Truncated-Newton Training Algorithm for Neurocomputational Viscoplastic Model

M.S. Al-Haik [a], H. Garmestani [a], I.M. Navon [b]

[a] Department of Mechanical Engineering, FAMU-FSU College of Engineering, Florida State University, Tallahassee, FL 32310

[b] Department of Mathematics and School of Computational Science and Information Technology, Florida State Universit, Tallahassee, FL 32306

Abstract

We present an estimate approach to compute the viscoplastic behavior of a polymeric composite under different thermomechanical approaches. This investagation incorporates computational neural network as the tool for determining the creep behaviour of the composite. We propose a new second-order learning algorithm for training the multilayer networks. Training in the neural network is generally specified as the minimization of an appropriate error function with respect to parameters of the network (weights and learning rates) corresponding to excitory and inhibitory connections. We propose here a technique for error minimization based on the use of the Truncated Newton (TN) large-scale unconstrained minimization technique. This technique offers a more sophisticated use the of the TN minimization for gradient information compared to simple steepest descent methods. The technique is used in the context of application to neural networks In this work we specify the necessary details for implementing the Truncated Newton methods for training of the neural networks, and provide comparative experimental results from use of these methods to depict the viscoplastic behavior of a polymeric composite. These results verify superiority of the present approach.

Key words: Viscoplasticity Newural Networks Optimization Truncated Newton

1   Introduction

Structural polymer matrix composites (PMCS) possessing superior strength-to-weight and modulus-to-weight ratios are being considered as alternative materials for structural applications (1). As polymer matrix composites continue to find applications in critical structural components, accurate constitutive modeling becomes increasingly important. Constitutive modeling of poly-

meric composite materials presents a diＣcult and distinct challenge. While significant progress has been made in constructing models applicable for small strain and limited strain-rate and temperature regimes, much less progress has been made for more general conditions. Polymers behave very di¤erently depending on the loading conditions to which they are subjected, from brittle to viscoelastic, to viscous (fluid-like). Another impediment to modeling polymeric behavior is that the mechanisms of deformation in polymers are distinct from those in metals, for which extensive modeling work has been done. Consequently, well-established constitutive models and concepts for metals are not directly transferable to modeling polymeric composites.

Polymeric materials behavior has been repeatedly demonstrated to be very temperature and rate dependent (2). The creep response of carbon fiber-polymeric matrix composite, compounds the inherent time-dependent behavior of the polymeric phase and the temporal behavior of carbon fibers due to fiber-matrix debonds, interlayer delamination and diffuse microcracking. Since all the later damage mechanisms are also time-dependent and occur cojointly with polymeric creep, it is extremely difficult to separate out the individual contributions of the composite constituents.

The creep and response of polymers within the linear range of behavior was modeled with a high degree of success by linear viscoelasticity theory (3), (4). In spite of steady but slow progress over the last 30 years, the understanding and modelling of nonlinear behavior of polymers is still the subject of ongoing research (5), (6). Beyond the linear range of behavior, progress in the understanding and modeling of creep in polymeric composites seems to be hindered as much by the absence of a comprehensive data base as by inadequacies and controversies in the mathematical and basic mechanics formulation.

While there is enormous literature related to the modeling of the creep behavior of polymers, there are few reviews on the modeling of creep in polymeric composites. Among the earlier investigations are the articles by Haplin (7) Schapery (8), and Dillard (9). Most of this literature utilizes linear viscoelastic models to describe the creep behavior of the composite.

 Another approach to the modeling of creep derives form plasticity theory, which was applied initially to metal matrix composites (10), (11). For uni-axially reinforced polymeric composites Sun, (12), (13) developed a simpli...ed single parameter plasticity model for creep in unidirectionally reinforced composites. Another unidirectional model was proposed by Robertson (14). These models were subsequently extended to laminated plates and thermal e¤ects by Gates (15), (16). Rate dependence was included by means of viscoplastic model by Gates and Sun (17), (18).

Viscoplastic constitutive equations presented earlier by the authors (19), (20),

were written explicitly and they involve many parameters, which signi...cantly in‡uence the behavior of the constitutive equations. Appropriate parameters must be determined accordingly, such that the accurate behaviors of the material can be expressed.

Some of the problems involved with the explicit constitutive models, from our earlier investigation (19), include:

(1) The models are simply based on the phenomenological investigation of material properties while real behaviors of material is very complex. Therefore, the model contains errors inevitably.
(2) All the models are limited in their mathematical capabilities when they are tackled as parameter identification problem, since they are written explicitly.
(3) Most of the models were veri...ed by experimental results of mechanical testing of metals, except Sun-Gates model which simulated the results of mechanical testing of polymer matrix composites. Composite materials which usually encounter extra parameters like fiber orientation, volume fraction, fiber-matrix interface,...etc. These parameters will raise the degree of complexity of the earlier constitutive models.

Bearing in mind the shortcoming of the phenomenological model, an alternative modeling technique, is to use a computation and knowledge representation paradigm, Neural Networks. Recently, the computational mechanics, which is quantitatively reliable in itself, has widened its feasibility to the practical engineering problems by merging the Artificial Neural Network's (ANN) flexibility. Neural networks architecture is a promising implicit-modeling scheme to replace the traditional explicit constitutive equations used to describe the material behavior. The rest of this investigation is dedicated to the design of implicit creep model by means of ANN to predict the creep behavior of the polymer matrix composites under different thermomechanical environments.

At this point we emphasize that the approach we followed, does not produce an "explicit formula" supplying the creep properties for each viscoplastic - tempreature-stress-strain law, but to the construction of an appropriate neural network, which, at a given time period "produces" a set of data corresponding to a viscoplastic conditions. This neural networks "learns", and if applied to another set of experimental data, may fulfill its task more accurately and shorter time. The present method may replace to some extent the experiments after a period of "learning". Moreover, the method developed here may replace the classical numerical methods for elastoplastic calcinations, since it takes better into account the experimental data and automatically improves itself through "learning".

The proposed neural-creep model strongly contradicts the reference to neural
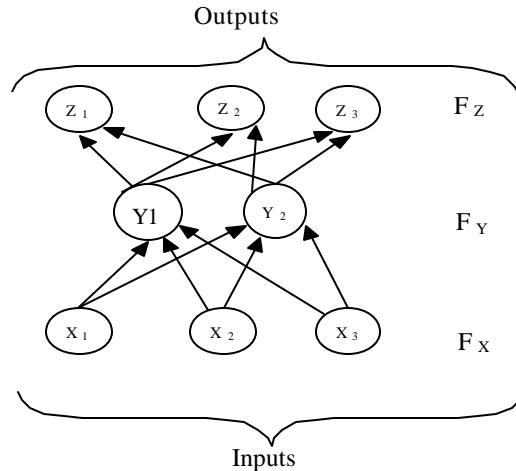
Fig. 1. Typical feedforward neural network composed of three layers

network as "a black box model", as referred to in the majority of material neural-models. The concept of black box arises in several investigations that use a "crude" training algorithm such as steepest descent where the learning rate is determined by a random walk approach. The current study eliminates this randomness by implementing recent but well-established numerical optimization techniques such as the conjugate gradient and the Truncated Newton methods. These techniques had several applications in numerical optimization, system theory, control systems, economy, metrology, and many more.

Surprisingly, not many applications in material science have adopted these powerful techniques. One explanation of the lack of interest is that: conjugate gradient algorithm is basically a nonconstrained optimization techniques that is usually used to find an optimal solution for a large-scale multi-dimensional problem, while most of the materials neural-models fall in the small-scale one-dimensional spectrum. (21), (22), (23), (24), (25), (26).

For the first time in neural network-based materials models, the powerful unconstrained multidimensional optimization algorithm i.e., truncated Newton method was introduced as training algorithm for neural networks. This novel application of the Truncated Newton was not cited before.

2 Neural Networks

Neural networks consist of processing elements and weighted connections, Figure-1 illustrates a typical neural network. Each layer in a neural network consists of a collection of processing elements (neurons); each processing element in a neural network collects the values from all of its input connections,

4

performs a prede...ned mathematical operation and produces a single output value. The neural network in Figure-1 has three layers: F_X, which consists of the neurons $fx_1; x_2; x_3g$; F_Y which consists of the neurons $fy_1; yg$; and F_Z which consists of the neurons $fz_1; z_2; z_3g$. The neurons are connected with weighted connections represented by the arrows in the figure. In Figure-1, there is a weighted connection from every F_X to every F_Y neuron and there is a weighted connection from every F_Y to every F_Z neuron. Each weighted connection acts as both label and a value. For example, in Figure-1 the connection from F_X neuron x_1 to the F_Y neuron y_1 is the connection weight w_{12}. The value of the connection weights is often determined by a neural network learning procedure, (27). It is through the adjustment of the connection weights that the neural network is able to learn. By performing the update operations for each of the neurons, the neural network is able to recall information.

The input to a neuron from another neuron is obtained by multiplying the output of the connected neuron by the weight of the connections between them. The artificial neuron then sums up all the weighted inputs coming to it. For example, the jth F_y PE in Figure-1 is y_j and the value of the PE is also y_j. The output value of the PE y_j in Figure-1 is a function of the outputs preceding layer F_x and the weights from F_x to y_j; W_j. Mathematically, the output of this PE is a function of its inputs and weights.

$$y_j = F(X; W_j) \tag{1}$$

The most common computation performed by a PE is a linear combination of the input value X with the abutting connection weights W_j in a dot product format, for example the output y_j in Figure-1 is computed as

$$y_j = f\left(\sum_{i=1}^{n} x_i w_{ij}\right) = f(X \cdot W_j) \tag{2}$$

where w_{ij} is the weight of the connection between the ith and jth processing elements. A processing element can have excitatory or inhibitory in‡uences on its neighboring elements. That is, it can either facilitate or hamper the activation of a connected element. Incoming excitation or inhibition from other neurons is combined (to the net input of the element) by summation. The new activation level is determined by evaluating a ...xed activation function f for this net input. The activation function is used to compare the weighted sum of inputs and the threshold value of that neuron.

The hyperbolic tangent is the most common activation function

$$f(x) = \tanh(\text{®}x) = \frac{e^{\text{®}x} \text{ i } e^{\text{i ®}x}}{e^{\text{®}x} + e^{\text{i ®}x}} \tag{3}$$

where $® > 0$. The hyperbolic tangent functions are useful in neural networks trained by backpropagation, which will be discussed in the next section, because the relationship between the value of the function at a point and the value of the derivative at that point reduces the computational burden during training.

2.1   Neural Networks Training: Backpropagation Algorithm

As opposed to a classical algorithm a neural network is not programmed, but is trained. In other words, if the problem at hand is a function $y = f(x)$ where y and x are, in general, vectors in a classical algorithm where x is the data and f is the form, while in a neural network x and y are the data and the function f is the unknown. In a neural network, f consists of the modes in which data is transmitted from one neuron to another and of the set of rules upon which the transformation of data within each neuron is based.

The main advantage of neural networks is the fact they are able to use some a priori unknown information hidden in data (but they are not able to extract it). The process of capturing the unknown information is called learning or training of neural networks. A training cycle consists of the following steps: An input vector is presented at the inputs together with a set of desired responses, one for each node, at the output layer. A forward pass is done and the errors or discrepancies, between the desired and actual response for each node in the output layer, are found. These are then used to determine weight changes in the net according to the prevailing learning rule. The best-known learning algorithm is the Backpropagation algorithm, (28), (29).

A popular measure of the error E for a single pattern, is the sum of the square di¤erences

$$E = \frac{1}{2} \sum_i (t_i \; i \; y_i)^2 \tag{4}$$

where t_i is the desired or target response on the ith unit and y_i is that actually produced on the same unit.

Figure-2 shows the three-layer feedforward backpropagation topology. The units in the input layer F_x serve only as distribution points; the input signal is simply passed through the weights on their outputs. Each unit in the hidden layer F_y and the output layer F_z produces the summation outputs.

Learning of backpropagation network assumes that each input pattern a_k is paired with a target pattern b_k representing the desired output. These are called training pairs. Before training is started, all the weights are assigned to a small random values usually in the range (-1,+1). This initialization pre-
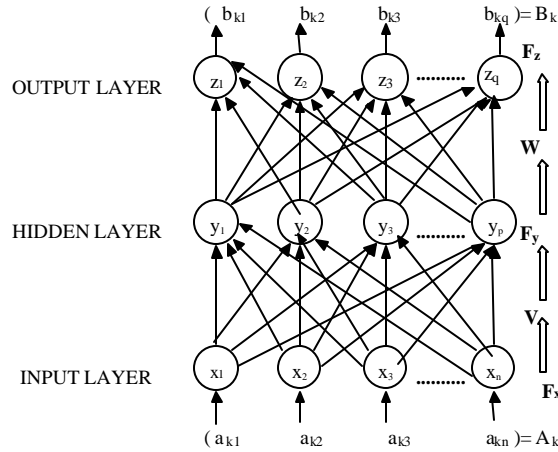
Fig. 2. Topology of the three layer feed-forward backpropagation

vents the network from getting saturated by large values of the gain. Learning is done by iteratively adjusting the weights of the connections in the network W and V, in order to minimize the cost function E. During the process, an input pattern is presented to the network and propagated forward to determine the actual output at the units in the output layer. An error signal for each output PE's is calculated and is backpropagated through the network in order to adjust the weights. The learning process continues until the networks responds with the sum of the squared error of the output signals is less than a predetermined value.

3   Neurocomputing and Optimization

3.1   The Steepest Descent with Momentum Algorithm

The standard backpropagation implements the steepest descent method (also called the gradient descent method). At each step of the steepest descent method the weights are adjusted in the direction in which the error function decrease most rapidly. This direction is determined by the gradient of the error surface at the current point in the weight space. In order to minimize the error function, the gradient vector is multiplied by (-1) because the gradient vector points to the maximum increasing error. The error function or the cost function that is used is the sum squared error; the sum of square di¤erences between the actual output and the desired output value for each unit in the output layer. The output error across all the F_z PEs is found using the cost function

7

$$E = \frac{1}{2}\sum_{j=1}^{q}(b_{kj} - z_j)^2 \qquad (5)$$

where b_{kj} is the desired output while z_j is the actual output. The output of an F_z PE, z, is computed as

$$z_j = \sum_{i=1}^{p} y_i w_{ij} \qquad (6)$$

and each F_y (hidden-layer) PE, is computed using

$$y_i = f\left(\sum_{h=1}^{n} a_{kh} v_{hi}\right) = f(r_i); \qquad r_i = \sum_{h=1}^{n} a_{kh} v_{hi} \qquad (7)$$

The hidden-layer activation function is a hyperbolic tangent function.

Implementing the steepest descent method, the weight adjustment is performed by moving along the cost function in the opposite direction of the gradient to a minimum of the input/output mapping producing the smallest amount of error. For example, in Figure-2 the connection weights between the F_y and the F_z PEs are adjusted by using the gradient

$$\frac{@E}{@w_{ij}} = \frac{@}{@w_{ij}}\left[\frac{1}{2}\sum_{j=1}^{q}(b_{kj} - z_j)^2\right] \qquad (8)$$
$$= (b_{kj} - z_j)y_i$$
$$= \pm_j y_i \ ; \ \pm_j = (b_{kj} - z_j)$$

where $\pm_j$ is the error for each F_z PE. Next, the adjustment to the connection weights between the F_x and F_y PEs are found by utilizing the chain rule of partial di¤erentiation, hence, we can calculate weigh changes for an arbitrary number of layers.

$$\frac{@E}{@v_{hi}} = \frac{@E}{@y_i}\frac{@y_i}{@r_i}\frac{@r_i}{@x_h}\frac{@x_h}{@v_{hi}} = \sum_{l=1}^{p}(b_{kl} - y_l)y_l w_{hl} f'(r_i)a_{kh} \qquad (9)$$

Having the gradient of the error (also referred to as error sensitivity), then the weight adjustments for the connections are updates in a negative direction to the gradient with a certain rate, as given by

$$w_{ij}^{new} = w_{ij}^{old} - ®\frac{@E}{@w_{ij}} \qquad (10)$$

and

8

Steepest descent with Momentum

$$w_{ij}^{new} = w_{ij}^{old} - \circledR \frac{\partial E}{\partial w_{ij}} + \mu \Phi w_{ij}^{old} \tag{12}$$

and

$$v_{hi}^{new} = v_{hi}^{old} - \bar{}\frac{\partial E}{\partial v_{hi}} + \mu \Phi v_{hi}^{old} \tag{13}$$

where μ is the momentum factor ($\mu \in [0; 1[$):

$$v_{hi}^{new} = v_{hi}^{old} - \bar{}\frac{\partial E}{\partial v_{hi}} \tag{11}$$

where $\circledR$ and $\bar{}$ are positive-valued constants that regulate the amount of adjustments made with each gradient move, they are called learning rates. The learning rates determine what amount of the calculated error sensitivity to weight change will be used for the weight correction. The momentum was introduced to allow for more prompt learning while minimizing unstable behavior. Here, the error function is modi...ed so that a portion of the previous weight is fed through to the current weight. This acts, in engineering terminology, as a low-pass ...lter on the weight terms since general trends are reinforced whereas oscillatory behavior is canceled out. This allows a low, normally slower, learning coe¢cient to be used, but creates faster learning. Momentum as a low pass ...lter allows the network to ignore small features in the error surface. Without momentum, a network may get stuck in a shallow local minimum (30). On the other hand, when using momentum term, the network can escape from such a minimum.

3.2   The Conjugate Gradient Algorithm

Since learning in realistic neural networks application often involves adjustment of several hundred weights, only optimization methods applicable to large-scale problems are relevant as alternative learning algorithms.

The general opinion in the numerical analysis community is that conjugate gradient methods are well suited to handle large scale problems in an e¤ective way, (31), (32). Several conjugate gradient algorithms have recently been introduced as learning algorithms in neural networks. Some earlier applications of conjugate gradient as training algorithm were carried out by Battiti, (33), and Johansson (34).

The conjugate gradient algorithm combines the advantages of simplicity of the steepest descent method and better convergence without the evaluation, inversion, and storage of the Hessian matrix. Among unconstrained optimization methods, it is widely acknowledged that the conjugate gradient method

Newton Method
FOR k=0, UNTIL convergence DO

² Solve iteratively for the search vector p:

$$H(w_k)p_k = \textvisiblespace g(w_k) \qquad (16)$$

² Set

$$w_{k+1} = w_k + p_k$$

is perhaps the easiest method applicable to large scale problems, (32). The algorithm provides a tool to update the learning rate to minimize the mean squared error, hence, the learning rate is updated at each epoch. The Polak-Ribiere version of the conjugate gradient algorithm is provided the Appendix

3.3 The Truncated Newton Algorithm

Consider the unconstrained optimization problem

$$minimize \ f(w) : w \in R^n \qquad (14)$$

where $f : R^n \to R$ is a nonlinear, twice continuously diﬀerentiable function. We wish to ﬁnd the local minimizer $w^\ast$, that is, a point $w^\ast$ for which there exist $\pm > 0$ so that

$$f(w^\ast) \cdot f(w) \quad for\ all\ w : \|w - w^\ast\| < \pm$$

A ﬁrst-order necessary condition for a local minimum of $f(w)$ is that

$$g(w) = 0 \qquad (15)$$

where $g : R^n \to R$, is the gradient vector of f. A well known method for solving this system of nonlinear equations is Newton's method, which starting with an initial guess w_0, computes the sequence of steps p_k and iterates w_k according to the Newton algoritm :

:

where $H(w_k)$ is the Hessian of the function $f(w_k)$. While this algorithm converges with a quadratic convergence rate , it is not deﬁned at a point where $H(w_k)$ is singular. Moreover, for nonconvex problems it does not necessarily generate a sequence of descent directions (that is directions satisfying $g(w_k)^T p_k < 0$): One approach to solve the system of equations-(16) is by using Cholesky factorization i.e. $H = LDL^T$, where L is a unit lower triangular matrix, while D is a diagonal matrix. The Cholesky factorization can often be

10

computed e₵ciently using sparse matrix techniques. However, it may be impractical to compute H itself due to storage and computational considerations. This problem arises in large-scale nonlinearly constrained minimization (35). A well known technique for the solution of large system of linear equations is the linear conjugate-gradient method of Hestenes and Stiefel (36). If the conjugate gradient iteration is stopped or truncated before the exact solution to the Newton equations has been found, a method by same name was proposed; Truncated Newton (TN). Truncated Newton methods were introduced in the early 1980s and have since been gaining popularity, see (37), (38), (39), (40), (41), (42), (43), (44), (45), (46), (47). They are based on the idea that an exact solution of the Newton equation at every step is unnecessary and can be computationally wasteful in the framework of a basic descent method. Any descent direction will su₵ce when the objective function is not well approximated by a convex quadratic. However, as a solution to the minimization problem is approached, more e¤ort in solution of the Newton equation may be warranted. The appeal of TN methods to scienti…c applications is their ability to exploit function structure to accelerate the convergence. Thus, the approximation for a nonzero residual norm; $r_k = \|r_k\| = \|H(w_k)p_k + g(w_k)\|$; is allowed at each step, and its size is monitored systematically according to the progress being made. This formulation leads to double- nested iteration structure: for every outer Newton iteration k (associated with w_k) there corresponds an inner loop for $\{p_k p_k^0; p_k^1; \ldots\ldots g\}$: As a computationally economical method for solving large positive de…nite linear system, the preconditioned conjugate gradient is the most suitable method for the inner loop in this context, (48). Function structure is introduced by using a preconditioner M that is a sparse approximation to the Hessian. If the Hessian matrix is available, a good generic choice of a preconditioner is based on an incomplete Cholesky factorization. The preconditioner is found by factoring the Hessian and ignoring some or all of the …ll-in that occurs during Gaussian elimination. It may be necessary to modify the factorization so that the preconditioner is positive de…nite, this idea is discussed in (49). Preconditioners can also be developed based on the partial separability in the objective function (A function $f(x)$ is partially separable if it can be written as the sum of functions $f_i(x)$, each of which has a large invariant subspace,(48). If neither of the preconditioners described above is possible (for example when Hessian information are not available), then the preconditioner is computed based on the quasi-Newton approximation, i.e. limited memory BFGS formula to update the Hessian approximation. This approach is described precisely in (41). The current study utilizes the Cholesky factorization to construct the preconditioner as proposed by Nash (39). The truncated Newton nested loops can be summarized as shown in the Appendixs, (50).There are three di¤erent ways in which the truncated Newton conjugate gradient inner iteration can terminate

² Case-1: The gradient vector g; points in a direction of negative curvature $gHg < 0$: In this case the inner iteration returns with $p = {}_i g$, the steepest

11

descent directions.

- Case-2: A direction of negative curvature is encountered in the CG iteration ($dHd < 0$) prior to satisfying the Truncated Newton termination criterion (step 5) in this case the CG iteration is terminated and p_k is taken as direction of descent.
- Case-3: The algorithm terminates by satisfying the Truncated Newton criterion.

3.3.1 Theoritical Basics of the Truncated -Newton Algorithm

The Truncated Newton algorithm has the following form

$$\nabla^2 f(w_k)p_k^N = -\nabla f(w_k) \text{ or } p_k^N = -\left[\nabla^2 f(w_k)\right]^{-1}\nabla f(w_k) \tag{17}$$
$$w_{k+1} = w_k + \alpha_k p_k$$

where α_k is the step size,and p_k is the descent direction. The problem can be viewed as the solution of a system of linear equations $Aw = b$, where $A = \nabla^2 f(w_k)$ i.e. the Hessian of $f(w_k)$; and $b = -\nabla f(w_k)$: A preconditioned conjugate -gradient is (C-G) used to iteratively solve the systems where the C-G iteration is truncated after a prescribed number of iterations. This can be done by monitoring the residual of the inner iteration r_k;

$$\|r_k\| = \|\nabla^2 f(w_k)p_k + \nabla f(w_k)\| \cdot \eta_k \|\nabla f(w_k)\| \tag{18}$$

where η_k is a prescribed accuracy criterion, and we conduct a truncated iteration, i.e., we stop prior to the exact solution to Newton equations being found. Another way to monitor truncation is via the decrease of the quadratic model

$$Q_k(p) \cdot f(w_k+p) \approx f(w) + \nabla f(w)p + \frac{1}{2}p^T\nabla^2 f(w)p \tag{19}$$

where at every Newton iteration we assume the TN algorithm approximates $f(w)$ by the ...rst three terms in a Taylor expansion of $f(w)$ about the point w_k: To avoid storing the Hessian matrix, one can consider the following discrete Hessian-vector product based on the following truncated Taylor expansion

$$\nabla f(w_k + hv) = \nabla f(w_k) + h\nabla^2 f(w_k)v + O(h^2) \tag{20}$$

Hence, the Hessian-vector product $\nabla^2 f(w_k)v$ is approximated by

$$\nabla^2 f(w_k)v = \lim_{h \to 0} \frac{\nabla f(w_k + hv) - \nabla f(w_k)}{h} \tag{21}$$

12

where h is chosen as

$$h = \frac{2^{\bar{\mathsf{p}_{2m}}}(1 + kw_k k}{kvk}$$

where ^2_m is the machine accuracy constant and v is a direction descent.
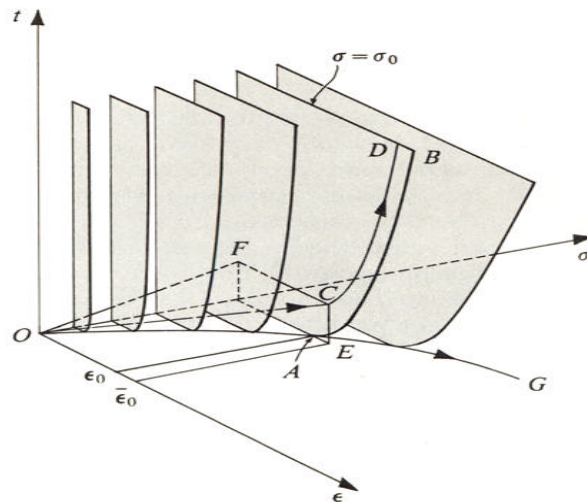
4   Neurocomputational Creep Model

Fig. 3. Stress-strain-time space with schematic creep curves.

The creep test is usually performed by loading a specimen using a Universal testing Machine to a speci...c load and the load is then maintained at a constant value and the strain is recorded. During creep experiment the load remains constant

$$\dot{P} = 0 \tag{22}$$

since the total strain can be decomposed into elastic and inelastic components;

$$" = "_e + "_i \tag{23}$$

the strain rate $\dot{"}$ is found by di¤erentiating Eq-23 and noting that $"_e$ is a constant

$$\dot{"} = \dot{"}_i \tag{24}$$

Figure-3 depicts the behavior of the creep strain at di¤erent stress levels, where $\frac{3}{4}_0$; $\frac{3}{4}_1$; and $\frac{3}{4}_2$ are normalized stress levels at which the creep testes were performed. The normalization of the stress level was done with respect

13

to the strength at the working temperature T^o. Hence, the input of the neural network was an array of 3£1 cells where the elements of each cell represents the [temperature; stress; time] respectively. The targets were chosen to be the corresponding values of the creep strain for each input cell.

The data sets were produced from a combination of eight temperatures (25°, 35°, 45°, 50°, 55°, 60°, 65°, and 75° C) that were selected according to the measured glass transition temperature, six normalized stress levels (30, 40, 50, 60, 70, and 80 %), and 36 time steps with 100 second increment ; i.e.100, 200, ..., 3600 s. The total number of data points produced was calculated as: 8(T°) £ 6(¾) £ 36(t) = 1728 [T°; ¾; t] input-target cells.

The neural network topology does not favor the"raw" data to be used for training. For example, in the standard backpropagation algorithm, if a sigmoidal function is used as the activation function, then the saturation limits are -1 and 1. Hence, if the training inputs have large values compared to these limits, the activation function will certainly be operating in a saturated mode and not allow the network to learn.

The input data [T°; ¾; t] and the targets [²] were normalized to values between -1 and 1 using the following formula

$$x_n = 2 \frac{x \ _i \ x_{min}}{x_{max} \ _i \ x_{min}} \ _i \ 1 \qquad (25)$$

Where x_n is the normalized value of the vector $x = [T°; ¾; t; ²]$, x_{min} and x_{max} are the minimum and maximum values in the database for the vector x. After scaling the 1728 cells of inputs-targets, the scaled results will be split into three subsets; one set will be used for training, another set for validation, and the last set for testing the network performance. The training and validation sets consist of 1200 (roughly 2=3 of the entire data sets) pairs of scaled data that covers the temperature ranges of 25°, 45°, 55°, 60°, and 75° C: These 1200 data points will be split into 800 pairs for training and 400 pairs for validation. The remaining 528 cells left; that covers the creep tests at temperatures: 35°, 50°, and 65 C° will be kept aside to test the neural network creep model after the network has been trained and validated . The scaled data sets should be randomized so that the training process of the network won't be a table look up problem, and to eliminate any bias that might exist in the training data set.

Based on the Universal Approximation Theorem, Hornik (51), proposed that: "A two hidden layer network is capable of approximating any useful function". Also, Hornik stated that the mapping power of feedforward neural network (FF NN), is not inherent in the choice of a speci...c activation function, rather it is the multilayer feed forward structure that leads to the general function

approximation capability.

Following the universal approximation theorem and the reasoning given by Hornik (52), the current investigation adopts a two-hidden layer neural network topology. The number of neurons at each hidden layer was obtained through training the network using a standard backpropagation algorithm with two design parameters: learning rate and momentum. The performance of several structures of neural networks is shown in Figure- 4. From this simulation it is obvious that, the [6-20-1] structure (six neurons at the ...rst hidden layer, twenty neurons at the second hidden layer and a single neuron at the output layer, Figure-5) achieved an optimal number of neurons. This optimal structure produced MSE = 0.12105, which is still higher than the preassigned error goal i.e.; $MSE_{goal} = 1 \times 10^{i\,5}$: The results of this crude network were used to monitor the performance index; i.e. the mean squared error, given by equation-(5). The structure of this optimal-size network is shown in Figure-5.

Fig. 4. Structure analysis for the two-hidden layers neural network based on the mean squared error for the 798 pairs of training data.

Both hidden layers have tansigmoidal activation functions ' $_2$ and ' $_2$ respectively as given by Eq-3, while the activation function of the output layer is a linear function ' $_3$. Thus, the objective function built through the proposed neural networks structure can be written in a compact notation as :

$$f_k(x; w) = \sum_{j=1}^{6} w_{jk} \,'_3 \left(@\,'_2 \left(@ \sum_{m=1;}^{20} w_{lm} \,'_1 (X) \right) \right)$$

$$\text{where } X = \sum_{i=1;}^{6} (w_{pi} x_i)$$

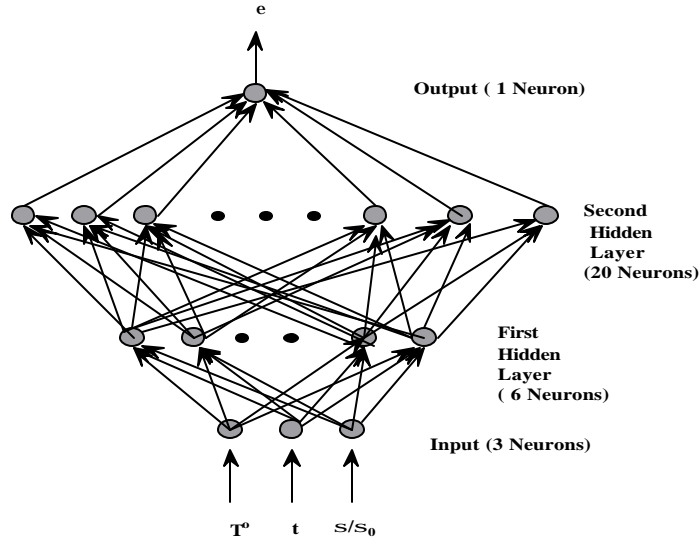$$p = 1; :::3; \quad m = 1; :::6, \ k = 1 \tag{26}$$

15

Fig. 5. The structure of the proposed neural model to predict the creep behavior of PMC.

and the corresponding objective function to be minimized is then given as the mean square error between the approximation function f and the actual target t for each training pair $k = 1, 2, \ldots 798$:

$$E(w; n) = \frac{1}{2} \sum_{k=1}^{798} (t_k - f_k)^T (t_k(n) - f_k) \tag{27}$$

This function should be minimized with respect to the weight values w_{jk}, w_{lm}, and w_{pi}, that can be stacked into one vector w consisting of : 3×6 values at ...rst hidden layer, 6×20 values at second hidden layer, and ...nally 20×1 values at the output layer, hence w is a variable-vector of 158 variables. Training the neural network consists of finding the optimal values of w that will minimize the error function $E(w)$ using any of the optimization techniques described before.

5   Results and Conclusion

Using the steepest descent algorithm with momentum, the network performance can be improved by finding optimal values for learning rate (®) and the momentum coefficient ‚. First the number of iterations (epochs) was extended to 1000 and the learning rates were set to di¤erent values (0.05, 0.1, 0.3, 0.5, and 0.8), similarly the momentum values varied from 0.05 to 0.8. The corresponding MSE surface can be visualized as shown in Figure-6. The visualized MSE surface at Figure-6 is highly nonsmooth. However, the learning

rate $\circledR = 0{:}10$ and momentum $_{¸} = 0{:}3$ generated the least MSE=0.119678, which is much higher than the preassigned MSE goal of $1 \; ¤ \; 10^{i \; 5}$:
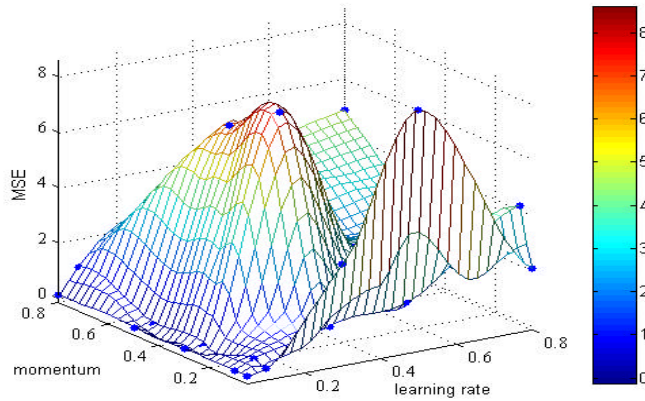

Fig. 6. The behavior of the MSE error as calculated using di¤erent values of learning rate and momentum for the steepest descent method.

Bearing in mind the problem associated with choosing an optimal combination of the training parameters, the implementation of the standard backpropagation algorithm to minimize the MSE is not the perfect learning algorithm for the implicit creep model. Validating a model in ANN requires to use some data points which were not utilized in the training phase, although, these data points still fall in the range of the training set. The validation phase makes use of the 400 data points that were left out of the 1200 data points initially generated. These validation data have to be preprocessed in the same scaling and randomizing fashion of the training data.

For testing the network we introduce a completely new data set that does not belong to the training and validation data. For example, one can test the network at T= 35 C$^{\text{o}}$ with stress levels 30%, 50%, and 80 % over one hour simulation time span. The performances for training, validation, and test sets is simulated as shown in Figure-7. The network was trained for 1000 epoch to check if the performance (MSE) for either validating or testing sets might diverge, which didn't happen as shown in Figure-7.

The results of MSE appear reasonable in terms of generalizing the ANN for new test sets, but in order to con...rm these results we need to compare the actual values for creep strain with those produced by the ANN. After scaling back the ANN results for creep strain, they were plotted together with the experimental values as shown in Figure-8.

Figure-8 shows clearly that the standard backpropagation algorithm failed to capture the creep behavior at 35$^{\text{o}}$C. Similar unsuccessful results were reported for the ANN creep model at T=50 C$^{\text{o}}$ and T=65 C$^{\text{o}}$:
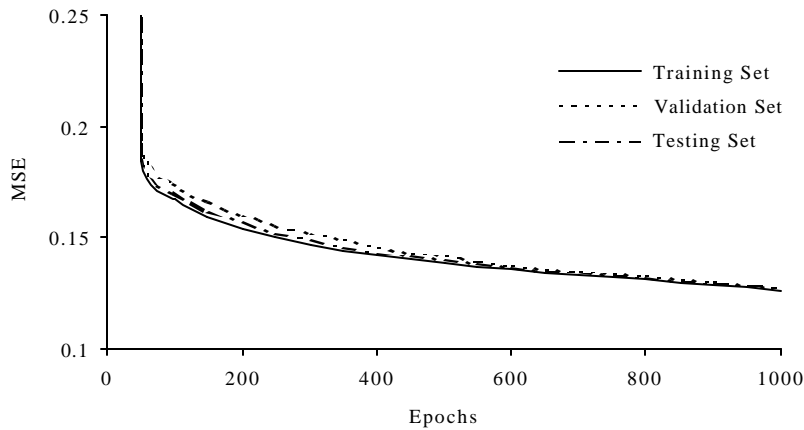
Fig. 7. MSE error for training, validation, and testing sets, for the [6-20-1] ANN based on the steepest descent with momentum backpropagation training algorithm.
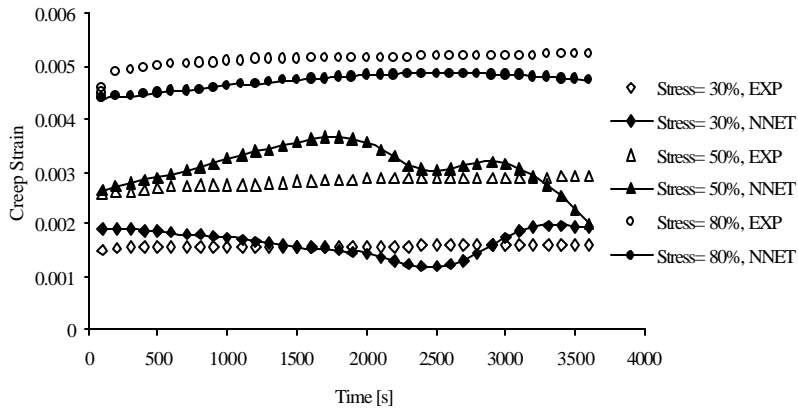

Fig. 8. Testing the [6-20-1] ANN, based on the steepest descent with momentum backpropagation, for T= 35C°; at 30%, 50%, and 80% stress levels respectively. The experimental creep strain is plotted together with the simulated results of the ANN.

By applying it to the implicit creep model, the conjugate gradient algorithm will be used to determine the weights updates. The [6-20-1] structure with tansgimoidal activation functions in a batch mode will be used; the gradient of the error function is computed after the entire training has been presented to the entire network.

Similarly, the training, validation, and testing sets as in the standard backpropagation were used. Compared to the standard backpropagation, the conjugate gradient minimization algorithm produced a smaller MSE for all the three phases of training, validation, and testing. Figure-9 shows that, the resulting MSE error for the conjugate gradient training algorithm is 50% less than that for the standard backpropagation. Another, important conclusion that can be drawn from Figure-9 is that, the conjugate gradient with line

search backpropagation is an order of magnitude faster than the steepest descent backpropagation. The conjugate gradient with line search required exactly 113 epochs for the MSE (for the training set) to drop to a value of 0.06997, compared to 1000 epochs required to reach a value of 0.12707 MSE for the standard backpropagation method.
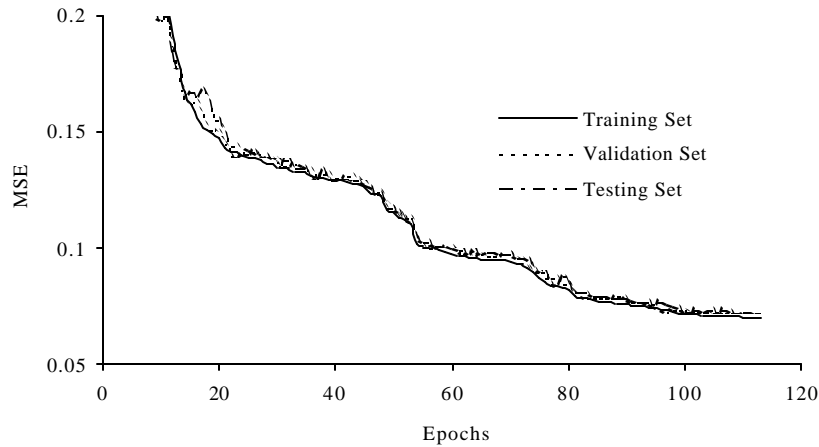

Fig. 9. MSE error for training, validation, and testing sets for the [6-20-1] ANN with backpropagation training algorithm that utilizes the conjugate gradient (Polak-Ribiere) algorithm.

We tested the backpropagation with conjugate gradient training algorithm for the implicit creep model at T=35 C^o for levels of stress of: 30%, 50%, and 80% respectively. The conjugate gradient based simulation produced far more acceptable results than those obtained via standard method, as shown in Figure-10. While the simulations for the cases of 30% and 50% stress levels look satisfactory, the ANN model results have a considerable error for the case of 80% stress level.

We conclude that the standard conjugate gradient algorithm discussed in the previous section was by far more superior than the standard backpropagation in both reducing the mean squared error in less number of epochs, and in better generalization for the network for the test data set. However, these attractive results were achieved at the expense of additional computational e¤ort, namely the line search technique in order to achieve the optimal learning rate which will be modi...ed at each successive step to reach the goal of minimizing the mean square error function.

Keeping the same architecture of the implicit creep model network; [6-20-1], and using the Truncated Newton minimization (see Appendix) as training algorithm, the gradient of the error function as a stopping criteria, the algorithm converged to a satisfactory error of 0.01323 after 160 epochs as
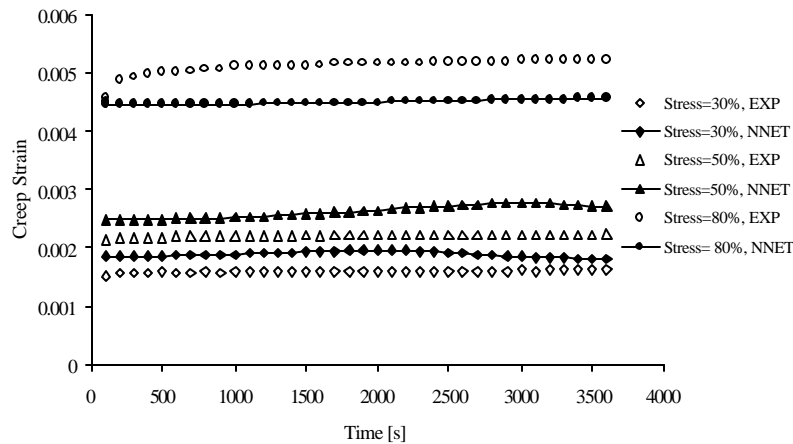
Fig. 10. Testing the [6-20-1] ANN, with nonlinear conjugate gradient (Polak-Ribere) backpropagation, for T= 35°C; at 30%, 50%, and 80% stress levels respectively. The experimental creep strain is plotted together with the simulated results of the ANN.

shown in Figure-11. The results of the simulation of the Truncated Newton minimization algorithm were very close to the actual experimental results as shown in Figure-12.
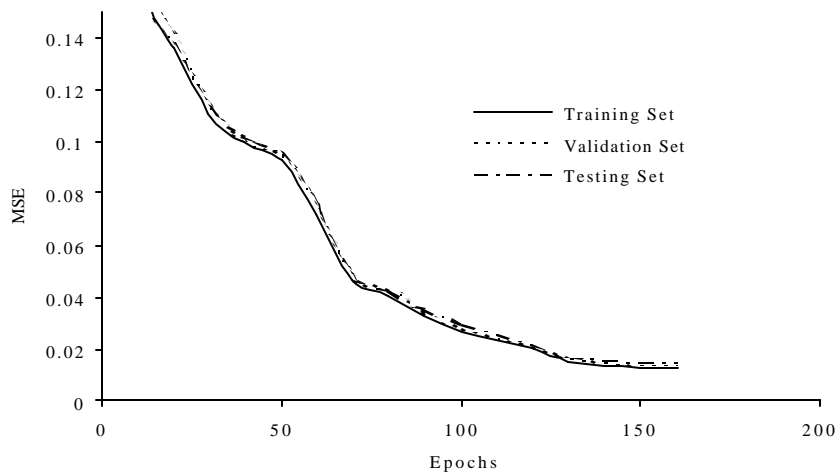

Fig. 11. MSE error for training, validation, and testing sets, for [6-20-1] ANN with backpropagation training algorithm that utilizes the Truncated Newton method

The performance of the Truncated Newton algorithm were compared to both the steepest descent and the nonlinear conjugate gradient as shown in Figures-13-14. The Truncated Newton algorithm attained the lowest MSE requiring only a relatively moderate number of epochs.

Considering the mean square error function described in equation-(27), we compared the performance of the steepest descent, Truncated Newton (TN),
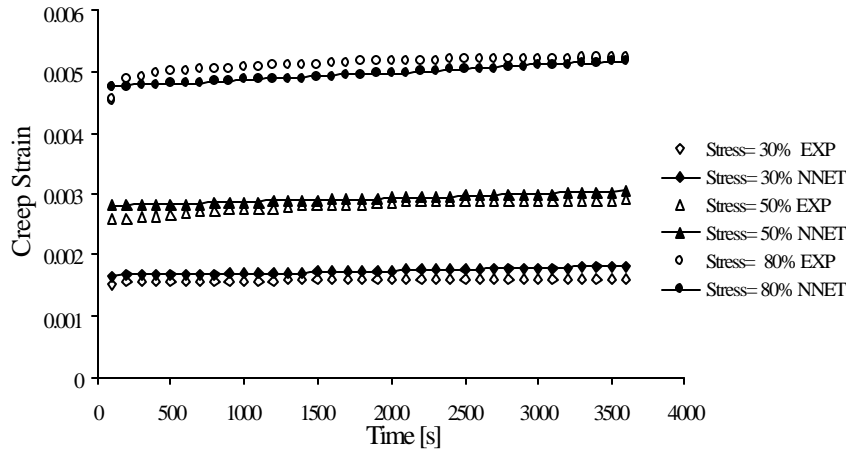
20

Fig. 12. Testing the [6-20-1] ANN, with Truncated Newton-based backpropagation, for T = 35°C;at 30%, 50%, and 80% stress levels. The experimental creep strain is plotted together with the simulated results of the ANN.

and the nonlinear conjugate gradient minimization method based on Polak Ribiere formula (CG-PR), all methods being allowed up to 350 iterations (the outer loop for the case of TN). The results show that TN algorithm converges by one order of magnitude faster than both steepest descent and CG-PR, and it attained a lower gradient norm as shown in Figure-13. Also, the TN algorithm signifiantly outperforms both the steepest descent and the CG-PR methods on two counts; the final accuracy achieved and the total number of function and gradient evaluations required to achieve a prescribed accuracy. The comparison is shown in Figures-13-and 14.

The numerical results reported here highlight the fact that the Truncated Newton algorithm using, in its inner-loop iteration, a conjugate gradient based preconditioner signi...cantly outperforms the standard nonlinear conjugate gradient in solving large-scale unconstrained minimization problems associated with neural networks models.

The neural-creep model is more cost efficient compared to the explicit viscoplastic model (Al-Haik and Garmestani, (19)); only one type of data is required, i.e., creep data at di¤erent thermomechanical histories, while the explicit viscoplastic model required both tensile tests data along with load relaxation data, and of course creep data is still required to verify the performance of the model. While the duration of tensile and load relaxation tests are short compared to the creep tests, the outcome of using tensile-load relaxation based viscoplastic model produced considerable errors when predicting the creep strain (53), (19). On the other hand, the results of the neural model, justi...es the cost of the longer duration creep tests.
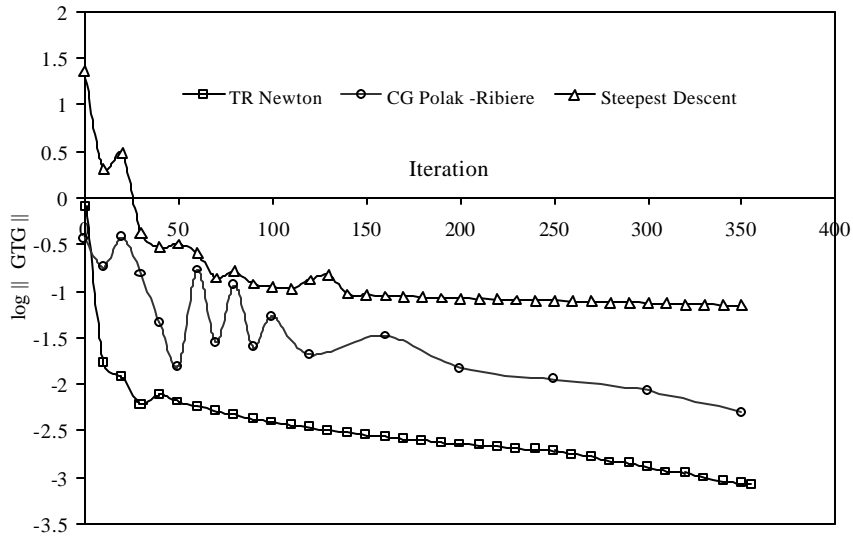
21

Fig. 13. Comparison of the gradient norm for the error function constructed by neural networks. Gradient was generated by steepest descen, Truncated Newton and Polak Rebiere conjugate gradient methods respectively.
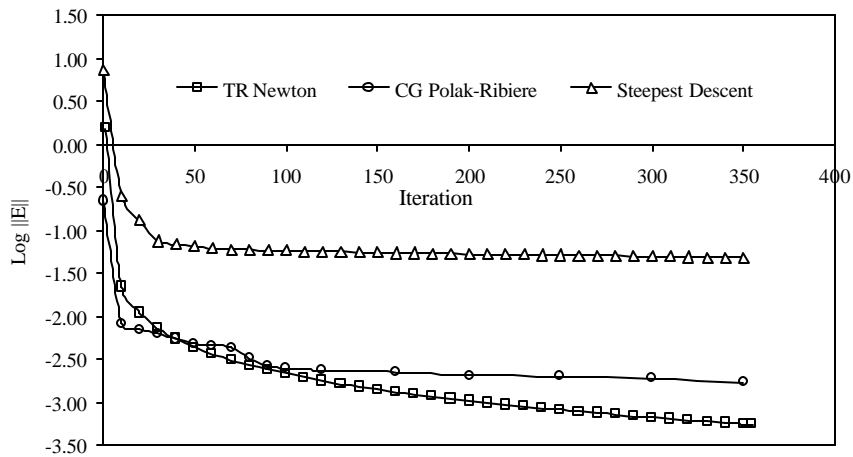

Fig. 14. The decrease of the neural networks mean squared error function by. Function was evaluated by steepest descent, Truncated Newton method and Polak Rebiere conjugate gradient methods respectively.

6

Algorithm 3 Inner Loop of the Truncated Newton Method

Algorithm 1 Polak- Ribiere Version of the Conjugate Gradient Algorithm for Training ANN :

(1) Choose the initial weight value $w(0)$.

(2) For $w(0)$, use backpropagation to compute the gradient vector $g(0)$:

(3) Set $s(0) = r(0) = -g(0)$:

(4) At epoch n use line search to find $\alpha(n)$ that minimize $J(\alpha)$ sufficiently. Here J is expressed as a function of α for fixed values of w and s:

(5) Update the weight vector

$$w(n + 1) = w(n) + \alpha(n)s(n) \tag{1}$$

(6) For $w(n + 1)$ use backpropagation to compute the update gradient vector $g(n + 1)$:

(7) Set $r(n + 1) = -g(n + 1)$:

(8) Use Polak-Ribiere method to calculate $\beta(n + 1)$

$$\beta(n + 1) = -\frac{r^T(n)(r(n) - r(n - 1))}{r^T(n - 1)r(n - 1)} \tag{2}$$

(9) Update the search direction vector

$$s(n + 1) = r(n + 1) + \beta(n + 1)s(n) \tag{3}$$

(10) Terminate the algorithm when the following condition is satisfied

$$\|r(n)\| \cdot \varepsilon \|r(0)\| \tag{4}$$

where ε is a prescribed accuracy .

(11) If the stopping criterion in step 10 is not satisfied, set $n - ! \quad n + 1$, and go back to step 3.

Algorithm 2 Outer Loop the Truncated Newton Method

(1) Initialization
- Set $k = 0$ and evaluate $f(w^0)$ and $g(w^0)$ for a given initial guess w^0:
- If $\|g(w^0)\| < 10^{-8}\max(1,\|w^0\|)$; exit algorithm. Where $\|:\|$ is the standard Euclidean norm divided by $\sqrt{n}$:

(2) Preparation
- Evaluate the preconditioner M at w^0:
- For computational efficiency the matrix/vector product $H(w_k)p_k$ can be computed satisfactorily by the following finite deference design of the gradient at expense of only one additional gradient evaluation per inner iteration

$$H(w)p = \frac{g(w^k + hv) - g(w^k)}{h} \tag{5}$$

where h is suitably chosen small number.

(3) Compute a search vector p by solving the Newton equation $Hp = -g$ approximately using preconditioned conjugate gradient with preconditioner M.

23

(4) Line Search:
- Compute step length by cubic interpolation or Wolf Armijo search

The sequence $\{p_j\}$ below represents the vectors used to construct p^k in step 3 of the outer loop.

(1) Initialization
- Set $j=1$, $p_1 = 0$; and $r_1 = -g$
- Set the parameter $\zeta_k = \min\{c_r = k; \|g\|\}$ and set It No: number of iterations for the truncation test in step 5.(cr=0.5, It=40 were used)

(2) The estimate for preconditioner to assure it is positive definite.
- Solve for z_j in $Mz_j = r_j$ by using the triangular system $Lw = r_j$ and $L^T z_j = D^{-1}w$
- set $d_j = z_j$

(3) Singularity test:
- Compute the matrix vector product $q_j = Hd_j$
- If either $\frac{r_j^T z_j}{\cdot} \cdot \epsilon$ or $\frac{d_j^T q_j}{\cdot} < \epsilon$ ($\epsilon = 10^{-10}$) : exit inner loop with $p^k = p_j$ (for j=1, set $p^k = -g^k$)

(4) Implement the following Descent Direction Test :
- Update the quantities

$$\alpha_j = r_j^T z_j = d_j^T q_j$$
$$p_{j+1} = p_j + \alpha_j d_j \tag{9}$$

- If $g^T p_{j+1} \geq g^T p_j + \epsilon$ exit inner loop with $p^k = p_j$ (for j=1, set $p^k = -g^k$))
  else
  update α_j and p_{j+1}

(5) Truncation test:
- Compute $r_{j+1} = r_j - \alpha_j q_j$
- If $\|r_{j+1}\| \cdot \zeta_k \|g\|$ or $j + 1 >$ Max Iteration Number.
  exit inner loop with search direction $p^k = p_{j+1}$

(6) Continuation
- Solve for z_{j+1} as in step 2
- Update $\beta_j = r_{j+1}^T z_{j+1} = r_j^T z_j$ and $d_{j+1} = z_{j+1} + \beta_j d_j$:
- Set $j = j + 1$ and go to step 3

References

[1] L. Franke, H. Meyer, Predicting the tensile strength and creep-rupture behavior of pultruded glass reinforced polymer rods, Journal of Materials Science 27 (1992) 4899.

[2] E. Shin, R. Morgan, Performance evaluation of glass ...ber-epoxy composites for durable goods applications, in: Proceedings of the 10th Annual ASM/ESD Advanced Composites Conference, 1994.

[3] R. Schapery, A method of viscoelastic stress analysis using elastic solutions, J. Franklin. Inst. 279 (74) (1965) 268.

[4] J. Corum, Durability of Polymer Matrix Composites for Automotive Structural Applications: A State of the Art Review, Oak Ridge National

Laboratory, 1995.

[5] R. Schapery, An engineering theory of nonlinear viscoelasticity with applications, Int. J. Solids. Struct 2 (1966) 407.

[6] D. Pertez, Y. Weitsman, Nonlinear viscoelastic characterization of fm-73 adhesive, J. Rheology 26 (3) (1966) 245.

[7] J. Haplin, Introduction to Viscoelasticity, Technomic Publishing Co., 1968.

[8] R. Schapery, Stress Analysis of Viscoelastic Composite Materials, Technomic Publishing Co., 1968.

[9] D. Dillard, Viscoelastic Behavior of Laminated Composite Materials, Elsevier Applied Science Publishers, Ltd., England, 1991.

[10] M. Liu, E. Krempl, A uniaxial viscoplastic model based on total strain and overstress, J. Mech. Phys.Solids 27 (1979) 377.

[11] E. Hart, H. Solomon, Load relaxation studies of polycrystalline high purity aluminum, Acta Metallurgica 21 (1973) 295.

[12] C. Sun, J. Chen, A micromechanical model for plastic behavior of ...brous composites, Comp. Sci. Tech. 40 (1991) 115.

[13] C. Sun, I. Chang, Modeling of elastic-plastic behavior of ldf and continuous ...ber reinforced as-4/peek composites, Comp. Sci. Tech. 43 (1992) 339.

[14] D. Robertson, S. Mall, Micromechanical analysis for thermoviscoplastic behavior of unidirectional ...brous composites, Comp. Sci. Tech. 50 (1994) 483.

[15] T. Gates, E¤ect of elevated temperature on the viscoplastic modeling of graphite-polymeric composites, in: T. Gold (Ed.), High Temperature and Environmental E¤ects on Polymeric Composites, no. 1174, ASTM, 1993, pp. 201–221.

[16] T. S. Gates, Matrix dominated stress-strain behavior in polymeric composites, in: Composites Materials: Testing and Design, Vol. 11, 1993, p. 177.

[17] T. Gates, C. Sun, Micromechanical characterization of nonlinear behavior of advanced polymer matrix composites, in: Composites Materials: Testing and Design, Vol. 12, 1996, p. 295.

[18] T. Gates, C. Sun, Elastic/viscoplastic consitutive model for ...ber reinforced thermoplastic composites, AIAA 29 (3) (1991) 457.

[19] H. G. M. AlHaik, M.R. Vaghar, M. Shahawy, Viscoplastic analysis of structural polymer composites using stress relaxation and creep data, Composites Part B 32 (2001) 165.

[20] M. AlHaik, H. Garmestani, Durability study of a polymeric composite material for structural applications, Journal of Polymeric Composites 22 (6) (2001) 779.

[21] A. Mukherjee, S. Biswas, Arti...cial neural network in prediction of mechanical behavior of concrete at hight temperature, Nuclear Engineering and Design 178 (4) (1997) 1.

[22] J. G. J. Ghaboussi, X. Wu, Knowledge-based modeling of materials be-

havior with neural networks, J. Engineering Mechanics 117 (1) (1991) 132.

[23] I. Yeh, Modeling of strength of high-performance concrete using arti...cial neural networks, Cement and Concrete research 28 (12) (1999) 1797.

[24] T. Furukawa, G. Yagawa, Implicit constitutive modelling for viscoplasticity using neural networks, International Journal For Numerical Methods In Engineering (1998) 195.

[25] C. Li, A. Ray, Neural network representation of fatigue damage dynamics, Smart Materials Structures 4 (1995) 126.

[26] X. Wu, Neural ntwork based material modeling, Ph.D. thesis, Department of Civil Engineering, University of Illinois at Urbana-Champaign, Urbana, IL (1991).

[27] P. Simpson, Foundation of neural networks, in: E. Sinencio, C. Lau (Eds.), Arti...cial Neural Networks: Paradigms Applications and Hardware Implementation, 1992, p. 3.

[28] P. Werbos, Backpropagation : Past and future, in: IEEE International Conferenmce of Neural Networks, Vol. 1, 1988, p. 343.

[29] B. Parker, Second order backpropagation, in: IEEE First International Conferenmce on Neural Networks, Vol. 2, 1987, p. 593.

[30] H. Demuth, M. Beale, Neural Network Toolbox for Use with Matlab, The Math Works Inc., Naticks MA, 1994.

[31] R. Fletcher, C. Reeves, Function minimization by conjugate gradients, Computer J. 7 (1964) 149.

[32] R. Fletcher, Practical Methods of Optimization, 2nd Edition, John Wiely and Sons., 1987.

[33] R. Battiti, Accelerated backpropagation learning: Two optimization methods, Complex Systems 3 (1989) 331.

[34] F. D. E.M. Johansson, D. Goodman, Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method, Int. J. Neural Systems 2 (4) (1991) 291.

[35] P. Gill, W. Murray, Newton-type methods for unconstrained and linearly constrained optimization, Mathematical Programming 28 (1974) 311.

[36] M. Hestenes, Conjugate Direction Method in Optimization, Springer-Verlag, Berlin, 1980.

[37] P. Dembo, T. Steihaug, Truncated newton algorithms for large scale unconstrained optimization, Math. Programming 26 (1983) 190.

[38] S. Nash, Preconditioning of truncated newton methods, SIAM J.Sci. Stat. Comput. 6 (1985) 599.

[39] S. Nash, Newton-type minimization via the lanczos method, SIAM J. Numer. Anal. 21 (1984) 770.

[40] S. Nash, A. Sofer, Block truncated-newton methods for parallel optimization, Math. Programming 45 (1989) 529.

[41] S. Nash, J. Nocedal, A numerical study of the limited memory bfgs method and the truncated-newton method for large scale optimization, SIAM J. Optimization 1 (1991) 358.

[42] T. Schlick, A. Fogelson, Tnpack: A truncated newton minimization package for large-scale problems: I. algorithm and usage, ACM Trans. on Math. Soft. 18 (1992) 46.

[43] X. Zou, ,I.M. Navon, M.Berger. P. Phua T. Schlick F. Le Dimet, Numerical experience with limited-memory quasi-Newton methods and truncated Newton methods, SIAM J. Optimization 3 (1993) 582.

[44] Z. Wang, I.M. Navon and F.X. Le Dimet, Truncated-Newton optimization algorithm in meteorology applications with analytic hessian/vector products, Computational Optimization and Applications 4 (1995) 241.

, [45] I.M. Navon, X.Zou, M. Berger, P. Phua, T.Schlick and F.X.Le Dimet, Numerical experience with limited memory quasi-newton and truncated newton methods, in: K. Phua (Ed.), Optimization Techniques and Applications, World Scienti...c Publishing Co., 1992, p. 33.

[46] I.M. Navon, X.Zou, M. Berger, P. Phua, T.Schlick and F.X.Le Dimet, , Testing for reliability and robustness of optimization codes for large scale optimization problems, in: K. Phua (Ed.), Optimization Techniques and Applications, World Scienti...c Publishing Co., 1992, p. 445.

[47] Z.Wang, K. K. Droegemeier, L. White and I.M.Navon, Application of a new adjoint Newton algorithm to the 3-D ARPS storm scale model using simulated data, Monthly Weather Review 125 (1997) 2460.

[48] S. Nash, A survey of truncated-newton methods, Journal of Computational and Applied Mathematics 124 (2000) 45.

[49] F. Campos, J. Rollett, Analysis of preconditioners for conjugate gradients through distribution of eginvalues, Int. J. Comp. Math. 58 (1995) 135.

[50] D. Xie, T. Schlick, E¢cient implementation of the truncated newton algorithm for large-scale chemistry application, SIAM J.Optim. 10 (1999) 132.

[51] M. S. K. Hornik, H. White, Multilayer feedforward networks are universal approximators, Neural Netowks 2 (1989) 359.

[52] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netowks 4 (1991) 251.

[53] M. Al-Haik, Durability of a polymer matrix composite : A neural networks approach, Ph.d. dissertation, Dept. of Mechanical Engineering, Florida State University, Tallahassee, FL (2002).