DEVELOPMENT OF AN INTEGRATED APPROACH COMBINING ARTIFICIAL NEURAL NETWORK MATERIAL BASED MODELING WITH FINITE ELEMENT ANALYSIS OF FORMING PROCESSES

A Dissertation

presented to

the Faculty of the Graduate School

University of Missouri-Columbia

---

in Partial Fulfillment

Of the Requirements for the Degree

Doctor of Philosophy

---

by

BRIAN SCOTT KESSLER

Dr. A. Sherif El-Gizawy, Dissertation Supervisor

Dr. D. Smith and Dr. K. Morsi, Co-Advisors

MAY 2005

©   Copyright by Brian Scott Kessler 2005

All Right Reserved

The undersigned, appointed by the Dean of the Graduate School,

have examined the dissertation entitled:

DEVELOPMENT OF AN INTEGRATED APPROACH COMBINING ARTIFICIAL NEURAL NETWORK MATERIAL BASED MODELING WITH FINITE ELEMENT ANALYSIS OF FORMING PROCESSES

Presented by Brian Scott Kessler

A candidate for the degree of Doctor of Philosophy

And hereby certify that in their opinion it is worthy of acceptance.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Dedication

My Wife, Kathy, who has cheerfully endeavored to put up with me through this long, drawn-out quest.

Also, to my kids, Anne and Dylan who serve to inspire the pursuit of learning through their boundless curiosity about all things.

Acknowledgements

Dr. Sherif El-Gizawy, regarding his help in putting this altogether in a coherent package and for the original idea behind the research.

Dr. Doug Smith, for his advice with finite element and help in securing the fellowship.

Dr. Khaled Morsi, for providing insights and information about a new and exciting class of materials.

Shane Vandendaele and Chris Fike at Weld Racing, for their cooperation and sharing of technical expertise.

Seid Koric at NCSA, for his advice and help with computational issues.

Dr. Ken Blundell, for offering suggestions and wisdom with things neural.

Dr. Bill Pedersen, for providing a sounding board whenever needed.

Richard Oberto, for help acquiring the all important data.

Dr. Cho, for advisement through the Ph.D. process.

Melanie Gerlach, for her efforts in securing misplaced equipment, software and training.

Kathy Kessler, for editorial comments regarding the dissertation.

This research was funded by the U.S. Department of Education,

Graduate Assistance in Areas of National Need.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEVELOPMENT OF AN INTEGRATED APPROACH COMBINING ARTIFICIAL NEURAL NETWORK MATERIAL BASED MODELING WITH FINITE ELEMENT ANALYSIS OF FORMING PROCESSES

Brian Scott Kessler

Dr. A. Sherif El-Gizawy, Dissertation Supervisor

ABSTRACT

The use of a finite element model for design and analysis of a metal forming processes is limited by the incorporated material model's ability to predict deformation behavior over a wide range of operating conditions. Conventionally generated rheological models prove deficient in several respects due to the difficulty in establishing complicated relations between many parameters. More recently, artificial neural networks (ANN) have been suggested as an effective means to overcome these difficulties. To this end, a robust ANN with the ability to determine flow stresses based on strain, strain rate, and temperature is developed and linked with finite element – based simulation model. Comparisons of this novel method with conventional means are carried out to demonstrate the advantages of this approach as applied to industrial applications.

The flow stress curves generated using the developed ANN method for 6061 alumimum show the typical behavior of high stacking fault energy materials, where the controlling softening mechanism is dynamic recovery (early strain hardening followed by a smooth transition to a plateau of stress). In contrast, the flow stress behavior of nickel aluminide exhibits the typical behavior of low stacking fault energy materials, where the controlling softening mechanism in hot working is dynamic recrystallization (early strain hardening to a peak stress followed by drop and oscillation of the flow stress about a steady average value).

A thermo-mechanical coupled finite element method (FEM) using the commercial code ABAQUS as a platform for development is introduced to simulate hot forming processes.

The FEM model is integrated with the developed ANN material based model in order to account for the effects of strain, strain rate, and temperature variations within the material during hot-forming. An industrial case study involves hot forging of an aftermarket automotive wheel made out of 6061 aluminum is used to evaluate the effectiveness of the integrated approach. The load-displacement curves predicted by the developed virtual model are in good agreement with the experimental observations of an industrial forging process.

The developed approach and knowledge gained from the present work, has a wide range of application in general, and is not limited to hot forming of the investigated materials. The new approach is applicable to all hot forming processes of different alloy systems.

1. Introduction

Finite element modeling of manufacturing processes has been gaining wider acceptance over the last several years. Modeling prior to the start of actual production can save considerable time, effort, and cost. While modeling may provide these benefits, it must be kept in mind that finite element software can only provide accurate simulations of a "real" process if appropriate material models are utilized. The present research aims at development of an integrated approach for generation of virtual models that are effective and precise for design and optimization of hot forming processes.

In the present work, a novel material model is presented and compared to conventional models. For lack of an exact mathematical material model, an intelligent algorithm, the artificial neural network (ANN), will be used in the present study. ANN is used to map relationships between the hot forming parameters and the flow stress of the material. The ANN learns the patterns and offers robust and adaptive processing capabilities by implementing learning and self-organization rules. A method based on characterization of artificial neural network models for prediction of flow stress of the target materials during high temperature forming will be developed. Robust ANNs have the ability to predict outputs between, and to some degree, outside the bounds established by a training set. For this application, values of strain, strain rate, and temperature not matching the family of curves used for training can be submitted to the network, and intermediate values of flow stress found. The conventional modeling approach requires each one of the curves to be fit to some form of hardening law and intermediate values interpolated by some means. The curve fitting process itself can be exceptionally tedious and in many cases does not produce particularly accurate fits of the data. The ANN is much simpler to implement.

An artificial neural network (ANN) will be developed and trained based on physical testing of two materials; 6061 aluminum and 396LZR nickel aluminide. This trained network will be then used as material model, which is linked with finite element-based model for virtual simulation of forming processes. To these ends, initially, a review of conventional material models and their limitations will be presented. The general implementation of a material model within the finite element method will also be discussed. The various factors leading to difficulties in addressing real problems will be summarized. The development and use of artificial neural networks will be covered with the specific aim of developing an unconventional material model for linking with finite element code. Simple compression of billets of both the aluminum and nickel aluminide will be modeled and comparisons will be made with published experimental data [1][2]. An industrial case study involving hot forging of a high performance aftermarket automotive wheel [3], will be used to verify the developed virtual models and to evaluate the advantages and limitations of the approach.

2. Background

2.1 Conventional Constitutive Modeling Techniques

Linear Elasticity

Within the linear elastic range, an isotropic material model only requires that the elastic modulus and Poisson's ratio are known. For many or most metals Hooke's law for the isotropic case, EQ 1, is all that is required for accurate modeling. Most forming processes require plastic deformation to take place, and hence an elastic model will not adequately describe material behavior. In fact, many times the material is modeled to have rigid behavior up to the onset of plastic behavior for purposes of simplification:

$$\begin{Bmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \varepsilon_{12} \\ \varepsilon_{13} \\ \varepsilon_{23} \end{Bmatrix} = \begin{bmatrix} 1/E & -v/E & -v/E & 0 & 0 & 0 \\ -v/E & 1/E & -v/E & 0 & 0 & 0 \\ --v/E & --v/E & 1/E & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/G & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/G & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/G \end{bmatrix} \begin{Bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{12} \\ \sigma_{13} \\ \sigma_{23} \end{Bmatrix} \quad \text{(EQ 1)}$$

where ε is strain, E is the elastic modulus, σ is stress, and G is found from EQ 2.

$$G = \frac{E}{2}(1 + v) \quad \text{(EQ 2)}$$

Inclusion of Plasticity

There are several commonly used constitutive relations as applied to material behavior. For the most part the models differ in their approach to describing the plastic portions of deformation (i.e. hardening behavior). The simplest would be to assume the material behaves linearly to its yield point in an elastic regime followed by purely plastic behavior (i.e., flat stress-strain relationship beyond yield) as shown in EQ 3.

$$\varepsilon = \varepsilon_e + \varepsilon_p = \frac{\sigma}{E} + \varepsilon_p \quad \text{(EQ 3)}$$

The plastic behavior could also be modeled assuming linear behavior, but with a slope less than the elastic portion (EQ 4). In each of these cases, elastic and plastic strains are treated separately and summed.

$$\varepsilon = \varepsilon_e + \varepsilon_p = \frac{\sigma}{E_1} + \frac{(\sigma - \sigma_0)}{E_2} \qquad \text{(EQ 4)}$$

Several significant difficulties arise when attempting to model the forging process. Real materials are complicated by nonlinear behavior due to loading the material beyond yield. Typically, forging processes assume that flow stresses, σ_f, follow a power law relation (EQ 5). Another approach, again treating elastic and plastic strains separately is the Ramberg-Osgood relation (EQ 6) which incorporates the power law treatment for plastic portion of deformation. The reference stress, K, and strain hardening exponent, n (C and m for the rate dependent form) are determined through curve fitting a stress/strain (σ_f/ε) diagram obtained from compression testing the material to be modeled.

$$\sigma_f = K\varepsilon^n \qquad \text{(EQ 5)}$$

$$\varepsilon = \frac{\sigma}{E} + \left(\frac{\sigma_f}{K}\right)^{\frac{1}{n}} \qquad \text{(EQ 6)}$$

If the material is being hot worked (i.e., as in many forging processes) the strain rate is substituted for strain, EQ 7. In this case, curve fits are generated from experiments conducted at constant strain rates.

$$\dot{\varepsilon} = \left(\frac{\sigma_f}{C}\right)^{\frac{1}{m}} \qquad \text{(EQ 7)}$$

To more accurately model behavior at elevated temperature the kinetic rate equation EQ 8 is frequently employed:

$$\dot{\varepsilon} = A\sigma_f^n e^{-\frac{Q}{RT}} \qquad \text{(EQ 8)}$$

where $\dot{\varepsilon}$ is the true strain rate as before, A a constant, Q is the activation energy, R the universal gas constant, and T the temperature is Kelvin. Again, compression tests are performed and the constants determined through curve fitting. Sellars and Tegart [4], suggest a somewhat more complicated form EQ 9 based on deformation as a thermally activated process:

$$\dot{\varepsilon} = A(\sinh \alpha \sigma_f)^{n'} e^{-\frac{Q}{RT}} \quad \text{(EQ 9)}$$

where an additional constant α is required. Again, the constants are found from fitting empirical data. Under small stresses EQ 9 reduces to EQ 8 while at high stresses EQ 9 becomes

$$\dot{\varepsilon} = A_2 e^{\alpha n' \sigma_f} e^{-\frac{Q}{RT}} = A_2 e^{\beta \sigma_f} e^{-\frac{Q}{RT}} \quad \text{(EQ 10)}$$

Other more complicated forms of the power law have been suggested to more adequately model behavior [5][6]. Severe difficulties arise in fitting the above equations to actual test data. Constant strain rate testing must be employed using several different strain rates and temperatures [7][8].

When constitutive models are used, one of the above equations is fit to empirical data. Obviously, some are easier to fit than others. EQ 9 requires that four constants be determined, and depending on the methods used, considerable errors may be generated [2]. In general, the fits are only properly obtained using steady state stresses. Typically, one strain value is selected and the strain rate versus flow stress plotted in log-log format to extract the exponent. Considerable time is involved in extracting each of the constants for a single equation. In addition, the reference stress needs to be determined for each of the curves. If the kinetic rate equation is to be fit, the activation energy, Q, also needs to be found. This is done through the use of an Arrhenius plot (i.e., log flow stress and the reciprocal of temperature) and again finding the slope. This value also varies based on strain, strain rate, and temperature. This still leaves two other constants to be determined. The exponent n‘ is approximately equal to 1/m at high stresses. The value of 1/m or n’ at low stresses is used for determining β and then in-turn α. Then by plotting the natural log of EQ 11, the Zener-Hollomon parameter, versus the natural log of sinh(ασ), n’ and A may be found.

$$Z = \dot{\varepsilon} e^{\frac{Q}{RT}} \quad \text{(EQ 11)}$$

As can be seen, attempts at modeling real materials has resulted in development of more and more complicated equations. As noted previously, constant strain rate testing is required at several different strain rates (e.g. 0.001, 0.01, 0.1, 1, and 10) for each of the temperatures of interest. In any case, severe difficulties arise in fitting the above equations to actual test data. Examples of fitting material test data will be given further along in this thesis.

2.2 Metallurgical Aspects of Forming

As put forth above, once a metal is stressed sufficiently it begins to flow. Factors such as strain, strain rate, and temperature are explicitly dependent on the deformation process itself. Several other factors, including but not limited to: chemical composition, metallurgical structure, phases present, grain size, segregation, and prior strain history, while not related to the process, do influence the material behavior [10].

The atomic arrangement of the crystalline system, or lattice, depends on the metal involved. Both the 6061 aluminum and the Ni_3Al investigated herein, have a face-centered cubic (FCC) structure as shown in Figure 1.

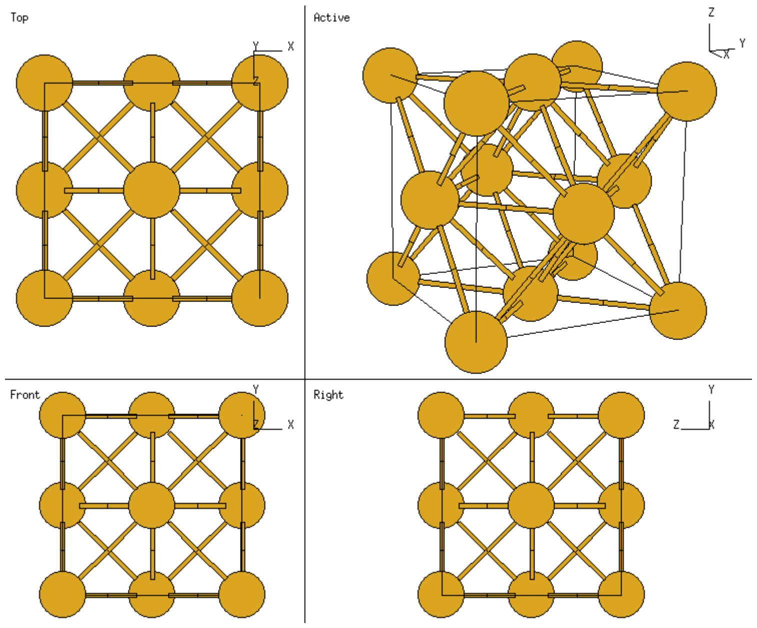


FIGURE 1. Face-centered cubic crystal structure [11].

During elastic loading the crystal lattice distorts, but due to the small displacements involved the material returns to its original position if the force is removed. Two mechanisms are involved once plastic deformation begins to take place; slip and twinning. Slip results from line defects, or dislocations, in a crystal being displaced along a lattice glide plane. The particular crystal structure determines the number of slip directions available during deformation. Twinning occurs when the lattice, instead of moving a constant distance by stepping over as with slip, deforms at an angle changing the orientation. Both slip and twinning can occur during plastic deformation, but twinning requires greater forces [12]. For FCC metals, slip is generally the mode available for plastic deformation with twins only formed during heating after cold working [13]. There are four slip planes, three slip directions, and twelve slip systems for the FCC structure [14].

Dislocations, or line defects, within a lattice decrease the amount of stresses required to produce plastic deformation. Initially, slip proceeds somewhat easier when dislocation defects are present, but over time dislocation tangles form as more and more dislocations are brought about due to slip. These dislocation tangles pile up hindering further movement. Strain-hardening results from this phenomenon as higher stresses are necessary to cause further deformation.

For forming operations, this increase in flow stress due to increases in dislocations can be lowered to a large degree by increasing the temperature of the workpiece. As thermal energy is added, atomic movement increases, and the movement of the lattice defects are thermally activated.

Heating decreases the effects of strain-hardening through the processes of recovery and recrystallization [15]. Thermally-activated processes take place at a finite rate and are hence governed by strain rate as the above equations suggest. During hot-forming operations some dislocations are formed and annihilated by others of opposite sign while remaining dislocations produce new subgrain boundaries [16]. In any case, the flow stress is reduced through the production of a less tangled set of dislocations.

Further reductions in flow stress can be achieved at a higher temperature due to recrystallization and grain growth. When the temperature is elevated sufficiently, new stress-free grains develop along grain boundaries, slip lines, and twin planes due to nuclei that form in areas of high atomic disarrangement [17]. If temperatures stay elevated for sufficient lengths of time, grain growth occurs.

Recovery, recrystallization, and grain growth can have significant effects on flow stresses depending on temperature, strain, and strain rate. Dynamic recovery and recrystallization can lead to strain flattening or strain softening as demonstrated by Figure 2.

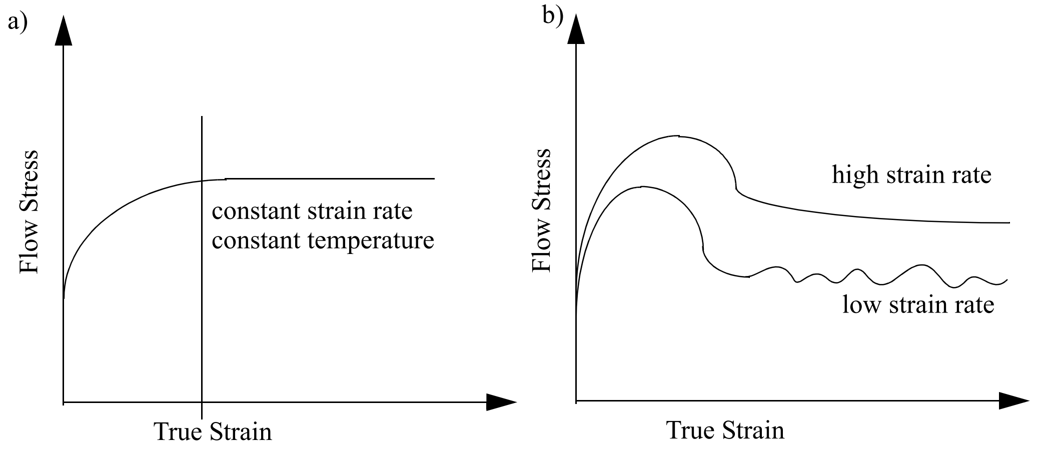


FIGURE 2. a) Strain flattening and b) strain softening resulting from dynamic recovery alone and both dynamic recovery and recrystallization, respectively [18].

In theory, a relationship exists such that a constant dislocation density is established through formation and annihilation which produces the constant flow stress as shown in Figure 2a. Serrated yielding, or oscillatory behavior of stress with increasing strain, subsequent to initial yielding occurs with some metals given appropriate conditions, but the mechanisms do not appear to be well understood. The kinetic rate equation presented previously is an attempt to reflect this behavior constitutively, but as will be demonstrated, fails to a large degree.

The above discussion, again points out the difficulty in relating many variable to predict flow stresses. The following section describes neural network development as a possible solution to the difficulties encountered.

2.3 Neural Networks

Basics

Over the last decade, several artificial intelligence tools such as artificial neural networks (ANN), fuzzy logic, and genetic algorithms (GA) have been introduced and applied in the field of manufacturing process engineering [19][20][21]. They provide for more accurate models than the available analytical ones. More recently, artificial neural networks (ANN) have been proposed to describe the material flow stress under the considered processing conditions [22][23].

As shown in Figure 3, the general idea behind artificial neural networks is to emulate the signal processing scheme used by nature. Several dendrites accept input that a given neuron processes before exiting at the axon. The axon then in-turn transmits a signal to another neuron's dendrites. In this way, information is processed or modified appropriately as it passes through the nervous system. It has been noted that very small children have a much greater ability to synthesize and organize sensory input into meaningful relations than even the fastest computers running exceptionally complicated code. Animal brains are composed of a large number (e.g., greater than 100 billion in the human brain [24]) of interconnected neurons. The vast interconnectedness produced by many thousands, millions, or billions of neurons interacting with each other allows for learning from experience by reinforcing connections that produce output consistent with the experience.

Artificial neural networks attempt to borrow this structure to produce computing structures, through software implementation or computer architecture, that have the ability to learn through experience or training. In similar fashion, a system of neurons is set-up so as to receive inputs and produce a modified output (i.e., act as a transfer function). There has been a recent resurgence of interest in neural networks. Artificial neural networks have performance characteristics similar to biological neural networks and are based on the following assumptions [25]:

- information processing occurs at many simple elements called neurons.
- signals are passed between neurons over connection links.
- each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.
- each neuron applies an activation function (usually nonlinear) to its net input (sum of weighted input signals) to determine its output signal.

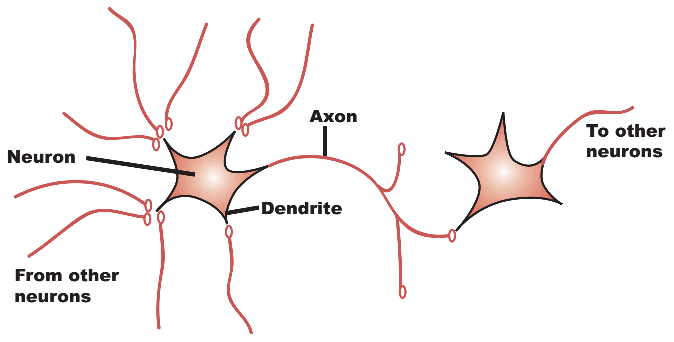


FIGURE 3. Communication between single neurons as found in nature.

A schematic of a simple multilayer artificial neural network is shown in Figure 4. Each of the inputs is connected to each of the first hidden layer neurons and each of the first hidden layer neurons connects to each of the second hidden layer neurons. Finally, the second hidden layer combines to form a single output.

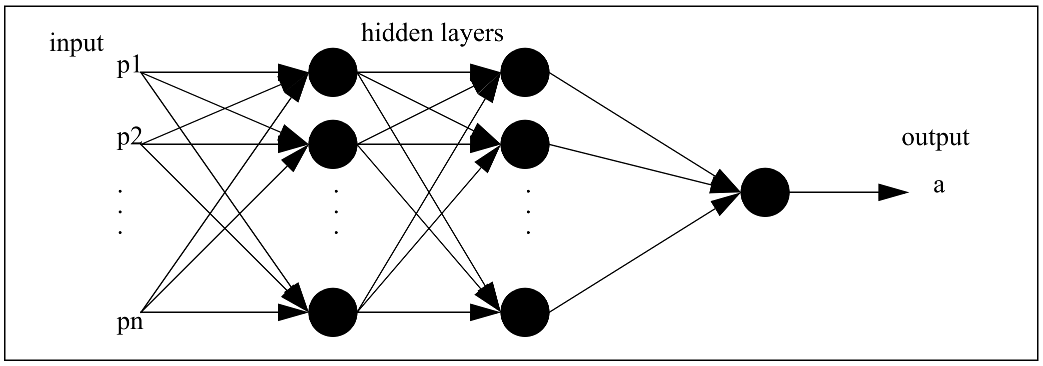


FIGURE 4. Schematic of a simple artificial neural network architecture.

Initial Development, Simple Neurons, and Their Architectures

The earliest artificial neurons have been named McCulloch-Pitts neurons after their developers [26]. Their neuron produces an output of 0 or 1 based on whether the sum of its input is greater than a particular threshold. Their networks had to be designed and did not have the ability to be trained. Rosenblatt [27] and others in the 1950s began to develop networks called perceptrons. They provided a learning rule that established weights and biases for each neuron in the network based on training. Perceptrons are limited in their abilities due to the simple transfer or activation function utilized. Networks of perceptrons have been typically used for categorizing items, which has proved to be one of neural networks greatest strengths.

It should be noted at this point, that a relatively famous paper published by Marvin Minsky and Seymour Papert seemed to prove that perceptrons were only capable of distinguishing linear-separable patterns and could not be trained to handle the XOR function [28]. Due to this perceived limitation, research into neural networks slowed to a standstill for several years. Later researchers established that Minsky and Papert were incorrect, which eventually lead to the reinvigoration of the field.

To expand of some of the above using MATLAB notation, EQ 12 defines the transfer function as shown in Figure 5.

$$a = hardlim(n) = \begin{pmatrix} 1 \text{ if } n \geq 0 \\ 0 \text{ otherwise} \end{pmatrix} \qquad \text{(EQ 12)}$$

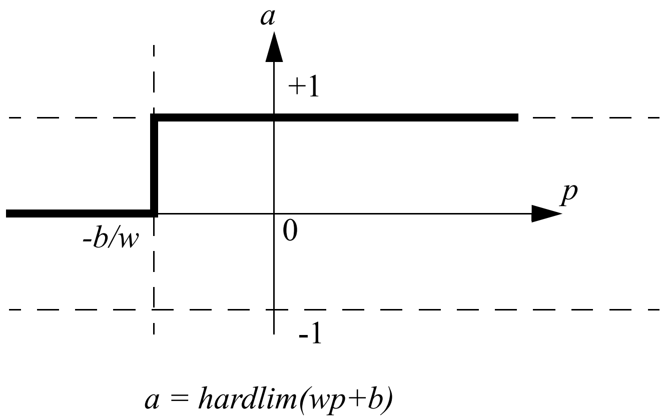
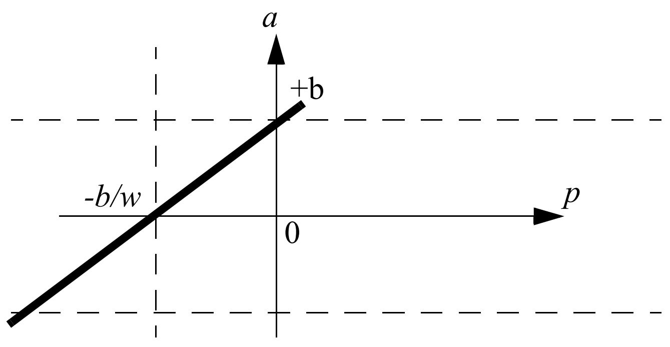


FIGURE 5. *Hardlim* transfer function within MATLAB, where **W** is the weight(s), **p** the input(s), and b the bias.

A single-neuron perceptron can be used to separate inputs into two categories while a multi-neuron perceptron can separate inputs into many categories (2^S, where S equals number of neurons). A linear transfer function is frequently employed for use as a linear filter. Again, using MATLAB notation, Figure 6 shows the graphical interpretation for *purelin*.

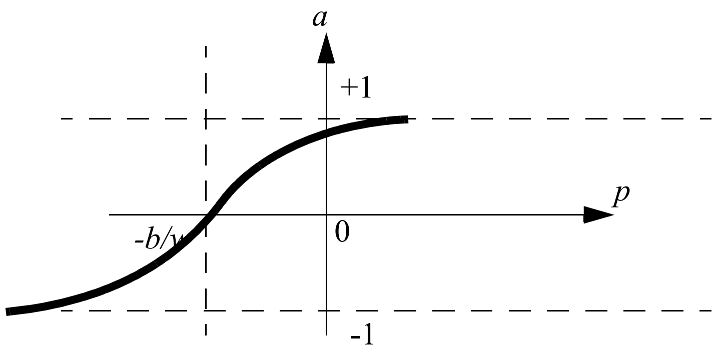


$a = purelin(wp+b)$

FIGURE 6. Linear transfer function, single-input *purelin* neuron.

Another commonly employed transfer function, tan-sigmoid, is provided by EQ 13 and in shown graphically in Figure 7.

$$a = \frac{2}{1 + e^{-2n}} - 1 \qquad \text{(EQ 13)}$$



$\boldsymbol{a} = tansig(\boldsymbol{w}p+\boldsymbol{b})$

FIGURE 7. Tan-sigmoid transfer function, single-input *tansig* neuron.

Figure 8 shows MATLAB notation and structure for a simple two layer network using a tan-sigmoid transfer function in the hidden layer and a linear transfer function in the output layer. This particular example is of importance due to the fact that "any continuous function can be represented by a two-layer network using a sigmoid hidden layer feeding a linear-output layer [29]."

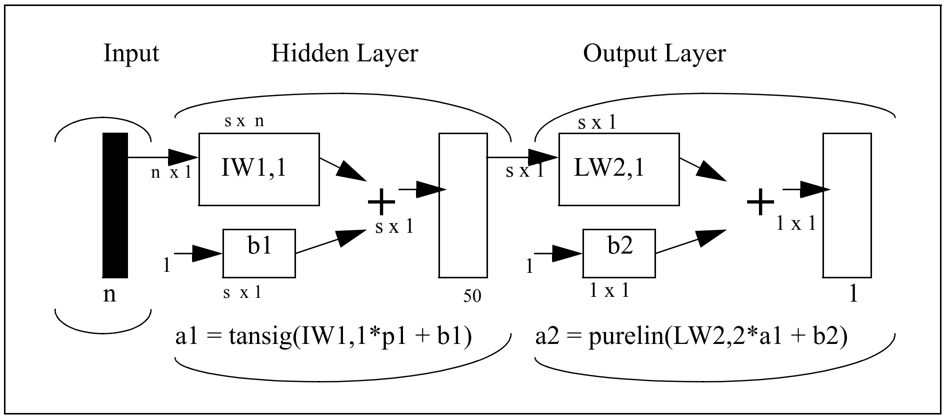


FIGURE 8. Schematic of a single-layer neural network within MATLAB; s equals the number of neurons and n the number inputs [30].

The *tansig* transfer function will produce output restricted to a range of -1 and +1. The *purelin* function has the ability to then scale, up or down, the final output to match any value, hence their usefulness when used in conjunction. As an example, the material model required for forging requires the determination of flow stress as it depends on strain, strain rate, and temperature; EQ 14 shows the matrix form for the hidden layer with strain, strain rate, and temperature as inputs. The hyperbolic tangent function is shown in this case and performs the same as *tansig*, but is more computationally expensive when used within MATLAB (tanh is used for the Fortran subroutine code generated for the FEA implementation).

$$\tanh\left(\begin{bmatrix} w1_{11} & w1_{12} & w1_{13} \\ w1_{21} & w1_{22} & w1_{23} \\ \dots & \dots & \dots \\ w1_{s1} & w1_{s2} & w1_{s3} \end{bmatrix}\begin{bmatrix} \varepsilon \\ \dot{\varepsilon} \\ T \end{bmatrix} + \begin{bmatrix} b1_1 \\ b1_2 \\ \dots \\ b1_s \end{bmatrix}\right) = \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_s \end{bmatrix} \qquad \text{(EQ 14)}$$

The subscript *s* refers to the particular neuron (i.e., *s* equals the number of neurons per input value in a layer). The hidden-layer values, *a*, are then fed into the output layer as shown in EQ 15 which results in a single value for the flow stress. The dot product in this case functions the same as a linear transfer function.

$$\sigma = \begin{bmatrix} w2_1 & w2_2 & \ldots & w2_s \end{bmatrix} \bullet \begin{bmatrix} a_1 \\ a_2 \\ \ldots \\ a_s \end{bmatrix} + b2 \qquad \text{(EQ 15)}$$

Several other transfer functions are available within MATLAB and others can be developed by the user, but due to the extensive abilities of *tansig,* others were not investigated at this time. Once the network is set-up, the weights and biases need to be determined (i.e., the network trained) to produce the required output. Training is many times accomplished through back propagation. The training set is repeatedly presented to the net and the net output compared to the target. At each iteration, or epoch, the error between the network output and the desired target is computed, normally the mean squared error. This error is then used to adjust the weights and biases of the last layer first, then the next to last layer and so on. Depending on the nature of the data used in the training set, the network may be trained adequately in a few epochs or tens of thousands of epochs may be required. The particulars of the backpropagation algorithm are described in the next section.

Feedforward Backpropagation

Feedforward backpropagation (FBP) networks are commonly utilized for function approximation. The presentation above describes the feedforward portion of the network. Backpropagation refers to the particular method of adjusting or correcting the weights and biases to produce a network output consistent with the training set. It should be emphasized that the network performance is inherently dependent on the quality of the training set. The training set needs to adequately represent the system or curves to be modeled. Another feature of FBP is due to the training algorithm itself, the transfer functions used must be differentiable.

Back Propagation Algorithm

One iteration of the basic algorithm can be represented as EQ 16:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \boldsymbol{g}_k \qquad \text{(EQ 16)}$$

where x_k is a vector of current weights and biases, g_k is the current gradient, and α_k is the learning rate. The algorithm requires a performance index for determining how well the network output approximates the training set. As stated previously, the general idea is to decrease the error with mean squared error (MSE) used in this case. If the network is supplied with a training set represented by:

$$\{\boldsymbol{p}_1, \boldsymbol{t}_1\}, \{\boldsymbol{p}_2, \boldsymbol{t}_2\}, \ldots, \{\boldsymbol{p}_Q, \boldsymbol{t}_Q\} \tag{EQ 17}$$

where p_Q is a vector input to the network and t_Q is a corresponding target output. The mean squared error can be computed by comparing the network output with the target and is given by:

$$F(\boldsymbol{x}) = E(e^2) = E[(t-a)^2] \tag{EQ 18}$$

where x is given by:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{b} \end{bmatrix} \tag{EQ 19}$$

and *w* is a vector representing a particular layer of weights and *b* the associated biases. The idea is to have the performance function, F(x) or MSE, decrease with each iteration. A more general form is required if multiple outputs are involved and is given by EQ 20.

$$F(\boldsymbol{x}) = E(\boldsymbol{e}^T\boldsymbol{e}) = E[(\boldsymbol{t}-\boldsymbol{a})^T(\boldsymbol{t}-\boldsymbol{a})] \tag{EQ 20}$$

This can be approximated by:

$$\hat{F}(\boldsymbol{x}) = (\boldsymbol{t}(k)-\boldsymbol{a}(k))^T(\boldsymbol{t}(k)-\boldsymbol{a}(k)) = \boldsymbol{e}^T(k)\boldsymbol{e}(k) \tag{EQ 21}$$

where the expectation of the squared error is replaced by the squared error at iteration *k*.

To find the minimum error, a form of the steepest descent algorithm is used and applied as follows:

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial}{\partial w^m_{i,j}}\hat{F} \tag{EQ 22}$$

$$b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial}{\partial b^m_i}\hat{F} \tag{EQ 23}$$

where α is the learning rate. The partial derivatives present in EQ 22 and EQ 23 are determined through the chain rule of calculus and are found to be:

$$\frac{\partial}{\partial w_{i,j}^{m}}\hat{F} = \frac{\partial}{\partial n_i^m}\hat{F} \times \frac{\partial}{\partial w_{i,j}^{m}} n_i^m \tag{EQ 24}$$

$$\frac{\partial}{\partial b_i^m}\hat{F} = \frac{\partial}{\partial n_i^m}\hat{F} \times \frac{\partial}{\partial b_i^m} n_i^m \tag{EQ 25}$$

The partials are used in this instance because the error is an indirect function of the weights and biases. This arises only with multi-layer networks. The second term in both of the above can be found from:

$$n_i^m = \sum_{j=1}^{s^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m \tag{EQ 26}$$

which results in:

$$\frac{\partial}{\partial w_{i,j}^{m}} n_i^m = a_j^{m-1}, \frac{\partial}{\partial b_i^m} n_i^m = 1 \tag{EQ 27}$$

Now defining sensitivity as:

$$s_i^m \equiv \frac{\partial}{\partial n_i^m}\hat{F} \tag{EQ 28}$$

Substituting EQ 28 into EQ 24 and EQ 25 results in:

$$\frac{\partial}{\partial w_{i,j}^{m}}\hat{F} = s_i^m a_j^{m-1} \tag{EQ 29}$$

$$\frac{\partial}{\partial b_i^m}\hat{F} = s_i^m \tag{EQ 30}$$

Now, the steepest descent algorithm can be represented by:

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha s_i^m a_j^{m-1} \tag{EQ 31}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha s_i^m \tag{EQ 32}$$

Or in matrix form:

$$\boldsymbol{W}^m(k+1) = \boldsymbol{W}^m(k) - \alpha \boldsymbol{s}^m(\boldsymbol{a}^{m-1})^T \tag{EQ 33}$$

$$\boldsymbol{b}^m(k+1) = \boldsymbol{b}^m(k) - \alpha \boldsymbol{s}^m \tag{EQ 34}$$

where

$$s^m \equiv \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{n}^m}\hat{F} = \begin{bmatrix} \frac{\mathrm{d}}{\mathrm{d}n_1^m}\hat{F} \\ \frac{\mathrm{d}}{\mathrm{d}n_2^m}\hat{F} \\ \ldots \\ \frac{\mathrm{d}}{\mathrm{d}n_{s^m}^m}\hat{F} \end{bmatrix} \tag{EQ 35}$$

The sensitivities must now be computed. The recurrent relationship that exists between a layer and the previous one is the idea behind backpropagation. To calculate the recurrent relation, the Jacobian matrix is utilized:

$$\frac{\partial \boldsymbol{n}^{m+1}}{\partial \boldsymbol{n}^m} = \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \ldots & \frac{\partial n_1^{m+1}}{\partial n_{s^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \ldots & \frac{\partial n_2^{m+1}}{\partial n_{s^m}^m} \\ \ldots & \ldots & \ldots & \ldots \\ \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_2^m} & \ldots & \frac{\partial n_{s^{m+1}}^{m+1}}{\partial n_{s^m}^m} \end{bmatrix} \tag{EQ 36}$$

An expression for the matrix is required, so considering the *i,j* element of the matrix:

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial}{\partial n_j^m}\left(\sum_{l=1}^{s^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1}\right) = w_{i,j}^{m+1}\frac{\partial a_j^m}{\partial n_j^m} = w_{i,j}^{m+1}\frac{\partial}{\partial n_j^m} f^m(n_j^m) \tag{EQ 37}$$

Letting

$$\dot{f}^m(n_j^m) = \frac{\partial}{\partial n_j^m} f^m(n_j^m) \qquad \text{(EQ 38)}$$

and substituting EQ 38 into EQ 37 results in:

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m(n_j^m) \qquad \text{(EQ 39)}$$

which allows the Jacobian to be written as

$$\frac{\partial \boldsymbol{n}_i^{m+1}}{\partial \boldsymbol{n}_j^m} = \boldsymbol{W}^{m+1} \dot{\boldsymbol{F}}^m(\boldsymbol{n}^m) \qquad \text{(EQ 40)}$$

where

$$\dot{\boldsymbol{F}}^m(\boldsymbol{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \dots & \dots & & \dots \\ 0 & 0 & \dots & \dot{f}^m(n_{s^m}^m) \end{bmatrix} \qquad \text{(EQ 41)}$$

The recurrence relationship for the sensitivities may now be written in the following form:

$$\boldsymbol{s}^m = \frac{\partial}{\partial \boldsymbol{n}^m} \hat{F} = \left(\frac{\mathrm{d}\boldsymbol{n}^{m+1}}{\mathrm{d}\boldsymbol{n}^{\boldsymbol{m}}} \right)^T \frac{\partial}{\partial \boldsymbol{n}^{m+1}} \hat{F} = \dot{\boldsymbol{F}}^m(\boldsymbol{n}^m)(\boldsymbol{W}^{m+1})^T \frac{\partial}{\partial \boldsymbol{n}^{m+1}} \hat{F} \qquad \text{(EQ 42)}$$

or

$$\boldsymbol{s}^m = \dot{\boldsymbol{F}}^m(\boldsymbol{n}^m)(\boldsymbol{W}^{m+1})^T \boldsymbol{s}^{m+1} \qquad \text{(EQ 43)}$$

At this point, it can easily be recognized that the sensitivities are found through backpropagating each from the last layer to the first. The starting point, $\boldsymbol{s}^m$, must also be found using the final layer.

$$s_i^M = \frac{\partial}{\partial n_i^M}\hat{F} = \frac{\partial}{\partial n_i^M}(\boldsymbol{t}-\boldsymbol{a})^T(\boldsymbol{t}-\boldsymbol{a}) = \frac{\partial}{\partial n_i^M}\sum_{j=1}^{s^M}(t_j-a_j)^2 = -2(t_i-a_i)\frac{\partial a_i}{\partial n_i^M} \qquad \text{(EQ 44)}$$

Now using the relation

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial}{\partial n_i^M}f^M(n_i^M) = \dot{f}^M(n_i^M) \qquad \text{(EQ 45)}$$

substituting

$$s_i^M = -2(t_i-a_i)\dot{f}^M(n_i^M) \qquad \text{(EQ 46)}$$

or in matrix form

$$\boldsymbol{s}^M = -2\dot{\boldsymbol{F}}^M(\boldsymbol{n}^M)(\boldsymbol{t}-\boldsymbol{a}) \qquad \text{(EQ 47)}$$

Finally, all of the equations required for carrying out backpropagation have been developed.

Training

Training itself is the process of repeated applications of the backpropagation algorithm until the error becomes acceptable or some other criteria is achieved. Each training iteration is termed an *epoch*. In summary, backpropagation is carried out through the following:

- assign initial weights and biases (typically small random numbers)
- input is feed forward through the network
- based on the error produced, the sensitivity is calculated at the last layer using EQ 47
- each prior layer sensitivity is calculated using EQ 43
- the weights are updated using EQ 33
- the biases are updated using EQ 34

Modeling Difficulties

The network architecture determines to a large extent how well it will model a particular set of data. For example, as the number of inflection points increases, a network comprised of a single sigmoid hidden layer followed by a linear output layer, would require the number of neurons in the hidden layer to be increased. If the number of neurons is excessive, the network may produce wild swings developing a greater number of inflections than the data, referred to as overfitting (see Figure 9). The network is said to generalize well when it reacts properly to new data that is close, but not exactly the same as the training set. As a guide, the network should have fewer parameters (i.e., total number of weights and biases) than data points in the train set [31].

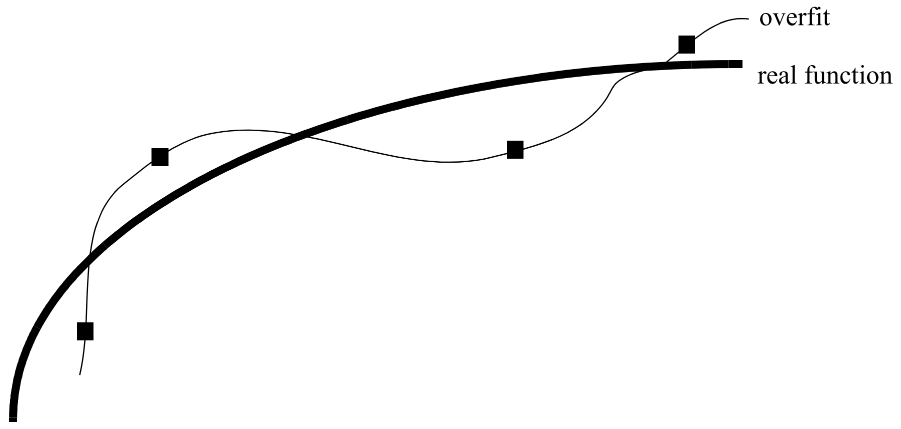


FIGURE 9. Example demonstrating overfitting.

Convergence, another issue, can result when training produces a network that may have found a local minimum, but not a global minimum. The learning rate specified and many times the initial conditions (i.e., starting values of the weights and biases) can produce a non-ideal solution as multi-layer networks may have many local minima. One solution is to run several trials with differing initial conditions to search for an optimum solution.

The basic backpropagation algorithm as shown above is an approximate steepest descent method. It can be an exceptionally slow minimization method and others exist that exhibit much faster convergence. Obtaining a converged solution is fundamentally an optimization problem, and as such, solutions are frequently borrowed from optimization techniques. One of the more common is explained in the next section.

Levenberg-Marquardt Algorithm

Commonly referred to as LM, Levenberg-Marquardt refers to a particular implementation of the backpropagation algorithm which employs a variation of Newton's method. Where the steep descent algorithm is based on a first order Taylor series expansion, Newton's method uses a second order Taylor series:

$$F(\boldsymbol{x}_{k+1}) \approx F(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k) \approx F(\boldsymbol{x}_k) + \boldsymbol{g}_k^T \Delta\boldsymbol{x}_k + \frac{1}{2}\Delta\boldsymbol{x}_k^T \boldsymbol{A}_k \Delta\boldsymbol{x}_k \qquad \text{(EQ 48)}$$

Taking the gradient of EQ 48 with respect to $\Delta\boldsymbol{x_k}$ and setting it equal to zero yields:

$$\boldsymbol{g}_k + \boldsymbol{A}_k \Delta\boldsymbol{x}_k = 0 \qquad \text{(EQ 49)}$$

and then solving for $\Delta\boldsymbol{x_k}$

$$\Delta\boldsymbol{x}_k = -\boldsymbol{A}_k^{-1}\boldsymbol{g}_k \qquad \text{(EQ 50)}$$

So, now defining Newton's method as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{A}_k^{-1}\boldsymbol{g}_k \qquad \text{(EQ 51)}$$

where the gradient and Hessian are defined as EQ 52 and EQ 53, respectively:

$$\boldsymbol{g}_k \equiv \nabla F(\boldsymbol{x})\big|_{\boldsymbol{x} = \boldsymbol{x}_k} \qquad \text{(EQ 52)}$$

$$\boldsymbol{A}_k \equiv \nabla^2 F(\boldsymbol{x})\big|_{\boldsymbol{x} = \boldsymbol{x}_k} \qquad \text{(EQ 53)}$$

Assuming F(**x**) is the sum of squares function:

$$F(\boldsymbol{x}) = \sum_{i=1}^{N} v_i^2(\boldsymbol{x}) = \boldsymbol{v}^T(\boldsymbol{x})\boldsymbol{v}(\boldsymbol{x}) \qquad \text{(EQ 54)}$$

with the *j*th element of the gradient being

$$[\nabla F(\boldsymbol{x})]_j = \frac{\partial}{\partial x_j}F(\boldsymbol{x}) = 2\sum_{i=1}^{N} v_i(\boldsymbol{x})\frac{\partial}{\partial x_j}v_i(\boldsymbol{x}) \qquad \text{(EQ 55)}$$

and writing the gradient in matrix form:

$$\nabla F(\boldsymbol{x}) = 2\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{v}(\boldsymbol{x}) \qquad \text{(EQ 56)}$$

where

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}v_1(\boldsymbol{x}) & \frac{\partial}{\partial x_2}v_1(\boldsymbol{x}) & \dots & \frac{\partial}{\partial n_n}v_1(\boldsymbol{x}) \\ \frac{\partial}{\partial x_1}v_2(\boldsymbol{x}) & \frac{\partial}{\partial x_2}v_2(\boldsymbol{x}) & \dots & \frac{\partial}{\partial x_n}v_2(\boldsymbol{x}) \\ \dots & \dots & \dots & \dots \\ \frac{\partial}{\partial x_1}v_N(\boldsymbol{x}) & \frac{\partial}{\partial x_2}v_N(\boldsymbol{x}) & \dots & \frac{\partial}{\partial x_n}v_N(\boldsymbol{x}) \end{bmatrix} \qquad \text{(EQ 57)}$$

Newton’s method is of second order and therefore also requires the Hessian matrix. The *k*, *j* element of the Hessian:

$$[\nabla^2 F(\boldsymbol{x})]_{k,j} = \frac{\partial^2}{\partial x_k \partial x_j}F(\boldsymbol{x}) = 2\sum_{i=1}^{N}\left\{\frac{\partial}{\partial x_k}v_i(\boldsymbol{x})\frac{\partial}{\partial x_j}v_i(\boldsymbol{x}) + v_i(\boldsymbol{x})\frac{\partial^2}{\partial x_k \partial x_j}v_i(\boldsymbol{x})\right\} \qquad \text{(EQ 58)}$$

or in matrix form

$$\nabla^2 F(\boldsymbol{x}) = 2\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x}) + 2\boldsymbol{S}(\boldsymbol{x}) \qquad \text{(EQ 59)}$$

where

$$\boldsymbol{S}(\boldsymbol{x}) = \sum_{i=1}^{N} v_i(\boldsymbol{x})\nabla^2 v_i(\boldsymbol{x}) \qquad \text{(EQ 60)}$$

If ***S(x)*** is assumed to be small, then the Hessian simplifies to

$$\nabla^2 F(\boldsymbol{x}) \cong 2\boldsymbol{J}^T(\boldsymbol{x})\boldsymbol{J}(\boldsymbol{x}) \qquad \text{(EQ 61)}$$

The Gauss-Newton method results from substituting EQ 61 and EQ 56 into EQ 51:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [2\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)]^{-1}2\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{v}(\boldsymbol{x}_k) = \boldsymbol{x}_k - [\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)]^{-1}\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{v}(\boldsymbol{x}_k) \qquad \text{(EQ 62)}$$

As can be seen from EQ 62, only the Jacobian remains in the Gauss-Newton method as the Hessian has been eliminated and computational efficiency has been improved now that second derivatives are not calculated.

However, another problem has arisen in that the approximate Hessian matrix, $\mathbf{H}=\mathbf{J}^\mathbf{T}\mathbf{J}$**,** may not be invertible. Using the following:

$$\boldsymbol{G} = \boldsymbol{H} + \mu \boldsymbol{I} \qquad \text{(EQ 63)}$$

The eigenvalues, λ_i, and eigenvectors of **G** are the same as **H**. If μ is increased such that $(\lambda_i + \mu)>0$ for all *i*, then **G** will be positive definite and hence invertible. It is this last modification that results in the Levenberg-Marquardt algorithm [32] as given by EQ 64 or EQ 65.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k) + \mu_k]^{-1}\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{v}(\boldsymbol{x}_k) \qquad \text{(EQ 64)}$$

$$\Delta\boldsymbol{x}_k = -[\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k) + \mu_k]^{-1}\boldsymbol{J}^T(\boldsymbol{x}_k)\boldsymbol{v}(\boldsymbol{x}_k) \qquad \text{(EQ 65)}$$

As μ_K increases, the algorithm approaches the steepest descent method and if μ_K is decreased to zero, the algorithm becomes Gauss-Newton.

The LM algorithm is thus similar to the standard backpropagation algorithm, except the Jacobian now is determined by computing the derivatives of the errors with respect to the weights and biases as opposed to finding the derivatives of the squared errors. So the Jacobian for a multi-layer Levenberg-Marquardt becomes:

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \frac{\partial e_{1,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,1}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{2,1}}{\partial w^1_{1,1}} & \frac{\partial e_{2,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{2,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{2,1}}{\partial b^1_1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{\partial e_{S^M,1}}{\partial w^1_{1,1}} & \frac{\partial e_{S^M,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{S^M,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{S^M,1}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{1,2}}{\partial w^1_{1,1}} & \frac{\partial e_{1,2}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,2}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,2}}{\partial b^1_1} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \quad \text{(EQ 66)}$$

In addition, the sensitivities for Levenberg-Marquardt differ and is given by:

$$\tilde{s}^m_{i,h} \equiv \frac{\partial v_h}{\partial n^m_{i,q}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \quad \text{(EQ 67)}$$

where

$$h = (q-1)S^M + k \quad \text{(EQ 68)}$$

the Jacobian elements for the weights are now found to be

$$[J]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w^m_{i,j}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times a^{m-1}_{j,q} \quad \text{(EQ 69)}$$

or for biases

$$[J]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b^m_i} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h} \quad \text{(EQ 70)}$$

As before, the Marquardt sensitivity for the final layer also found

$$\tilde{s}^{M}_{i,h} \equiv \frac{\partial v_h}{\partial n^{M}_{i,q}} = \frac{\partial e_{k,q}}{\partial n^{M}_{i,q}} = \frac{\partial}{\partial n^{M}_{i,q}}(t_{k,q} - a^{M}_{k,q}) = \frac{\partial a^{M}_{k,q}}{\partial n^{M}_{i,q}} = \begin{pmatrix} -\dot{f}^{m}(n^{m}_{i,q}) \text{ for i=k} \\ 0 \text{ for } \mathrm{i} \neq k \end{pmatrix} \qquad \text{(EQ 71)}$$

Now the network can be initialized with

$$\tilde{\boldsymbol{S}}^{M}_{q} = -\dot{\boldsymbol{F}}^{M}(\boldsymbol{n}^{M}_{q}) \qquad \text{(EQ 72)}$$

where the right side is defined previously by EQ 41. The columns of the matrix $\tilde{\boldsymbol{S}}^{M}_{q}$ are backpropagated through the network using EQ 42 to produce one row of the Jacobian. The columns may also be backpropagated together using

$$\tilde{\boldsymbol{S}}^{m}_{q} = \dot{\boldsymbol{F}}^{m}(\boldsymbol{n}^{m}_{q})(\boldsymbol{W}^{m+1})\tilde{\boldsymbol{S}}^{m+1}_{q} \qquad \text{(EQ 73)}$$

The total Marquardt sensitivity matrices for each layer are formed through augmenting the matrices from each input:

$$\tilde{\boldsymbol{S}}^{m} = \left[\tilde{\boldsymbol{S}}^{m}_{1} \middle| \tilde{\boldsymbol{S}}^{m}_{2} \middle| \ldots \tilde{\boldsymbol{S}}^{m}_{Q}\right] \qquad \text{(EQ 74)}$$

In summary, Levenberg-Marquardt is carried out iteratively as follows:

- assign initial weights and biases (typically small random numbers)
- input is feed forward through the network
- compute to the sum of squared errors using EQ 54
- compute the Jacobian, EQ 66, and the sensitivities using EQ 72 and then EQ 67
- find the elements of the Jacobian using EQ 69 and EQ 70 (i.e., modify weights and biases)
- solve for $\Delta\boldsymbol{x}_k$ using EQ 65
- find the new squared errors, if smaller, compute $\frac{\mu}{\nu}$ and let $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$, if not then compute $\mu\nu$ and refind $\Delta\boldsymbol{x}_k$.

Other Improvements

While the Levenberg-Marquardt significantly increases training rate, it by itself does not completely address several of the difficulties previously mentioned. Speed of training and overall performance may be improved through several other means.

Variables Involved

If prior knowledge about possible relations between particular input variables and expected output exists, it is many times beneficial to include those relations through modification of the inputs. In the case of flow stresses, it is known that the log of the stress may relate linearly to the log of the strain rate. So, it may make sense to provide the network with log strains or strain rates in addition to or instead of strains and strain rates.

Normalization

Additionally scaling or normalizing, EQ 75, the input variables and/or training values to range from 0 to 1 or -1 to 1 may enable the transfer functions to better handle the data, though with the added complication of re-processing the output from the trained network. Of course, subsequent queries of the network require normalizing inputs and de-normalizing outputs.

$$x_n = \frac{x - x_{min}}{x_{max} - x_{min}} - \frac{1}{2} \qquad \text{(EQ 75)}$$

Early Stopping

The early stopping technique splits the data into thirds; target, validation, and test. The targets are used as before with the error being determined. In addition, the error on the validation set is also found. While training, the error on the targets will always go down, but once overfitting starts to occur, the error on the validation set will increase. Training will be stopped after a certain number of epochs that continue to produce this divergence. The test set can further be used to compare one model to another (i.e., various numbers of neurons, layers, etc.). Overfitting generally occurs due to having a larger number of parameter, weights and biases, than necessary to fit the data. Early stopping does not aid in reducing the number of parameters.

Bayesian Regularization

Bayesian regularization addresses overfitting, or improving generalization, by attempting to reduce the model complexity to the minimum necessary for reasonable performance. Bayesian methods automatically incorporate the principle of "Occam’s razor" which states that the simplest model that adequately fits the data should be preferred over more complex representations [33][34]. MacKay suggests that early stopping, which tries to prevent overlearning, is really "patching up a bad model" as opposed to finding the proper model [35]. Bayesian regularization also allows all of the compiled data to be used in training without the need of splitting into subsets.

Bayesian regularization adds a term to the performance function, or squared errors, used previously. If E_D represents the squared error and E_W the sum of squares of the weights, then a modified performance index can be developed EQ 76.

$$F = \beta E_D + \alpha E_W \tag{EQ 76}$$

where α and β are objective function parameters. As β grows larger and α grows smaller, then network errors are forced to be smaller. If the reverse is true, training attempts to minimize the squared weights. Decreasing the values of the weights aids in smoothing the network response and should improve generalization.

Current Research

There is been a small, but increasing effort to utilize ANNs in the development of predictive constitutive material models. Examples include the predictions of strength in superalloys [36], modeling of fatigue crack growth in superalloys [37], determination of martensite start temperatures for steel [38], analysis of continuous cooling transformation of steel [39], and Jominy correlations with hardenability of steel during heat treatment [39], just to site a few. In each of the above, neural networks appear to produce models that appropriately model behavior and in many cases enable the inclusion of variables that have proved difficult or impossible to incorporate within conventional approaches.

Forming processes have also been modeled using ANNs. From steel processing, the hot rolling of cast steel into final shape has been aided by utilizing a neural network that predicts yield and tensile strength based on 108 parameters [40]. These variables include such factors as chemical composition and various rolling conditions. Another similar model utilizes microstructural information to determine ferrite grain size and property distribution through a rolled plate [41]. This particular example suggests a possible offshoot related to the efforts described in this paper. It would be highly advantageous to have the ability to predict grain size distribution, and hence property distribution in a forged wheel.

2.4 Finite Element Approach

Finite element analysis, especially in conjunction with computer aided design (CAD), has become a widely used tool for modeling engineered products. While the mathematical approach may be traced to the 1940's and 50's [42][43][44], it did not become a viable analysis tool until the advent of the digital computer in the 1960's and 70's. Even at this point, large-expensive mainframe computers were the only option. Most recently, with the emergence of much more powerful microcomputers, more and more practicing engineers are beginning to use commercial finite element codes to aid in the design and development of a huge variety of applications.

One of the most promising areas of application results from the ability to model manufacturing processes such as casting, forging, extrusion, and forming of sheet metals. In each case, it is imperative that the engineer developing the model understand the necessary requirements and limitations of the method. One of the most important aspects of the model is the manner in which the material under study behaves. Commercial codes are continually evolving with an ever-increasing number of material models available. In fact, this development in and of itself confirms the need for more accurate and precise material models for proper prediction of a particular processes outcome.

The background information presented below only touches on the aspects of the finite element method that directly impact the analyses performed. The desciptions are general and fairly basic. ABAQUS user manuals should be consulted if specifics are needed. In the present case, material rheology during plastic deformation is of primary concern.

Implementation of Plasticity

The following section lays out the general approach to metal plasticity as utilized by many finite element codes. These codes require three definitions be supplied for implementation of classic metal plasticity:

- Yield Surface Definition
- Flow Rule
- Evolution Law (i.e., hardening)

As discussed previously, the elastic portion of a material's response is separated from its plastic response, generally expressed using strain rate decomposition, by EQ 77:

$$.d\varepsilon = d\varepsilon^{el} + d\varepsilon^{pl} \tag{EQ 77}$$

Or using the integrated form given by EQ .

$$\varepsilon = \varepsilon^{el} + \varepsilon^{pl} \tag{EQ 78}$$

It is common, for metals, to use von Mises' yield criteria which results from the idea that elastic strains may be separated from plastic strains by noting that the elastic portion causes volumetric changes and the plastic portion arises from deviatoric strains. If the volumetric strain is defined by EQ 79.

$$\varepsilon_{vol} = trace(\varepsilon) \tag{EQ 79}$$

Then the deviatoric strain is given by

$$\boldsymbol{e} = \varepsilon - \frac{1}{3}\varepsilon_{vol}\boldsymbol{I} \tag{EQ 80}$$

where ***I*** is the identify matrix.

So now the volumetric and equivalent pressure stress can be written as EQ 81 and EQ 82, respectively.

$$p = -K\varepsilon_{vol} \tag{EQ 81}$$

$$p = -\frac{1}{3}trace(\sigma) \tag{EQ 82}$$

The bulk modulus, *K*, is found using Young's modulus, *E*, and Poisson's ratio, *v.*

$$K = \frac{E}{3(1-2v)} \qquad \text{(EQ 83)}$$

$$G = \frac{E}{2(1+v)} \qquad \text{(EQ 84)}$$

The deviatoric stress is given by:

$$\boldsymbol{S} = 2G\varepsilon^{el} \qquad \text{(EQ 85)}$$

$$\boldsymbol{S} = \sigma + p\boldsymbol{I} \qquad \text{(EQ 86)}$$

Finite element codes have the facility to accept elastic modulus values and Poisson's ratio to handle modeling prior to yielding. It should be kept in mind that these values are frequently temperature dependent and in many cases not available for the temperatures desired during modeling.

Next, flow of the material must be represented and is found from the following relation

$$d\boldsymbol{e}^{pl} = d\bar{\boldsymbol{e}}^{pl}\boldsymbol{n} \qquad \text{(EQ 87)}$$

where

$$\boldsymbol{n} = \frac{3\boldsymbol{S}}{2q} \qquad \text{(EQ 88)}$$

$$q = \sqrt{\frac{3}{2}\boldsymbol{S} \bullet \boldsymbol{S}} \qquad \text{(EQ 89)}$$

The equivalent plastic strain rate, de^{pl}, results from determination of the value, *q*, the equivalent Mises' stress through use of EQ 88 and EQ 89.

Assuming that the material behavior is rate dependent, an evolution or hardening function, *h*, must be known. Then the uniaxial flow rate equation, EQ 90, can be used to find the equivalent plastic strain rate.

$$\dot{\bar{e}}^{pl} = h(q, \bar{e}^{pl}, \theta) \qquad \text{(EQ 90)}$$

Some of the common hardening functions were given previously when discussing constitutive relations. At this point, material behavior has been completely described. Of course, what really occurs is basically the reverse solution of the above series of equations. Integrating the flow rule, EQ 87, results in:

$$\Delta \boldsymbol{e}^{pl} = \Delta \bar{e}^{pl} \boldsymbol{n} \tag{EQ 91}$$

now, combining EQ 91, EQ 85, and EQ 78 forms EQ 92.

$$\boldsymbol{S} = 2G(\boldsymbol{e}^{el}\big|_t + \Delta e - \Delta \bar{\boldsymbol{e}}^{pl} \boldsymbol{n}) \tag{EQ 92}$$

Further combining EQ 91 and EQ 87, results in:

$$\left(1 + \frac{3G}{q} \Delta \bar{e}^{pl}\right) \boldsymbol{S} = 2G(\boldsymbol{e}^{el}\big|_t + \Delta \boldsymbol{e}) \tag{EQ 93}$$

Simplifying notation using:

$$\hat{\boldsymbol{e}} = \boldsymbol{e}^{el}\big|_t + \Delta \boldsymbol{e} \tag{EQ 94}$$

and substituting

$$\left(1 + \frac{3G}{q} \Delta \bar{e}^{pl}\right) \boldsymbol{S} = 2G\hat{\boldsymbol{e}} \tag{EQ 95}$$

or in another form

$$q + 3G\Delta \bar{e}^{pl} = 3G\tilde{e} \tag{EQ 96}$$

where

$$\tilde{e} = \sqrt{\frac{2}{3} \hat{\boldsymbol{e}} \bullet \hat{\boldsymbol{e}}} \tag{EQ 97}$$

The Mises' equivalent stress, q, is

$$q = \bar{\sigma}(\bar{e}^{pl}) \tag{EQ 98}$$

which is found by inverting the integrated form of EQ 90. This stress is initially computed assuming elastic conditions and compared to the uniaxial yield stress, provided by the user. If the yield stress has been exceeded then

$$3G(\tilde{e} - \Delta \bar{e}^{pl}) - \bar{\sigma} = 0 \tag{EQ 99}$$

must be satisfied. Solving EQ 99 using Newton's method:

$$c^{pl} = \frac{3G(\tilde{e} - \Delta\bar{e}^{pl}) - \bar{\sigma}}{3G + H} \qquad \text{(EQ 100)}$$

where

$$H = \frac{d\bar{\sigma}}{d\bar{e}^{pl}} \qquad \text{(EQ 101)}$$

and

$$\Delta\bar{e}^{pl} = \Delta\bar{e}^{pl} + c^{pl} \qquad \text{(EQ 102)}$$

Iteration on the above is continued until convergence is achieved. Once the change in equivalent plastic strain is found, the solution is defined by

$$q = \bar{\sigma} \qquad \text{(EQ 103)}$$

and

$$\boldsymbol{S} = \frac{2G}{\left(1 + \frac{3G}{q}\Delta\bar{e}^{pl}\right)}\hat{\boldsymbol{e}} \qquad \text{(EQ 104)}$$

with the direction vector determined using EQ 88 and the change in plastic strain from EQ 105.

$$\Delta\boldsymbol{e}^{pl} = \Delta\bar{e}^{pl}\boldsymbol{n} \qquad \text{(EQ 105)}$$

So, in summary, the material is assumed to behave linearly up yield. When the Mises stress criteria is exceeded the hardening rule takes over. For metals, many times the assumption of linear behavior prior to yield is reasonable, especially in a forging operation where the plastic deformation dominates. It is the hardening behavior that tends to depart from easily described relations.

Hardening Law Models

Many codes allow for several methods of incorporating hardening. As presented above, linear hardening and Ramberg-Osgood hardening is provided, though in slightly modified form as given by EQ 106.

$$\dot{\varepsilon}_{plastic} = D\left(\frac{\sigma_{flow}}{\sigma_{\text{static yield}}} - 1\right)^{p} \text{ for } (\sigma_{flow} \geq \sigma_{\text{static yield}}) \qquad \text{(EQ 106)}$$

The user provides the reference stress, *D*, the static yield stress, and the strain hardening exponent. The values can be entered as dependent on strain and temperature. As mentioned earlier, these constants are determined by curve fitting experimental data which may not adequately reflect real material behavior if the fitting method proves deficient.

In addition, tabular data of yield stress, strain, strain rate, and temperature could also be input and used in the model. The data is regularized and some care must be exercised in how the it is provided [45]. Subroutines provide for a much more flexible and powerful method within ABAQUS. The neural network model developed subsequently is implemented through the VUMAT subroutine available within ABAQUS Explicit. The VUMAT developed and its inner workings are described fully in a subsequent chapter.

Explicit Dynamics

ABAQUS offers two integration methods for solving the equations of motion; implicit (within ABAQUS standard) and explicit. Explicit is used for all of the models constructed. Either method would prove appropriate for the simple compression models, but the forged wheel model places several demands on the code that suggest an explicit scheme would be advantageous. The wheel billet undergoes excessive plastic deformation, which as will be seen, requires almost constant adjustment of the mesh to prevent severe element distortion. Explicit has adaptive meshing capabilities that automatically enable mesh adjustment throughout the analysis. ABAQUS Standard does not have this ability. Unless exceptionally small time increments are chosen, convergence issues frequently result with implicit methods when "sharp" objects are brought into contact with a deformable body of interest. While an implicit operator is unconditionally stable, very small time increments result in very large numbers of iterations. Explicit operators are conditionally stable and do not require iterations. The kinematic state is advanced from one increment to the next once the stable time increment is determined, given by EQ 107.

$$\Delta t \leq \frac{2}{\omega_{max}}\left(\sqrt{1+\xi_{max}^2}-\xi_{max}\right) \qquad \text{(EQ 107)}$$

The highest frequency of the system, ω_{max}, and a fraction of the critical damping at this frequency, ξ_{max}, are determined based on the model's number of elements, element type, material, and other factors.

Explicit does not need to solve large numbers of simultaneous equations as a result. So, assuming large numbers of increments are required, explicit becomes much more efficient [46]. Disk space and memory usage are also much smaller with ABAQUS Explicit [47]. Further, ABAQUS Explicit has a more robust contact functionality allowing for solution of more complex contact situations [48]. Additionally, ABAQUS Explicit allows for parallel execution so that multiple processors can be used to speed up analysis. Finally, mass scaling, discussed below, can be employed within Explicit further enhancing speed of simulation [49].

Adaptive Meshing

While not required for the simple compression models that follow, adaptive meshing is a powerful tool available as part of ABAQUS/Explicit. The wheel billet is to be reduced from a height of approximately 457 mm (18 inches) to 38 mm (1.5 inches) resulting in severe plastic deformation. Any initial mesh will distort to a degree preventing completion of a solution. The code allows the mesh to be modified automatically as the analysis proceeds. The method is termed arbitrary Lagrangian-Eulerian analysis (ALE). With conventional meshing, material does not enter or leave the mesh (Lagrangian),but Eulerian analysis allows material to flow through the mesh [50]. Adaptive meshing involves two steps; create a new mesh with improved geometry and then remap the solution from the old mesh to the new mesh using a process call advection. The specified frequency of remeshing can be controlled along with the number sweeps attempted for improvement before advancing to the next increment. The principal disadvantage in employing adaptive meshing is an increased computational cost of 3 to 5 times that necessary for pure Lagrangian analysis [51]. It also limits the element choice to 1st order, reduced-integration solid elements (e.g., 4-node quads).

Mass Scaling

For models requiring considerable computing time, mass scaling is frequently employed. The stable time increment can be expressed in the form give by EQ 108 [52].

$$\Delta t \le min\left(L_e\sqrt{\frac{\rho}{\lambda + 2\mu}}\right) \tag{EQ 108}$$

The characteristic element length, L_e, could be increased, to the detriment of the mesh quality or the density, ρ, could be artificially raised. The other variables, λ and μ, are Lame’s constant which are fixed for the material modeled. The density can affectively be increased by increasing the mass. The finite element code includes this capability, but as mass is increased artificially, the model response must be monitored to ensure the ratio of kinetic energy to internal energy does not exceed 10 percent. It should also be noted, that the analysis time could be artficially decreased by increasing the velocities or accelerations of forces or displacements applied. This approach only works in non-strain rate dependent situations and hence is not applicable for the models developed herein. The specifics of mass scaling, when applied, is addressed where appropriate.

Contact

Forging operations require contact between dies and the workpiece. Proper modeling of the interface interactions must also be handled with care. Several general aspects must be addressed:

- Over/Under Closure
- Contact Algorithm
- Friction
- Heat Transfer

Contact can be exceptionally complicated from a modeling perspective and is outside the scope of this discussion. The default methods offered by the code are used unless noted otherwise. Only a brief overview is provided here.

Over/Under closure is a principal difficulty when initially bringing bodies into contact within the code. A master/slave approach is used to account for nodal interactions. Using this approach, slave nodes are not allowed to penetrate the master surface while the reverse is not true. Typically, for forging processes, the dies are designated as master surfaces with the billet as slave. This method of formulation produces local gaps and penetrations that technically violate compatibility. Acceptable accuracy can be achieved as long as the slave surface is sufficiently refined [53].

The contact algorithm itself determines how the bodies interact with each other. Bodies may be defined as deformable, rigid, or analytically rigid. While the body to be shaped must be deformable, dies for instance could be modeled as analytically rigid or as a rigid body to significantly decrease computational effort. When two bodies come into contact, additional constraints come into play; normal and tangential. By default, a Lagrange multiplier method is employed which essentially augments the stiffness matrix for the contact points implementing "classical" hard contact [54]. Using this approach, pressure is transmitted normal to the master surface when contact with the slave is established.

In the tangential direction, friction is potentially developed. The algorithm is based on isotropic Coulomb friction (termed penalty method) [55]. From the user’s standpoint, entering an estimated coefficient friction is required.

2.5 Neural Networks in Conjunction with Finite Element Methods

At this point in time, very little usage of ANNs in conjunction with finite element modeling has been reported. Most of the work has involved using finite element models for creation of data for subsequent processing by an ANN. Proper training of a ANNs require sufficiently large training sets which vary depending on the phenomena to be modeled. As noted previously, several researchers have used conventional FEA to produce the necessary training sets for further application of neural network models, but as yet the full incorporation of a ANN based material model within, or linked with, finite element analysis was not noted.

3. Investigative Approach

3.1 Description

The first sections that follow detail the conventional method of determining constants through curve fitting. The data used for fitting was extracted using digitizing software operating on graphs or from tables of values from published literature previously sited. Two examples are provided: 6061 aluminum alloy and nickel aluminide IC-396. For the aluminum, the values for reference stress and strain hardening exponent for use with the power law were found. Due to the more complicated behavior of the superalloy, both power law fits and constants for the kinetic rate equation were determined. This exercise was mostly carried out to demonstrate the difficulties in properly curve fitting alloys that exhibit unusual plastic flow behavior.

Next, several ANNs, for both materials, were set-up and trained varying the number of neurons, layers, weights, biases, and training algorithm. The number and condition of inputs were also varied. After training each network architecture was queried using the same strain rates and temperatures, but differing strains to establish a measure of performance. In addition, intermediate values of strain rate and temperatures were also presented to the networks and performance accessed.

At this point, two parallel finite element models were developed. Both models use the same conditions (e.g., load rate, temperature, material conductivity, density, etc.) within the finite element code with the exception that one model uses a material model (i.e., conventional) supplied with the commercial code, while the other uses the ANN material model. The initial models are simple compression tests of billets of the aluminum and nickel aluminide. This is done to ensure both codes are performing within reasonable limits and so they can be verified against experimental data.

Subsequently, modeling of the hot forging of the aftermarket wheel is carried out in the same manner using both methods. These two approaches are then compared to experimental measurements carried out as the actual manufacturing process is performed under normal working conditions. The actual load versus displacement curves are compared to those generated by finite element analysis. Metallurgical examinations, including microhardness, of the forged wheel were conducted in an effort to characterize microstructural differences associated with the varying stress-strain-temperature experiences of the material. In addition, comparisons of die fill and general part dimensions are made between the model and forging.

3.2 Model Materials

Two strategic materials, 6061 aluminum and a nickel aluminide alloy, were used in this investigation as model materials. The 6061 aluminum is a high strength light-weight alloy that is used extensively in the aerospace, automotive and other industries when structural weight savings are crucial to the application. The second material is an intermetallic alloy with outstanding high temperature strength, fatigue resistance, and corrosion resistance. Typical applications include engine components, high temperature forming tools, and turbo-machines.

6061 Aluminum

This particular alloy results from additions of magnesium and silicon. It is precipitation hardenable due to the metastable phase of the intermetallic compound Mg_2Si [56]. Its applications include hydraulic pistons, valves, valve parts, automotive wheels, hardware, electrical fittings, bicycles, and many others. It possesses relatively high strength with good workability and weldability [57]. Due to the above characteristics, 6061 is one of the most widely used alloys of aluminum. Table 1 shows selected properties for 6061 aluminum with an O temper.

TABLE 1. Selected properties of 6061-O aluminum [58].

Property	Typical Value
Density	2.70 Mg/m^3 @ 20ºC
Coefficient of thermal expansion	23.6 μm/m·K
Specific Heat	896 J/kg·K @ 20ºC
Thermal Conductivity	180 W/m·K @ 20ºC
Solution Temperature	529ºC
Yield Strength	18 MPa
Tensile Strength	55 MPa
Elongation	30%
Shear Strength	83 MPa
Poisson’s Ratio	0.3
Elastic Modulus, Tension	68.3 GPa
Elastic Modulus, Compression	69.7 GPa

Nickel Aluminide Properties and Processing

Nickel aluminide is an ordered intermetallic with a cubic $L1_2$ structure. Initially, they were found to be susceptible to embrittlement, but alloying was found to eliminate this problem. Ductilities of greater than 40% have been achieved, at room temperature, when microalloyed with boron [59]. Interest in Ni_3Al has mostly resulted from its excellent strength and oxidation resistance at high temperatures [60]. In addition, it has the anomalous behavior that its yield strength increases with increasing temperature (to about 600 to 700 °C) [61]. Ni_3Al has also been shown to have better fatigue and fatigue crack growth resistance than the nickel based superalloys [62]. Due to the above mentioned properties, Ni_3Al shows promise for use in heating element wires, wear parts, high performance diesel engine applications (e.g., turbocharger rotors, values, valve seats, piston rings, cylinder liners, and cylinder heads), aircraft fasteners, etc. [63].

Nickel aluminides have been used in their as-cast condition, but many industrial applications require products be in a wrought condition. Hot working is generally required due to the high-flow stresses encountered at room temperature. For the current study, precision forging is the ultimate end and as such, certain characteristics are desirable as suggested by Stoloff and Sikka [64]:

- ductility (30% or greater) at forging temperatures
- reasonable response at strain rates approaching 10^{-1} s^{-1}
- flow stresses 1/5 to 1/10 room temperature flow stresses
- wide temperature range with high ductility
- lack of low melting point liquid formation at the processing temperature
- lack of environmental effect and good intermediate-temperature ductility to prevent cracking during cooling from processing temperature

Recently, alloys have been developed that have improved the ductility and workability in an effort at commercialization. One of these alloys, IC-396LZR, has been investigated by Oak Ridge National Laboratory to determine hot deformation mechanisms [65]. The composition of this alloy is shown in Table 2.

TABLE 2. Composition of IC-396 alloy tested.

Element	Weight Percent
Nickel	81.075
Aluminum	7.98

Element	Weight Percent
Molybdenum	3.02
Chromium	7.72
Zirconium	0.20
Boron	0.005

The raw material for hot forging could be in several forms. Traditionally, cast ingots are utilized. The castings are formed from the appropriate ratios various materials to yield an alloy suitable for hot working. Several cast nickel aluminide alloys have been commercialized. Another method would be to hot forge a powder metallurgy product, of which there are several. Typically, P/M products exhibit low ductility and are unsuitable for subsequent forging operations. A possibly promising method would be combustion forging [66][67]. The forging operation could be carried out using the heat generated during the forming of Ni_3Al, resulting in significant energy savings.

Table 3 shows the material properties expected from the IC-396LZR alloy of Ni_3Al used in the flow stress experiments from which the preliminary results and initial model are developed.

TABLE 3. Approximate IC-396LZR nickel aluminide properties.

Property	Value
Yield Stress (0.2% offset)	580 MPa @ 650 °C [68]
Elastic Modulus	179 GPa [60]
Poisson's Ratio	0.295[69]
Coefficient of Thermal Expansion	11.90×10^{-6} K^{-1} [69]
Density	7.50 Mg/m^3 [60]
Conductivity	unknown
Specific Heat	unknown
Inelastic Heat Fraction	unknown

4. Conventional Model Development

4.1 Development of Hardening Behavior Relations

The following sections detail the initial development of hardening relations (i.e., curve fits) for the aluminum and nickel aluminide based on published literature [1][2].The data relates flow stress dependence to strain, strain rate, and temperature. The two alloys, as previously noted, are 6061 aluminum and nickel aluminide 396LZR. The aluminum was chosen due to its extensive usage in the forging industry and as such an accurate model may find wide application. Nickel aluminide exhibits highly non-linear hardening behavior and for this reason is modeled primarily to show the advantages of a neural network approach.

4.2 6061 Aluminum Conventional Model

Figure 10 shows flow stress curves obtained from compression testing 6061 at two temperatures for several strain rates [1]. The curves are reasonably flat above 0.2 strain, but some strain softening is noted at the slowest strain rate. The curves were generated using digitizing software that produces numerical data from scanned graphics.

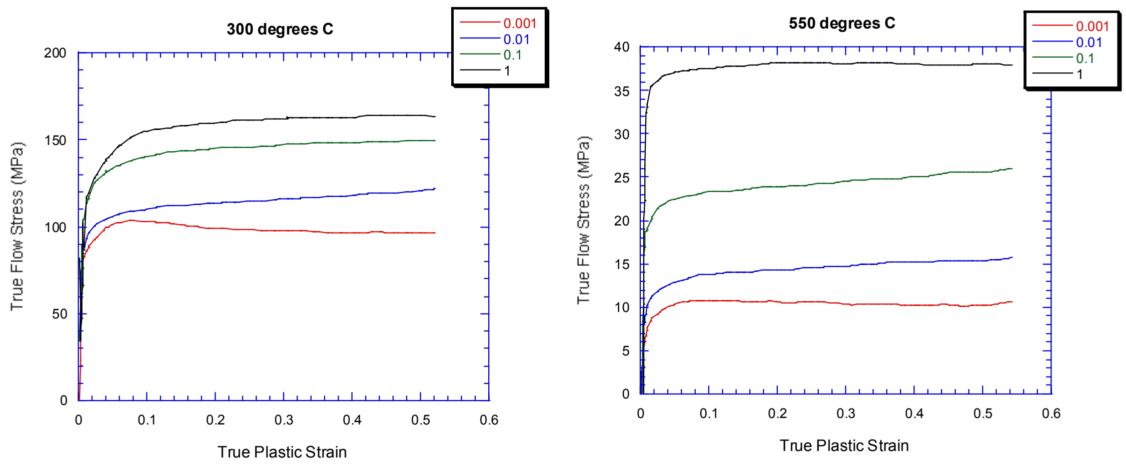


FIGURE 10. Flow stress curves for 6061 aluminum at two temperatures.

The graphs that follow in Figure 11 show power law curve fits generated based on tabular data at four strain rates (0.001, 0.01, 0.1, and 1) for a particular value of strain. While it would be preferential to have data for more values of strain rate, this requires more testing at constant strain rates for the values required. It is therefore typical to only test at the logarithmic values shown. Table 4 and Table 5 summarize the values of reference flow stresses and exponents determined for the power law fit. In theory, the values should be constant for a given temperature, and while they do not vary appreciably, some variation is noted. The greatest variance is noted for the 350ºC values is also supported by the lower R values noted during fitting. The kinetic rate equation could also be fit, but at this point, it was determined based on the above fits that the power law reasonably reflects the resulting flow stresses. As will be seen, this is not the case for more complicated material behavior.

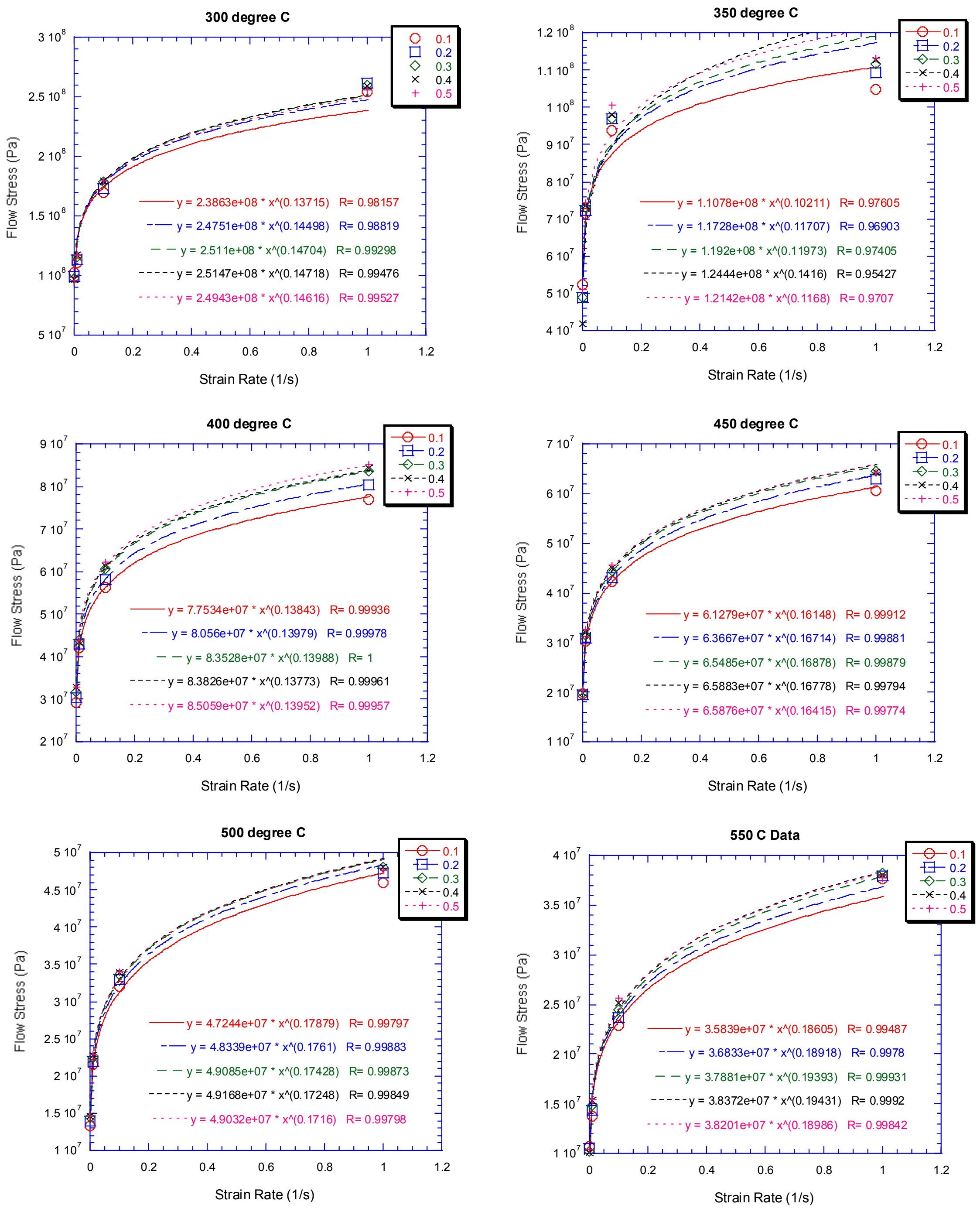


FIGURE 11. Plots of flow stress at several strains (see legend) showing curve fits generated.

TABLE 4. Values of reference flow stress.

Parameter	300ºC	350ºC	400ºC	450ºC	500ºC	550ºC
0.1	239	102	77.5	61.3	47.2	35.8
0.2	248	117	80.6	63.7	48.3	36.8
0.3	251	119	83.5	65.5	49.1	37.9
0.4	251	124	83.8	65.9	49.2	38.4
0.5	249	121	85.1	65.9	49.0	38.2

TABLE 5. Values of the exponent.

Parameter	300ºC	350ºC	400ºC	450ºC	500ºC	550ºC
0.1	0.137	0.102	0.138	0.161	0.179	0.186
0.2	0.145	0.117	0.140	0.167	0.176	0.189
0.3	0.147	0.120	0.140	0.169	0.174	0.194
0.4	0.147	0.141	0.138	0.168	0.172	0.194
0.5	0.146	0.117	0.140	0.164	0.172	0.190

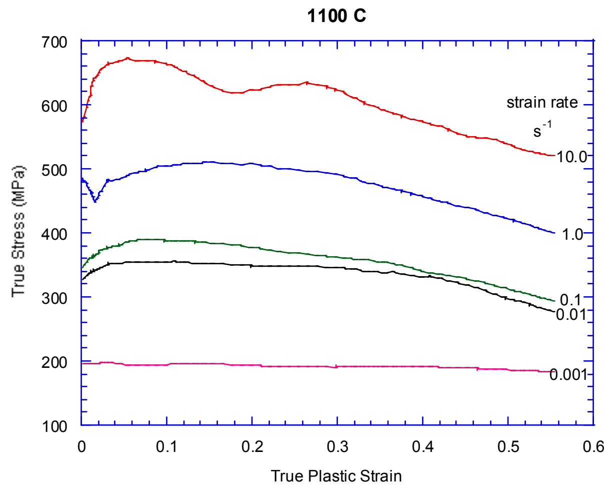
4.3 Nickel Aluminide Conventional Model

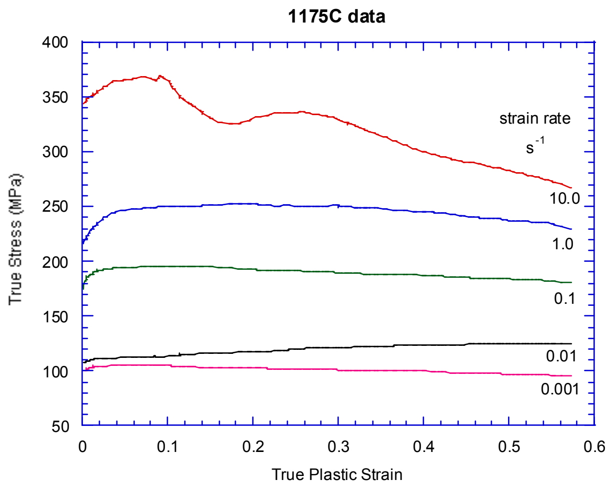
The flow stress versus plastic strain for the nickel aluminide is shown in Figure 12 [2]. As can be seen, nickel aluminide exhibits oscillatory behavior at high strain rates and strain softening at high levels of strain. Again, using digitizing software, data was extracted from Figure 12. In this way, the values for plastic strain and flow stress were found for strain rates of 10, 1, 0.1, 0.01, 0.001 s^{-1} and temperatures of 1100°C, 1175°C, and 1250°C. Using the data extracted, the coefficients for the power law, EQ 5, and the kinetic rate equation, EQ 9, were determined by curve fitting. Figure 13 shows log-log plots at various strain values for determination of the exponent, m, which is the slope of the line. It should be noted that only values of 0.3, 0.4, and 0.5 strain were used, as the fitting method requires use of steady state stresses. Values of strain rate were matched up with the chosen strain values and a flow stress found. At this time, this process has to be performed by hand and is not automatically found using software. The reference stress is also found from the plots. Both the exponent and the reference are seen to vary depending on temperature and strain. Also note that the curves are fit with straight lines which do not perfectly match the data. The activation energy is required for the kinetic rate equation and is estimated using an Arrhenius plot as shown in Figure 14. Next, alpha is determined from stress versus natural log strain rate plot as shown in Figure 15 using the relation

$\alpha = m\beta$ (m determined from low stresses). As before, the value for the constant varies considerably based on temperature. Lastly, Figure 16 is used to find n’ and A. Note the poor fit using a straight line for the data. Figure 17 shows comparisons of the constants.The fits demonstrate that the data is not totally linearized in a log-log plot. With this method only one curve at a single temperature is theoretically described. The exponent clearly varies with temperature, strain, and strain rate. To quote Prasad, et.al., "the kinetic rate equation is not obeyed in the entire temperature range of testing. In view of this limitation, it is necessary to apply the rate equation within narrow ranges of temperature [2]." It should also be pointed out that small changes in the exponent have a large effect on the computed flow stress.

Figure 18 and Figure 19 show power law and kinetic rate equation predictions of flow stress for strains of 0.3, 0.4, and 0.5. The power law fit appears to fit the higher strain rates fairly well, but results in much lower than actual flow stress for a strain rate of 0.01 and a somewhat higher value for a rate of 0.1. The kinetic rate equation fits, produce values much further from true.

The foregoing demonstrates the difficulties in extracting constants. Considerable time and effort is expended in generating each of the plots necessary and many cases neither the power law nor the kinetic rate equation properly reflect true material behavior. It is with this in mind, that an artificial neural network approach may prove advantageous.





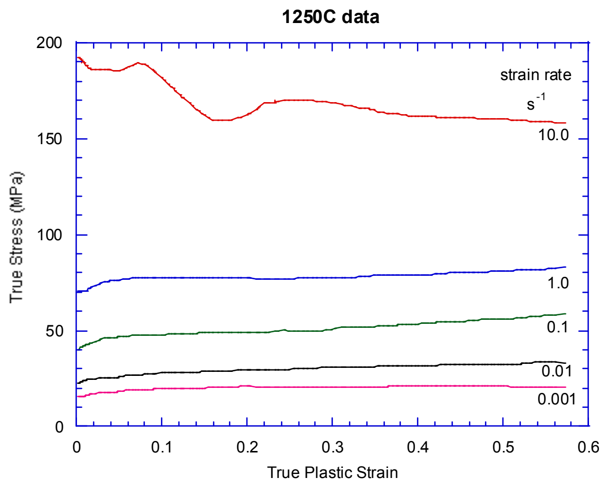


FIGURE 12. True flow stress versus true plastic strain for nickel aluminide IC-396.

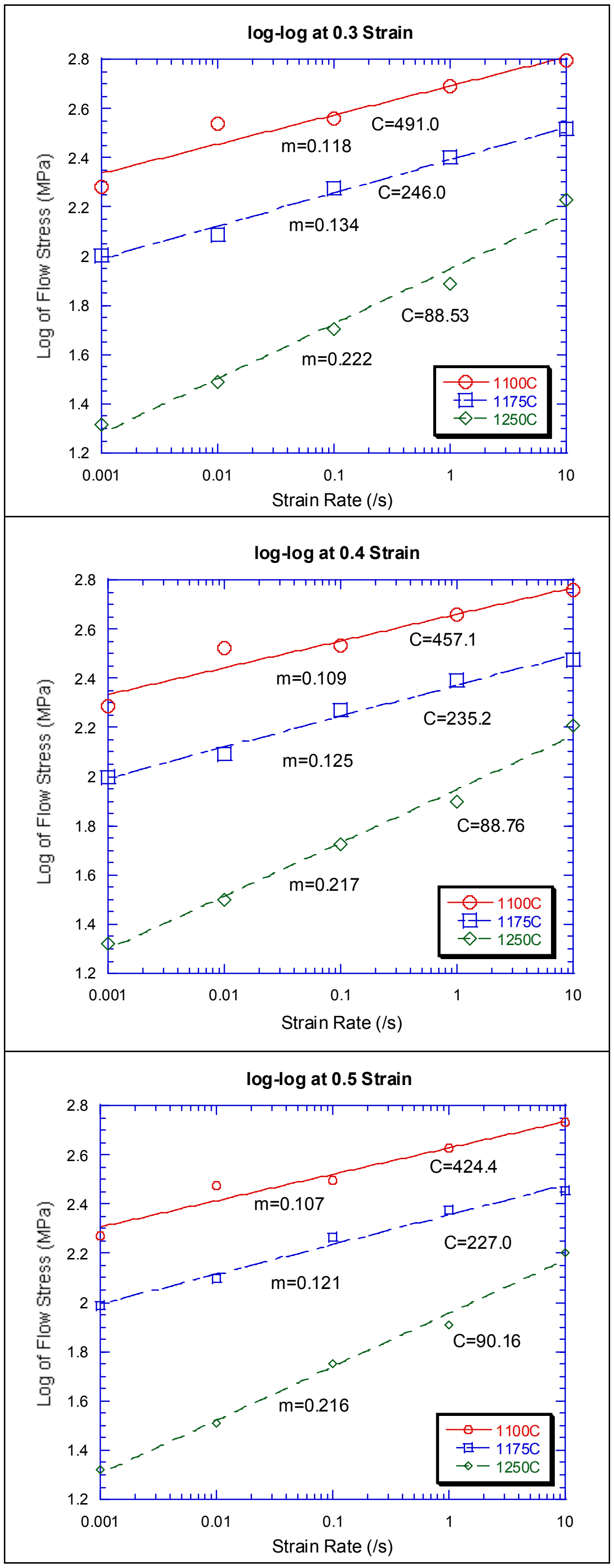
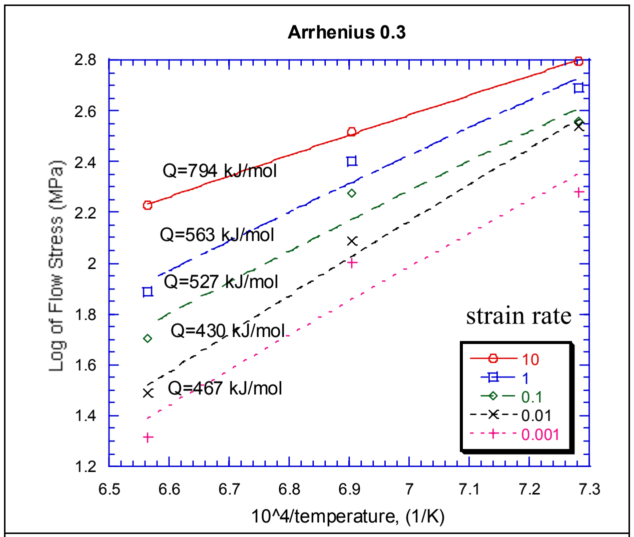
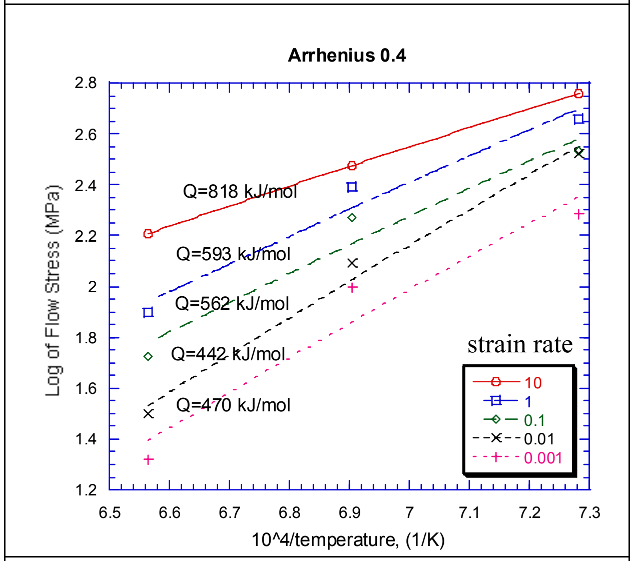


FIGURE 13. Extraction of strain exponents at various plastic strain values at three temperatures (see legend).





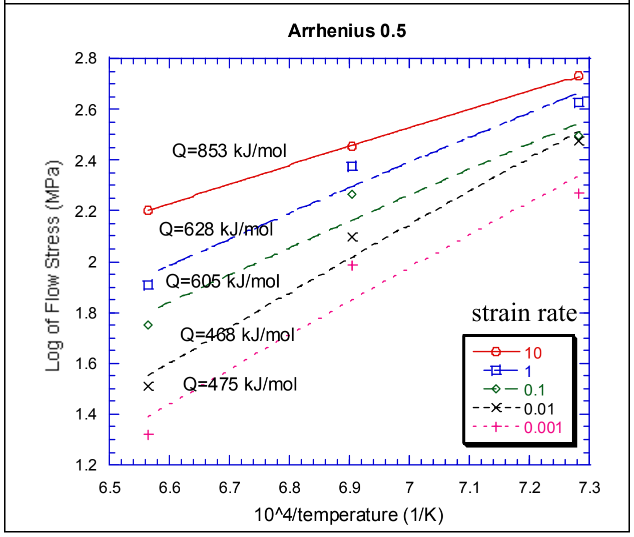


FIGURE 14. Arrhenius plots at various strains showing variation in activation energy .

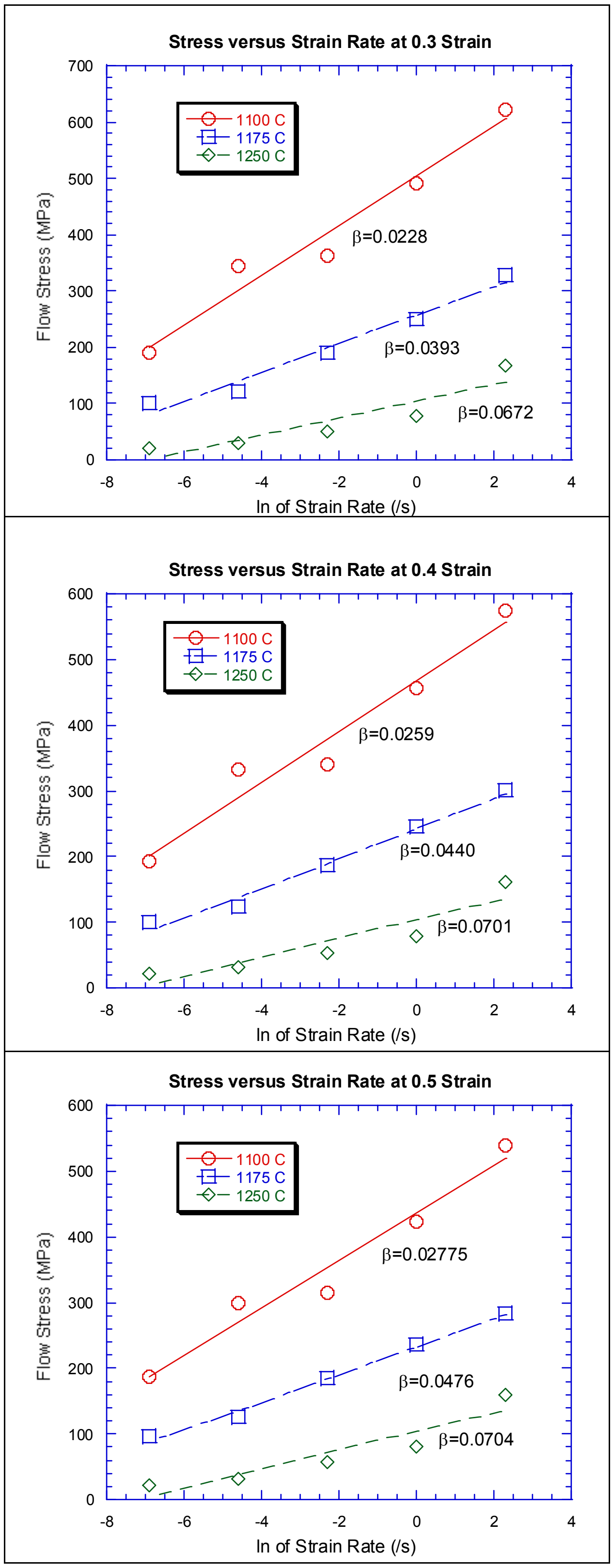


FIGURE 15. Beta values found from inverse of slopes.

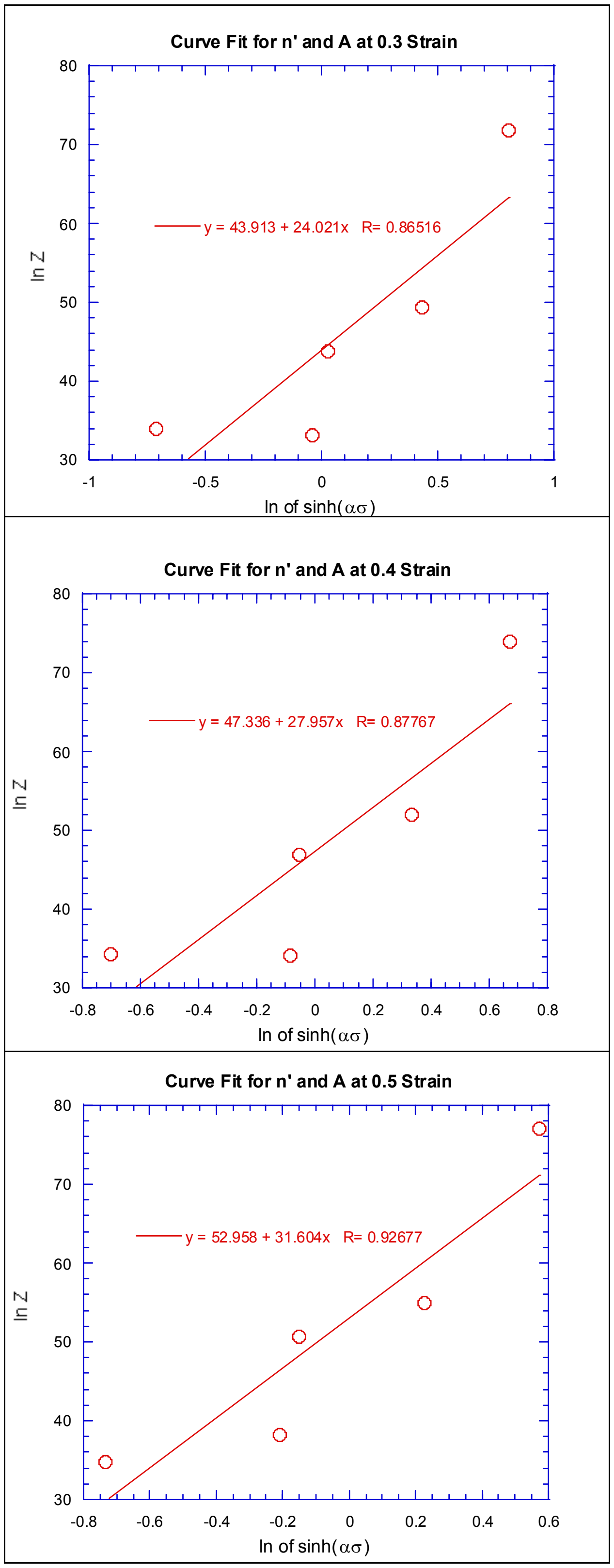


FIGURE 16. Slope and intercept for determining n’ and A (=$e^{\text{intercept}}$) for various strains.

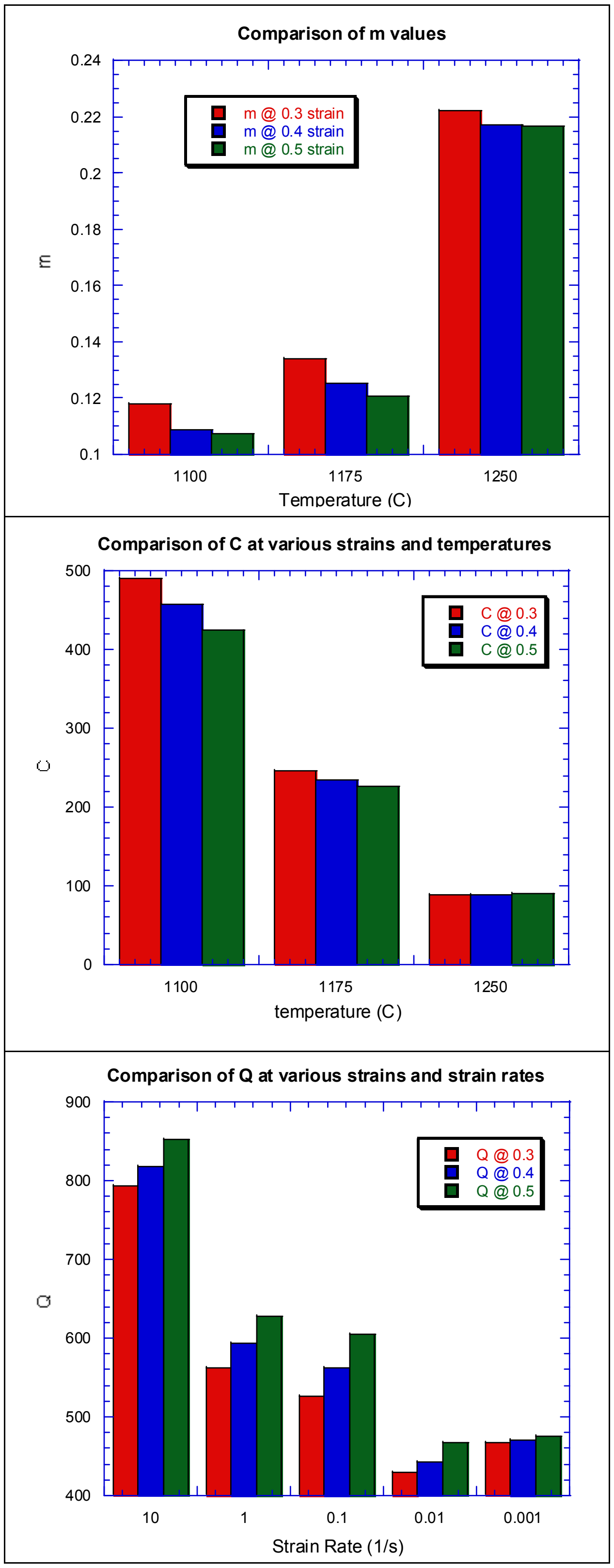


FIGURE 17. Comparisons of the constants found from curve fits.

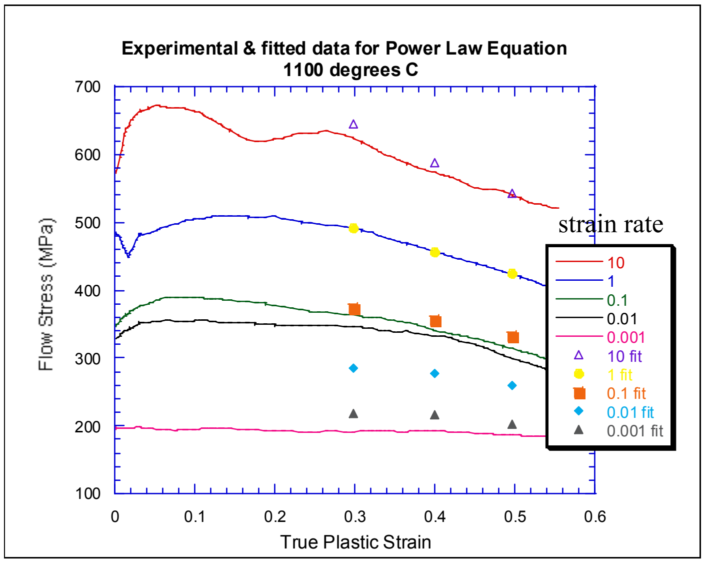


FIGURE 18. Comparison of fitted power law with actual experimental data.

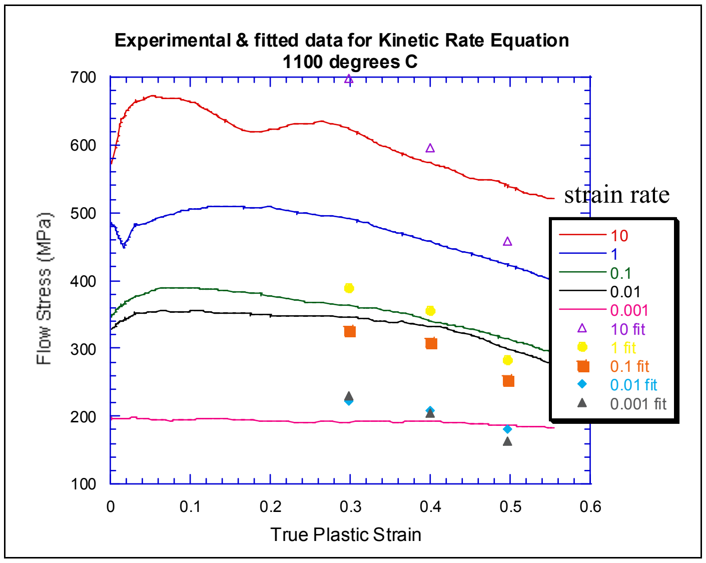


FIGURE 19. Comparison of fitted kinetic rate equation with actual experimental data.

5. Artificial Neural Network Model Development

5.1 6061 Aluminum Neural Network Development

As a starting point, the values extracted from the true stress versus true plastic strain curves, Figure 10, were used for training several single layer networks with varying numbers of neurons. The training set is comprised of 1008 total data points, split evenly for 300ºC and 550ºC, with equal numbers of points for each strain rate (i.e., 126 points for each separate curve, four curves per temperature). Table 6 below shows network architectures and the condition of the data used along with the "best" results obtained from five training attempts under each condition. The "best" results were generally based on achieving the lowest value for mean squared error (MSE), but several figures were generated for use in judging performance. It should be noted that a particular architecture may have produced widely varying results. The variance is largely dependent on the initial weights and biases which are randomly initialized by MATLAB.

Initial Training Attempts

Generally what follows are two figures generated (300 and 550ºC) for each set in Table 6 showing the target flow stress values used in training and the network output of predicted flow stress versus the strain for strain rates of 0.001, 0.01, 0.1, and 1 using the same input values. In addition, on these same figures, flow stress values are plotted that result from network output using the same constant strain rates, but using values of strain varying from 0.01 to 0.5 at intervals of 0.01. A properly trained network should produce three sets of curves, basically one laying on top of the other. In some cases, another figure was added preceding the plots of flow stresses. These figures show the training progression through each epoch with the resulting error. These figures are provided by MATLAB during training and can be used to stop the process when an acceptable error level has been achieved or when the error is no longer decreasing. The third figure reports the results of a linear regression analysis performed on the target versus the network's output; the R values are shown in the table.

TABLE 6. Network training parameters and results for "best" training attempt.

Data Condition	Training Algorithm	Number of Neurons	Epochs Completed	Mean Squared Error	R	Relevant Figures
unaltered	LM[a]	10	9[b]	1.07E15	0.803	none[c]
unaltered	LM	20	6[d]	9.85E14	0.819	none
unaltered	LM	50	9[e]	5.60E14	0.902	none
$\frac{\sigma}{1\times10^6}$	LM	10	508[a]	27.64	0.993	20, 21, 22, 23
$\frac{\sigma}{1\times10^6}$	LM	20	5000	98.78	0.983	24, 25, 26
$\frac{\sigma}{1\times10^6}$	LM	50	3558[a]	13.42	0.998	27, 28, 29, 30, 31
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$	LM	10	500	0.00278	0.998	32, 33, 34, 35, 36, 37
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$	LM	20	50	0.00164	0.999	38, 39, 40, 41, 42, 43
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$	LM	50	50	0.000511	1.000	44, 45, 46, 47, 48, 49

a. Levenberg-Marquardt.
b. maximum mu reached.
c. errors too large for consideration.
d. maximum mu reached.
e. maximum mu reached.

While an R value of 1 indicates near perfect matching of network output to the targets used in training, it should be kept in mind that this does not reflect a network's ability to produce acceptable outputs from inputs that stray from the training set. To demonstrate this phenomenon, a fourth and fifth figure was also produced in some cases, again showing the targets and the corresponding network output, but this time with intermediate values of strain rate (0.005, 0.05, 0.5, and 5 /s).

The networks trained to this point generally show improved performance as you move down Table 6 when considering only the network's ability to mirror the training set and input at the same strain rates. In fact, it is fairly straight forward and easy to produce a network that almost perfectly models the training data. Difficulties arise when attempting to utilize the network for predictions of strain rates or temperatures that were not part of the training set. As figures 30, 31, 36, 37, 42, 43, 48, and 49 show, when intermediate values of strain rate are used as input, the network's output of flow stress produces large swings that are not reflective of real material behavior.

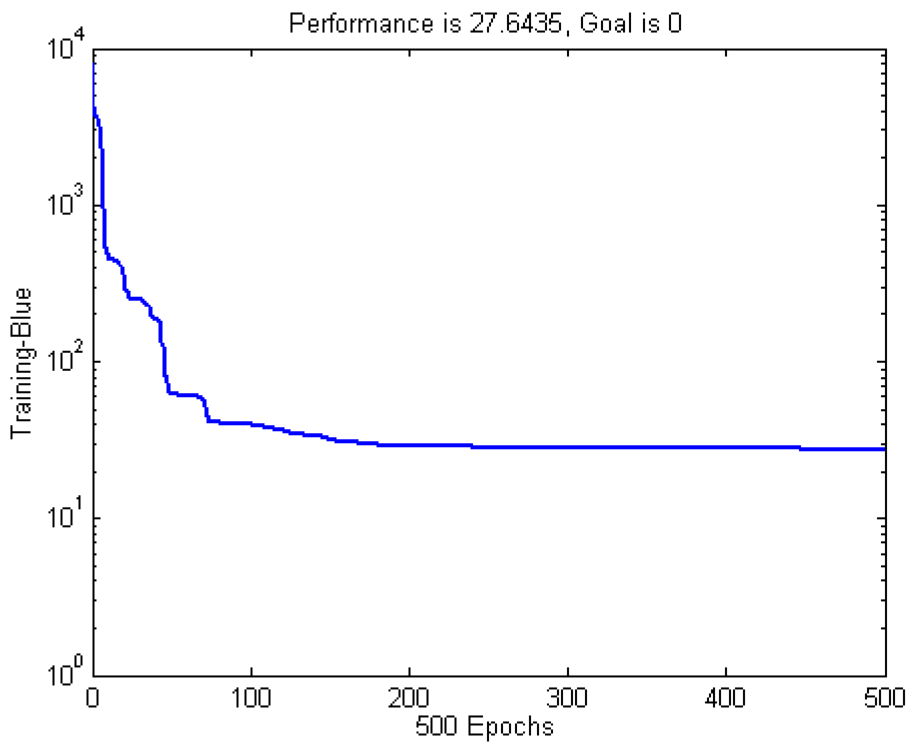


FIGURE 20. Progression of training using stress/1E6, 10 neurons, LM training algorithm.

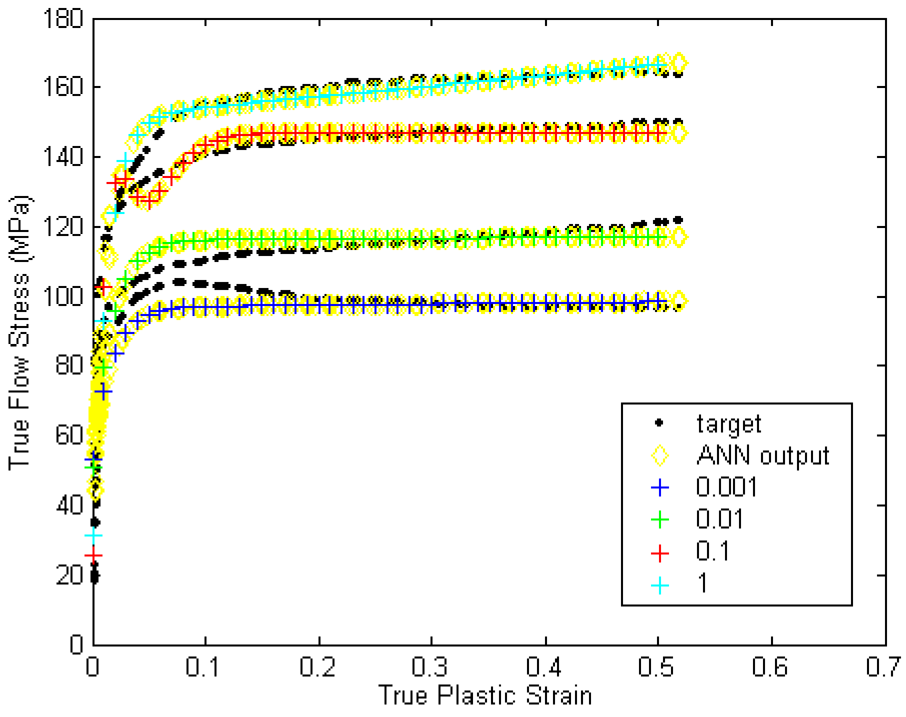


FIGURE 21. Network performance, 10 neurons, LM training, for 300ºC at various strain rates.

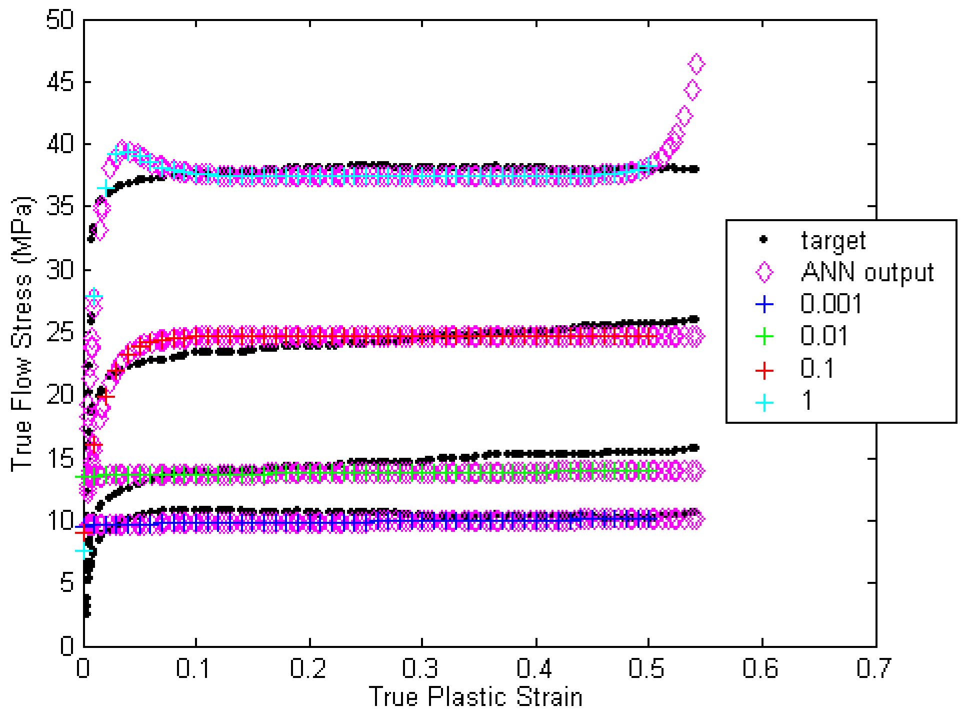


FIGURE 22. Network performance, 10 neurons, LM training, for 550ºC at various strain rates.

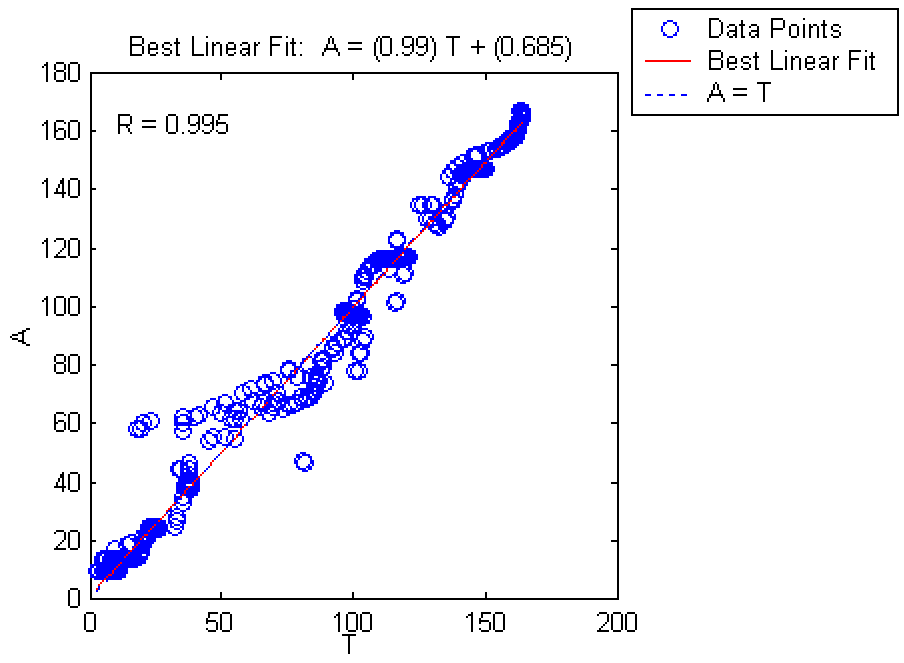


FIGURE 23. Linear regression for 10 neurons, LM trained network, network output (A) versus targets (T).

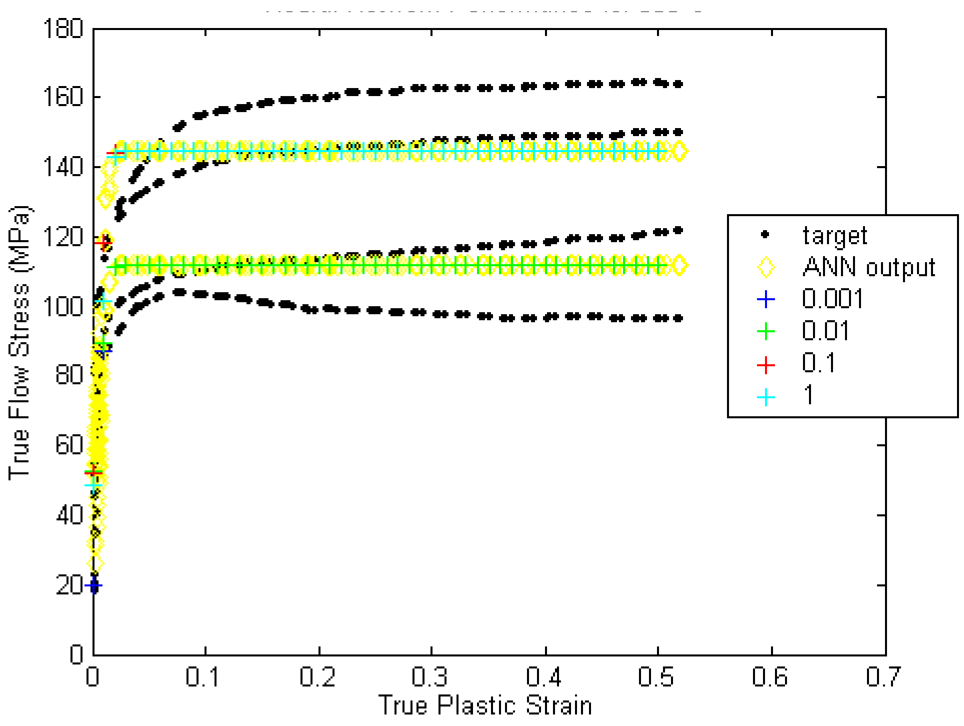


FIGURE 24. Network performance, 20 neurons, LM training, for 300ºC.

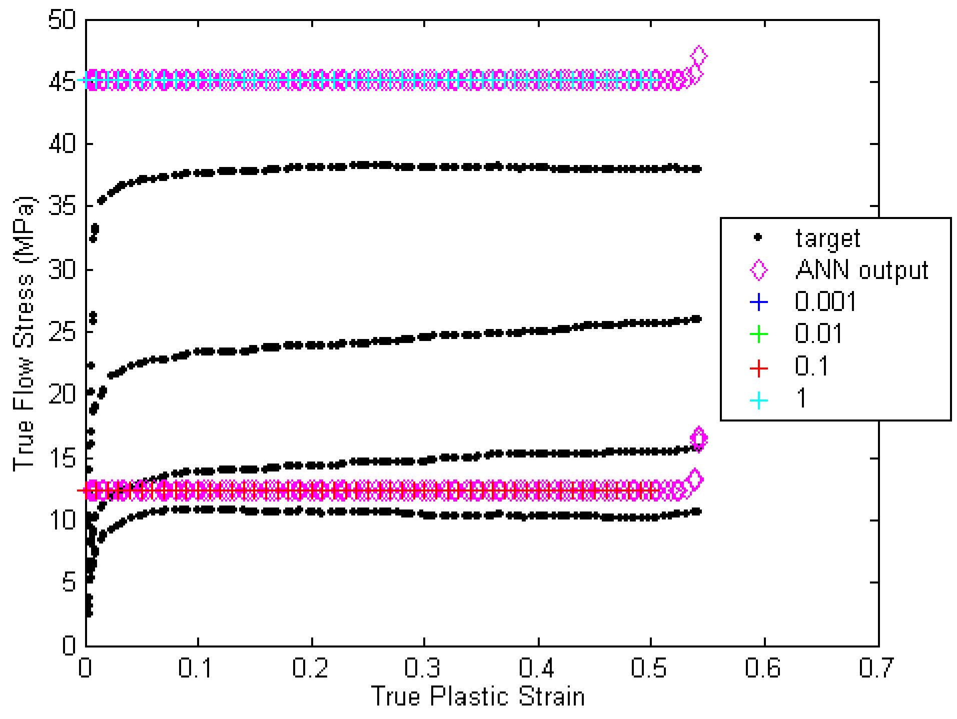


FIGURE 25. Network performance, 20 neurons, LM training, for 550ºC.

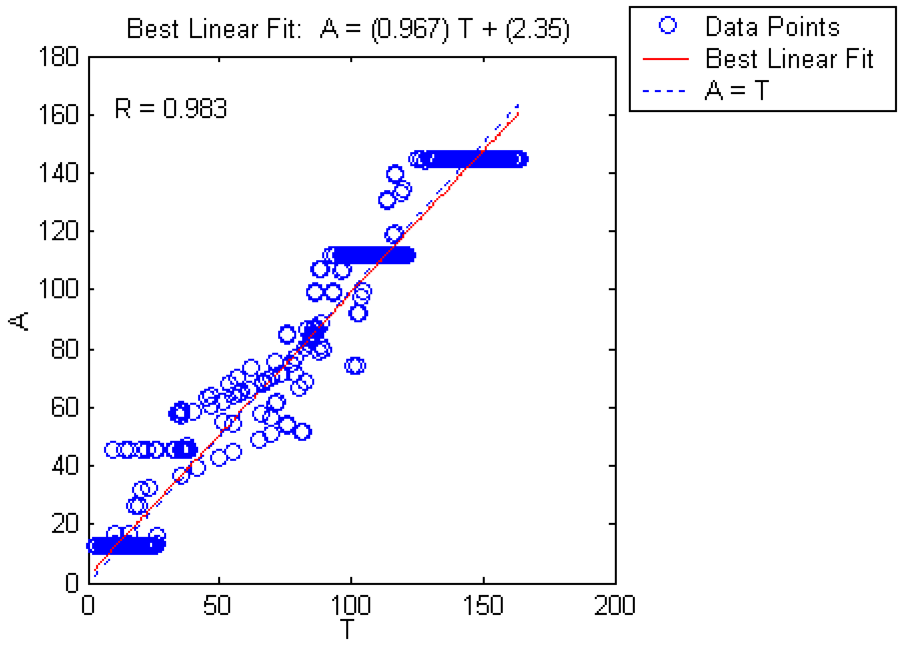


FIGURE 26. Linear regression for 20 neurons, LM trained network, network output (A) versus targets (T).

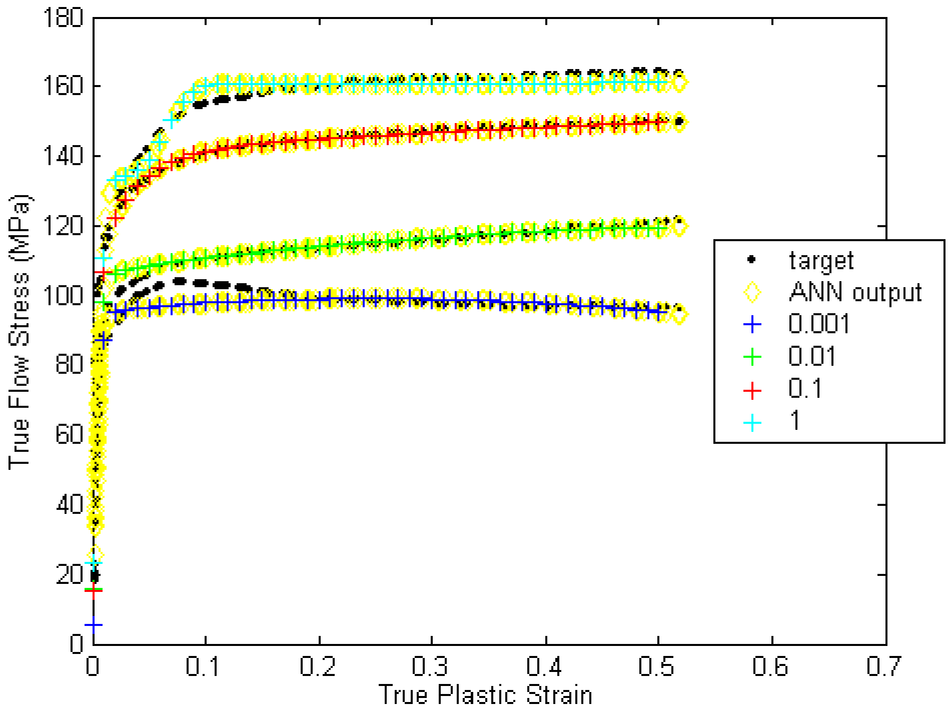


FIGURE 27. Network performance, 50 neurons, LM training, for 300ºC.

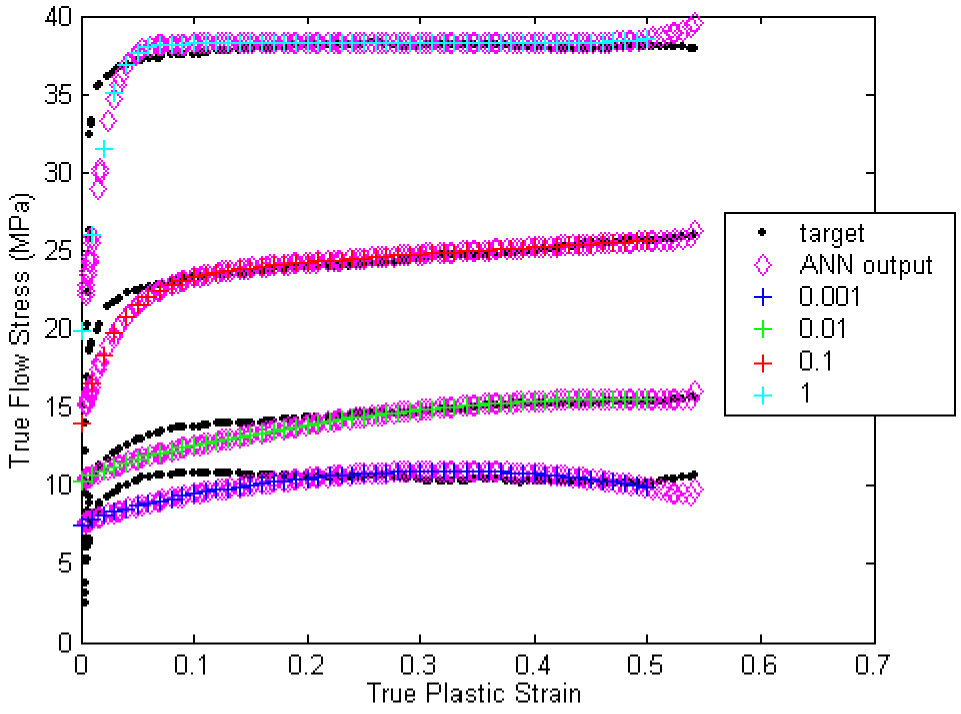


FIGURE 28. Network performance, 50 neurons, LM training, for 550ºC.

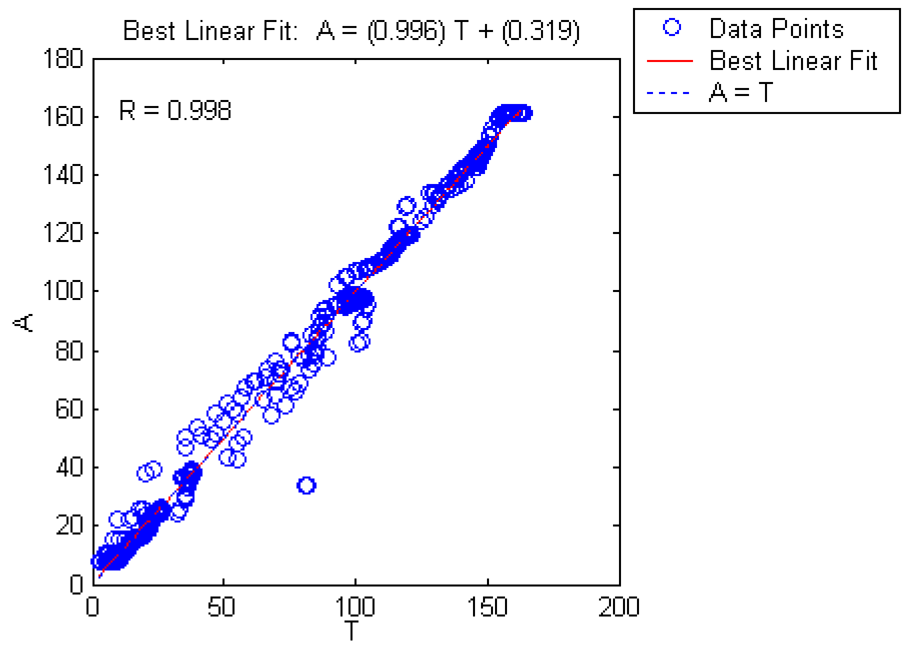


FIGURE 29. Linear regression for 50 neurons, LM trained network, network output (A) versus targets (T).

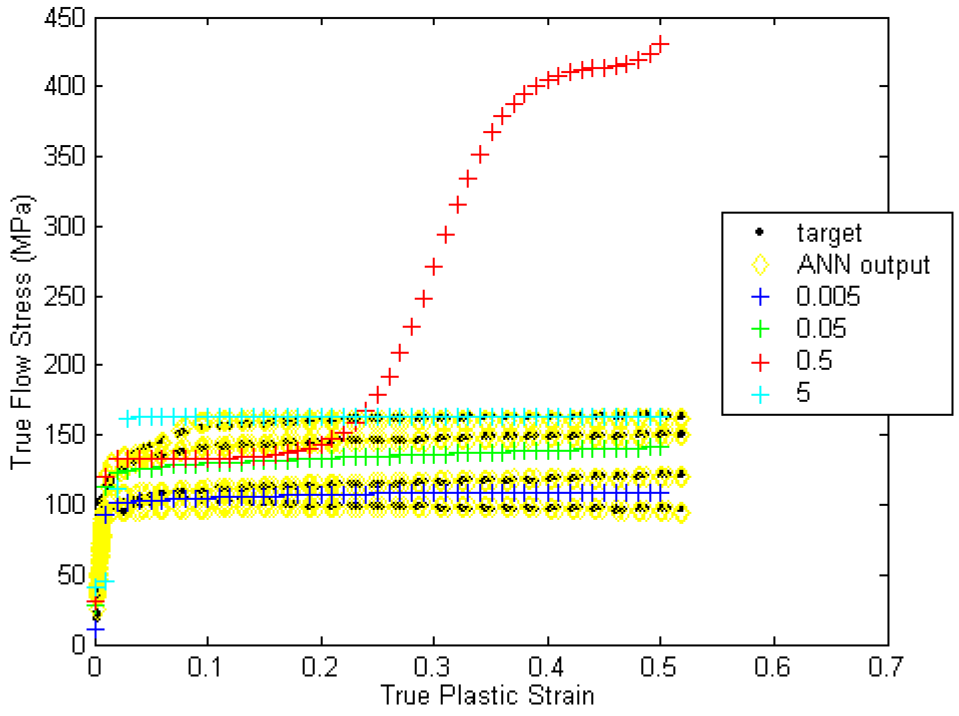


FIGURE 30. Network performance, 50 neurons, LM training, for 300ºC using intermediate strain rates.

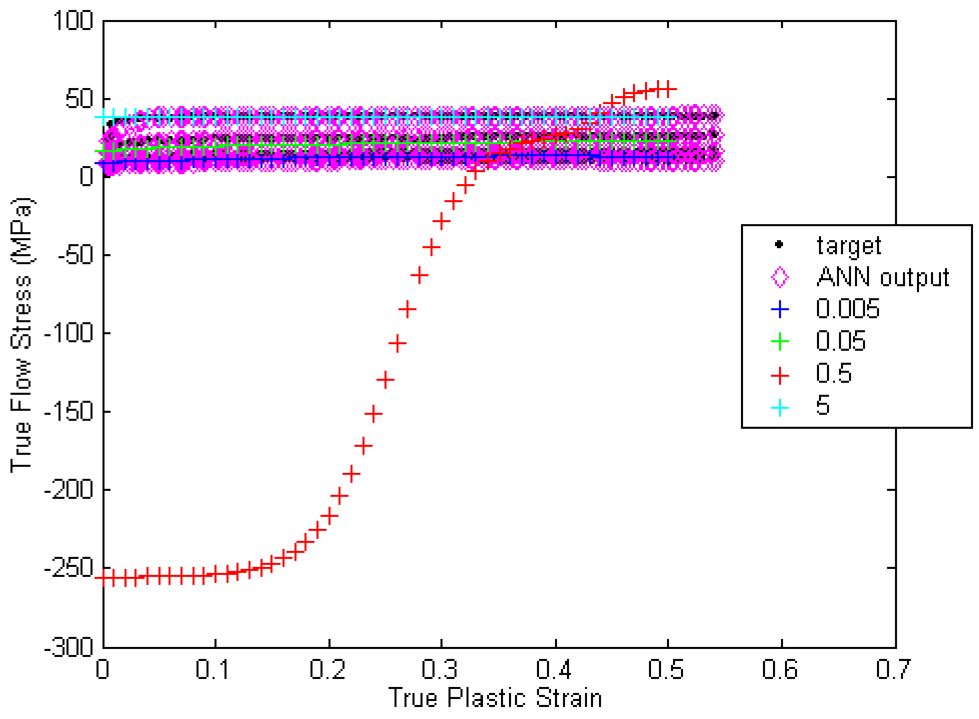


FIGURE 31. Network performance, 50 neuron s, LM training, for 550ºC using intermediate strain rates.

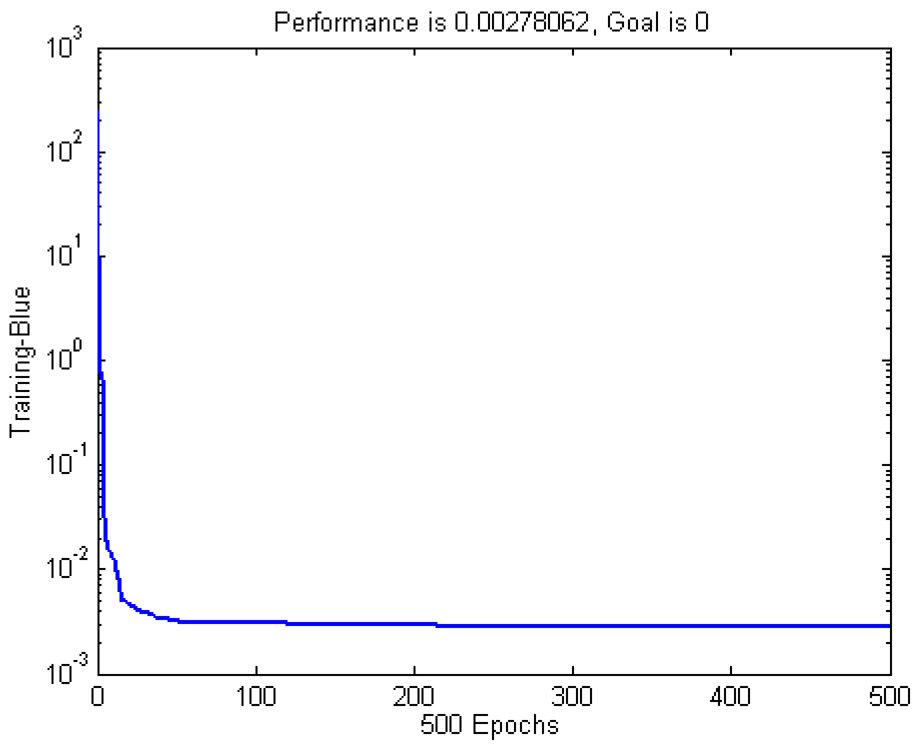


FIGURE 32. Progression of training using ln(strain), ln(strain rate), 1/temperature, ln(stress) with 10 neurons, LM training.

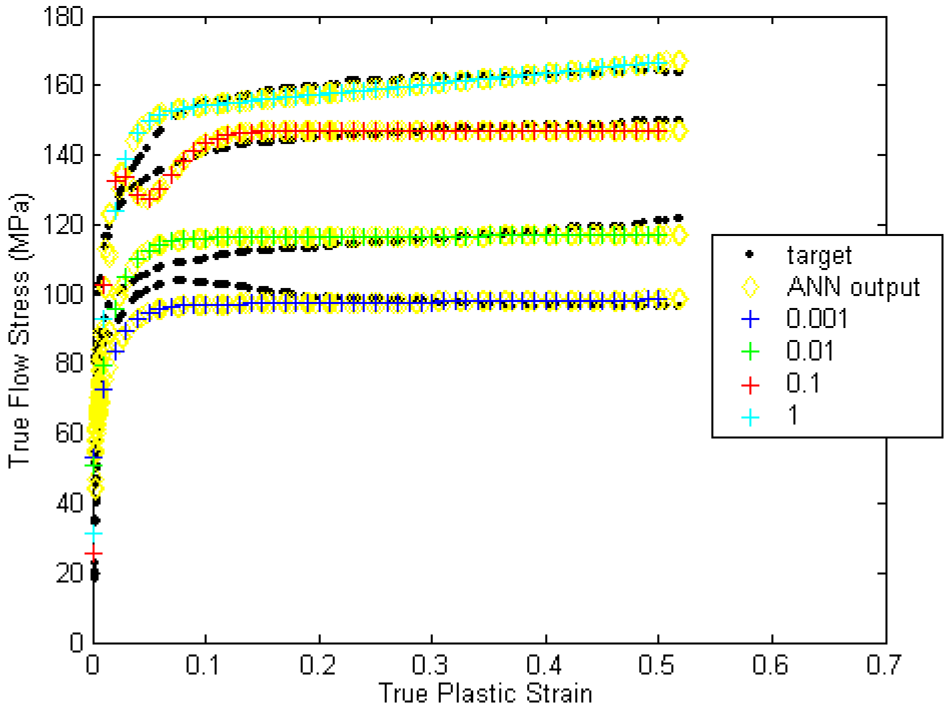


FIGURE 33. Network performance using ln(strain), ln(strain rate), 1/temperature, ln(stress) with 10 neurons, LM training, for 300ºC.

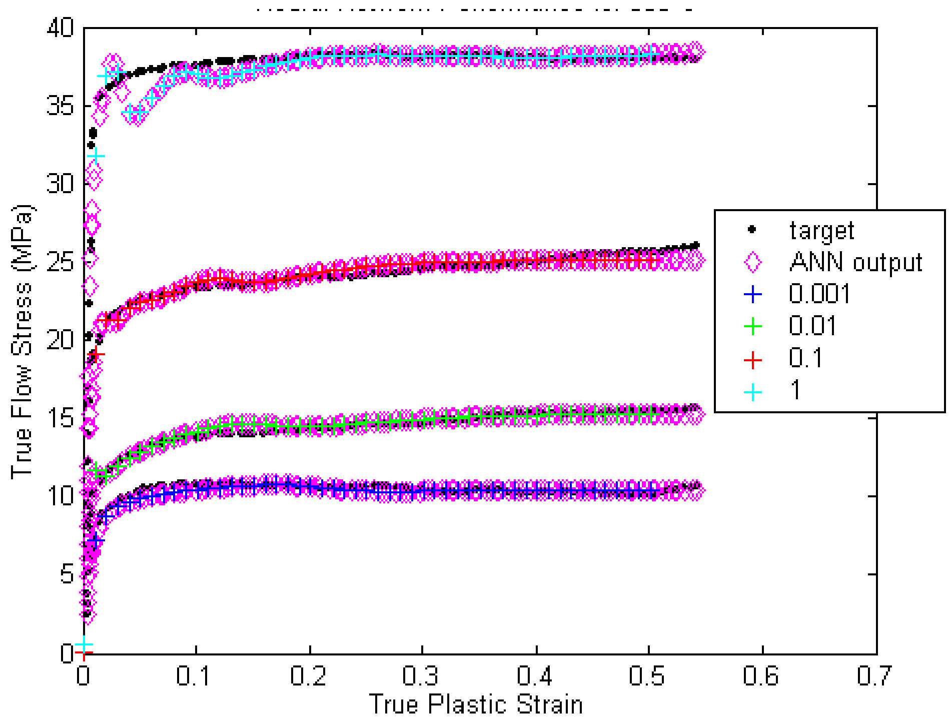


FIGURE 34. Performance with ln(strain), ln(strain rate), 1/temperature, ln(stress) with 10 neurons, LM training, for 550ºC.

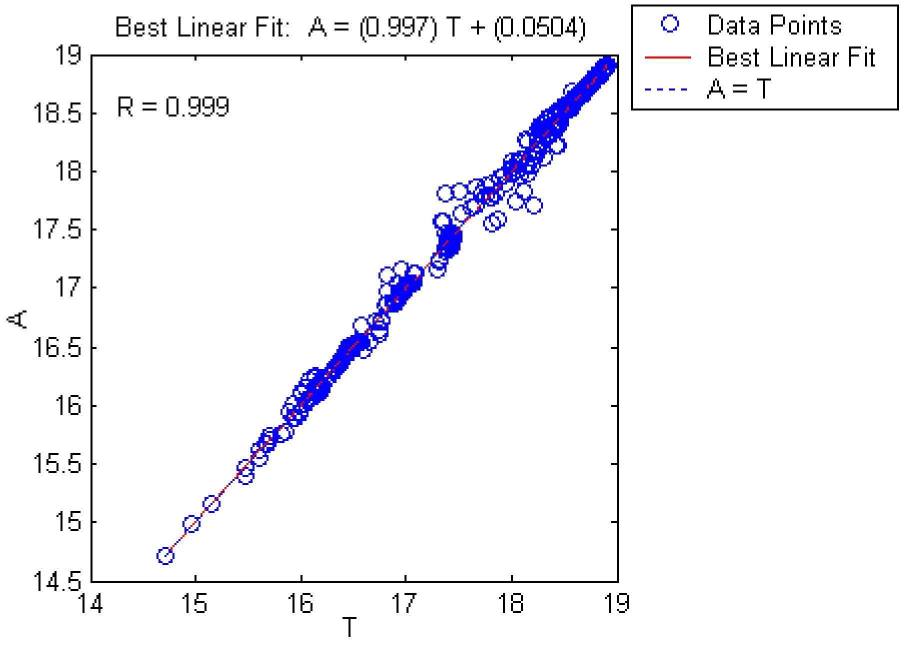


FIGURE 35. Linear regression using ln(strain), ln(strain rate), 1/temperature, ln(stress) for 10 neurons, LM trained network showing network output (A) versus targets (T).

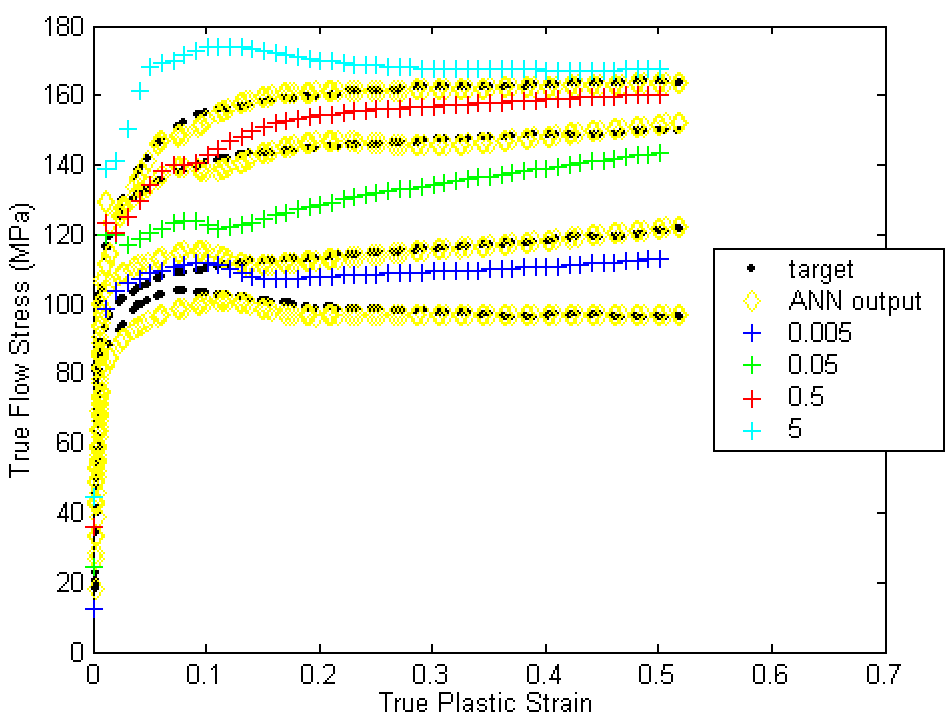


FIGURE 36. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 10 neurons, LM training, for 300ºC at intermediate strain rate values.

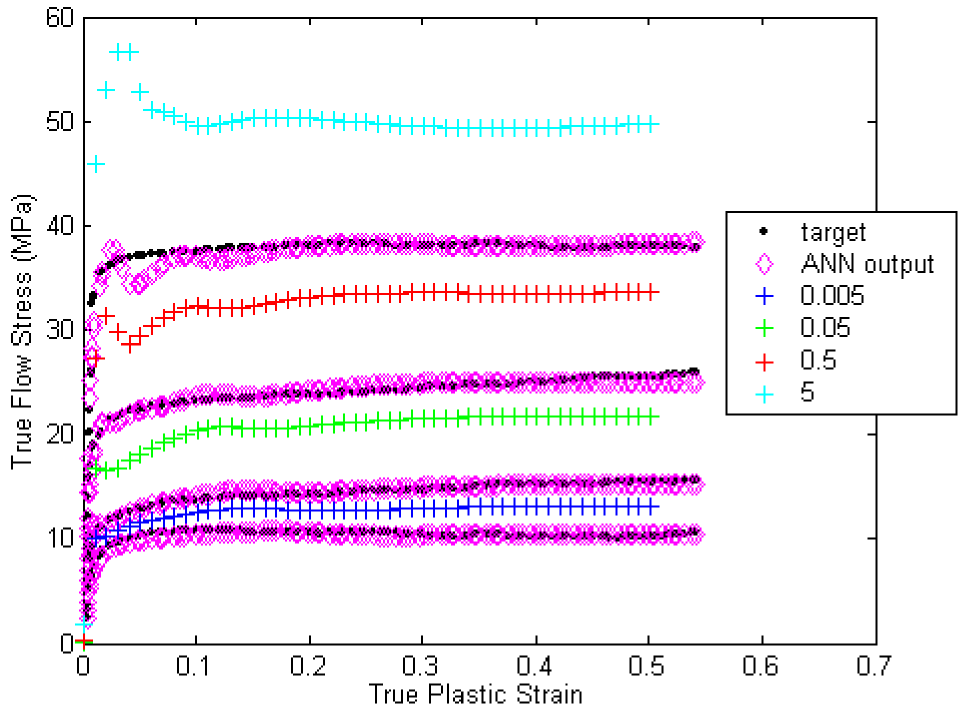


FIGURE 37. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 10 neurons, LM training, for 550ºC at intermediate strain rates.

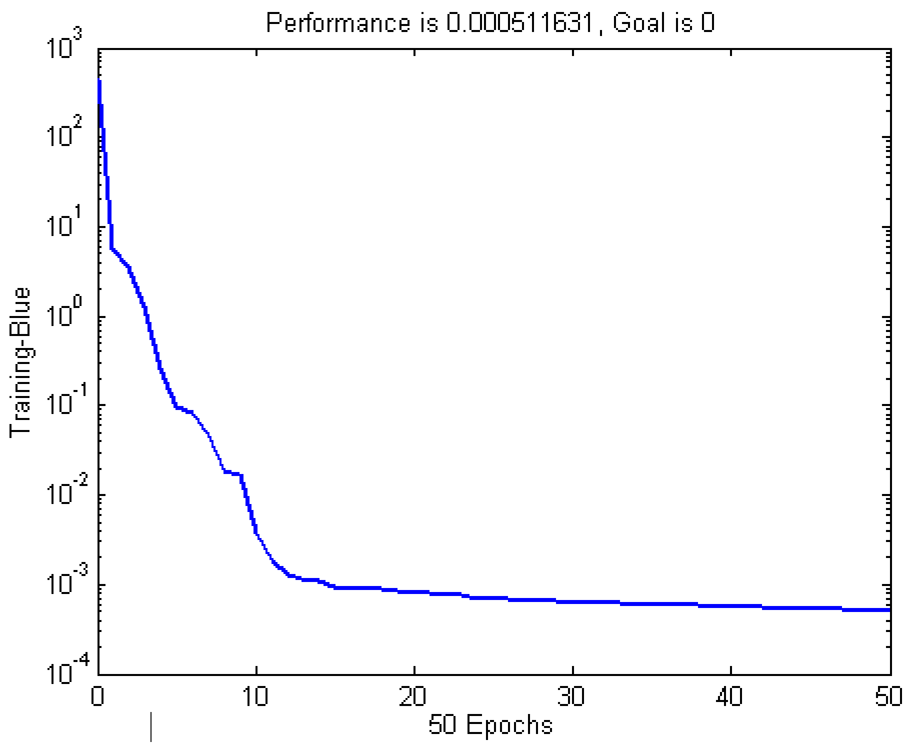


FIGURE 38. Progression of training using ln(strain), ln(strain rate), 1/temperature, ln(stress) with 20 neurons, LM training.

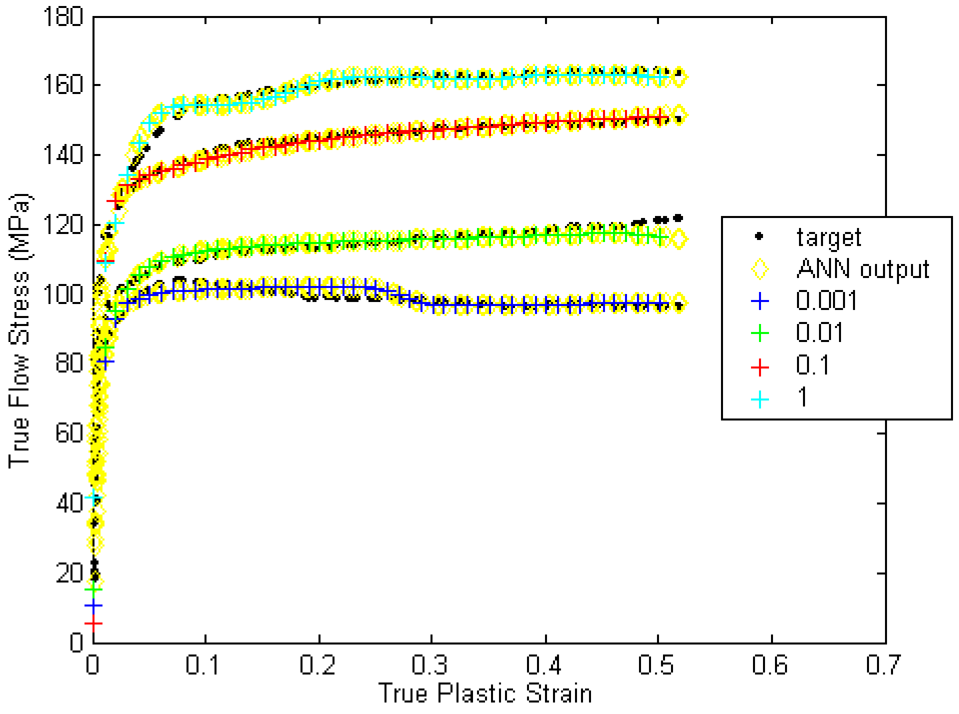


FIGURE 39. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 20 neurons, LM training, for 300ºC.

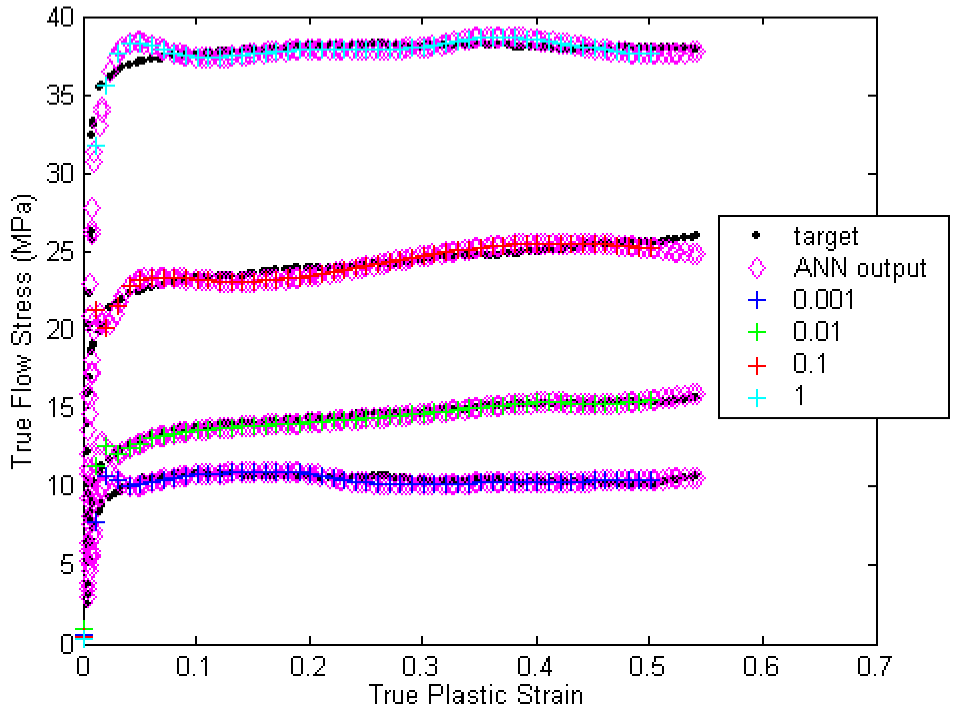


FIGURE 40. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 20 neurons, LM training, for 550ºC.

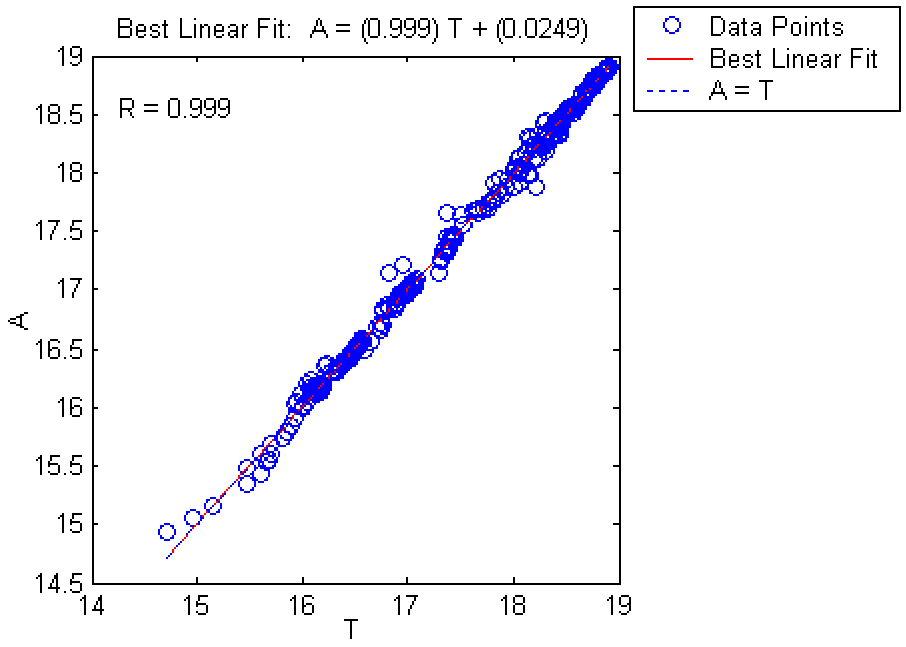


FIGURE 41. Linear regression using ln(strain), ln(strain rate), 1/temperature, ln(stress) for 10 neurons, LM trained network showing network output (A) versus targets (T).

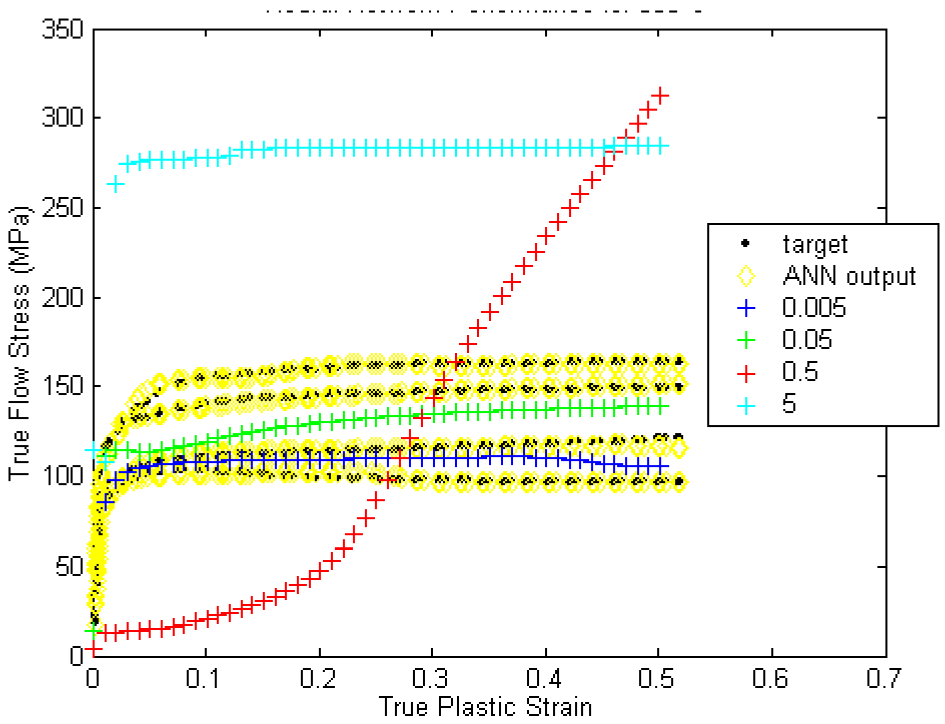


FIGURE 42. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 20 neurons, LM training, for 300ºC at intermediate strain rate values.

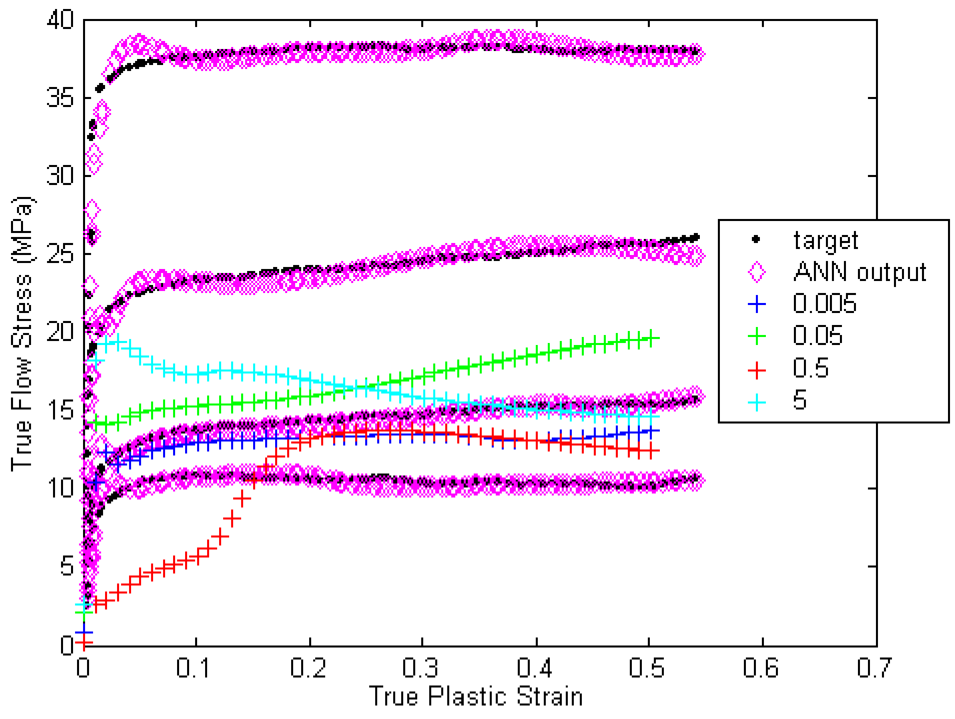


FIGURE 43. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 20 neurons, LM training, for 550ºC at intermediate strain rate values.

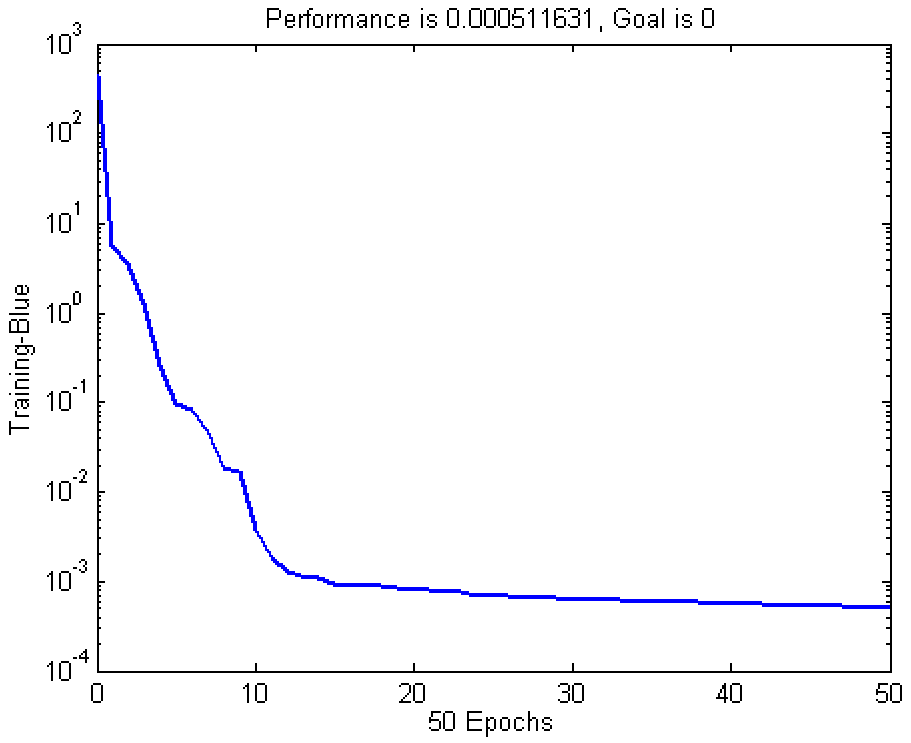


FIGURE 44. Progression of training using ln(strain), ln(strain rate), 1/temperature, ln(stress) with 50 neurons, LM training.

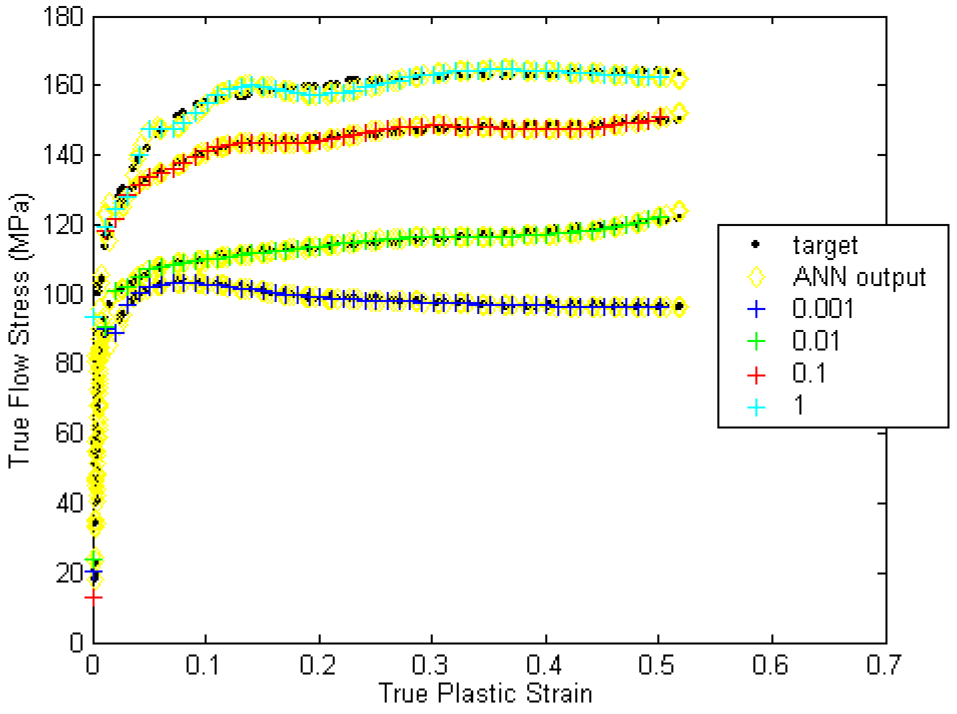


FIGURE 45. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 50 neurons, LM training, for 300ºC.

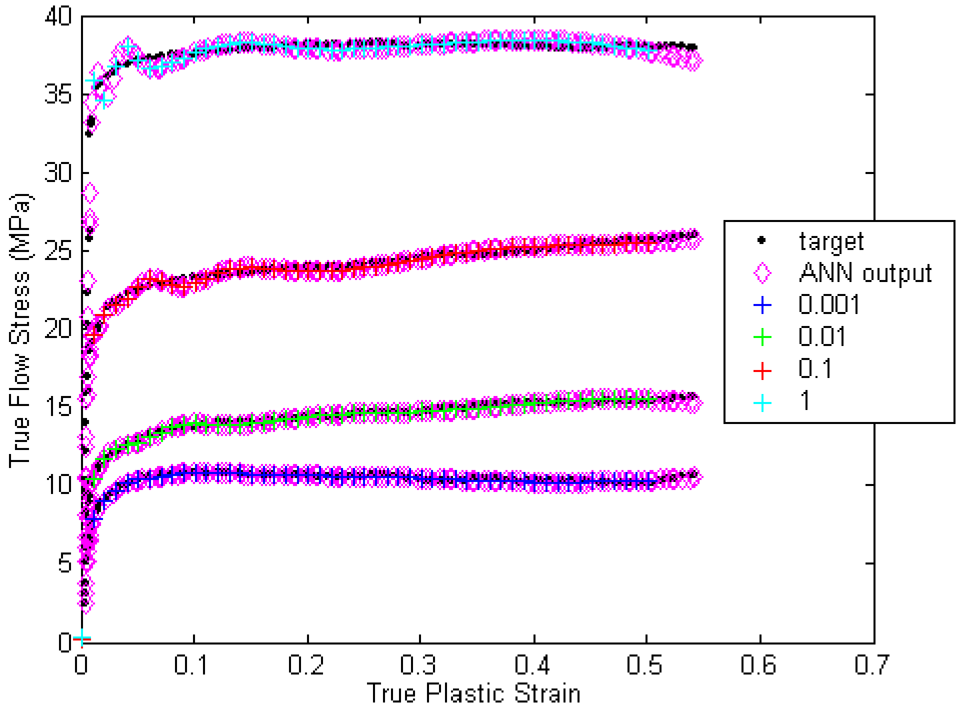


FIGURE 46. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 50 neurons, LM training, for 550ºC.

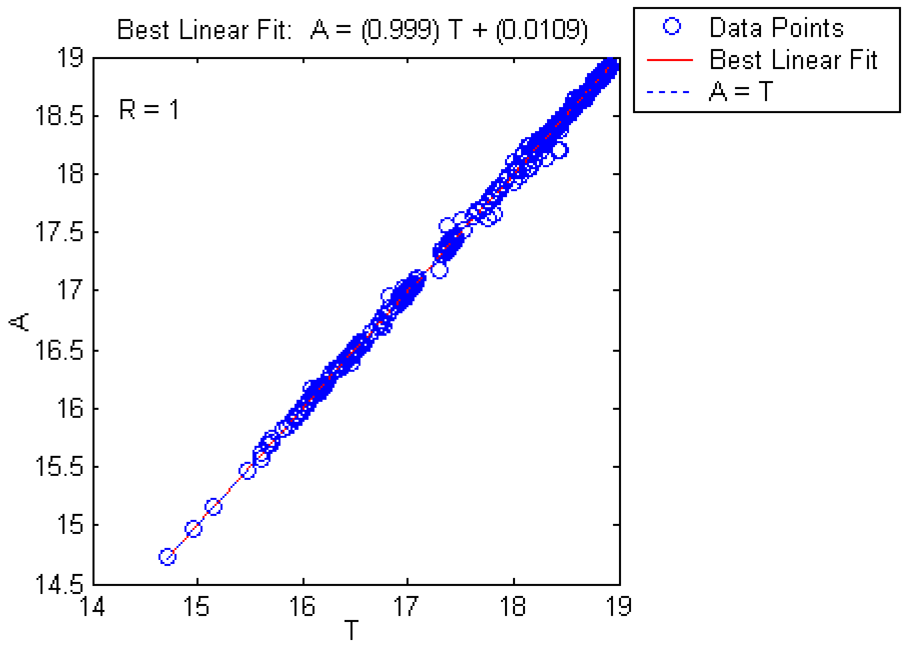


FIGURE 47. Linear regression using ln(strain), ln(strain rate), 1/temperature, ln(stress) for 50 neuron, LM trained network showing network output (A) versus targets (T).

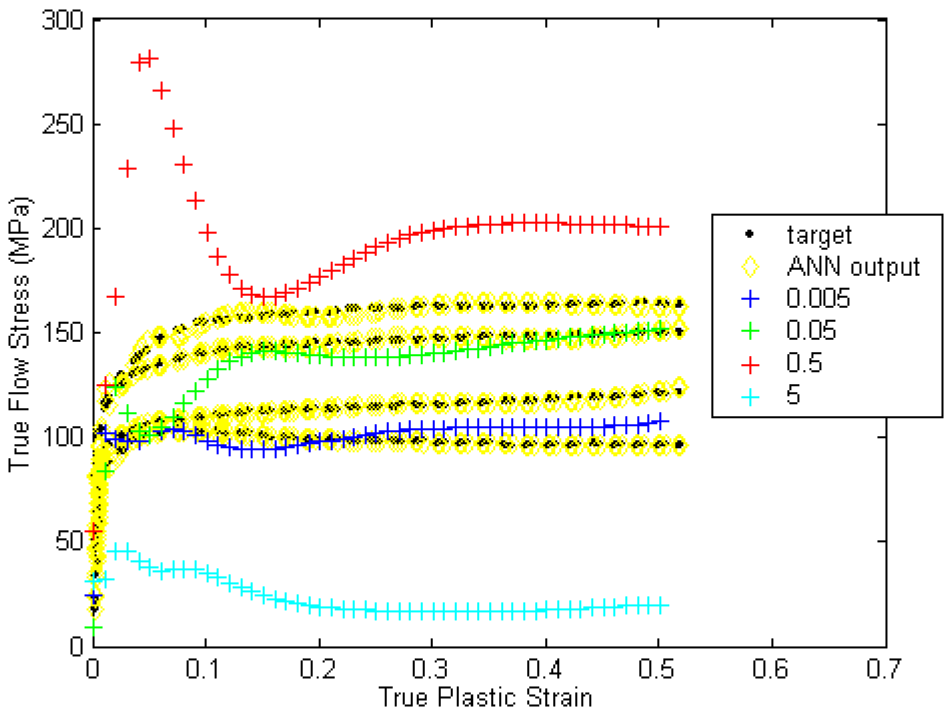


FIGURE 48. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 50 neurons, LM training, for 300ºC at intermediate strain rate values.

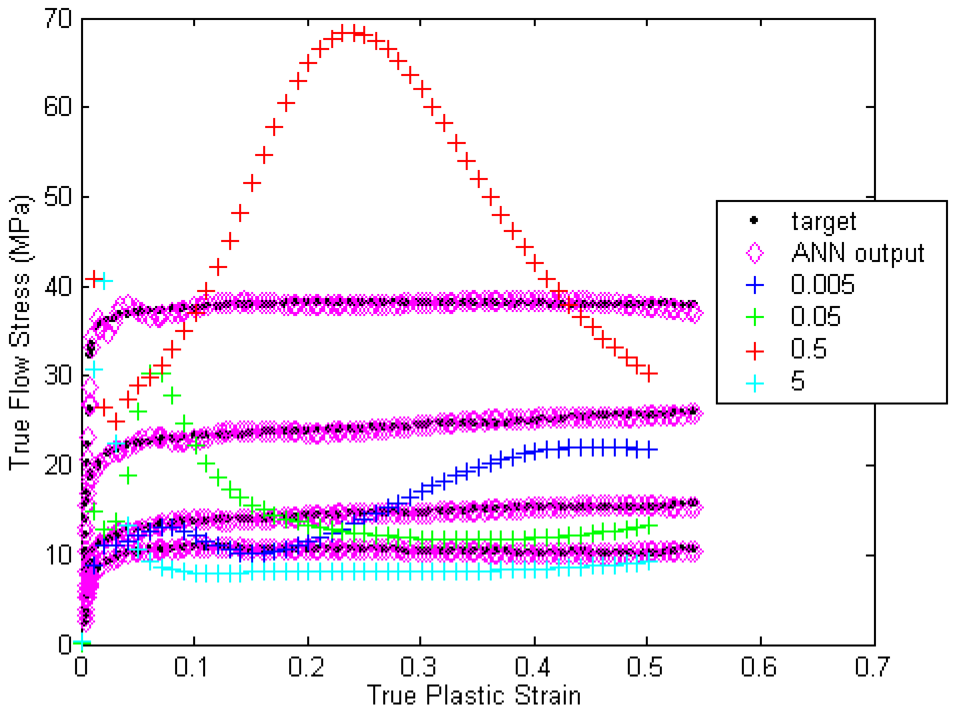


FIGURE 49. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 20 neurons, LM training, for 550ºC at intermediate strain rate values.

Modeling Strain Rates Outside Network Training Set

In an effort to more properly reflect intermediate values of strain rate, several two layer networks were trained and queried. Table 7 shows the architecture, training algorithm, and other network information for this set. The networks in the table again use modified data, but in this case Bayesian Regularization (BR) as provided by MATLAB is used as the training algorithm. This enables one to see the actual number of parameters (weights and biases) that are actually important to the model. Many network configurations were run and the figures that follow are given as pertinent examples. As the first three network figures indicate, intermediate values of strain rate still produce greater swings than desired. To potentially remedy this situation, the data is further modified by normalization and similar network architectures are then trained. The normalization greatly improves the response at intermediate rates. At this point the 8-3 BR network appears to provide the "best" network output.

TABLE 7. Network training parameters and results for "best" training attempt, 5 trials.

Data Condition	Training Algorithm	Number of Neurons	Epochs Completed	Mean Squared Error	Parameters	R	Figures
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$	BR	6-3	500	2.304	41.3/49	0.999	50, 51, 52, 53, 54, 55
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$	BR	8-3	500	1.308	53.4/63	0.999	56, 57, 58, 59, 60, 61
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$	BR	10-3	500	0.304	74.23/77	1.000	62, 63, 64, 65, 66, 67
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$ normalized	BR	8-3	500	0.103	51.4/63	1.000	68, 69, 70, 71, 72, 73

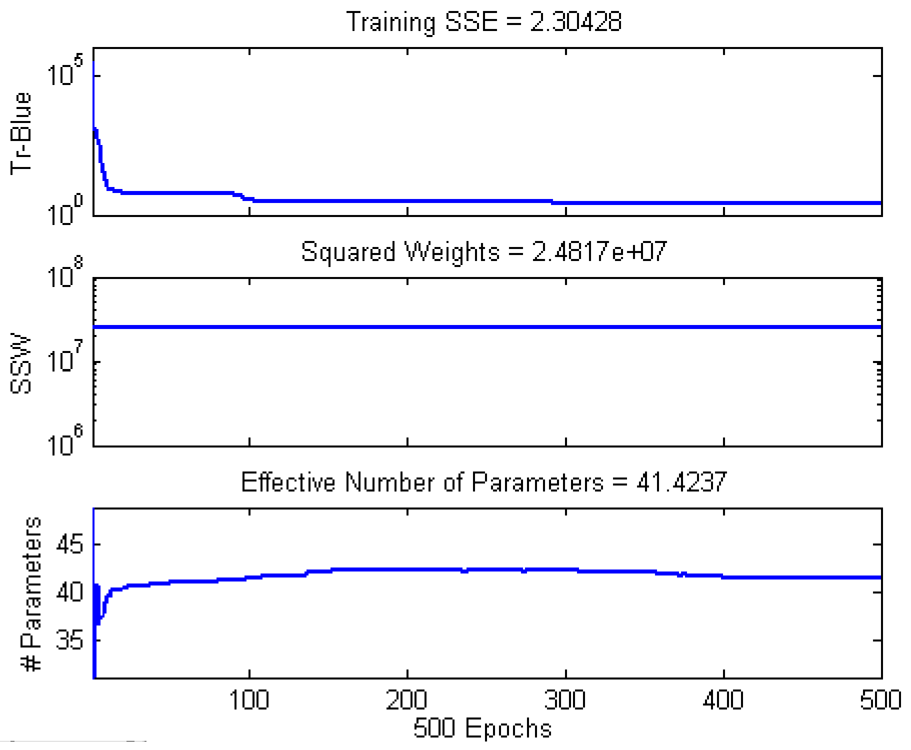


FIGURE 50. Progression of training using ln(strain), ln(strain rate), 1/temperature, ln(stress) with 6-3 neurons, BR training algorithm.

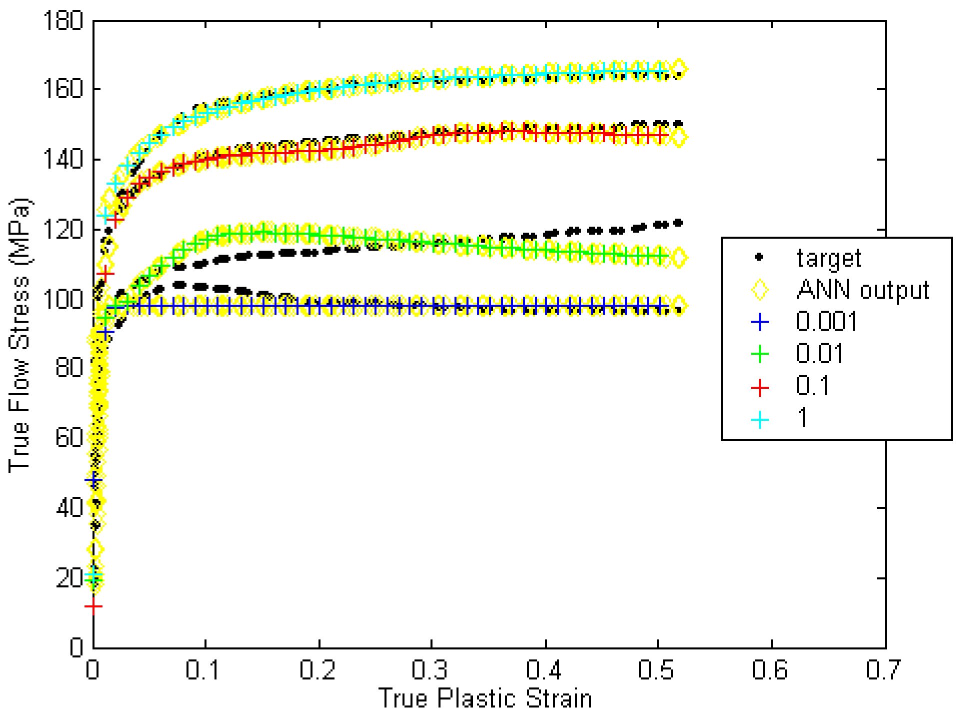


FIGURE 51. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 6-3 neurons, BR training, for 300ºC.

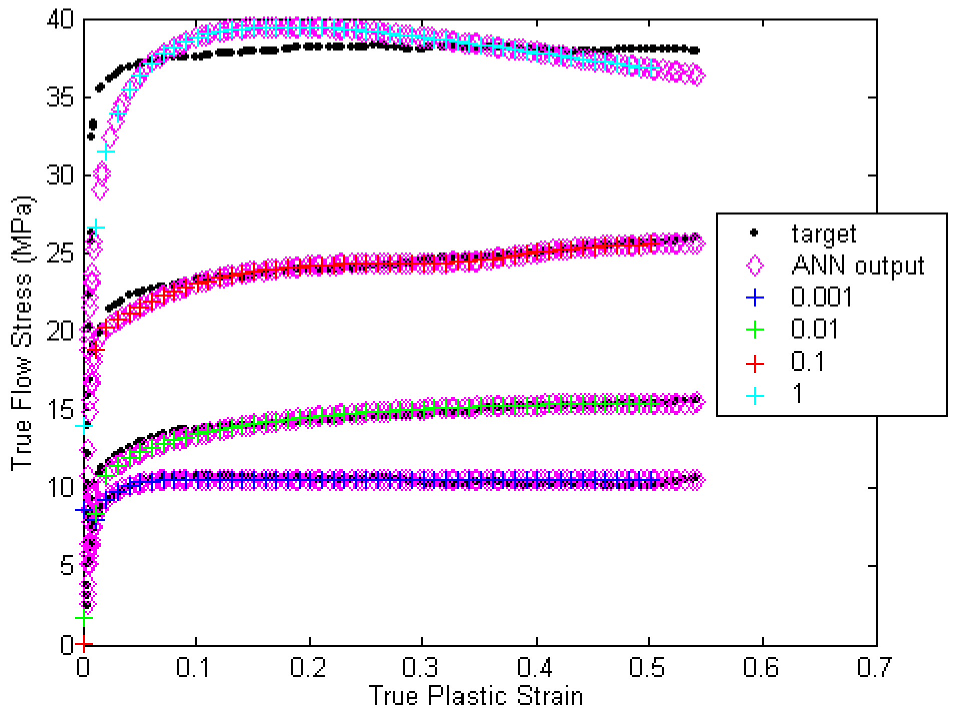


FIGURE 52. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 6-3 neurons, BR training, for 550ºC.

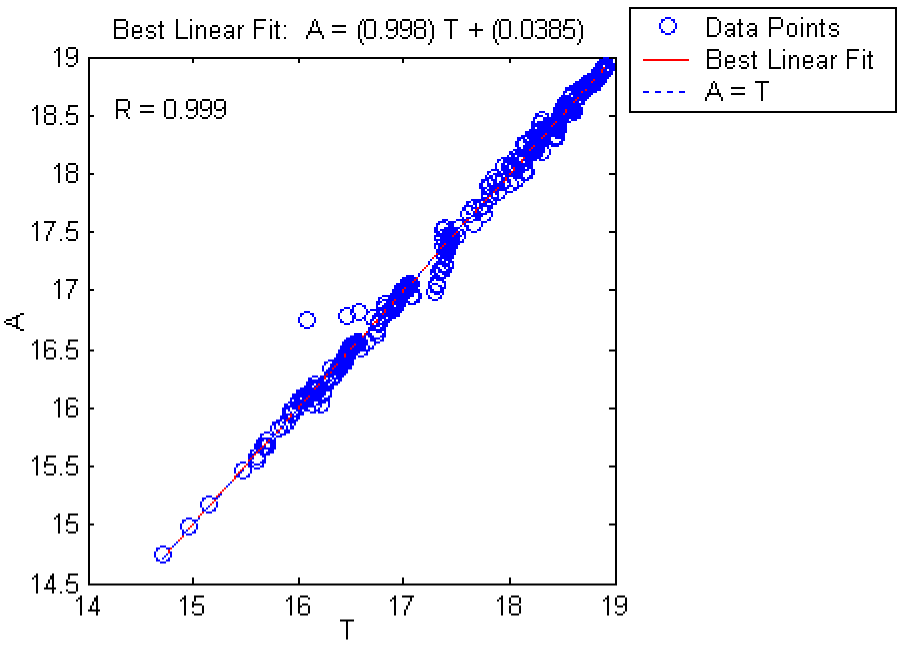


FIGURE 53. Linear regression using ln(strain), ln(strain rate), 1/temperature, ln(stress) for 6-3 neurons, BR trained network showing network output (A) versus targets (T).

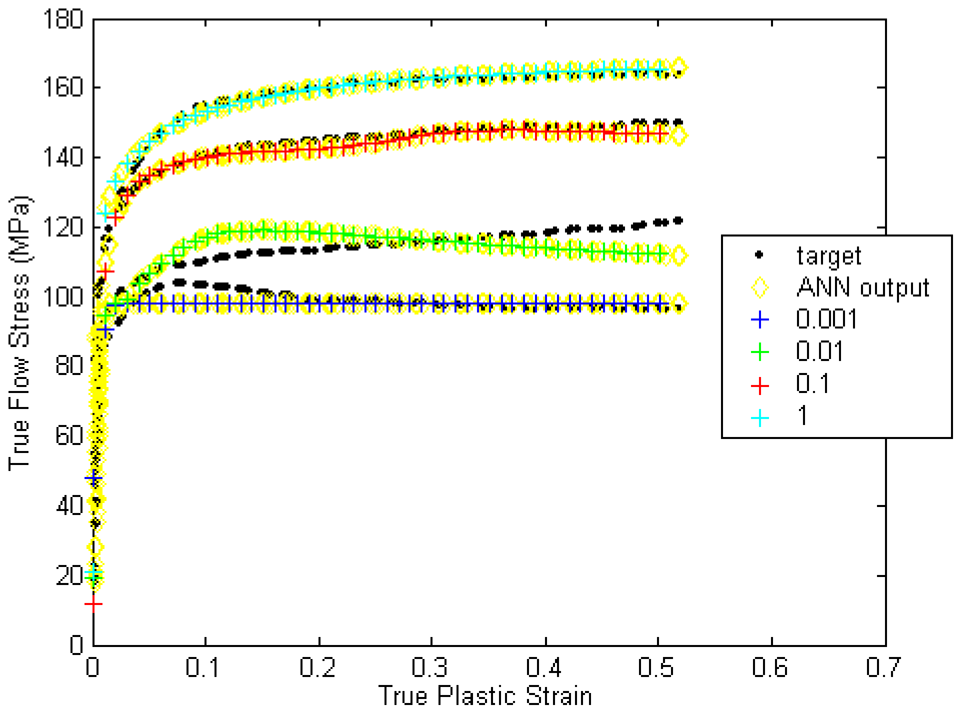


FIGURE 54. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 6-3 neurons, BR training, for 300ºC at intermediate strain rate values.

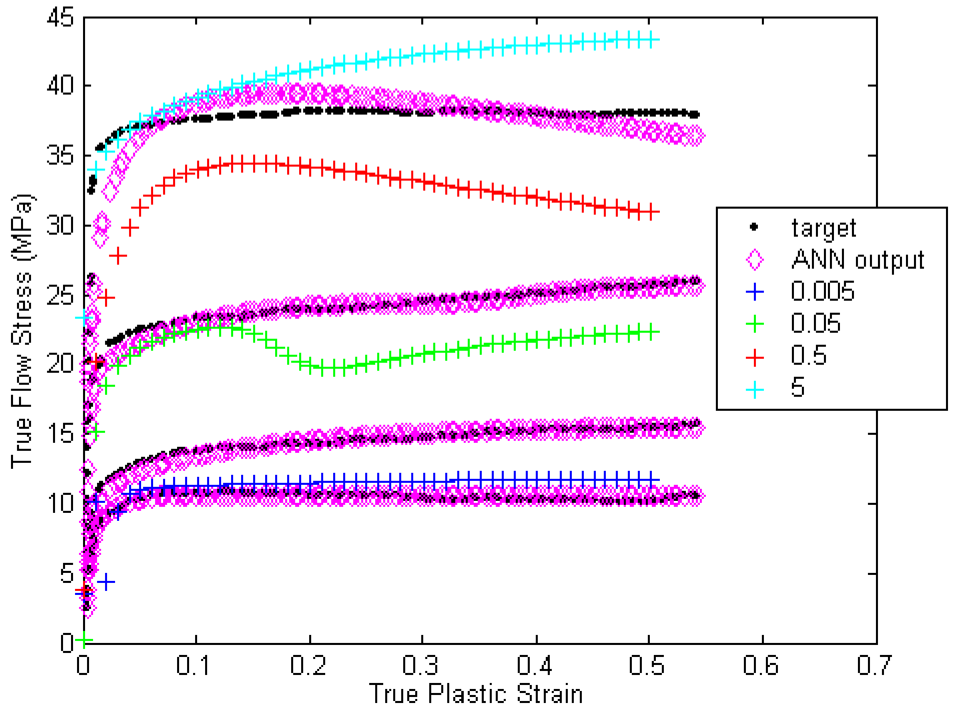


FIGURE 55. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 6-3 neurons, BR training, for 550ºC at intermediate strain rate values.

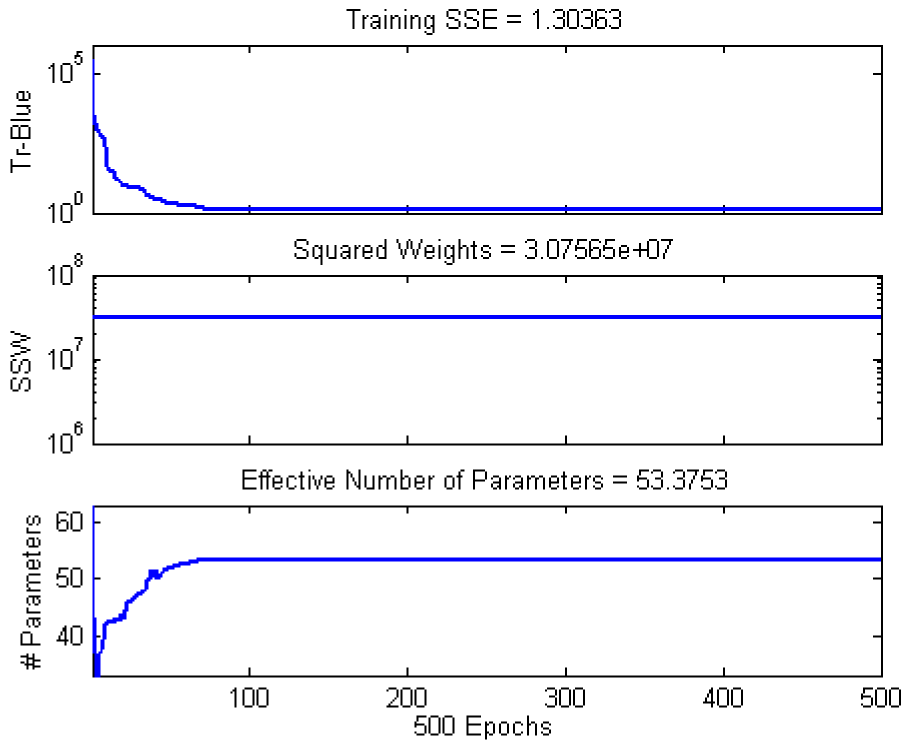


FIGURE 56. Progression of training using ln(strain), ln(strain rate), 1/temperature, ln(stress) with 8-3 neurons, BR training algorithm.

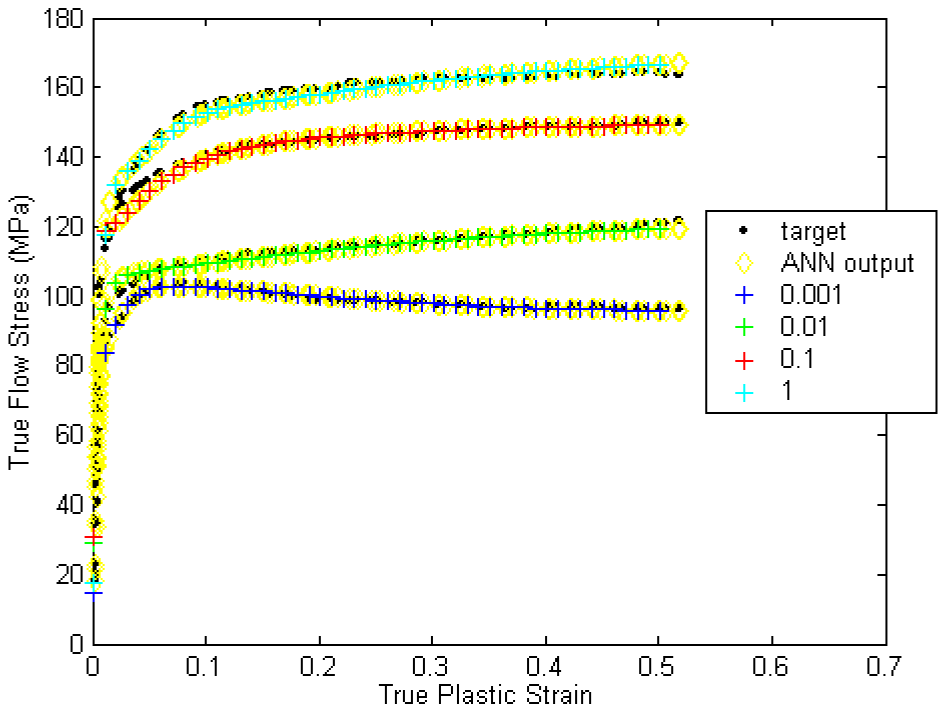


FIGURE 57. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, for 300ºC.

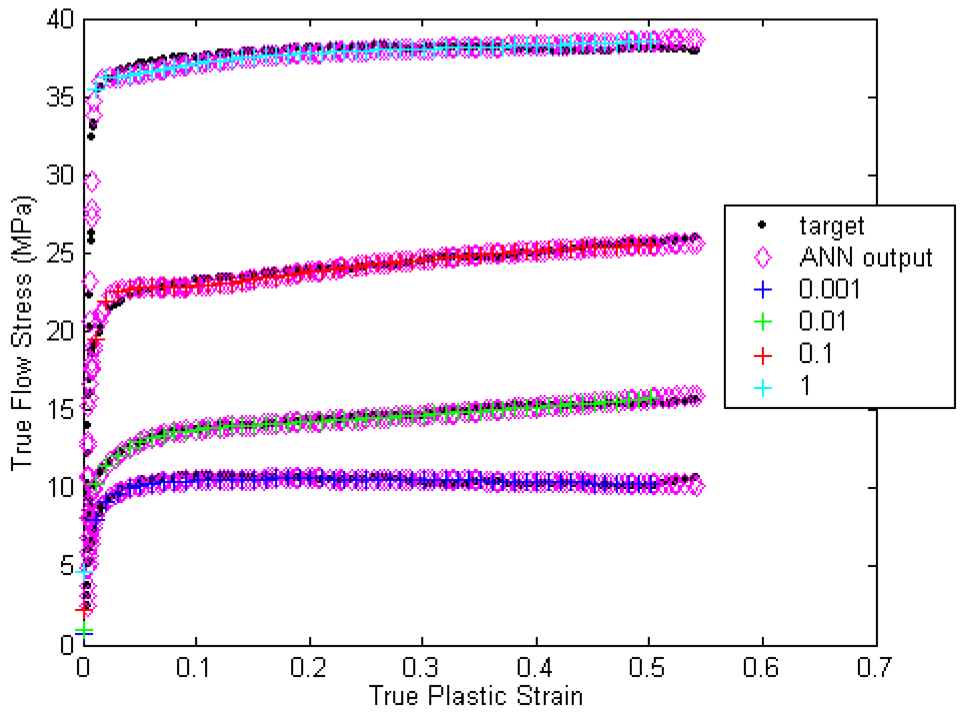


FIGURE 58. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, for 550ºC.

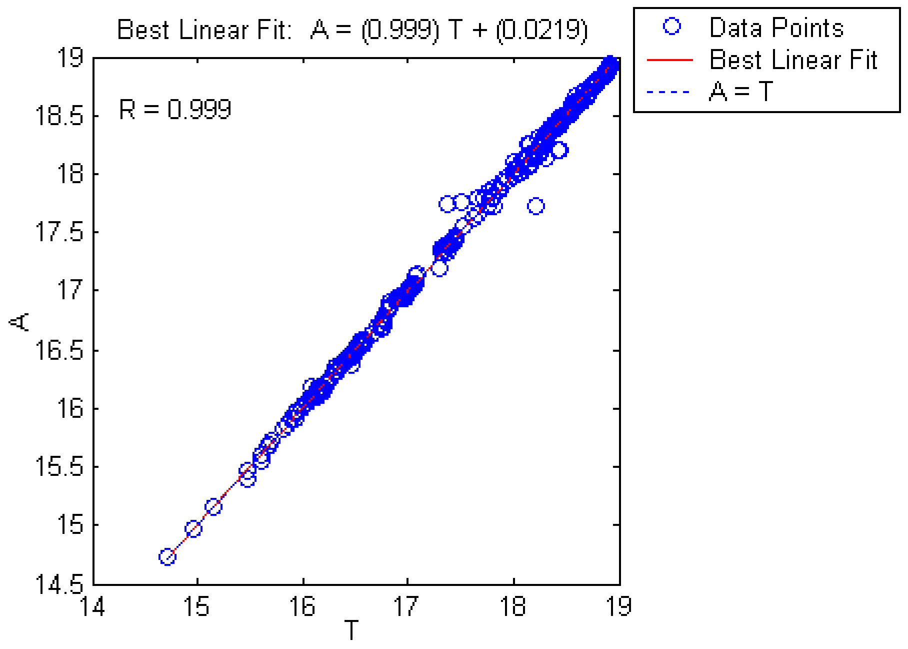


FIGURE 59. Linear regression using ln strain, ln strain rate, reciprocal temperature, ln stress for 8-3 neurons, BR trained network showing network output (A) versus targets (T).

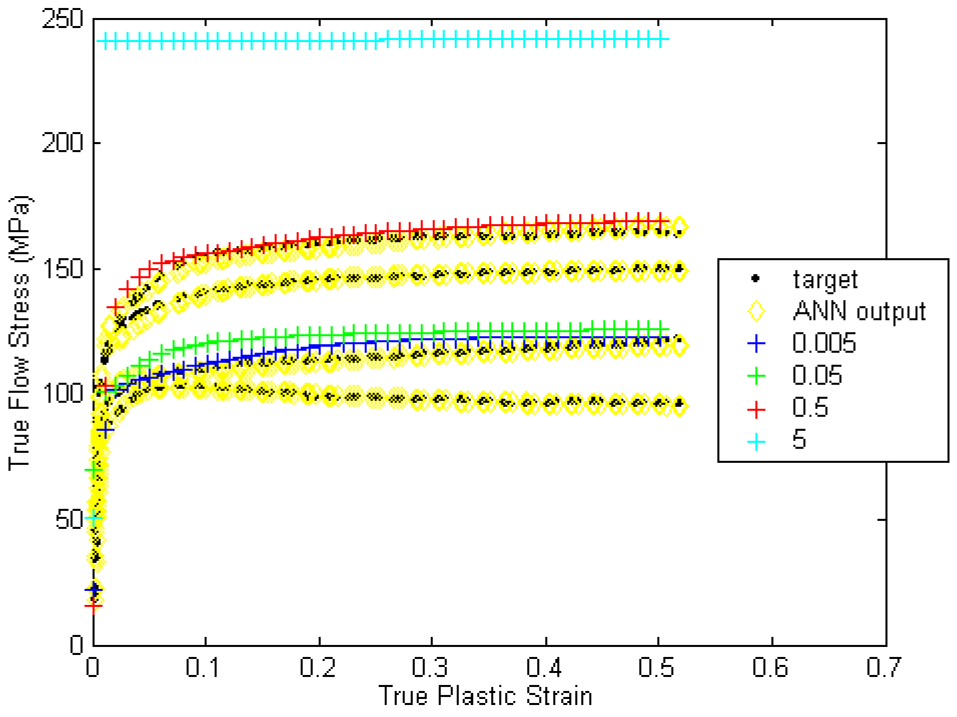


FIGURE 60. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, for 300ºC at intermediate strain rate values.

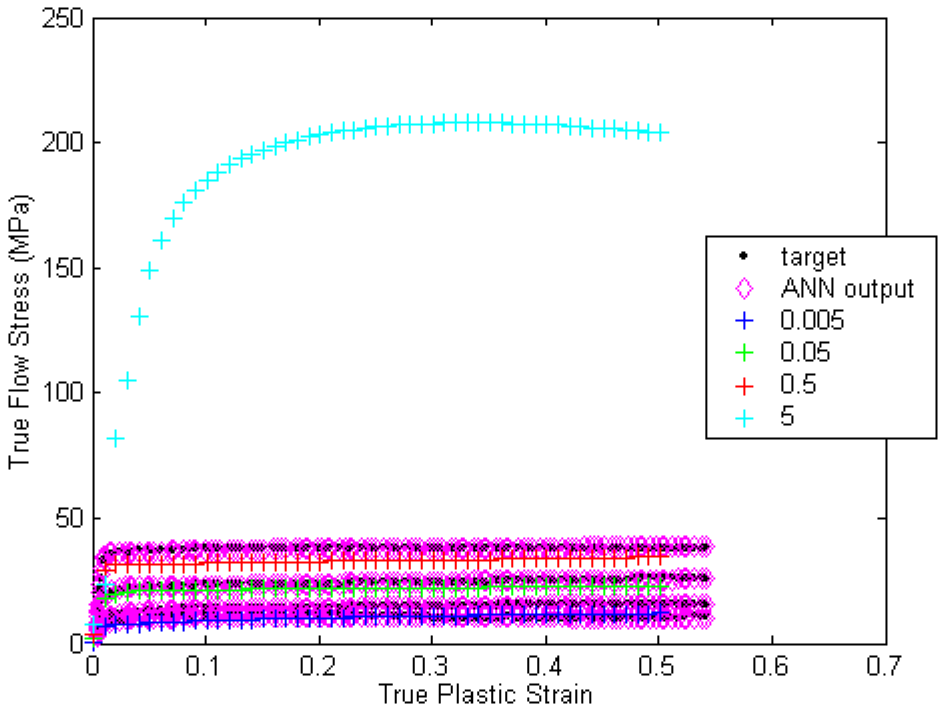


FIGURE 61. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, for 550ºC at intermediate strain rate values.

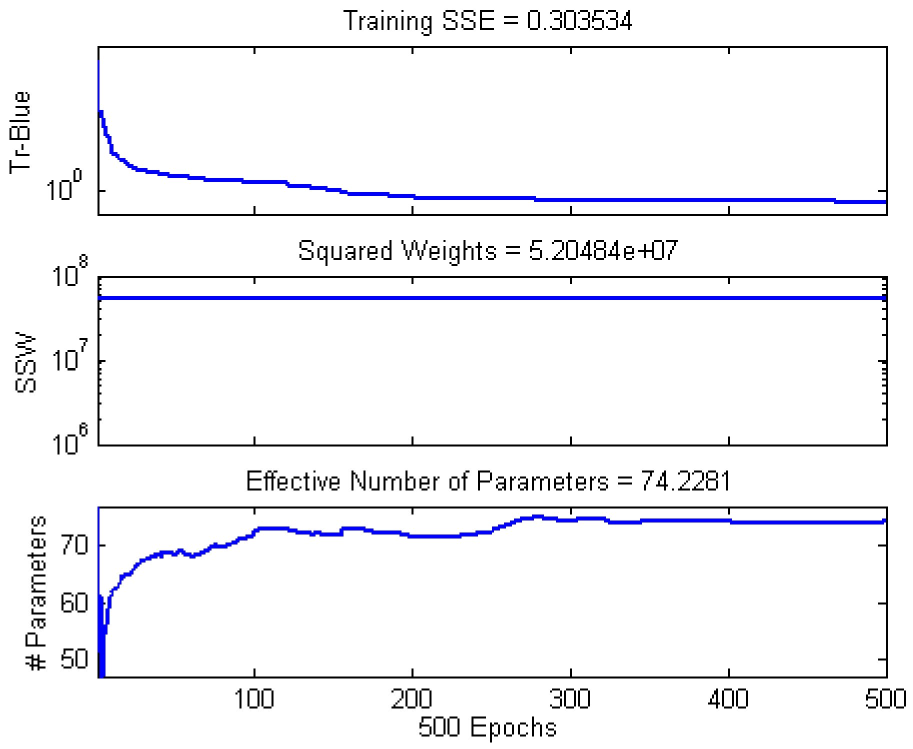


FIGURE 62. Progression of training using ln(strain), ln(strain rate), 1/temperature, ln(stress) with 10-3 neurons, BR training algorithm.

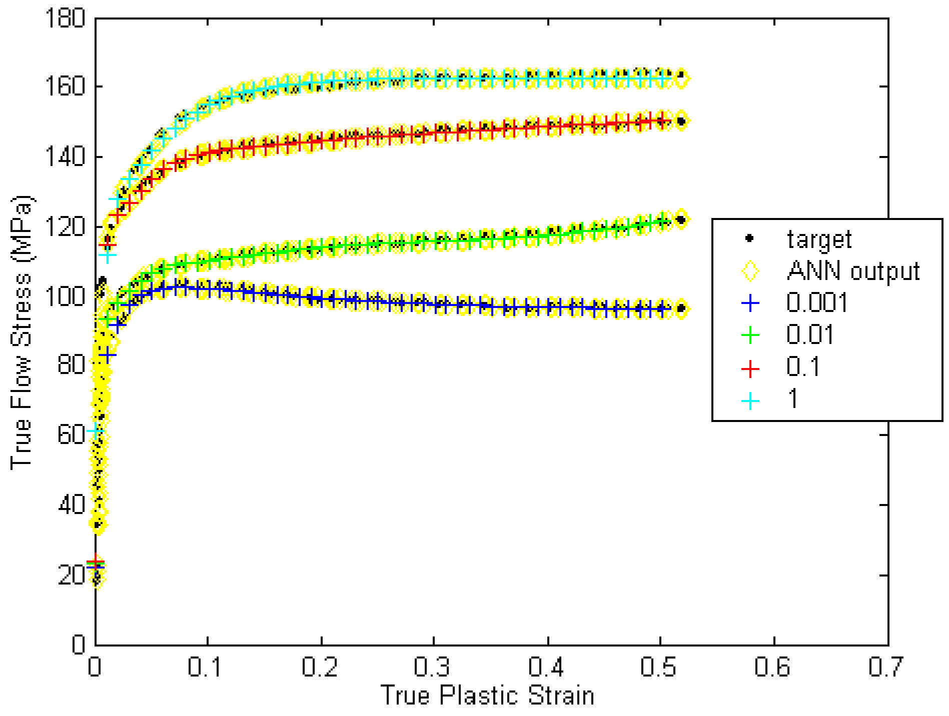


FIGURE 63. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 10-3 neurons, BR training, for 300ºC.

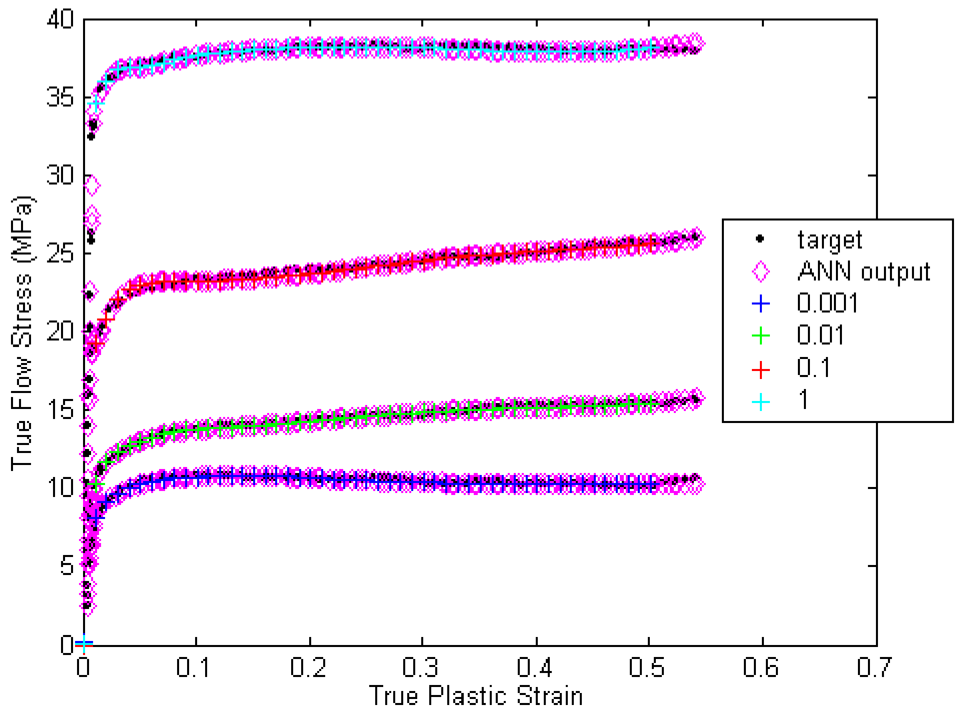


FIGURE 64. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 10-3 neurons, BR training, for 550ºC.

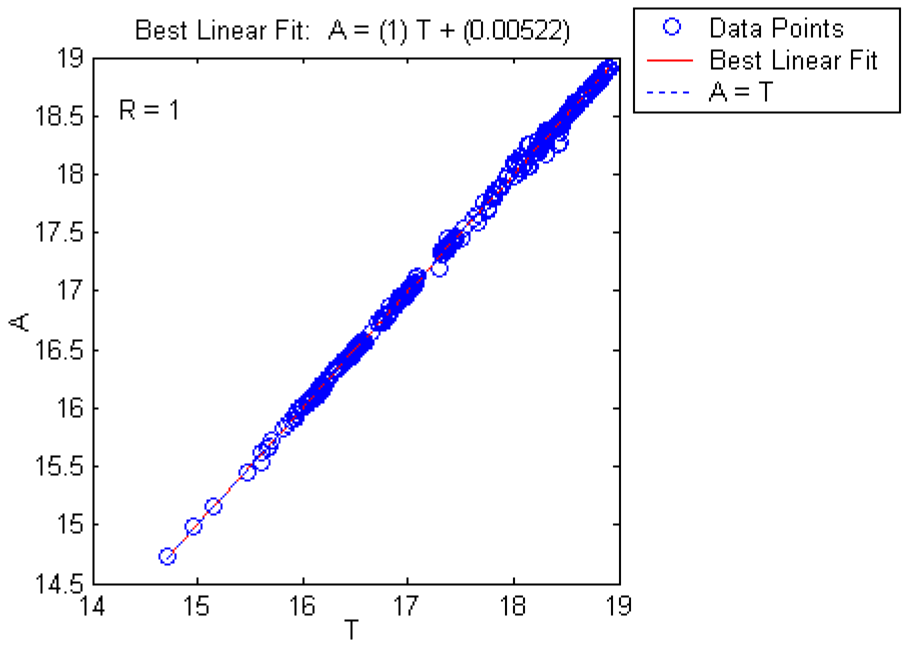


FIGURE 65. Linear regression using ln strain, ln strain rate, reciprocal temperature, ln stress for 10-3 neurons, BR trained network showing network output (A) versus targets (T).

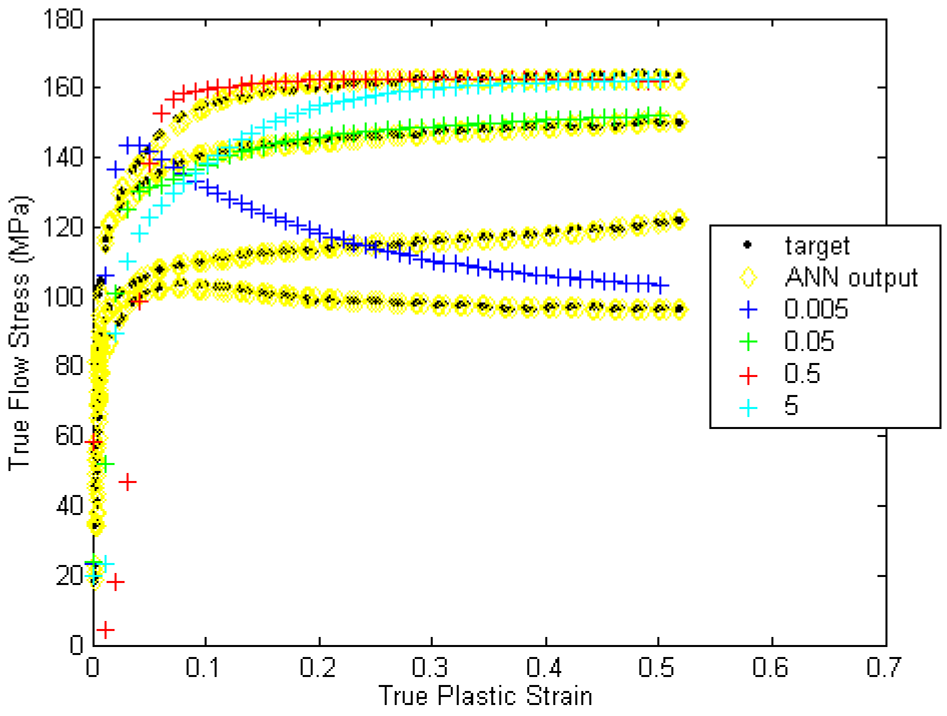


FIGURE 66. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 10-3 neurons, BR training, for 300ºC at intermediate strain rate values.

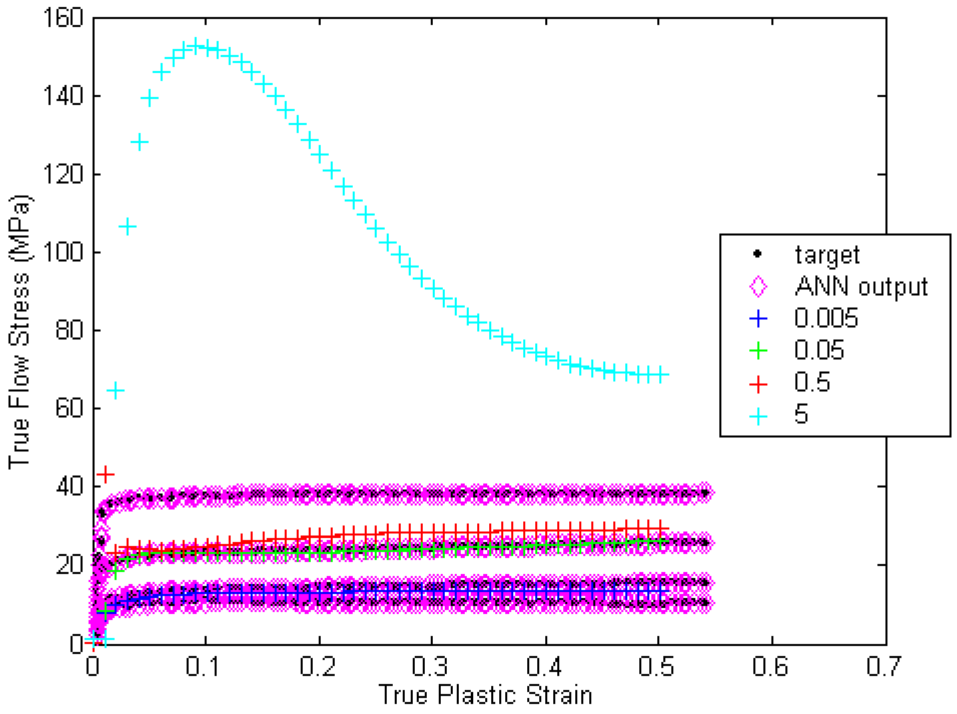


FIGURE 67. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 10-3 neurons, BR training, for 550ºC at intermediate strain rate values.

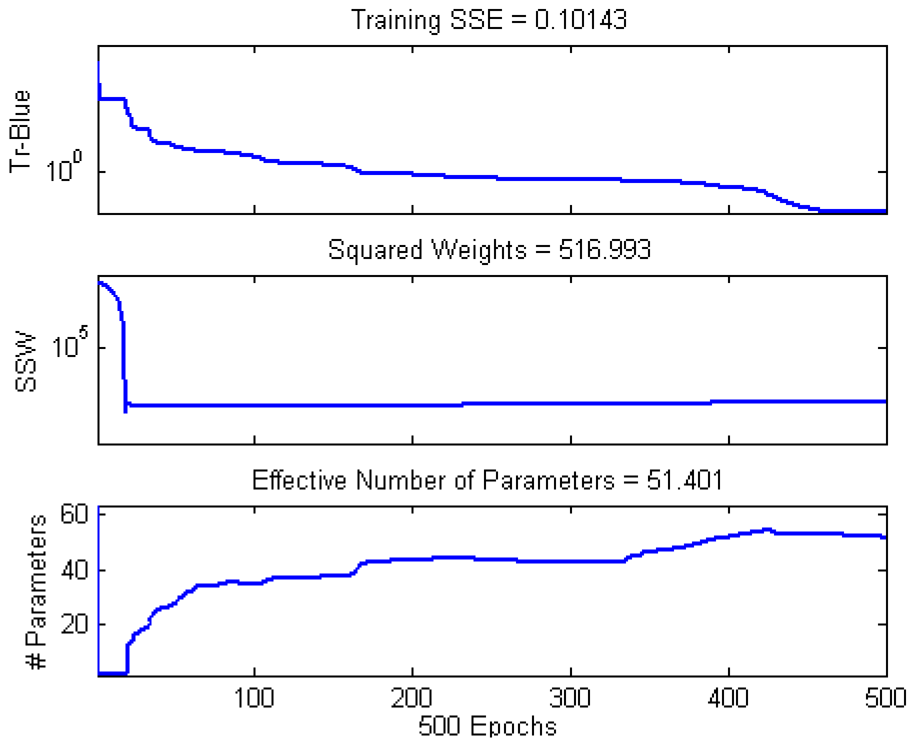


FIGURE 68. Progression of training using normalized values of ln(strain), ln(strain rate), 1/temperature, ln(stress) with 8-3 neurons, BR training algorithm.

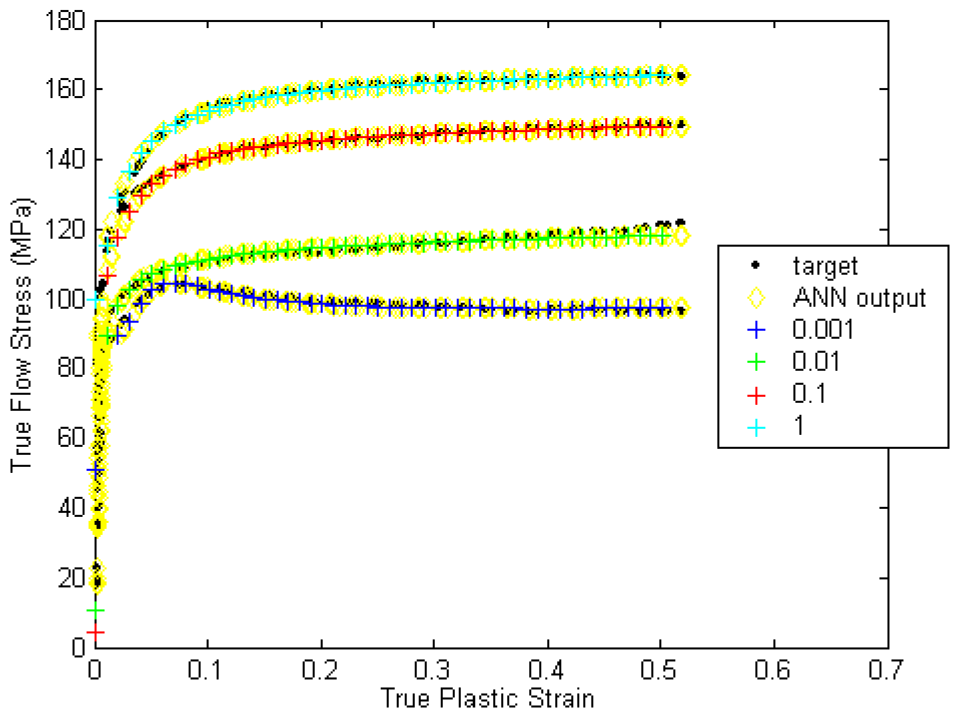


FIGURE 69. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, and normalization for 300ºC.

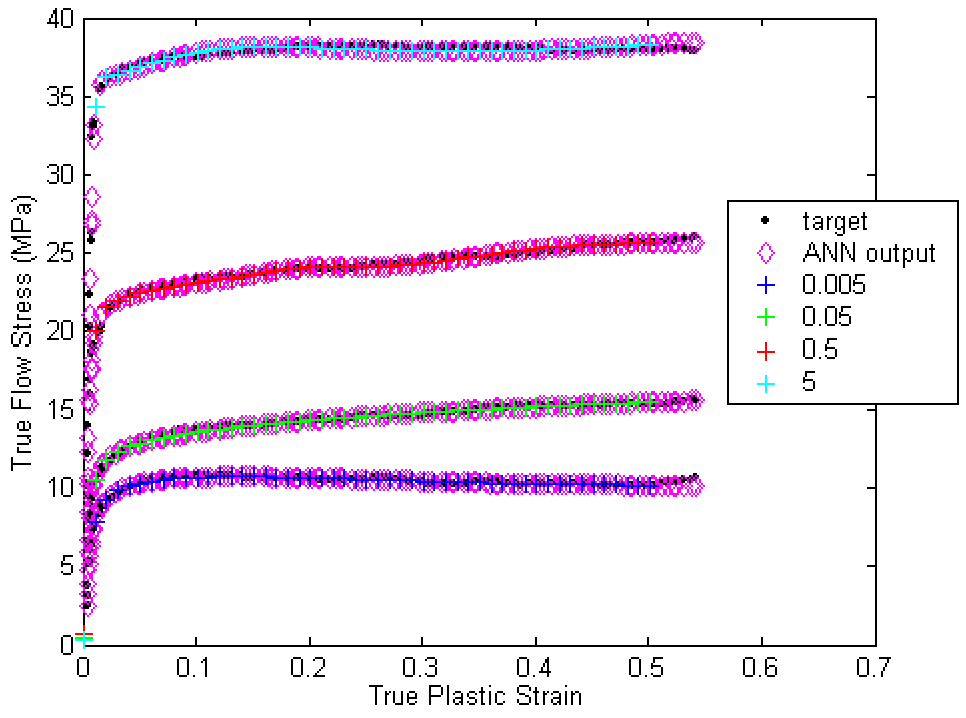


FIGURE 70. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, and normalization for 550ºC.

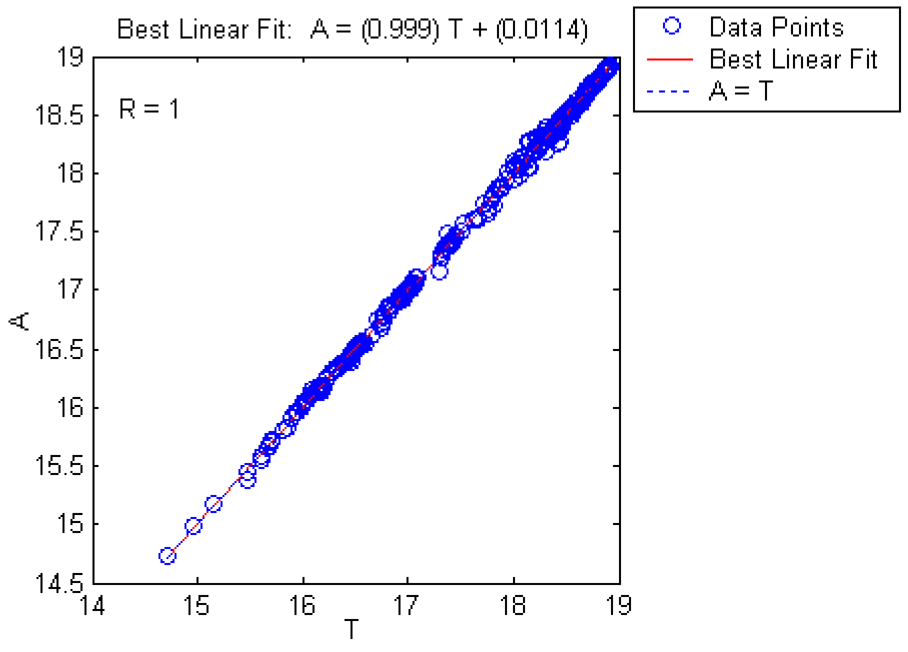


FIGURE 71. Linear regression using ln strain, ln strain rate, reciprocal temperature, ln stress for 8-3 neurons, BR trained network, with normalization showing network output (A) versus targets (T).

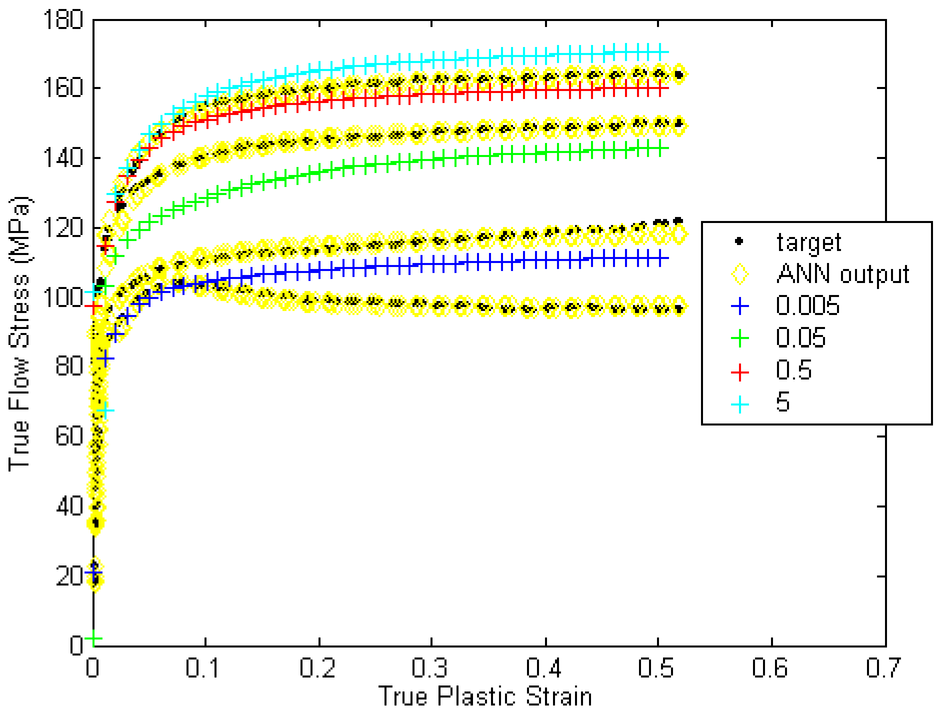


FIGURE 72. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, and normalization for 300ºC at intermediate strain rate values.

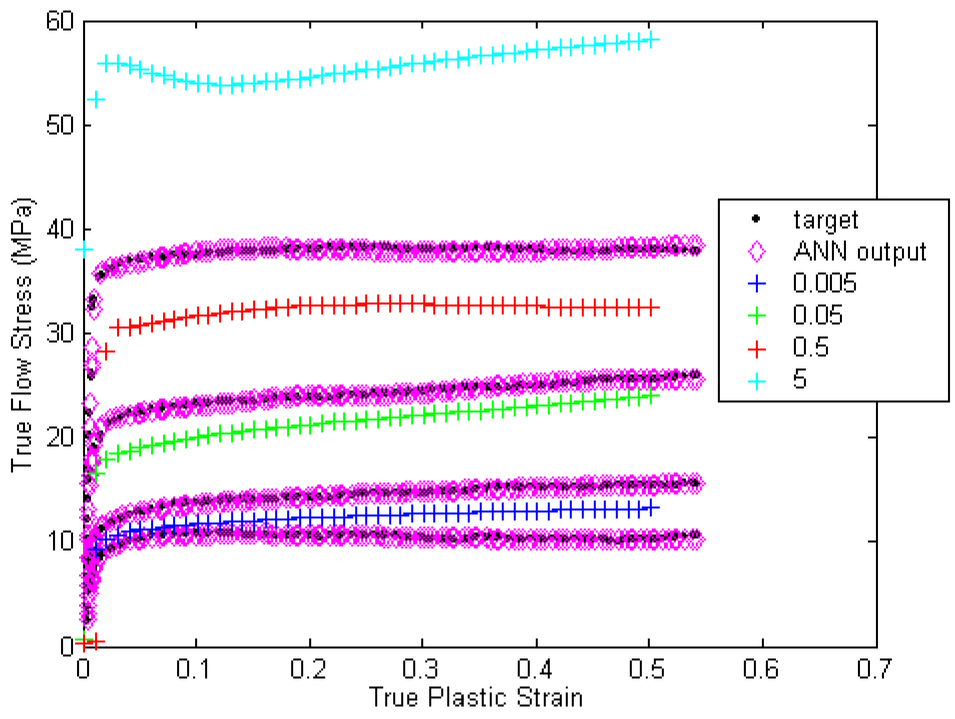


FIGURE 73. Network with ln(strain), ln(strain rate), 1/temperature, ln(stress), 8-3 neurons, BR training, and normalization for 550ºC at intermediate strain rate values.

Modeling Temperature & Strain Rates Outside Network Training Set

While the 8-3 BR trained network using the normalized values appears to produce reasonable responses for a wide range of strain rates, another factor needs to be considered. The network should also produce plausible values of flow stress as temperature varies. To this end, two additional figures, 74 and 75, were generated for the 8-3 BR network. It clearly shows widely varying response for temperatures that should produce curves similar to those for 300 and 550ºC, but falling in between. Several attempts at training the 8-3 BR network along with others failed to result in acceptable response for intermediate temperature values.

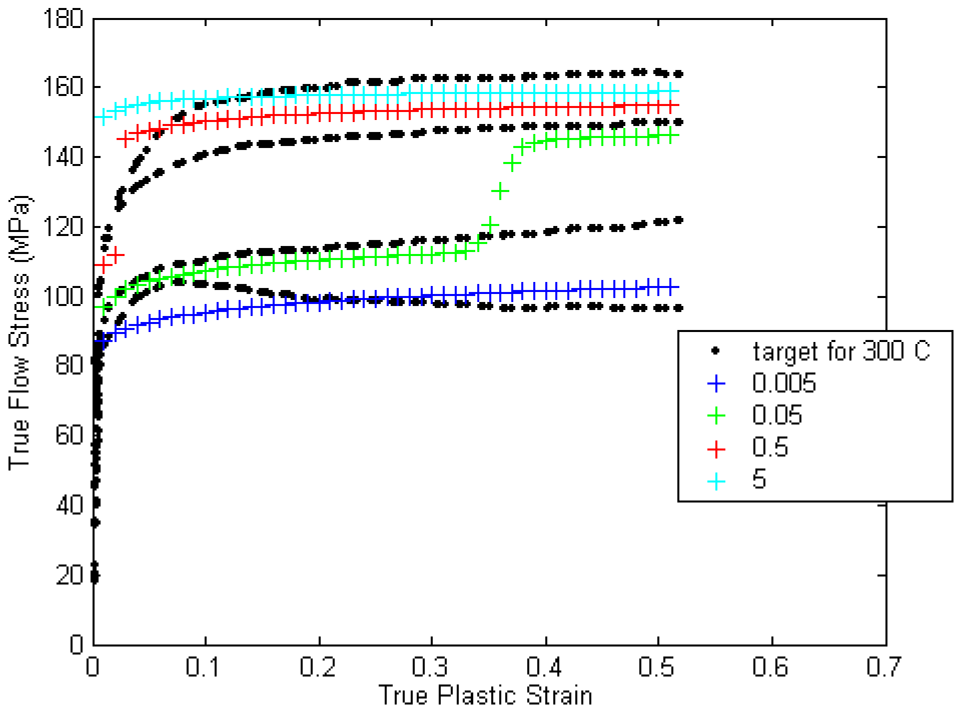


FIGURE 74. Intermediate temperature of 375ºC produce suspect flow stresses from the 8-3 BR network when trained with normalized values of ln(strain), ln(strain rate), and 1/temperature.

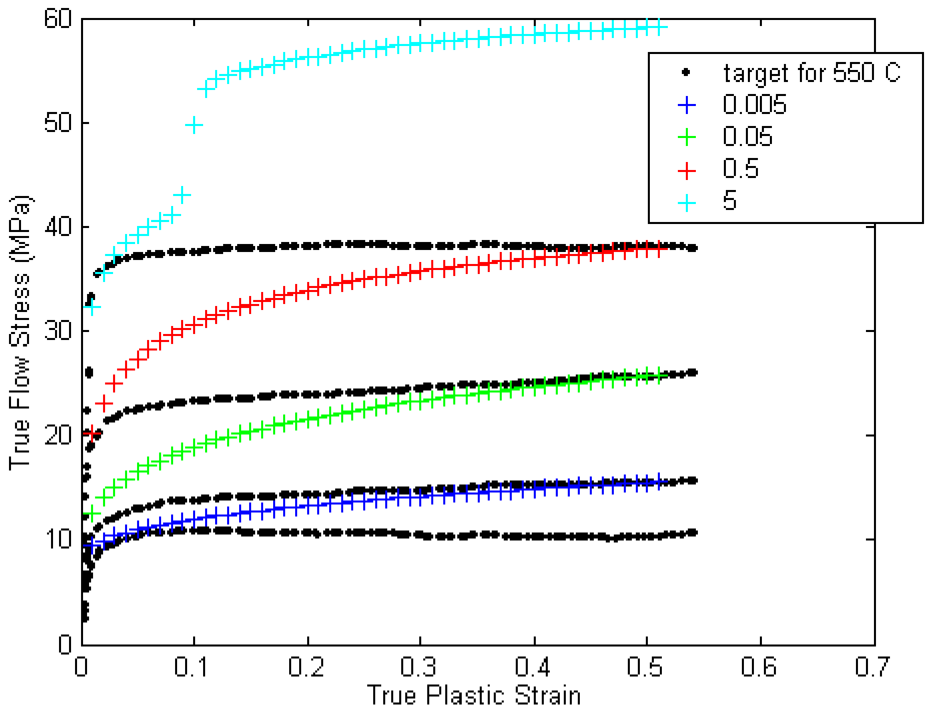


FIGURE 75. Intermediate temperature of 450ºC produce suspect flow stresses from the 8-3 BR network when trained with normalized values of ln(strain), ln(strain rate), and 1/temperature.

The *Hot Working Guide* [1] sited earlier also has a table of values for six temperatures (300, 350, 400, 450, 500, and 550ºC), though only adds 120 more data points. The data set was added to the training set and then several networks trained for five attempts as before.

Table 8 shows the results for three of the network architectures using the additional tabular data (other networks were trained and queried, only the last few are reported below).

TABLE 8. Network training parameters and results for "best" training attempt, 5 trials.

Data Condition	Training Algorithm	Number of Neurons	Epochs Completed	Mean Squared Error	Parameters	R	Figures
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$ normalize + tabular	BR	8-3	3111	0.316	59.72/63	0.997	76, 77, 78, 79, 80, 81, 82, 83
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$ normalize + tabular	BR	10-3	4550	0.305	73.65/77	0.999	84, 85, 86, 87, 88, 89, 90, 91
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$ normalize + tabular	BR	12-3	5000	1.287	29.37/91	0.999	92, 93, 94, 95, 96, 97, 98, 99
$\ln(\varepsilon)$, $\ln(\dot{\varepsilon})$, $\ln(\sigma)$, $\frac{1}{T}$ normalize + tabular	BR	15-3	4575	0.283	73.65/77	0.999	100, 101, 102, 103, 104, 105, 106, 107

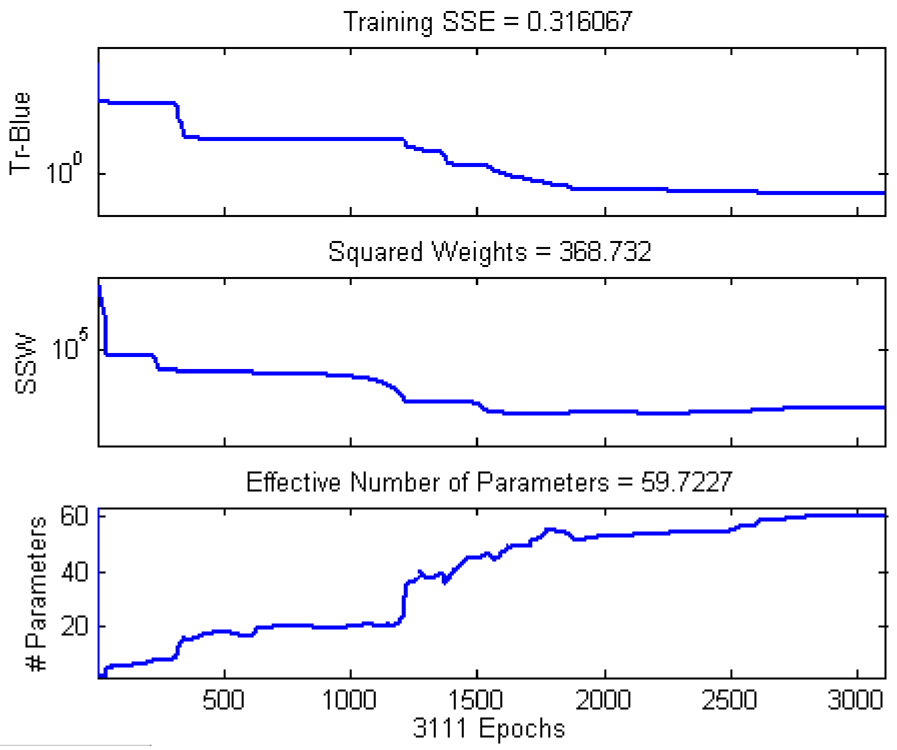


FIGURE 76. Progression of training using normalized values of ln(strain), ln(strain rate), 1/temperature, ln(stress) with 8-3 neurons, BR training algorithm.

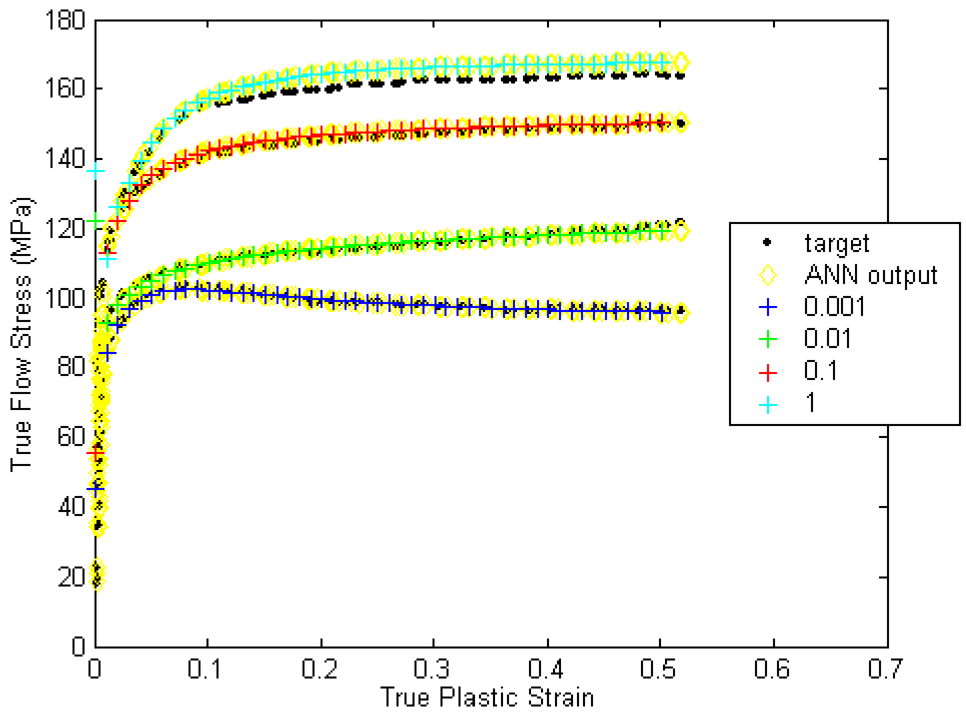


FIGURE 77. 8-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC.

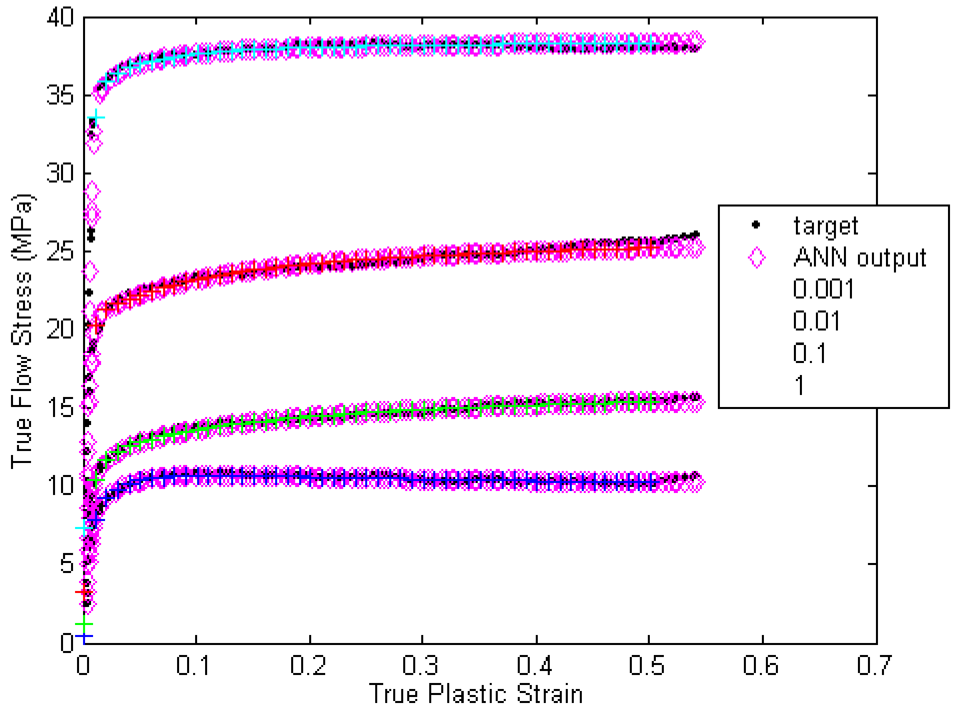


FIGURE 78. 8-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC.

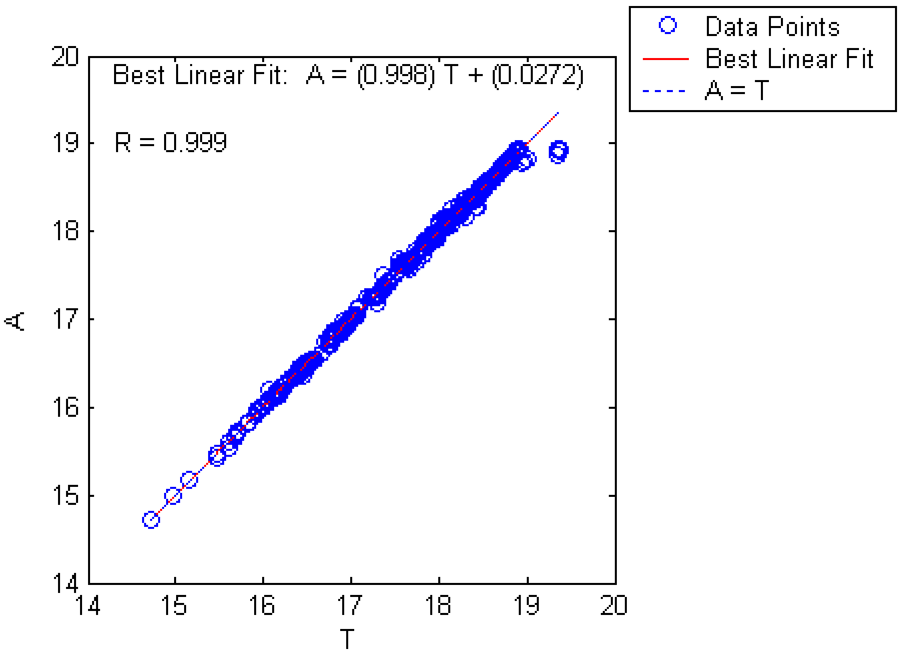


FIGURE 79. Linear regression using ln(strain), ln strain rate, reciprocal temperature, ln stress for 8-3 neurons, BR trained network, with normalization showing network output (A) versus targets (T).

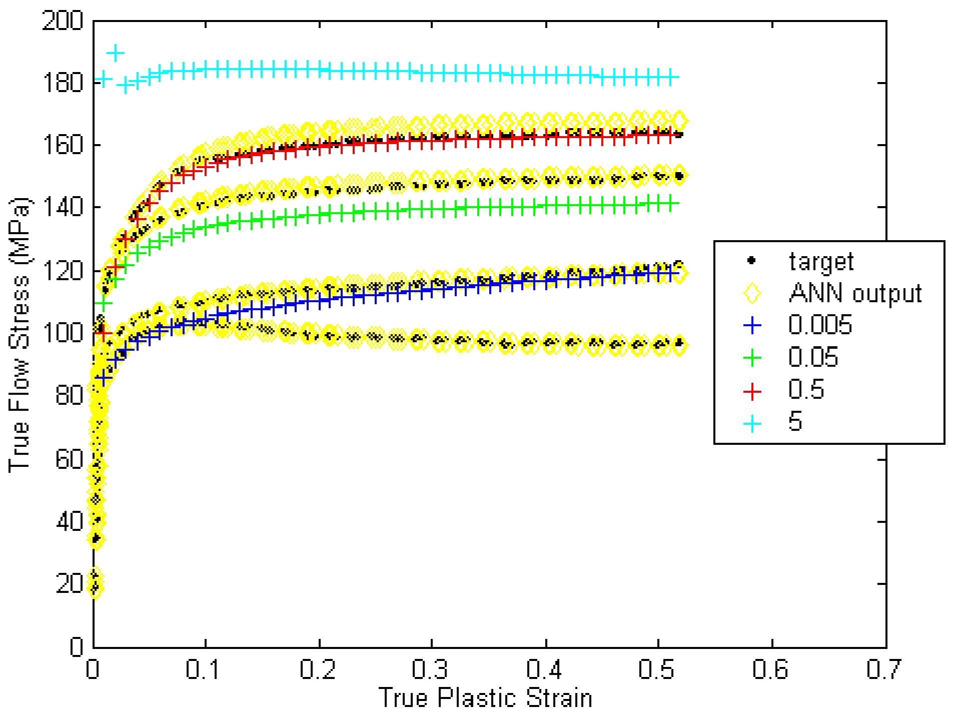


FIGURE 80. 8-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC at intermediate values of strain rate.

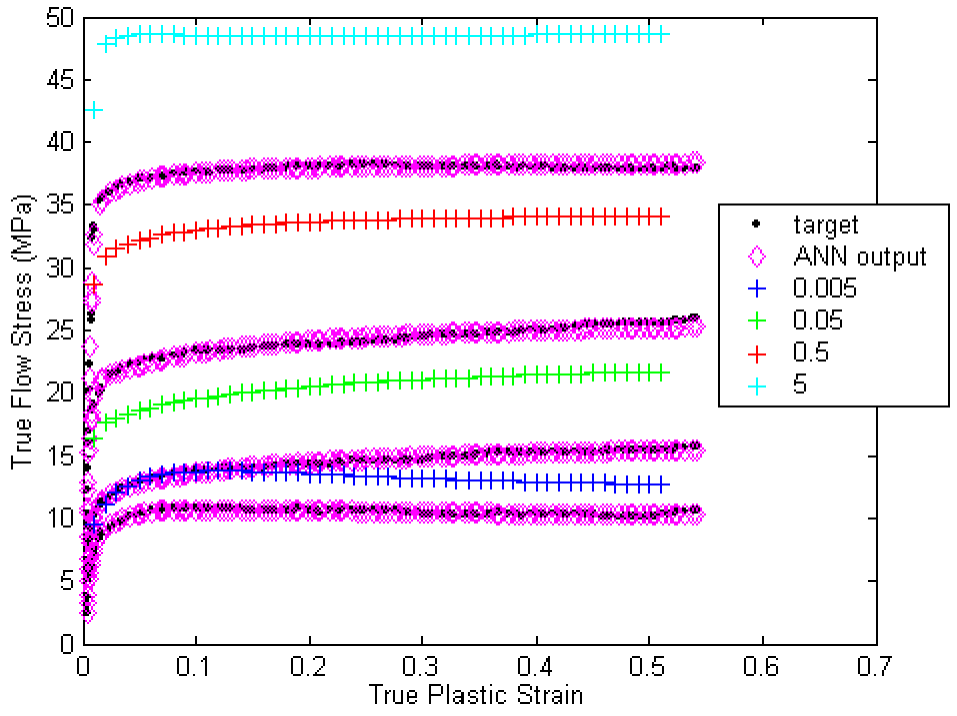


FIGURE 81. 8-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate.

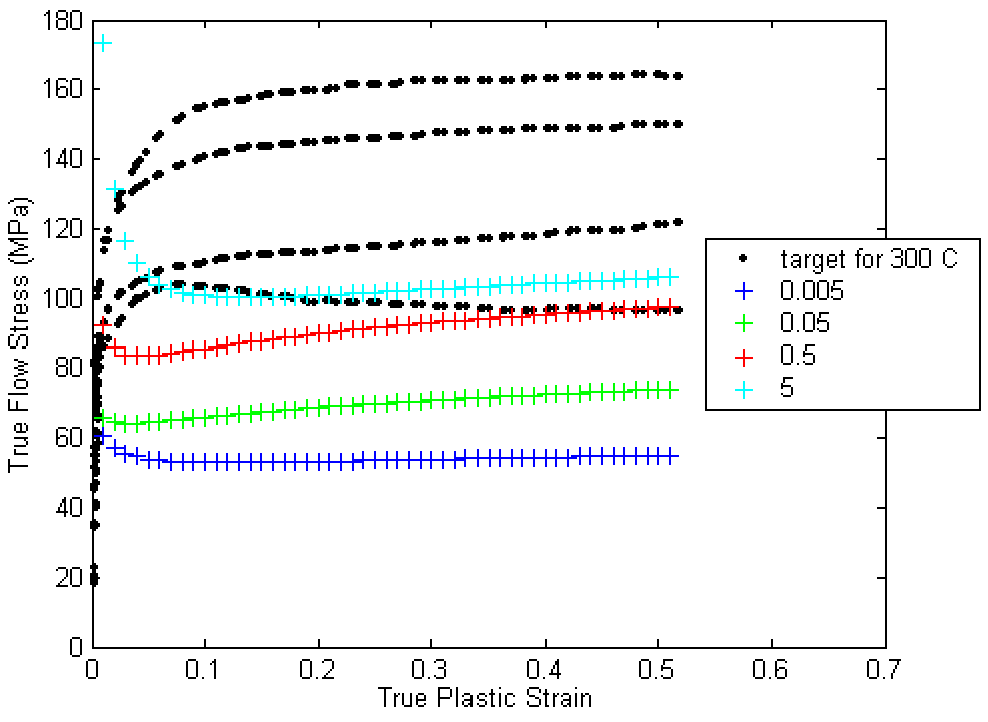


FIGURE 82. 8-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 375ºC.

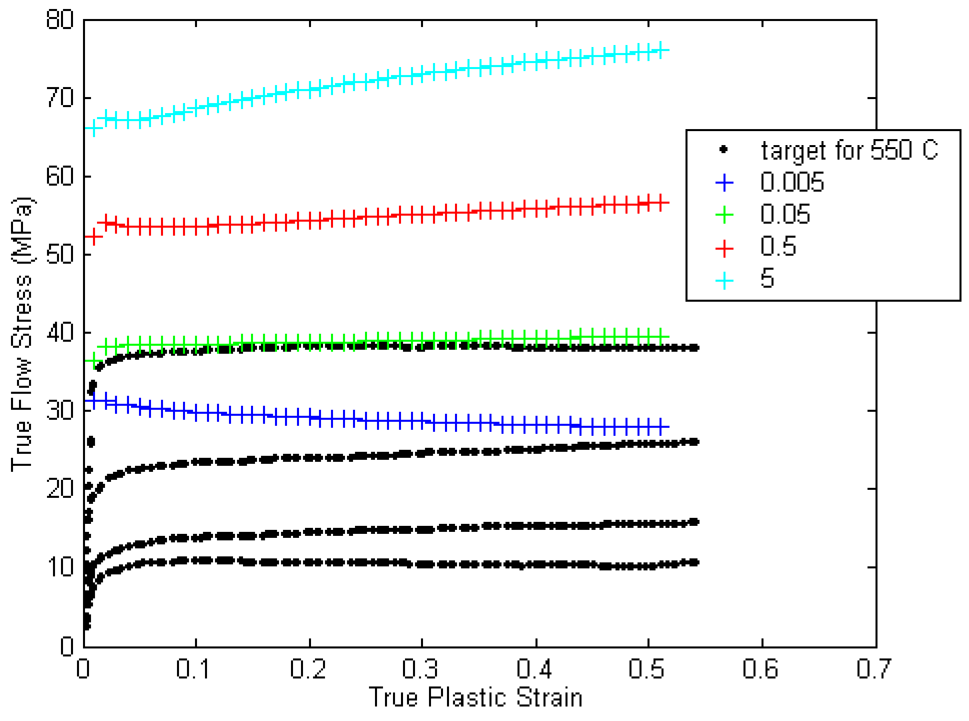


FIGURE 83. 8-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 450ºC.

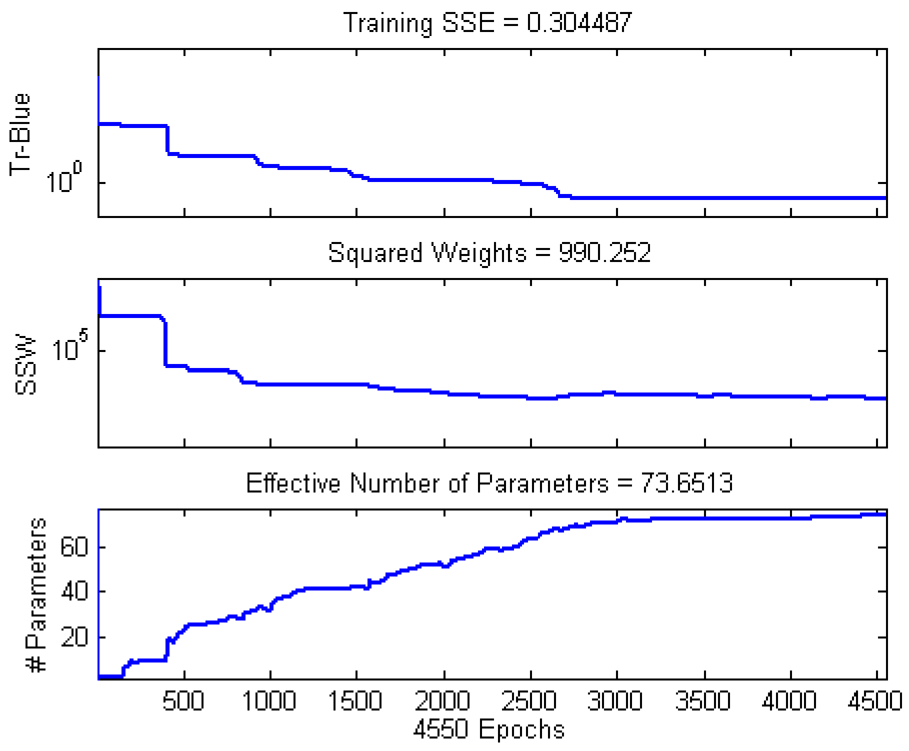


FIGURE 84. Progression of training using normalized values of ln(strain), ln(strain rate), 1/temperature, ln(stress) with 10-3 neurons, BR training algorithm.

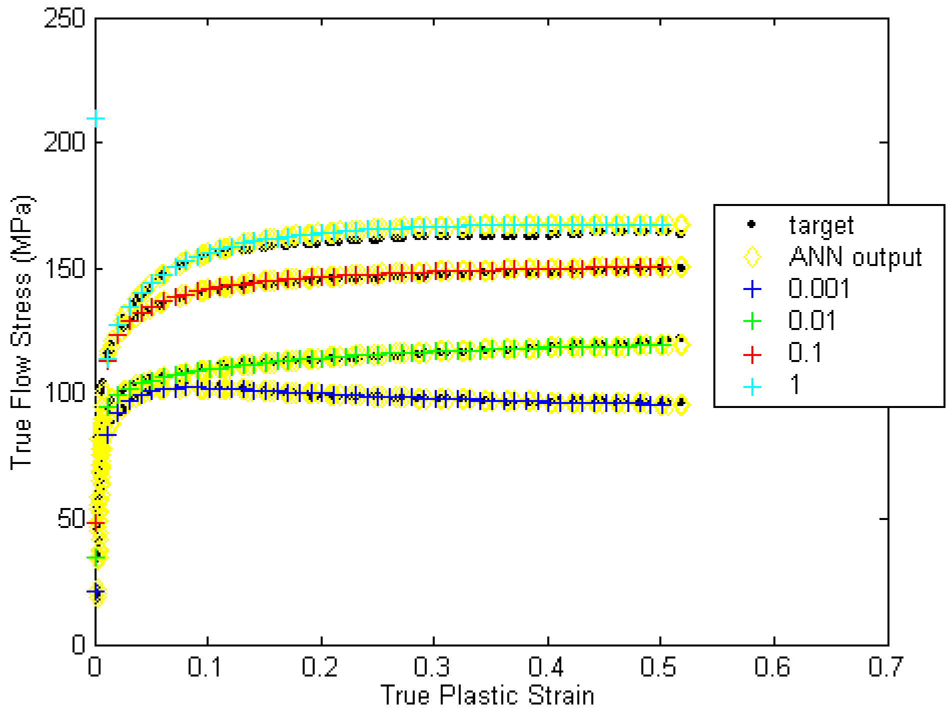


FIGURE 85. 10-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC.

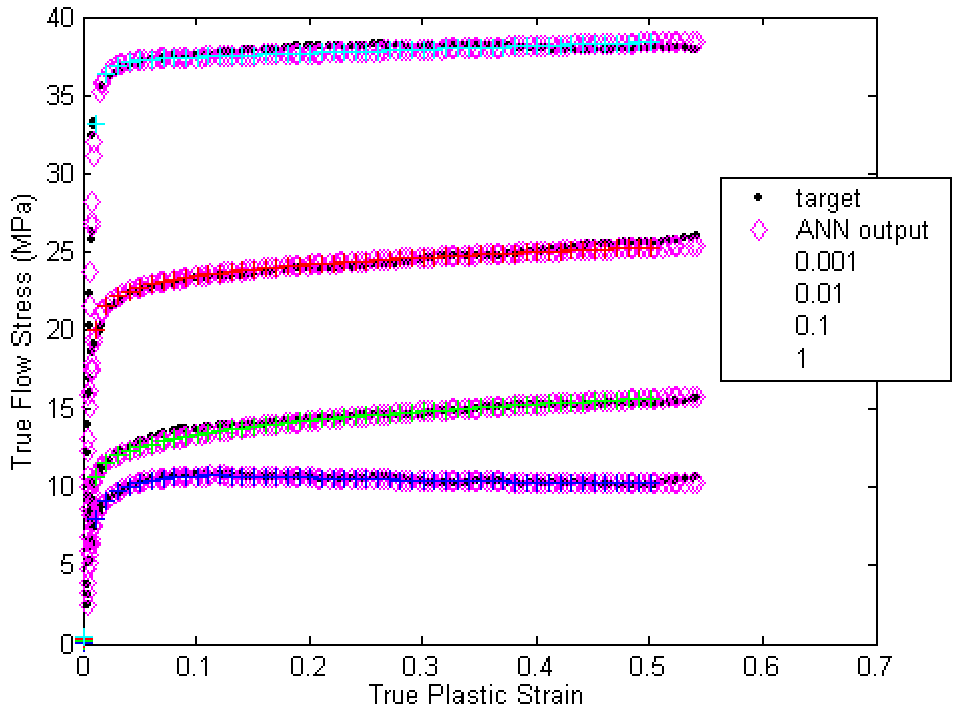


FIGURE 86. 10-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC.

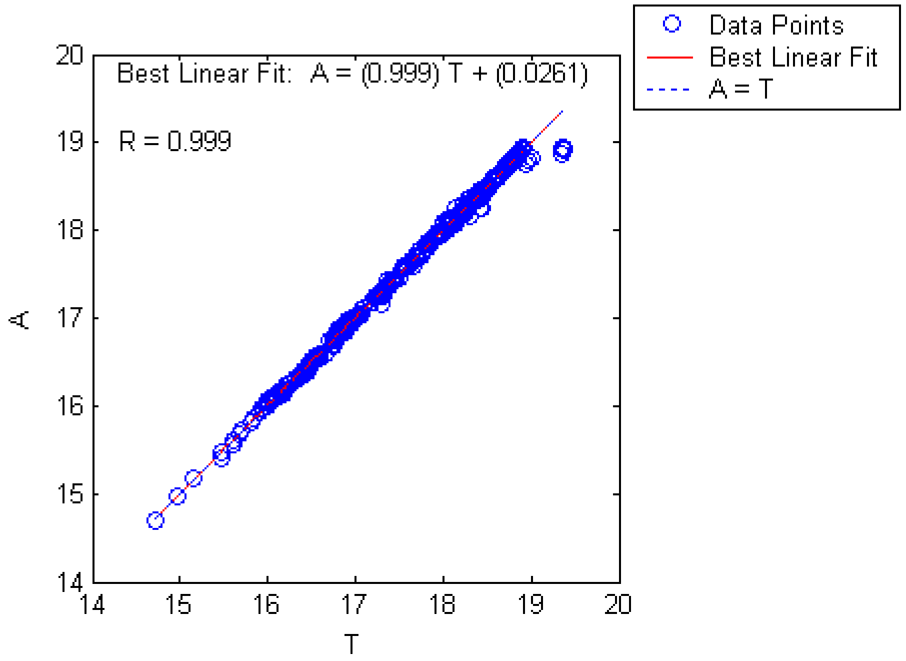


FIGURE 87. Linear regression using ln(strain), ln strain rate, reciprocal temperature, ln stress for 10-3 neurons, BR trained network, with normalization showing network output (A) versus targets (T).

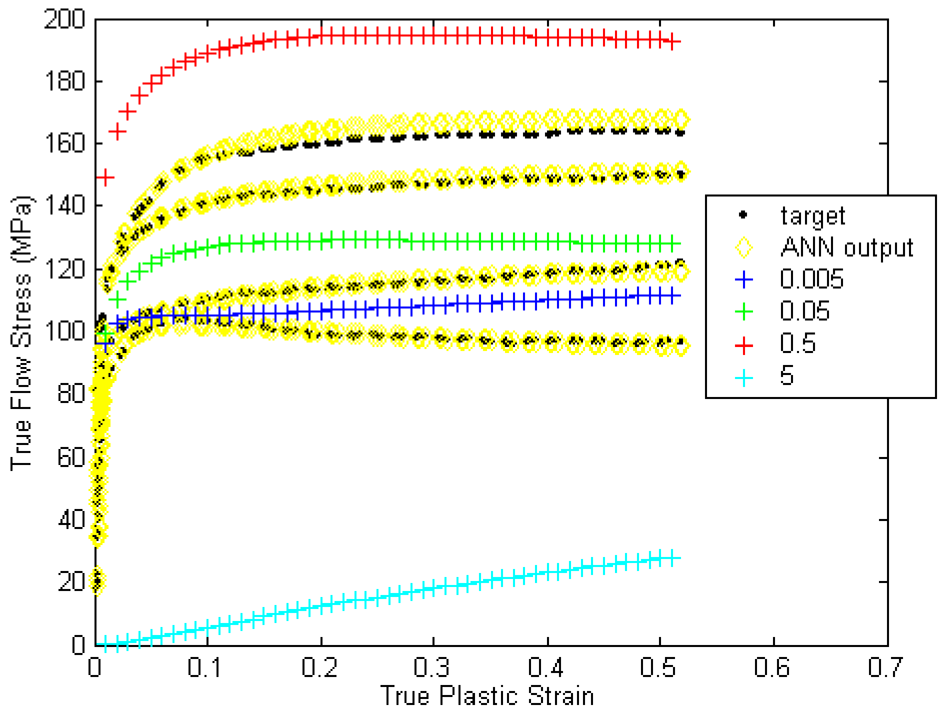


FIGURE 88. 10-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC at intermediate values of strain rate.

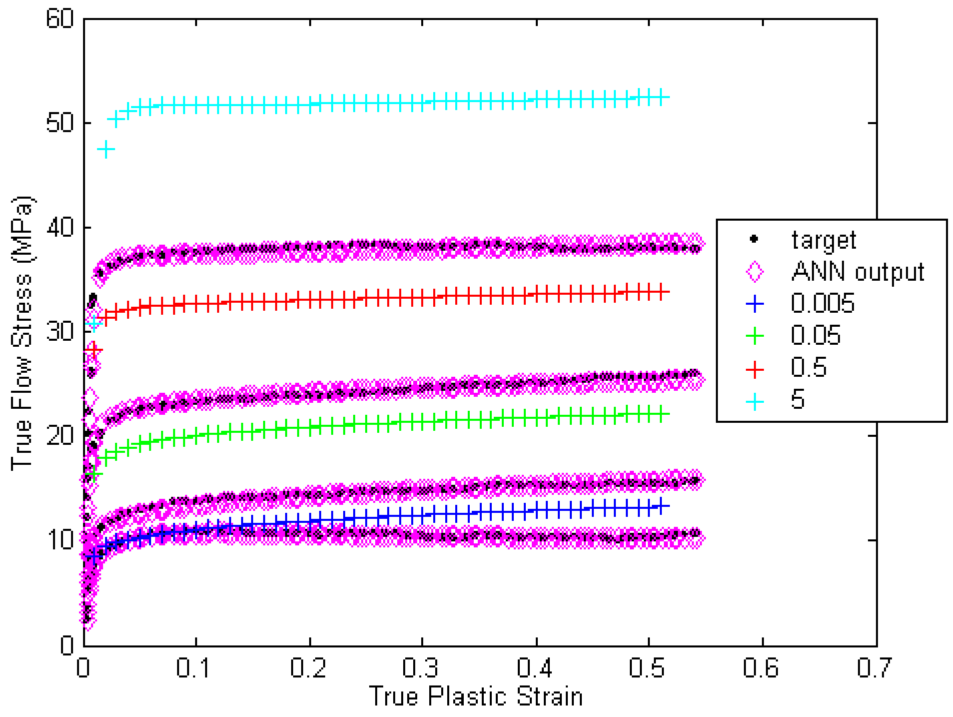


FIGURE 89. 10-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate.

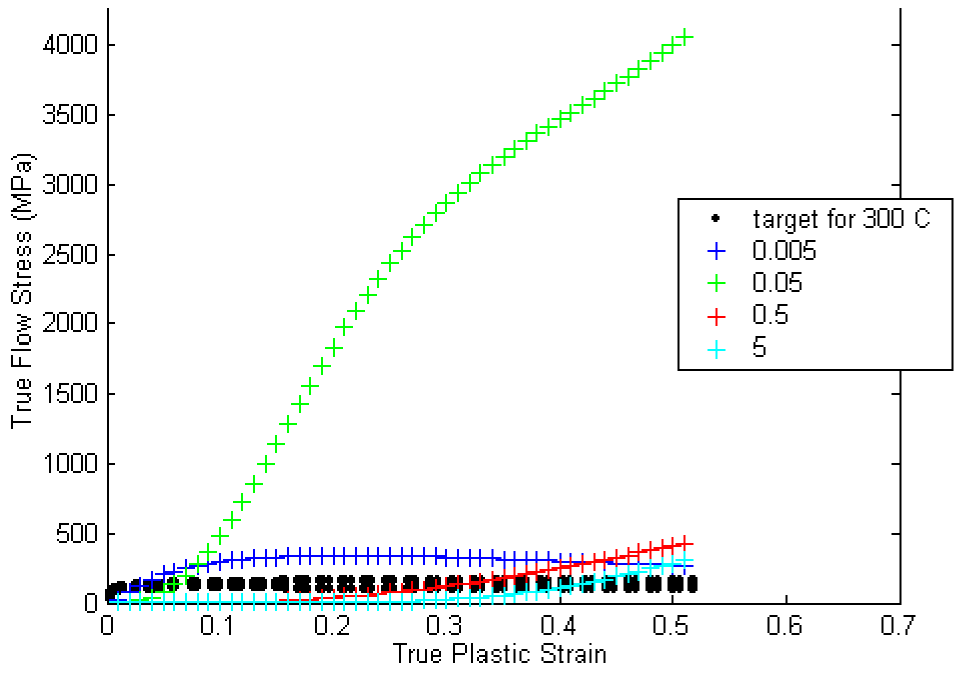


FIGURE 90. 10-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 375ºC.

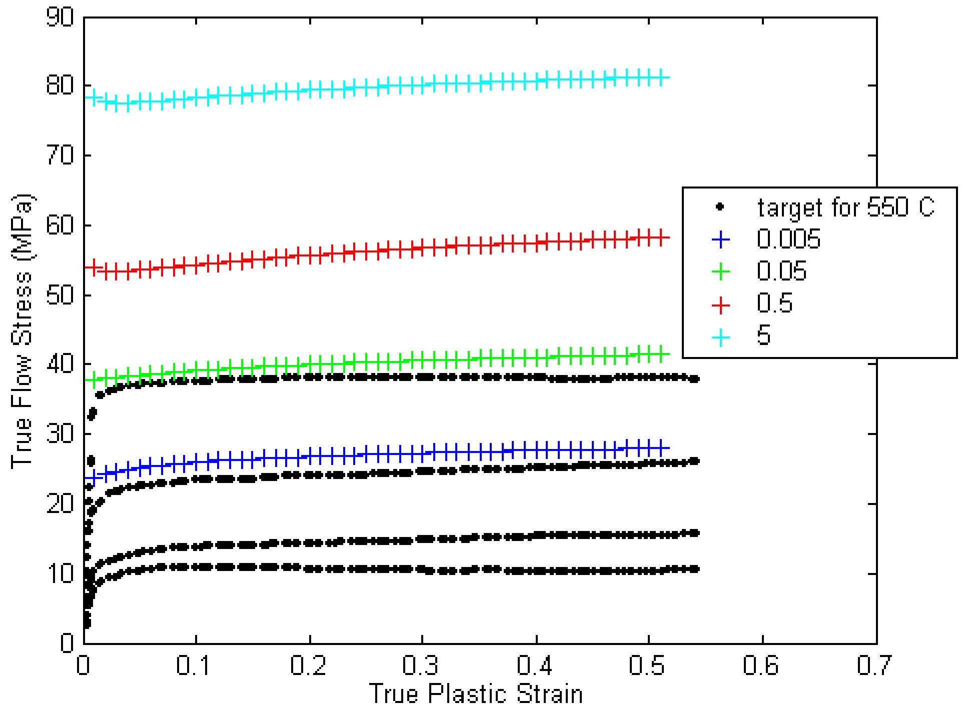


FIGURE 91. 10-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 450ºC.

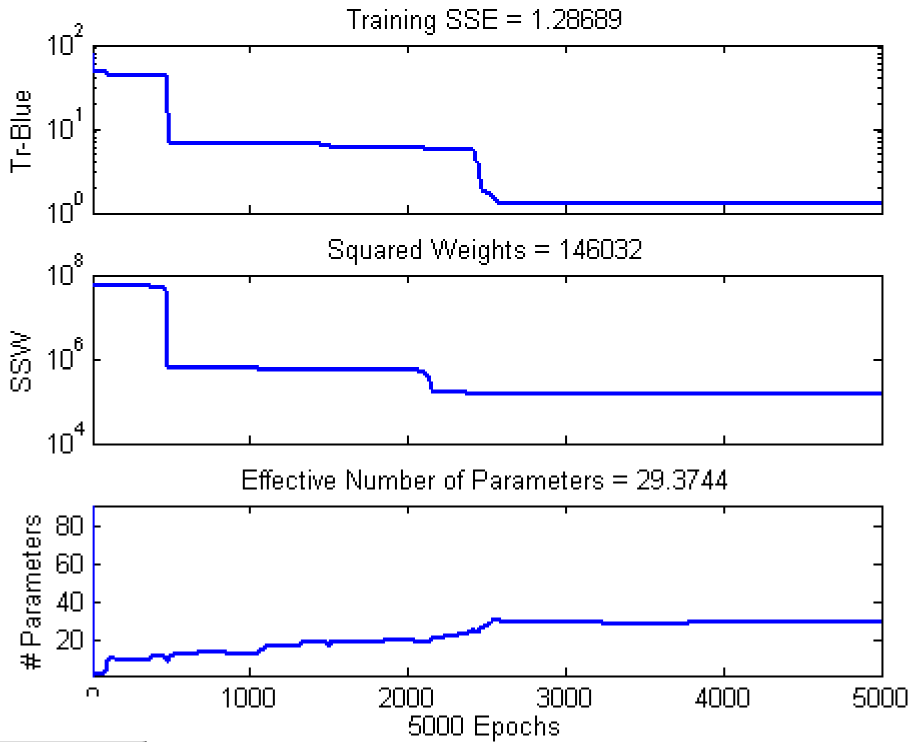


FIGURE 92. Progression of training using normalized values of ln(strain), ln(strain rate), 1/temperature, ln(stress) with 12-3 neurons, BR training algorithm.

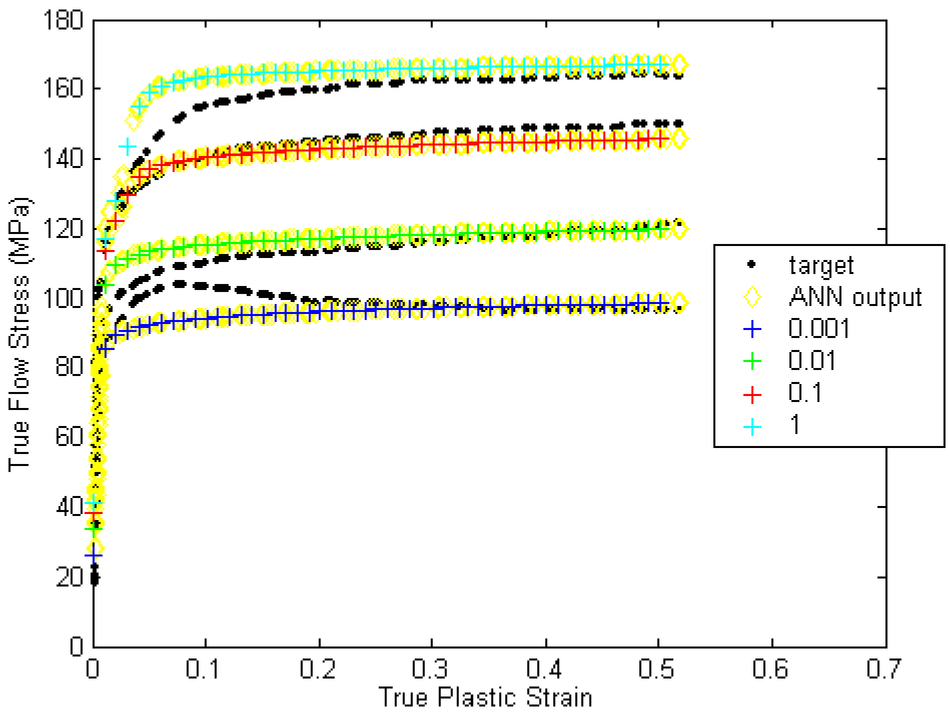


FIGURE 93. 12-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC.

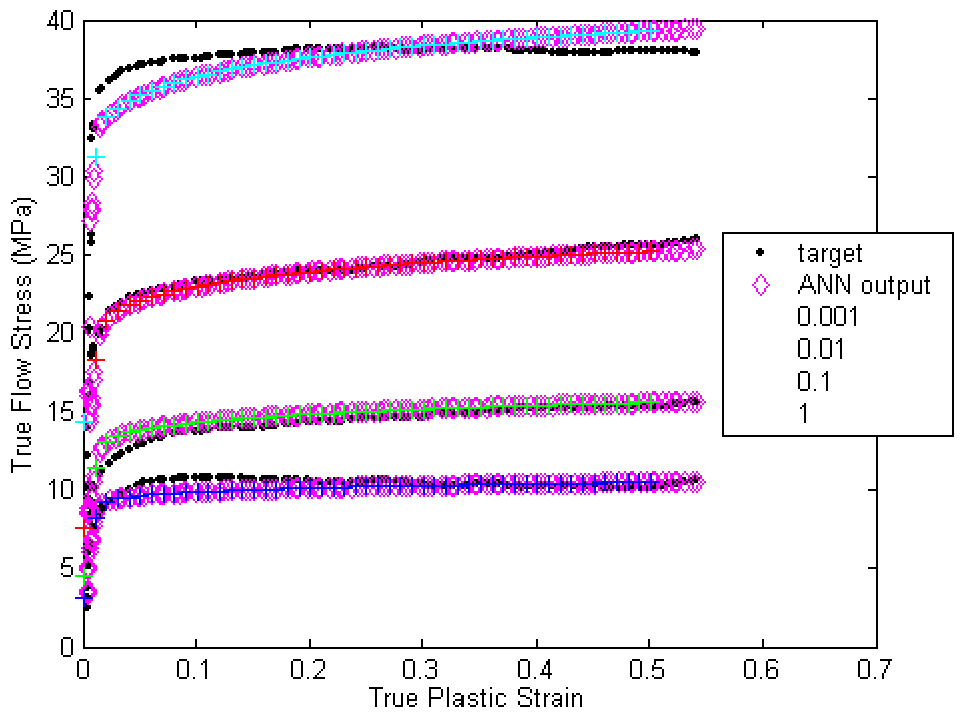


FIGURE 94. 12-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC.

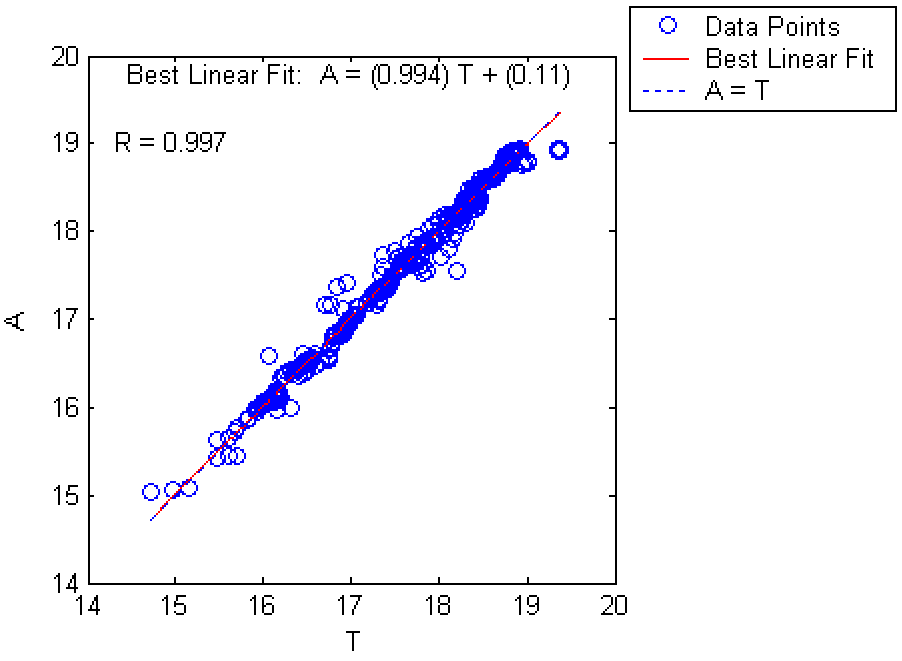


FIGURE 95. Linear regression using ln(strain), ln strain rate, reciprocal temperature, ln stress for 12-3 neurons, BR trained network, with normalization showing network output (A) versus targets (T).

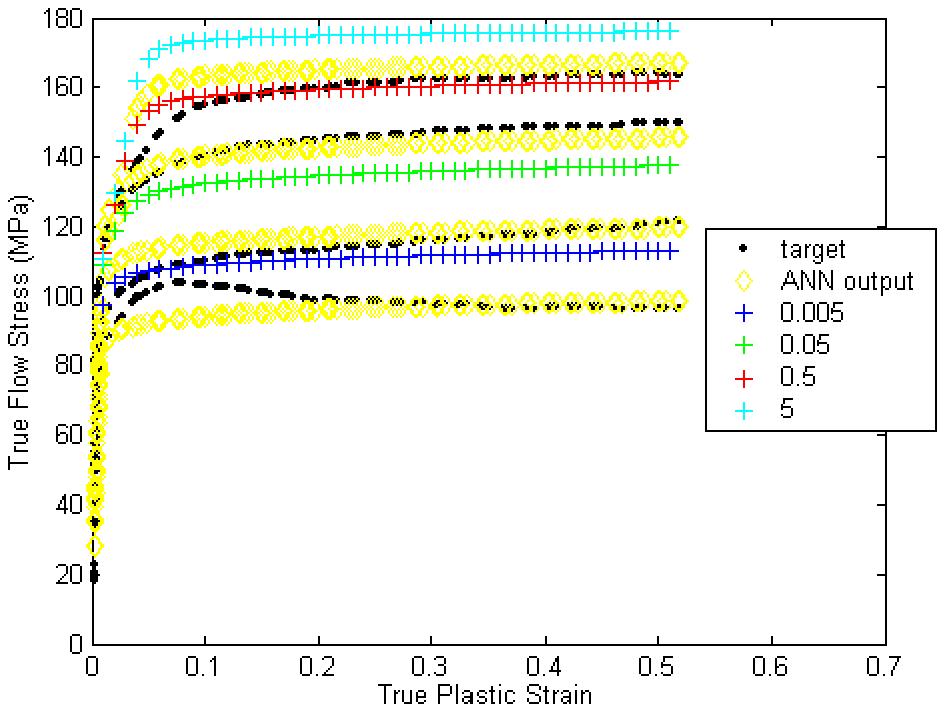


FIGURE 96. 12-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC at intermediate values of strain rate.

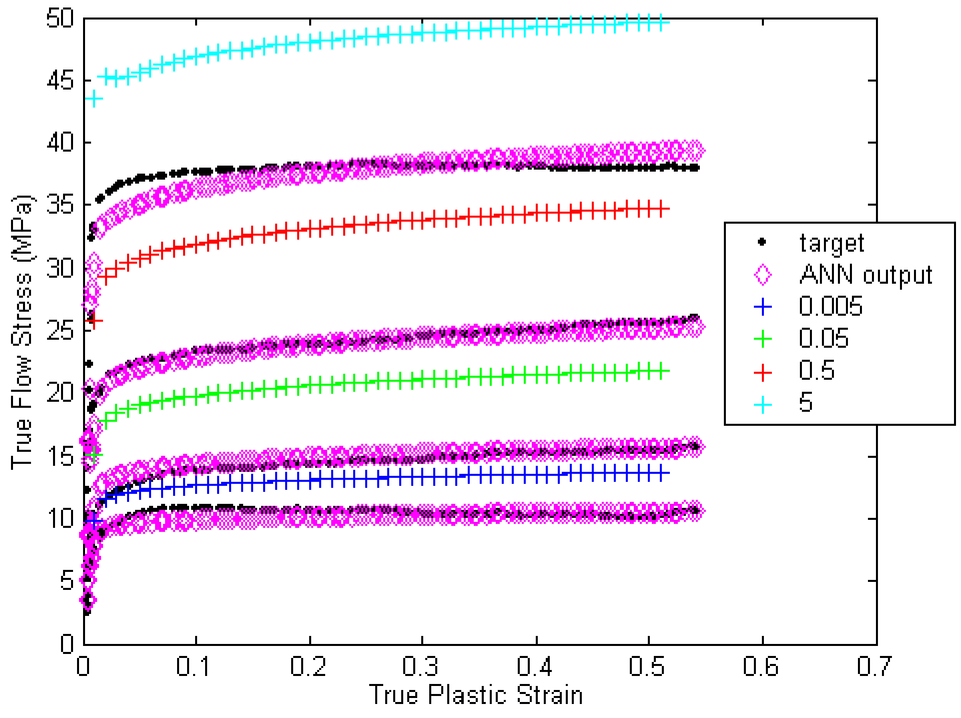


FIGURE 97. 12-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate.

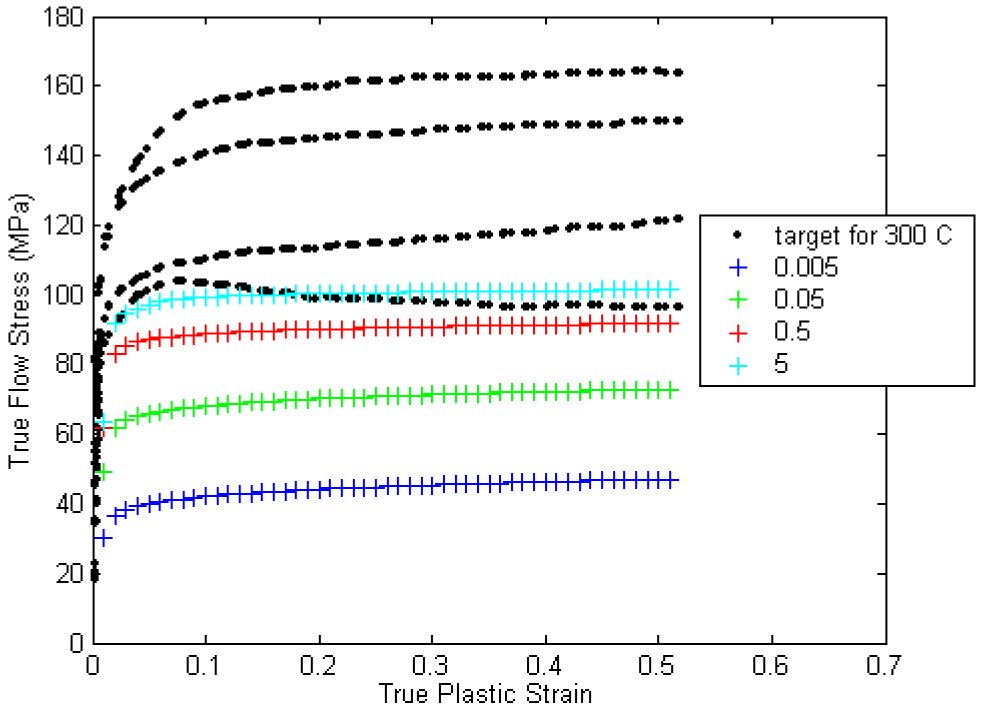


FIGURE 98. 12-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 375ºC.

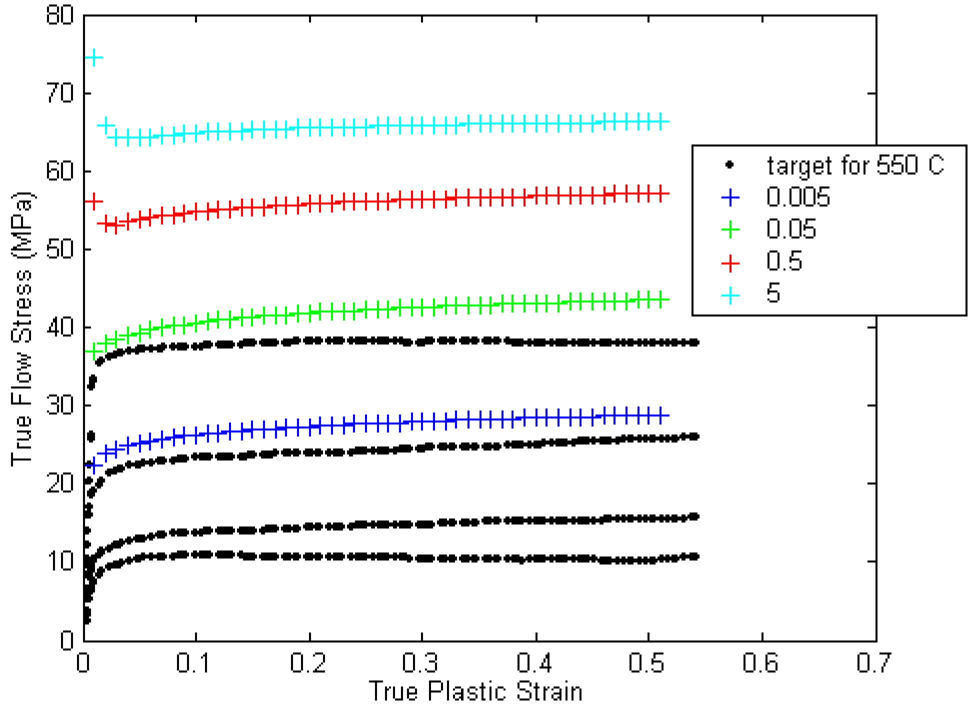


FIGURE 99. 12-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 450ºC.

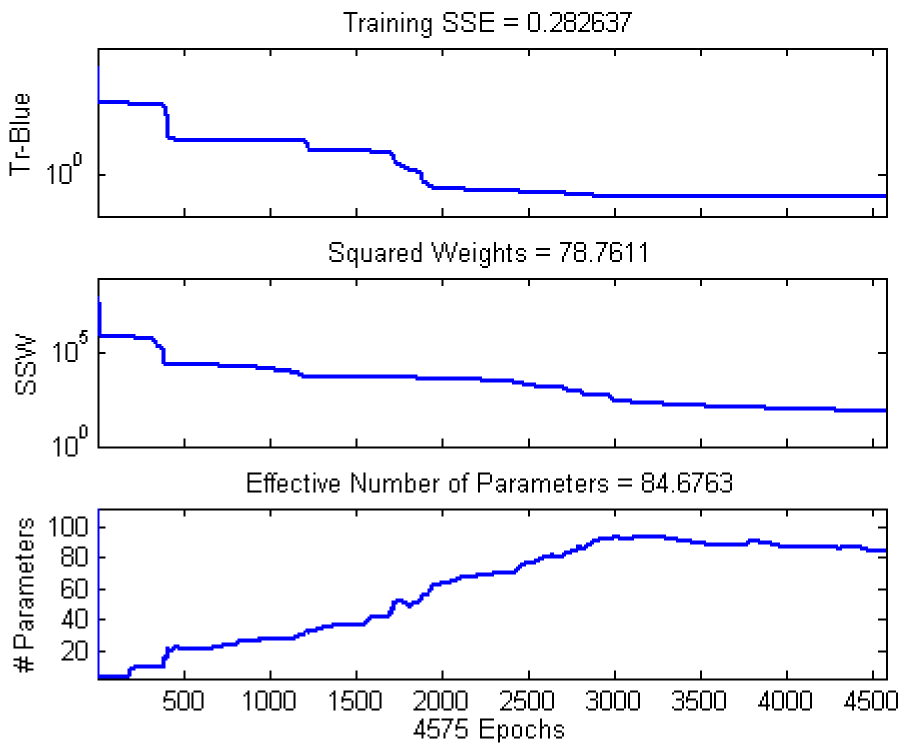


FIGURE 100. Progression of training using normalized values of ln(strain), ln(strain rate), 1/temperature, ln(stress) with 15-3 neurons, BR training algorithm.

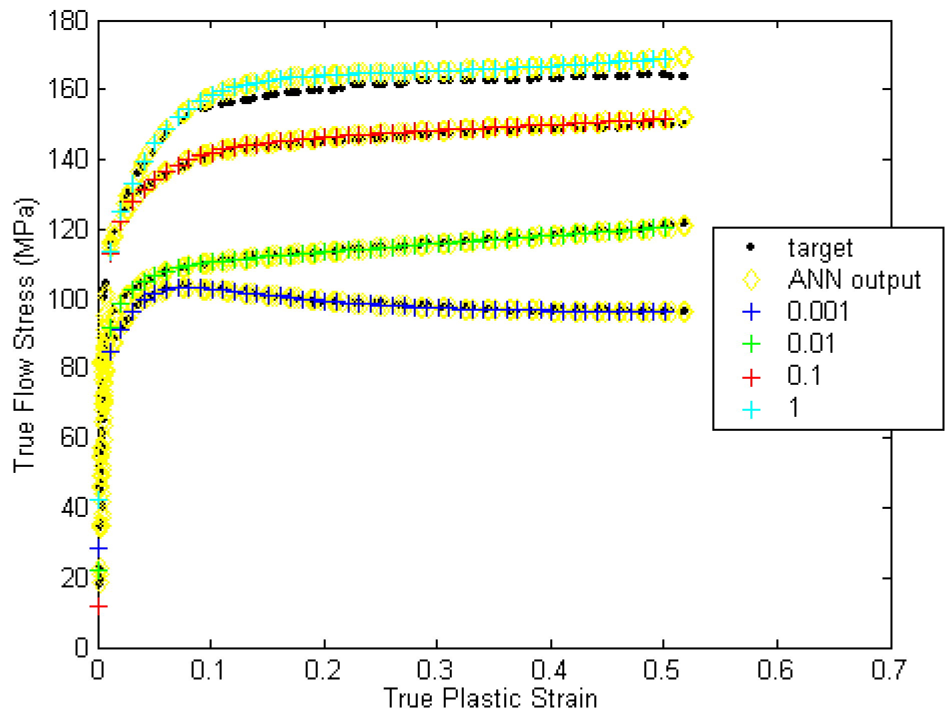


FIGURE 101. 15-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC.

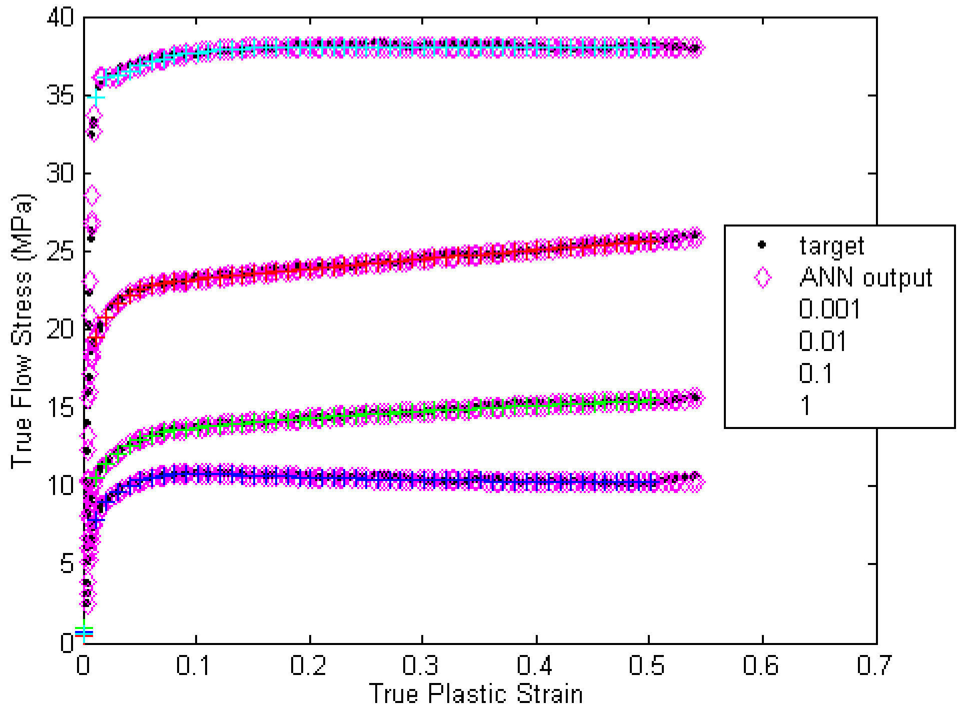


FIGURE 102. 15-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC.

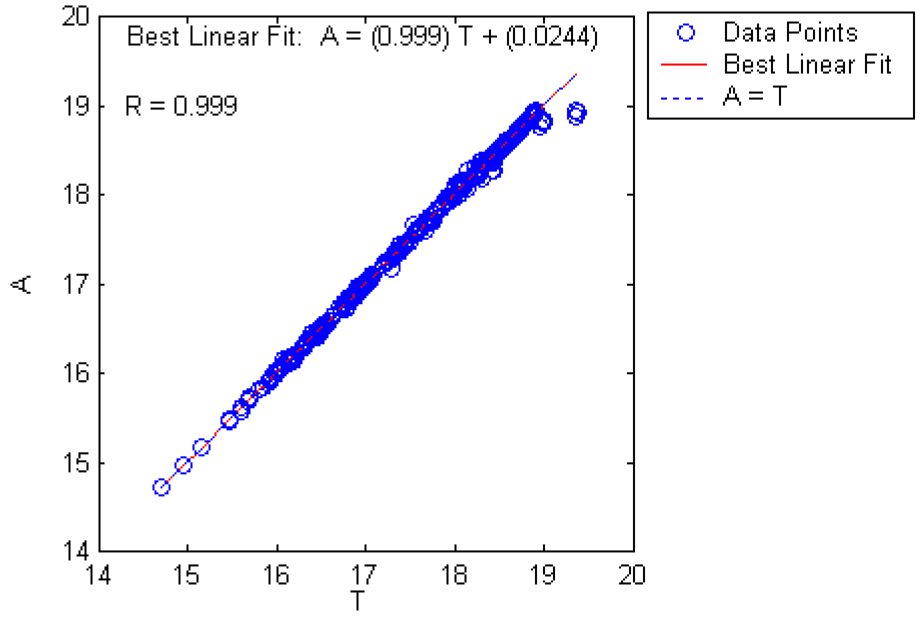


FIGURE 103. Linear regression using ln(strain), ln strain rate, reciprocal temperature, ln stress for 15-3 neurons, BR trained network, with normalization showing network output (A) versus targets (T).

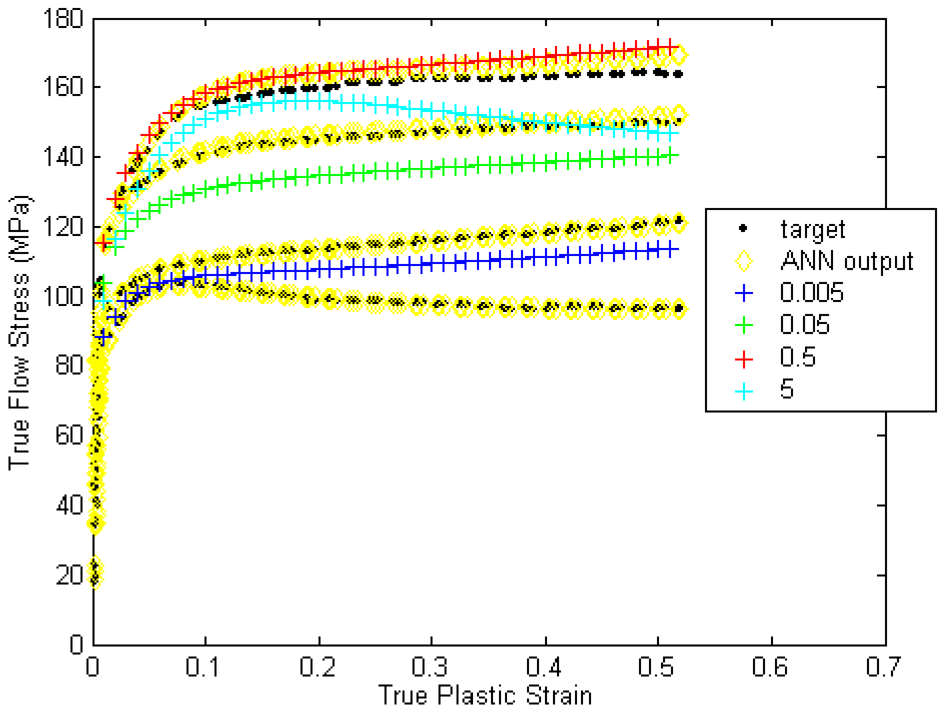


FIGURE 104. 15-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 300ºC at intermediate values of strain rate.

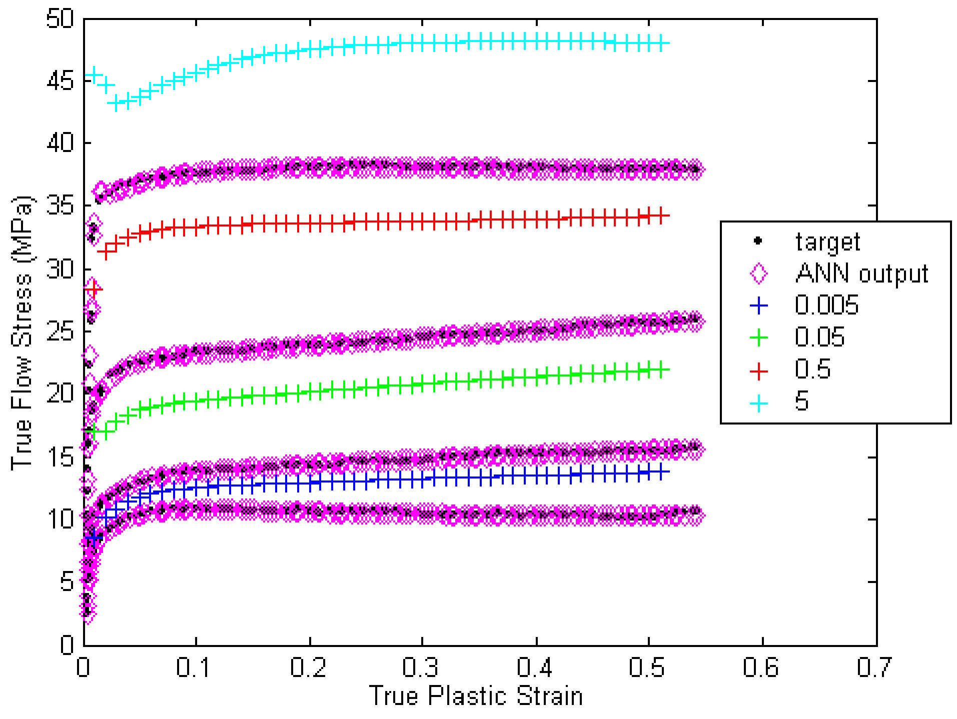


FIGURE 105. 15-3 BR Network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate.

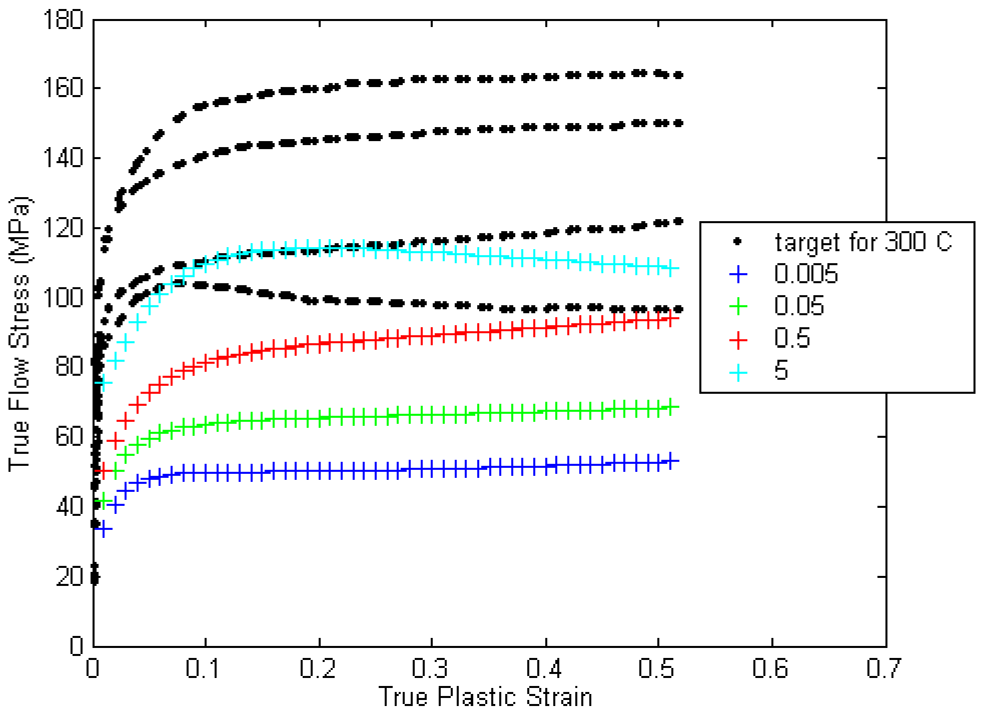


FIGURE 106. 15-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 375ºC.

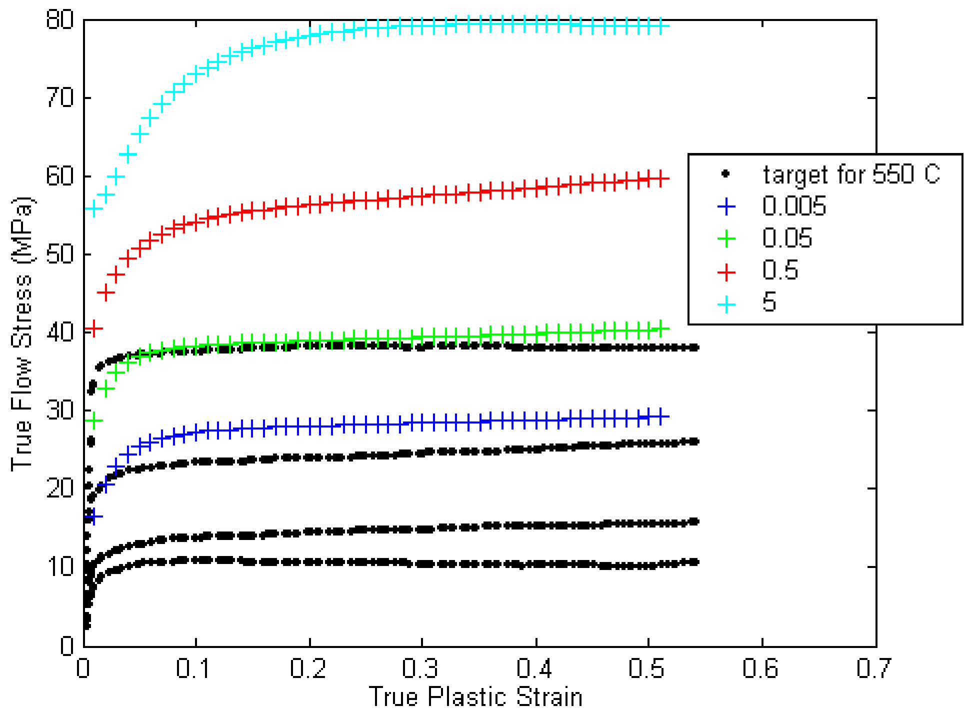


FIGURE 107. 15-3 BR network using normalized ln(strain), ln(strain rate), 1/temperature, ln(stress) for 550ºC at intermediate values of strain rate and a temperature of 450ºC.

Final Aluminum Neural Network Architecture

The 15-3 BR network provides excellent matching of the experimental data when queried with the training inputs. It also correctly mimics experimental data when presented with varying strains (those not used in training) at strain rates and temperatures within the training set. Without further experimentation, the network performance at intermediate strain rates and temperature cannot be verified completely, but the response appears reasonable based on the data available.

5.2 Nickel Aluminide Neural Network Development

For trial purposes, the data for a single temperature, 1100°C, was used as the training set for a single layer feed forward neural network. By limiting the training to two inputs, training could be accomplished much faster. The two input values for strain and strain rate feed 8 tan-sig neurons with bias. Figure 108 through Figure 110 show the relevant plots of training progression and network output. The network output mirrors the training set to a high degree.

Using the experience gained from developing a network architecture for aluminum, a similar trial table was constructed and the networks queried. In this case, the training data consisted of natural log of strain and strain rate, the reciprocal of the temperature, and the natural log of the flow stress. For each trial, the data was normalized before training using Bayesian regularization. Table 9 shows the trials attempted with pertinent results.

As can be seen below, the 20-3 BR network does not properly reflect anticipated performance at intermediate strain rate values. It could be argued that it is asking too much for the network to produce reasonable values for a strain rate of 50 /s, but even if these values were disregarded, the network still proves unsatisfactory. This lead to training 30-3 and 25-3 networks with the results shown below.

TABLE 9. Network training trials for nickel aluminide.

Data	Number of Neurons	Epochs Completed	Sum Squared Error	Effective Parameters	R	Relevant Figures
1100ºC only	8	298	0.5046	21.01/33	0.999	108, 109, 110
1100ºC only	10	500	0.0926	39.63/41	1.000	111, 112, 113
1100, 1175, 1250ºC	20-3	5000	0.0358	57.05/147	1.000	114, 115, 116, 117, 118, 119, 120, 121
1100, 1175, 1250ºC	30-3	5000	0.0330	158.9/217	1.000	122, 123, 124, 125, 126, 127, 128, 129
1100, 1175, 1250ºC	25-3	5000	0.0113	176.4/182	1.000	130, 131, 132, 133, 134

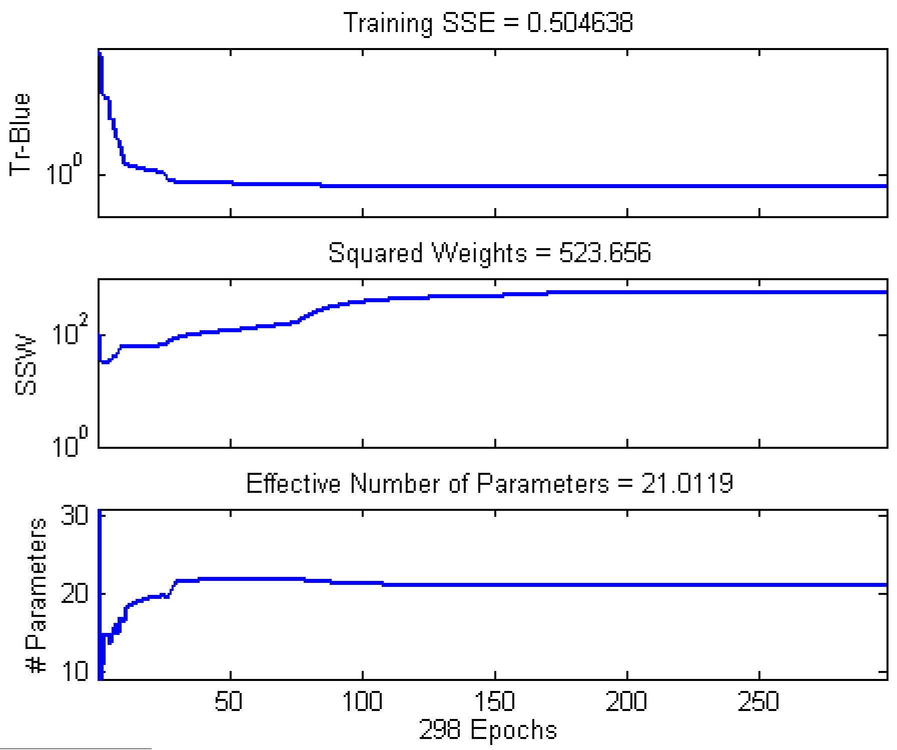


FIGURE 108. Progression of training using fully conditioned data, 8 neurons, BR training algorithm.

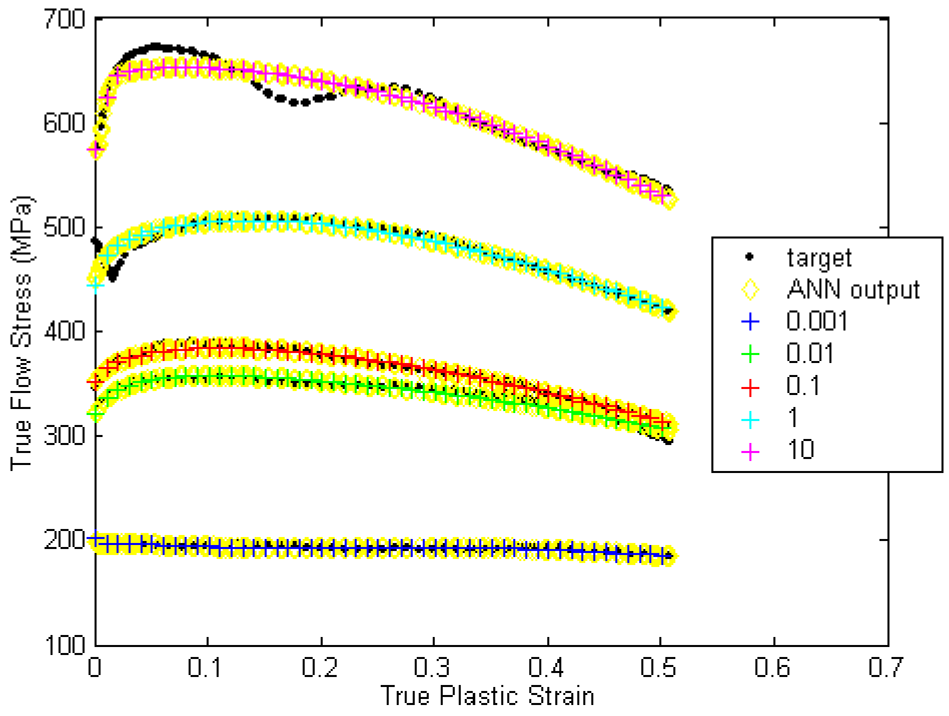


FIGURE 109. Network performance, 8 neurons, BR training, for 1100ºC.

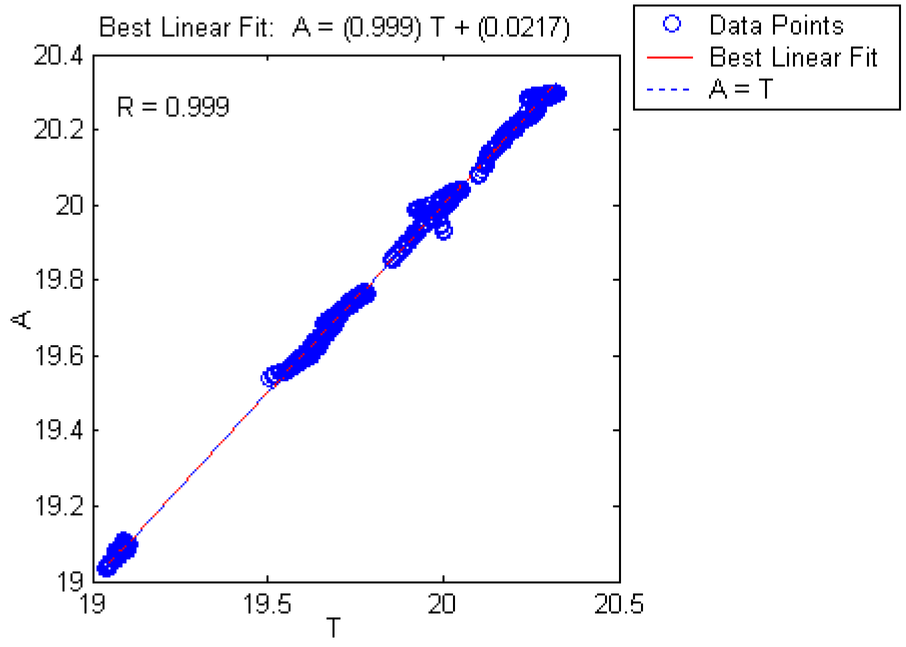


FIGURE 110. Linear regression for the

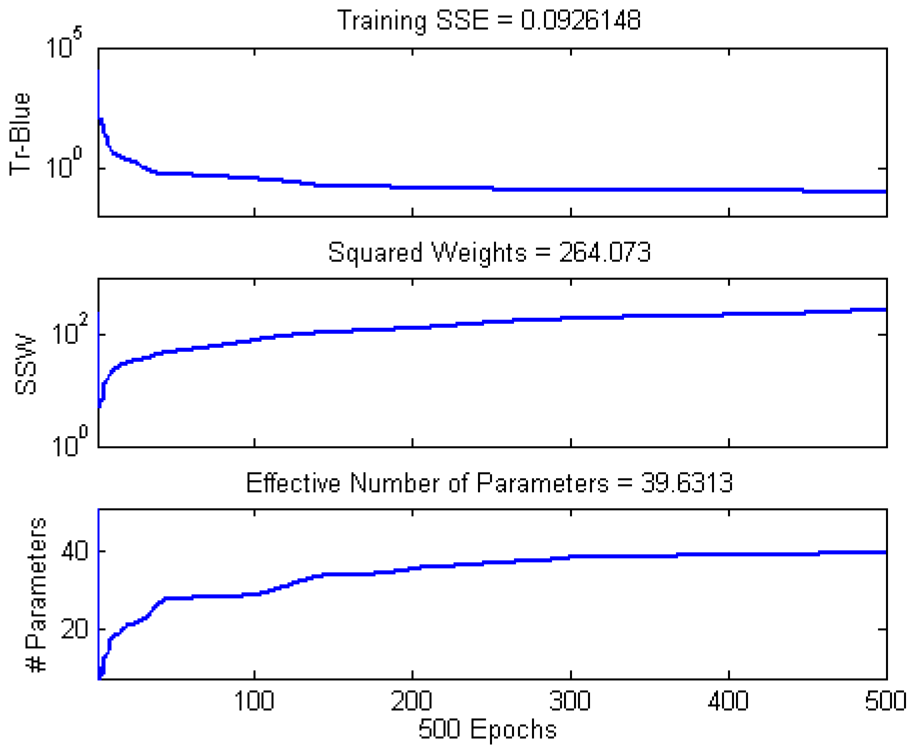


FIGURE 111. Progression of training using fully conditioned data, 10 neurons, BR training algorithm.

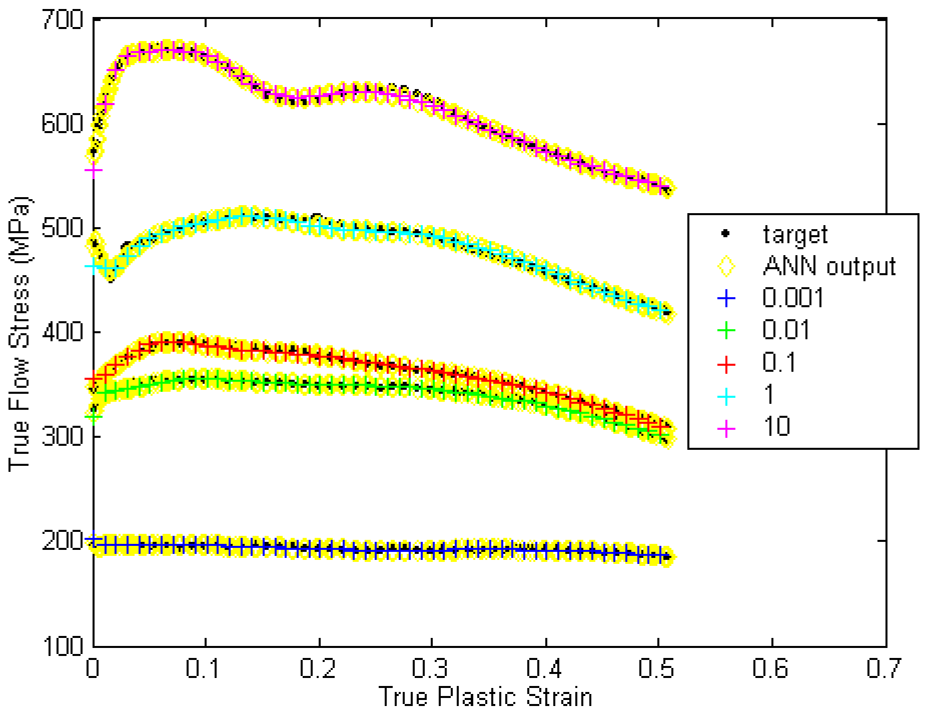


FIGURE 112. Network performance, 10 neurons, BR training, for 1100ºC.

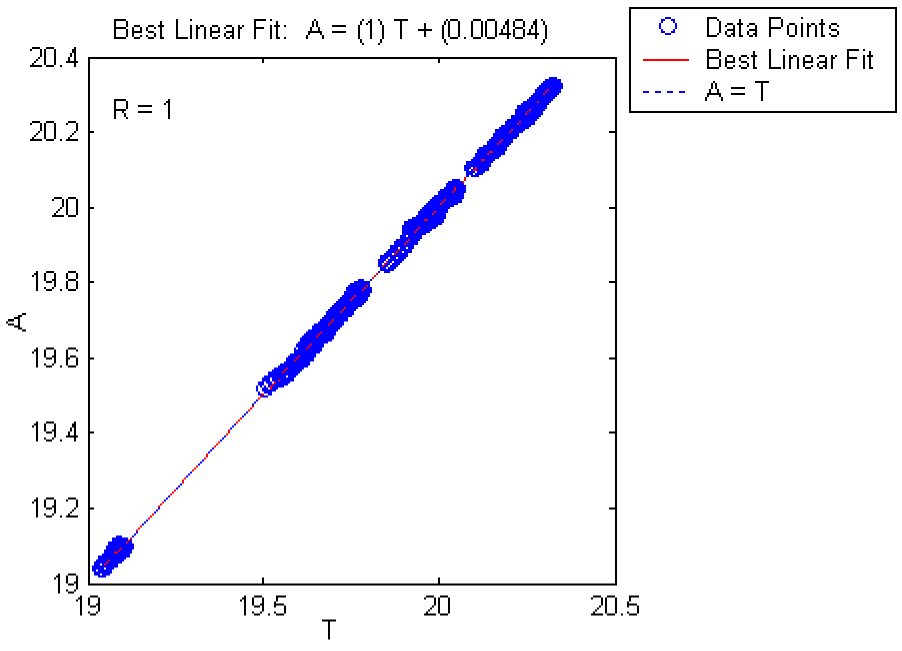


FIGURE 113. Linear regression for the 10 neurons, BR trained network for 1100ºC.

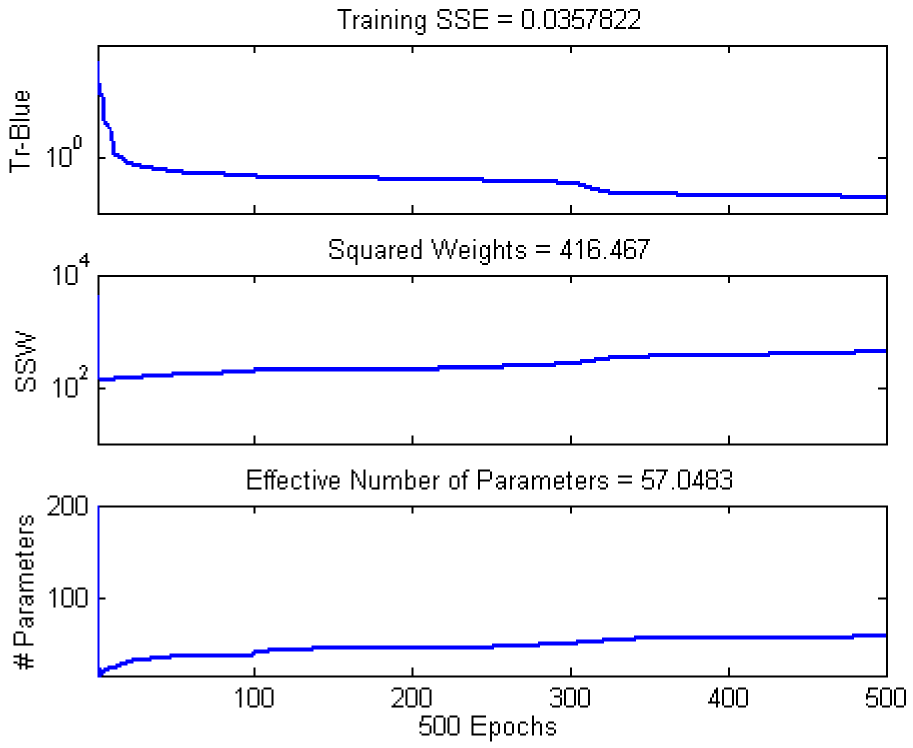


FIGURE 114. Progression of training using fully conditioned data for 1100, 1175, and 1250ºC, 20-3 neurons, BR training algorithm.

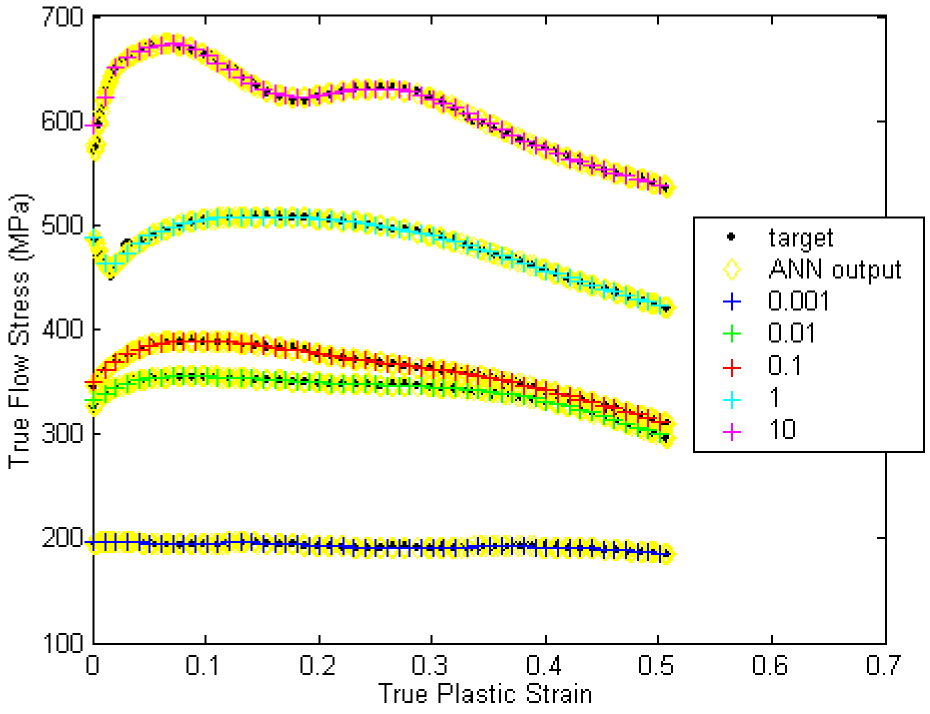


FIGURE 115. Network performance, 20-3 neurons, BR training, for 1100ºC.

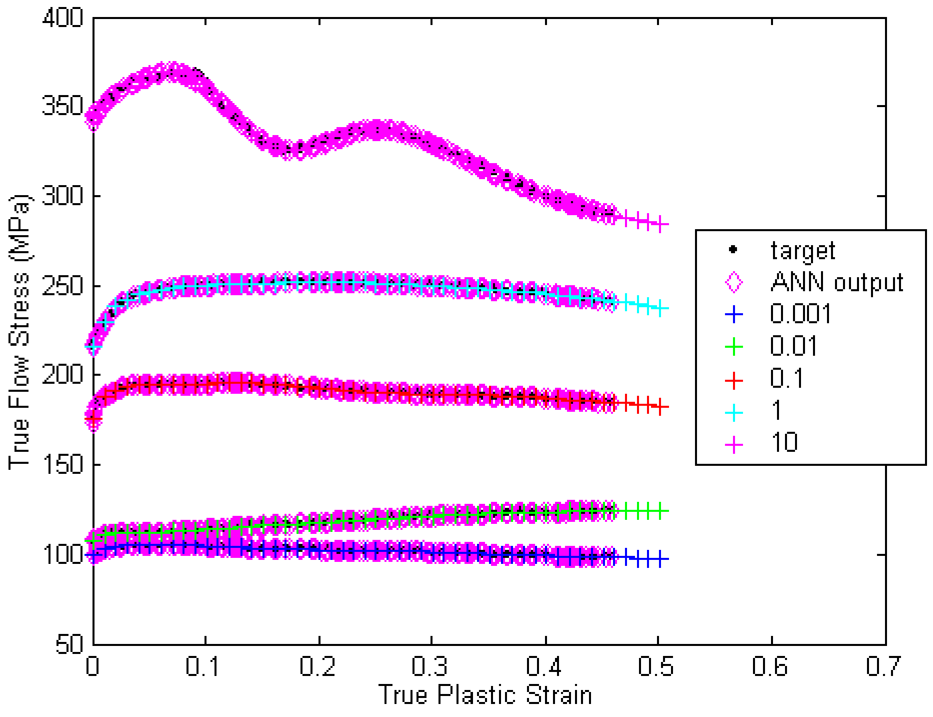


FIGURE 116. Network performance, 20-3 neurons, BR training, for 1175ºC.

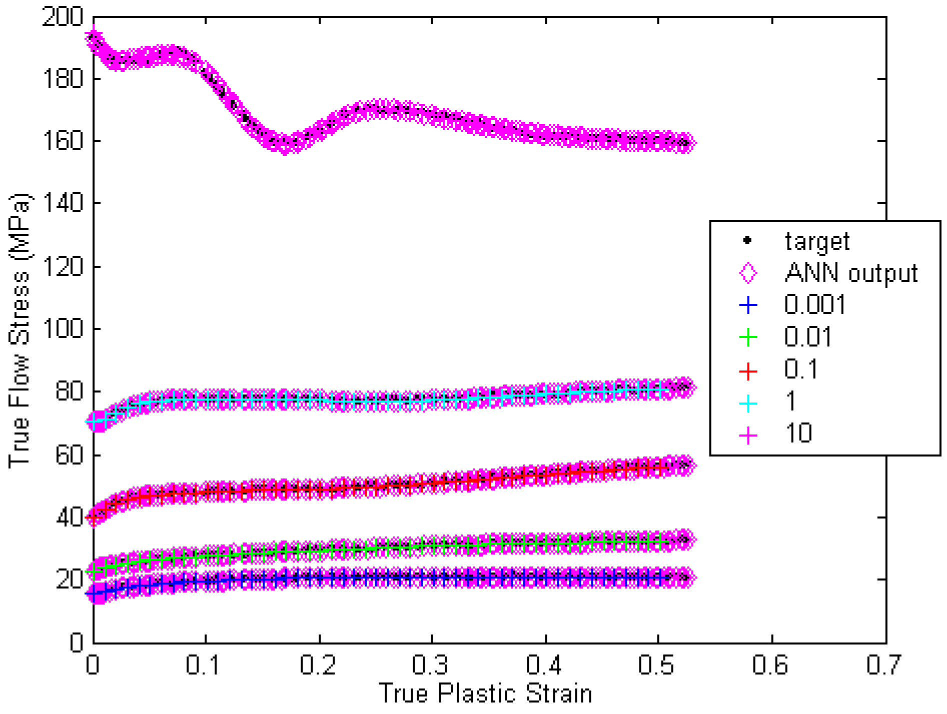


FIGURE 117. Network performance, 20-3 neurons, BR training, for 1250ºC.

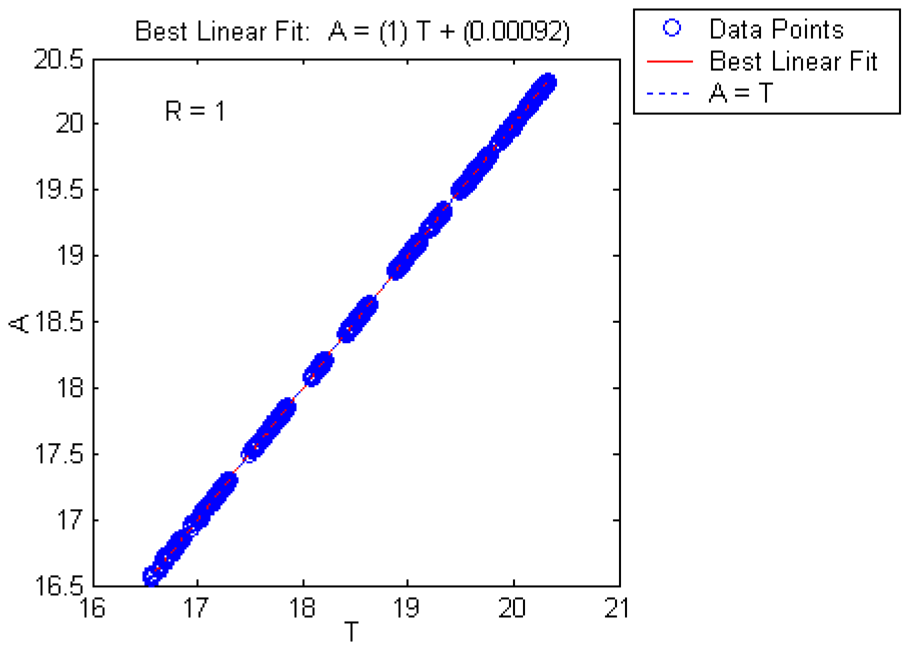


FIGURE 118. Linear regression for the 20-3 neurons, BR trained network for 1100, 1175, and 1250ºC.

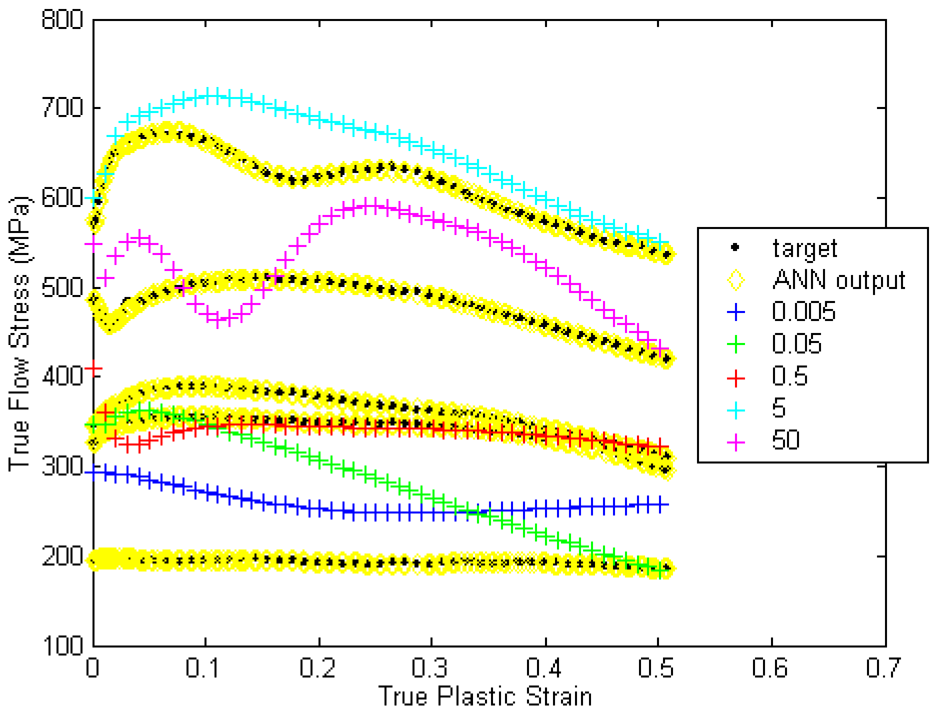


FIGURE 119. Network performance, 20-3 neurons, BR training, for 1100ºC for intermediate values of strain rate.

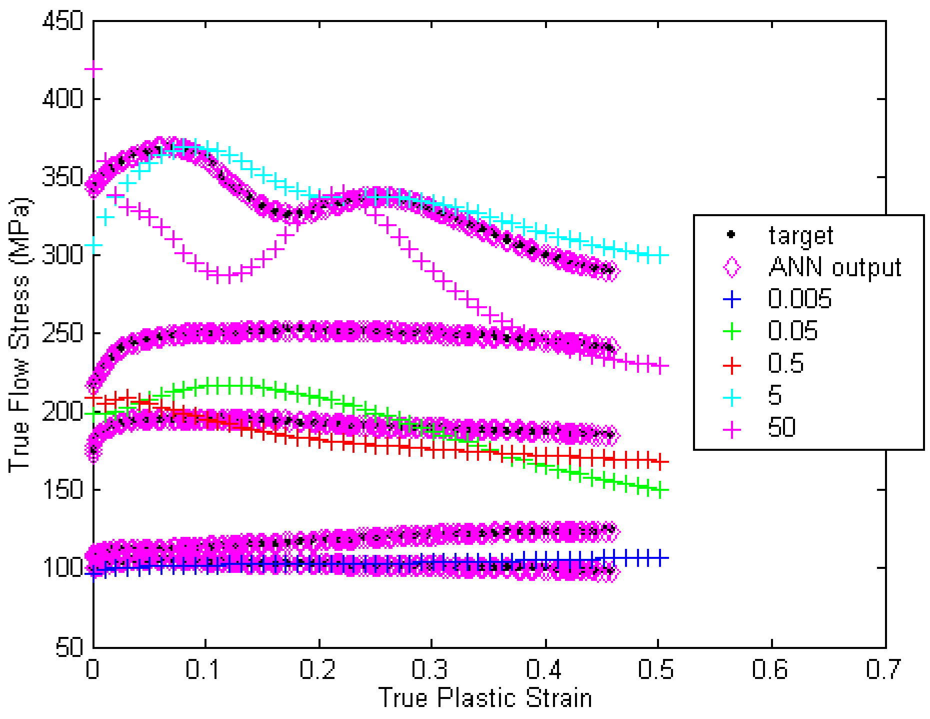


FIGURE 120. Network performance, 20-3 neurons, BR training, for 1175ºC for intermediate values of strain rate.

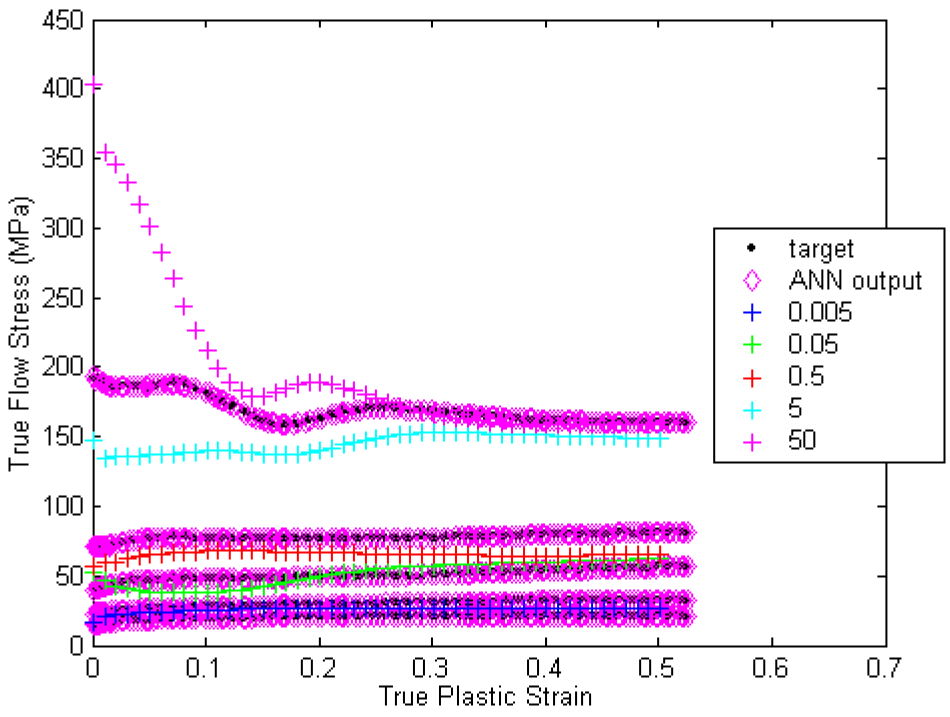


FIGURE 121. Network performance, 20-3 neurons, BR training, for 1250ºC for intermediate values of strain rate.

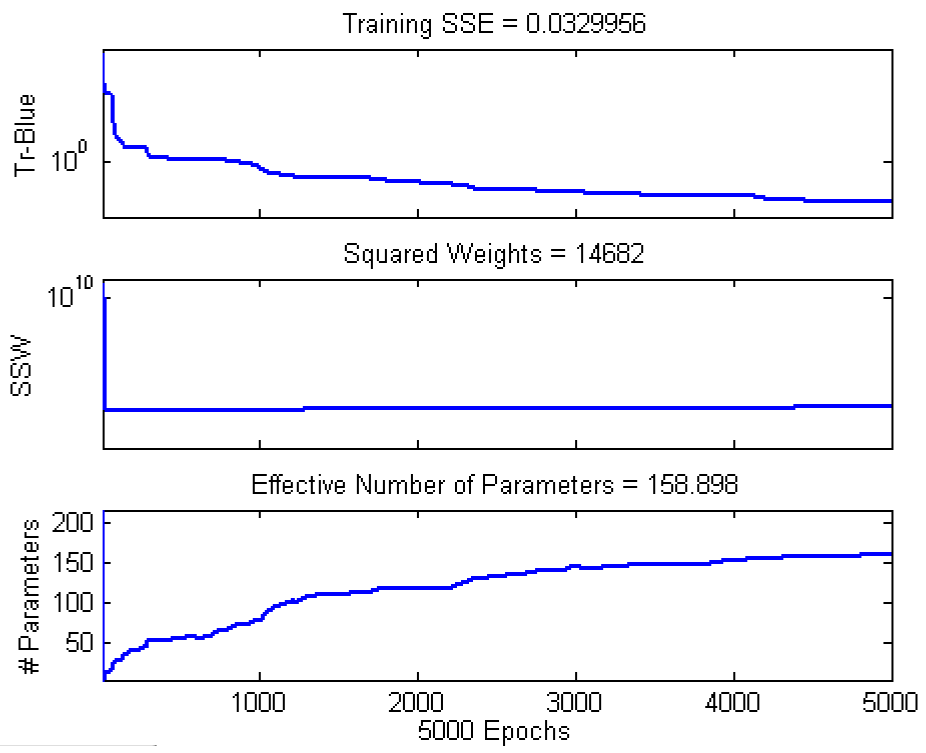


FIGURE 122. Progression of training using fully conditioned data, 30-3 neurons, BR training algorithm.

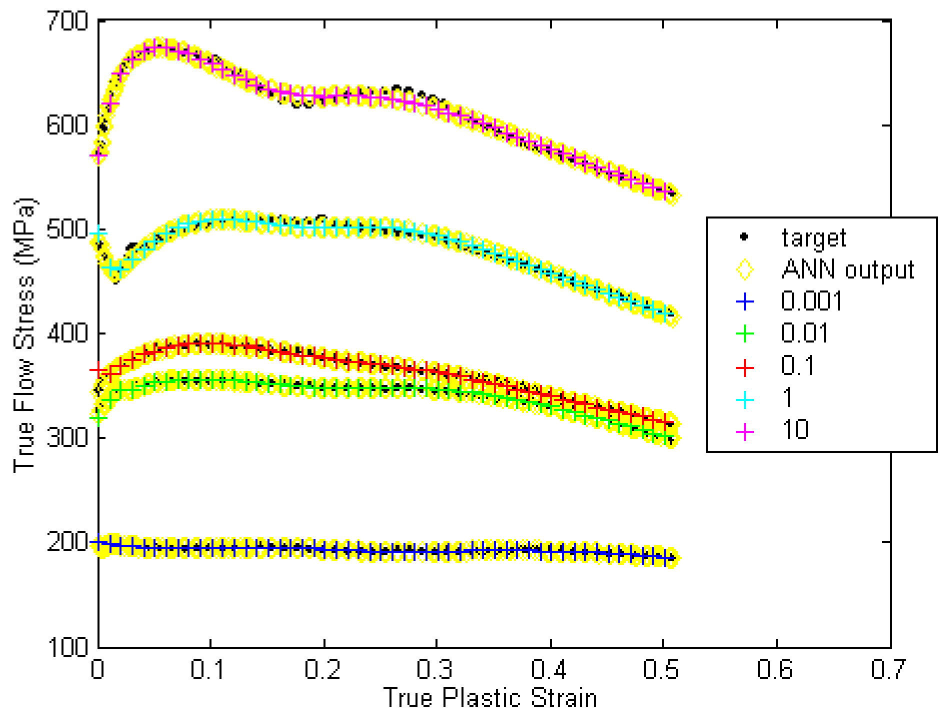


FIGURE 123. Network performance, 30-3 neurons, BR training, for 1100ºC.

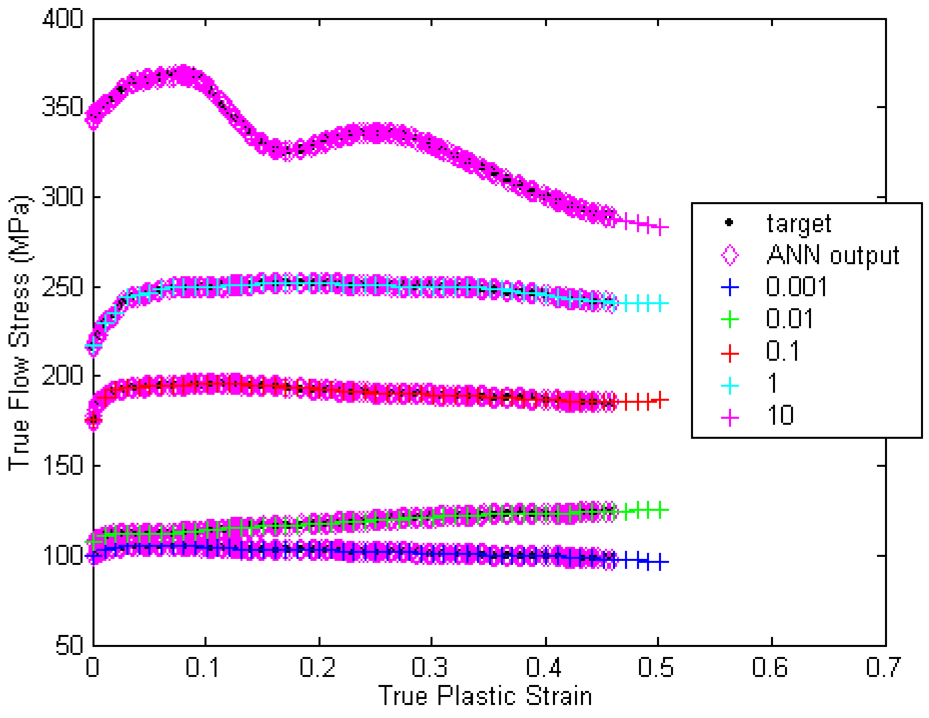


FIGURE 124. Network performance, 30-3 neurons, BR training, for 1175ºC.

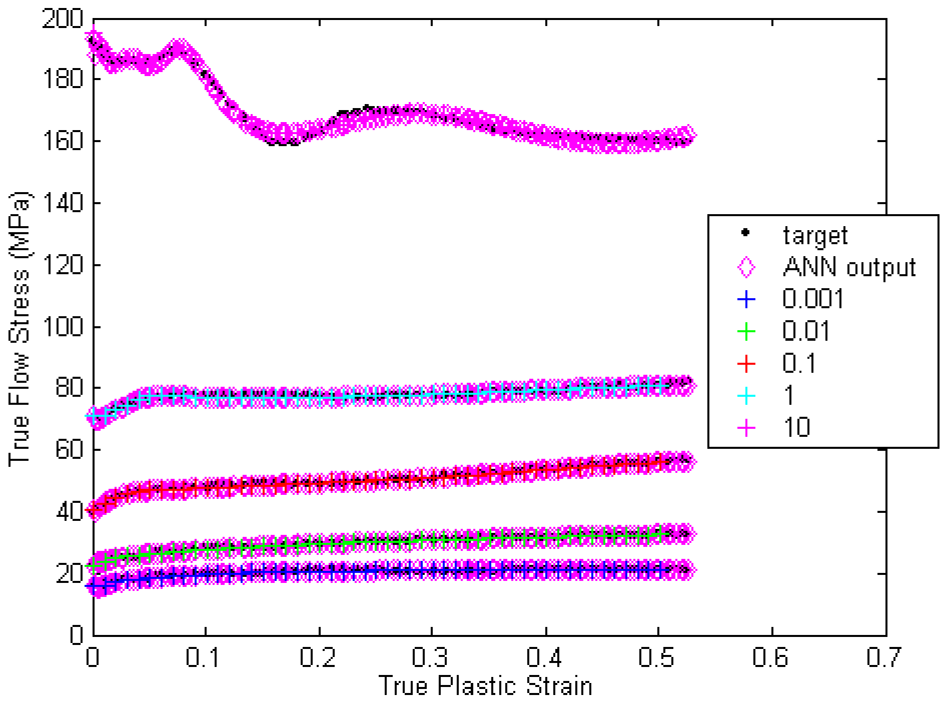


FIGURE 125. Network performance, 30-3 neurons, BR training, for 1250ºC.

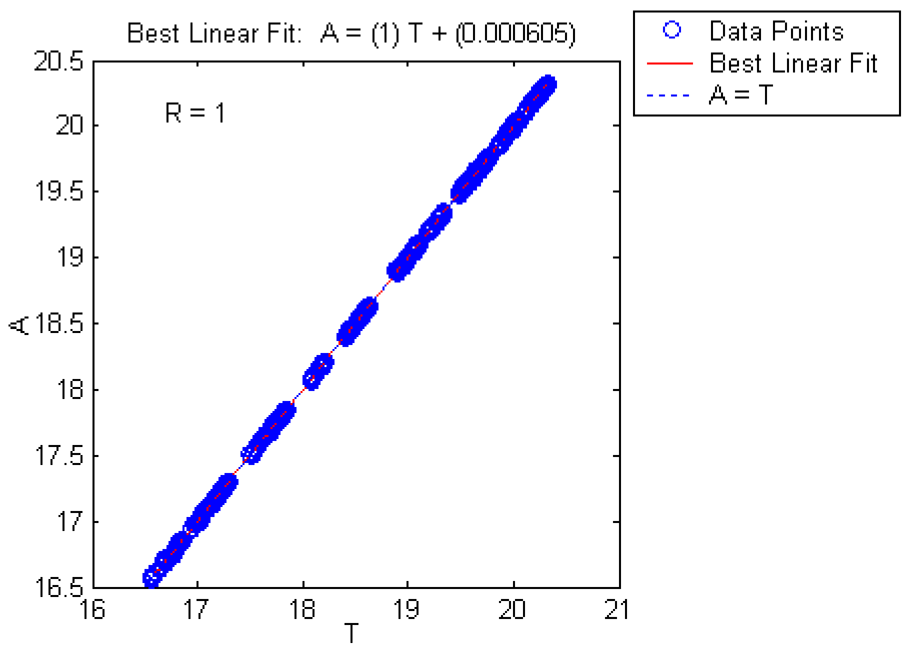


FIGURE 126. Linear regression for the 30-3 neurons, BR trained network for 1100, 1175, and 1250ºC.

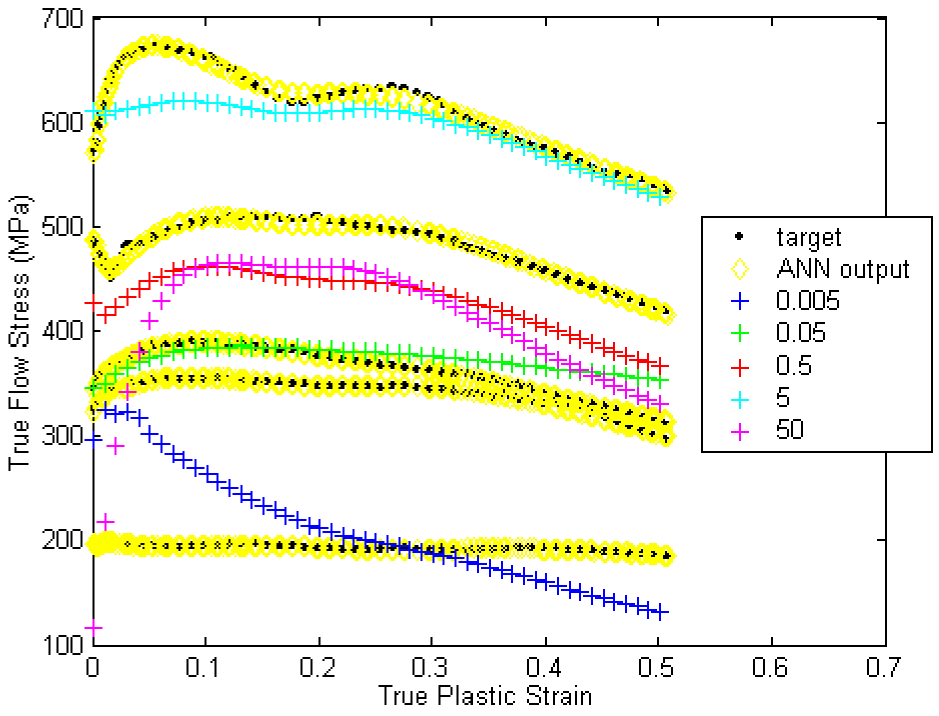


FIGURE 127. Network performance, 30-3 neurons, BR training, for 1100ºC for intermediate values of strain rate.

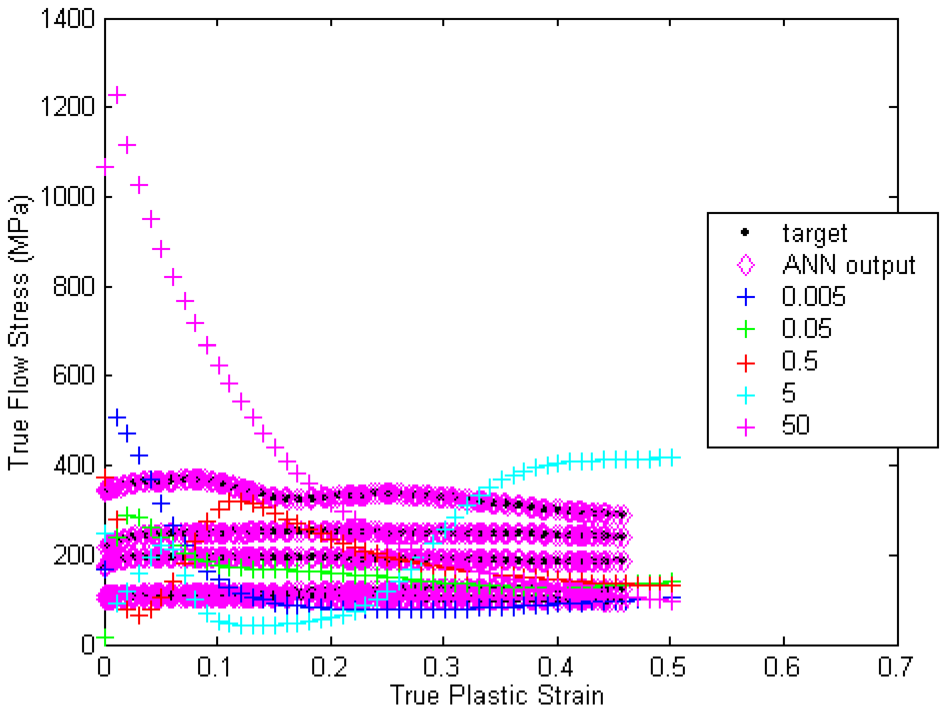


FIGURE 128. Network performance, 30-3 neurons, BR training, for 1175ºC for intermediate values of strain rate.

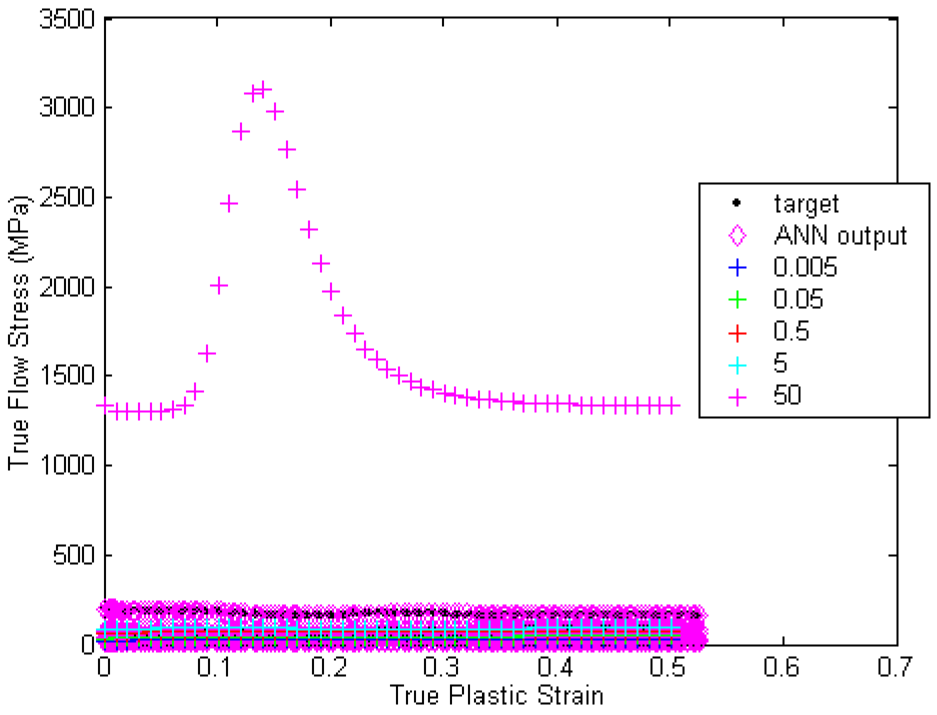


FIGURE 129. Network performance, 30-3 neurons, BR training, for 1250ºC for intermediate values of strain rate.

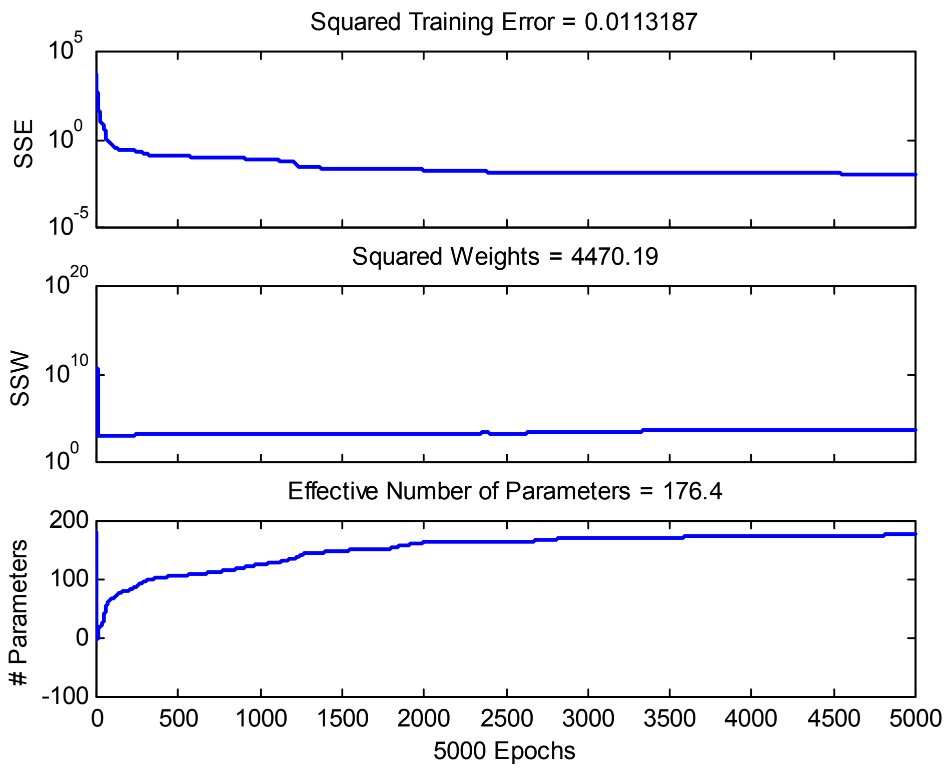


FIGURE 130. Progression of training using fully conditioned data, 25-3 neurons, BR training algorithm.

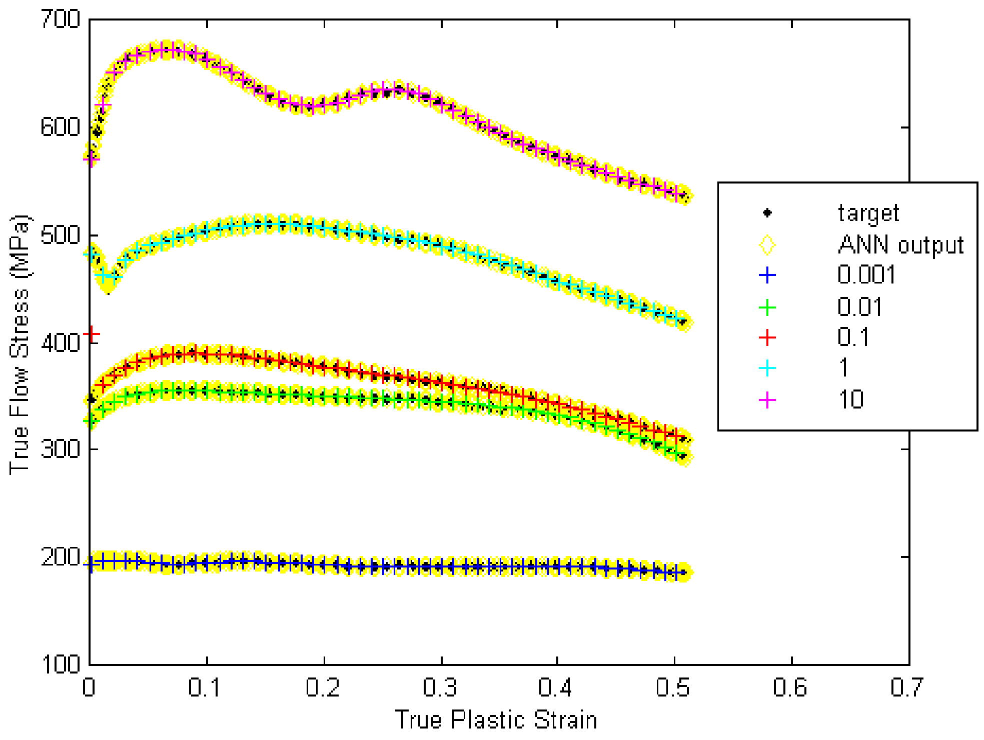


FIGURE 131. Network performance, 25-3 neurons, BR training, for 1100ºC.

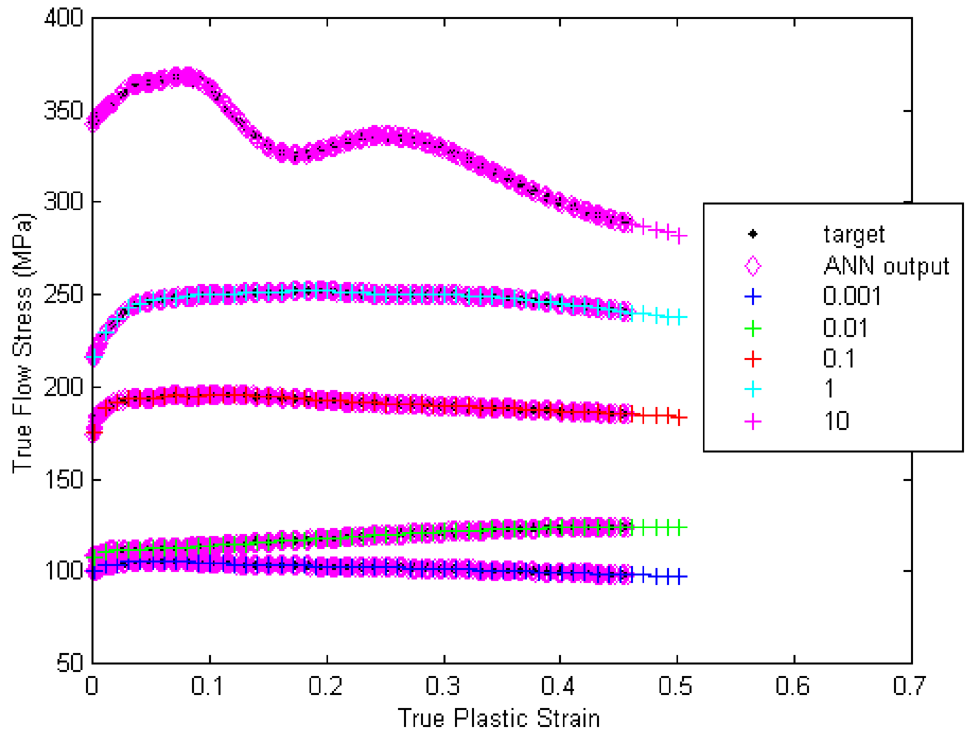


FIGURE 132. Network performance, 25-3 neurons, BR training, for 1175ºC.

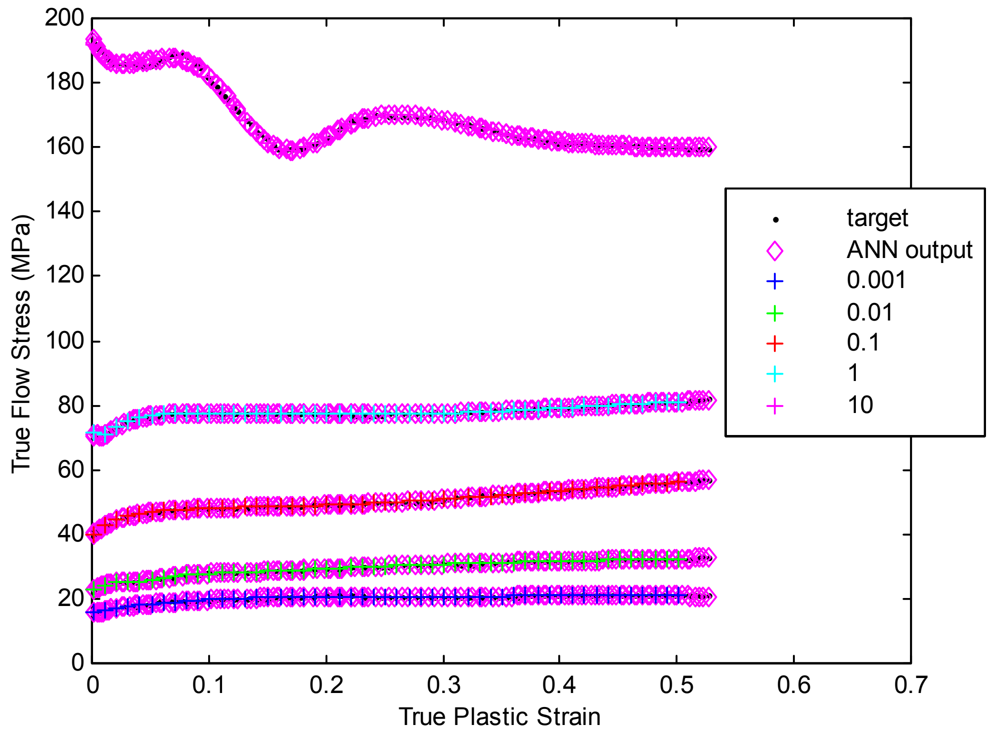


FIGURE 133. Network performance, 25-3 neurons, BR training, for 1250ºC.

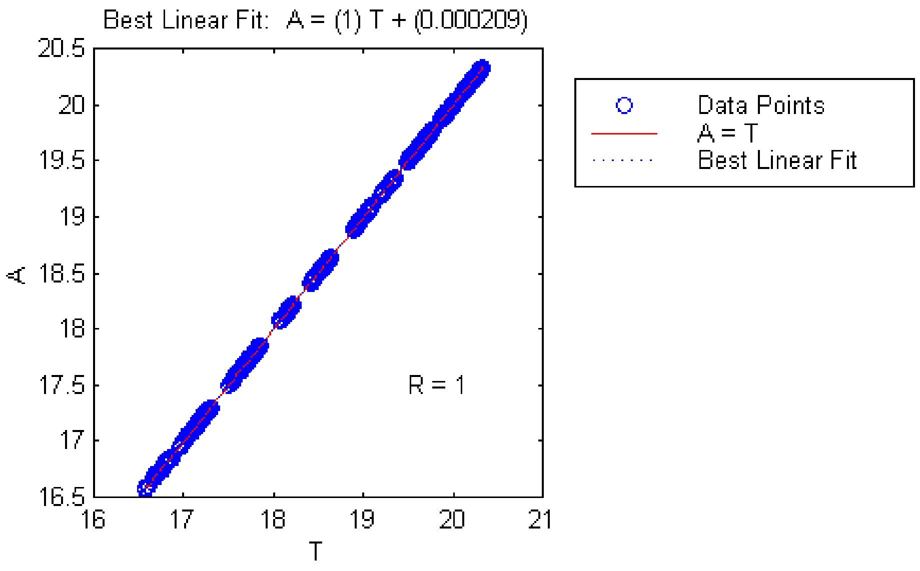


FIGURE 134. Linear regression analysis for the 25-3 BR network.

Final Nickel Aluminide Neural Network Architecture

As was the case with the aluminum network, the 25-3 BR network provides near perfect matching of the experimental data when queried with the training inputs. It also correctly mimics experimental data when presented with varying strains (those not used in training) at strain rates and temperatures within the training set. Without further experimentation, the network performance at intermediate strain rates and temperature cannot be verified completely, but the response appears reasonable based on the data available.

6. Establishment of ANN to FEA Link

The flowchart below, Figure 135, lays out the basic linking, or information passing that occurs between the finite element code and the neural network. As MATLAB was used to fully train the desired network, what remains is providing ABAQUS with a means to pass the appropriate information (i.e., strain, strain rate, temperature) to the feedforward portion of the network. This was accomplished by developing FORTRAN code that accepts these inputs, performs the feedforward function, and subsequently outputs stress and any other material parameters that required for modeling. The general approach is provided below with the actual code found in the Appendix.

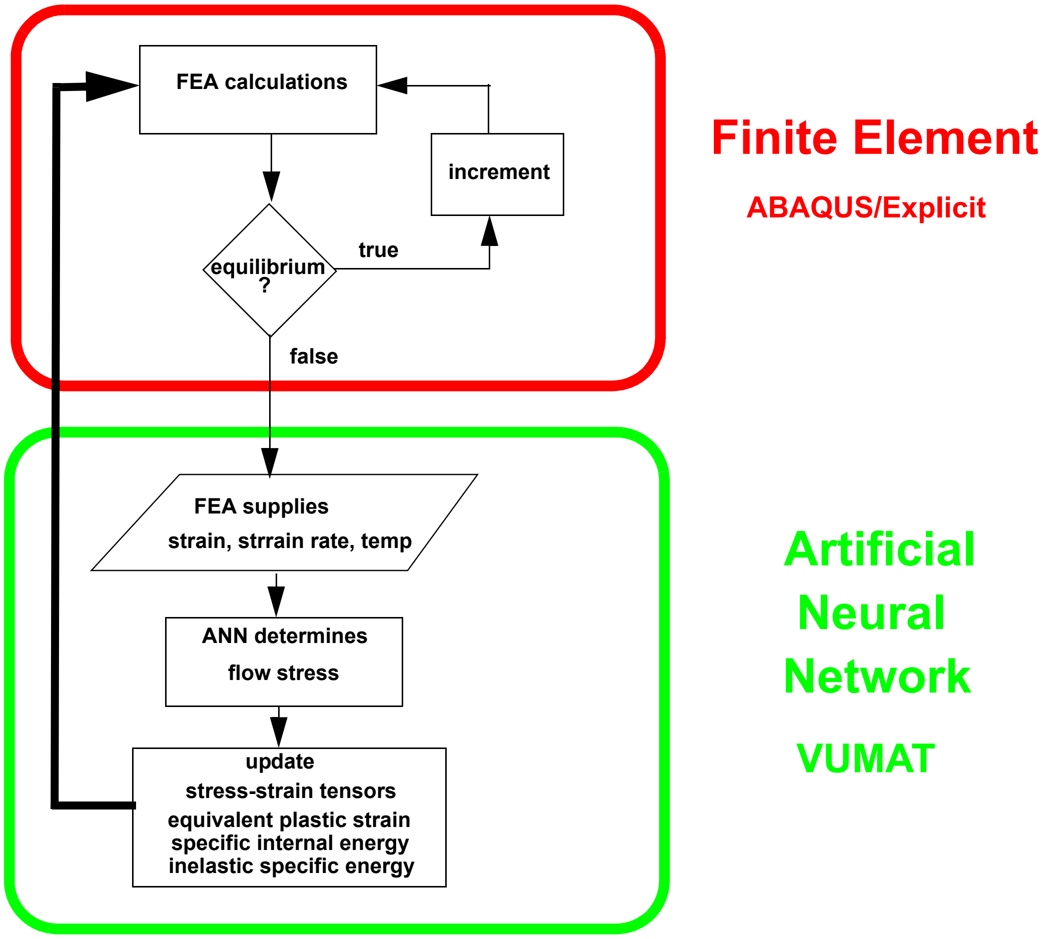

FEA calculations
increment
equilibrium ?
true
false
Finite Element
ABAQUS/Explicit
FEA supplies strain, strrain rate, temp
ANN determines flow stress
update
stress-strain tensors
equivalent plastic strain
specific internal energy
inelastic specific energy
Artificial Neural Network
VUMAT

FIGURE 135. Flowchart detailing linking of the finite element code with the artificial neural network.

The neural network needs the equivalent plastic strain and strain rate which requires the solution of EQ 109 and EQ 110 with the previous increments flow stress used initially, $\sigma_{flowold}$.

$$\varepsilon_{eqplastic} = \frac{\sigma_{mises} - \sigma_{flowold}}{\frac{3}{2}\left(\frac{E}{1+\nu}\right)} \qquad \text{(EQ 109)}$$

$$\dot{\varepsilon}_{eqplastic} = \frac{\sqrt{\frac{2}{9}[(\varepsilon_{inc(k,11)} - \varepsilon_{inc(k,22)})^2 + (\varepsilon_{inck,22)} - \varepsilon_{inc(k,33)})^2 + (\varepsilon_{inc(k,33)} - \varepsilon_{inc(k,11)})^2]}}{\text{time increment size}} \qquad \text{(EQ 110)}$$

The finite element code supplies the stress tensors, $\sigma_{old(k,n)}$, and incremental strain tensors, $\varepsilon_{inc(k,n)}$, at each material point at the beginning of the increment (with *k* indicating the particular material block and *n* the direction). These stress and strain tensors are used to compute the following:

$$\sigma_{11} = \sigma_{old(k,11)} + \frac{E}{(1+\nu)}\varepsilon_{inc(k,11)} + \frac{\nu\left(\frac{E}{1+\nu}\right)}{1-2\nu}trace \qquad \text{(EQ 111)}$$

$$\sigma_{22} = \sigma_{old(k,22)} + \frac{E}{(1+\nu)}\varepsilon_{inc(k,22)} + \frac{\nu\left(\frac{E}{1+\nu}\right)}{1-2\nu}trace \qquad \text{(EQ 112)}$$

$$\sigma_{33} = \sigma_{old(k,33)} + \frac{E}{(1+\nu)}\varepsilon_{inc(k,33)} + \frac{\nu\left(\frac{E}{1+\nu}\right)}{1-2\nu}trace \qquad \text{(EQ 113)}$$

$$\sigma_{12} = \sigma_{old(k,12)} + \frac{E}{(1+\nu)}\varepsilon_{inc(k,12)} \qquad \text{(EQ 114)}$$

with trace equaling:

$$trace = \varepsilon_{inc(k,11)} + \varepsilon_{inc(k,22)} + \varepsilon_{inc(k,33)} \qquad \text{(EQ 115)}$$

The mean stress is then found with EQ 116.

$$\sigma_{mean} = \frac{\sigma_{11} + \sigma_{22} + \sigma_{33}}{3} \qquad \text{(EQ 116)}$$

Now, new values in each stress direction can be found with EQ 117.

$$\sigma_1 = \sigma_{11} - \sigma_{mean}, \sigma_2 = \sigma_{22} - \sigma_{mean}, \sigma_3 = \sigma_{33} - \sigma_{mean} \qquad \text{(EQ 117)}$$

Plugging these values into EQ 118 to find von Mises, and hence update the equivalent plastic strain with EQ 109 above.

$$\sigma_{mises} = \sqrt{\frac{1}{2}(\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + 2\sigma_{12}^2)} \qquad \text{(EQ 118)}$$

Initially, the code preprocesses the newly calculated equivalent plastic strains and strain rates by taking the natural logs. The inverse of the temperature is also computed. In addition, normalization of the modified inputs is also carried out. The weights and biases from the trained network must be input into the FORTRAN code for carrying out the feedforward operations. The code then calculates EQ 119 (the combined form of EQ 14 and EQ 15 for two hidden layers) using FORTRAN's MATMUL (matrix multiplication command) and the exponential form of the hyperbolic tangent function as given by EQ 120.

$$\sigma = \begin{bmatrix} w2_1 & w2_2 & \dots & w2_s \end{bmatrix} \bullet \tanh\left\{ \begin{bmatrix} w2_{11} & w2_{12} & w2_{13} \\ w2_{21} & w2_{22} & w2_{23} \\ \dots & \dots & \dots \\ w2_{s1} & w2_{s2} & w2_{s3} \end{bmatrix} \tanh\left(\begin{bmatrix} w1_{11} & w1_{12} & w1_{13} \\ w1_{21} & w1_{22} & w1_{23} \\ \dots & \dots & \dots \\ w1_{s1} & w1_{s2} & w1_{s3} \end{bmatrix} \begin{bmatrix} \varepsilon \\ \dot{\varepsilon} \\ T \end{bmatrix} + \begin{bmatrix} b1_1 \\ b1_2 \\ \dots \\ b1_s \end{bmatrix} \right) + b2 \right\} + b3 \qquad \text{(EQ 119)}$$

$$\tanh x = \frac{2}{(1 + e^{-2x})} - 1 \qquad \text{(EQ 120)}$$

Once the flow stress, σ_{new}, is computed it must be post processed by de-normalizing. The stress is then used to find the new equivalent plastic strain given by EQ 121.

$$\varepsilon_{eqplasticnew} = \frac{\sigma_{mises} - \sigma_{new}}{\frac{3}{2}\left(\frac{E}{1 + \nu}\right)} \qquad \text{(EQ 121)}$$

The equivalent plastic strain along with the flow stress is passed back to as state variables for use in the next increment (i.e., they become $\varepsilon_{eqplasticold}$ and σ_{old}). In addition, the updated stress tensors (EQ 122 through EQ 125), the specific internal energies (EQ 126 and EQ 127), and the dissipated inelastic specific energies (EQ 128) are computed and passed back.

$$\sigma_{new(k,\,11)} = \sigma_{11}\left(\frac{\sigma}{\left(\sigma + \frac{3}{2}\left(\frac{E}{1+\nu}\right)\right)}\right) + \sigma_{mean} \tag{EQ 122}$$

$$\sigma_{new(k,\,22)} = \sigma_{22}\left(\frac{\sigma}{\left(\sigma + \frac{3}{2}\left(\frac{E}{1+\nu}\right)\right)}\right) + \sigma_{mean} \tag{EQ 123}$$

$$\sigma_{new(k,\,33)} = \sigma_{33}\left(\frac{\sigma}{\left(\sigma + \frac{3}{2}\left(\frac{E}{1+\nu}\right)\right)}\right) + \sigma_{mean} \tag{EQ 124}$$

$$\sigma_{new(k,\,12)} = \sigma_{12}\left(\frac{\sigma}{\left(\sigma + \frac{3}{2}\left(\frac{E}{1+\nu}\right)\right)}\right) \tag{EQ 125}$$

$$\begin{gathered}\sigma_{power} = \frac{1}{2}[(\sigma_{old(k,\,11)} + \sigma_{new(k,\,11)})\varepsilon_{inc(k,\,11)} + \\ (\sigma_{old(k,\,22)} + \sigma_{new(k,\,22)})\varepsilon_{inc(k,\,22)} + \\ (\sigma_{old(k,\,33)} + \sigma_{new(k,\,33)})\varepsilon_{inc(k,\,33)} + (\sigma_{old(k,\,12)} + \sigma_{new(k,\,12)})]\end{gathered} \tag{EQ 126}$$

$$\text{specific internal energy}_{new} = \text{specific internal energy}_{old} + \frac{\sigma_{power}}{density} \tag{EQ 127}$$

$$\text{inelastic specific energy}_{new} = \text{inelastic specific energy}_{new} + \frac{1}{2}(\sigma_{old} + \sigma_{new})\varepsilon_{eqplastic} \tag{EQ 128}$$

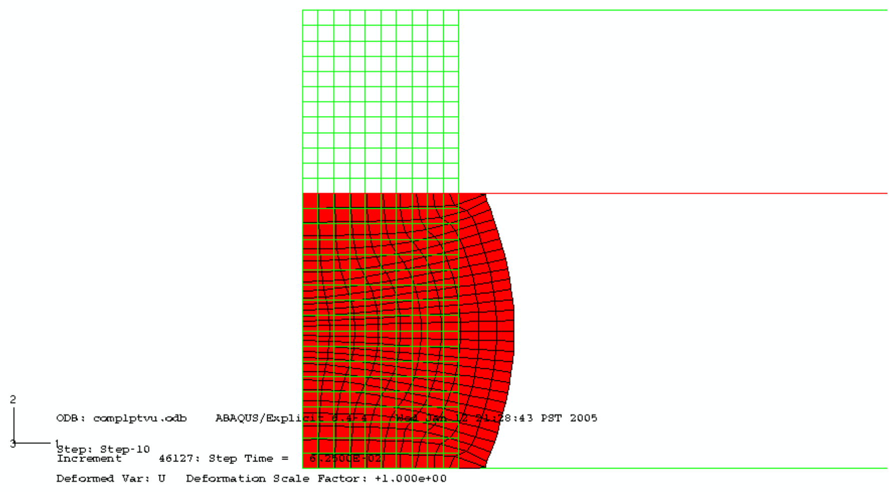
7. Simple Compression FEA Models

7.1 Basic Model Construction

At this point, with both conventional and ANN-based material models constructed, along with the appropriate linking of the finite element platform with the ANN established, a finite element model for simple compression of an aluminum and a nickel aluminide billet is developed. The intent, in each case, is to test both approaches against the experimental results sited in the published literature (the same sources used to fashion the materials models). These simple models are also used to verify the proper functioning of the ANN linking.

Several aspects are common to the FEA models that are presented below. The basic construction of each is the same. The materials constants, those not addressed directly by the material model, are provided to the finite element software (e.g., density, conductivity, Poisson’s ratio, etc.) as previously given in Table 1 and Table 3. For purposes of simplification, the compression was assumed to be isothermal without adiabatic heat rise. The models are axisymmetric, comprised of 4-node bilinear, reduced integration, hourglass-controlled elements (CAX4R). The dies are analytically rigid with the bottom one fixed (encastre) and the top moved downward at a constant strain rate. The top die is controlled by a reference point preventing motion other than in the up and down (U2) direction (i.e., U1=UR2=UR3=0). Another boundary condition is established along the plan of symmetry preventing sideways motions or rotation, but allowing movement up or down.

ABAQUS’s "classical" hard contact algorithm is used throughout. Friction is accounted for using the penalty method with a coefficient of friction of 0.15. The modeled billets were upset 40% to achieve approximately 0.5 strain to match experimental results (see Figure 136).



```
ODB: complptvu.odb    ABAQUS/Explicit 6.4-4    Wed Jan 12 21:28:43 PST 2005

Step: Step-10
Increment    46127: Step Time =   6.2500E-02

Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

FIGURE 136. Undeformed and 40% reduced billet.

To enable comparisons with the experimental data, runs of each model were made at constant strain rates of 0.01, 0.1, and 1.0. A strain rate of 10.0 was also added for the nickel aluminide. Constant strain rates were achieved by splitting the total displacement of 6 mm into ten equal steps of 0.6 mm. These displacements were then accomplished using a constant velocity over the appropriate length of time. As an example, Table 10 gives the values needed to cause compression at a strain rate of 0.01/s. Rates of 0.1, 1.0, and 10/s require decreasing the duration and increasing the velocity by factors of 10, 100, 1000, respectively. While this method does not produce exact constant strain rates through the simulation, it does reflect the method commonly employed with testing equipment. Many universal testing machines do not have the ability to drive a crosshead at a constant strain rate, but can be driven at constant velocities, so a step-wise approach is frequently applied.

TABLE 10. Step durations and velocity to simulate a strain rate of 0.01/s.

Step	Duration (s)	Velocity (mm/s)
1	4.000	-0.150
2	4.167	-0.144
3	4.348	-0.138
4	4.545	-0.132
5	4.762	-0.126
6	5.000	-0.120
7	5.263	-0.114
8	5.555	-0.108
9	5.882	-0.102
10	6.250	-0.096
Total	*47.772*	*na*

The displacement and the force applied by the upper platen as it varied through time was calculated by the finite element code. The experimental results, both for the aluminum and the nickel aluminide, report true flow stress versus true strain. This requires the force versus displacement relations to be transformed as given by EQ 129 and EQ 130.

$$\text{true stress} = \bar{\sigma} = \frac{F}{A_{initial}}\left(1 + \frac{\Delta l}{l}\right) \qquad \text{(EQ 129)}$$

$$\text{true strain} = \bar{\varepsilon} = \ln\left(1 + \frac{\Delta l}{l}\right) \qquad \text{(EQ 130)}$$

What follows concentrates on verifying the aluminum models as they are required for the much larger and more complicated forged wheel analysis. The nickel aluminide model is provided to further demonstrate the applicability of the ANN-linked FEA model to more difficult to describe rhealogies.

7.2 Compression of Aluminum Billet

The sited literature [1] does not provide the size of the specimen compressed, so one 15 mm tall and 10 mm in diameter was assumed as this is a typical size recommendation per ASTM standards [8]. Using the elements noted above, the initial mesh is comprised of squares with sides measuring 0.5 mm (i.e., 10 x 30 elements).

For the conventional model, both power law and tabular approaches where tested. The code requires fitting EQ 106. Using the tabular data for 450ºC, Figure 137 was generated and a coefficient of 3.83 and an exponent of 0.33 was used for the power law model. For the tabular model, the values were input directly.

For implementation of either the overstress power law or the tabular-based model, estimates of static yield stress and elastic modulus, in this case at 450ºC, are also required. Values of 15 MPa, and 54 GPa were used for static yield and modulus, respectively.

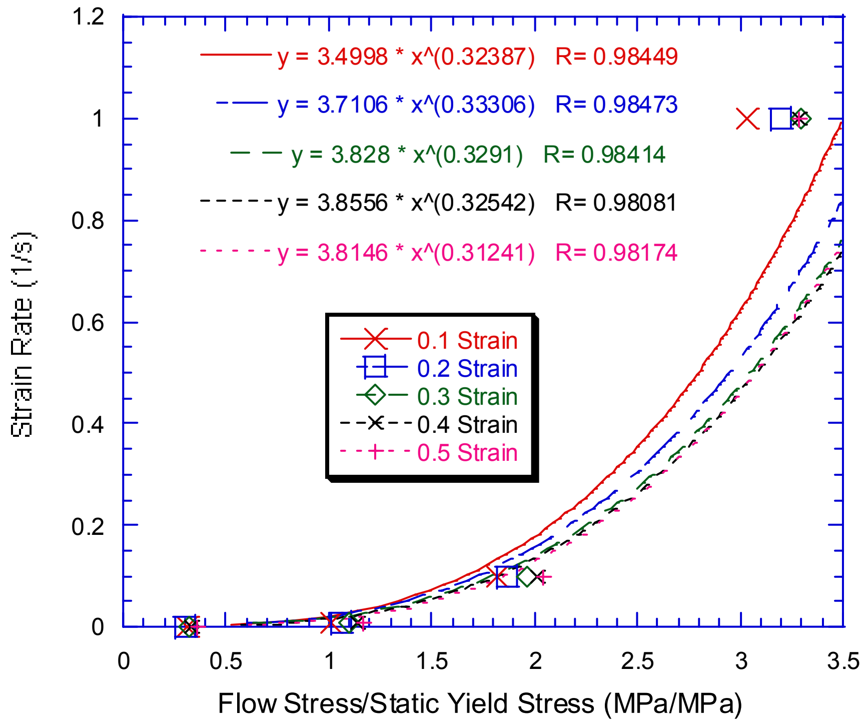


FIGURE 137. Flow stress to static yield stress ratio plot for determining coefficient and exponent.

The weights and biases from the 15-3 neural network developed above were input into the FORTRAN program that produces the feedforward portion of the ANN. ABAQUS, through its VUMAT capability calls the program, which includes definitions for the yield surface (based on von Mises stresses), the flow rule, and the evolution law (i.e. hardening behavior).

The next set of few figures, Figure 138 through Figure 140, exhibit how force varies through time based on each of the models. Clearly, the tabular and ANN models predict similar forces with the exception that the ANN has a smoother, likely more reasonable, application of force. The power law model, as could be anticipated based on the poor fit, forecasts significantly different forces at the high and low strain rates.

Figure 141 through Figure 149 provides comparisons of von Mises stress distributions across each of the models. It should be noted that the stress distribution plots vary based on the calculated maximum and minimum stresses present.

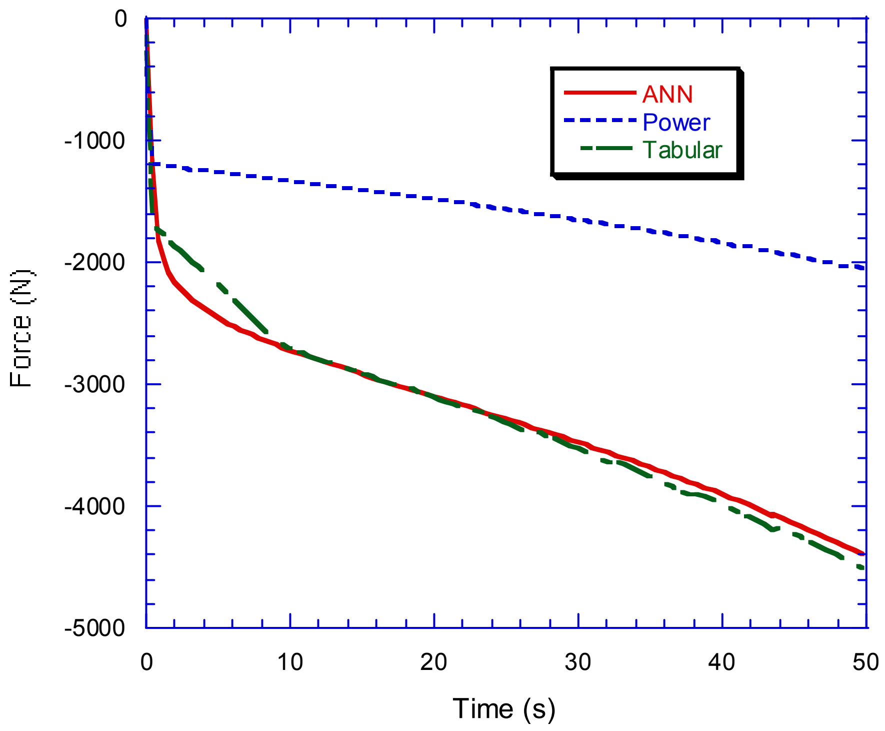


FIGURE 138. Force versus time at a strain rate of 0.01/s for each model.

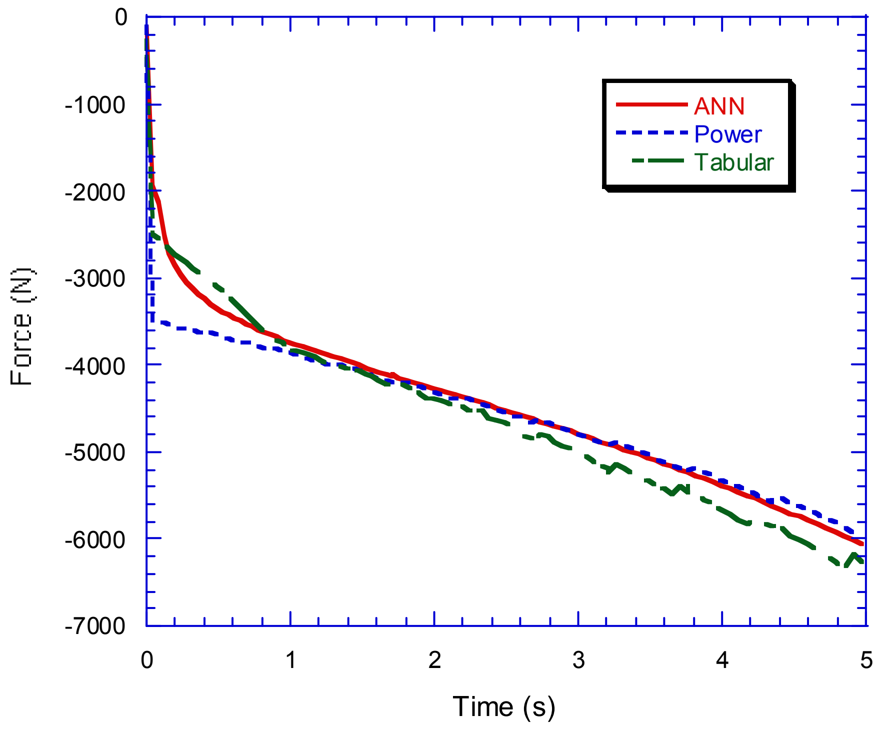


FIGURE 139. Force versus time at a strain rate of 0.1/s for each model.

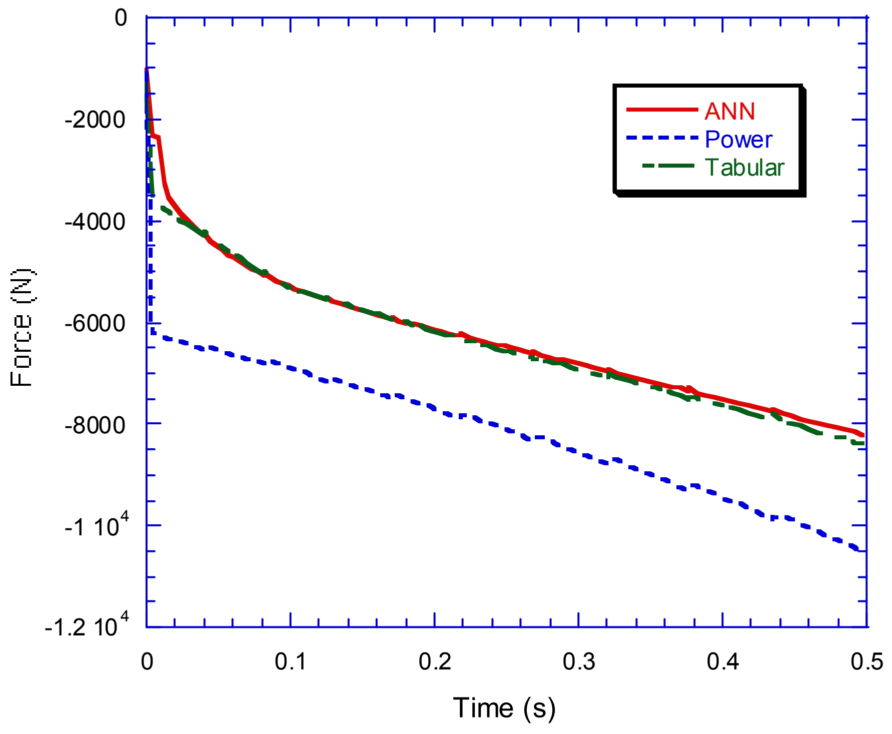


FIGURE 140. Force versus time at a strain rate of 1/s for each model.

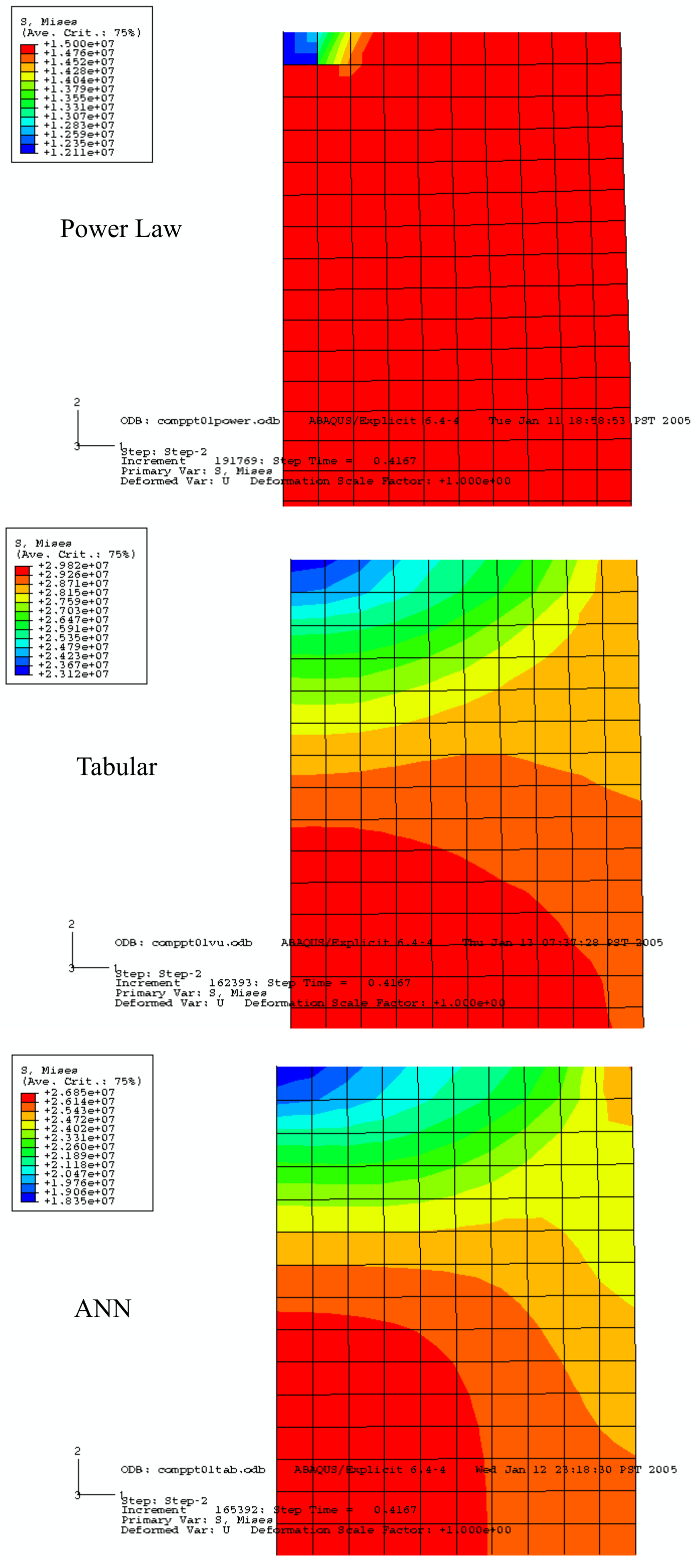


FIGURE 141. Comparisons of von Mises stress distributions with a strain rate of 0.01/s at 4% reduction.

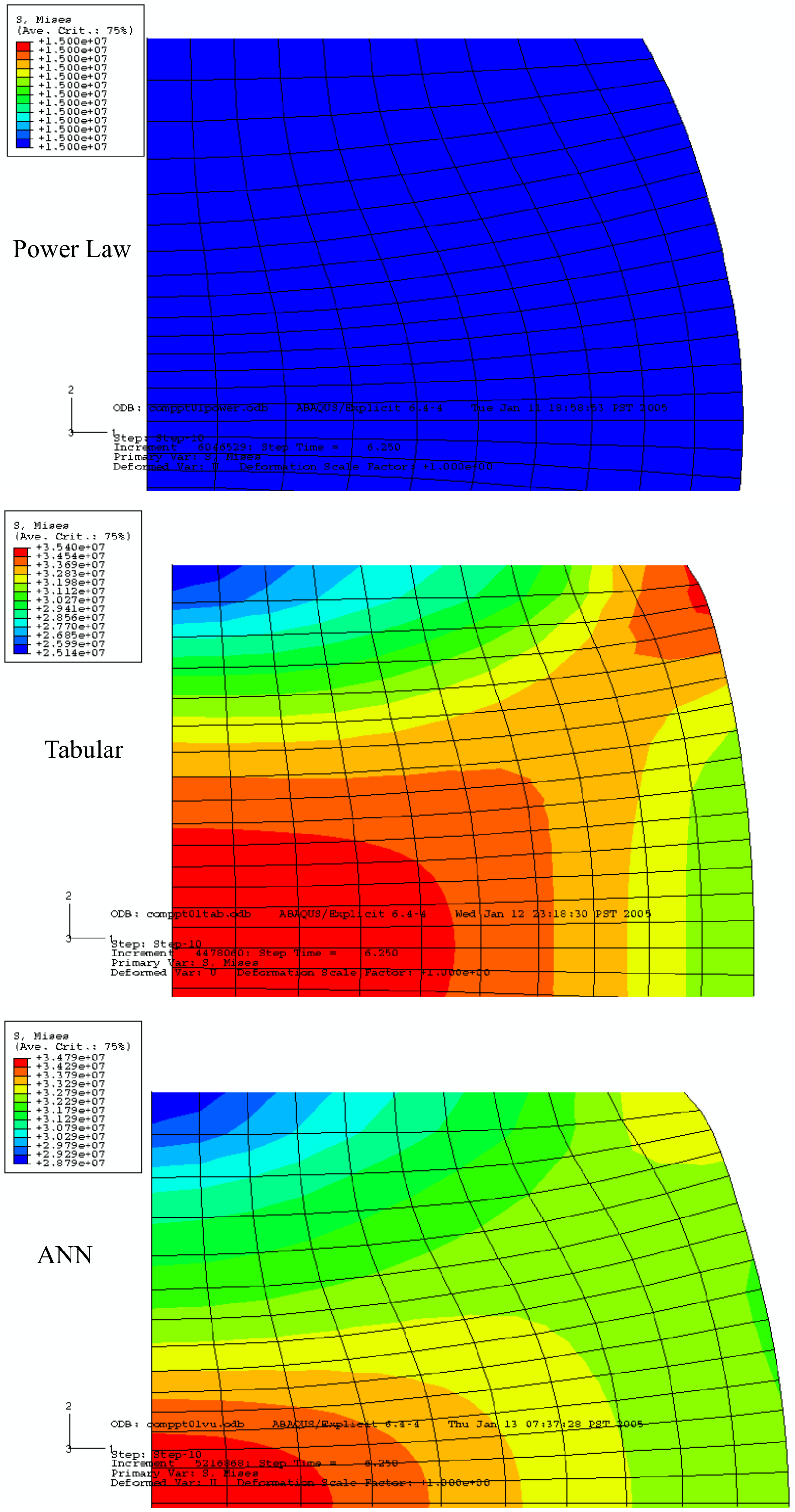


FIGURE 142. Comparisons of von Mises stress distributions with a strain rate of 0.01/s at 40% reduction.

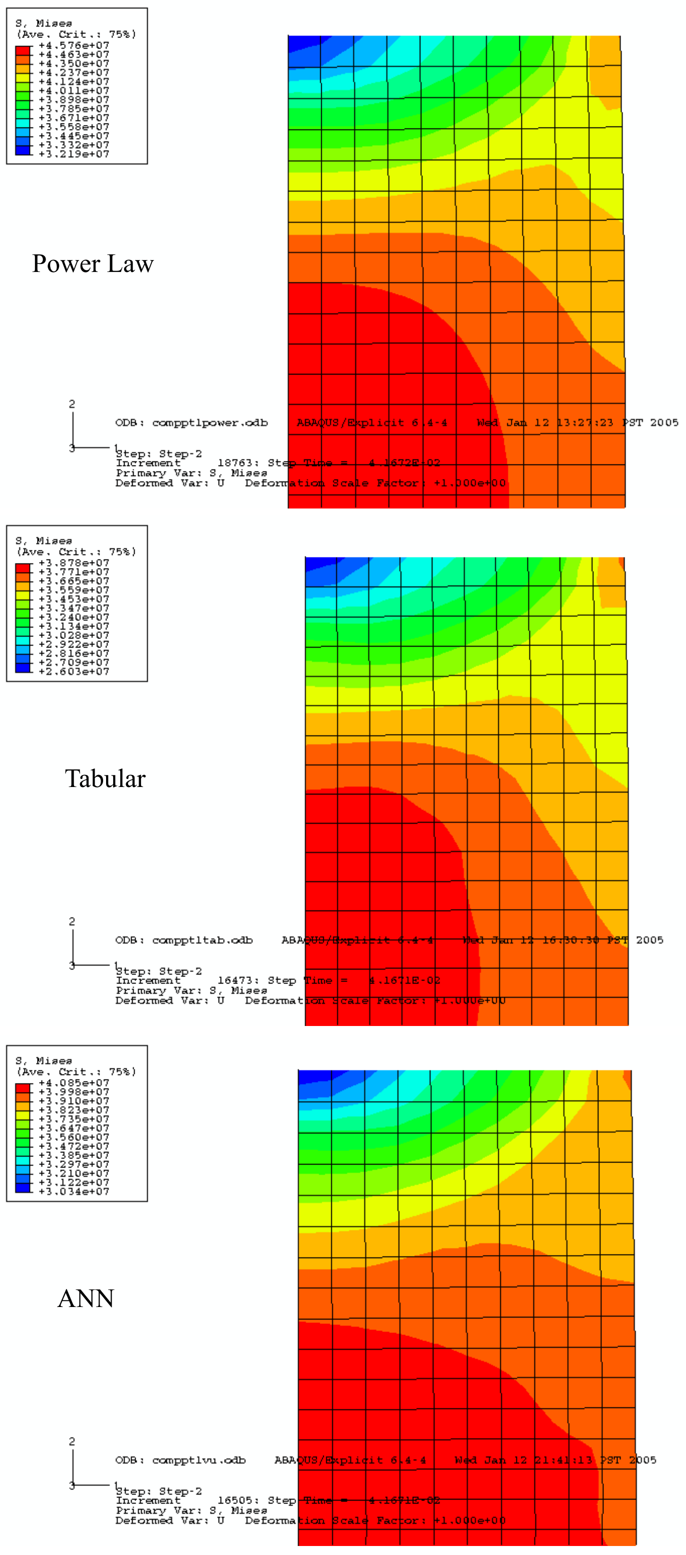


FIGURE 143. Comparisons of von Mises stress distributions with a strain rate of 0.1/s at 4% reduction.

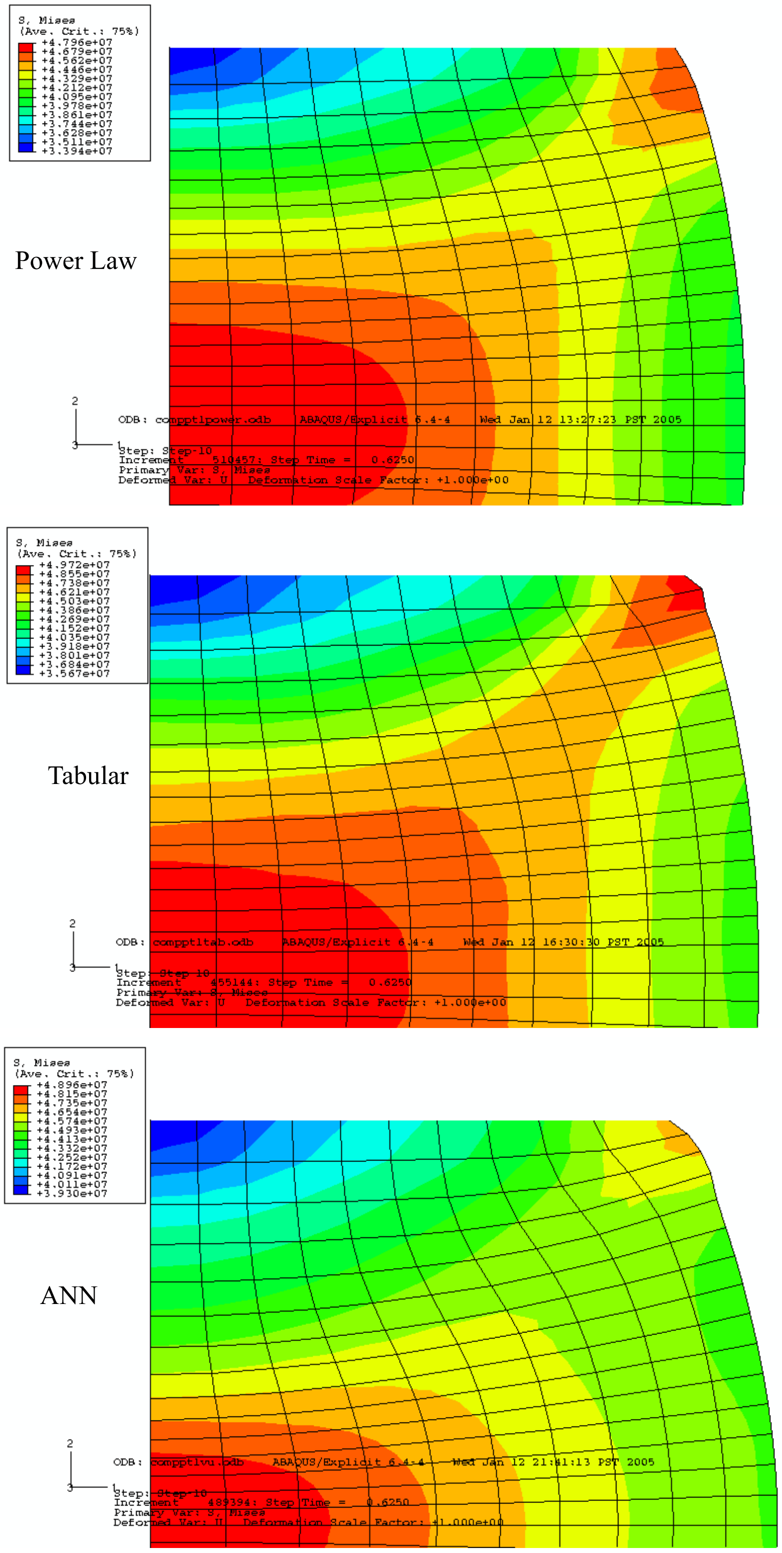


FIGURE 144. Comparisons of von Mises stress distributions with a strain rate of 0.1/s at 40% reduction.

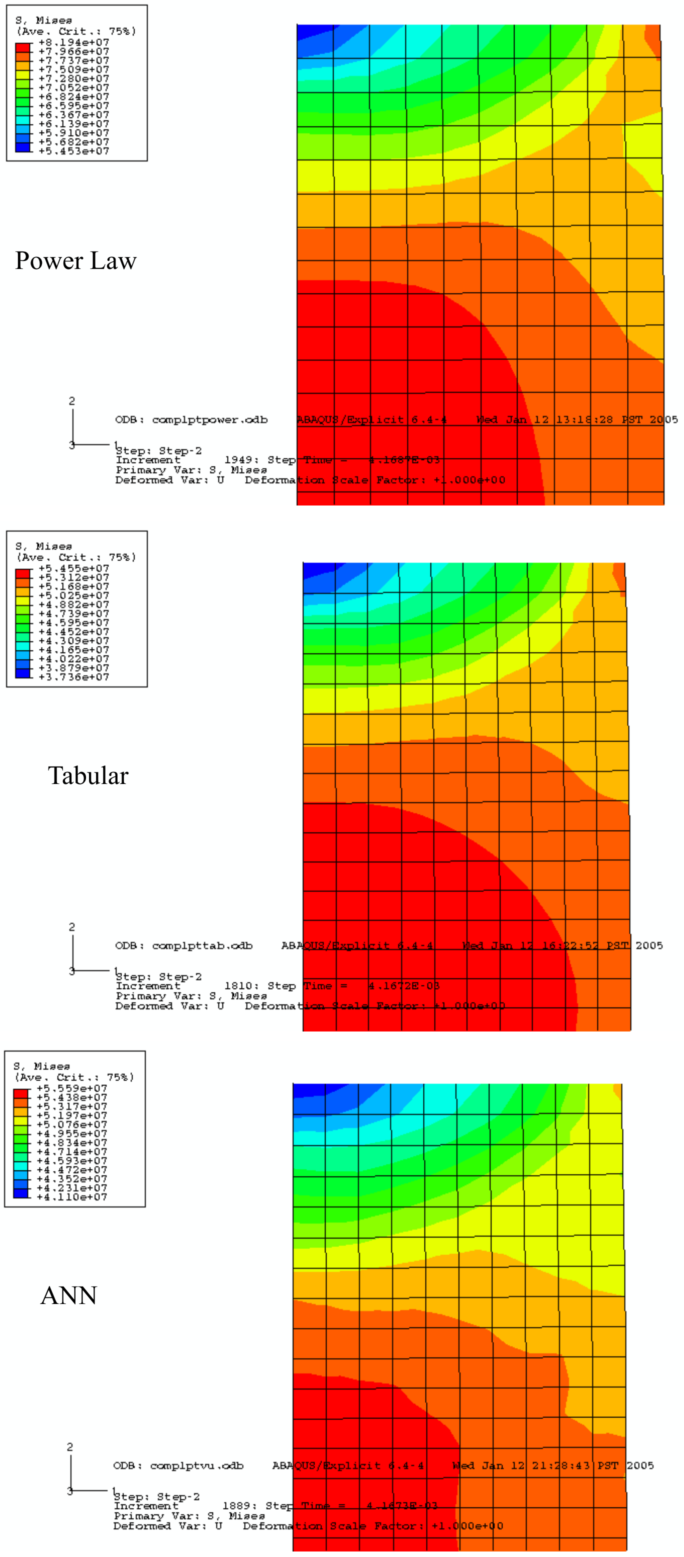


FIGURE 145. Comparisons of von Mises stress distributions with a strain rate of 1/s at 4% reduction.

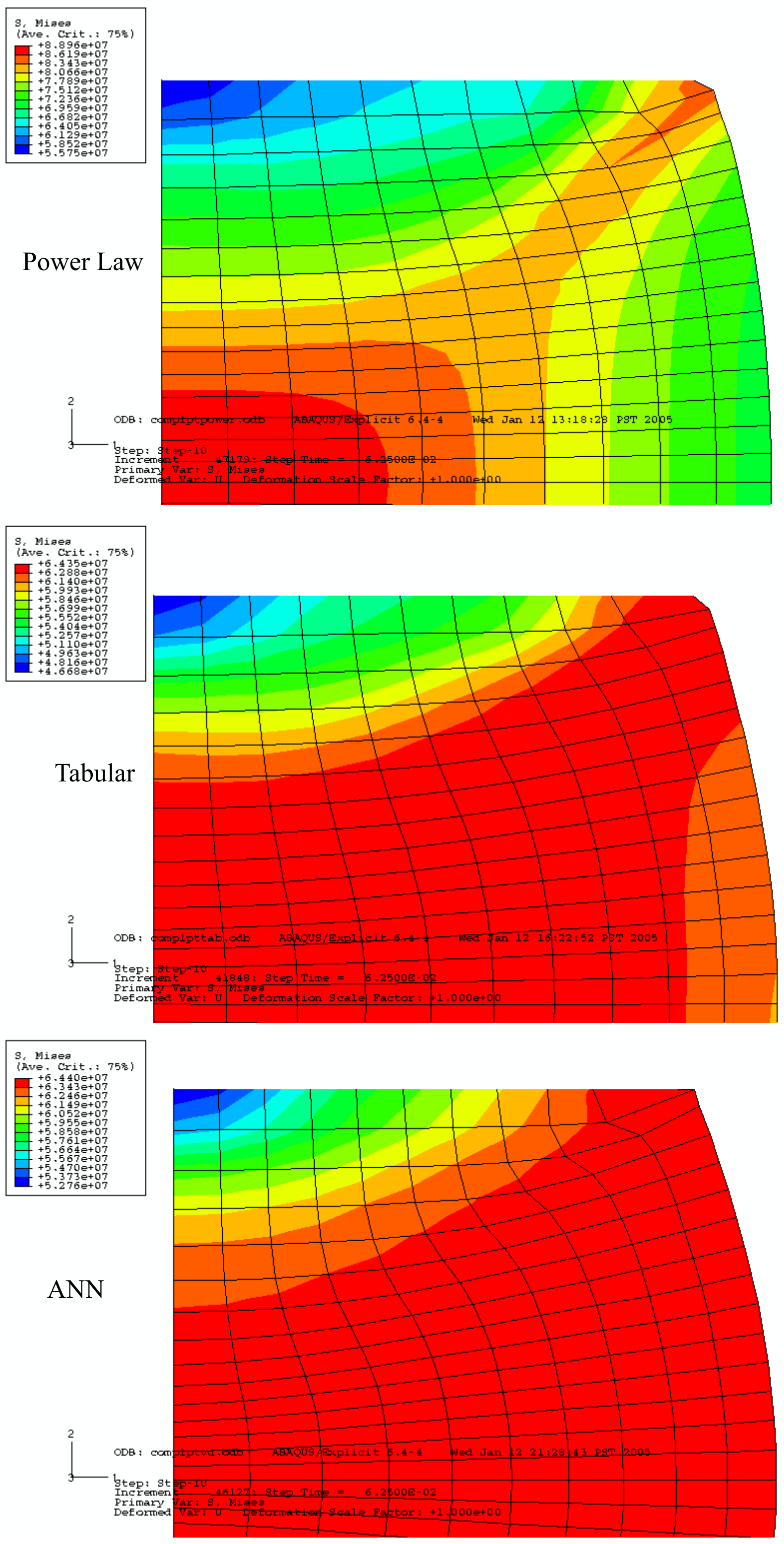


FIGURE 146. Comparisons of von Mises stress distributions with a strain rate of 1/s at 40% reduction.

7.3 Summary of Aluminum Billet Compression Results

The figures, Figure 147 through Figure 149, clearly demonstrate that the ANN material model possesses a superior ability to mirror experimental results. The power law model only produces reasonable results once significant strain has occurred and for only a rate of 0.1/s. This results from the curve fit straying from the actual values at lower and higher rates. The tabular-based model provides a more reasonable approximation, but it should be noted that values of static yield stress and elastic modulus had to be estimated for input into the model. The tabular model also yields a less smooth or rounded curve as might be expected when examining experimental data at 300 or 550ºC. The ANN model does not require yield estimates as it has the ability to supply flow stress values for the entire range of strain. It also represents the entire flow stress curve so high temperature estimates of the elastic modulus are not needed.

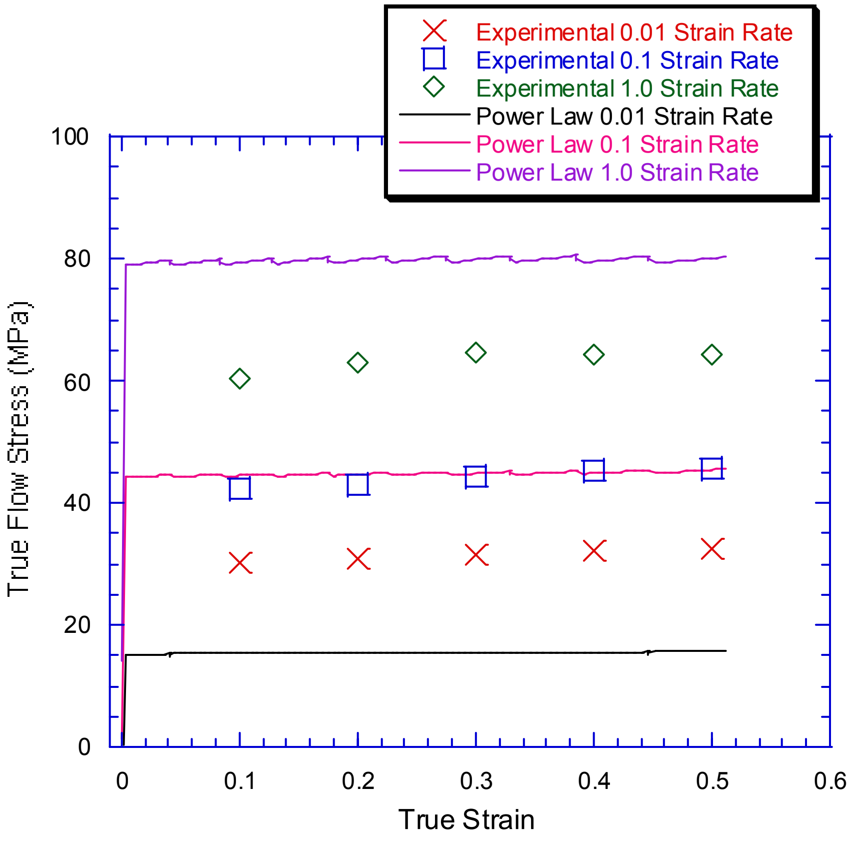


FIGURE 147. Comparison of power law model with experimental during compression at 450ºC.

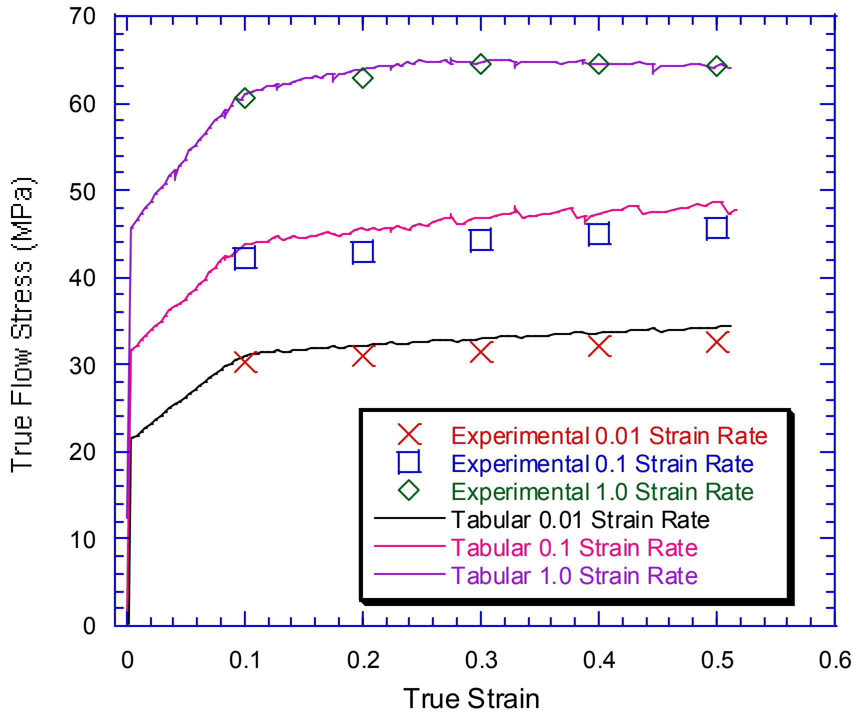


FIGURE 148. Comparison of tabular based model with experimental during compression at 450ºC.

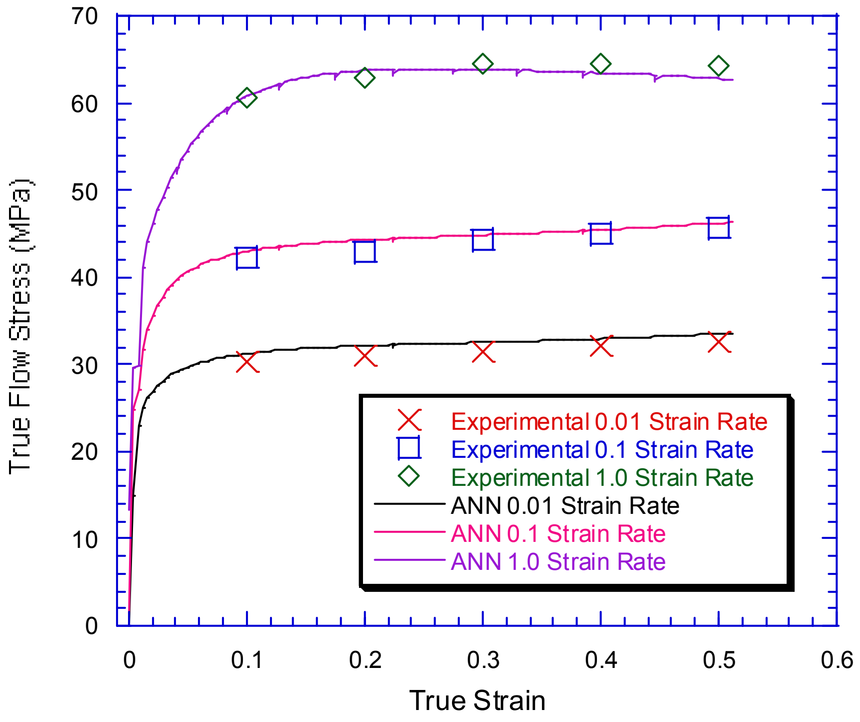


FIGURE 149. Comparison of ANN-based model with experimental during compression at 450ºC.

7.4 Compression of Nickel Aluminide Billet

The nickel aluminide model specimen measures 10 mm in diameter by 15 mm tall to match the experimental sample [2]. Again, using the elements noted above, the initial mesh is comprised of squares with sides measuring 0.5 mm (i.e., 10 x 30 elements). Based on the experience with the aluminum model and due to the desire to principally test the ANN approach, only the power law material model was run. The ANN model utilizes the 25-3 BR trained network established previously.

As before, a plot of stress ratios is needed to determine the coefficient and exponent. In this case, only a strain of 0.3 is shown in Figure 150. The power law has a somewhat poor ability to fit the highest strain rate of 10.

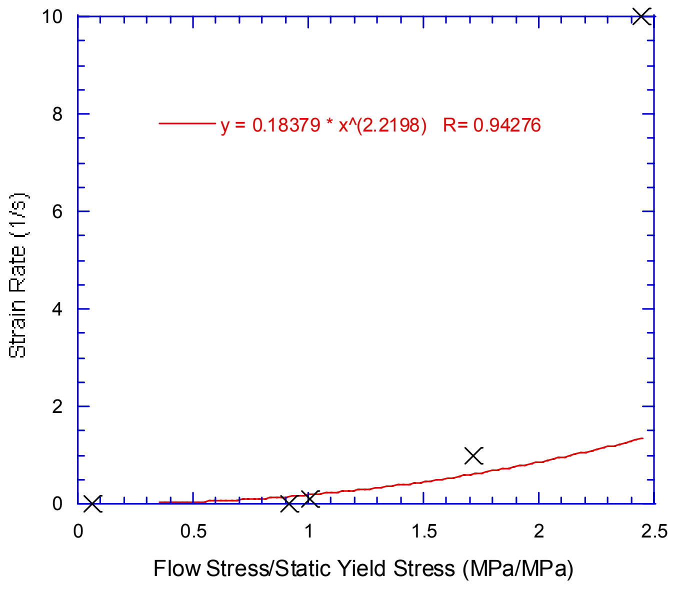


FIGURE 150. Flow stress to static yield stress ratio plot for determining coefficient and exponent at a strain of 0.3.

For implementation the overstress power law, the finite element code requires estimation of static yield stress and elastic modulus, in this case at 1100ºC. Values of 180 MPa, and 179 GPa were used for static yield and modulus, respectively. Both values were estimated based on the curves supplied above.

Figure 151 through Figure 158 provide comparisons of von Mises distributions.

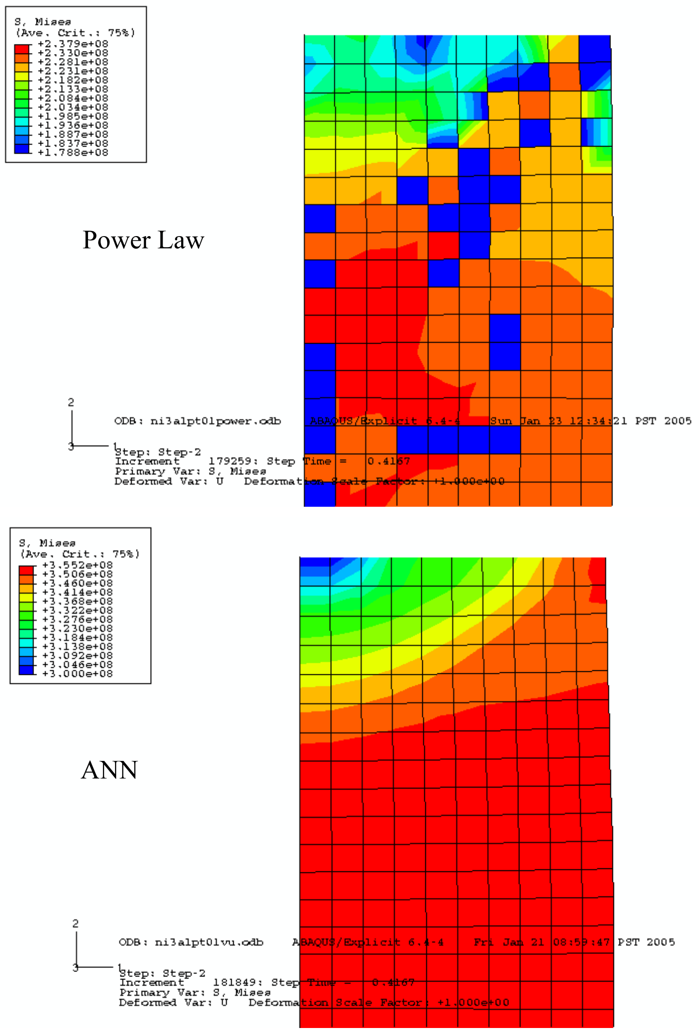


FIGURE 151. Comparisons of von Mises stress distributions with a strain rate of 0.01/s at 4% reduction.

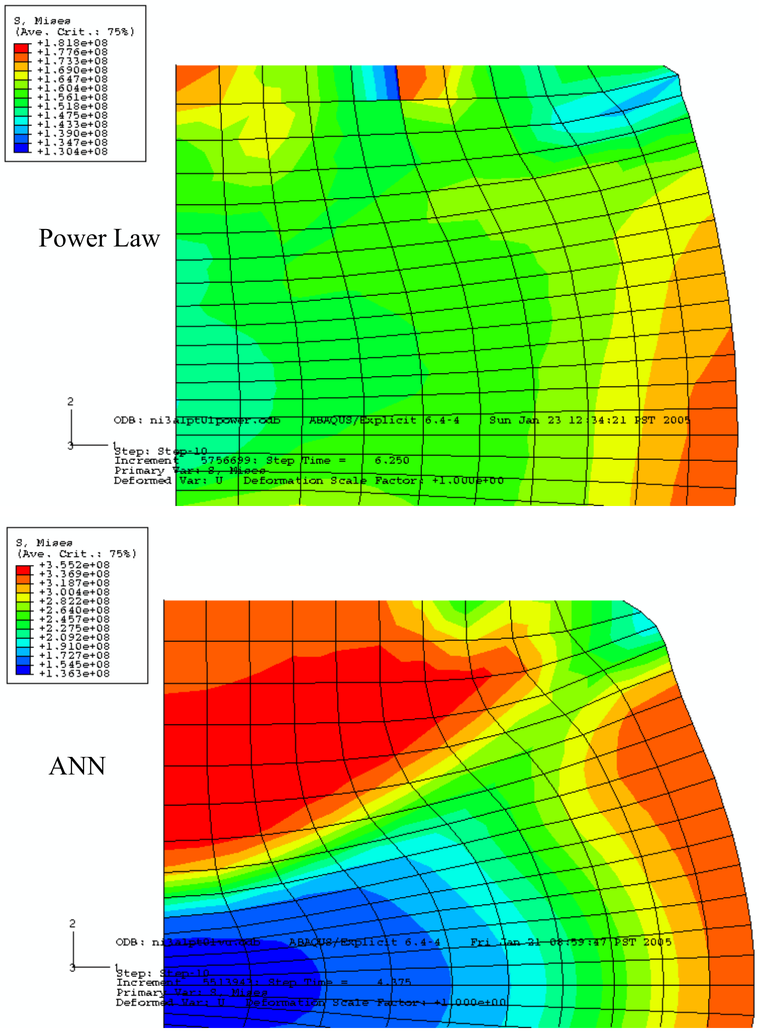


FIGURE 152. Comparisons of von Mises stress distributions with a strain rate of 0.01/s at 40% reduction.

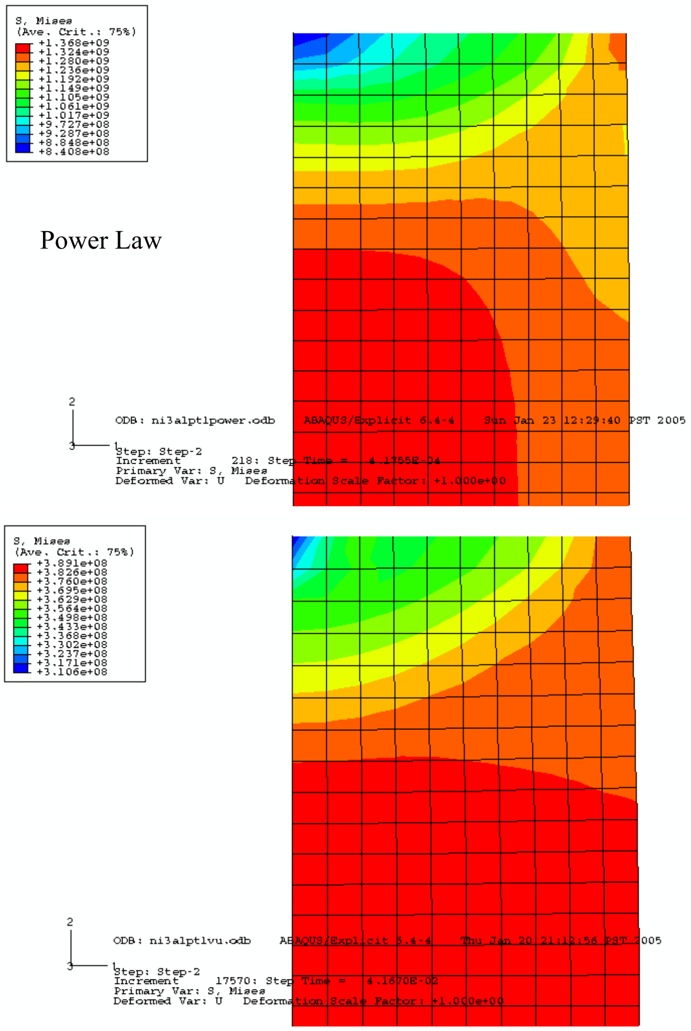


FIGURE 153. Comparisons of von Mises stress distributions with a strain rate of 0.1/s at 4% reduction.

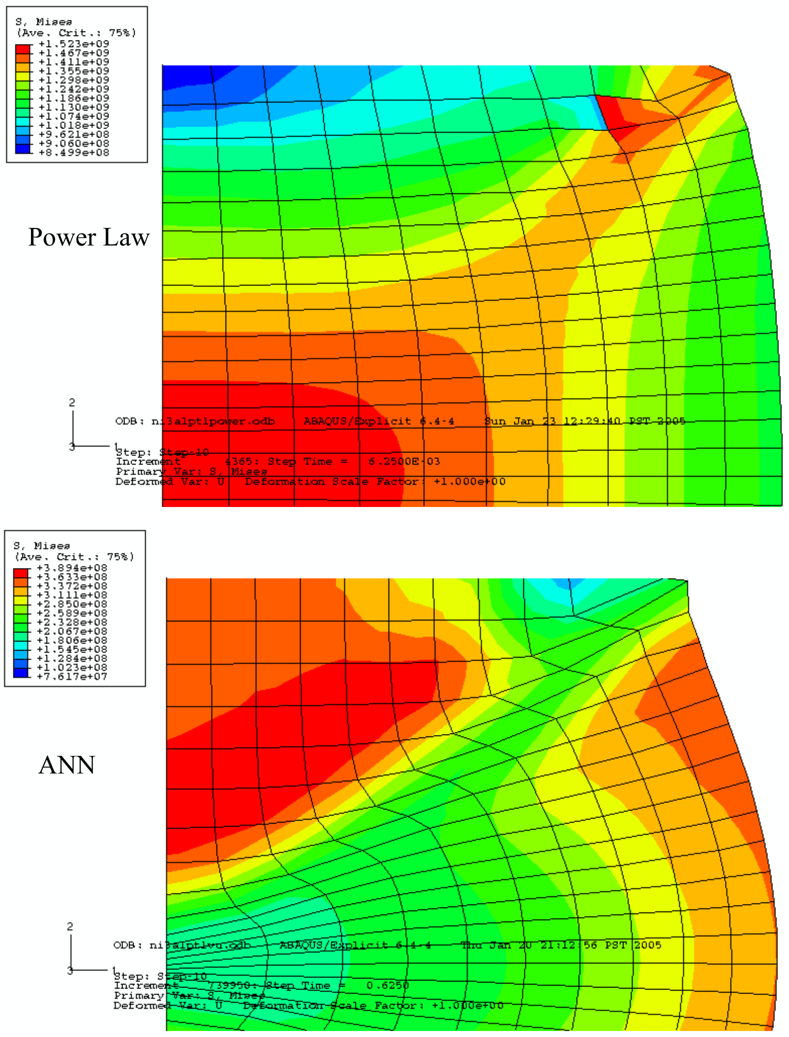


FIGURE 154. Comparisons of von Mises stress distributions with a strain rate of 0.1/s at 40% reduction.

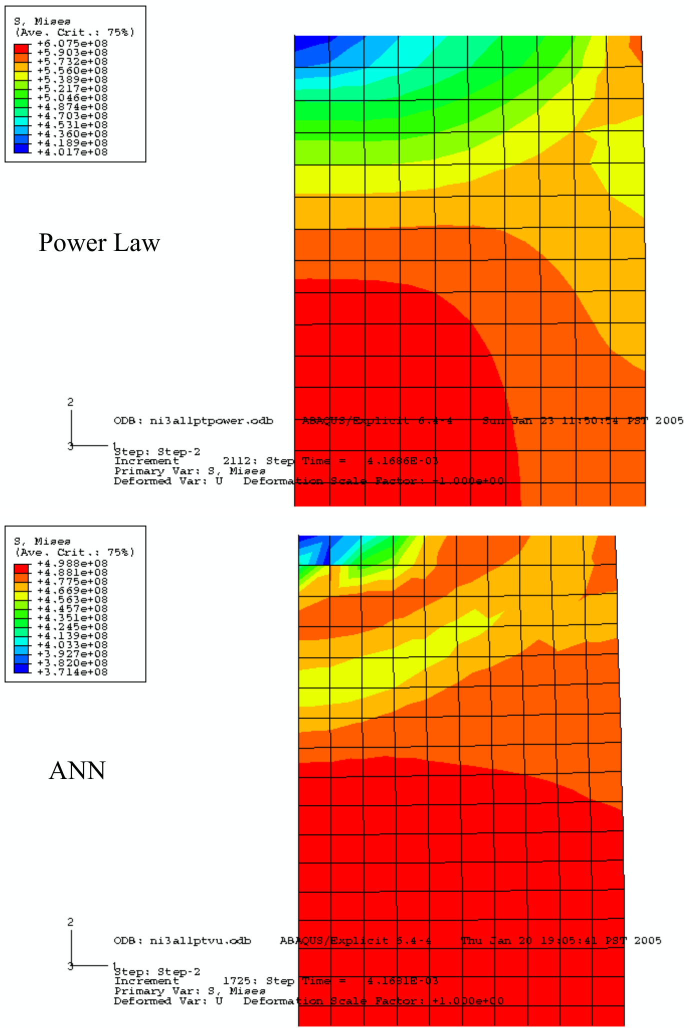


FIGURE 155. Comparisons of von Mises stress distributions with a strain rate of 1/s at 4% reduction.

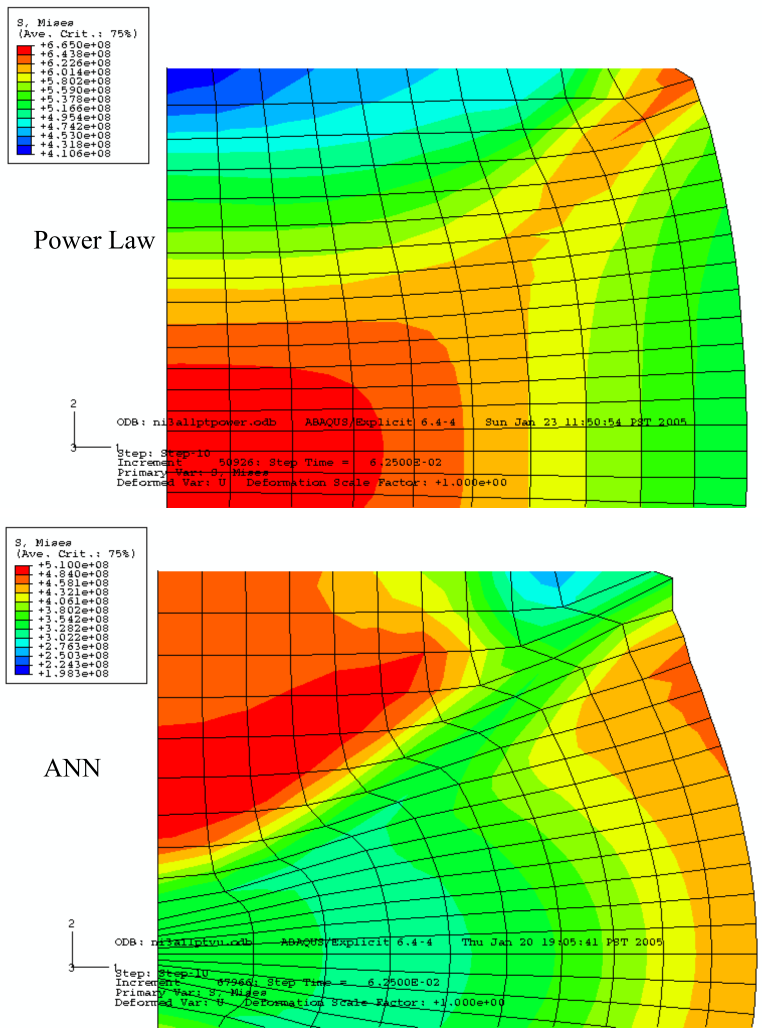


FIGURE 156. Comparisons of von Mises stress distributions with a strain rate of 1/s at 40% reduction.

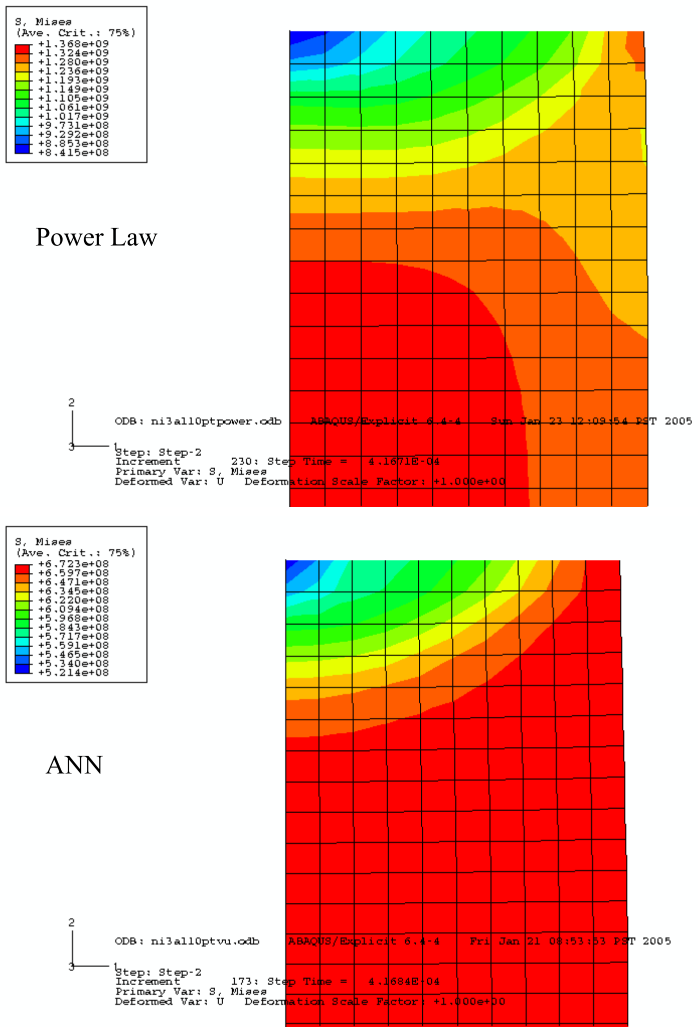


FIGURE 157. Comparisons of von Mises stress distributions with a strain rate of 10/s at 4% reduction.

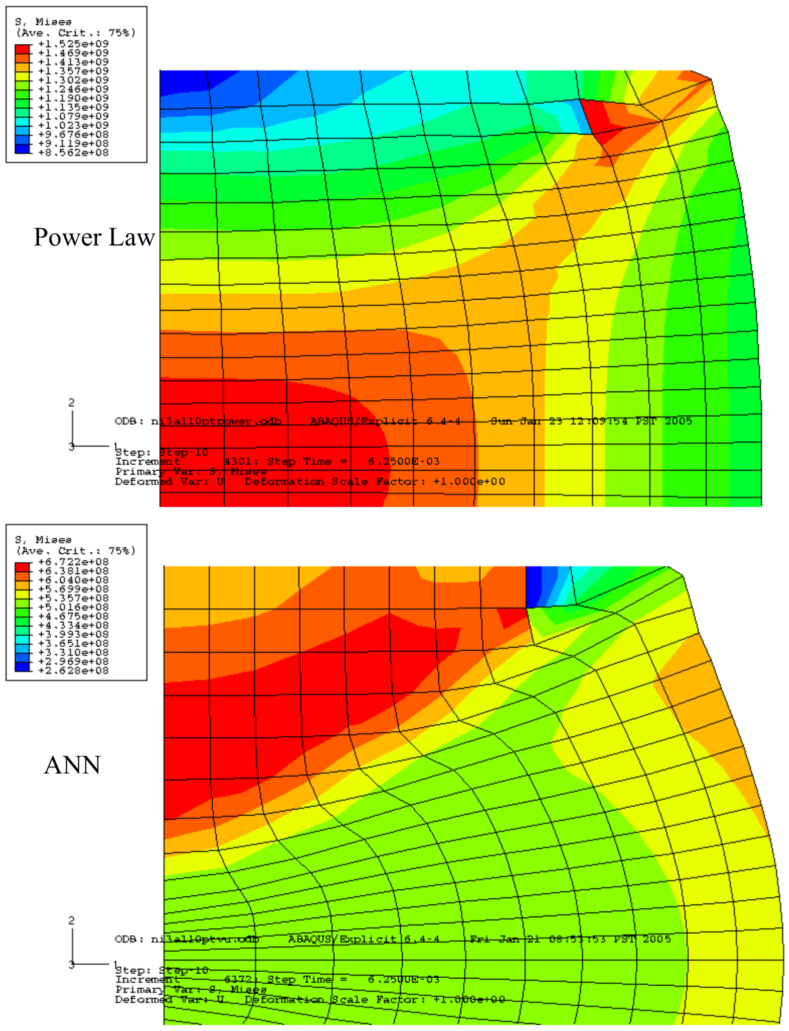


FIGURE 158. Comparisons of von Mises stress distributions with a strain rate of 10/s at 40% reduction.

7.5 Summary of Nickel Aluminide Billet Compression Results

The two figures below, Figure 159 and Figure 160, demonstrate the superior ability of the ANN model to mirror experimental experience. The power law, as may be anticipated due to the lack of an adequate fit, produces much higher flow stress values at greater strain rates. While the ANN model may not reproduce the experimental results exactly, it does perform much better. It is likely a "better" network, whether requiring a different architecture or more training, could be constructed that would more closely match real world behavior.

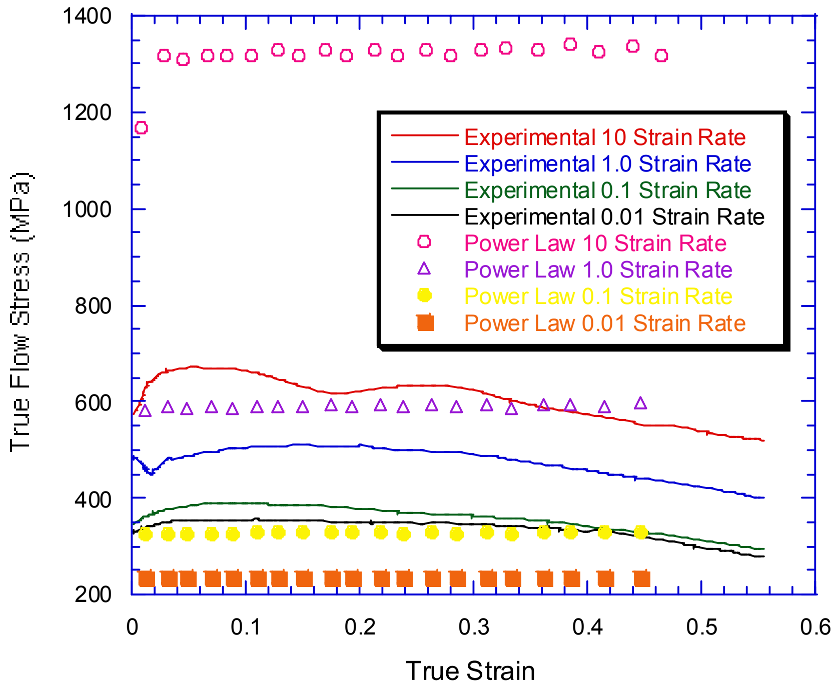


FIGURE 159. Comparison of power law based model with experimental during compression at 1100ºC, every 5th FEA data point plotted.

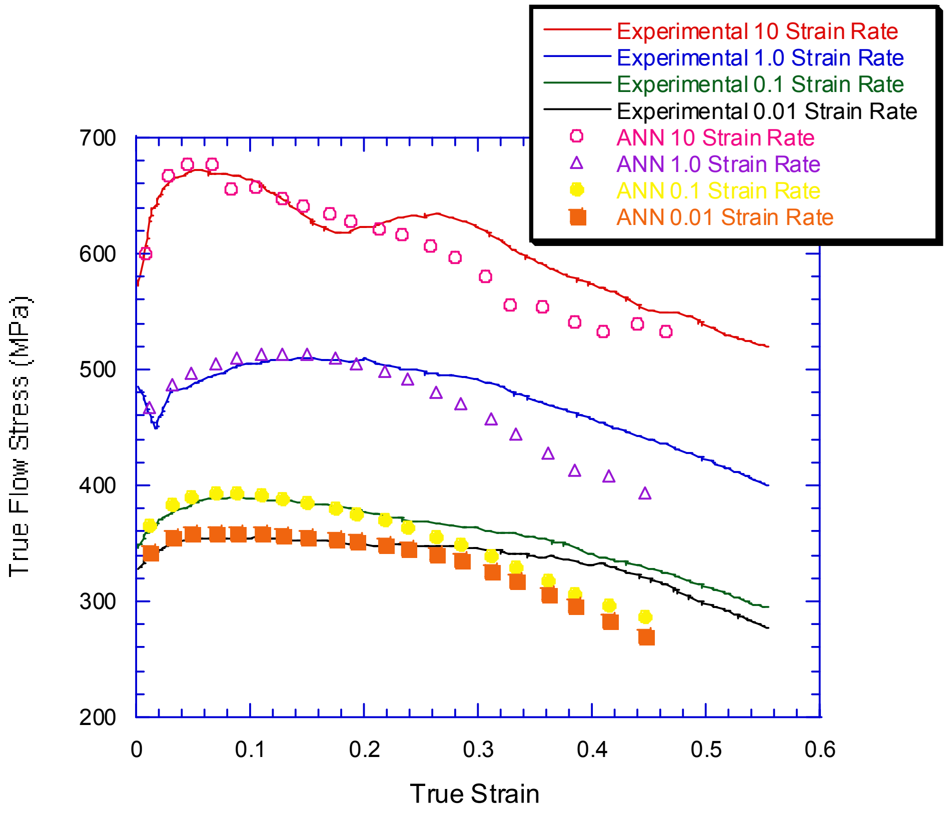


FIGURE 160. Comparison of ANN-based model with experimental during compression at 1100ºC, every 5th FEA data point plotted.

8. Case Study: Wheel Manufacturing Process

The ultimate aim of this research is to provide an accurate model for use in predicting material behavior during forging operations. After having established the ANN-FEA linked model's ability to mirror experimental results during simple compression, a real-world example is needed to more fully verify model performance under production conditions. In an effort to establish accuracy of the ANN/FEA model developed above, relevant measurements of the forging of a 16 inch aftermarket wheel were taken during manufacture. The wheel is a product of Weld Wheel Industries Inc. of Kansas City, Missouri. Weld Wheels produces a large selection of aftermarket automotive and motorcycle wheels for the racing and vanity markets. The auto accessories industry is estimated to be worth $29 billion with performance and custom wheel comprising $3.3 billion. Weld does not disclose their exact revenues, but states that they are between $50 and $100 million annually [70].

Currently, when a new wheel design is to be manufactured, the forging portion of the wheel's production is designed through experience coupled with trial and error. Weld does not currently employ FEA as an aid to the design process of forging while they do use finite element modeling to establish the strength of a particular final design. The forging under consideration has been produced for many years and is machined into several different final products. A sketch of the wheel crossection at the completion of forging is shown in Figure 161.

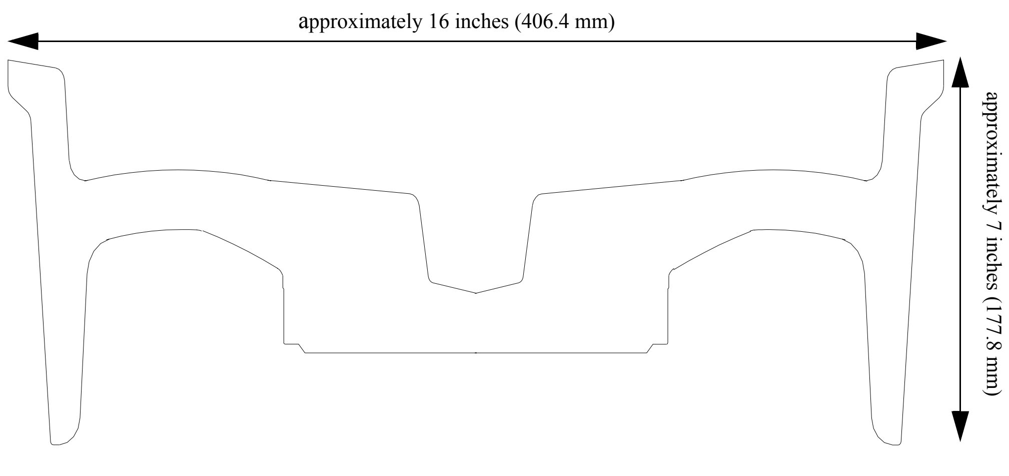

approximately 16 inches (406.4 mm)
approximately 7 inches (177.8 mm)

FIGURE 161. Crossectional view of aluminum wheel after forging operations (exact dimensions omitted at the request of Weld Racing).

8.1 Experimental Results

Data Acquisition and Description of Process

A National Instruments DAQCard-AI-16XE-50 multifunction I/O card in conjunction with Labview software was used to acquire the forging press crosshead displacement using an LVDT (G.L. Collins Corp., LMT-689VO1 with a Schaevitz ATA-101 amplifier) and hydraulic pressure using a pressure transducer (Sensometrics SP97KFS). Surface temperature measurements were also made through the process (Fluke 87). Readings were taken for ten consecutive wheel forging operations. Figure 162 shows the experimental set-up used for pressure and displacement acquisition. Figure 163 provides the front side view of the press and the location of the pressure transducer. Due to limitations with the instrumentation and calibration, pounds per square inch and inches of displacement were acquired as opposed to SI units.

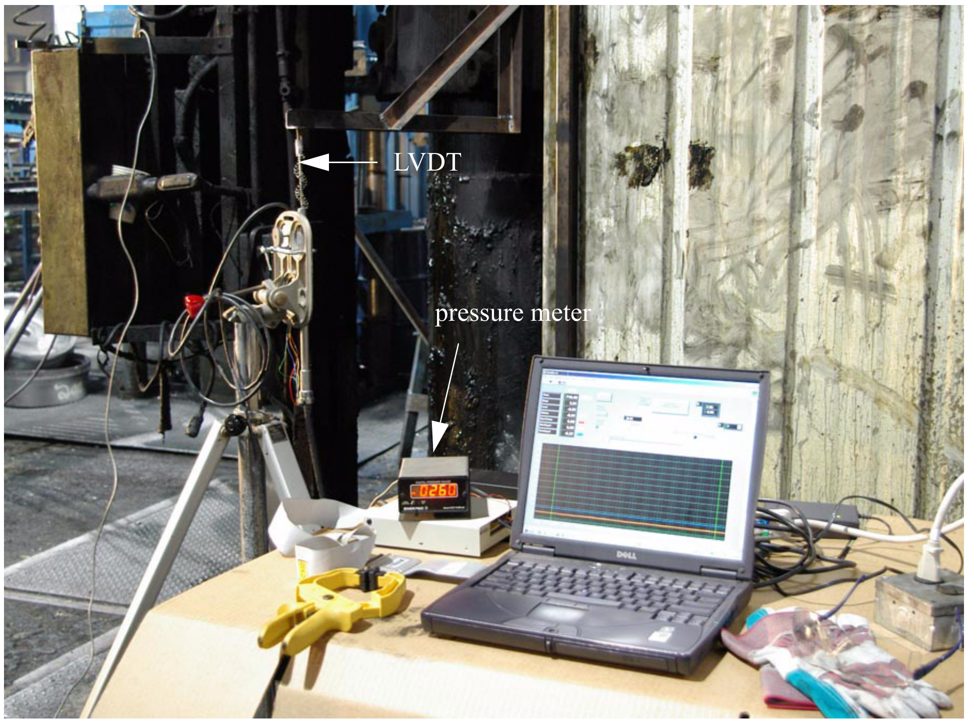


FIGURE 162. Experimental set-up: LVDT, pressure meter, and laptop data acquisition system.

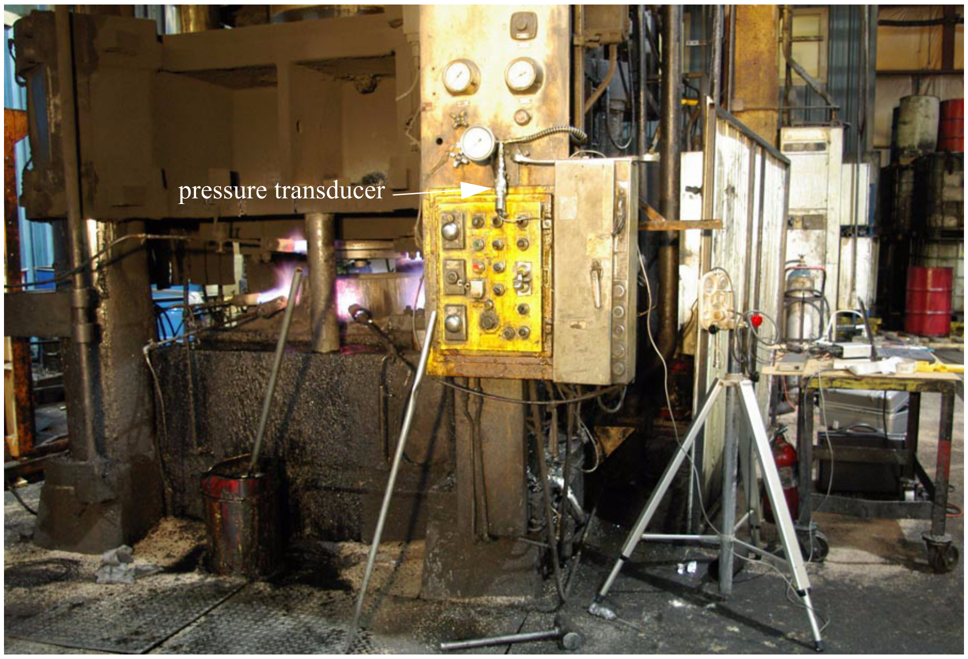


FIGURE 163. View showing front of the forging press and pressure transducer.

The forging operation is basically a three step process. A billet at approximately 450ºC (842ºF), measuring 203.2 mm (8 in) in diameter by 467.36 mm (18.400 in) long is place upright in the center of the bottom die in a hydraulic press. The top die is run downward until the maximum press capacity is achieved (approximately 34.47 MPa or 5000 psi). The crosshead is then raised and the part ejected so a graphite lubricant at 74ºC (165ºF) can be applied. The part is reseated and the top die closed again with maximum pressure applied. The force is then released momentarily and the maximum applied again.The first application of force utilizes two side cylinders, each of 330.2 mm (13 inches) diameter. The later two operations add the center ram having a 889 mm (35 inch) bore. It should be noted at this point, that temperature measurements taken of the billet at the start (457ºC) and end of the forging operation(448ºC) decreased by less than 10ºC (15ºF) and as such if fairly close to isothermal.

The photographs that follow, Figure 164, show the blocking portion of the forging operation at four points and Figure 165 provides a view of the billet as it bulges or barrels.

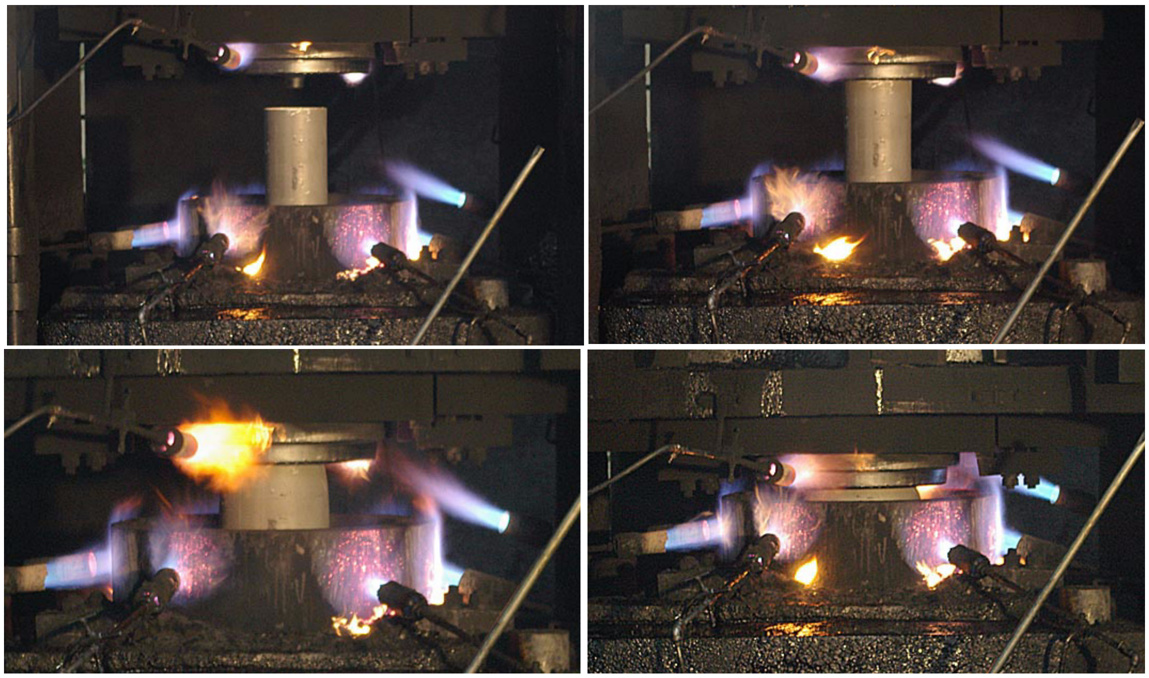


FIGURE 164. Photographs of billet compression during blocking portion of forging.

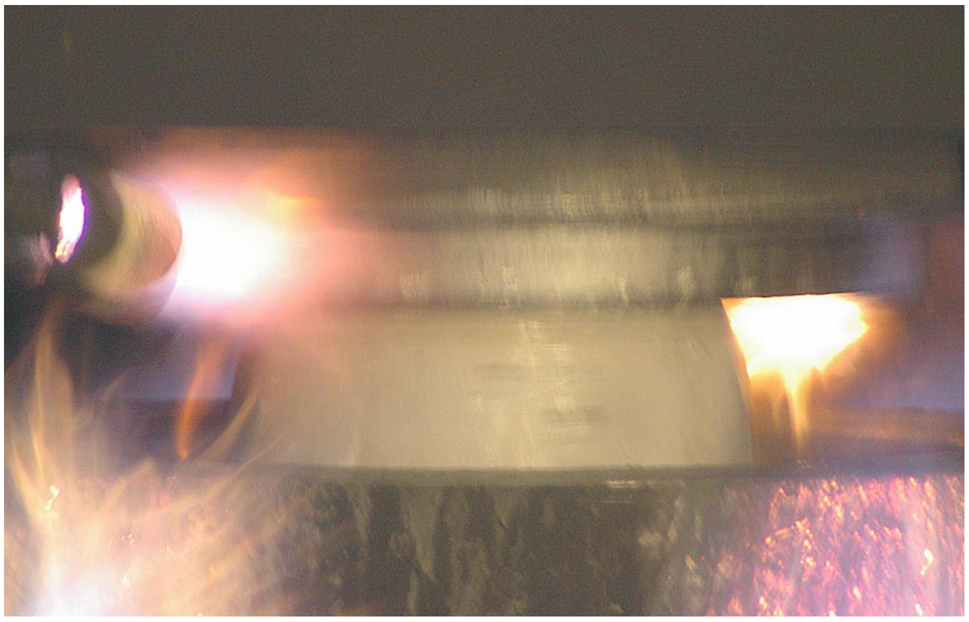


FIGURE 165. Close-up of billet (blurred due to movement), note bulged appearance.

After blocking, once the lubricant is applied, flames obscure the billet from view, shown in Figure 166. The excessive flames prevented the taking of useful photographs or infrared thermography (was attempted for billet temperature verification).

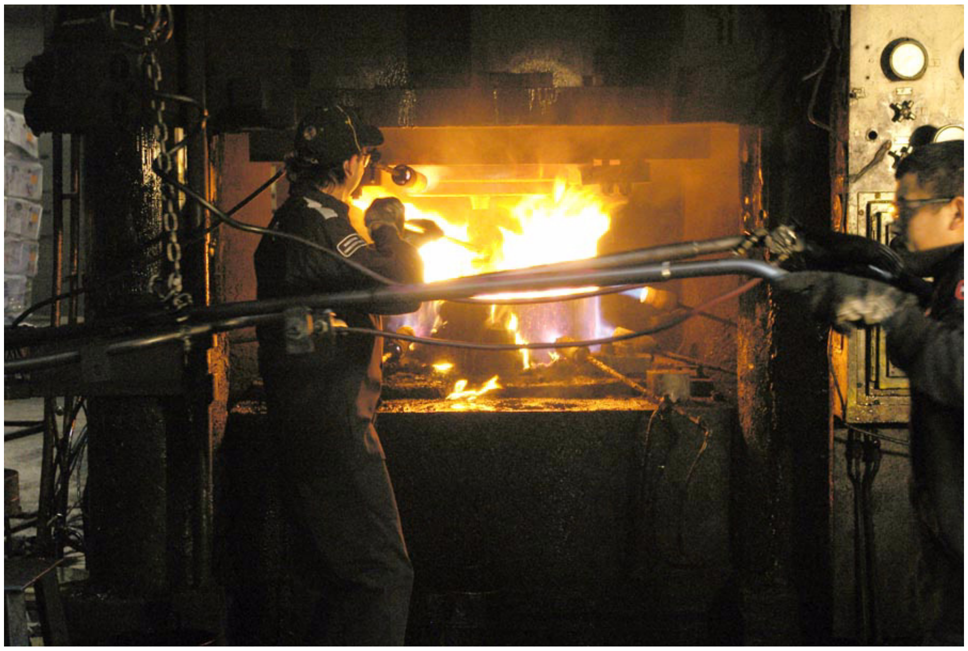


FIGURE 166. Lubricant application and excessive flashing produced.

Figure 167 and Figure 168 are views of the billet top and bottom, respectively, after completion of the forging operation.

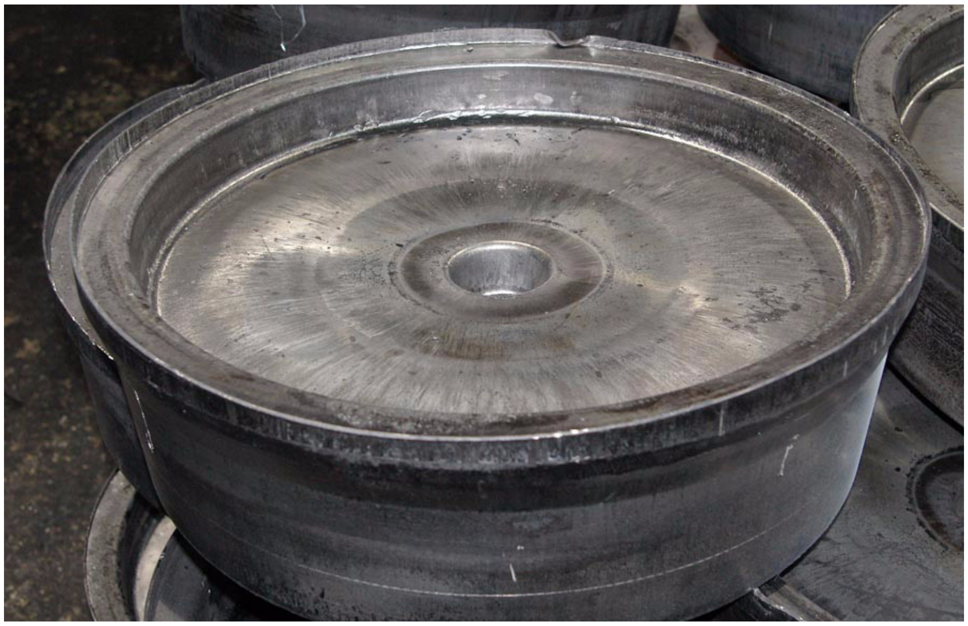


FIGURE 167. Top view of completed forging.

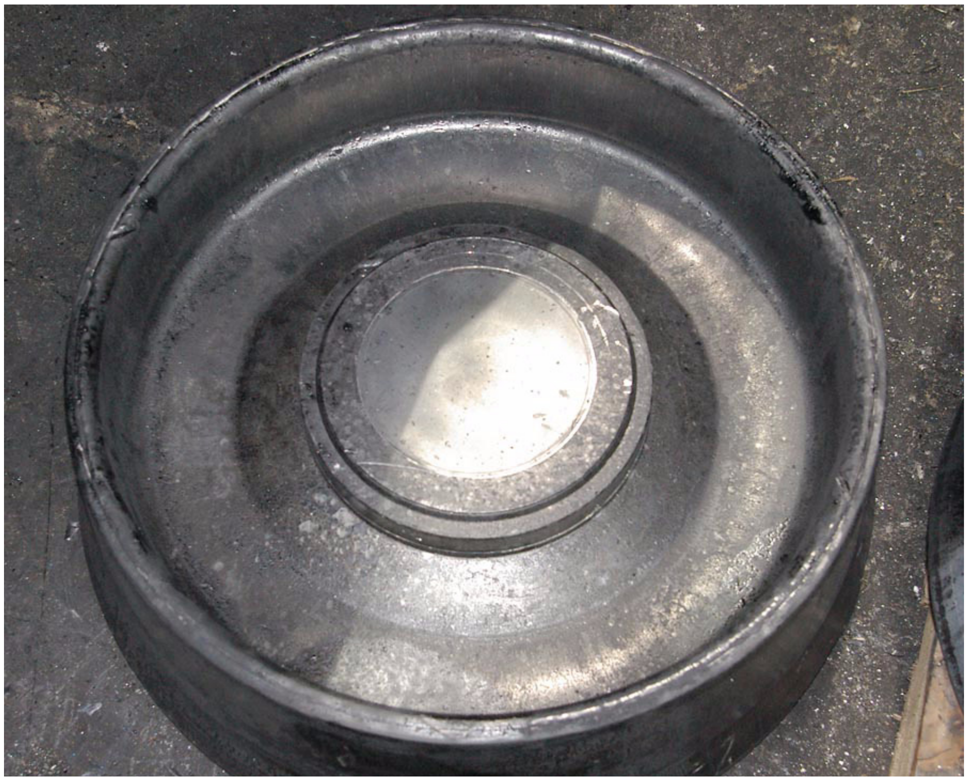


FIGURE 168. Bottom view of billet after forging.

Forging Pressures and Displacements

The process is very consistent with Figure 169 showing the pressure-displacement curves for three consecutive wheels. Figure 170 provides the pressure and displacement as it varied through time for a single wheel (Test 2).

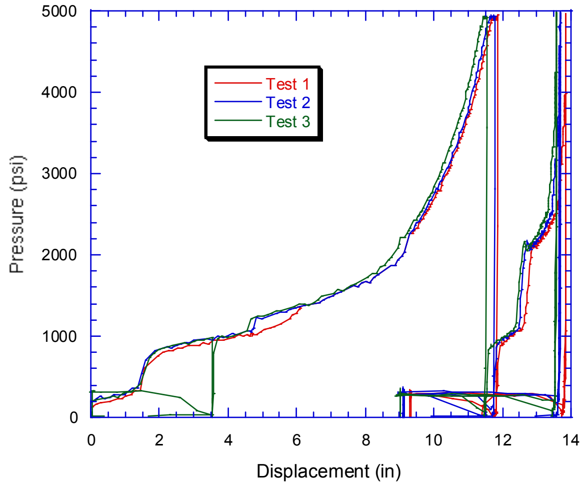


FIGURE 169. Pressure/displacement curves for three consecutive wheels, note consistency of the operation.

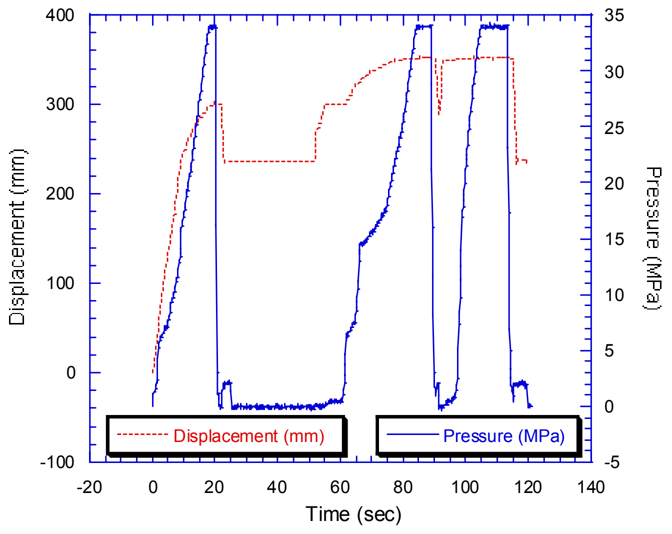


FIGURE 170. Pressure and displacement as a function of time for the 16 inch wheel forging (Test 2).

For later comparisons with the models developed, the displacement along with the forging force necessary to produce deformation is required. The measured hydraulic pressure is converted to force using the effective ram diameters and their areas. The blocking operation only involves the side cylinders with the remaining operations also utilizing the center ram. Only the second test is shown, due to the process consistency. The 2nd and 3rd operations essentialy produce the sample force-displacement relations, so they plot on top of each other.

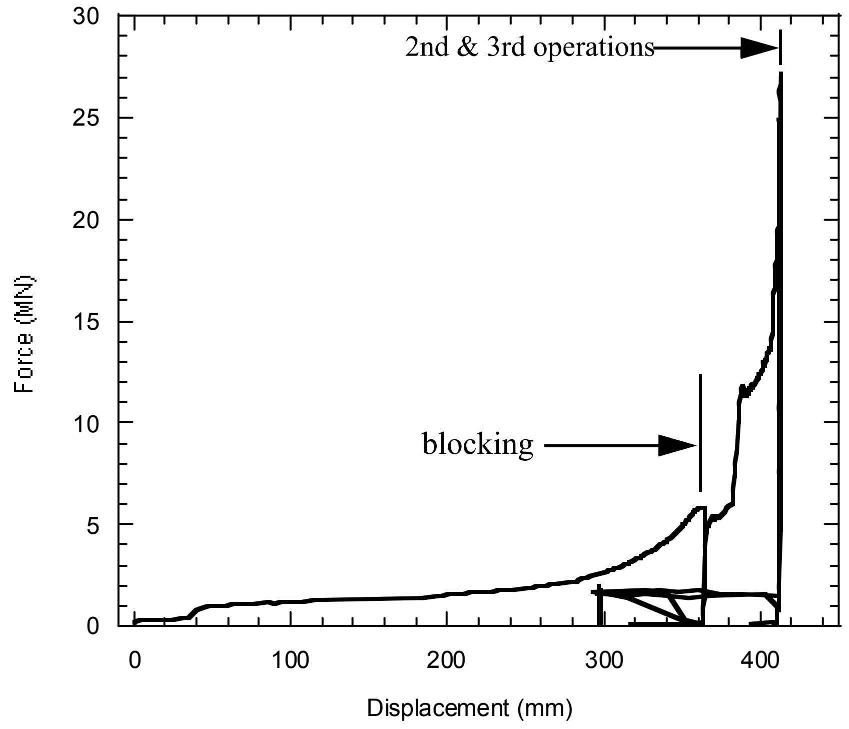


FIGURE 171. Force versus displacement through the entire forging operation. The second and third applications of pressure basically follow the same path.

Billet Condition, Post Forging Operations

Figure 172 shows the billet in crossection after completion of the forging operations. The crossection has been polished and etched by swabbing using Tucker's reagent (45 mL HCl (conc), 15 mL HNO_3 (conc), 15 mL HF (48%), 25 mL H_2O). No flow lines are evidenced due to the temperature used during forging.

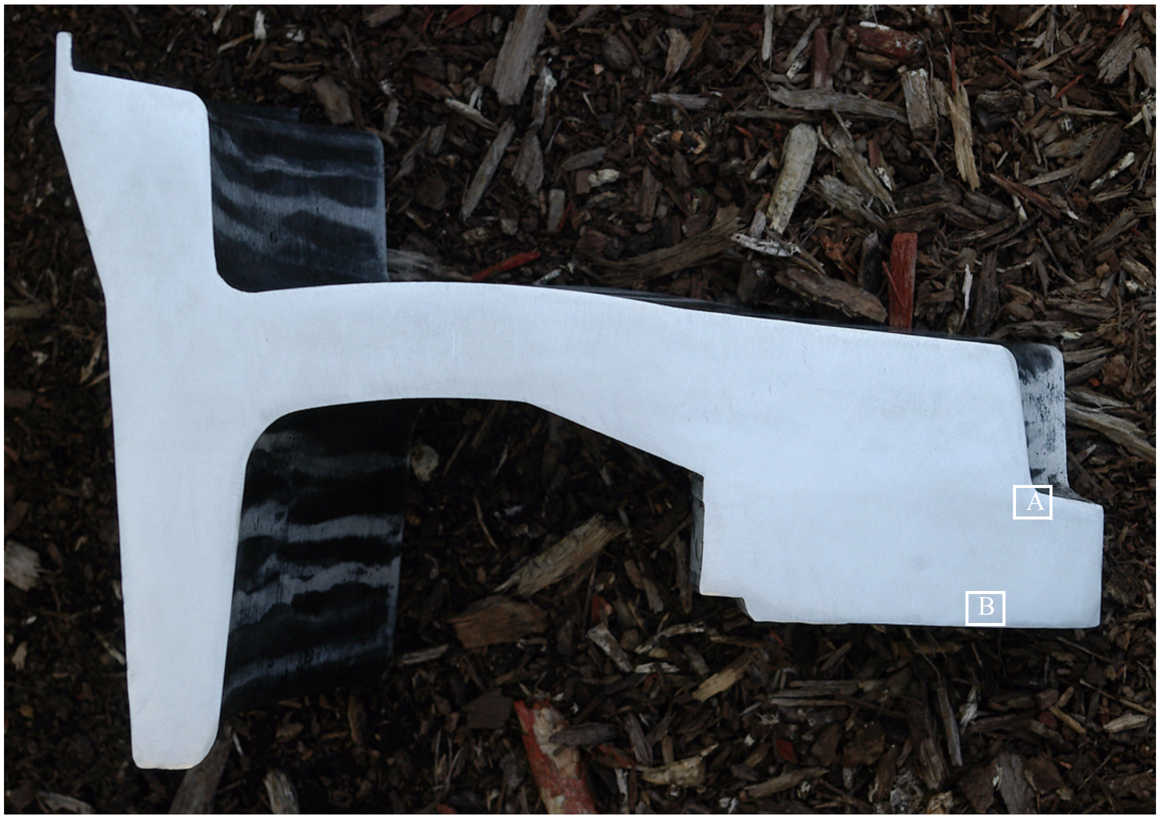


FIGURE 172. Half crossection of billet after completion of forging operation, polished and then etched using Tucker's reagent. Metallurgical samples removed from areas A and B.

Metallurgical samples were removed from two areas in an effort to establish differences in microstructure produced as the result of differing stress histories experienced within the forging. These areas were specifically chosen due to preliminary FEA results suggesting the highest stresses are produced in area A and the lowest in area B. The samples were mounted, polished, and etched (Keller's reagent: 2 mL HF (48%), 3 mL

HCl (conc), 5 mL HNO_3 (conc), 190 mL H_2O) by immersion (10 s). The photomicrographs shown in Figure 173 evidence essentially the same microstructure for both areas.

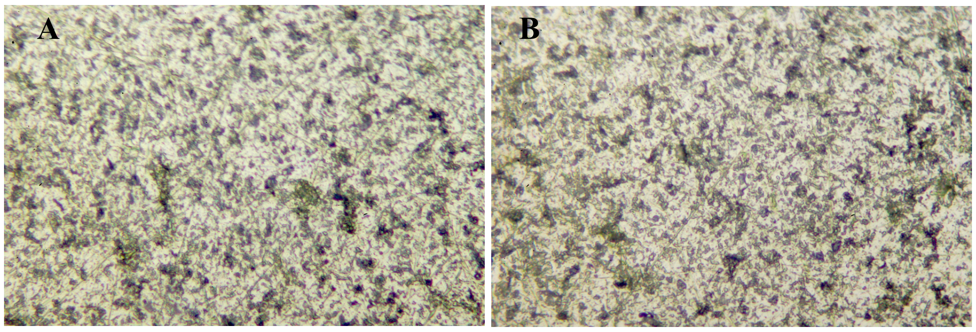


FIGURE 173. Photomicrographs of microstructures found for areas A (left) and B (right). MAG: 400X, etchant: Keller's reagent.

Microhardness readings were also taken in each area and are provided in Table 11. The Brinell values are conversions from Knoop and as such are only very rough estimates [71].

TABLE 11. Microhardness measurements, 3 taken per area.

Area	Knoop (100 g)	Brinell (converted from Knoop)
A	47.5, 47.0, 47.0	36
B	47.0, 46.5, 47.0	36

8.2 Initial Calculations

As an initial gauge of forces and pressures involved, simple hand calculations are shown below. The calculations are carried using English units as Weld Racing prefers, SI is shown for comparison to the models developed herein. Table 12 provides some of the estimated operating conditions along with other relevant information.

TABLE 12. Estimated operating conditions.

Parameter	Dimension
Initial Billet Height (h_o)	18.40 in
Initial Billet Diameter (d_0)	8 in
Final Billet Diameter (h_f)	20 in
Approximate Billet Temperature	845ºF
Ram Velocity (v)	1.0 in/s
Ram Effective Diameter (d_r)	1 ram @ 35 in, 2 rams @ 13 in
Reference Stress (C)	5.37 ksi @ 500ºC & 0.5 strain [72]
	7.25 ksi @ 400ºC & 0.5 strain [72]
Strain Hardening Exponent (m)	0.17 @ 500ºC & 0.5 strain [72]
	0.16 @ 400ºC & 0.5 strain [72]
Multiplying Factor for Forces, Q_c	5 to 8 [73]
Coefficient of Friction (μ)	0.1-0.2 [74]

Initially, the average height during the operation is found by assuming constant volume,

$$\text{average height} = h_{ave} = \frac{\text{volume}}{A_t} = \frac{\frac{\pi d_i^2}{4} h_i}{\frac{\pi d_f^2}{4}} = \frac{\frac{\pi(8\text{ in})^2}{4}(18.40\text{ in})}{\frac{\pi(20\text{ in})^2}{4}} = 2.944\text{ in} \quad \text{(EQ 131)}$$

followed by the average strain

$$\text{average strain} = \varepsilon_{ave} = \log\frac{h_0}{h_{ave}} = \log\frac{18.40\text{ in}}{2.94\text{ in}} = 6.26 \quad \text{(EQ 132)}$$

and average strain rate.

$$\text{average strain rate} = \dot{\varepsilon}_{ave} = \frac{v}{h_{ave}} = \frac{1.0\text{ in/s}}{2.94\text{ in}} = 0.34\text{ /s} \quad \text{(EQ 133)}$$

The flow stress is then found based on the power law equation.

$$\text{flow stress} = \sigma_f = C\dot{\varepsilon}^m = (5370\text{ psi})(0.34)^{0.17} = 4470\text{ psi} \cong 30.8\text{ MPa} \qquad \text{(EQ 134)}$$

At this point, the forging press force required can be found based on the flow stress, the projected area of the finished product, and a multiplying estimating factor related to the complexity of the part.

$$\text{force required} = P = \sigma_f Q_c A_t = (4470\text{ psi})(5)(314.16\text{ in}^2) = 7.02E6\text{ lbf} \qquad \text{(EQ 135)}$$

$$7E6\text{ lbf}{=}3500\text{ tons} \cong 31.13\text{ MN} \qquad \text{(EQ 136)}$$

Based on the press force required and the size of the ram used an estimate of hydraulic pressure is found.

$$\text{hydraulic pressure} = \frac{P}{\text{ram area}} = \frac{P}{\frac{\pi d_r^2}{4}} = \frac{7E6\text{ lbf}}{\frac{\pi(35\text{ in})^2}{4} + 2\frac{\pi(13\text{ in})^2}{4}} = 5702\text{ psi} \qquad \text{(EQ 137)}$$

It should be noted that this produces only a very rough estimate. As suggested by Schey [75], considerable computational effort must be expended in determining optimum die configuration. It is generally assumed that optimization must be determined by iteration.

The die pressure can also be estimated using

$$\text{die pressure} = \sigma_f Q_c = (4470\text{ psi})5 = 22.3\text{ ksi} = 154\text{ MPa} \qquad \text{(EQ 138)}$$

which is normally kept below 50 ksi (350 MPa) for dies used in aluminum forging [76]. The above, assumes a billet temperature of 975ºF (500ºC) due to the handbook constants available. If the actual temperature during forging decreases to around 750ºF (400ºC) the resulting flow stress would be approximately 6100 psi which translates to a force of over 9580 kips requiring greater than 7800 psi hydraulic pressure. The press capacity was reported as 3500 tons, and as such, if the temperature decreases appreciably, the above calculations would suggest that the press would not be sufficient for production of the wheel. The lower value corresponds closely with the values noted on an installed pressure gauge.

9. Case Study: Verification of Developed Approach

At this point the manufacturing process itself has been measured. What follows, describes the finite element model and its basic construction common to both the conventional and ANN-based approaches. As was the case with simple compression, the wheel model is axisymmetric and composed of CAX4R elements (4-node linear, quadrilateral, reduced integration, hourglass control). The basic material constants are also unchanged. Due to previous experience and its superior performance, only the tabular conventional approach is tested against the 15-3 neural network previously utilized. The upper and lower dies are modeled as analytically rigid bodies with the lower fixed (encastre) and the upper constrained except in the up or down direction (U1=UR2=UR3=0). The center axis of the billet is also constrained to maintain x-symmetry.

The models that follow were principally run at the National Center for Supercomputing Applications facility on an IBM POWER4 p690 supercomputer. The supercomputer is a cluster of 12 IBM eServer p690 UNIX systems and has been online since November 2002. The POWER4 p690 is comprised of 384 1.3 GHz processors with a total of 1.5 terabytes of memory [76]. The CPU times, when noted below, are estimates based on batch reports from this system, typically utilizing 2 processors and loop threading. For purposes of comparison, the simple compression models were run on a local Pentium machine (2.8 GHz, 1 Gbyte RAM) running Linux and generally took less than 60 minutes.

In an attempt to mirror the manufacturing process, the model consists of two steps. The first being blocking and the second the full compression of the billet after lubricant application. The blocking step assumes a coefficient of friction of 0.3 and the lubed step 0.1. These are estimates based on Weld Wheels' experience and are consistent with published handbook data [77]. In order to drive the model's upper die at an appropriate rate, the crosshead displacement over time, as shown in Figure 174 and Figure 175, is used to calculate the downward velocity for both steps.

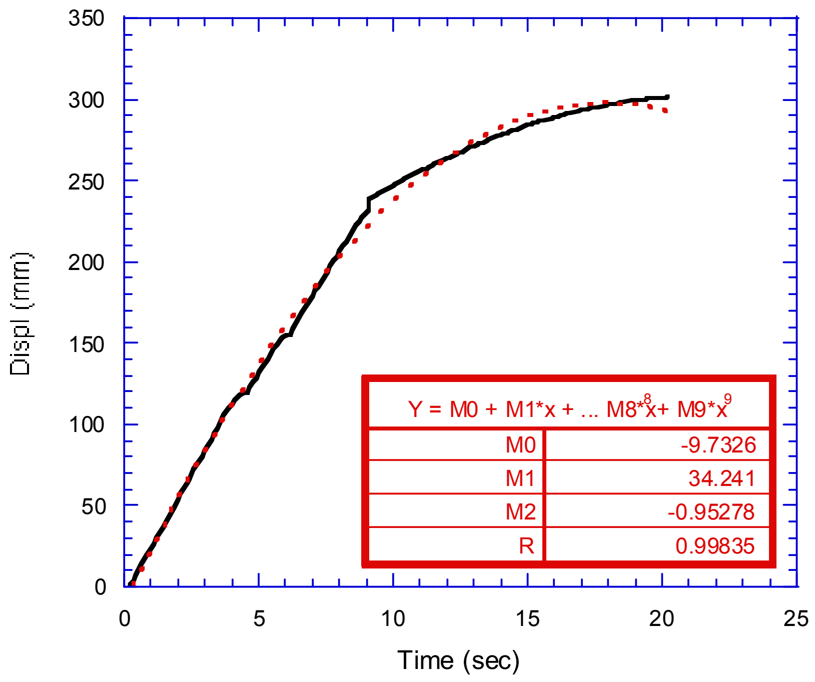


FIGURE 174. Displacement of top die over time during the blocking operation with polynomial curve-fit used for determining velocity profile.

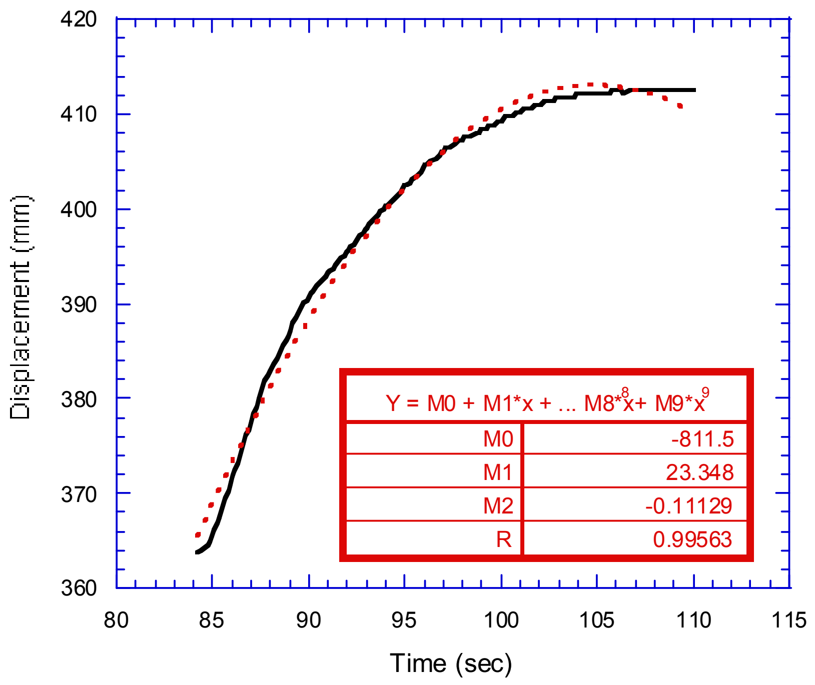


FIGURE 175. Displacement of top die over time up to complete closure, 2nd step, with polynomial curve-fit used for determining velocity profile.

The finite element code has the ability to accept tabular data to represent changes in velocity imposed on the upper die reference point. Taking the derivative of the curve-fits generated for both steps:

$$\frac{\mathrm{d}y_{step1}}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}(-9.7326 + 34.241t - 0.95278t^2) = -1.90556t + 34.241 \qquad \text{(EQ 139)}$$

$$\frac{\mathrm{d}y_{step2}}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}(-811.5 + 23.348t - 0.11129t^2) = -0.22258t + 23.348 \qquad \text{(EQ 140)}$$

and calculating velocities at several convenient points, results in Table 13 and Table 14. The code adjusts the velocity based on an amplitude or multiplier at specific times. For both steps, the curve-fits actually result in slightly negative velocities near the end. These negative velocities were, in practice, replaced by a an amplitude of 0.001 m/s to continue the application of a downward force, matching the actual process.

TABLE 13. Velocity profile and amplitudes for blocking, Step-1.

Time (s)	Velocity (mm/s)	Velocity (m/s) rounded off	Amplitude
0	34.24	0.034	1
3	28.52	0.029	0.83
6	22.81	0.023	0.67
9	17.09	0.017	0.5
12	11.37	0.011	0.33
15	5.66	0.006	0.17
18	-0.06	0.000	0.001
20	not used	not used	0.001

TABLE 14. Velocity profile and amplitudes for blocking, Step-2.

Time (s)	Velocity (mm/s)	Velocity (m/s) rounded off	Amplitude
85	4.43	0.004	0.13
90	3.32	0.003	0.1
95	2.2	0.002	0.06
100	1.09	0.001	0.03
105	-0.002	0.000	0.001
110	not used	not used	0.001

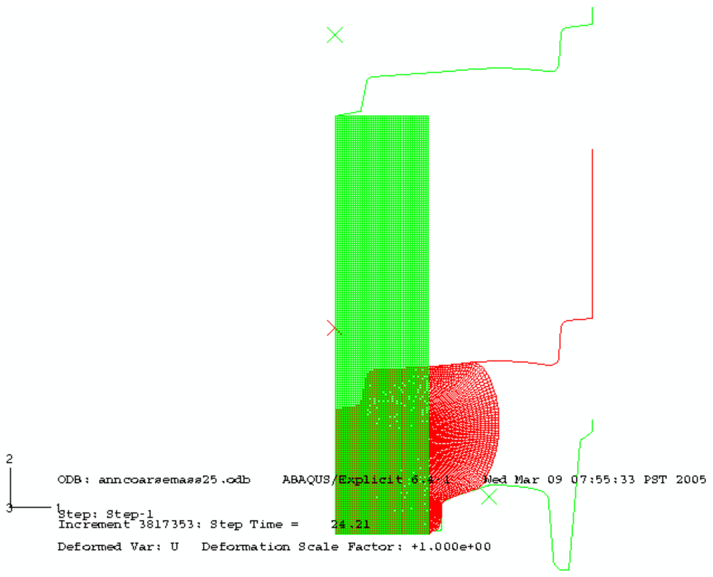
In anticipation of the severe plastic deformation that is to occur, the default settings for adaptive meshing are implemented initially, re-meshing attempted every 5 increments, with 3 potential sweeps.

The billet is modeled as isothermal, in one respect because the measured temperatures are consistent with isothermal behavior, and additionally due to difficulties in including adiabatic heat and/or heat transfer in the model. The finite element code does not provide for modeling adiabatic heat rise when VUMATs are called requiring the VUMAT itself to perform the calculations. Also, does not calculate heat transfer between analytically rigid bodies (the dies) as they do not posses mass. As will be demonstrated, even without accounting for temperature changes, the model demands significant computing resources.

Additionally, mass scaling was employed to reduce CPU time with the requisite checks on kinetic to internal energy carried out to ensure the amount of scaling was acceptable. Further modifications to the models are discussed as they become necessary.

9.1 Blocking Operation, Model Results, Step-1

Figure 176 shows the change in the aluminum billet from its initial condition to that achieved at the completion of the blocking operation.



```
ODB: anncoarsemass25.odb    ABAQUS/Explicit 6.4-1    Wed Mar 09 07:55:33 PST 2005
Step: Step-1
Increment 3817353: Step Time =    24.21
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

FIGURE 176. View of undeformed billet and the forging at the completion of blocking.

Initial attempts at modeling with a a relatively coarse mesh proved unsuccessful as the elements distorted excessively, crashing the analysis. The center-top indention very quickly produces significant plastic deformation, requiring an original mesh size of approximately 2.5 mm squares just to complete blocking. Using this mesh size, with adaptive meshing (frequency 5, sweeps 3), and a mass scaling factor of 25, the conventional model takes approximately 96 hours to complete to this point. The ANN-FEA model taking around 30% longer due to the subroutine calls.

Double precision execution was also employed due to the large number of increments necessary for producing a solution. It is necessary to employ double precision (64-bit word lengths) as opposed to single precision (32-bit word lengths) whenever greater than 300,000 increments are executed (all of the models that follow required more than 500,000 increments). The main drawback inherent in double precision is an increase in computing time of approximately 20 to 30% [78]. Approximately 550,000 increments were carried out to complete blocking.

Figure 177 and Figure 178 are plots of internal and kinetic energy for the entire model up to blocking. The ratio of maximum kinetic to maximum internal energy is something less than 0.025%, significantly under the 1% limit suggested previously.

The next sets of figures, Figure 179 through Figure 186, evidence the differences in stress distributions between the conventional-tabular based model and the ANN-linked FEA approach.

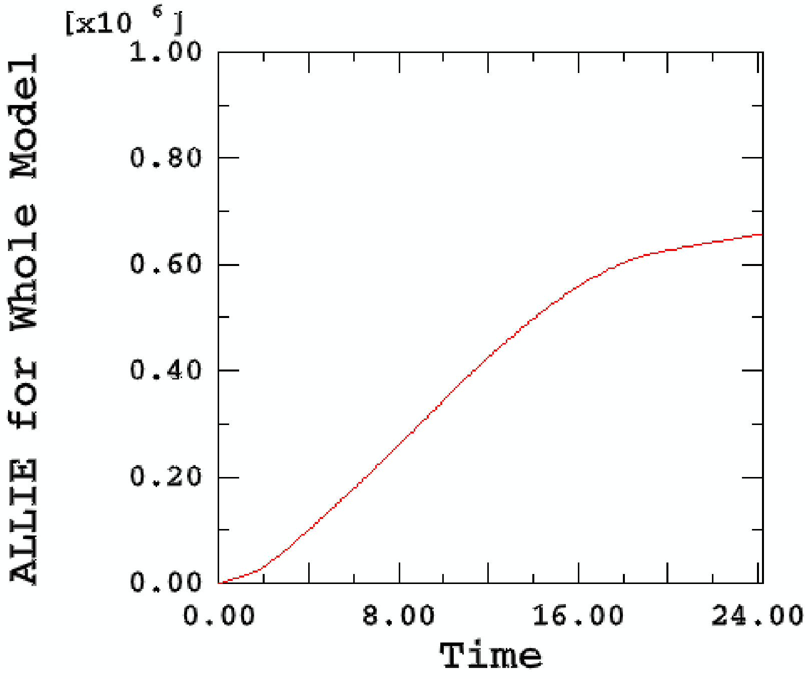


FIGURE 177. Plot of internal energy for the whole model to the end of blocking.

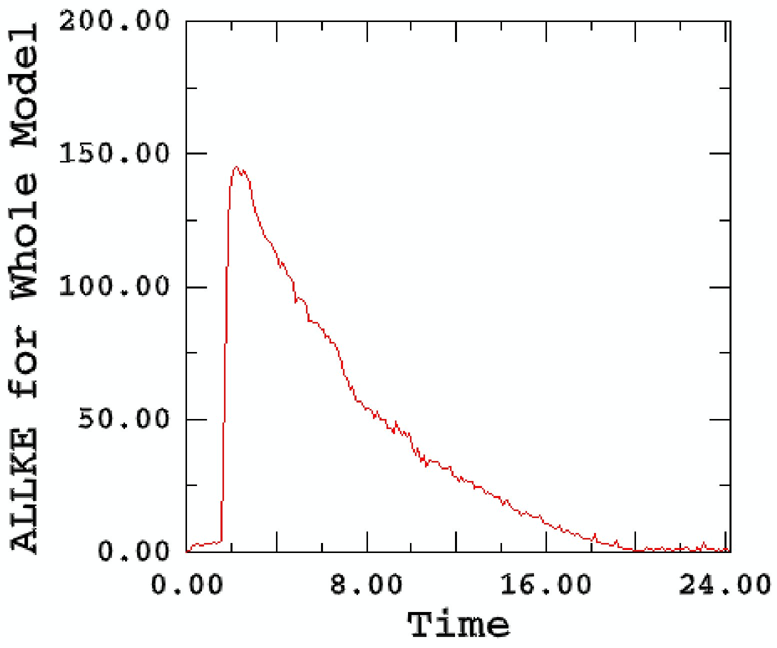


FIGURE 178. Plot of kinetic energy for the whole model to the end of blocking.

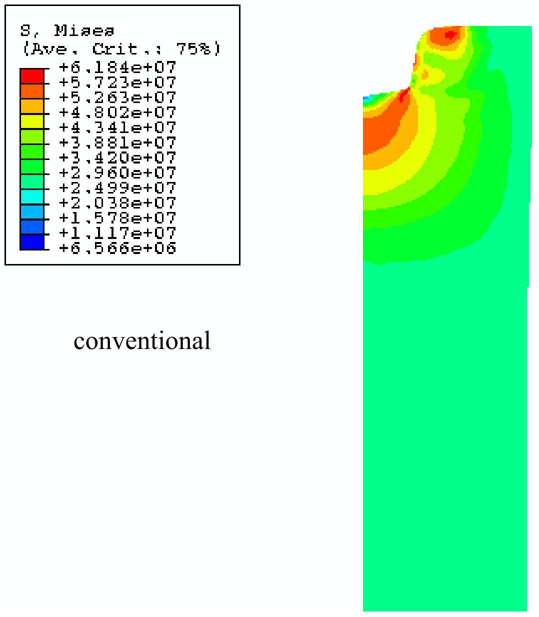


FIGURE 179. Comparison of von Mises' stress distributions after top-center indention near completion, conventional model.

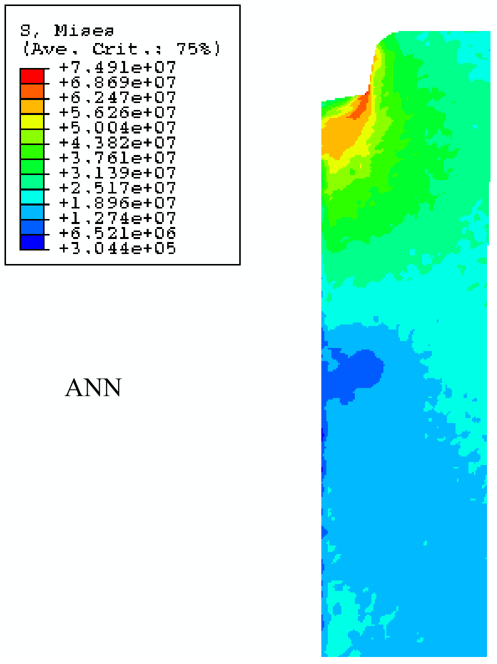


FIGURE 180. Comparison of von Mises' stress distributions after top-center indention near completion, ANN model.

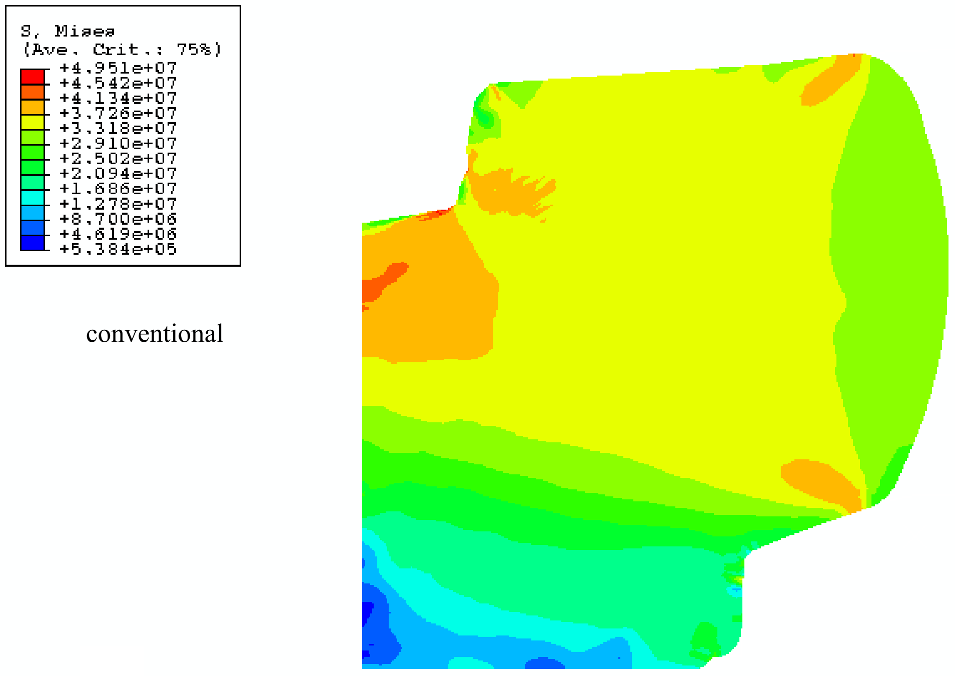


FIGURE 181. Stress distribution for the tabular-based conventional model at completion of blocking.

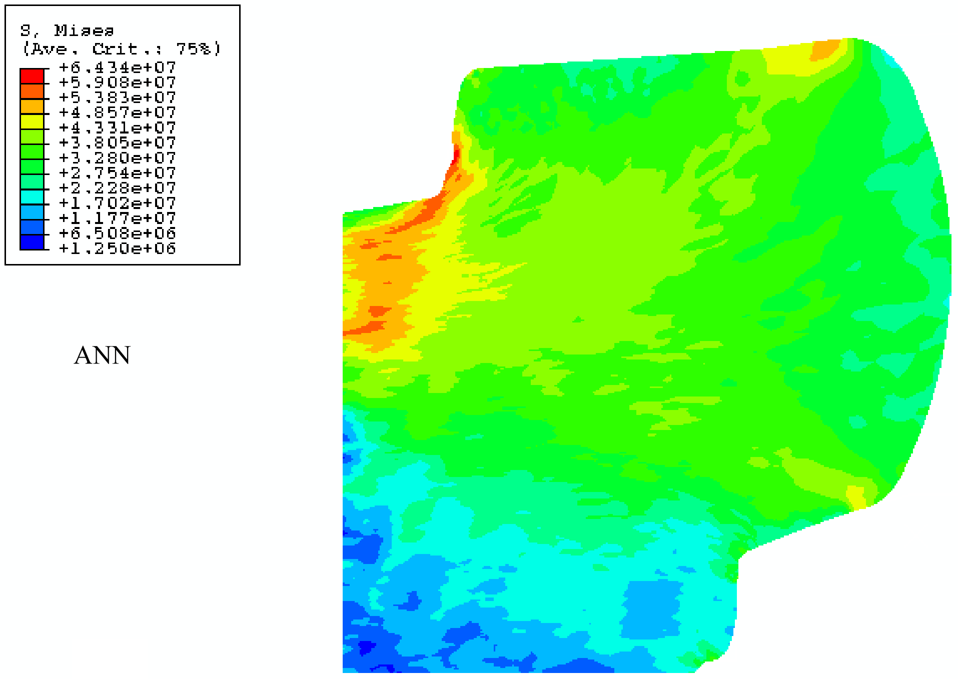


FIGURE 182. Stress distribution for the ANN-FEA model at completion of blocking.

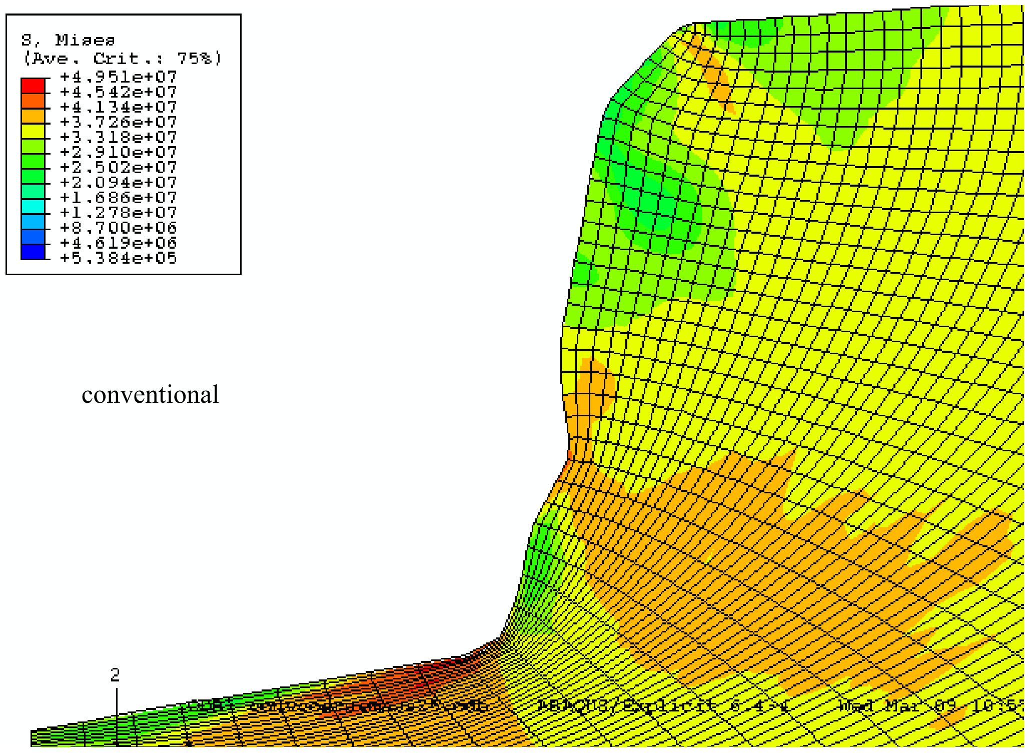


FIGURE 183. Close-up of center indention at completion of blocking, conventional model.

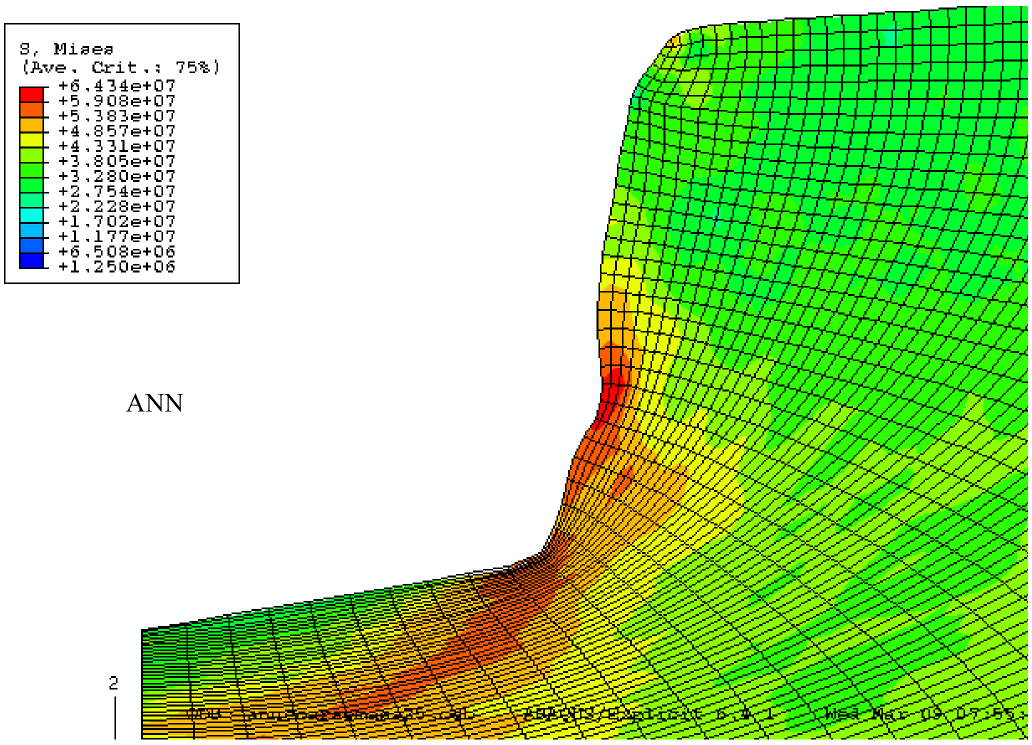


FIGURE 184. Close-up of center indention at completion of blocking, ANN model.

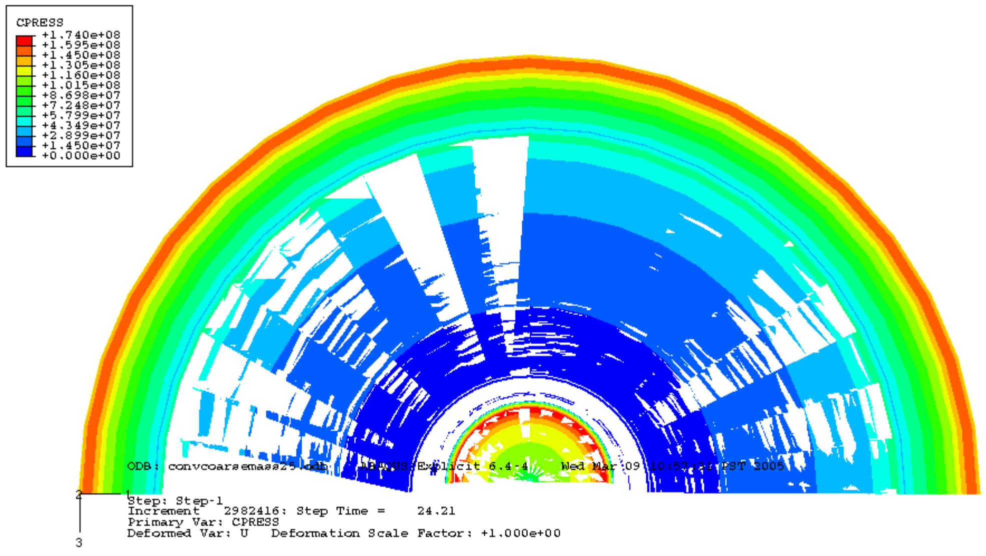


FIGURE 185. Surface pressure distribution at the end of blocking (i.e., die pressure), conventional model.

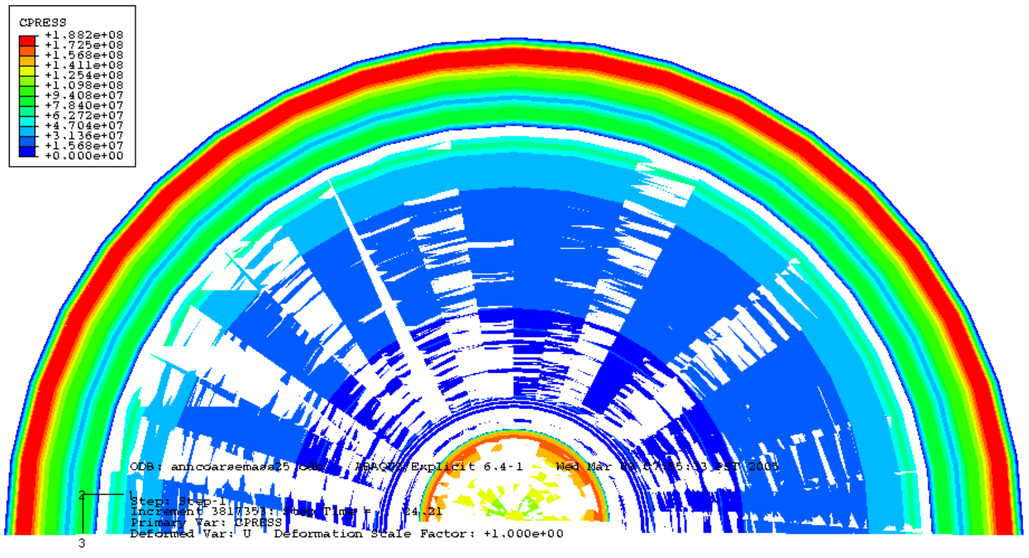


FIGURE 186. Surface pressure distribution at the end of blocking (i.e., die pressure), ANN model.

9.2 Blocking Operation, Finite Element Model vs. Data Acquired

Figure 187 provides a comparison of experimental, conventional FEA, and ANN-based FEA force-displacement curves, through and somewhat past blocking. Even though both the conventional and the ANN-based models are identical with the exception of the material model employed, the conventional model collapses shortly into Step-2. As shown in Figure 188, during most of the initial loading the ANN-based model predicts lower forces which more closely matches experimental values.

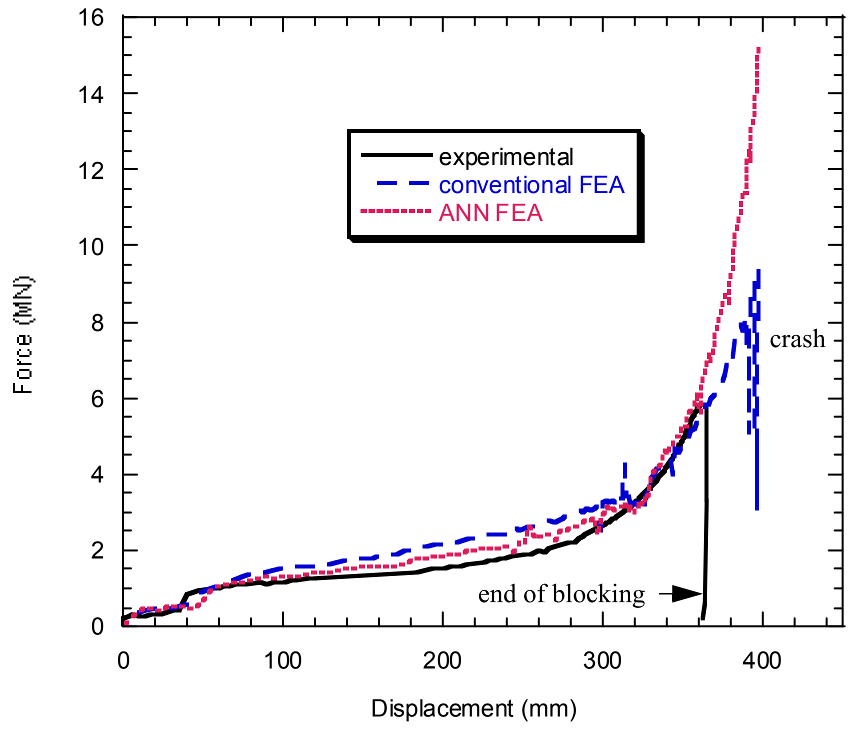


FIGURE 187. Comparison of measured forces to conventional and ANN-based FEA, run somewhat past blocking.

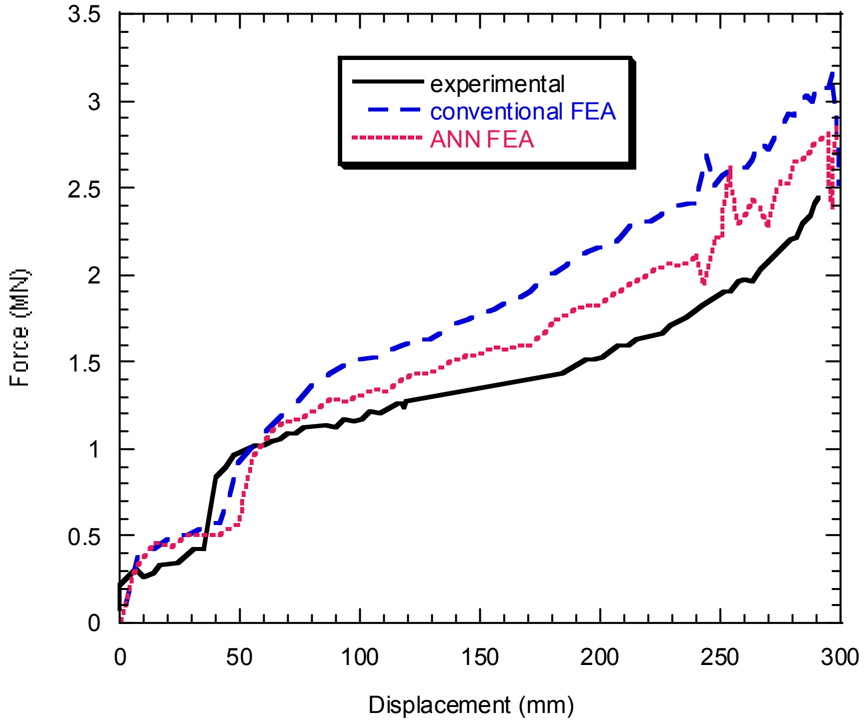


FIGURE 188. Comparison of measured forces to conventional and ANN-based FEA, approximately to end to blocking, Step-1.

9.3 Initial Attempts, Second Step Models

The initial attempts at continuing the analysis through the second step resulted in failure of the models due to excessive element distortions. As noted above, the conventional model failed shortly after completion of blocking while the ANN-based model continued to trend upward as would be expected. The ANN model also failed, but at a latter point in the analysis (see Figure 189).

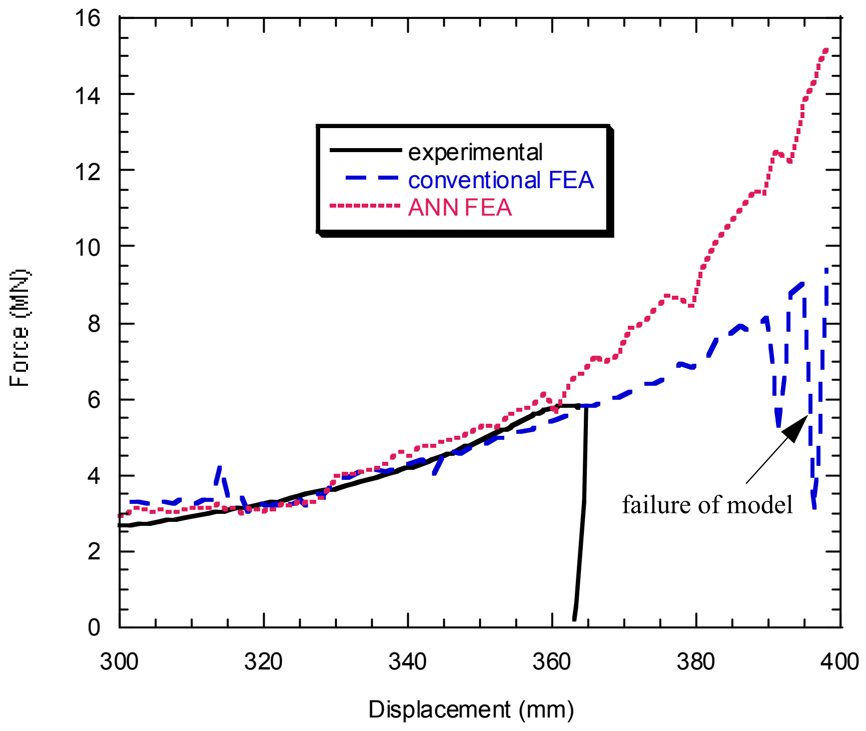


FIGURE 189. Comparisons at the end of blocking and into final compression; note failure of conventional based model.

In an effort to prevent the excessive element distortions, the adaptive meshing parameters were modified in Step-2 and both analyses restarted. The meshing frequency was changed to 1 (evaluate mesh at every increment) and the number of possible sweeps to 5. The restarts used approximately 150 hours and 200 hours of CPU time to reach the points shown, for the conventional and ANN-based models, respectively. Top die reaction forces as they vary through time are shown in Figure 190 and Figure 191. There is a sudden decrease in force indicating the model is collapsing. Figure 192 and Figure 193 provide contour plots of both models as they fail, this time due to element collapse. A close-up view is provided in Figure 194. Figure 195 compares these model forces against the experimental data obtained.

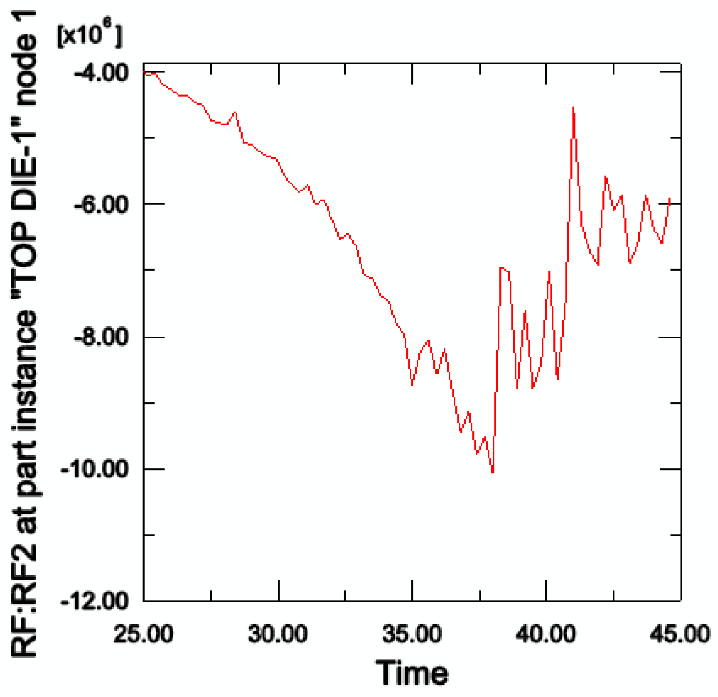


FIGURE 190. Top die reaction force through time for the conventional model; note sudden decrease in absolute force indicative of model failure

.

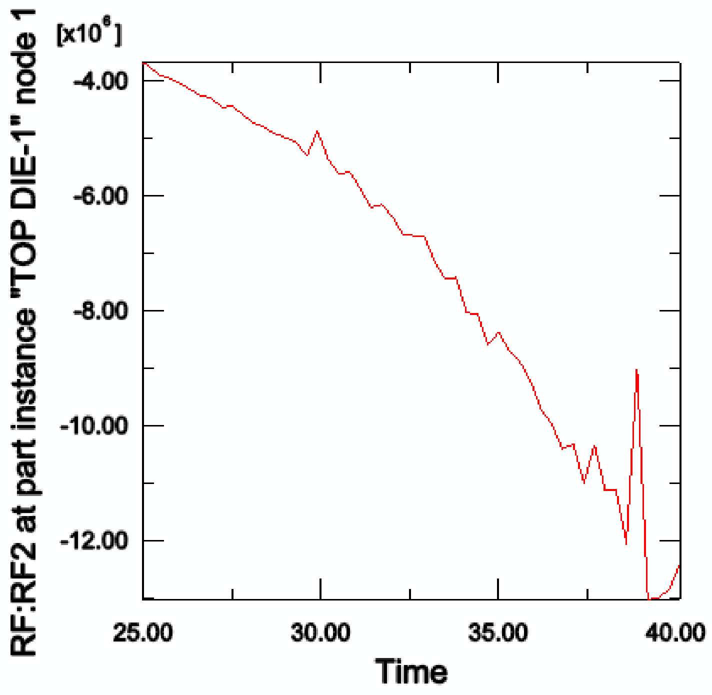
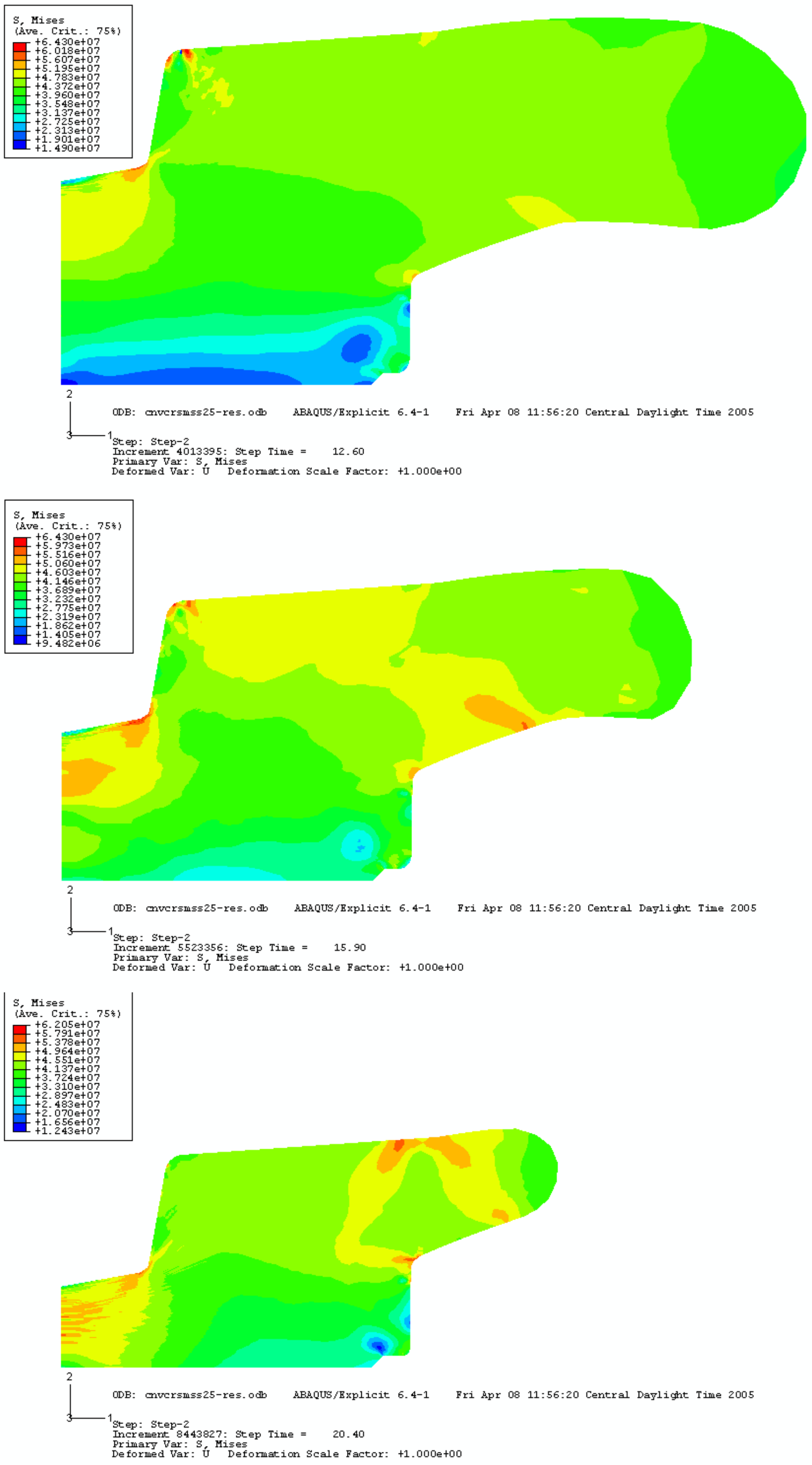


FIGURE 191. Top die reaction force through time for the ANN-based model; note sudden decrease in absolute force indicative of model failure.



```
ODB: cnvcrsmss25-res.odb    ABAQUS/Explicit 6.4-1    Fri Apr 08 11:56:20 Central Daylight Time 2005

Step: Step-2
Increment 4013395: Step Time =    12.60
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

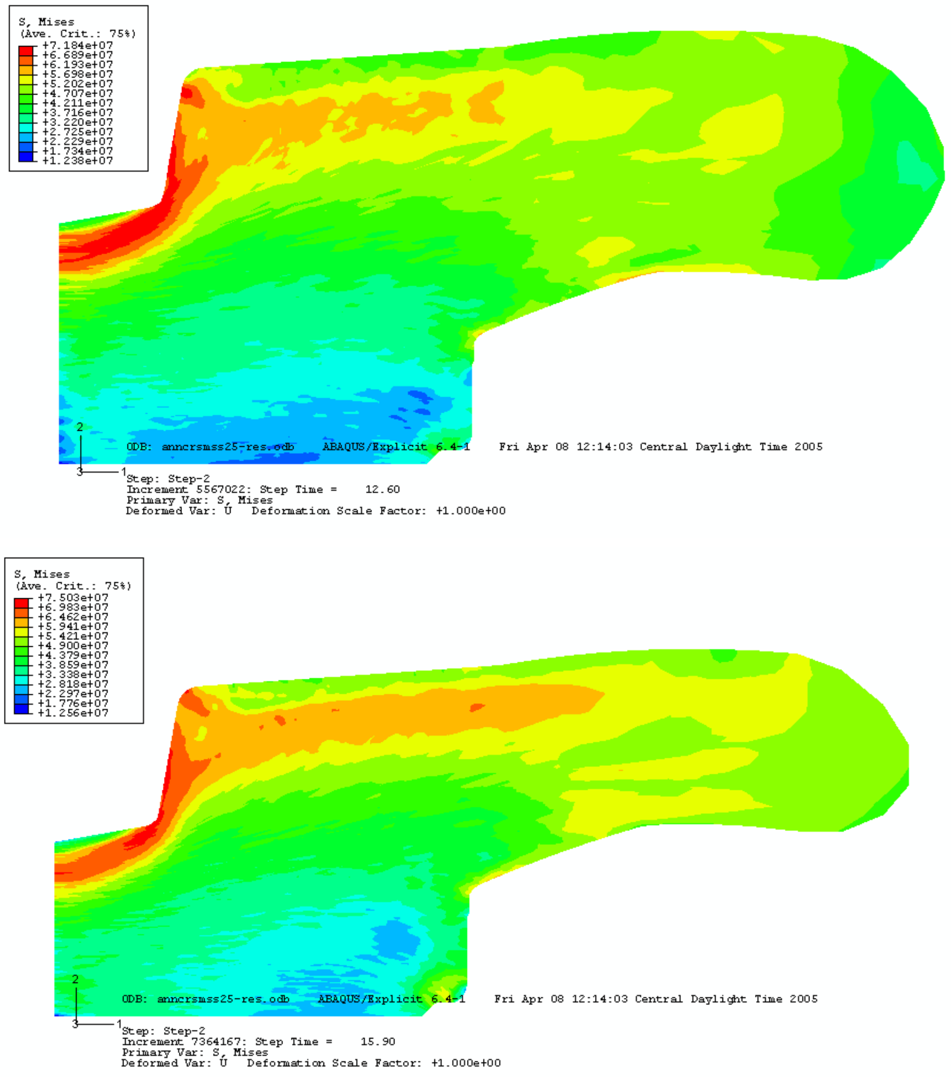
```
ODB: cnvcrsmss25-res.odb    ABAQUS/Explicit 6.4-1    Fri Apr 08 11:56:20 Central Daylight Time 2005

Step: Step-2
Increment 5523356: Step Time =    15.90
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

```
ODB: cnvcrsmss25-res.odb    ABAQUS/Explicit 6.4-1    Fri Apr 08 11:56:20 Central Daylight Time 2005

Step: Step-2
Increment 8443827: Step Time =    20.40
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

FIGURE 192. Views showing progression as conventional model fails due to elements collapsing.



```
ODB: anncrsmss25-res.odb    ABAQUS/Explicit 6.4-1    Fri Apr 08 12:14:03 Central Daylight Time 2005

Step: Step-2
Increment 5567022: Step Time =    12.60
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

```
ODB: anncrsmss25-res.odb    ABAQUS/Explicit 6.4-1    Fri Apr 08 12:14:03 Central Daylight Time 2005

Step: Step-2
Increment 7364167: Step Time =    15.90
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

FIGURE 193. Progression of ANN-FEA model as failure occurs, note drooping. Model was stopped once forces began to decrease.

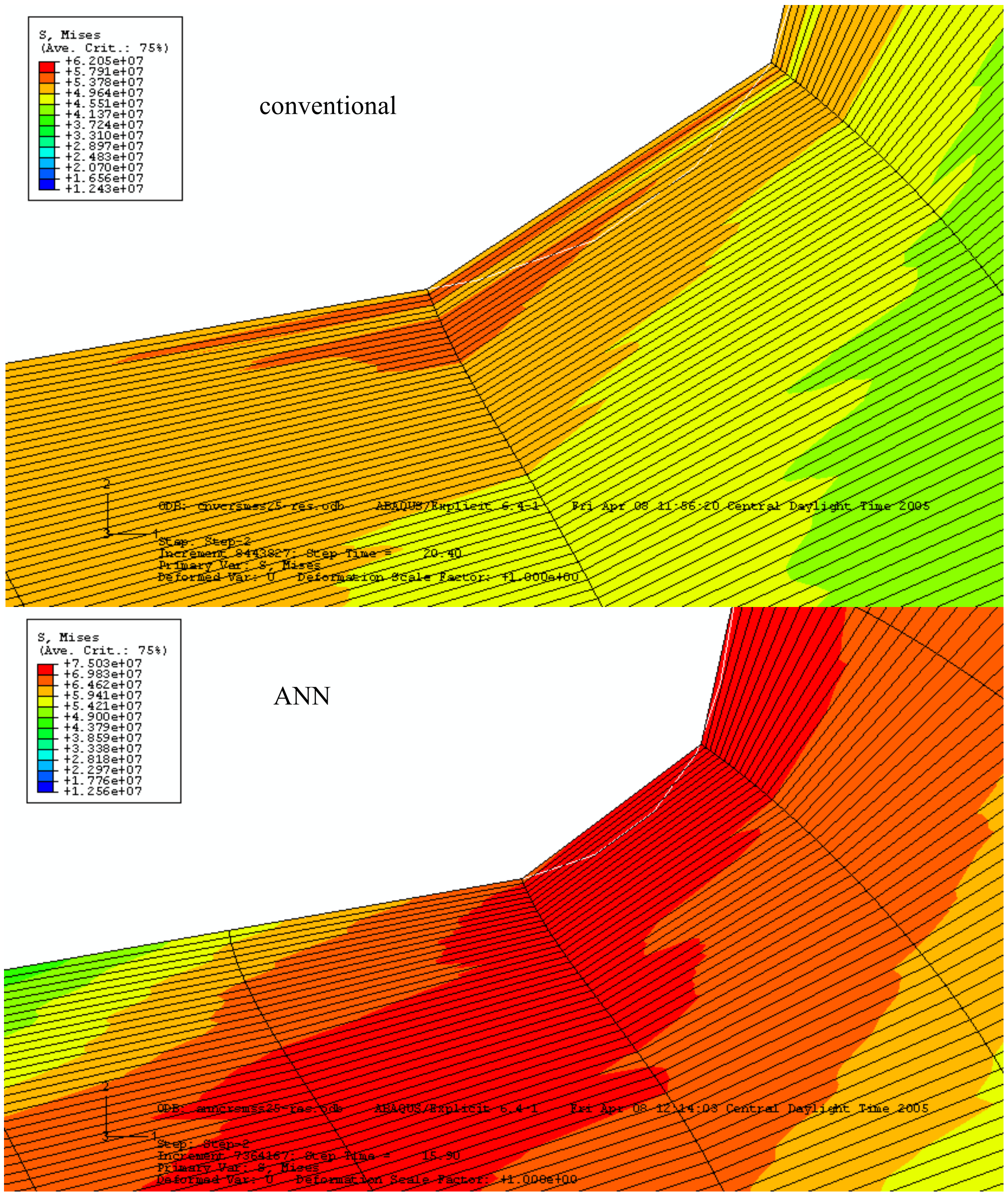


FIGURE 194. Close-up view showing severe element elongation as models begin to collapse; conventional and ANN-based.

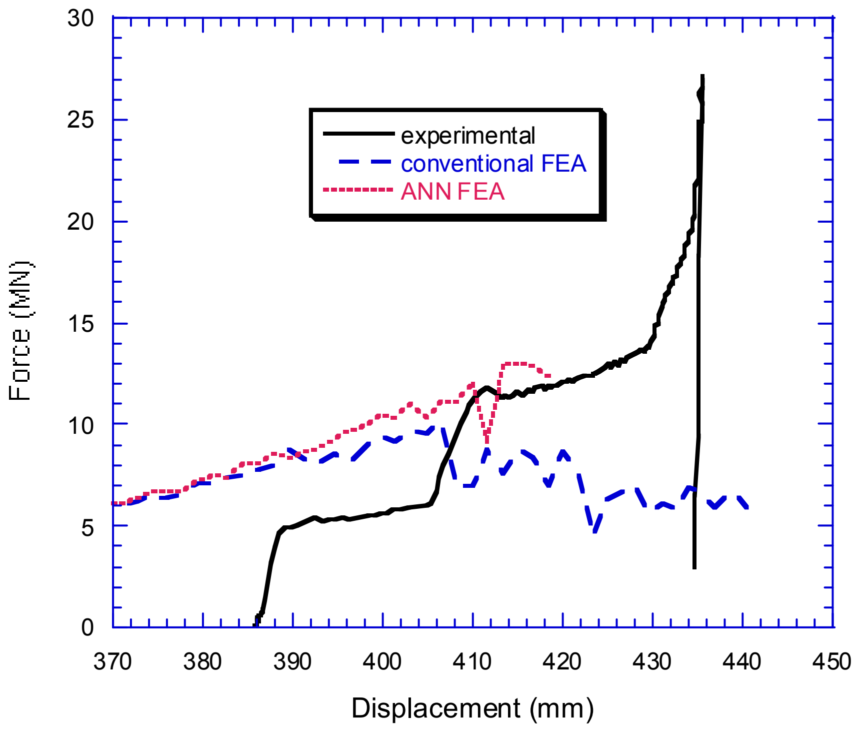


FIGURE 195. Comparisons into final compression, both models fail as elements collapse.

The finite element code does not allow modification of a mesh when attempting to restart analysis after partial completion. Therefore, to try out other meshes, the analysis has to be started from the beginning, obviously requiring significant CPU time. It was judged, based on the element elongations, that an initial mesh consisting of rectangles with higher aspect ratios, measuring 1 mm wide by 5 mm tall (see Figure 196), would allow excessive compression of the elements eventually resulting in significantly less elongation. Other meshes (e.g., 1.0 x 25, 1 x 15, 1 x 10) were attempted, but failed long before the end of blocking. In addition, the adaptive mesh frequency, from the beginning, was set at 1 with 5 sweeps. This was done to help improve the mesh prior to the start of the second step. The greater height has the added advantage of decreasing the degrees of freedom through the reduction of numbers of nodes, from 15,798 to 10,722, which ultimately uses less CPU time.

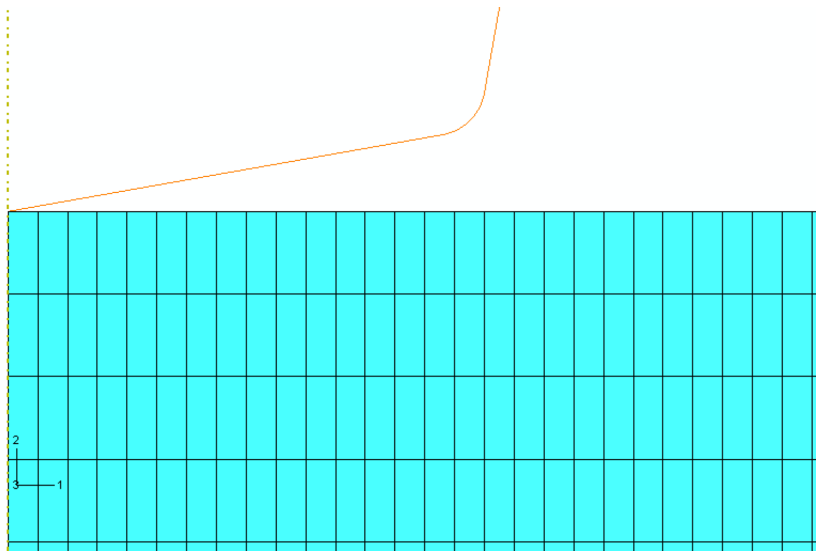


FIGURE 196. View of 1 mm x 5 mm mesh against top die.

The next set of figures, Figure 197 and Figure 198, shows the force-time curves and the progression of the billet through blocking and part way into the second step. Neither the conventional or the ANN-based models ran anywhere near completion before the model failed. Based on the force-time curves, the conventional model goes unstable at around 10 seconds with the ANN-based mesh losing integrity at approximately 11 seconds into the second step. The contour plots, Figure 199 through Figure 201, show the progression of each of the models and their subsequent failure as elements distort excessively.

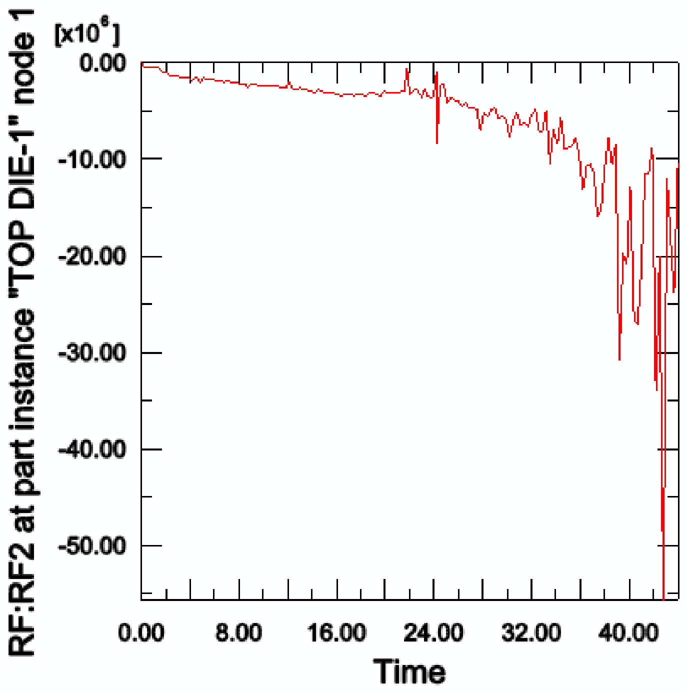


FIGURE 197. Top die reaction force through time for the conventional model. Model becomes unstable at around 35 seconds, or approximately 10 seconds into the second step.

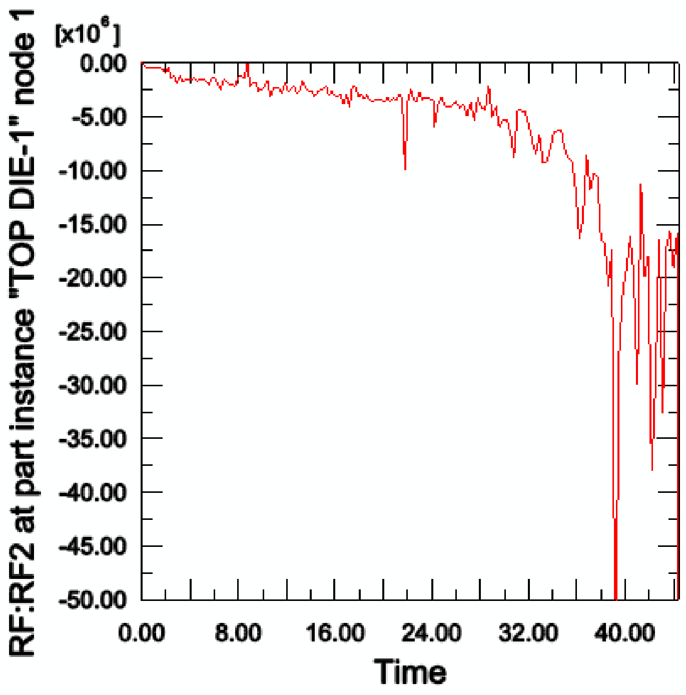
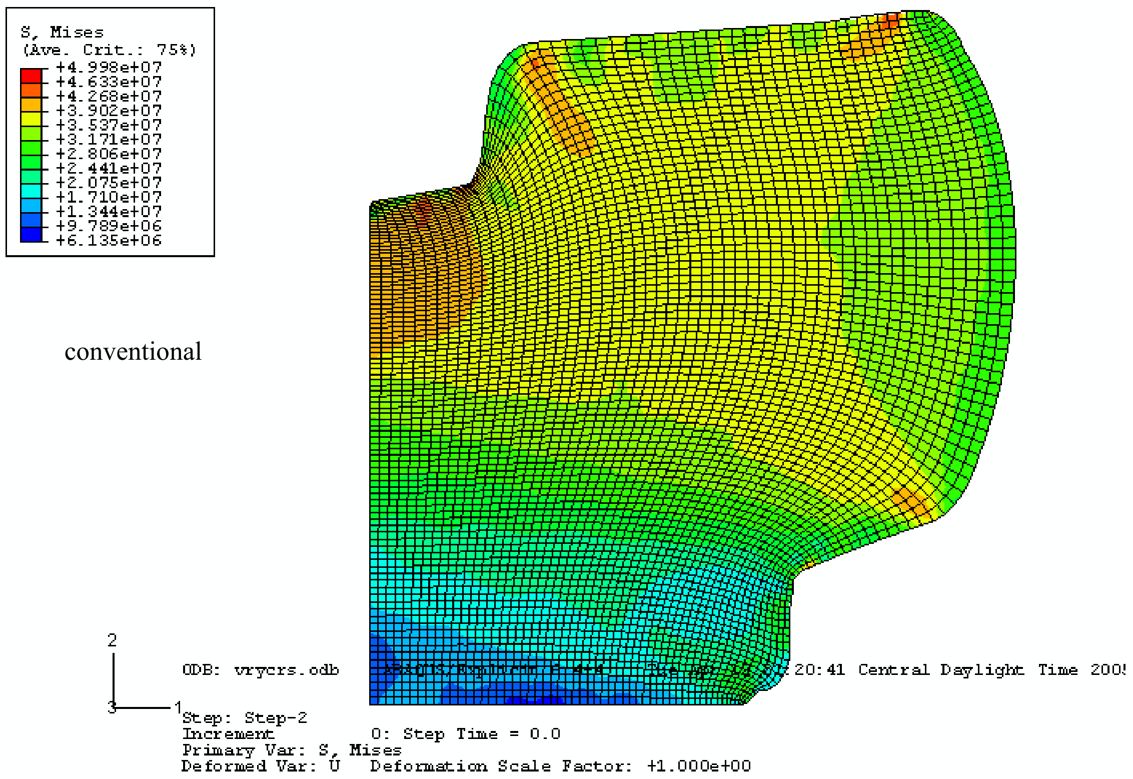


FIGURE 198. Top die reaction force through time for the ANN-based model. Model becomes unstable at around 36 seconds, or approximately 11 seconds into the second step.



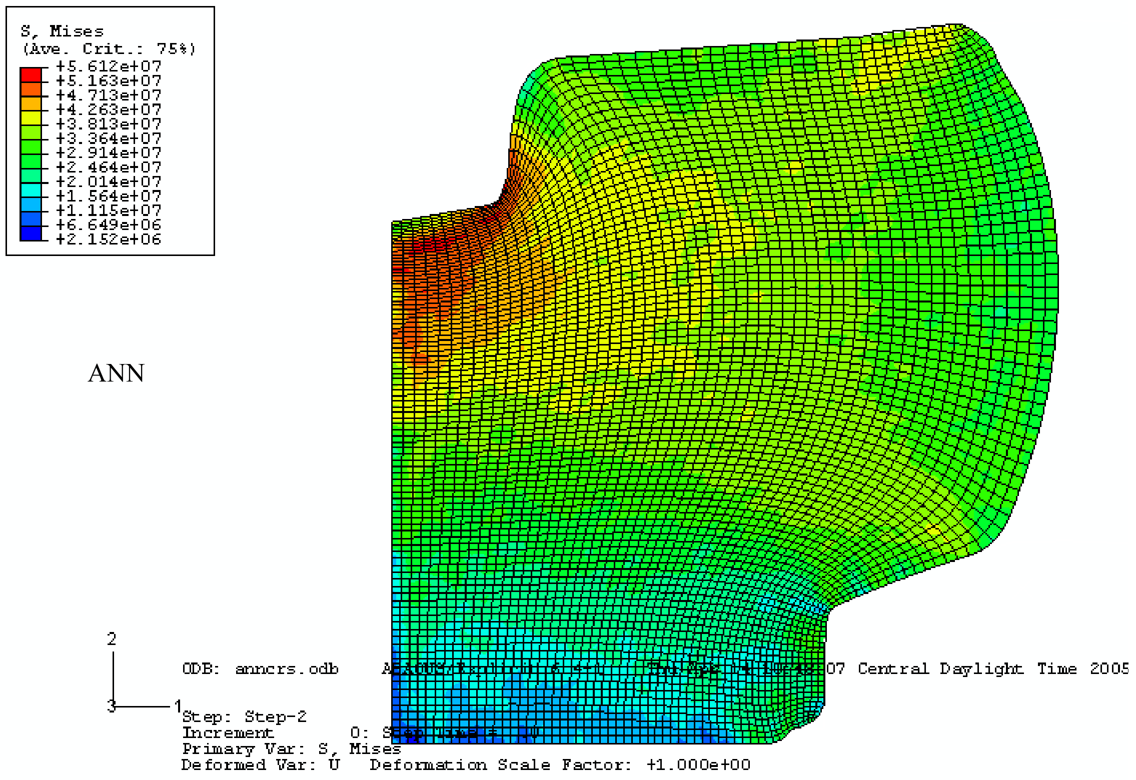


FIGURE 199. Mises stress plots of the conventional and ANN-based meshes at the start of the second step.

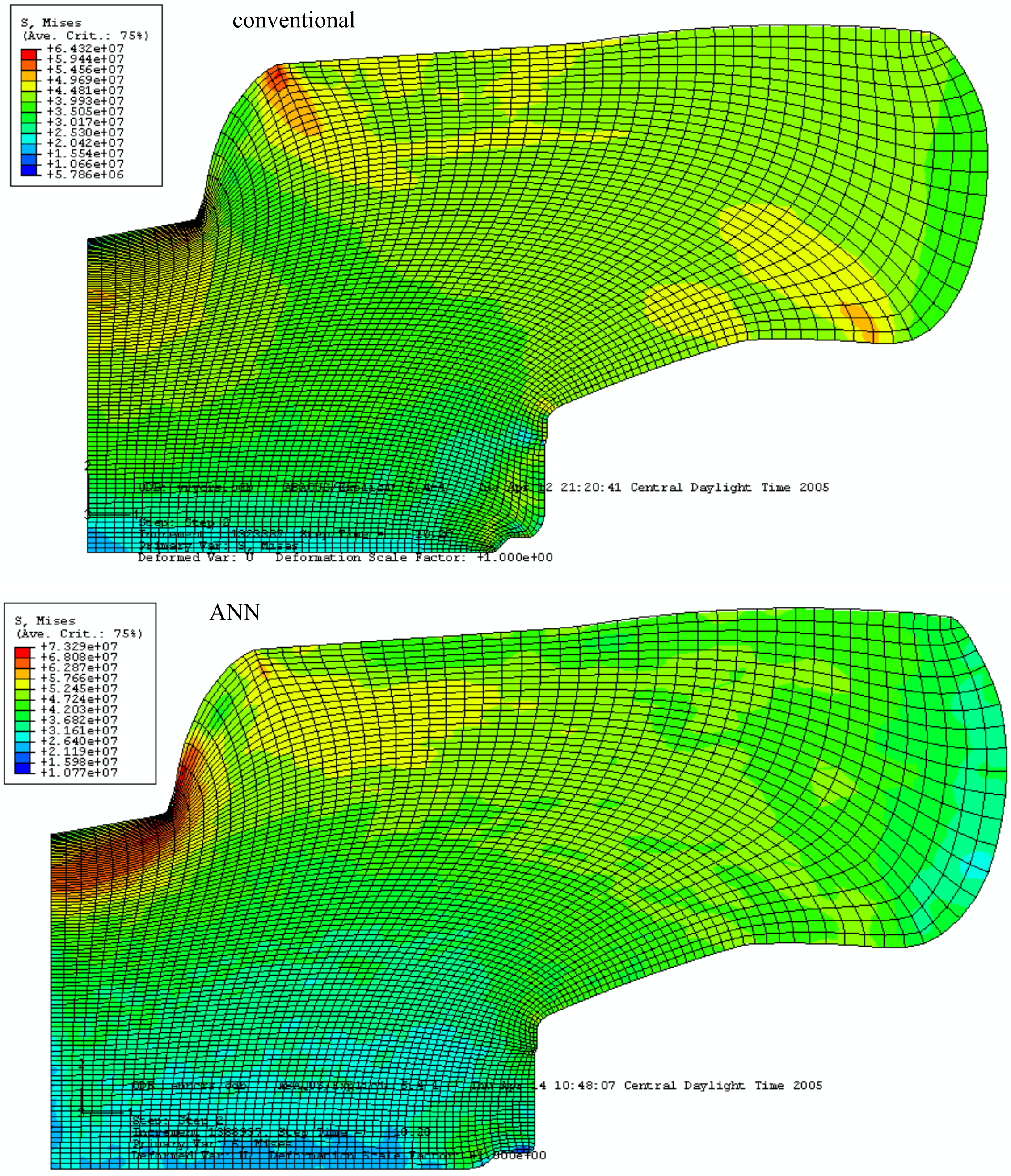


FIGURE 200. Mises stress plots of the conventional and ANN-based meshes at 10.2 seconds into the second step.

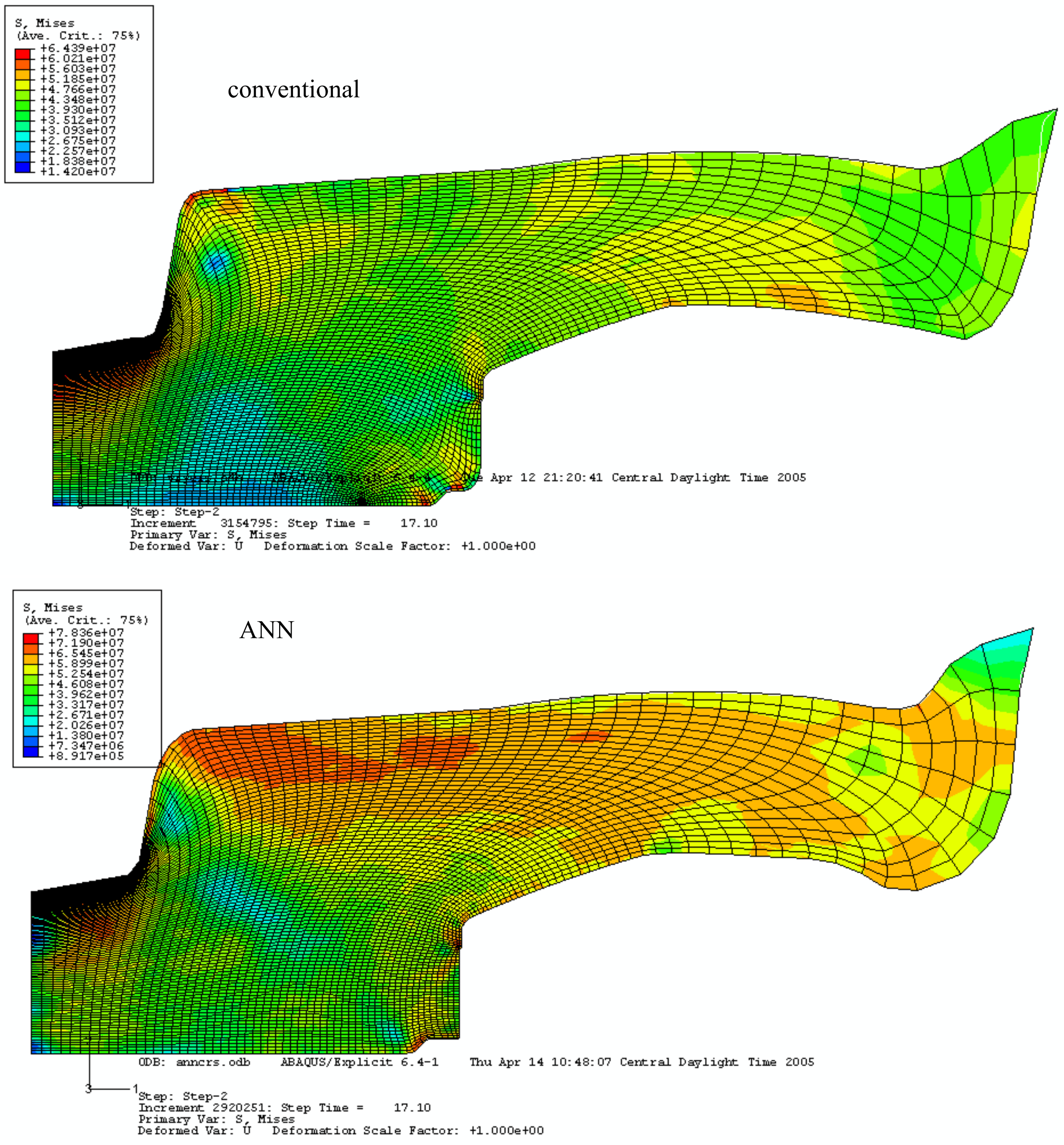


FIGURE 201. Mises stress plots of the conventional and ANN-based meshes at 17.2 seconds into the second step, not very large elements produced.

Neither the square nor rectangular meshed models run appropriately for longer than approximately 35 seconds total time. As can been seen in the above figures, the dies are still far from complete closure at this point. It is possible that a finer mesh would remain stable further into the simulation, but significant CPU times would result.

The 1.5 mm-square based model took approximately 12 days of CPU time requiring a total of approximately 3 weeks of real time. The NCSA supercomputing batch system allots a maximum of 96 hours (4 days) of CPU time per job submission. This requires multiple restarts or re-submissions if long run times are necessary. In addition, the more computing resources (e.g., number of processors, amount of memory, time, etc.) requested the longer a job may sit in batch queue before starting. Submission to start times generally varied from a few hours to many days.

The computational costs associated with explicit integration schemes increase proportionally with the number of elements involved and decrease roughly proportional to the smallest element dimension [79]. If the mesh size was reduced by a factor of two, the increase in time would roughly increase by a factor of eight (2 x 2 x 2). So, if the original mesh was refined down to 0.75 mm squares the analysis may take as long as 6 months.

Inspection of the element sizes near model failure, for either the square or rectangular meshes, suggest even a factor of 10 decrease in original element size would result in elements larger than acceptable at completion of the analysis. An alternate approach is developed below.

9.4 Second Step, Alternate Model Approach

The excessive amounts of plastic deformation taking place during blocking result in severe changes in element dimensions. The blocking operation principally produces compression of the elements in the y-direction. The second step results in more complicated behavior and to this point has started with already severely distorted elements. While the adaptive meshing capabilities allow completion of blocking, they do not readily produce an acceptable mesh through progression of the second step.

One possible solution, and the one carried out below, is to extract the deformed shape of the billet from the model at completion of blocking and then establish a new mesh. The software does not provide this ability directly within its code, but this task can be performed through scripting, or programming, using Python. It should be noted that re-meshing the deformed shape does not transfer loads or the previous state of stress. This may be possible, but an extensive amount of Python coding would be required and is beyond the scope of this work. Fortunately, this particular forging operation is carried out near the solution temperature of aluminum and as such, it may be reasonable to assume very little residual stress exists. The experimental results previously detailed support this hypothesis in that no noticeable hardness or microstructural differences were noted when high stress areas were compared to low stress areas. In addition, the operation itself consists of blocking followed by ejection of the billet during lubricant application, so dies forces have been removed prior to starting the second step.

To begin re-meshing, the deformed shapes produced by the conventional and ANN-based models at the end of blocking were extracted from the originally square shaped meshes. This is achieved by importing an orphaned mesh shape from the model. A two dimensional profile is then extracted from the orphan to produce a sketch or part that can be meshed. A sample Python script that achieves these ends is supplied in the Appendix.

The re-meshed part generated is shown in Figure 202. The new mesh is comprised of various sized quadrilateral elements, with much smaller elements in areas anticipated as requiring greater refinement (smallest dimension 0.1 mm). This new mesh results in 30,596 degrees of freedom for the model, effectively doubling the number of nodes from the previous mesh. This increases the computational times significantly, but starting at the end of blocking eliminates that step from the analysis.

The quad re-meshed models continued to run without failure until full die closure was achieved (see Figure 203 through Figure 207). The conventional model needed approximately 12 days of CPU time to reach this point. The ANN model ran for closer to 18 days. The again points out the severe difficulties the code has in reach a solution.

The final mesh exhibits significant element distortion, particularly enlarged elements near the top and bottom areas in the outer portions of the dies. The large elements sizes prevent true assessment of die filling;

one of the ultimate aims of running forging simulations. This confirms a need for a more refined mesh, but time constraints prevent attempting a simulation with small enough elements to achieve these ends.

Additionally, a triangular element-based model was also generated, as shown in Figure 208. This model has 33,696 degrees of freedom. These models did not run to completion after longer than 15 days of CPU time and were terminated at that point (see Figure 209).

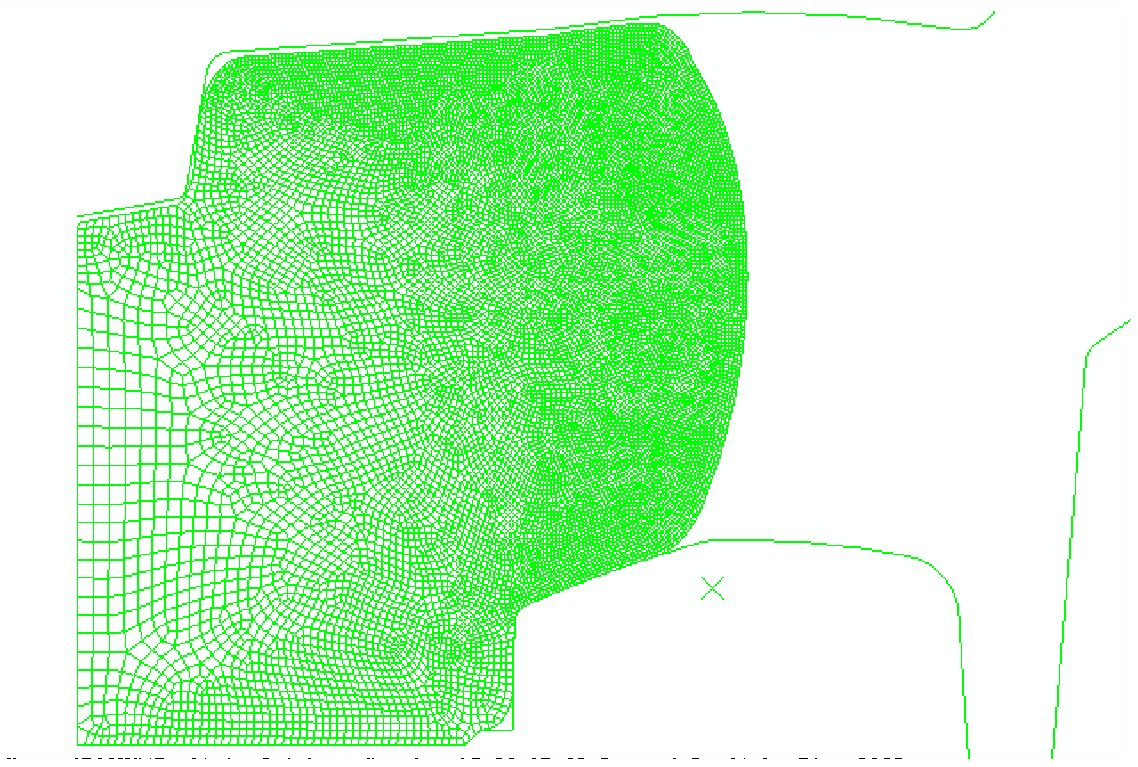


FIGURE 202. Quadrilateral re-meshed deformed billet at the end of blocking.

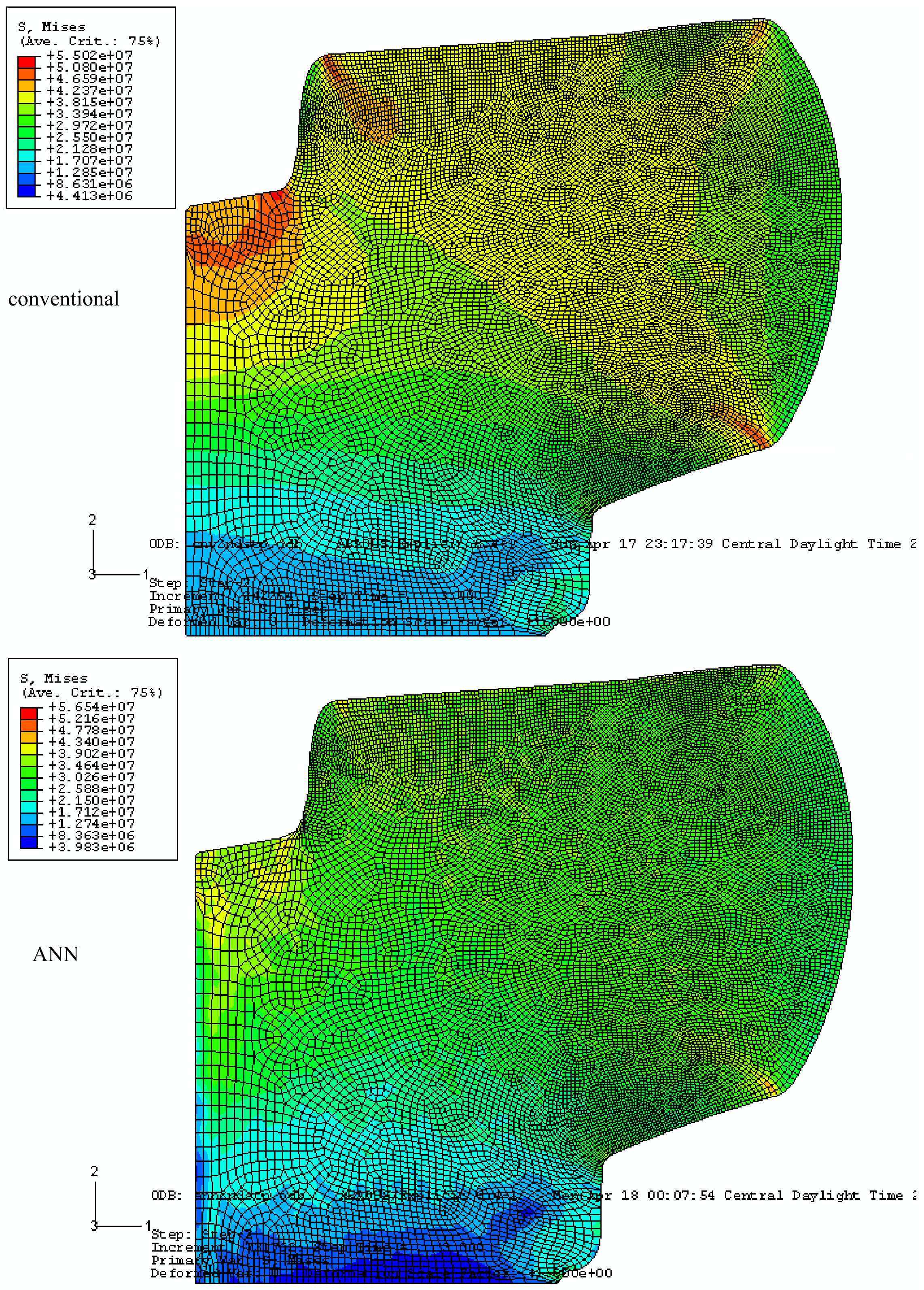


FIGURE 203. Mises stress plots of the conventional and ANN-based meshes at 3.0 seconds into the second step.

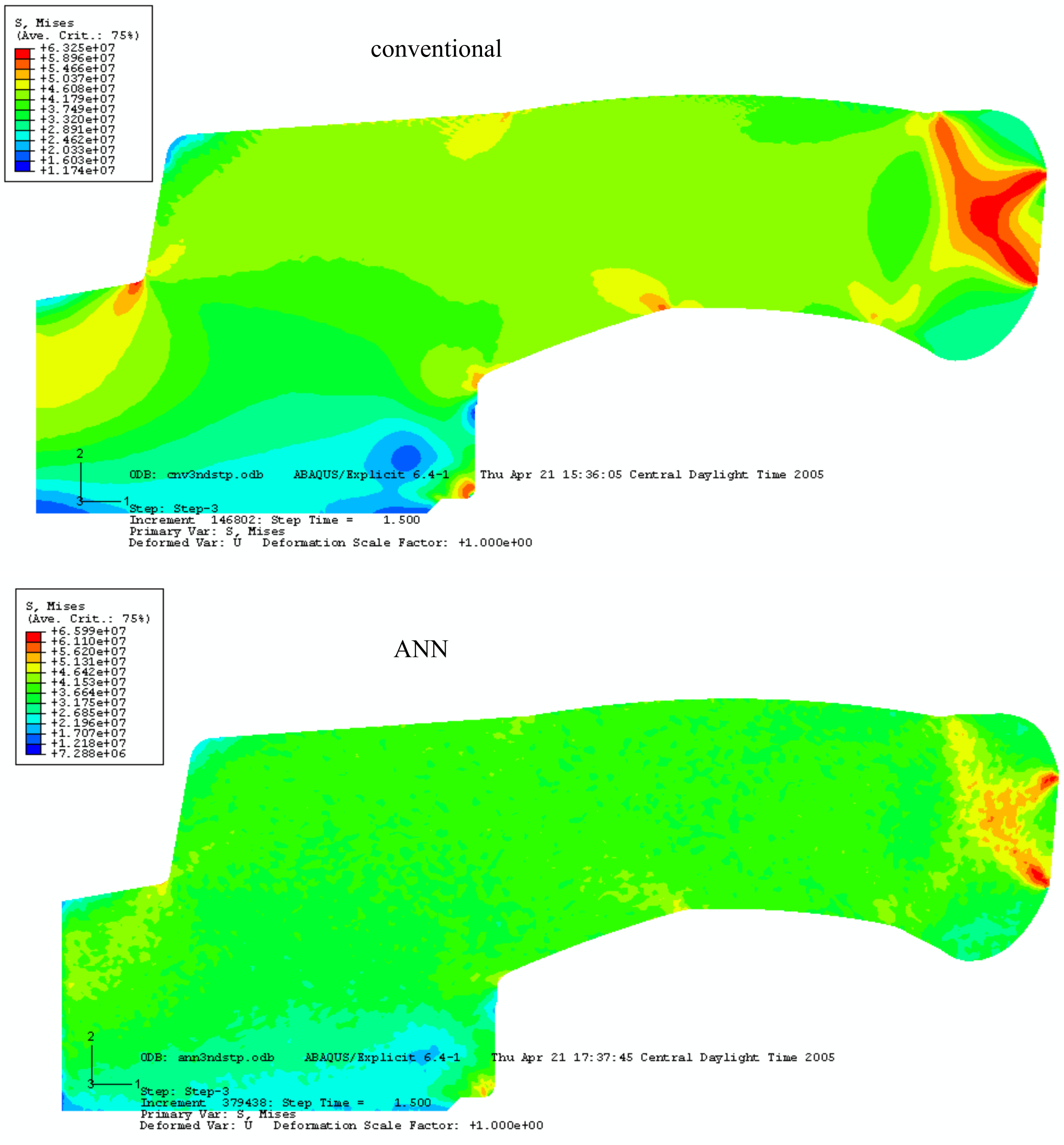
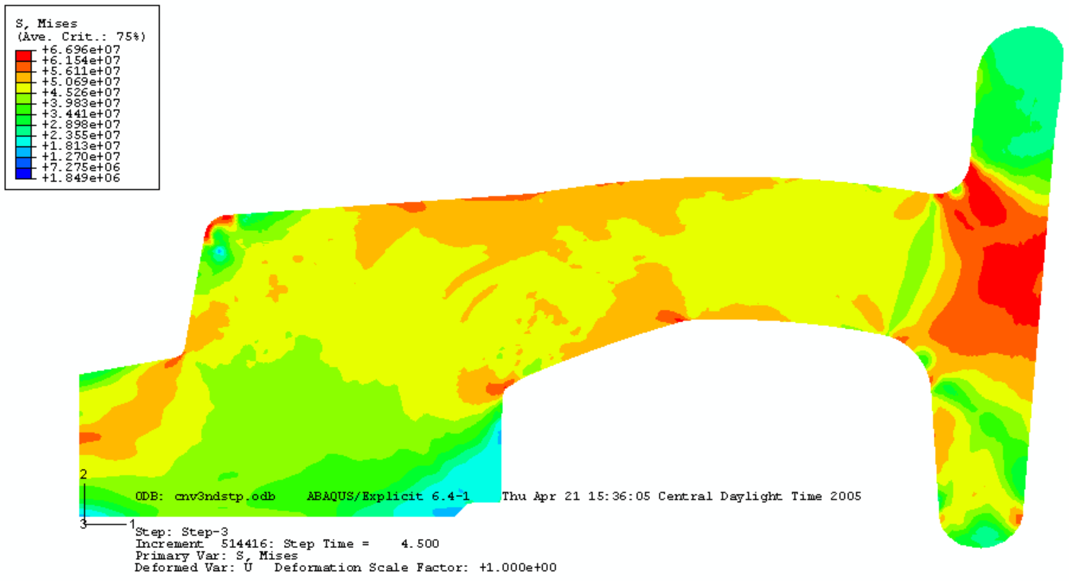


FIGURE 204. Mises stress plots of the conventional and ANN-based meshes at 15 seconds into the second step.



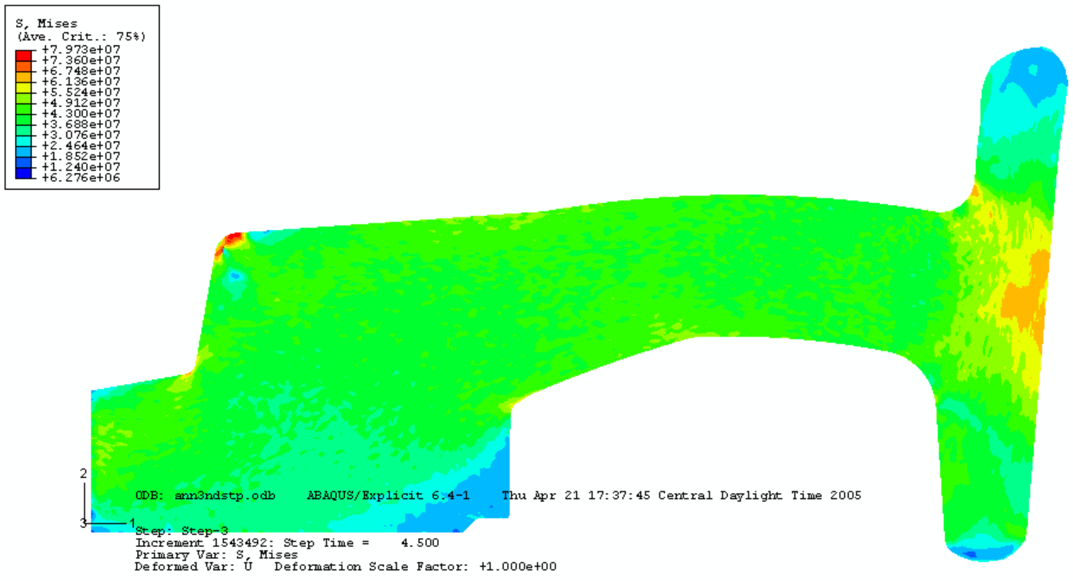
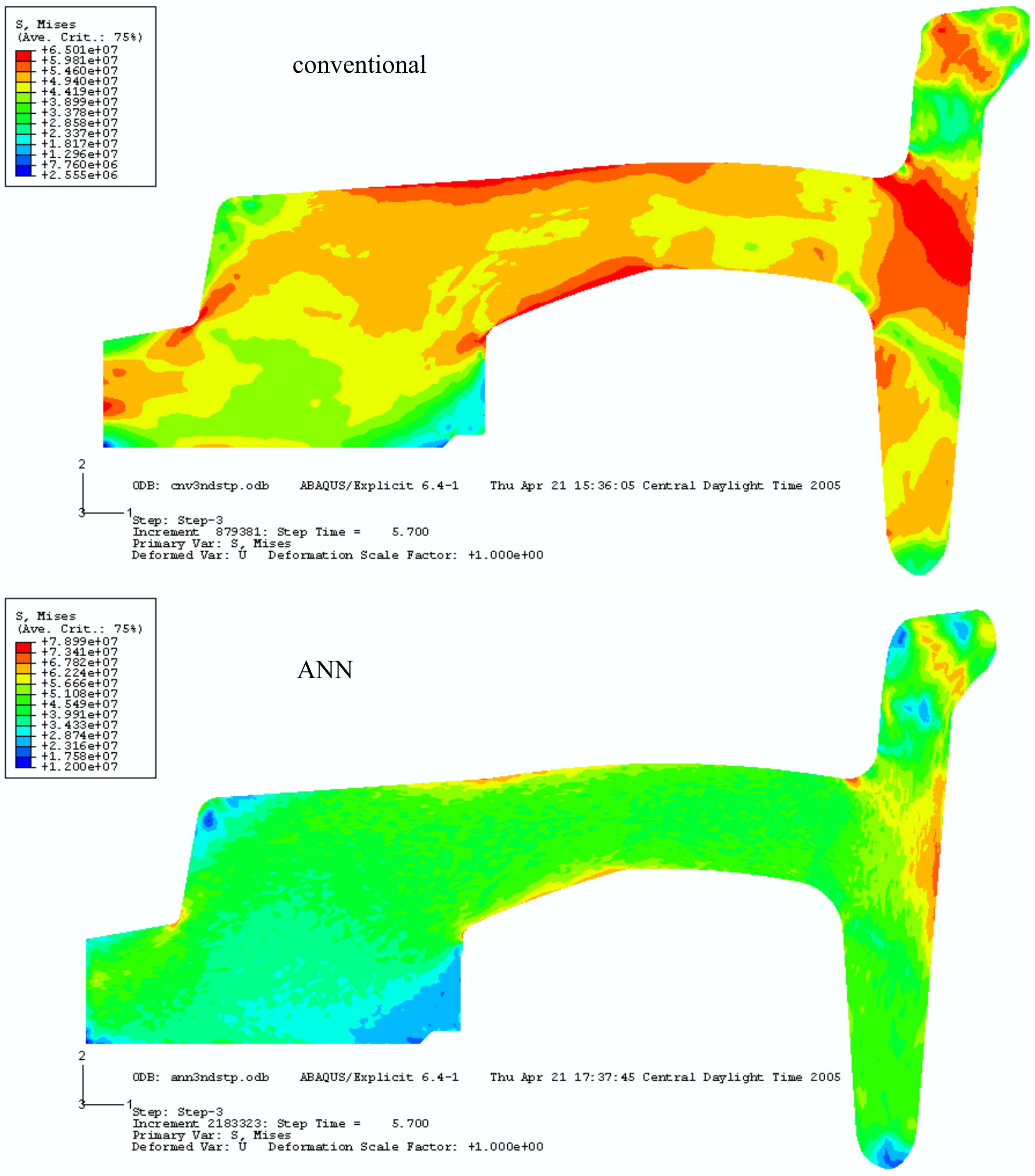


FIGURE 205. Mises stress plots of the conventional and ANN-based meshes at 18 seconds into the second step.



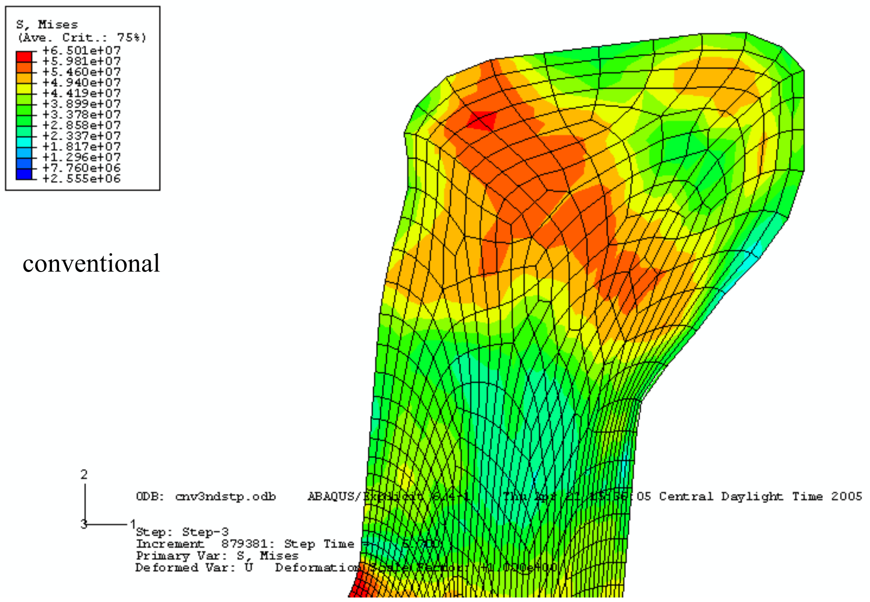
```
ODB: cnv3ndstp.odb    ABAQUS/Explicit 6.4-1    Thu Apr 21 15:36:05 Central Daylight Time 2005

Step: Step-3
Increment  879381: Step Time =    5.700
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

```
ODB: ann3ndstp.odb    ABAQUS/Explicit 6.4-1    Thu Apr 21 17:37:45 Central Daylight Time 2005

Step: Step-3
Increment 2183323: Step Time =    5.700
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

FIGURE 206. Mises stress plots of the conventional and ANN-based meshes at completion of the second step; full die closure.



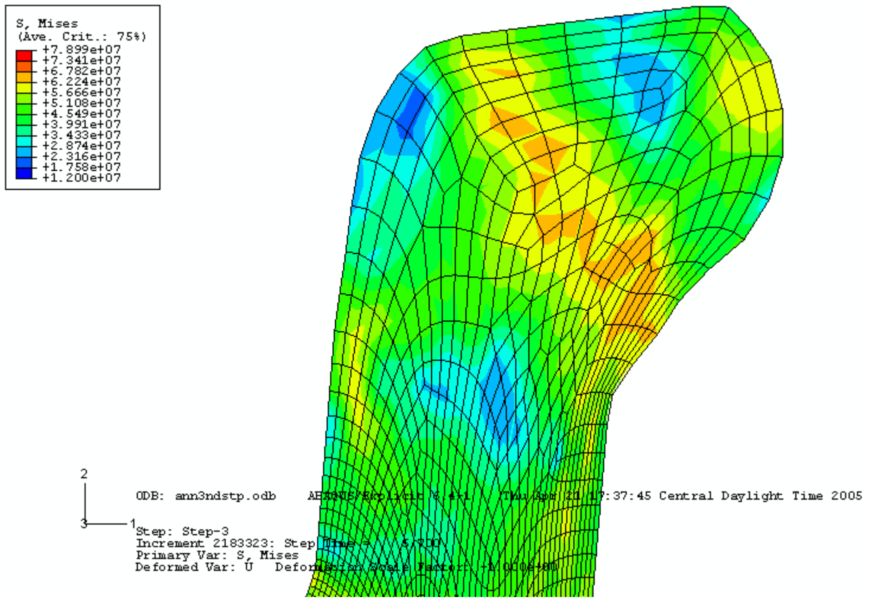


FIGURE 207. Close-up view of mesh distortions for both models at complete die closure.

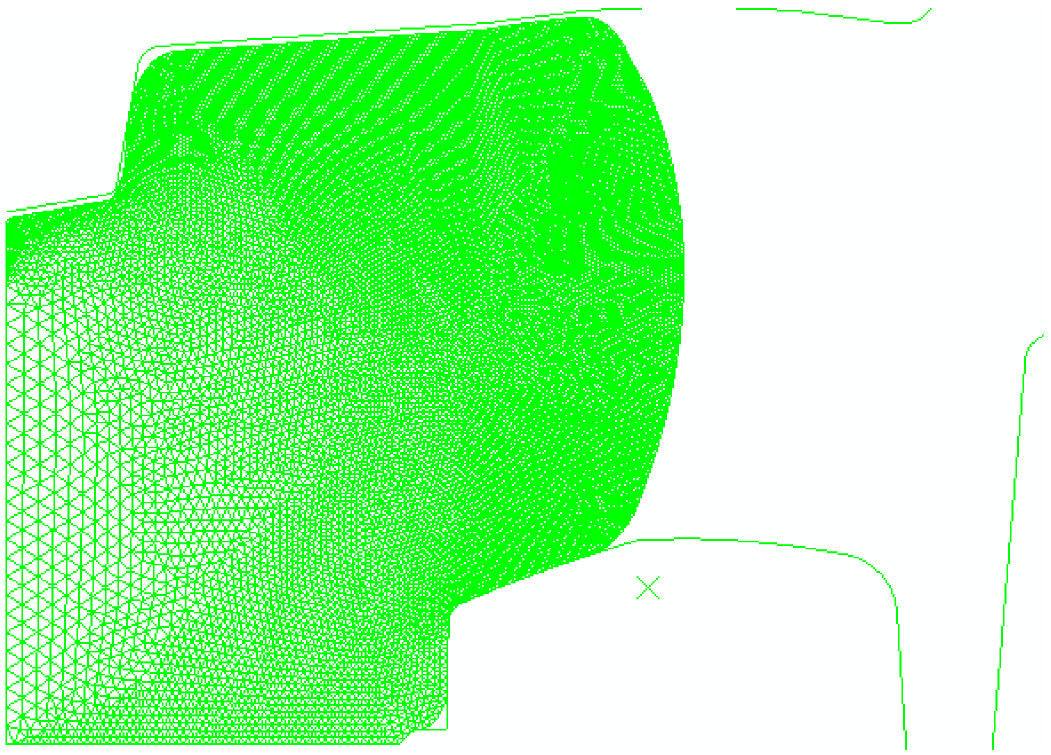
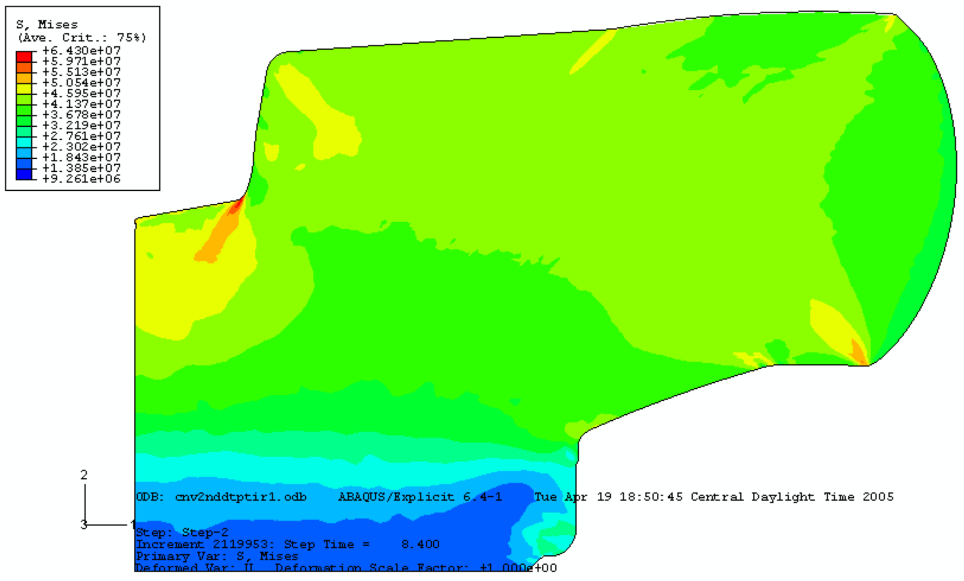


FIGURE 208. Triangular re-meshed deformed billet at the end of blocking.



```
ODB: cnv2nddtptir1.odb    ABAQUS/Explicit 6.4-1    Tue Apr 19 18:50:45 Central Daylight Time 2005

Step: Step-2
Increment 2119953: Step Time =    8.400
Primary Var: S, Mises
Deformed Var: U   Deformation Scale Factor: +1.000e+00
```

FIGURE 209. Mises contour plot of triangular meshed conventional model after approximately 15 days of run time; only 8.4 seconds into second step.

9.5 Second Step, Finite Element Model vs. Data Acquired

Figure 210 provides a comparison of the force required to compress the dies for the second step of the forging operation. The ANN-based model clearly predicts a reaction force more consistent with that measured experimentally, but neither of the models really performs all that well as the dies near closure. The is likely due to lack of adequate mesh refinement near the end of simulation along with possible discrepancies between model die velocity and experimental.

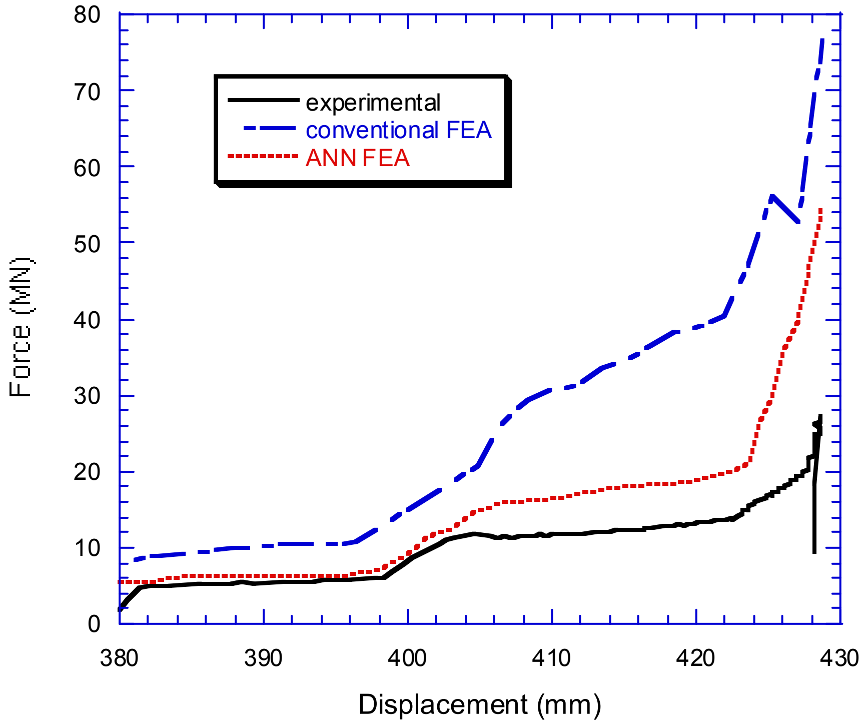


FIGURE 210. Comparison of FEA models against experimental data acquired.

10. Conclusions and Recommendations

10.1 Conclusions

The present study was directed towards development of an integrated approach combining neural network material based modeling with FEM simulation of forming processes. Special consideration was given to the hot forging of aluminum alloy 6061 and a nickel aluminide alloy.

The following specific conclusions could be drawn:

1. The present study demonstrates the applicability of artificial neural network (ANN) material models as implemented within, or linked to, finite element code.
2. The conventional method of curve-fitting experimental material data, while not particularly difficult for the aluminum, proved exceptionally tedious and not particularly accurate for a more complex rheology such as presented by the nickel aluminide.
3. Once the appropriate network architecture is achieved, an ANN has the capability to almost perfectly match the experimental data available for training and to adequately predict behavior over a wider range of strain, strain rates, and temperatures.
4. The flow stress curves generated using the ANN method for 6061 aluminum show the typical behavior of high stacking fault energy materials, where the controlling softening mechanism is dynamic recovery (early strain hardening followed by a smooth transition to a plateau of stress).
5. In contrast to 6061 aluminum, the flow stress behavior of nickel aluminide exhibits the typical behavior of low stacking fault energy materials, where the controlling softening mechanism in hot working is the dynamic recrystallization (early strain hardening to a peak stress followed by drop and oscillation of the flow stress about a steady average value).
6. A thermo-mechanical coupled finite element method (FEM) using the commercial code ABAQUS as a platform for development is introduced to simulate hot forming processes.
7. The FEM model is integrated with the developed ANN material based model in order to account for the effects of strain, strain rate, and temperature variations within the material during hot-forming.
8. An industrial case study involving hot forging of an aftermarket automotive wheel made out of 6061 aluminum is used to evaluate the effectiveness of the integrated approach.
9. The load-displacement curves predicted by the developed virtual model are in good agreement with the experimental observations of an industrial forging process.
10. The developed approach and knowledge gained from the present work, has wide range of application in general, not only for hot forming of the investigated materials, but also for different alloy systems.

10.2 Recommendation for Future Development and Applications

The present investigation suggests several avenues for further research.

Simulations of Material Behavior

While the above ANN-FEA linked model shows promise for production of more accurate simulations, the forged wheel simulation severely tests the ability of ABAQUS itself to provide a solution. Many other material modeling simulations, particularly those that take place over much shorter time spans should benefit from implementation of an ANN-based approach. Small time increments are not an issue for problems, such as crash analysis, to occur over very short real-time spans. Hammer forging, composite fracture, brittle metal behavior, and other material modeling applications that exhibit rate and temperature dependencies would likely benefit from this approach.

Flexible Testing for Model Development

The artificial neural networks developed above were based on constant strain rate testing, carried out in laboratory settings. The ANN does not require constant strain rates, just that the rate is measured and known for input along with the other parameters. This ability should allow testing and model development to be performed on conventional constant velocity testing equipment more typically found or even on forging presses 196 already in-place in a manufacturing setting. This could save considerable time and investment with specialized testing services. Model data could even be acquired during actual manufacture for further refinement.

Enhancement of Finite Element Code

More recently, due to the extreme difficulties encountered in solving forging problems with excessive amounts of plastic deformation, two basic approaches within finite element analysis are being explored. Researchers have begun to develop meshing methods that attempt to construct original meshes that survive extreme deformation [78]. The basic idea is to perform a pre-analysis that establishes geometric and stress information in an attempt to determine the optimal nodal placements at the beginning of analysis. The start-in mesh is set up to achieve reasonable mesh shapes at the completion of analysis. A second tack is to employ meshless or finite volume methods preventing the shape problems altogether. Recently, commercial software solutions such as MSC.SuperForge have been released that incorporate these methods[81]. Importation of an ANN-based material model into one of these packages may prove able to more efficiently solve forging problems.

Inclusion of Other Material Properties

Researchers have successfully trained neural networks to predict material parameters such as strength, hardness, and microstructure [82][83][84]. The linking of an ANN with finite element code achieved above could also be utilized for prediction of these other material properties not currently available with conventional codes.

10.3 Computational Costs

As was noted in the foregoing, significant computational times result when running forging models with the complexity encountered with the aftermarket wheel. The initial simulations employing 10,000 to 15,000 degrees of freedom required anywhere from 6 to 10 days of computational time on NCSA's system. The ANN-based model requires approximately 30% longer to run due to the subroutine calls. The second step model using 30,000 or greater degrees of freedom took around 12 days for the conventional model to run to completion. The ANN-based model's CPU time was approximately 50% greater. There appears to be more frequent subroutine calls increasing computational effort.

It should be noted, that with both conventional and the ANN-based approaches, the stable time increment varies from approximately 10^{-5} to 10^{-6} seconds. The wheel forging process takes around 40 seconds in real time and due to the strain rate dependence the simulation must be run over this total time. If processes requiring significantly less "real" time were to be modeled significantly less CPU times would be required as the stable time increment would not change significantly. In addition, models of lower complexity would likely require less frequent adaptive meshing, enabling shorter solution times.

REFERENCES

1. Prassad, Y.V.R.K., Sasidhara, S., Gegel, H.L., and Malas., J.C., 1997, *Hot Working Guide: A Compendium of Processing Maps*, ASM International, Metals Park, p. 104-105.
2. Prasad, Y.V.R.K., Sasidhara, S., and Sikka, V.K., 2000, "Characterization of mechanisms of hot deformation of as-cast nickel aluminide alloy," *Intermetallics*, **8**, p 987-995.
3. Smith, Joyce., 2005, "The Weld wheels of fortune," *The Kansas City Star*, Knight Ridder, **189**, p C-2.
4. Sellars, C.M. and Tegart, W.J.McG., 1963, *Mem. Sci. Rev. Metall.,* **63**, p 86.
5. Laasroui, A., and Jonas, J.J., 1966, "Prediction of Steel Flow Stresses at High Temperatures and Strain Rates," *Metall. Trans. A,* **22**, p 1545-1558.
6. Brown, S.B., Kim, K.H., and Anand, L., 1989, "An Internal Variable Constitutive Model for Hot Working of Metals," *Int. J. Plast.,* **5**, p 95-130.
7. "Standard Test Methods of Compression Testing Metallic Materials at Room Temperature," ASTM E9-89a, *Annual Book of ASTM Standards.*
8. "Standard Practice for Compression Tests of Metallic Materials at Elevated Temperatures with Conventional or Rapid Heating Rates and Strain Rates," ASTM E209-00, *Annual Book of ASTM Standards.*
9. *ASM Handbook Volume 08: Mechanical Testing and Evaluation,* 2000, ASM International, Metals Park, p 798-809.
10. Altan, Taylan, Oh, Soo-OK, and Gegel, Harold L., 1983, *Metal Forming, Fundamentals and Applications*, American Society for Metals, Metals Park, p. 45.
11. http://cst-www.nrl.navy.mil/lattice/struk.picts/a1.s.png.
12. Lange, Kurt, 1985, *Handbook of Metal Forming,"* McGraw-Hill, New York, p. 3.5.
13. Flinn, Richard A., and Trojan, Paul K., 1990, *Engineering Materials and Their Applications*, Houghton Mifflin, p. 93.
14. Lange, Kurt, 1985, *Handbook of Metal Forming,"* McGraw-Hill, New York, p. 3.6.
15. Byrne, J.G., 1965, *Recovery, Recrystallization and Grain Growth*, Macmillan, New York.
16. Flinn, Richard A., and Trojan, Paul K., 1990, *Engineering Materials and Their Applications*, Houghton Mifflin, p. 94.
17. Allen, Dell K., 1981, *Metallurgy Theory and Practice*, American Technological Publishers, p. 93.
18. Lange, Kurt, 1985, *Handbook of Metal Forming,"* McGraw-Hill, New York, p. 3.21.
19. Rao, K.P., and Prasad, Y., 1990, "Neural Network Approach to Flow Stress Evaluation in Hot Deformation," *Journal of Materials Processing Technology,* **53**, p 552-566.

20.Hodgson, P.D., Kong, L.X., and Davies, C., 1999, “The Prediction of the Hot Strength in Steels with an Integrated Phenomenological and Artificial Neural Network Model,” *Journal of Materials Processing Technology,* **87**, p 131-138.

21.Enemuoh, E.U., and El-Gizawy, A. Sherif, 2002, “A Robust Neural Network Model for Online Prediction of Process-Induced Damage in Drilling Epoxy Composites,” Proceeding 3rd CIRP Int. Sem. On Intelligent Computation in Mfg. Eng., Naples, Italy, p 183-188.

22.Bariani, P.F., Bruschi, S., and Negro, T.Dal, 2002, “Neural Networks to Describe Superalloys Rheological Behavior Under Hot Working Conditions,” Proceeding 3rd CIRP Int. Sem. On Intelligent Computation in Mfg. Eng., Naples, Italy, p 585-588.

23.Teti, R. and D’Addona, D., 2002, “Selection of Optimum Neural Network Model for Hot Forging Material Behaviour Prediction,” Proceeding 3rd CIRP Int. Sem. On Intelligent Computation in Mfg. Eng., Naples, Italy, p 577-584.

24.*The Scientific American Book of the Brain*, 1999, Scientific American, **3**, New York.

25.Fausett, L., 1994, *Fundamentals of Neural Networks, Architectures, Algorithms, and Applications*, Prentice-Hall, Englwood Cliffs, NJ, p xiii.

26.McCulloch, W. and Pitts, W., 1943, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, **5**, p 115-133.

27.Rosenblatt, F., 1958, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, **65**, p 386-408.

28.Minsky, Marvin Lee, and Pappert, Seymour, 1969, *Perceptrons; An Introduction to Computational Geometry*, MIT Press, Boston.

29.Haykin, Simon, 1999, *Neural Networks: A Comprehensive Foundation 2nd Ed.,* Prentice Hall, New Jersey, p 203.

30.Demuth, H., Beale, M.,2002, *MATLAB Neural Network Toolbox User’s Guide*, The Mathworks, Inc., Natick, p 2-17.

31.Hagan, Martin T., Demuth, Howard B., and Beale, Mark, 1996, *Neural Network Design*, PWS Publishing Company, p. 11-23.

32.Scales, L.E., 1985, *Introduction to Non-Linear Optimization*, Springer-Verlag, New York.

33.Jeffreys, H., 1939, *Theory of Probability*, Oxford University Press, Oxford.

34.Gull, S.F., 1988, "Bayesian inductive inference and maximum entropy," *Maximum Entropy and Bayesian Methods in Science and Engineering, Vol. 1: Foundations*, G.J.Erickson and C.R. Smith, eds., Kluwer, Dordrecht, p. 53-74.

35.MacKay, David J.C., 1992, "Bayesian Interpolation," *Neural Computation*, Massachusetts Institute of Technology, **4**, p. 415-447.

36.Tancret, F., Bhadeshia, H.K.D.H., and MacKay, D.J.C., 1999, "Comparison of artificial neural networks with Gaussian processes to model the yield strength of nickel-base superalloys," **n 39 v10**, ISIJ International, p. 1020-1026.

37.Fujii, Hidetoshi, Mackay, D.J.C., Bhadeshia, H.K.D.H., 1996, "Bayesian neural network analysis of fatigue crack growth rate in nickel base superalloys," ISIJ International, **v36 n11**, p. 1373-1382.

38.Vermeulen, W.G., Morris, P.F., de Weijer, A.P., van der Zwaag, S., 1996, "Prediction of martensite start temperature using artificial neural networks," *Iron & Steelmaking*, **v23 n5**, p. 433-437.

39.Vermeulen, W.G., Morris, P.F., de Weijer, A.P., van der Zwaag, S., 1996, "Prediction of Jominy hardness profiles of steels using artificial neural networks," *Journal of Materials & Performance*, **v5 n1***,* p.57-63.

40.Singh, S.B., Bhadeshia, H.K.D.H., Mackay, D.J.C., Carey, H., and Martin, I., 1998, "Neural network analysis of steel plate processing," *Ironmaking & Steelmaking*, **v25 n5**, p. 355-365.

41.Korczak, P., Dyja, H., 2001, "Investigation of microstructure prediction during experimental thermo-mechanical plate rolling," *Journal of Materials Processing Technology,*" **v109 n1-2**, p. 112-119.

42.Courant, R., 1943, "Variational Methods for the Solution of Problems of Equilibrium and Vibrations," *Bulletin of the American Mathematical Society*, **49**, p. 1-23.

43.Synge, J.L., 1957, *The Hypercircle in Mathematical Physics*, Cambridge University Press, London

44.Argyris, J.H. and Kelsey, S., 1954, "Energy Theorems and Structural Analysis," *Aircraft Engineering*, **26 & 27**.

45.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 5.2.3.

46.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 6.3.3.

47.*Getting Started with ABAQUS*, 2003, ABAQUS, Inc., Pawtucket, 2.4.1.

48.*Getting Started with ABAQUS*, , 2003, ABAQUS, Inc., Pawtucket, 2.4.

49.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 6.1.1.

50.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 7.16.1.

51.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 7.16.3.

52.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 6.3.3.

53.*Contact in Abaqus,* 2003, Abaqus Training Seminar, Abaqus Inc., L2.10.

54.*Contact in Abaqus,* 2003, Abaqus Training Seminar, Abaqus Inc., L6.14.

55.*Contact in Abaqus,* 2003, Abaqus Training Seminar, Abaqus Inc., L7.8.

56.Smith, William F., 1993, *Structure and Properties of Engineering Alloys*, McGraw-Hill, 2nd ed., p. 203-204.

57.http://asm/matweb.com/search/SpecificMaterial.asp?bassnum=MA6061T6 (23 December 2003).

58.*Metals Handbook Volume 2: Properties and Selection: Nonferrous Alloys and Pure Metals,* 1979, American Society for Metals, Metals Park, p.115-116.

59.Liu, C.T. and Inouye, H., 1973, *Metall. Trans.,* **4**, p 1743.

60.*ASM Handbook Volume 02: Properties and Selection: Nonferrous Alloys and Special Purpose Metals*, 1990, ASM International, Metals Park, p 913-918

61.Thornton, P.H., Davies, R.G., and Johnston, T.L.,1970, *Metall. Trans., **1**, p 207.*

62.Stoloff, N.S., Fuchs, G.E., Kuruvilla, A.K., and Choe, S.J., 1959, *Mechanical Properties of Intermetallic Compounds,* J.H. Westerbrook, Ed., Willey, p 247-260.

63.Payne, J.E. and Desai, P.D., 1994, *Properties of Intermetallic Alloys I. Aluminides,* Metals Information Analysis Center, Purdue University Publishers, p 11.1

64.Stoloff, N.S., and Sikka, V.K., eds., 1996, *Physical Metallurgy and Processing of Intermetallic Compounds*, Chapman & Hall, New York, NY.

65.Prasad, Y.V.R.K., Sasidhara, S., and Sikka, V.K., 2000, “Characterization of mechanisms of hot deformation of as-cast nickel aluminide alloy,” *Intermetallics*, **8**, p 988.

66.Morsi, K.and Rodriguez, J., 2004, "Combustion Forging of FeAl (40at.%AL)," *Journal of Materials Science*, **39**, p. 4849-4854.

67.Rodrigues, J., Moussa, S.O., and Morsi, K., 2003, "Low-Energy Forging of Aluminide Intermetallics," *Scripta Materialia*, **48**, p. 707-712.

68.Liu, C.T. and White, C.L., 1985, “High Temperature Ordered Intermetallic Alloys,” Materials Research Society Symposia Proceedings, Koch, C.C., Liu, C.T., and Stoloff, N.S., Ed., Materials Research Society, p 365-380.

69.Zhang, L.M., Liu, J., Yuan, R.Z., and Hirai, T., 1995, “Properties of TiC-Ni_3Al composites and structural optimization of TiC-Ni_3Al functionally gradient materials,” Materials Science and Engineering, **203**, p 273.

70.Smith, Joyce., 2005, “The Weld wheels of fortune,” *The Kansas City Star*, Knight Ridder, **189**, p C-2.

71.“Standard Hardness Conversion Tables for Metals Relationship Among Brinell Hardness, Vickers Hardness, Rockwell Hardness, Superficial Hardness, Knoop Hardness, and Scleroscope Hardness,” ASTM E140-02e1, *Annual Book of ASTM Standards.*

72.Schey, John A., 2000, *Introduction to Manufacturing Processes*, 3rd ed., McGraw-Hill, p. 282.

73.Schey, John A., 2000, *Introduction to Manufacturing Processes*, 3rd ed., McGraw-Hill, p. 332.

74.Schey, John A., 2000, *Introduction to Manufacturing Processes*, 3rd ed., McGraw-Hill, p. 286.

75.Schey, John A., 2000, *Introduction to Manufacturing Processes*, 3rd ed., McGraw-Hill, p. 332.

76.http://access.ncsa.uiuc.edu/Releases/03Releases/03.31.03_NCSA's_IBM.html (10 March 2005).

77.*Metals Handbook Vol. 5 Forging and Casting*, 8th ed., 1970, American Society for Metals, Metals Park, p.128.

78.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 6.1.1.

79.*ABAQUS Analysis User's Manual*, 2003, ABAQUS, Inc., Pawtucket, 2.4.

80.Dheeravongkit, Arbtip and Shimada, Kenji, 2004, "Inverse Pre-Deformation of Finite Element Mesh for Large Deformation Analysis," Proceedings, 13th International Meshing Roundtable, Williamsburg, VA, Sandia National Laboratories, SAND #2004-3765C, p.81-94.

81.http://www.mscsoftware.com/assets/2221_SF303ZZZLTDAT.pdf.

82. Crotaz, C R. Shercliff, H R., and Mackay, D J C., 2002, "Advanced statistical modelling of processing of aluminium alloys," *Materials Science Forum*, **396-40,** n 2, p. 643-648.

83.Furukawa, Tomonari. Hoffman, Mark, 2004, "Accurate cyclic plastic analysis using a neural network material model," *Engineering Analysis with Boundary Elements*, **28,** n 3, p.195-204.

84.Giorleo, G. Teti, R. Prisco and U. D'Addona, D, 2003, "Merging neural network material rheological behaviour modelling with FEM simulation of orthogonal metal cutting," *Machining Science & Technology*, **7**, n 3, p. 401-417.

APPENDIX

Sample MATLAB m-files

m-file for input of raw data obtained from digitized curves

```
% Reads in data from a file
% then mixes curves, sets up ANN, trains, then plots
% also outputs weights and biases

load values300C.txt -ascii
T=300
plus=0
mega=1E6

% loop for all p's, p(1,x) are the strain values, p(2,x) are strain rates
% t's are flow stress values
% every values is read into pall

for i=1:1:130
   pall(1,i)=transpose(values300C(i,1));
   pall(1,i+130)=transpose(values300C(i,1));
   pall(1,i+260)=transpose(values300C(i,1));
   pall(1,i+390)=transpose(values300C(i,1));

   pall(2,i)=0.001;
   pall(2,i+130)=0.01;
   pall(2,i+260)=0.1;
   pall(2,i+390)=1;

   pall(3,i)=T;
   pall(3,i+130)=T;
   pall(3,i+260)=T;
   pall(3,i+390)=T;

   tall(i)=transpose(values300C(i,2))*mega;
   tall(i+130)=transpose(values300C(i,3))*mega;
   tall(i+260)=transpose(values300C(i,4))*mega;
   tall(i+390)=transpose(values300C(i,5))*mega;
end

% calculates natural log of p(1:2) and p(3) inverse of temperature or
unmodified
```

```
% read p(1,k) for every 3rd value as input, val.p for validation, and
test.p, etc
% reads 130 values per strain rate curve, total of 520 data points (sin-
gle temp)

% unmodified
%p(1:3,1+plus)=pall(1:3,1);
%t(1+plus)=tall(1);

% natural log and inverse
p(1:2,1+plus)=log(pall(1:2,1));
p(3,1+plus)=1/(pall(3,1));
t(1+plus)=log(tall(1));

   for k=2:1:520;
       % unmodified
      %p(1:3,k+plus)=pall(1:3,k);
      %t(k+plus)=tall(k);
       % natural log and inverse
      p(1:2,k+plus)=log(pall(1:2,k));
      p(3,k+plus)=1/(pall(3,k));
      t(k+plus)=log(tall(k));
   end
```

m-file for training aluminum neural network. produces initial plots, saves weights and biases

```
% Set-up and train, single tansigmoid w/ 25 neurons in one linear out

net=newff([minmax(p)],[15,3,1],{'tansig','tansig','purelin'},'trainbr');

net.trainParam.show=100;

net.trainParam.epochs=10000;

%net.trainParam.goal=0.0001;

mega=1E6


%Train and Simulate non-preprocessed data

%net=train(net,p,t);

%a=sim(net,p);


%Train and simulate preprocessed network, scaled to fall w/in -1 to 1
```

```
[pn,minp,maxp,tn,mint,maxt]=premnmx(p,t);

net=train(net,pn,tn);

an=sim(net,pn);

a=postmnmx(an,mint,maxt);


%Train and Simulate non-preprocessed data w/ early stopping/validation

%[pn,minp,maxp,tn,mint,maxt]=premnmx(p,t);

%[pn,minp,maxp,tn,mint,maxt]=premnmx(p,t);

%net=train(net,p,t,[],[],val);

%a=sim(net,p);


figure

%pnew=p;

%tnew=t;

%anew=a;

pnew=exp(p);

tnew=exp(t)/mega;

anew=exp(a)/mega;

plot(pnew(1,1:520),tnew(1:520),'k.') %Target Data, 300C

hold

plot(pnew(1,1:520),anew(1:520),'yd') %ANN output


strain=0.001:0.01:0.501;

estrain=log(strain);

%rate005(1:51)=log(0.005);

rate005(1:51)=log(0.001);

%rate05(1:51)=log(0.05);

rate05(1:51)=log(0.01);

%ratep5(1:51)=log(0.5);

ratep5(1:51)=log(0.1);
```

```
%rate5(1:51)=log(5);

rate5(1:51)=log(1);

temp300(1:51)=1/300;

temp550(1:51)=1/550;

slowest300=[estrain;rate005;temp300];

slow300=[estrain;rate05;temp300];

med300=[estrain;ratep5;temp300];

fast300=[estrain;rate5;temp300];


%for non-normalized data below

%test1=exp(sim(net,slowest300))/mega;

%test2=exp(sim(net,slow300))/mega;

%test3=exp(sim(net,med300))/mega;

%test4=exp(sim(net,fast300))/mega;


%plot(strain,test1,'b+')

%plot(strain,test2,'g+')

%plot(strain,test3,'r+')

%plot(strain,test4,'c+')


%for normalized data use below

slowest300n=tramnmx(slowest300,minp,maxp);

slow300n=tramnmx(slow300,minp,maxp);

med300n=tramnmx(med300,minp,maxp);

fast300n=tramnmx(fast300,minp,maxp);

test1n=sim(net,slowest300n);

test2n=sim(net,slow300n);

test3n=sim(net,med300n);

test4n=sim(net,fast300n);

test1=exp(postmnmx(test1n,mint,maxt))/mega;
```

```
test2=exp(postmnmx(test2n,mint,maxt))/mega;
test3=exp(postmnmx(test3n,mint,maxt))/mega;
test4=exp(postmnmx(test4n,mint,maxt))/mega;

plot(strain,test1,'b+')
plot(strain,test2,'g+')
plot(strain,test3,'r+')
plot(strain,test4,'c+')

xlabel('True Plastic Strain')
ylabel('True Flow Stress (MPa)')
%title ('Neural Network Performance for 300 C')
legend('target', 'ANN output', '0.001', '0.01', '0.1', '1')

figure
plot(pnew(1,521:1008),tnew(521:1008),'k.') %Target Data, 550C
hold
plot(pnew(1,521:1008),anew(521:1008),'md') %ANN output

slowest550=[estrain;rate005;temp550];
slow550=[estrain;rate05;temp550];
med550=[estrain;ratep5;temp550];
fast550=[estrain;rate5;temp550];

%for non-normalized data use below
%test1=exp(sim(net,slowest550))/mega;
%test2=exp(sim(net,slow550))/mega;
%test3=exp(sim(net,med550))/mega;
%test4=exp(sim(net,fast550))/mega;
```

```
%plot(strain,test1,'b+')

%plot(strain,test2,'g+')

%plot(strain,test3,'r+')

%plot(strain,test4,'c+')


xlabel('True Plastic Strain')

ylabel('True Flow Stress (MPa)')

%title ('Neural Network Performance for 550 C')

legend('target', 'ANN output', '0.001', '0.01', '0.1', '1')


%for normalized data use below

slowest550n=tramnmx(slowest550,minp,maxp);

slow550n=tramnmx(slow550,minp,maxp);

med550n=tramnmx(med550,minp,maxp);

fast550n=tramnmx(fast550,minp,maxp);

test1n=sim(net,slowest550n);

test2n=sim(net,slow550n);

test3n=sim(net,med550n);

test4n=sim(net,fast550n);

test1=exp(postmnmx(test1n,mint,maxt))/mega;

test2=exp(postmnmx(test2n,mint,maxt))/mega;

test3=exp(postmnmx(test3n,mint,maxt))/mega;

test4=exp(postmnmx(test4n,mint,maxt))/mega;


plot(strain,test1,'b+')

plot(strain,test2,'g+')

plot(strain,test3,'r+')

plot(strain,test4,'c+')


figure
```

[m,b,r]=postreg(a,t)

netIW=net.IW{1,1};

netLW1=net.LW{2,1};

netLW2=net.LW{3,2};

netb1=net.b{1};

netb2=net.b{2};

netb3=net.b{3};

save /home/kestek/matlab_files/weights.txt netIW netLW1 netLW2 -ascii

save /home/kestek/matlab_files/biases.txt netb1 netb2 netb3 -ascii

save /home/kestek/matlab_files/results.txt p t a -ascii

Feedforward FORTRAN Test Code (Ni3Al weights and biases)

```
!     Program for testing two layer neural network


       INTEGER, PARAMETER :: neurons1=25                !1st layer
       INTEGER, PARAMETER :: neurons2=3                 !2nd layer
       INTEGER, PARAMETER :: n=3                        !inputs

       REAL(KIND=8), DIMENSION(3,1) :: pnup             !strain
       REAL(KIND=8), DIMENSION(3,1) :: pndown           !strain
       REAL(KIND=8), DIMENSION(25,3) :: netIW           !1st layer
       REAL(KIND=8), DIMENSION(3,25) :: netLW1          !2nd layer
       REAL(KIND=8), DIMENSION(1,3) :: netLW2           !output layer
       REAL(KIND=8), DIMENSION(25,1) :: netb1           !input bias
       REAL(KIND=8), DIMENSION(3,1) :: netb2            !2nd bias
       REAL(KIND=8) :: netb3                            !output bias

       REAL(KIND=8), DIMENSION(25,1) :: first_layerup   !1st out
       REAL(KIND=8), DIMENSION(3,1) :: second_layerup   !2nd output
       REAL(KIND=8), DIMENSION(25,1) :: first_layerdown !1st output
       REAL(KIND=8), DIMENSION(3,1) :: second_layerdown !2nd output
       REAL(KIND=8), DIMENSION(1) :: maxt               !max input values
       REAL(KIND=8), DIMENSION(1) :: mint               !min input values
       REAL(KIND=8), DIMENSION(1) :: ah                 !max input values
       REAL(KIND=8), DIMENSION(1) :: asig               !max input values
       REAL(KIND=8), DIMENSION(1) :: aup                !max input values
       REAL(KIND=8), DIMENSION(1) :: anup               !max input values

       REAL(KIND=8), DIMENSION(1) :: strainup
```

```
        REAL(KIND=8), DIMENSION(1) :: strainupn

        REAL(KIND=8), DIMENSION(1) :: maxstrain
        REAL(KIND=8), DIMENSION(1) :: minstrain
        REAL(KIND=8), DIMENSION(1) :: maxrate
        REAL(KIND=8), DIMENSION(1) :: minrate
        REAL(KIND=8), DIMENSION(1) :: temp
        REAL(KIND=8), DIMENSION(1) :: maxtemp
        REAL(KIND=8), DIMENSION(1) :: mintemp
        REAL(KIND=8), DIMENSION(1) :: strainrate
        REAL(KIND=8), DIMENSION(1) :: strainraten
        REAL(KIND=8), DIMENSION(1) :: tempn
        REAL(KIND=8), DIMENSION(1) :: arealh            !max input values
        REAL(KIND=8), DIMENSION(1) :: arealsig          !max input values

          one=1.d+000
          two=2.d+000
          half=0.5d+000

*       Use neural network to determine hardening slope

*       Determine flow stress using artificial neural network
*        based on strain, strain rate, and temperature

*       Read in weight and biases, based on preprocessed p's


           netIW= RESHAPE ((/-2.29119041d+00,0.666477934d+00,
     *  0.91600241d+00,-1.226800935d+00,-1.841852852d+00,
     *  2.893219015d+00,-1.798615385d+00,1.61326515d+00,
     * -0.728342539d+00,-1.636279857d+00,-3.46512053d+00,
     *  0.12201889d+00,2.5344013d+00,1.550179412d+00,
     *  1.208989647d+00,9.650207076d+00,-0.253336193d+00,
     * -1.807373106d+00,-4.183698243d+00,-3.096770882d+00,
     * -24.99397817d+00,0.0393668d+00,1.146489124d+00,
     *  9.742068974d+00,-2.490954165d+00,4.189050952d+00,
     *  3.757590902d+00,-0.615077326d+00,-3.232009528d+00,
     *  2.75278065d+00,4.296580296d+00,-0.6777341d+00,
     * -3.128746794d+00,-3.167354077d+00,9.244429925d+00,
     *  1.915321854d+00,3.708249161d+00,-0.979461807d+00,
     *  0.776142482d+00,0.711049148d+00,1.472400885d+00,
     *  1.476057771d+00,-2.205727831d+00,-2.186002979d+00,
     *  2.113130534d+00,12.90329822d+00,1.827893718d+00,
     * -0.489343906d+00,1.520786932d+00,3.578131075d+00,
     *  9.179352538d+00,5.368149361d+00,-2.742999836d+00,
     * -8.012603576d+00,-5.04511507d+00,9.326064529d+00,
     *  3.35741578d+00,6.306729129d+00,-2.582604467d+00,
     * 11.48264359d+00,9.404368165d+00,2.955619946d+00,
     * -4.89490626d+00,-2.281130016d+00,5.428026141d+00,
     *-11.79798518d+00,7.393329215d+00,1.406222196d+00,
     * -9.762059483d+00,5.180592163d+00,-26.04797851d+00,
     * -2.935171182d+00,3.353779085d+00,-10.49421754d+00,
     * -3.143929269d+00/),(/25,3/))
```

```
         netLW1= RESHAPE((/-6.50005033d+00,0.102232722d+00,
      * -2.427843577d+00,6.814699063d+00,-8.629530543d+00,
      *  3.691531625d+00,3.292231573d+00,-3.311479439d+00,
      *  1.714479246d+00,5.252147294d+00,-3.285274511d+00,
      *  2.69001286d+00,4.116109377d+00,-0.81238531d+00,
      *  1.313934223d+00,2.929757651d+00,-5.152304322d+00,
      *  2.052400252d+00,0.2028993d+00,5.29314923d+00,
      *  8.716643371d+00,-0.708215283d+00,-3.259909669d+00,
      *  3.301723114d+00,2.774479196d+00,-6.111966758d+00,
      * -2.184157062d+00,0.972771277d+00,-3.545680397d+00,
      * -0.22284937d+00,3.768332614d+00,-2.186050604d+00,
      * -0.760718915d+00,-1.597868506d+00,9.301487171d+00,
      * -0.867697617d+00,-2.484194728d+00,-3.49798649d+00,
      * -0.576803757d+00,0.303146846d+00,3.240455993d+00,
      *  4.420462317d+00,-14.74115997d+00,4.052738192d+00,
      *  1.27687621d+00,0.38124621d+00,-0.92728602d+00,
      *  0.122898231d+00,1.419868074d+00,2.100473393d+00,
      * -4.359140825d+00,-2.289970252d+00,-0.527315834d+00,
      *  0.64922091d+00,0.280195067d+00,-1.86235429d+00,
      *  0.496473331d+00,-1.031799514d+00,-1.296610424d+00,
      *  1.083419493d+00,0.025958463d+00,0.05452448d+00,
      * -0.018816562d+00,-9.735636672d+00,2.8020526d+00,
      *  0.168723681d+00,9.933221701d+00,5.670650605d+00,
      * -0.557575203d+00,-0.66678803d+00,0.933064839d+00,
      * -0.374850262d+00,-1.53917131d+00,2.057828651d+00,
      *  1.250216882d+00/),(/3,25/))

         netLW2=RESHAPE((/-0.37917000216232d+00,0.38755741219099d+00,
      *  0.65037690501326d+00/),(/1,3/))

         netb1=RESHAPE((/2.02822571942842d+00,-1.53602772095237d+00,
      *  1.43107742355452d+00,4.36027849798139d+00,3.16178269171670d+00,
      *  2.88790768133384d+00,0.95862530186741d+00,4.18641901964530d+00,
      * -2.96126024520751d+00,-1.37432328853973d+00,2.23672679213150d+00,
      * -1.26714976058775d+00,0.51655752905691d+00,-0.64834890266690d+00,
      *  0.07004496813564d+00,1.15625725991524d+00,2.12728940005436d+00,
      *  2.10455891206278d+00,-3.8810159966337d+00,-1.08745420749906d+00,
      *  9.48545439528936d+00,0.28396437781384d+00,0.68415777299914d+00,
      *  1.24793269846129d+00,0.31375548016631d+00/),(/25,1/))

         netb2=RESHAPE((/1.71061050361251d+00,0.16336983882718d+00,
      * -2.99880322696162d+00/),(/3,1/))

         netb3=   0.16215715902842d+00

*      Read in input values, p equals natural log of equivalent
*       plastic strain (stateOld(k,5)) and strain rate (stateOld(k,5)/dt)
*       with the inverse of the temperature (tempOld(k))
*      Finds two values to compute slope of hard
*

       do k = 1, 500
          strainup = log(k * 0.001d+000)
*          strainup = log(1.00017262519621D-003)
```

```
          strainrate=log(10.0d+000)
          temp=one/1100.0d+000

*     Preprocess input (p) to fall w/in -1 to 1, i.e., normalize
*      maxp and minp are logs of strain range

          maxstrain=-0.64170529721228d+00
          minstrain= -7.39542605728164d+00
          maxrate=   2.30258509299405d+00
          minrate=  -6.90775527898214d+00
          maxtemp= 0.00090909090909d+00
          mintemp=   0.00080000000000d+00

         strainupn=((two*(strainup-minstrain))/(maxstrain-minstrain))-one

         strainraten=two*(strainrate-minrate)/(maxrate-minrate)-one
         tempn=two*(temp-mintemp)/(maxtemp-mintemp)-one

          pnup=RESHAPE((/strainupn,strainraten,tempn/),(/3,1/))

        WRITE(*,*) strainupn, strainraten,tempn, pnup
*     Preprocess input (p) to fall w/in -1 to 1, i.e., normalize
*      maxp and minp are logs of strain range



*     Carry out feedforward portion of ANN using tanh

          first_layerup=tanh(MATMUL(netIW,pnup)+netb1)
          second_layerup=tanh(MATMUL(netLW1,first_layerup)+netb2)
          anup=SUM(MATMUL(netLW2,second_layerup))+netb3

*     Post process network output based on target ranges
          maxt=   20.32646954583388d+00
          mint=  16.56555966075972d+00
          ah=half*(anup+one)*(maxt-mint)+mint


*     Carry out feedforwared portion of ANN using tansig

          first_layerup=two/(one+
     *            exp(-two*(MATMUL(netIW,pnup)+netb1)))-one
          second_layerup=two/(one+
     *            exp(-two*(MATMUL(netLW1,first_layerup)+netb2)))-one
          anup=SUM(MATMUL(netLW2,second_layerup))+netb3


*     Post process network output based on target ranges
          maxt=   20.32646954583388d+00
          mint=  16.56555966075972d+00
          asig=half*(anup+one)*(maxt-mint)+mint
```

```
*     Hard (slope of uniaxial yield stress vs. plastic strain)
*      value based on postprocessed ANN output

          arealh=exp(ah)/1.0d+006
          arealsig=exp(asig)/1.0d+006

          WRITE (*,*) 'inputs =', exp(strainup)

*          WRITE (*,*) 'pup =', pup
*          WRITE (*,*) 'pdown =', pdown
*          WRITE (*,*) 'anup =', anup
*          WRITE (*,*) 'andown =', andown
*          WRITE (*,*) 'aup =', aup
*          WRITE (*,*) 'adown =', adown
          WRITE (*,*) 'a reals/1d6 =', arealh, arealsig
*          WRITE (*,*) 'hardann =', hardann
*          WRITE (*,*) 'netb1 =', netb1
*          WRITE (*,*) 'first layerup = ', first_layerup
*          WRITE (*,*) 'second layerup = ', second_layerup
*          WRITE (*,*) 'netIW =', netIW

          OPEN (UNIT=9, FILE='annout.txt', STATUS='OLD')
          WRITE (9,*) exp(strainup), ',',arealh


        end do
         CLOSE (UNIT=9)
         END
```

Artificial Neural Network VUMAT (aluminum weights and biases)

```
C
C User subroutine VUMAT
      subroutine vumat (
C Read only -
     *     nblock, ndir, nshr, nstatev, nfieldv, nprops, lanneal,
     *     stepTime, totalTime, dt, cmname, coordMp, charLength,
     *     props, density, strainInc, relSpinInc,
     *     tempOld, stretchOld, defgradOld, fieldOld,
     *     stressOld, stateOld, enerInternOld, enerInelasOld,
     *     tempNew, stretchNew, defgradNew, fieldNew,
C Write only -
     *     stressNew, stateNew, enerInternNew, enerInelasNew )
C
      include 'vaba_param.inc'
C
      dimension coordMp(nblock,*), charLength(nblock), props(nprops),
     1     density(nblock), strainInc(nblock,ndir+nshr),
     2     relSpinInc(nblock,nshr), tempOld(nblock),
     3     stretchOld(nblock,ndir+nshr),
     4     defgradOld(nblock,ndir+nshr+nshr),
     5     fieldOld(nblock,nfieldv), stressOld(nblock,ndir+nshr),
     6     stateOld(nblock,nstatev), enerInternOld(nblock),
     7     enerInelasOld(nblock), tempNew(nblock),
     8     stretchNew(nblock,ndir+nshr),
     9     defgradNew(nblock,ndir+nshr+nshr),
     1     fieldNew(nblock,nfieldv),
     2     stressNew(nblock,ndir+nshr), stateNew(nblock,nstatev),
     3     enerInternNew(nblock), enerInelasNew(nblock)
C
      character*80 cmname
      parameter ( zero = 0.d0, one = 1.d0, two = 2.d0,
     *     third = 1.d0 / 3.d0, half = 0.5d0, op5 = 1.5d0 )

      INTEGER, PARAMETER :: n1=15              !1st layer
      INTEGER, PARAMETER :: n2=3               !2nd layer
      INTEGER, PARAMETER :: n=3                !inputs
      REAL, DIMENSION(n,1) :: pnup             !normalized input
      REAL, DIMENSION(n1,n2) :: netIW          !1st layer weights
      REAL, DIMENSION(n2,n1) :: netLW1         !2nd layer weights
      REAL, DIMENSION(1,n2) :: netLW2          !output layer weights
      REAL, DIMENSION(n1,1) :: netb1           !1st layer bias
      REAL, DIMENSION(n2,1) :: netb2           !2nd layer bias
      REAL :: netb3                            !output layer bias
      REAL, DIMENSION(n1,1) :: first_layerup   !1st layer output
      REAL, DIMENSION(n2,1) :: second_layerup  !2nd layer output
      REAL :: temp                             !temperature
      REAL :: maxstrain                        !max training strain
      REAL :: minstrain                        !min training strain
      REAL :: maxrate                          !max training rate
      REAL :: minrate                          !min training rate
      REAL :: maxtemp                          !max training temp
      REAL :: mintemp                          !min training temp
      REAL :: maxt                             !max target stress
      REAL :: mint                             !min target stress
      REAL :: anup                             !max input values
      REAL :: strainup                         !strain input
      REAL :: strainupn                        !normalized strain in
      REAL :: strainrate                       !strain rate input
      REAL :: strainraten                      !normalized rate in
      REAL :: tempn                            !normalised temp
      REAL :: arealup                          !ANN output normalized
      REAL :: flowstresslog                    !ANN output
      REAL :: flowstress                       !ANN output


C
C For plane strain, axisymmetric, and 3D cases using
C the J2 Mises Plasticity with linear hardening.
C The state variable is stored as:
C            STATE(*,1) = equivalent plastic strain
C            STATE(*,2) = yield stress
*            STATE(*,3) = previous equivalent plastic strain
C
C User needs to input
C     props(1)      Young's modulus
```

```
C     props(2)      Poisson's ratio
*     props(3)      Yield Stress

      e      = props(1)
      xnu    = props(2)
      yield  = props(3)


      twomu  = e / ( one + xnu )
      alamda = xnu * twomu / ( one - two * xnu )
      thremu = op5 * twomu
*
      if ( stepTime .eq. zero ) then
        do k = 1, nblock
          trace = strainInc(k,1) + strainInc(k,2) + strainInc(k,3)
          stressNew(k,1) = stressOld(k,1)
     *         + twomu * strainInc(k,1) + alamda * trace
          stressNew(k,2) = stressOld(k,2)
     *         + twomu * strainInc(k,2) + alamda * trace
          stressNew(k,3) = stressOld(k,3)
     *         + twomu * strainInc(k,3) + alamda * trace
          stressNew(k,4)=stressOld(k,4) + twomu * strainInc(k,4)
          if ( nshr .gt. 1 ) then
            stressNew(k,5)=stressOld(k,5) + twomu * strainInc(k,5)
            stressNew(k,6)=stressOld(k,6) + twomu * strainInc(k,6)
          end if
        end do
      else
        do k = 1, nblock

          if (stateOld(k,2) .gt. yieldNew) then
             yieldNew=stateOld(k,2)
          else
             yieldNew=yield
          endif

          trace = strainInc(k,1) + strainInc(k,2) + strainInc(k,3)
          s11 = stressOld(k,1) + twomu * strainInc(k,1) + alamda * trace
          s22 = stressOld(k,2) + twomu * strainInc(k,2) + alamda * trace
          s33 = stressOld(k,3) + twomu * strainInc(k,3) + alamda * trace
          s12 = stressOld(k,4) + twomu * strainInc(k,4)
          if ( nshr .gt. 1 ) then
            s13 = stressOld(k,5) + twomu * strainInc(k,5)
            s23 = stressOld(k,6) + twomu * strainInc(k,6)
          end if
*
          smean = third * ( s11 + s22 + s33 )
          s11 = s11 - smean
          s22 = s22 - smean
          s33 = s33 - smean
          if ( nshr .eq. 1 ) then
            vmises = sqrt( op5*(s11*s11+s22*s22+s33*s33+two*s12*s12) )
          else
            vmises = sqrt( op5 * ( s11 * s11 + s22 * s22 + s33 * s33 +
     *          two * s12 * s12 + two * s13 * s13 + two * s23 * s23 ) )
          end if

          sigdif = vmises - yieldNew
          facyld = zero
          if ( sigdif .gt. zero ) facyld = one
          deqps = facyld * sigdif / thremu


*      Skip over ANN calculations if plastic strain very small

         if ( stateOld(k,1) .le. 0.001 ) then
           goto 10
         end if


*      Use neural network to determine flow stress for yield

*      Determine flow stress using artificial neural network
*       based on strain, strain rate, and temperature

*      Read in weight and biases, based on preprocessed p's

        netIW= RESHAPE ((/-1.30886320844e0,0.61350866278e0,
     * -0.31604362172e0,-0.22276340900e0,-0.30166074069e0,
     * -0.76650083986e0,0.92982643065e0,3.21985852116e0,
     * -1.15737590810e0,0.33449798237e0,0.18718939680e0,
     *  1.33021644786e0,0.77346362644e0,-1.08968275549e0,
     *  1.16284062592e0,-1.19030123942e0,-0.65184667768e0,
     *  0.97759448388e0,0.55036131643e0,-1.20214314211e0,
     *  0.85443336084e0,0.59086173838e0,-0.15184903229e0,
     * -0.38853726133e0,0.34461081231e0,1.17269107707e0,
     *  0.16217868250e0,-0.63960312037e0,0.87060443083e0,
     *  0.50883444461e0,-1.20798592088e0,0.90696255121e0,
     * -0.13811751356e0,0.55513892115e0,-0.33511666380e0,
     *  0.44612113199e0,-0.42632688837e0,0.46454383804e0,
     * -0.17686688961e0,0.10852735163e0,1.01993230960e0,
     * -0.48579948547e0,0.50485127912e0,-0.10800388527e0,
     *  0.83701259741e0/),(/15,3/))

        netLW1= RESHAPE((/-1.4036302048926e0,-0.0888981061288e0,
     * -0.77988883093434e0,1.14932327030400e0,0.03563088517640e0,
     *  1.23798557874999e0,-0.25655899688045e0,0.40719969805190e0,
     *  0.40904417209983e0,0.62039690883432e0,-0.72025862537237e0,
     * -0.23374857296522e0,0.28340021345402e0,0.61920163275620e0,
     *  0.86282036844925e0,-0.02402741491012e0,0.08526120573753e0,
     * -0.94397272963354e0,-0.35288275799876e0,-0.23392108723201e0,
     *  0.39603214815881e0,-1.79031334163575e0,-0.00965649047407e0,
     * -1.52818348495388e0,-0.28208090039197e0,-0.71981820431881e0,
     *  0.05230217548034e0,-0.55155487755926e0,0.01195532611218e0,
     *  0.62774358764050e0,-0.53922161269398e0,0.44705714548159e0,
     * -0.03472304108462e0,0.24995195567965e0,-0.21200045976412e0,
     * -0.57596737006593e0,-0.84475130155020e0,-0.35667482375627e0,
     *  0.00646994907276e0,-0.11602459670144e0,-0.23254213693042e0,
     *  0.90710908849066e0,0.14502144438566e0,-0.39409513436252e0,
     *  1.09273348749336e0/),(/3,15/))

        netLW2=RESHAPE((/-1.4245448734401740e+0,
     * -1.6711139565947069e+0,2.3390343682859784e+0/),(/1,3/))

        netb1=RESHAPE((/1.1575676768921091e-01,-6.5047404542647169e-01,
     *  1.2766949448816717e-001,5.3679527386221526e-001,
     *  3.7747607029160657e-001,-6.3753815187336793e-001,
     * -2.8142817371979052e-003,1.9541520960737284e+0,
     *  1.0934959041082201e-001,-6.0523288337514469e-001,
     *  6.5735502459205897e-002,-6.2729105634754045e-001,
     *  6.1207896092349023e-001,-1.7568553865705844e-001,
     *  2.7762803902490091e-001/),(/15,1/))

        netb2=RESHAPE((/3.3642811734368651e-002,2.4889359134725526e-002,
     *  -6.2964059568444697e-001/),(/3,1/))

        netb3=5.4866629594521854e-001

*      Read in input values, p equals natural log of equivalent
*       plastic strain (stateOld(k,5)) and strain rate (stateOld(k,5)/dt)
*       with the inverse of the temperature (tempOld(k))
*      Finds two values to compute slope of hard

         stateNew(k,1) = stateOld(k,1) + deqps
         strainup=log(stateOld(k,1))

        straineq=sqrt((2.0d0/9.0d0)*((strainInc(k,1)-strainInc(k,2))**2
+
     *      (strainInc(k,2)-strainInc(k,3))**2 +
     *       (strainInc(k,3)-strainInc(k,1))**2))

         strainrate=log(straineq/dt)

*          strainrate=log(1.0)
*          PRINT*, straineq, dt, exp(strainrate)

         temp=one/450.0

*      Preprocess input (p) to fall w/in -1 to 1, i.e., normalize
*       maxp and minp are logs of strain range

         maxstrain=-6.1274207702126193e-01
         minstrain=-6.2463361517885900e+00
         maxrate=0.0000000000000000e+00
         minrate=-6.9077552789821404e+00
         maxtemp=3.3333333333333335e-003
         mintemp= 1.8181818181818182e-003

        strainupn=((two*(strainup-minstrain))/(maxstrain-minstrain))-one
        strainraten=two*(strainrate-minrate)/(maxrate-minrate)-one
        tempn=two*(temp-mintemp)/(maxtemp-mintemp)-one
        pnup=RESHAPE((/strainupn,strainraten,tempn/),(/3,1/))

*      Preprocess input (p) to fall w/in -1 to 1, i.e., normalize
*       maxp and minp are logs of strain range


*      Carry out feedforward portion of neural network tanh function

          first_layerup=tanh(MATMUL(netIW,pnup)+netb1)
          second_layerup=tanh(MATMUL(netLW1,first_layerup)+netb2)
          anup=SUM(MATMUL(netLW2,second_layerup))+netb3

*      Carry out feedforwared portion of ANN using tansig

*          first_layerup=two/(one+
*      *            exp(-two*(MATMUL(netIW,pnup)+netb1)))-one
*          second_layerup=two/(one+
*      *            exp(-two*(MATMUL(netLW1,first_layerup)+netb2)))-one
*          anup=SUM(MATMUL(netLW2,second_layerup))+netb3


*      Post process network output based on target ranges
          maxt=  1.9382327177547300e+001
          mint=  1.4710361082439562e+001
          flowstresslog=half*(anup+one)*(maxt-mint)+mint
          flowstress=exp(flowstresslog)

          yieldNew = flowstress
*


           sigdif = vmises - yieldNew
           facyld = zero
           if ( sigdif .gt. zero ) facyld = one
           deqps = facyld * sigdif / thremu



*
* Update the state variable
  10       stateNew(k,1) = stateOld(k,1) + deqps
           stateNew(k,2) = yieldNew
           stateNew(k,3) = stateOld(k,1)
*
* Update the stress
           factor = yieldNew / ( yieldNew + thremu * deqps )
           stressNew(k,1) = s11 * factor + smean
           stressNew(k,2) = s22 * factor + smean
           stressNew(k,3) = s33 * factor + smean
           stressNew(k,4) = s12 * factor
           if ( nshr .gt. 1 ) then
             stressNew(k,5) = s13 * factor
             stressNew(k,6) = s23 * factor
           end if
*
```

```
* Update the specific internal energy -
          if ( nshr .eq. 1 ) then
            stressPower = half * (
     *        ( stressOld(k,1) + stressNew(k,1) ) * strainInc(k,1) +
     *        ( stressOld(k,2) + stressNew(k,2) ) * strainInc(k,2) +
     *        ( stressOld(k,3) + stressNew(k,3) ) * strainInc(k,3) ) +
     *        ( stressOld(k,4) + stressNew(k,4) ) * strainInc(k,4)
          else
            stressPower = half * (
     *        ( stressOld(k,1) + stressNew(k,1) ) * strainInc(k,1) +
     *        ( stressOld(k,2) + stressNew(k,2) ) * strainInc(k,2) +
     *        ( stressOld(k,3) + stressNew(k,3) ) * strainInc(k,3) ) +
     *        ( stressOld(k,4) + stressNew(k,4) ) * strainInc(k,4) +
     *        ( stressOld(k,5) + stressNew(k,5) ) * strainInc(k,5) +
     *        ( stressOld(k,6) + stressNew(k,6) ) * strainInc(k,6)
          end if
          enerInternNew(k) = enerInternOld(k) + stressPower / density(k)
*
* Update the dissipated inelastic specific energy -
          yieldOld = stateOld(k,2)
          plasticWorkInc = half * ( yieldOld + yieldNew ) * deqps
          enerInelasNew(k) = enerInelasOld(k)
     *         + plasticWorkInc / density(k)
        end do
      end if
*

*      PRINT*, dt,stateOld(100,1),stateNew(100,1),yieldNew/1e6
      return
      end
```

Sample ABAQUS Input File (keywords portion)

```
*Heading
** Job name: anncrsmss25 Model name: anncoarse
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
**
*Part, name=billet
*End Part
*Part, name="bottom die"
*End Part
*Part, name="top die"
*End Part
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
*Instance, name=billet-1, part=billet
*Node
      1,           0.,           0.
...
```

```
     7896,       0.1016,      0.46736
*Element, type=CAX4R
   1,    1,    2,   44,   43
...
     7667, 7853, 7854, 7896, 7895
** Region: (Section-1:Picked)
*Elset, elset=_PickedSet3, internal, generate
    1,  7667,     1
** Section: Section-1
*Solid Section, elset=_PickedSet3, material="ANN Al model"
1.,
*End Instance
**
*Instance, name="top die-1", part="top die"
      -6e-16, 0.557726013777,           0.
*Node
      1,           0.,            0.,           0.
*Nset, nset="top die-1-RefPt_", internal
1,
*Surface, type=SEGMENTS, name=RigidSurface_, internal
START, 0.281279474497, 0.200285285711
 LINE, 0.281560709073, 0.0121485412482
 LINE, 0.25311727616499, 0.00865612682795865
 CIRCL, 0.247516476289, 0.00280786549265, 0.253897337458,
0.00230303742046
 LINE, 0.244757311521, -0.0320670717363
 CIRCL, 0.233584909803957, -0.0407484990818504, 0.235211340645, -
0.031311832914
 CIRCL, 0.125342461893, -0.040700874284, 0.179325160528719, -
0.355567381773363
 LINE, 0.0403520136402053, -0.046631269461076
 CIRCL, 0.0345414380169, -0.051863135252, 0.0407949672485, -
0.0529658011802
 LINE, 0.0290281403057, -0.0831306003274
 CIRCL, 0.026457983138, -0.0857007574951, 0.0259077793038, -
0.0825803964932
 LINE,           0., -0.0903660137774
*Rigid Body, ref node="top die-1-RefPt_", analytical
surface=RigidSurface_
*End Instance
**
*Instance, name="bottom die-1", part="bottom die"
          0., 0.116487179748,           0.
*Node
      1,    0.1687972,  -0.07521171,           0.
*Nset, nset="bottom die-1-RefPt_", internal
1,
*Surface, type=SEGMENTS, name=RigidSurface_, internal
START,           0., -0.116487179746
 LINE,    0.1031875, -0.116487179746
 LINE,    0.1069975, -0.112677179746
 LINE,   0.11509375, -0.112677179746
 CIRCL,    0.1158875, -0.111883429746,   0.11509375, -0.111883429746
 LINE,    0.1158875, -0.0883685577212
 CIRCL,     0.116459, -0.0876065577212,   0.11566525, -0.0876065577212
 LINE,     0.116459, -0.0825382835546
 CIRCL, 0.119702129393023, -0.0793491565604519,    0.1222375, -
0.0851710459346
 CIRCL, 0.16531592256702, -0.0629981722327693, 0.265221561343559, -
0.413500713077876
 CIRCL, 0.168310491326995, -0.0625210443135777, 0.168797214607, -
0.0752117141324001
 CIRCL, 0.220665983525995, -0.0665554752543277, 0.177061394564823, -
0.290689345004376
 CIRCL, 0.234069119117885, -0.0817943907762074, 0.217494205639, -
0.0828588705792
 LINE, 0.238251761409033, -0.145042314938998
 CIRCL, 0.250616221153, -0.156885603708, 0.250993406697, -0.144115892926
 LINE, 0.254343371508001, -0.156891439026509
 CIRCL, 0.25594044832, -0.155425337088, 0.254345876818, -0.155291240988
 LINE, 0.267770324186, -0.0147532146774
 CIRCL, 0.270473947382155, -0.0102057295345242, 0.274084200601, -
0.0154295759806
 LINE, 0.278820962291995, -0.00443702293886266
 CIRCL, 0.281560709073, 0.000786823507458, 0.275210709073,
0.000786823507458
 LINE, 0.281560709073, 0.0121485412482
*Rigid Body, ref node="bottom die-1-RefPt_", analytical
surface=RigidSurface_
*End Instance
*Nset, nset=_PickedSet10, internal, instance=billet-1, generate
    1,  7855,    42
*Elset, elset=_PickedSet10, internal, instance=billet-1, generate
    1,  7627,    41
*Nset, nset=_PickedSet13, internal, instance="top die-1"
 1,
*Nset, nset=billet, instance=billet-1, generate
    1,  7896,     1
*Elset, elset=billet, instance=billet-1, generate
    1,  7667,     1
*Nset, nset=_PickedSet22, internal, instance="bottom die-1"
 1,
*Nset, nset="top die ref pt", instance="top die-1"
 1,
*Elset, elset=__PickedSurf9_S2, internal, instance=billet-1, generate
   41,  7667,    41
*Elset, elset=__PickedSurf9_S3, internal, instance=billet-1, generate
 7627,  7667,     1
*Surface, type=ELEMENT, name=_PickedSurf9, internal
__PickedSurf9_S2, S2
__PickedSurf9_S3, S3
*Elset, elset=__PickedSurf21_S1, internal, instance=billet-1, generate
  1,  41,   1
*Elset, elset=__PickedSurf21_S2, internal, instance=billet-1, generate
   41,  7667,    41
*Surface, type=ELEMENT, name=_PickedSurf21, internal
__PickedSurf21_S1, S1
__PickedSurf21_S2, S2
```

```
*End Assembly
*Amplitude, name=Amp-1
0., 1., 20., 0.0667
*Amplitude, name=Amp-2, definition=SMOOTH STEP
0., 1., 6., 1., 14., 1., 30., 0.
**
** MATERIALS
**
*Material, name="ANN Al model"
*Conductivity
180.,
*Density
 2.7e+06,
*Depvar
      3,
*Specific Heat
896.,
*User Material, constants=3
 5.4e+10,     0.3, 1.4e+07
*Material, name="al 6061-0 450C"
*Conductivity
180.,
*Density
 2.7e+06,
*Elastic
 6.97e+10, 0.3
*Plastic
 2e+07,0.
*Rate Dependent
 6.55e+06, 0.168
*Specific Heat
896.,
**
** INTERACTION PROPERTIES
**
*Surface Interaction, name=friction
*Friction
 0.3,
*Surface Behavior, pressure-overclosure=HARD
*Surface Interaction, name=friction-2
*Friction
 0.1,
*Surface Behavior, pressure-overclosure=HARD
*Surface Interaction, name=frictionless
*Friction
0.,
*Surface Behavior, pressure-overclosure=HARD
**
** BOUNDARY CONDITIONS
**
** Name: billet center Type: Symmetry/Antisymmetry/Encastre
*Boundary
_PickedSet10, XSYMM
** Name: bottom die Type: Symmetry/Antisymmetry/Encastre
```

```
*Boundary
_PickedSet22, ENCASTRE
** Name: top die Type: Velocity/Angular velocity
*Boundary, type=VELOCITY
_PickedSet13, 1, 1
_PickedSet13, 2, 2
_PickedSet13, 6, 6
** ----------------------------------------------------------------
**
** STEP: Step-1
**
*Step, name=Step-1
*Dynamic, Explicit
, 24.21
*Bulk Viscosity
0.06, 1.2
** Mass Scaling: Semi-Automatic
**               billet
*Fixed Mass Scaling, elset=billet, factor=25.
**
** BOUNDARY CONDITIONS
**
** Name: top die Type: Velocity/Angular velocity
*Boundary, amplitude=Amp-1, type=VELOCITY
_PickedSet13, 1, 1
_PickedSet13, 2, 2, -0.03
_PickedSet13, 6, 6
*Adaptive Mesh, elset=billet, frequency=5, mesh sweeps=3, op=NEW
**
** INTERACTIONS
**
** Interaction: Int-1
*Contact Pair, interaction=friction, mechanical constraint=KINEMATIC,
cpset=Int-1
"top die-1".RigidSurface_, _PickedSurf9
** Interaction: Int-2
*Contact Pair, interaction=friction, mechanical constraint=KINEMATIC,
cpset=Int-2
"bottom die-1".RigidSurface_, _PickedSurf21
**
** OUTPUT REQUESTS
**
*Restart, write, overlay, number interval=10, time marks=YES
*Monitor, dof=2, node="top die ref pt"
**
** FIELD OUTPUT: F-Output-1
**
*Output, field, variable=PRESELECT, number intervals=100
**
** HISTORY OUTPUT: H-Output-1
**
*Output, history, variable=PRESELECT
*End Step
** ----------------------------------------------------------------
```

```
**
** STEP: Step-2
**
*Step, name=Step-2
*Dynamic, Explicit
, 30.
*Bulk Viscosity
0.06, 1.2
**
** BOUNDARY CONDITIONS
**
** Name: top die Type: Velocity/Angular velocity
*Boundary, amplitude=Amp-2, type=VELOCITY
_PickedSet13, 1, 1
_PickedSet13, 2, 2, -0.00566
_PickedSet13, 6, 6
*Adaptive Mesh, elset=billet, frequency=1, mesh sweeps=5, op=NEW
**
** OUTPUT REQUESTS
**
*Restart, write, overlay, number interval=30, time marks=YES
**
** FIELD OUTPUT: F-Output-1
**
*Output, field, variable=PRESELECT, number intervals=100
**
** HISTORY OUTPUT: H-Output-1
**
*Output, history, variable=PRESELECT
*End Step
```

Sample Python Script for Deformed Shape Extraction

```
>>> from abaqus import *
>>> from abaqusConstants import *
>>> from odbAccess import *
>>> import visualization
>>> myViewport=session.Viewport(name='flattened')
>>> myOdb=visualization.openOdb('c:/316528/cnv2ndstp.odb')
>>> myViewport.setValues(displayedObject=myOdb)
>>> odb=openOdb('c:/316528/cnv2ndstp.odb')
>>> myAssembly=odb.rootAssembly
>>> for instanceName in odb.rootAssembly.instances.keys():
... print instanceName
...
"BOTTOM DIE-1"
"TOP DIE-1"
DEFORMED-1
```

```
>>> myModel=mdb.Model(name='flattened')
The model "flattened" has been created.
>>> myPart=mdb.models['flattened'].PartFromOdb(name='myPart',odb=openOdb
                                          ('c:/316528/cnv2ndstp.odb',
                                           instance='DEFORMED-1',
                                           shape=DEFORMED,step=0)
...
>>> myPart=mdb.models['flattened'].PartFromOdb(name='myPart45',odb=openOdb
                                          ('cnv2ndstp.odb'),instance='DEFORMED-1',
                                          shape=DEFORMED,step=0,frame=45)
>>> myPart=mdb.models['flattened'].Part2DGeomFrom2DMesh(name='myPart',
                                           part=myPart,featureAngle=15)
```

VITA

Scott Kessler was born January 29, 1963 in Kansas City, Missouri. After attending public schools in Kansas City, he received the following degrees from the University of Missouri: B.S. in Mechanical & Aerospace Engineering (1997); M.S. in Mechanical & Aerospace Engineering (2000); Ph.D. in Mechanical & Aerospace Engineering (2005). He is married to the former Katherine A. Clark of Prairie Village, Kansas and has two children; Anne and Dylan. Currently, he is self-employed as a consulting engineer and is registered with the State of Missouri as a Professional Engineer.