weight or bias and α is the learning rate. For each iteration when the performance decreases toward the set goal, then the learning rate is increased by the factor (typically 1.05). If performance increases by more than the factor (typically 1.04), the learning rate is adjusted by the factor (typically 0.7) and the change that increased the performance is not made.

2.3 Model Development and Performance Assessment

2.3.1 Data Normalisation

The collected data had varying input ranges and hence the possibility of the larger range values to affect the outcome of the prediction. Hence to avoid this, the various input parameters were normalised into the range [-1, 1] using Equation (25).

$$P_i = P_{min} + \frac{\left(P_{max} - P_{min}\right) \times \left(Q_i - Q_{min}\right)}{Q_{max} - Q_{min}} \tag{25}$$

where P_i denotes the normalised data, Q_i denotes the collected blast data and Q_{max} and Q_{min} represent maximum and minimum values of the collected data with P_{min} and P_{max} values equalling to –1 and 1, respectively.

2.3.2 Model Development

In order to ascertain the predictive performance of the BPNN based on the various training algorithms, the other critical parameters that required fine-tuning were predetermined to serve as the background for this study. Hence, one hidden layer with a hyperbolic tangent transfer function as well as one output layer with a linear transfer function were used for this study as iterated by Hornik *et al.* (1989), Braspenning *et al.* (1995) and Beale *et al.* (2017). Throughout the experimental process, 1 neuron out of 1 to 20 neurons investigated, in the hidden layer was established to be the optimum number of neuron required for the effective development of the BPNN models used in this study. It is worth mentioning that the sequential trial and error procedure for the establishment of the optimal structure of the BPNN models was not presented in this study. Therefore, a model structure of [5 – 1 – 1] meaning, 5 inputs, one hidden layer with 1 neuron and 1 output layer was used in this study to ascertain the performance of the various training algorithms. Moreover, in this study, the BPNN was trained for 8000 epochs with a learning rate of 0.03 and a momentum coefficient of 0.7. The MATLAB R2019a program was used to run the BPNN based on the 13 algorithms (Table

2) discussed in Section 2.2. It is noteworthy that a computer with an Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz, 1.99 GHz processor was used to run the MATLAB program for the various training functions. In Table 2, the syntax for the various training functions defined in the MATLAB environment are presented.

2.3.3 Model Performance Assessment

Performance indices of Mean Squared Error (MSE) (Equation (26)), correlation coefficient (R) (Equation (27)), number of epochs (iterations) and duration for convergence were used to assess the performance of developed BPNN models with their respective training algorithms. The values for each set of performance indices for the respective training algorithms were ranked according to the order of performance, with good performing values having higher ranking values. Afterwards, the total ranking values were computed to ascertain the best performing training algorithm.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(m_i - p_i\right)^2 \qquad (26)$$

$$R = \frac{\sum_{i=1}^{n}\left(m_i - \bar{m}\right)\left(p_i - \bar{p}\right)}{\sqrt{\sum_{i=1}^{n}\left(m_i - \bar{m}\right)^2} \times \sqrt{\sum_{i=1}^{n}\left(p_i - \bar{p}\right)^2}} \qquad (27)$$

where n, m_i, p_i, $\bar{m}$ and $\bar{p}$ are the total number of samples, the measured field values, the predicted field values, the mean of the measured field values and the mean of the predicted values respectively.

3 Results and Discussion

The obtained ranking results based on the number of training epochs (iterations) and duration of convergence (time) are outlined in Table 3.

From Table 3, it can be gathered that, the trainLM algorithm used the minimum number of training epoch of 12 to converge at the optimal solution and thus had the highest-ranking value. It also had the fastest convergence speed of 2 seconds. This is because the trainLM algorithm works by combining the steepest descent and the Gauss-Newton methods to give optimal solution. Thus, the algorithm performs like steepest descent when the current solution is close to local minimum but exhibit fast convergence in the Gauss-Newton condition when the algorithm approaches the correct solution (Lourakis and Argyros, 2005). The trainBR algorithm followed up with training epoch and fast convergence speed of 34 and 5 seconds respectively (Table 3).

The trainCGF algorithm had a faster convergence speed (7 seconds) than the trainBFGS algorithm (9 seconds), even though the trainBFGS algorithm used a smaller number of iterations to converge. The trainOSS (Table 3) was also faster than the trainSCG algorithm but required more iterations to converge. The trainCGB and the trainCGP algorithms had close convergence speed (13 and 12 seconds) and number of iterations (82 and 81) respectively to arrive at the optimal solution. The trainRP had a relatively fast convergence speed of 38 seconds. However, it required a large number of epochs (3817) to converge at the optimal solution. In Table 3, it can also be observed that, the trainGD algorithm and its variations were the slowest with a convergence speed above 1320 seconds (22 minutes) and training epochs of more than 7000 to converge to their optimal solutions. To illustrate graphically the performance of the various training algorithms, the ranking results of the training epochs and duration of convergence were plotted against each other (see Fig. 2).

Table 2 Backpropagation Training Functions and their Respective Algorithms

Training Function Syntax in MATLAB	Algorithm Type	Abbreviation
trainlm	Levenberg-Marquardt	trainLM
trainbr	Bayesian Regularisation	trainBR
trainscg	Scaled Conjugate Gradient	trainSCG
trainbfg	Broyden–Fletcher–Goldfarb–Shanno Quasi-Newton	trainBFGS
traincgb	Conjugate Gradient with Powell/Beale Restarts	trainCGB
traincgp	Polak-Ribiére Conjugate Gradient	trainCGP
traincgf	Fletcher-Reeves Conjugate Gradient	trainCGF
traingd	Gradient Descent	trainGD
traingdm	Gradient Descent with Momentum	trainGDM
traingda	Gradient Descent with Adaptive Learning Rate	trainGDA
traingdx	Gradient Descent with Momentum and Adaptive Learning Rate	trainGDX
trainoss	One Step Secant	trainOSS
trainrp	Resilient Backpropagation	trainRP

The analysis from Fig. 2 is that any training algorithm that appears on the top most right corner is better than those that appear on the below and found at the bottom left corner. In Fig. 2, it can be seen that the trainLM appeared at the top right corner emerging as the best in training epoch and fast convergence. The trainGD algorithm and its variations performed badly as they were located at the left bottom corner. The slowness of the trainGD algorithms to converge has been reiterated by Luhaniwal (2019). It was evident that the gradient descent methods move down a local gradient such that this gradient does not point towards the minimum, given the curvature of the underlining function differs significantly with direction (Nabney, 2002). Furthermore, even if a smaller learning rate is chosen, there is a high possibility for successive iterations to oscillate across 'valleys' in the function (Nabney, 2002).

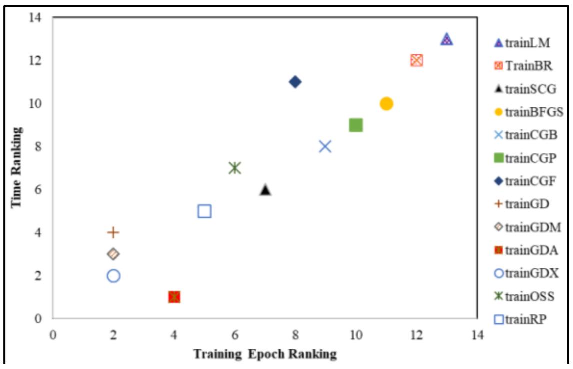

Fig. 2 Training Epoch Ranking against Time Ranking

In furtherance to the performance analysis, *R* and MSE training results with their respective rankings are presented in Table 4. With reference to Table 4, it can be observed that the prediction results based on the training datasets were marginally the same. However, to ascertain the optimal training algorithm, the obtained *R* and MSE values for each training algorithm were ranked. The ranking results (Table 4) showed that, the trainBFGS algorithm gave the highest R value and lowest MSE value. This was closely followed in the order of decreasing performance by trainLM, trainSCG, trainRP, trainOSS, trainCGB, trainCGF, trainGDX, trainCGP, trainBR, trainGDA, trainGD and trainGDM. This can additionally be viewed from Fig. 3 where trainBFGS algorithm appeared on the top right corner indicating its superiority over the other training algorithms. Similarly, the ranking testing results based on R and MSE values for each training algorithm are presented in Table 5.

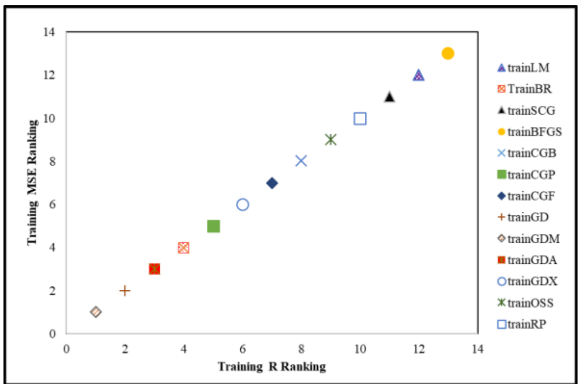

Fig. 3 Training *R* Ranking against MSE Ranking Results

Table 3 Training Epoch and Time Ranking Results

Training Algorithm	Training Epoch	Ranking	Time (sec)	Ranking
trainLM	12	13	2	13
trainBR	34	12	5	12
trainBFGS	61	11	9	10
trainCGF	105	8	7	11
trainCGP	81	10	12	9
trainCGB	82	9	13	8
trainOSS	294	6	14	7
trainSCG	188	7	31	6
trainRP	3817	5	38	5
trainGD	8000	2	1330	4
trainGDM	8000	2	1361	3
trainGDA	7183	4	2001	1
trainGDX	8000	2	1510	2

Table 4 Training Ranking Results

Training Algorithm	**Training Results**			
	R	**Ranking**	**MSE**	**Ranking**
trainBFGS	0.9090005754583	13	0.0209017892774	13
trainLM	0.9090005754579	12	0.0209017892775	12
trainSCG	0.9090005754561	11	0.0209017892778	11
trainRP	0.9090005754453	10	0.0209017892802	10
trainOSS	0.9090005754161	9	0.0209017892866	9
trainCGB	0.9090005751572	8	0.0209017893469	8
trainCGF	0.9090003062234	7	0.0209018497356	7
trainGDX	0.9089999984241	6	0.0209019166768	6
trainCGP	0.9089991077789	5	0.0209021134156	5
trainBR	0.9089275530420	4	0.0209230456845	4
trainGDA	0.9079688866458	3	0.0211538725254	3
trainGD	0.9077256163441	2	0.0211813116335	2
TrainGDM	0.9076340091090	1	0.0212013319689	1

Table 5 Testing Ranking Results

Training Algorithm	**Testing Results**			
	R	**Ranking**	**MSE**	**Ranking**
trainOSS	0.8537001504187	13	0.0216959930735	13
trainBFGS	0.8536998643984	12	0.0216960613531	12
trainSCG	0.8536998515411	11	0.0216960677706	11
trainLM	0.8536998272836	10	0.0216960710372	10
trainRP	0.8536997263504	9	0.0216960941277	9
trainCGB	0.8536982722252	8	0.0216964067015	8
trainCGF	0.8536625479130	7	0.0217023197188	7
trainGDX	0.8536620969792	6	0.0217034459559	6
trainCGP	0.8536313149995	5	0.0217110148640	5
trainBR	0.8528831275419	4	0.0218197614089	4
trainGD	0.8496588256014	2	0.0224272490166	3
trainDGA	0.8510589037442	3	0.0225051033507	1
trainGDM	0.8494726322899	1	0.0224601694095	2

It can be noticed from Table 5 that, a very closely related results (*R* and MSE) was produced by the training algorithms and that their differences are very insignificant. In comparison, the trainOSS algorithm produced the highest *R* value and lowest MSE which was followed by trainBFGS algorithm. The trainSCG, trainLM, trainRP, trainCGB, trainCGF, trainGDX, trainCGP, trainBR, trainGD, trainDGA and trainGDM algorithms followed in that order of decreasing performance as additionally illustrated in Fig. 4. Finally, the obtained ranking results (training and testing) based on the various performance indicators were summed to obtain the overall ranking results for the various training algorithms as shown in Table 6.

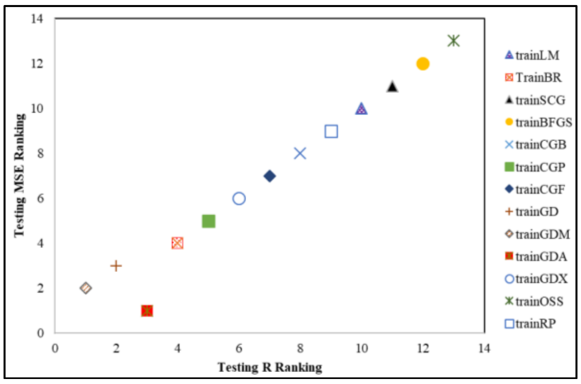

Fig. 4 Testing *R* Ranking against MSE Ranking Results

Table 6 Overall Ranking Based on Training and Testing Results for Epoch, Time, *R* and MSE

Training Algorithm	**Sum of All Ranking Values**	**Overall Rank Position**
trainBFGS	71	**1**
trainLM	70	**2**
trainSCG	57	**3**
trainOSS	57	**3**
trainCGB	49	**5**
trainRP	48	**6**
trainCGF	47	**7**
trainBR	40	**8**
trainCGP	39	**9**
trainGDX	28	**10**
trainGD	15	**11**
trainGDA	15	**11**
trainGDM	10	**13**

With reference to Table 6, it can be observed that the trainBFGS algorithm had the highest total ranking value of 71 making it the best training algorithm for this study. This was closely followed by the trainLM algorithm which had a total ranking of 70. The trainBFGS algorithm and the trainLM algorithm have been stated by Beale *et al.* (2019) to have similar performance as was observed in this study. Both the trainSCG and trainOSS algorithms had the same total ranking value of 57 and thus the same rank position. These were followed by the trainCGB, trainRP, trainCGF, trainBR, trainCGP algorithms in increasing overall rank value, as higher overall rank position signifies lower performance. It can also be seen that, the trainGD and its variational algorithms had very poor total ranking values and thus were the worst performing training algorithms for this study. These rank positions of the various training algorithms are graphically illustrated in Fig. 5.

The trainBFGS algorithm came out the best due to its robustness and self-correcting properties to maintain a satisfaction of the secant condition. In addition to that, it has a good initial approximation of the inverse Hessian matrix (Ding *et al*., 2004; Eisen *et al*., 2017).

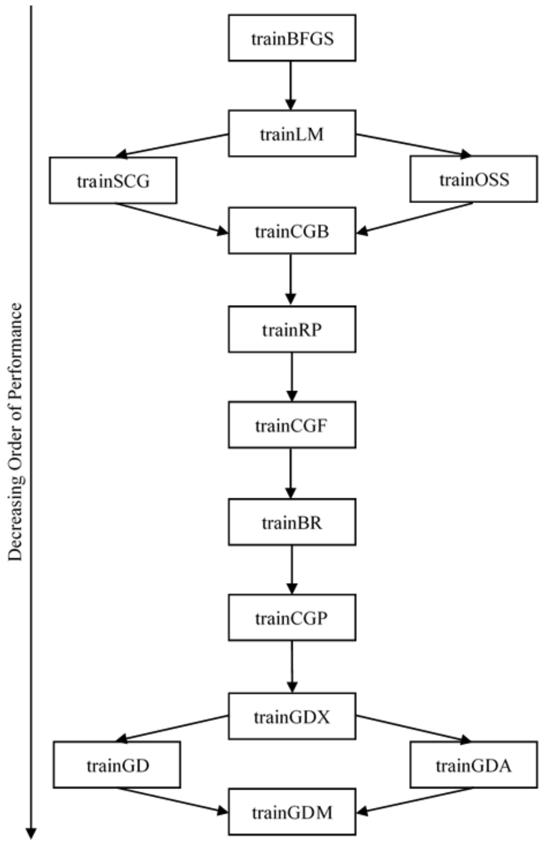

Fig. 5 Order of Rank of Training Algorithms

4 Conclusions

In this study, 13 backpropagation neural network training algorithms namely; Levenberg-Marquardt, Bayesian Regularisation, BFGS Quasi-Newton, Resilient Backpropagation, Scaled Conjugate Gradient, Conjugate Gradient with Powell-Beale Restarts, Fletcher-Powell Conjugate Gradient, Polak-Ribiére Conjugate Gradient, One Step Secant, Gradient Descent with adaptive Learning Rate, Gradient Descent with Momentum, Gradient Descent and Gradient Descent with Momentum and Adaptive Learning Rate were investigated to ascertain their performance based on the prediction of blast-induced ground vibration. In that regard, 13 BPNN models were developed using a total of 210 blasting events data collected from a Manganese Mine in Ghana. One hundred and thirty (130) datapoints out of the 210 were used as the training set while the remaining 80 data points were used to independently assess the BPNN models developed. The input parameters for the models include number of blast holes, maximum instantaneous charge (kg), distance between blasting point and monitoring station (m), hole depth (m) and powder factor (kg/m^3) with PPV (mm/s) serving as the measuring indicator of blast-induced ground vibration in the output layer. With the aim of ascertaining the performance of the training algorithms, the optimum structure of [5-1-1] meaning, five inputs, one hidden layer with one neuron and one output layer was observed for all the training algorithms. The maximum training epoch, learning rate and momentum coefficient were set to 8000, 0.03 and 0.7 respectively. Furthermore, *R*, MSE, number of training epochs and the duration of convergence to the optimal solution were used in ascertaining the performance of the various training algorithms. Each resulting performance indicator was ranked and then summed up to ascertain the overall predictive strength of the training algorithms. The obtained results showed that the Levenberg-Marquardt algorithm had the fastest computational speed as it used 2 seconds and 12 epochs to arrive at its optimal solution. The gradient descent and its variation algorithms were found to be very slow as they used more than 1320 seconds (22 minutes) to arrive at their optimal solution. They also used a maximum training epoch of more than 7000. In the case of training prediction results, the BFGS Quasi-Newton algorithm had the highest R values and lowest MSE values and thus the highest-ranking value even though the other training algorithms achieved marginal results. In the case of the testing results, it was found that the One Step Secant algorithm was able to perform slightly better than all the other training algorithms. However, the summed ranking results showed that the BFGS Quasi-Newton algorithm was the best training algorithm for this study as it had the highest total value of 71 and thus an overall rank value of 1. This was closely followed by the Levenberg-Marquardt, Conjugate Gradient, One Step Secant algorithms, Conjugate Gradient with Powell/Beale Restarts algorithm, Resilient Backpropagation, Fletcher-Reeves Conjugate Gradient, Bayesian Regularisation, Polak-Ribiére Conjugate Gradient, Gradient Descent with Momentum and Adaptive Learning Rate, Gradient Descent, Gradient Descent with Adaptive Learning Rate and finally the Gradient Descent with Momentum in a decreasing order of performance.

Acknowledgements

The authors would like to thank the Ghana National Petroleum Corporation (GNPC) for providing funding to support this work through the GNPC Professorial Chair in Mining Engineering at the University of Mines and Technology (UMaT), Ghana.

References

Adeoti, O. A. and Osanaiye, P. A. (2013), “Effect of Training Algorithms on The Performance of ANN for Pattern Recognition of Bivariate Process”, *International Journal of Computer Applications*, Vol. 69, No. 20, pp. 8 – 12.

Anifowose, F., Labadin, J. and Abdulraheem, A. (2017), “Towards an Improved Ensemble Learning Model of Artificial Neural Networks: Lessons Learned on Using Randomized Numbers of Hidden Neurons”, *In Artificial Intelligence: Concepts, Methodologies, Tools, and Applications*, IGI Global, pp. 325-356.

Arthur, C. K., Temeng, V. A. and Ziggah, Y. Y. (2020a), “Multivariate Adaptive Regression Splines (MARS) Approach to Blast-Induced Ground Vibration Prediction”, *International Journal of Mining, Reclamation and Environment*, Vol. 34, No .3, pp. 198-222.

Arthur, C. K., Temeng, V. A. and Ziggah, Y. Y. (2020b), “Novel Approach to Predicting Blast-Induced Ground Vibration Using Gaussian Process Regression”, *Engineering with Computers*, Vol. 36, No. 1, pp.29-42.

Baghirli, O. (2015), “Comparison of Lavenberg-Marquardt, Scaled Conjugate Gradient and Bayesian Regularization Backpropagation Algorithms for Multistep Ahead Wind Speed Forecasting Using Multilayer Perceptron Feedforward Neural Network”, *Published MSc Thesis Report*, Uppsala University, Gotland, 35 pp.

Beale, E. M. L. (1972), “A Derivation of Conjugate Gradients”, *In Numerical methods for nonlinear optimization*, Lootsma, F.A. (ed.), Academic Press, London, pp. 39–43.

Beale, M. H., Hagan, M. T. and Demuth, H. B. (2017), *Neural Network Toolbox™ User's Guide*, The MathWorks Inc, USA, 431 pp.

Beale, M. H., Hagan, M. T. and Demuth, H. B. (2019), *MATLAB Deep Learning Toolbox™ User’s Guide: PDF Documentation for Release R2019a*, The MathWorks Inc, USA, 431 pp.

Biglari, F. and Ebadian, A. (2015), “Limited Memory BFGS Method Based on a High-Order Tensor Model”, *Computational Optimization and Applications*, Vol. 60, No. 2, pp. 413-422.

Braspenning, P. J., Thuijsman, F. and Weijters, A. J. M. M. (1995), *Artificial Neural Networks: An*

Introduction to ANN Theory and Practice, Springer Science & Business Media, 293 pp.

Ceke, D., Kunosic, S., Kopric, M. and Lincender, L. (2009), "Using Neural Network Algorithms in Prediction of Mean Glandular Dose Based on the Measurable Parameters in Mammography", *Acta Informatica Medica*, Vol. 17, No. 4, pp.194-197.

Ding, Y., Lushi, E. and Li, Q. (2004), "Investigation of Quasi-Newton Methods for Unconstrained Optimization", *Published Report*, Simon Fraser University, Canada, 23 pp.

Eisen, M., Mokhtari, A. and Ribeiro, A. (2017), "Decentralized Quasi-Newton Methods", *IEEE Transactions on Signal Processing*, Vol. 65, No. 10, pp. 2613-2628.

Fletcher, R. and Reeves, C. M. (1964), "Function Minimization by Conjugate Gradient", *The Computer Journal*, Vol. 7, No. 2, pp.149-154.

Foresee, F. D. and Hagan, M. T. (1997), "Gauss-Newton approximation to Bayesian learning", *Proceedings of the International Joint Conference on Neural Networks*, Vol. 3, pp. 1930-1935

Galaviz, J. P., Melin, P. and Trujillo, L. (2010), "Improvement of the Backpropagation Algorithm Using (1+ 1) Evolutionary Strategies", *In Soft Computing for Recognition Based on Biometrics*, Springer, Berlin, Heidelberg. pp. 287-302.

Garcez, A. S. A., Lamb, L. C. and Gabbay, D. M. (2008), *Neural-Symbolic Cognitive Reasoning*, Springer Science & Business Media, 198 pp.

Garson, G. D. (1998), *Neural Networks: An Introductory Guide for Social Scientists*, SAGE, 194 pp.

Ghasemi, E., Ataei, M. and Hashemolhosseini, H. (2013), "Development of a Fuzzy Model for Predicting Ground Vibration Caused by Rock Blasting in Surface Mining", *Journal of Vibration and Control*, Vol. 19, No. 5, pp.755-770.

Hasanipanah, M., Amnieh, H. B., Khamesi, H., Armaghani, D. J., Golzar, S. B. and Shahnazar, A. (2018), "Prediction of an Environmental Issue of Mine Blasting: An Imperialistic Competitive Algorithm-Based Fuzzy System", *International Journal of Environmental Science and Technology*, Vol. 15, No. 3, pp. 551-560.

Hornik, K., Stinchcombe, M. and White, H. (1989), "Multilayer Feed Forward Networks are Universal Approximators", *Neural Network*, Vol. 2, No. 5, pp. 359 – 366.

Huang, G. B., Zhou, H., Ding, X. and Zhang, R., (2011), "Extreme Learning Machine for Regression and Multiclass Classification", *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 42, No. 2, pp. 513-529.

Huang, G. B., Zhu, Q. Y. and Siew, C. K. (2006), "Extreme Learning Machine: Theory and Applications", *Neurocomputing*, Vol. 70, pp.489-501.

Ibrahim, M. A. H., Mamat, M. and Leong, W. J. (2014), "The Hybrid BFGS-CG Method in Solving Unconstrained Optimization Problems", *Abstract and Applied Analysis*, Vol. 2014, pp. 1 – 6.

Iramina, W. S., Sansone, E. C., Wichers, M., Wahyudi, S., Eston, S. M. D., Shimada, H. and Sasaoka, T. (2018), "Comparing Blast-Induced Ground Vibration Models Using ANN and Empirical Geomechanical Relationships', *REM-International Engineering Journal*, Vol. 71, No. 1, pp.89-95.

Kaur, H. and Salaria, D. S. (2013), "Bayesian Regularization Based Neural Network Tool for Software Effort Estimation", *Global Journal of Computer Science and Technology*, Vol. 13, No. 2, pp. 44 – 50.

Kayri, M. (2016), "Predictive Abilities of Bayesian Regularization and Levenberg–Marquardt Algorithms in Artificial Neural Networks: A Comparative Empirical Study on Social Data", *Mathematical and Computational Applications*, Vol. 21, No. 20, pp. 1 – 11.

Khandelwal, M. and Singh, T. N. (2007), "Evaluation of Blast-Induced Ground Vibration Predictors", *Soil Dynamics and Earthquake Engineering*, Vol. 27, No. 2, pp. 116 – 125.

Khandelwal, M. and Singh, T. N. (2009), "Prediction of Blast-Induced Ground Vibration Using Artificial Neural Network", *International Journal of Rock Mechanics and Mining Sciences*, Vol. 46, No. 7, pp. 1214–1222.

Kişi, Ö. and Uncuoğlu, E. (2005), "Comparison of Three Back-propagation Training Algorithm for Two Case Studies", *Indian Journal of Engineering and Materials Science*, Vol. 12, pp. 434 – 442

Lourakis, M. L. A. and Argyros, A. A. (2005), "Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment?", *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05)*, pp. 1526-1531.

Luhaniwal, V. (2019), "Why Gradient Descent Isn't Enough: A Comprehensive Introduction to Optimization Algorithms in Neural Networks", *https://towardsdatascience.com/why-gradient descent-isnt-enough-a-comprehensive-introduction-to-optimization-algorithms-in-59670fd5c096*, Accessed: March 27, 2020.

Moini, M. and Lakizadeh, A. (2011), *Concrete Workability: An Investigation on Temperature Effects Using Artificial Neural Networks*, AuthorHouse. 148 pp.

Møller, M. F. (1993), "A Scaled Conjugate Gradient Algorithm for Fast Supervised

Learning", *Neural Networks*, Vol. 6, No. 4, pp. 525-533.

Mukkamala, S., Sung, A. H. and Abraham, A. (2003), "Intrusion Detection Using Ensemble of Soft Computing Paradigms", *In Intelligent Systems Design and Applications*, Springer, Berlin, Heidelberg, pp. 239-248.

Nabney, I. (2002), *NETLAB: Algorithms for Pattern Recognition*, Springer Science & Business Media, 420 pp.

Orozco, J. and García, C. A. R. (2003), "Detecting Pathologies from Infant Cry Applying Scaled Conjugate Gradient Neural Networks", *In European Symposium on Artificial Neural Networks*, Bruges (Belgium), Vol. 2003, pp. 1-7.

Park, J. and Sandberg, I. W. (1991), "Universal Approximation Using Radial-Basis-Function Networks", *Neural Computation*, Vol. 3, No. 2, pp.246-257.

Peteiro-Barral, D. and Guijarro-Berdiñas, B. (2013), "A Study on the Scalability of Artificial Neural Networks Training Algorithms Using Multiple-Criteria Decision-Making Methods", *Proceedings of the International Conference on Artificial Intelligence and Soft Computing*, Springer, Berlin, Heidelberg, pp. 162-173.

Polak, E. R. and Ribière, G. (1969), "Note Sur la Convergence de Methodes de Directions Conjugat", *Revue Francaise d'Informatique et Recherche Operationnelle*, Vol. 16, pp. 35 – 43.

Powell, M. J. D. (1977), "Restart Procedures for the Conjugate Gradient Method", *Mathematical Programming*, Vol. 12, No. 1, pp.241-254.

Prasad, N., Singh, R. and Lal, S. P. (2013), "Comparison of Back Propagation and Resilient Propagation Algorithm for Spam Classification", *Proceedings of the 2013 Fifth International Conference on Computational Intelligence, Modelling and Simulation*, pp. 29-34.

Riedmiller, M. and Braun, H. (1992), "RPROP. A Fast Adaptive Learning Algorithm" *Proceedings of the 1992 International Symposium on Computer and Information Sciences*, Antalya, Turkey, pp.279-285.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1986), "Learning Representations by Back-Propagating Errors", *Nature*, Vol. 323, No. 6088, pp.533-536.

Saadat, M., Khandelwal, M. and Monjezi, M. (2014), "An ANN-Based Approach to Predict Blast-Induced Ground Vibration of Gol-E-Gohar Iron Ore Mine, Iran", *Journal of Rock Mechanics and Geotechnical Engineering*, Vol. 6, No. 1, pp. 67 – 76.

Sandhu, P. S. and Chhabra, S. (2011), "A Comparative Analysis of Conjugate Gradient Algorithms and PSO Based Neural Network Approaches for Reusability Evaluation of Procedure Based Software Systems", *Chiang Mai Journal of Science*, Vol. 38, No. 2, pp.123-135.

Sharma, B. and Venugopalan, K. (2014), "Comparison of Neural Network Training Functions for Hematoma Classification in Brain CT Images", *IOSR Journal of Computer Engineering*, Vol. 16, No. 1, pp.31-35.

Sheela, K. G. and Deepa, S. N. (2013), "Review on Methods to Fix Number of Hidden Neurons in Neural Networks", *Mathematical Problems in Engineering*, pp. 1 – 11.

Taheri, K., Hasanipanah, M., Golzar, S. B. and Majid, M. Z. A. (2017), "A Hybrid Artificial Bee Colony Algorithm-Artificial Neural Network for Forecasting the Blast Produced Ground Vibration", *Engineering with Computers*, Vol. 33, No. 3, pp.689-700.

Yegnanarayana, B. (2009), *Artificial Neural Networks*, PHI Learning Pvt. Ltd., 461 pp.

Zhu, Q. Y., Qin, A. K., Suganthan, P. N. and Huang, G. B. (2005), "Evolutionary extreme learning machine", *Pattern Recognition*, Vol. 38, No. 10, pp.1759-1763.

Authors

C. K. Arthur is a Lecturer in Mining Engineering at the University of Mines and Technology (UMaT), Tarkwa. He obtained his BSc (Hons.) and PhD degrees in Mining Engineering from UMaT. His areas of specialisation include Machine Learning, Artificial Intelligence, Operation Research Engineering, Optimisation of Mining Systems, Explosive and Blasting Technology.

V. A. Temeng is an Associate Professor in Mining Engineering at the University of Mines and Technology (UMaT), Tarkwa. He obtained his BSc (Hons.) and PgD in Mining Engineering degrees from UMaT. He holds MSc degree from the University of Zambia and PhD degree from the Michigan Technological University. He is a member of the Society of Mining, Metallurgy and Exploration (MSME) and a member of the Ghana Institution of Engineers (GhIE). His areas of specialisation include Operations Research, Materials Handling and Computer Applications.

Y. Y. Ziggah is a Lecturer at the Geomatic Engineering Department of the University of Mines and Technology (UMaT). He holds a BSc in Geomatic Engineering from Kwame Nkrumah University of Science and Technology, Kumasi, Ghana. He obtained his Master of Engineering degree and PhD in Geodesy and Survey Engineering from China University of Geosciences (Wuhan). His research interests include artificial intelligent application in engineering, geodetic coordinate transformation, gravity field modelling, height systems and geodetic deformation modelling.